Digital Image Processing via Combination of Low-Level and High-Level Approaches

Dong Wang

Submitted for the Degree
of Doctor of Philosophy

The Digital Media & Systems Research Institute
School of Computing, Informatics & Media
University of Bradford

2011

Acknowledgements

First and foremost, thanks to my supervisor, Prof. Jianmin Jiang, who has provided continued support and guidance, and spent a great deal of time and effort throughout the course of this project to ensure its success.

My sincere appreciation to my colleague, Dr. Jinchang Ren, for his generous assistance in my three years research. He also provided useful ideas and suggestions in the image-processing field.

On my personal note, I would like to thank my parents who supported my decision to choose research over employment and kept the wheels of research lubricated.

Dong Wang

Bradford

2011

Abstract

With the growth of computer power, Digital Image Processing plays a more and more important role in the modern world, including the field of industry, medical, communications, spaceflight technology etc. There is no clear definition how to divide the digital image processing, but normally, digital image processing includes three main steps: low-level, mid-level and high-level processing.

Low-level processing involves primitive operations, such as: image pre-processing to reduce the noise, contrast enhancement, and image sharpening. Mid-level processing on images involves tasks such as segmentation (partitioning an image into regions or objects), description of those objects to reduce them to a form suitable for computer processing, and classification (recognition) of individual objects. Finally, higher-level processing involves "making sense" of an ensemble of recognised objects, as in image analysis.

Based on the theory just described in the last paragraph, this thesis is organised in three parts: Colour Edge and Face Detection; Hand motion detection; Hand Gesture Detection and Medical Image Processing.

In Colour Edge Detection, two new images G-image and R-image are built through colour space transform, after that, the two edges extracted from G-image and R-image respectively are combined to obtain the final new edge. In Face Detection, a skin model is built first, then the boundary condition of this skin model can be extracted to cover almost all of the skin pixels. After skin detection, the knowledge about size, size ratio, locations of ears and mouth is used to recognise the face in the skin regions.

In Hand Motion Detection, frame differe is compared with an automatically chosen threshold in order to identify the moving object. For some special situations, with slow or smooth object motion, the background modelling and frame differencing are combined in order to improve the performance.

In Hand Gesture Recognition, 3 features of every testing image are input to Gaussian Mixture Model (GMM), and then the Expectation Maximization algorithm (EM)is used to compare the GMM from testing images and GMM from training images in order to classify the results.

In Medical Image Processing (mammograms), the Artificial Neural Network (ANN) and clustering rule are applied to choose the feature. Two classifier, ANN and Support Vector Machine (SVM), have been applied to classify the results, in this processing, the balance learning theory and optimized decision has been developed are applied to improve the performance.

Keywords: Digital Image Processing, Colour Edge Detection, Image Features Classification and Cluster, Features Selection.

Declaration

Some parts of the work presented in this thesis have been published in the following articles:

• **D. Wang**, J. Ren, S. S. Ipson. "Skin Detection from Different Colour Space for Model-Based Face Detection", Proc. Communications in Computer and Information Science, Springer, 15, pp. 487-494, 2008

• J. Ren, **D. Wang**, Jianmin Jiang, Stanley S. Ipson, "Fusion of Intensity and Channel Difference for Improved Colour Edge Detection," Proc. Visual Information engineering (VIE), Xi'an, China, pp. 18-22, 2008

• J. Jia, J. Jiang and **D. Wang**, "Recognition of hand gesture based on Gaussian Mixture Model," in Proc. CBMI (Content-Based Multimedia Indexing) workshop: 353-356, 2008.

• Ren. J, Jiang. J, **D. Wang**, Ipson, S.S. "Fusion of Intensity and Intercomponent Chromatic Difference for Effective and Robust Colour Edge Detection," IET Image Processing, volume 4, pp. 294-301, August 2010.

• **D. Wang**, J.Ren, Jianmin Jiang, Stanley S. Ipson. "Applying Feature Selection for Effective Classification of Microcalcification Clusters in Mammograms," The 10th IEEE International Conference on Computer and Information Technology (CIT 2010), 2010.

• Ren J, **Wang D**, Jiang J "Effective Recognition of MCCs in Mammograms Using an Improved Neural Classifier", accepted to International Scientific Journal Engineering Applications of Artificial Intelligence, 2011, ELSEVIER.

List of Abbreviations

Range	Difference of the maximum and the minimum intensity values within the image
TPR	True Positive Rate
FPR	False Positive Rate
img(i)	image
GMM	Gaussian Mixture Model
EM	the Expectation Maximization algorithm
ML	Maximum Likelihood
ANN	Artificial Neural Network
SVM	Support Vector Machine
MSE	Mean squared normalized error performance
MLP	Multilayer Perceptron
trainlm	Levenberg-Marquardt Algorithm
BP	Back Propagation
RBF	Gaussian Radial Basis Function
ROC	Receiver Operating Characteristic

List of Symbols

ω_r	Weight for channel R
ω_g	Weight for channel G
ω_b	Weight for channel B
T_h	High thresholding
T_l	Low thresholding
μ	Mean
σ	Standard deviation
t_p	True positive
f_p	False positive
f_n	False negative
$X_i = \{x_t, 1 \leq t \leq T^i\}$	Feature vectors for data points belong to i^{th} class
θ^i_{GMM}	Model parameters for Gaussian Mixture Model
$P_{z_i}(x_t \vert \mu_j, \Sigma_j)$	The Gaussian distribution for the j^{th} class
Σ_j	Covariance matrix

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

An image may be defined as a two-dimensional function, $f(x, y)$, where x and y are spatial (plane) coordinates, and the amplitude of f at any pair of coordinates (x, y) is called the intensity or grey level of the image at that point. When x, y and the amplitude value f are finite, discrete quantities, the image is called as a Digital Image.

Interest in digital image processing methods stems from two principal application areas: improvement of pictorial information for human interpretation; and processing of image data for storage, transmission, and representation for autonomous machine perception [1].

In general, digital image processing refers to a procedure to apply computer algorithms and process digital images to achieve some expected targets, such as enhancement, compression, etc. Here, three levels of computerized processes are always used, namely, low-level, mid-level, and high-level processes.

- Low-level processing is pre-processing for further analysis, and it aims to reduce noise and increase the contrast in the image. Traditional methods for low-level image processing include low-pass filtering for noise suppression, grey-level operations such as histogram equalization for contrast improvement as well as edge detection using image gradient and related techniques, etc.
- Mid-level processing focus on extracting descriptions of the scene from

the image descriptions extracted at the low level, the output is usually in some more symbolic form, describing the position and shape of portions of the scene. The analysis usually does not know anything about what objects are in the scene, but does use a lot of knowledge of scene shape and how shape appears in an image, such as region/object segmentation. Accurate region segmentation facilitates subsequent higher level processing.

- High-level image processing is the most intelligent part, which attempts to identify the regions or features previously detected and interpreted them, i.e. classification and recognition. Techniques such as fuzzy logic, (artificial) neural network, mathematics model and approaches, and other artificial intelligent techniques can be applied for recognition and classification.

In general, most users are only interested in the outputs from high level processing, especially those non-professional ones, because such outputs can provide straightforward semantic information as whether the image contains certain object of interest or not and their qualitive/quantative measurements et al.

On the other hand, through comparing these high-level outputs with expected results, i.e. the ground truth, feedbacks can be generated and fed to other two stages of image processing. The feedbacks are useful in determining optimal parameters and further improve the performance of low- and middle-level processing.

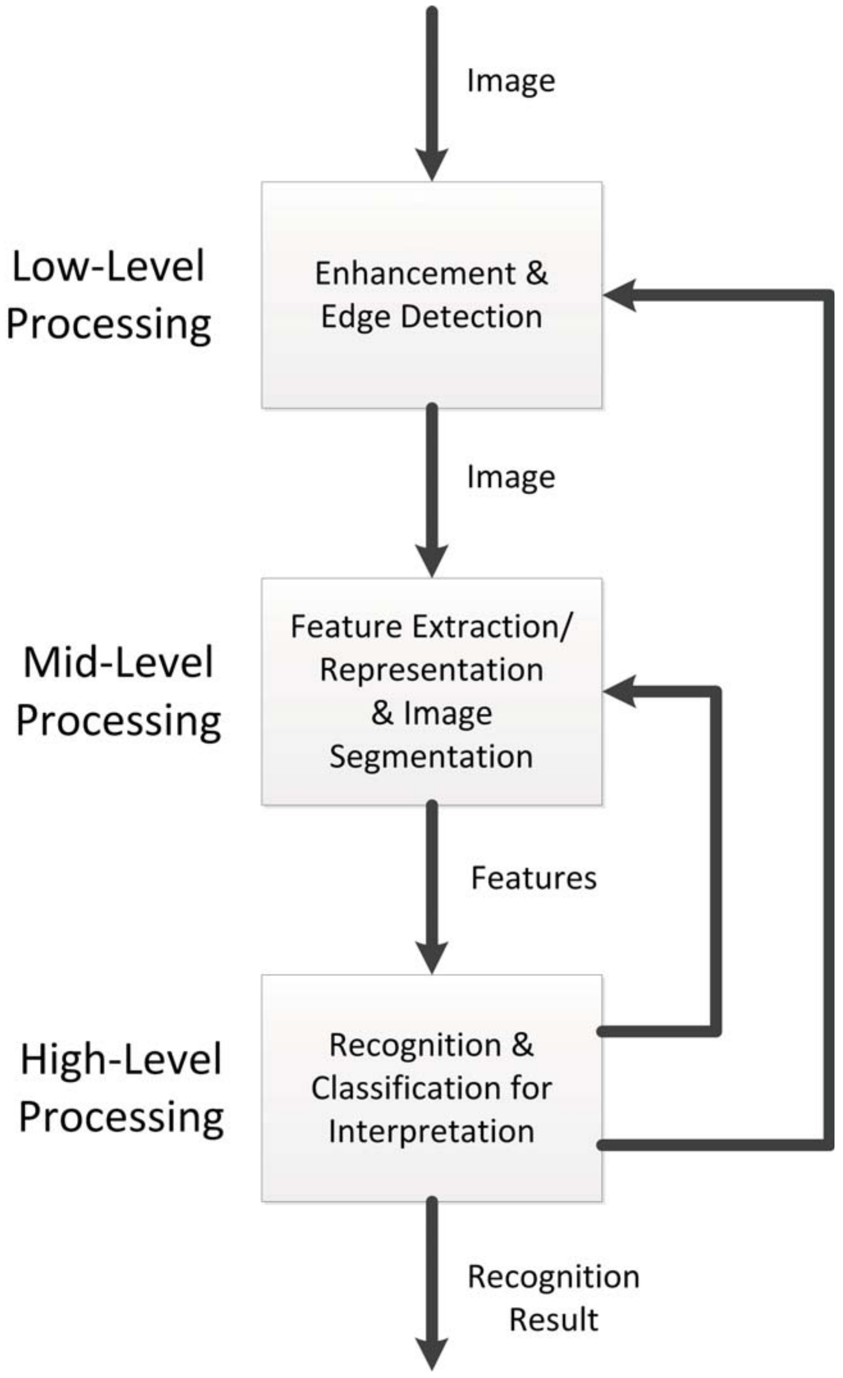


Image
Low-Level Processing
Enhancement & Edge Detection
Image
Mid-Level Processing
Feature Extraction/ Representation & Image Segmentation
Features
High-Level Processing
Recognition & Classification for Interpretation
Recognition Result

Figure 1.1: Flowchart of Digital Image Processing

From Fig 1.1, the main steps can be noticed in the digital image processing. In the other words, low level processing is image processing, it's include:

1. Image data are not interpreted, ie semantic is not explored;
2. Signal processing methods, e.g, 2D Fourier transformation;
3. Same methods for a wide class of problems;

High level processing is image understanding or computer vision, it's include:

1. Interpretation to a specific application domain;
2. Complex, artificial intelligence techniques and feedback;
3. A tough problem often needs to be simplified;

Fig 1.2 shows the relationships among computer vision, machine learning and image processing [2].

From Fig 1.2, the fields most closely related to image processing that are computer vision and machine learning. There is a significant overlap in the range of techniques and applications that these cover. With the growth of computer power, more and more researchers focus on the performance of high level processing in digital image processing, that means: machine learning and computer vision have been become the core issues in digital image processing.

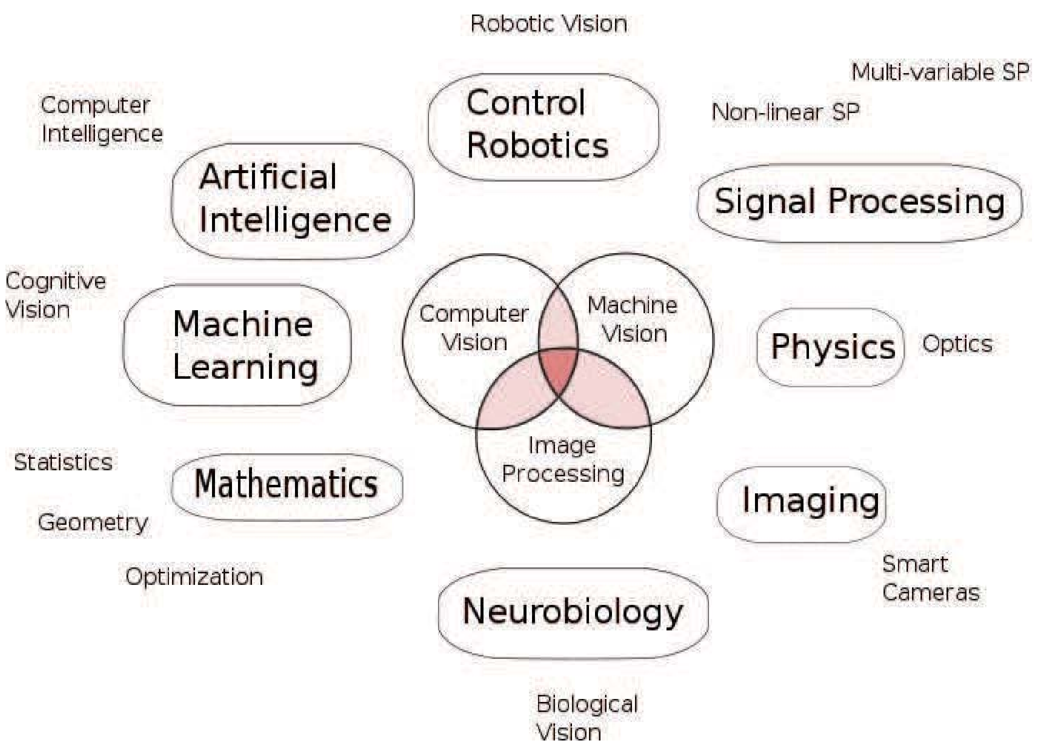


Figure 1.2: Relationships among Computer Vision, Machine Learning, Image Processing and the Others Scientific Fields [2]

Computer vision means that the machine can extract the information from an image in order to solve a specific problem. As one scientific discipline, computer vision is focus on the theory behind artificial system that extracts information from images. As an example, extract features from images belong to this area. And these features extracted from images are the foundation for further research, like machine learning.

Machine learning, as a part of artificial intelligence, is a scientific discipline concerned with the design and development of algorithms that allow computers to evolve behaviours based on empirical data. A major focus of machine learning is to automatically understand and recognise the example data and make one intelligence decision based on learning. An example of pattern recognition is classification, classification could be simply explained as classifying one new substantial data into several specific groups based on the characteristic of these data from the training data.

1.2 Motivation

The main steps and overlapping scientific field's for digital image process have been described in last section. In Fig 1.3, the more details of digital image processing will be shown below.

But there are still some challenge problems in image processing, such as follows:

1. Effective edge detection from colour images, one has to decide an optimal way to fuse chromatic components and illumination intensity for this purpose;
2. Motion detection and moving object segmentation;
3. The image classification with severely unbalanced data in medical imaging, as normally there are much more benign samples than malignant ones;

Basically, the main work in this thesis focus on the three important problems listed above. Colour edge detection, as discussed in last section, belongs

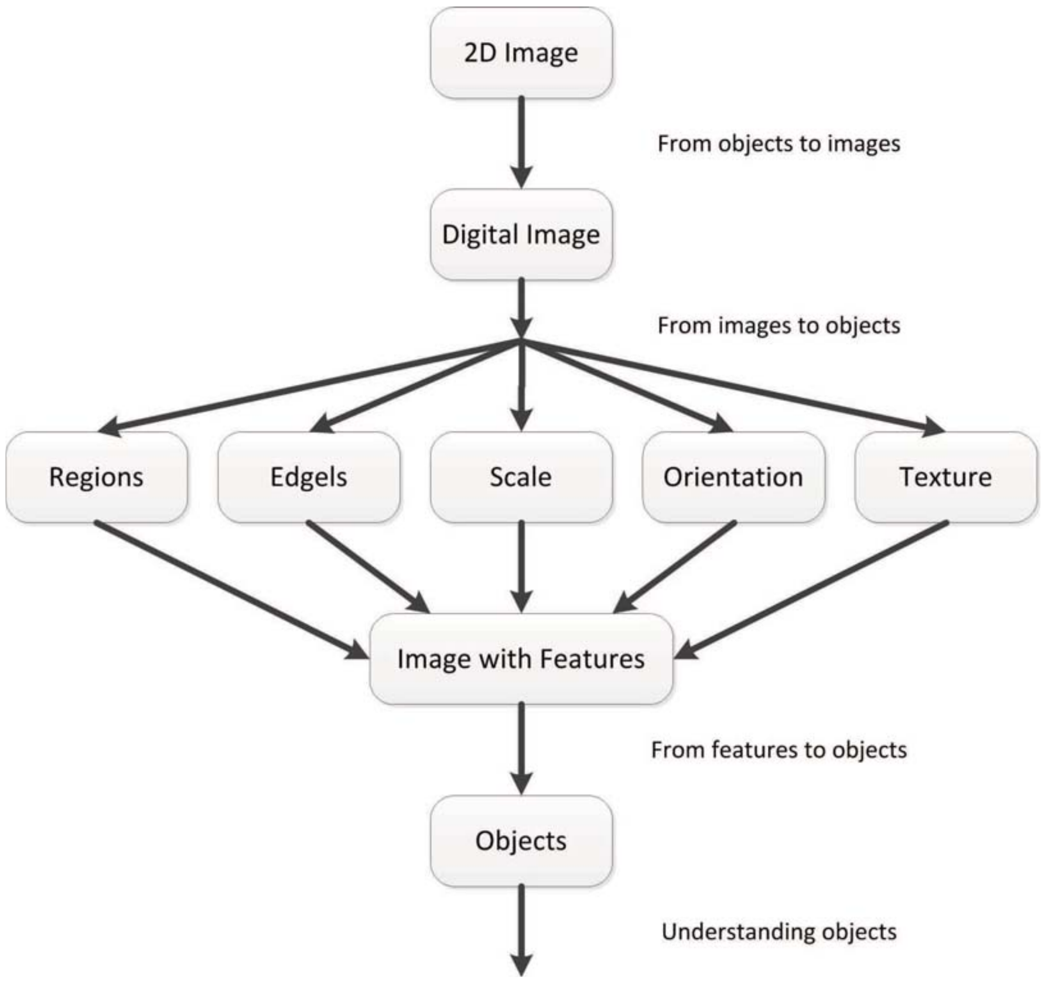


Figure 1.3: Image-based Recognition Hierarchy of Representations

to low level image processing, and how to fuse colour and intensity information for effective colour edge detection will be investigated. Extending from edge detection, one skin model in order to detect the colour face has been built in this project. Regarding motion detection and motion segmentation, hand gesture recognition is used as the background to develop the algorithm, and this is taken as transition from low level processing to high level processing. Since medical image processing remains to be more challenging, it provides more spaces for high level image processing.

As a result, computer-aided mammogram analysis is comprehensively investigated as a major application case to illustrate the effectiveness of proposed approaches in machine learning towards high-level image processing, where the performance of two classifiers: neural network and Support Vector Machine (SVM) are compared and evaluated.

the background of colour edge and face detection, hand gesture recognition and high level image processing.In next section will be introduced in next section.

1.3 Bakcground

1.3.1 Colour Edge Detection

As discussed in last section, low-level image processing operations use the values of image pixels to modify individual pixels in an image. They can be divided into point-to-point, neighbourhood-to-point and global-to-point operations [10]. Point-to-point operations depend only on the values of the corresponding pixels from the input image and the parallelization is simple. Neighbourhood operations produce an image in which the output pixels depend on a group of neighbouring pixels around the corresponding pixel from the input image. Operations like smoothing, sharpening, filtering, noise reduction and edge detection are highly parallelizable.

So according to the description in last paragraph, in this project, the colour edge detection as an example has been applied for Low Level Image processing. In the following, the background is also introduced about the colour edge detection.

Today state of art in image processing needs to pay more attention in colour images. So, colour edge detection plays very important roles in many applications for image analysis, segmentation and recognition.

In monochrome images, an edge usually corresponds to object boundaries or change in physical properties such as reflectance or illumination. In this sense, a colour (multi-spectral) image contains more detailed edge information. Besides this, monochrome edge detection may not be sufficient for certain applications since no edges will be detected in grey level images if neighbouring objects have different hue but same intensity and also according to psychological research on human visual system colour plays an important role in deciding object boundaries.

To this end, the basic problem here is how to fuse chromatic and intensity information for effective colour edge detection. In this thesis, an improved method on colour edge detection is proposed in which the significant advantage is the use of inter-component difference information for effective colour edge detection.

For any given colour image C, a grey D-image is defined as the accumulative differences between each of its two colour components, and another grey R-image is then obtained by weighting of D-image and the grey intensity image G. The final edges are determined through fusion of edges extracted from R-image and G-image. Quantitative evaluations under various levels of Gaussian noise are performed for further comparisons. Comprehensive results from different test images have proved that our approach outperforms edges detected from traditional colour spaces like RGB, YCbCr and HSV in terms of effectiveness and robustness.

1.3.2 Colour Face Detection

Middle level image processing operations work on images and output other data structures, such as detected objects (e.g., faces) or statistics, thereby reducing the amount of information. Operations such as Hough transform (to find a line in an image), centre-of-gravity calculation, labelling an object, are examples of mid-level image operations. They are more limited from the aspect of data parallelism when compared to low level operations. They can be defined as image to object operations.

In this project, the colour face detection as one sample has been applied for Middle Level Image Processing. From the next paragraph, the background of colour face detection is also discussed.

Same as colour edge detection, automatic detection of skin and face plays very important roles in many vision applications, such as face and gesture recognition in intelligent human-machine intelligence and visual surveillance [9,10,11], naked adult image detection [12,13], video phone or sign language recognition [14, 15] as well as content-based multimedia retrieval [16, 17].

Although face regions can be detected using template matching and Haar based features, however, these may be affected by internal factors like facial expression, beard and glass etc. and of course also by external factors, like scale, lighting condition, the orientation of face etc.

On the other hand, in most cases the overall shape and size remain the same. Therefore, it is essential to explore new approach for face detection, which using knowledge-based constraints of shape, size and skin colour information.

In this thesis, an efficient and effective method for frontal-view face detection is proposed, based on skin detection and knowledge-based modelling. Firstly, skin pixels are modelled by using supervised training, and boundary conditions are then extracted for skin segmentation. Faces are further detected by shape filtering and knowledge-based modelling. Skin results from different colour spaces are compared. In addition, experimental results have demonstrated our method robust in successful detection of skin and face regions even with variant lighting conditions and poses.

1.3.3 Hand Gesture Recognition

Before starting the research for High Level Image Processing, Hand Gesture Recognition has been chosen as transition from Low Level Image Processing to High Level Image Processing. In this example, the technologies of Low and High Level image processing is applied at the same time, such as, locate the moving target between two continually frames and GMM as classifier in my experiment.

Human hand gestures have their specific meanings and are widely used for communications between deaf people. Actually, gesturing is so deeply rooted in communications that people often continue gesturing when speaking. Recently, hand gesture recognition has gained a lot of interests, which plays a crucial role in a wide range of applications including automatic sign language understanding, entertainment, and human computer interaction (HCI). Because hand gestures are natural and intuitive in providing rich information to computers without extra cumbersome devices, they can offer a great potential for next generation user interfaces, being especially suitable for large scale displays, 3D volumetric displays or wearable devices.

For human-computer interaction, vision-based recognition of hand gestures can provide a natural and modest solution [18]. To achieve this, three steps are required, including:

1. analyzing signals acquired by imaging sensors such as video, infrared or ultrasonic;
2. inferring the geometry and motion of the hand;
3. mapping to a set of predefined gestures;

An important potential application of this technology is to develop advanced interfaces for the interaction with virtual objects. These objects can be images on a computer screen and the user can manipulate the objects by moving his/her hand and performing actions like ”grasping” and ”releasing”. Using gesture recognition, the user actions on the virtual object will be reproduced by the computer and the operational result is shown in the graphical interface so as to provide feedback to the user. Another important application is to provide computing devices that can interpret gestures from the sign-language alphabet and aid natural interaction of hearing impaired people [19].

In our system, the core issues of gesture recognition have been addressed in extracting robust features, leading to a more accurate estimation. The new approach that proposed is different from existing efforts reported in the literature. It focuses on estimating the gesture contained in an image by analysing different complex features including shape, colour and orientation histogram quantized in Gaussian Mixture Model (GMM).

1.4 High Level Image Processing

High-level image processing operations work on vector data or objects in the image and return other vector data or objects. They usually have irregular access patterns and thus are difficult to run data parallel computations. They can be divided into object-to-object or object-to-point operations. Position estimation and object recognition theory are examples of this category.

From the definition of high level image processing, the aim of the high level image processing is understanding object as it can be seen in last paragraph. This is machine learning. Recently, many approaches have been develope-d in the machine learning area, such as decision learning tree, association rule learning, artificial neural network, genetic programming, inductive logic programming, support vector machines, clustering, bayesian networks, reinforcement learning.

Medical image processing remains to be more challenging and provide more spaces for high level image processing. In order to deeply dig in the area of high level image processing, medical image as example has been chosen in the experiment.

Normally, in medical image processing, single one computer-aided diagnosis (CAD) system should be built, like that shown in Fig 1.4:

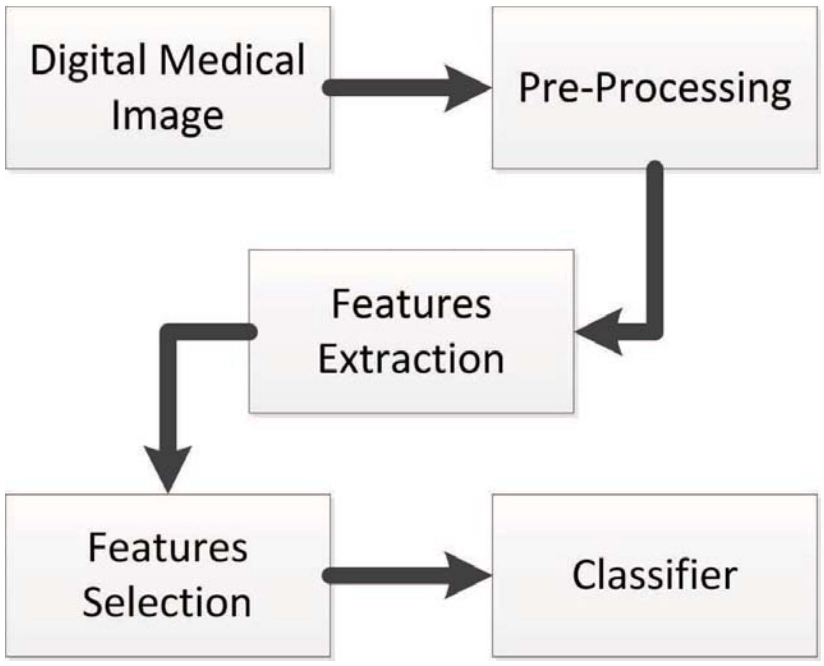


Figure 1.4: Flowchart of Typical CAD System

Therefore, Feature Selection and Classifier have been focused in this project.

Feature selection is foundation of the final results, how to find the best performance feature and how to combine the features are important steps in this stage. In this thesis, neural network and clustering rules have been involved to solve this problem.

Classifier plays a role of classification, in this thesis, two classifiers have been discussed: neural network and SVM, and compare their results. Last, an improved over-sampling based balanced learning strategy is proposed, which can avoid drawbacks of existing techniques. The performance along with the proposed optimized decision making has been fully validated using two individual classifiers including SVM and ANN. The proposed method is found to be effective in improving both the sensitivity and specificity rate while not increase the computing complexity of the classifier.

1.4.1 Thesis Organization

The main content of this thesis is organised as follows.

Chapter 2 surveys existing work in terms of colour edge detection, colour skin and face detection, hand gesture recognition and high level image processing

In chapter 3, one novel approach to detect the colour edge will be introduced at first, in this step, an improved method on colour edge detection will introduced. The significant advantage of the proposed method is the use of inter-component difference information for effective colour edge detection. Secondly, for colour skin and face detection, an efficient and effective method will proposed for frontal-view face detection based on skin detection and knowledge-based modelling.

In chapter 4, hand gesture recognition, one method that integrates the features of shape, colour and orientation histograms will be introduced, which are extracted from images, and estimate the comparability with all the different types of gestures by a proposed Expectation-Maximization algorithm in Gaussian Mixture Model.

In chapter 5, firstly, how to select features which already get from mammograms, and how to combine the features in order to find the best combination will be introduced. In this step, two methods will be emphasised: Neural network and PCA. Secondly, the classifier will be discussed, which includes Neural network and SVM. In this step, one novel approach will also applied: balance learning to get the most performance, and the finally, a comparison of the results from Neural network and SVM respectively will be listed at the end of section.

Chapter 6 concludes the thesis and discusses the summary of research and some suggestions for further research.

Chapter 2

Literature Review

2.1 Introduction

Following the introduction in the previous chapter. A review of relevant literature is presented in this chapter. In accordance with the objectives previously specified, the main contents focus on the following four parts including:

1. Colour edge and face detection;
2. Hand gesture recognition;
3. High level image processing;

2.2 A Review of Techniques for Colour Edge Detection

Since physical edges usually correspond to apparent variations in the illumination and colours, edge detection is very useful and important in many low-level vision applications as to provide essential visual information for feature extraction, segmentation and scene understanding [53-57].

In general, edge detection contains three main stages, namely preprocessing or smoothing, image difference and gradient detection for edge pixel judgment, and continuous edge extraction. Gradient-based methods are almost the earliest edge detector which only uses convolution templates to obtain local difference for edge detection, and then Canny introduced a well-defined edge detector with good performance, high precision and unique response [53].

From convolution templates to Canny edge detector, traditional edge detection methods are usually defined on grey images, and some improvements or new methods are necessary for edge detection from colour images according to human colour perceptions. A simple idea is to convert $<$r,g,b$>$ colour image to its luminance intensity image G, from which traditional edge detectors are applied to extract colour edges. As the conversion from colour to grey is multiple to one mapping, edges detected from G-image are less accurate and usually edge pixels with obvious colour difference but less intensity variation are missing.

Another simple idea for colour edge detection is to apply the edge detectors to each colour component and the final edges will be the combined results of the edges from different component images. Although the combined edges have more accuracy and detail information than the edges from G-image, they are still not accurate enough and have missing edges due to the fact that inter-component information is ignored in the process of edge detection.

Since colour vision is synthetic perception of R, G and B components, the combined edges extracted from multiple single components have intrinsic limitations according to human visual perception. To acquire more reasonable and accurate edges, quite a few colour spaces and colour models have been investigated [58, 59, 60], such as HLS (hue-lightness-saturation), HSV (hue-saturation-value), YUV, XYZ, YCbCr, etc. When <r,g,b> colour images are converted to a specified colour space, edges will be extracted from the components of the new space. Since different components are independent of each other, the final edges are also a combination of the edges from each component including colour and luminance information [61].

Alternatively, some other efforts have been made on edge detection from colour images, such as the compass operator in [62], direction information measurement in [63], cluster analysis in [64], and invariance analysis [61, 65]. However, choosing a suitable colour space is still a very fundamental task in such a context on which the above operators or processing can then be applied [66]. In addition, quite a few combined approaches have been proposed for colour edge detection, such as morphological gradient followed by outlier rejection [67], statistical analysis of R-G and B-Y colour components [68], clustering of pixels using the minimal spanning tree [69], combination of self-organising map (SOM) and a grey scale edge detector [70], and neighbourhood hypergraph and validation of hyperedge [56].

Although the combined edges have more accuracy and detail information than the edges from the intensity image, they are still not accurate enough for effective object detection and image segmentation because of missing of certain edges. So, how to choose a new suitable colour space is still a very fundamental task in such a context on which the operators or processing can be applied.

2.3 A Review of Techniques for Skin and Face Detection

Employing skin detection to locate human objects in videos is a straightforward approach owing to the fact that human skin has a consistent appearance which is significantly different from many other objects [71]. Some other common methods of detecting human objects include face detection (using Haar-like features, for example) [72, 73] and motion and appearance modelling [74, 75].

In general, at least three issues need to be considered in skin classification, i.e. colour representation and quantization, skin colour modelling, and classification approaches. In real applications, some post-processing is also required for the detection and recognition of more semantic events including faces, hands or even special skin patches as naked images, etc.

Although many different colour spaces have been introduced in skin detection, such as RGB or normalized RGB [76], HSV (or HSI, HSL, TSL) [77-80], YCbCr (or YIQ, YUV, YES) [81], and CIELAB (or CIELUV) [82], etc., they can be simply classified into two categories by examining whether the luminance intensity component is considered. Due to the differences between the training and test data, various results have been reported: Some people argue that ignoring luminance component helps to achieve more robust detection [83-85]; however, others still insist that luminance information is essential for accurate modelling of skin colours [86].

Moreover, it becomes widely acknowledged that training from different colour spaces produces comparable results as long as the Y component is included [86], i.e. invertible conversion between colours spaces can be achieved [87]. Consequently, choosing a suitable colour space merely depends on the intrinsic requirements of efficiency, rather than effectiveness, i.e. the chosen colour space should have its components extracted from image or videos as simply as possible. For instance, YCbCr and RGB spaces are naturally used in compressed and uncompressed images and videos.

As for colour quantization, various quantization levels have been suggested, such as 32, 64, 128 and 256 [86, 76]. Higher levels mean that more storage space is required hence lower efficiency in the detection process. However, there is no well-accepted scheme in such a context. Therefore, the performance under different levels needs to be compared, especially on test data under varying illumination.

To model skin (and non-skin) colours, two main approaches are generally utilized, i.e. parametric and nonparametric ones. The former usually model skin colours as Gaussian or mixture of Gaussian distributions, and the number of components in the mixed model varies from 2 to 16 [82]. Other parametric models include elliptic boundary models, etc [126]. Parameters in the models are usually obtained by the EM (Expectation Maximization) approach [80]. Non-parametric approaches include histogram-based models [86, 76] and neural networks, etc. [86].

In addition, there are also some imprecise models using fixed ranges of thresholds such as the work in [85] and [78], although the latter also contains a further step to adapt with image content. It is found that histogram-based approaches and neural network based ones usually generate the best results and outperform parametric approaches [86].

With colour models of skin and non-skin provided, skin pixels are usually determined by using Bayesian decision rules of maximum a posteriori, minimum cost and even maximum likelihood strategies [76]. The latter has only skin colour model and is similar to those using a look-up table for decisions whilst the first two also have a model for non-skin colours and thus the likelihood ratio of the pixel's colour in skin and non-skin models are obtained for decision.

Other classification approaches include those using linear or elliptic decision boundaries [77, 82, 85]. Nevertheless, one (or more) threshold(s) is (are) then required for such a decision, and unsuitable threshold(s) may lead to quite poor performance.

Furthermore, existing approaches work mainly on uncompressed images and videos, which make them less efficient owing to the fact that most such media is provided in the compressed format and thus an expensive decompression is required before detection. Consequently, it provides an efficient and fast implementation. Comparing with previous work reported in [84] and [88], an optimal threshold of likelihood ratio between skin and non-skin pixels is derived which skip the iterative processing in [88].

Furthermore, even without a dynamic model as introduced in [79], the output results from sequences under varying illumination still seem very promising.

2.4 A Review of Techniques for Hand Gesture Recognition

Human hand gesture, show in Fig 2.1, have their deeply meaning especially for the deft people, how to identify the hand gesture become one challenge problem in human-computer interaction.

Since early nineties, Chaitanya Gurrapu [98] adopts GMM to classify the human body's gestures and track its changes through time using HMM. The key features extracted from body images are polygonal vertices obtained from body shapes. Accordingly, GMM is trained on the vertices feature and the relationship between vertices which is represented in the form of gradient of the line joining two vertices. The final accuracy is close to 98%. Yang Liu [99] used a method integrating shape and depth information for robust hand tracking. Shape is the primary measurement which builds an important function describing areas of state-space and contains critical information about the posterior. Recognition rate is between 70% and 92% under various numbers of samples. Sebastien Marcel [100] presented a hand gesture recognition algorithm based on input/output Hidden Markov Models. This approach already achieves a recognition rate between 90% and 100% of the sequence.

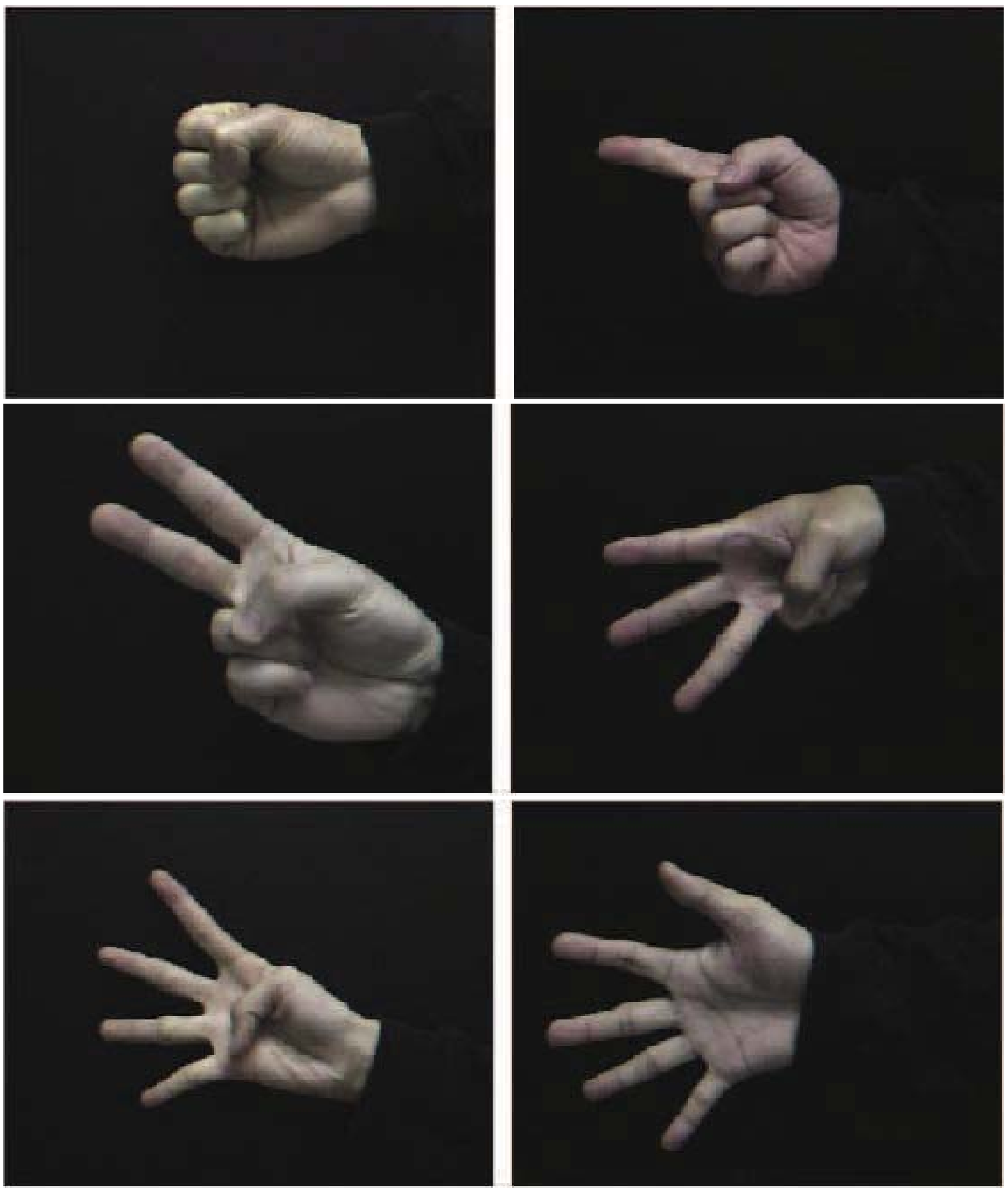


Figure 2.1: Human Hand Gesture

2.5 A Review of High Level Image Processing Techniques

In high level image processing, machine learning techniques have been widely used, more and more researchers focus on this area.

There are several approaches in machine learning area:

1. *Decision tree learning* is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree. Decision tree learning is one of the most widely used and practical methods for inductive inference [159]. Uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. More descriptive names for such tree models are classification trees or regression trees. In these tree structures, leaves represent classifications and branches represent conjunctions of features that lead to those classifications [160]. Decision tree learning usually applied in these three area: instance is represented as attribute-value pairs; the target function has discrete output values; the training data may contain errors.

2. *Association rule learning* is a popular and well researched method for discovering interesting relations between variables in large databases. Piatetsky-Shapiro [161] describes analyzing and presenting strong rules discovered in databases using different measures of interestingness. Based on the concept of strong rules, Agrawal et al. [162] intro-

duced association rules for discovering regularities between products in large scale transaction data recorded by point-of-sale (POS) systems in supermarkets.

3. *An artificial neural network* is a collection of simple artificial neurons connected by directed weighted connections. When the system is set running, the activation levels of the input units are clamped to desired values. After this the activation is propagated, at each time step, a-long the directed weighted connections to other units. The activations of non-input neurons are computed using each neuron's activation function. The system might either settle into a stable state after a number of time steps, or in the case of a feedforward network, the activation might flow through to output units [163].

4. *Genetic programming (GP)* is an evolutionary algorithm-based methodology inspired by biological evolution to find computer programs that perform a user-defined task. It is a specialization of genetic algorithms (GA) where each individual is a computer program. It is a machine learning technique used to optimize a population of computer programs according to a fitness landscape determined by a program's ability to perform a given computational task [164].

5. *Inductive logic programming (ILP)* is a subfield of machine learning which uses logic programming as a uniform representation for examples, background knowledge and hypotheses. Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesised

logic program which entails all the positive and none of the negative examples [165].

6. *Support vector machines (SVMs)* are a set of related supervised learning methods that analyze data and recognise patterns, used for classification and regression analysis. The original SVM algorithm was invented by Vladimir Vapnik and the current standard incarnation (soft margin) was proposed by Corinna Cortes and Vladimir Vapnik [166].

7. *Cluster analysis or clustering* is the assignment of a set of observations into subsets (called clusters) so that observations in the same cluster are similar in some sense [167].

8. *A Bayesian network*, is a probabilistic graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases [168].

9. *Reinforcement learning* is an area of machine learning in computer science, concerned with how an agent ought to take actions in an environment so as to maximize some notion of cumulative reward [169].

It is considered that choose the mammogram as an example, to illustrate the effectiveness of proposed approaches in machine learning towards high-level image processing, which based on the reviews of how the other researchers applying the machine learning techniques on mammogram.

Detection and classification of microclacification clusters (MCCs) from mammograms plays important roles in early diagnosis of breast cancer. In early detected cases, MCCs can be found in 30-50% of the screened mammograms. This will increase to 60-80% if histological examinations of cancer cases are considered. The difficulty for the detection of MCCs is due to i) small size but various shapes, ii) low contrast and unclear boundary from surrounding normal tissue, etc. [20, 21].

To solve such problems, a typical CAD system contains at least four stages including preprocessing, feature-based extraction of regions of interest (ROI), detection of MCCs, and classification. The preprocessing covers noise suppression and contrast enhancement, including histogram equalization etc. [22], which is useful for robust extraction of features and ROIs. The features include local statistics and texture modelling [23, 24], wavelets [25-28], and morphological features [29]. From the segmented ROIs, MCCs can be detected using heuristics [30,21], fuzzy sets [31,32], sub-image decomposition and filtering [33], and machine learning algorithms [34-36], where shape features such as linear structure are widely used [33, 37-39].

Regarding classification of MCCs, a number of techniques have been presented using machine learning approaches to classify samples as malignant and benign, and this is also the focus of this thesis. Among these techniques, two main streams are those using artificial neural networks (ANN) [35, 36, 38, 40-44,] and support vector machines (SVM) [45, 46-48, 43], along with other approaches like linear discriminant analysis (LDA) [38], Bayes classifiers [33], K-nearest-neighbour (KNN) clustering [49], genetic algorithms (GA) [50, 49] and decision-rules [31, 43]. According to the evaluation work in [46], SVM and other kernel based approaches including relevance vector machine and kernel Fisher discriminant (KFD) outperform ANN classifier in classification of MCCs. However, the area under the ROC curve achieved by SVM is only 0.85, which apparently has space for further improvement.

The reasons for the classification accuracy in terms of above is not only the complexity of the problem, i.e. containing cases that cannot be judged even by radiologists as analysed in [46], but also the shortcomings of single classifiers, especially the difficulty in dealing with imbalanced training set in machine learning. The imbalance here refers to the fact that one class is more heavily represented than the other. This is a common problem in real-world domains in detecting rare but important cases from large suspiciously normal samples [51]. Most existing machine learning algorithms fail in dealing with imbalanced data set as their predictions are biased to the class of majority samples [52].

2.6 Summary

In this chapter, existing techniques in colour edge detection, colour face detection, hand gesture recognition and medical imaging are respectively discussed.

For effective colour edge detection, the fundamental problem here is how to combine colour and intensity information for improved accuracy. Basically, chromatic components can help to extract edges from pixels of same intensity but various hues. Therefore, fusion of chromatic difference and intensity information can benefit better accuracy in colour edge detection, and relevant details are presented in chapter 3.

In hand gesture recognition, multiple features extracted from gesture images could be organised and controlled by GMM to formulate new discriminating vector for classification and recognition of human gestures. The application of Gaussian Mixture Model illustrates the advantage that it provides improved performance over other existing methods, yet requiring only modest computational cost to complete the gesture recognition. Details of the proposed approach are discussed in Chapter 4.

In medical imaging, imbalanced data is a major problem which has not been solved. Therefore, in my project, before applying SVM or ANN for the classification of MCC candidates, balanced learning is introduced to solve the problem of imbalanced data. In addition, optimized decision making is also proposed to choose the threshold of SVM or ANN automatically; this is also can improve the performance of the classifier, especially, when no balance learning was employed. Relevant details are presented in Chapter 5.

Chapter 3

Colour Edge and Face Detection

3.1 Introduction

Edge detection, is very useful and important in many low-level vision applications as to provide essential visual information for feature extraction, segmentation and scene understanding [53-57]. Most existing methods extract colour edges via fusing edges detected from each colour components or detecting from the intensity image where inter-component information is ignored. In this chapter, an improved method on colour edge detection is proposed in which the significant advantage is the use of inter-component difference information for effective colour edge detection.

For any given colour image C, a grey D-image is defined as the accumulative differences between each of its two colour components, and another grey R-image is then obtained by weighting of D-image and the grey intensity image G. The final edges are determined through fusion of edges extracted from R-image and G-image. Quantitative evaluations under various levels of Gaussian noise are performed for further comparisons. Comprehensive results from different test images have proved that our approach outperforms edges detected from traditional colour spaces like RGB, YCbCr and HSV in terms of effectiveness and robustness.

High-level image processing normally means located or extract one specific object from a set of input images. In the rest of the chapter, the skin detection from different colour space transform for model-based face detection will be discussed.

Skin and face detection has many important applications in intelligent human-machine interfaces, reliable video surveillance and visual understanding of human activities. In this chapter, an efficient and effective method for frontal-view face detection, based on skin detection and knowledge-based modelling, is proposed. Firstly, skin pixels are modelled by using supervised training, and boundary conditions are then extracted for skin segmentation. Faces are further detected by shape filtering and knowledge-based modelling. Skin results from different colour spaces are compared.

In addition, experimental results have demonstrated our method robust to be in successful detection of skin and face regions even with variant lighting conditions and poses.

3.2 Colour Space Transform

Linear Colour Space, An image stored in a Linear Colour Space contain-s evenly distributed linear steps of colour values across all ranges — low, medium, and high. Unlike a nonlinear colour space, image stored in a linear colour space do not take into account the fact that the human eye is much more sensitive to the low to middle ranges of intensity than it is to the high or brighter ranges. It is most common to work with images stored in a linear colour space for visual effects work, although some facilities have opted to composite their work in logarithmic space.[170][171][172]

Oppositely, Non-linear Colour Space is a very useful concept to understand if dealing with higher bit depth images, which a higher bit depth image does not merely contain a large number of colours and spans a greater range of values. Non-linear colour space takes advantage of the fact that the human eye is more sensitive to brightness change in the darker areas if an images that the lighter areas. Most non-linear colour space is a variation of the logarithmic curve. [173][174][175]

Comparing to texture and shape, colour is the chief discrimination attribute in human visual system [148, 58]. Therefore, colour edge detection is more important in scene analysis and understanding. Although many colour transforms and colour space models have been developed, they can be converted between each other by mapping from and to RGB space.

In general, luminance and the differences or proportions between luminance and different colour components from RGB space compose the new components in the transformed space. Following is an example transform from RGB space to YUV space:

$$y = \omega_r r + \omega_g g + \omega_b b \tag{3.1}$$

$$u = b - y, \quad v = r - y \tag{3.2}$$

In eqn (3.1) and (3.2), from RGB to YUV colour space is a linear transform has been listed, in which the three components in the new space are defined simply by linearly combining the components of the RGB space. In YUV space, illumination intensity is decided by weighting R, G and B values using weights ω_r, ω_g, ω_b, where these weights are non-negative and sum up to 1.

Actually, the weights here are firstly introduced in converting colour TV signal to grey one, where $\omega_r = 0.299$, $\omega_g = 0.587$, $\omega_b = 0.114$ is widely adopted. In other applications like image retrieval, all the weights may have a value of one-third to measure an overall brightness.

Generally, these linear transforms can be defined as:

$$\begin{bmatrix} Y \\ A \\ B \end{bmatrix} = \begin{bmatrix} \omega_r & \omega_g & \omega_b \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

As hue is more effective in distinguishing different colours than illumination intensity, HSV (Hue, Saturation, Value) and HIS (Hue, Intensity, Saturation) transforms are taken as suitable colour spaces that correspond to human visual perceptions and have been widely utilized in colour clustering for image segmentation and coding.

The RGB to HSV transform can be defined as [55]:

$$\begin{aligned} &V = \max(R, G, B) \\ &S = V'/V \\ &V' = V - M \\ &M = \min(R, G, B) \end{aligned}$$

Let $r' = (V - R)/V'$, $g' = (V - G)/V'$ and $b' = (V - B)/V'$, then H is given by:

$$H = \frac{1}{6}\begin{cases} 5 + b' & \text{if } R = V \ \& \ G = M \\ 1 - g' & \text{if } R = V \ \& \ G \neq M \\ 1 + r' & \text{if } G = V \ \& \ B = M \\ 3 - b' & \text{if } G = V \ \& \ B \neq M \\ 3 + g' & \text{if } B = V \ \& \ R = M \\ 5 + r' & \text{otherwise} \end{cases}$$

YIQ and YCbCr have similar transforms like YUV above, and HSV and HLS spaces have more complex transform formulas [59]. Although some of these transforms can achieve coherent distance measurement with human perceptions, they are not effective in accurate edge detection due to the fact that inter-component information has not been fully considered.

Let f be the original three component colour image, and it defines one component D-image as:

$$\begin{aligned} D(i,j) &= \omega_1|r(i,j) - g(i,j)| \\ &+ \omega_2|r(i,j) - b(i,j)| \\ &+ \omega_3|b(i,j) - g(i,j)| \end{aligned} \tag{3.3}$$

where D-image gives the total colour differences between different colour components. Currently, same weights of one-third has been set for three components.

For grey pixels in image f, they have same values in three colour channels, however, it is difficult that distinguish them from D-image; therefore, another channel image R is obtained by weighting of D-image and luminance intensity image G as follows:

$$R(i, j) = k\frac{\omega_d G(i, j) + \omega_g D(i, j)}{\omega_d + \omega_g} \tag{3.4}$$

where ω_d and ω_g are the weights and determined in (3.5) and (3.6) by the statistical properties of D-image and G-image, and the parameter k is used to scale the determined values of R-image within $[0, 255]$.

$$\omega_d = 1.5 \times Range(D) + \sigma(D) \tag{3.5}$$

$$\omega_g = 1.5 \times Range(G) + \sigma(G) \tag{3.6}$$

Range is a function to determine the valid intensity range of a given image, i.e. difference of the maximum and the minimum intensity values within the image, and σ is the standard deviation.

Generally, (3.4) is used to make the weighted values, $\omega_d G(i, j)$ and $\omega_g D(i, j)$, more comparable to achieve more robust results. In addition, the parameter 1.5 used is empirically determined as it helps to yield particular good results than other values.

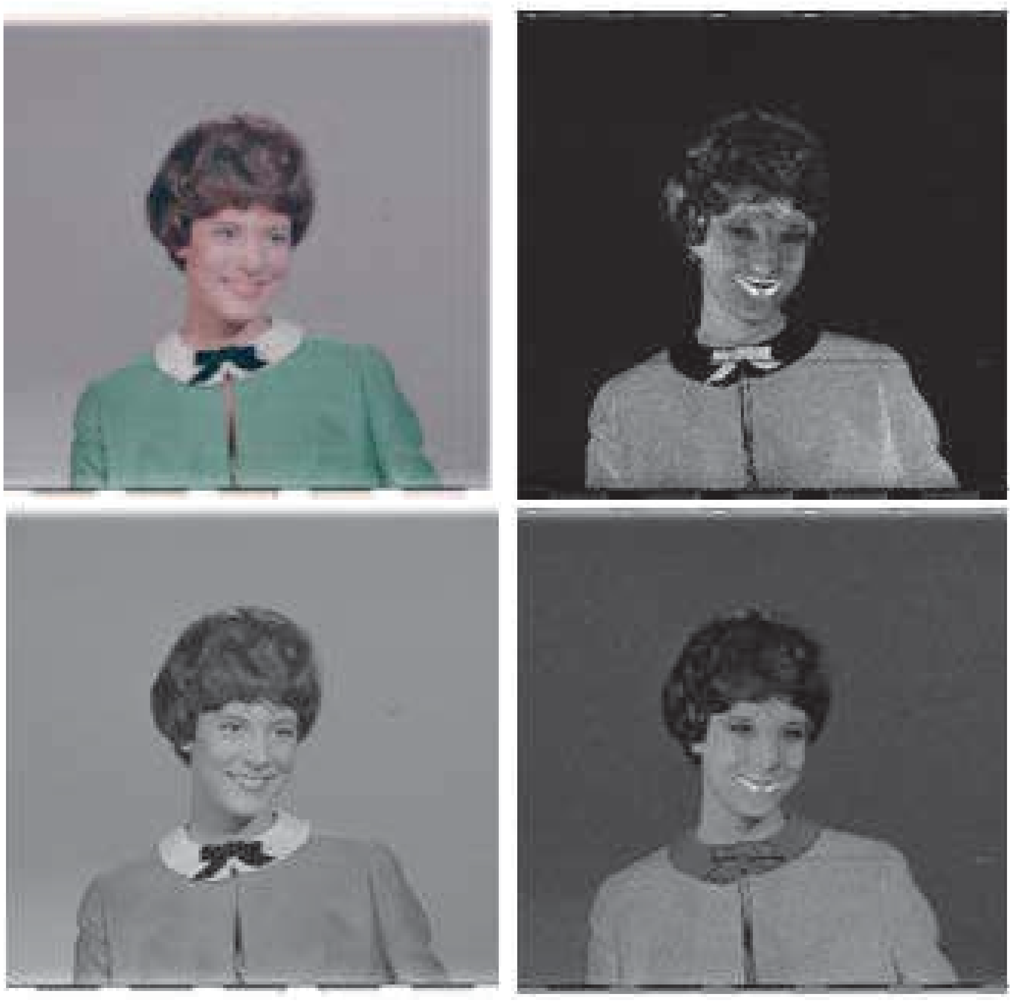


Figure 3.1: One Original Colour Image (top-left) and Its Corresponding Three Single Channel Images Including *D*-image, *G*-image and *R*-image, Respectively.

In comparison with [149], our solution can obtain a new pseudo-grey image without PCA analysis. For a given colour image, its corresponding D-image, G-image and R-image are all illustrated in Fig 3.1 for the comparisons of all.

3.3 Low-level Image Processing Approaches

3.3.1 Colour Edge Detection

Since our main focus is the fusion scheme for improved colour edge detection, standard edge detectors for consistent measurements and evaluations has been used. To this end, the well-known Canny operator is used for its relative good performance.

Taken $f(x)$ and in the interval of $[-w, w]$ as impulse response and bandwidth of the Canny operator, the corresponded filter should make formula (3.7) maximum:

$$\sum = \frac{|\int_{-w}^{0} f(x)dx|}{\sqrt{\int_{-w}^{w} f^2(x)dx}} \cdot \frac{|f'(0)|}{\sqrt{\int_{-w}^{w} f'^2(x)dx}} \tag{3.7}$$

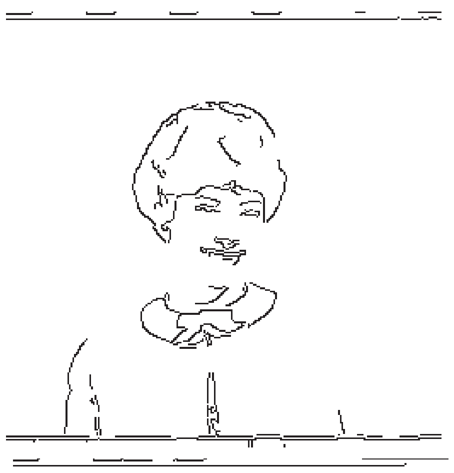
According to canny's theory [53], two parts are included in Eq. (3.7) where the first corresponds to signal to noise ratio achieved for Canny detector, and the second refers to localisation accuracy. By putting these two together, it is expected that the edge detector can reach a trade-off between them. Canny Also proved that the product of the two parts in (3.7) would generate scale-invariant output, which was very important in edge detection.

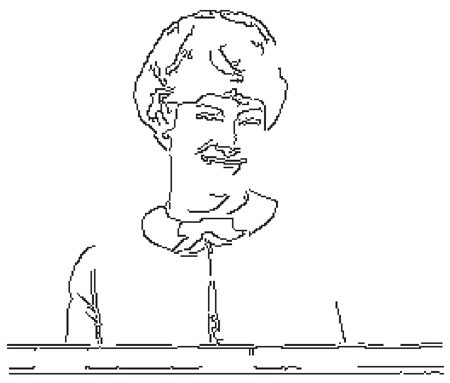
Edges detected by the Canny operator are all local extrema to ensure the precision of detection. Firstly, the grey image is smoothed by a Gauss function. Then, normalized gradient image is obtained from the smoothed image to determine possible edge pixels.

Two thresholds T_h and T_l are used to get continuous and robust results. If a pixel has a gradient more than T_h, it belongs to the edge. If the gradient is less than T_l, it is not edge pixel. Otherwise, the pixel will be determined as edge pixels if there are edge pixels in its neighbourhood and this process helps to improve the continuity of detected edges.

In our experiments, the implementation of the Canny operator in the OpenCV package is adopted for edge detection. As seen, T_h and T_l are two important parameters in the Canny operator and different thresholds will lead to quite different edge results. For the G-image in Fig 3.1, the detected edges used by different thresholds are illustrated in Fig 3.2.

For a given T_h, small T_l can achieve more detail edges, but too small T_l may cause noise (see the girl's face in Fig 3.2). Therefore, how to automatically select suitable thresholds is a basic problem for effective edge detection of Canny operator.


(a) $T_l = 100$


(b) $T_l = 55$

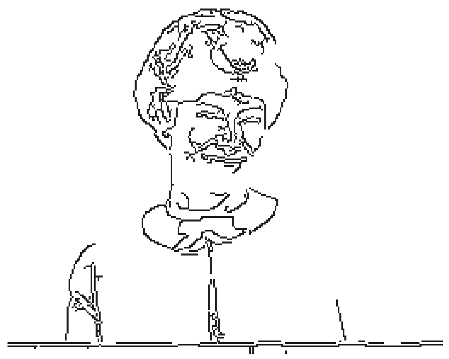

(c) $T_l = 10$

Figure 3.2: Extracted Edges Using the Canny Edge Detector with $T_h = 208$ and T_l Changes from 100, 55 to 10, Respectively

For consistency in evaluations, in our method T_h and T_l are automatically determined as follows:

$$T_h = \mu + \max(\mu/2, \sigma) \tag{3.8}$$

$$T_l = |\mu - \sigma|/2 \tag{3.9}$$

where μ and σ are the mean and standard deviation of any given grey image in this process.

For G-image in Fig 3.1, $T_h = 208$ and $T_l = 55$ can be found, and the corresponding edges is given in Fig 3.2(b), which is better than Fig 3.2(a) and Fig 3.2(c).

For a given colour image, its edges are then extracted as follows.

1. the associate G-image and R-image are obtained based on same source image;
2. by using the Canny operators with automatically determined parameters, edges are detected from these two images as E_G and E_R, respectively;
3. the edges in the image for the colour image E_{final} are determined as follows:

$$E_{final} = E_R \bigcup E_G \tag{3.10}$$

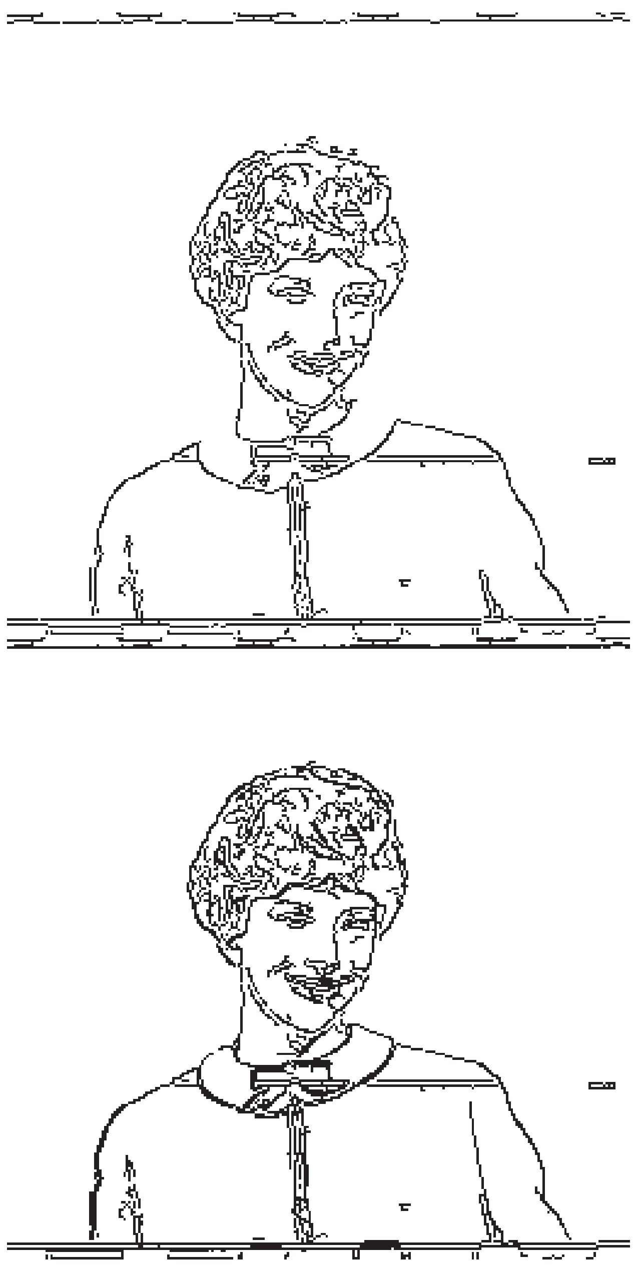


Figure 3.3: E_R (top) and E_{final} (bottom) of Colour Image in Fig 3.1

E_R and E_{final} in Fig. 3.3 have been detected for the colour image in Fig. 3.1 and E_G in Fig. 3.2.

Note that E_G is useful to recover edges from grey part (no component difference) of images, as grey pixels in image f with same values in three colour components will appear as zero and cannot be distinguished from both the D-image and R-image.

3.3.2 Implementation and Experimental Results

The basic problem in colour edge detection is how to fuse chromatic and intensity information for effective colour edge detection. In this implementation, an improved method on colour edge detection is proposed in which the significant advantage is the use of inter-component difference information for effective colour edge detection.

That means, in the processing of this experiment, the key point is extracted the edge from different colour transform space, even one new colour transform space is built: R-Image; in the final step, combine the results from different colour transform spaces in order to improve the performance of edge detection.

Now, the experimental details will be described as follows. According the flow chart of colour edge detection, it shown in Fig 3.4, the experimental details and results will be described step by step:

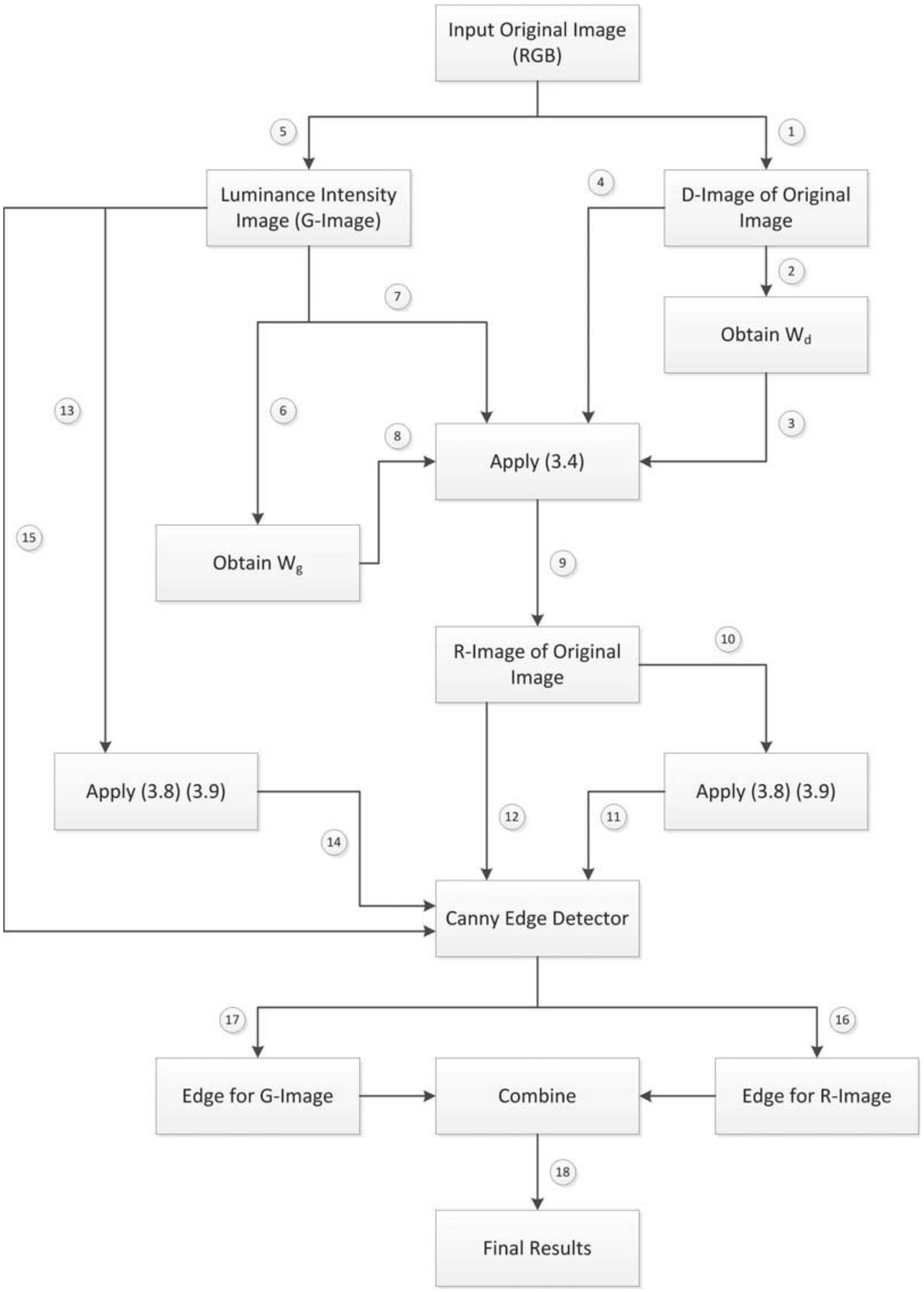


Figure 3.4: Flow chart for Colour Edge Detection

In this detection system, the original image called "Green Girl", it shown in Fig 3.5:

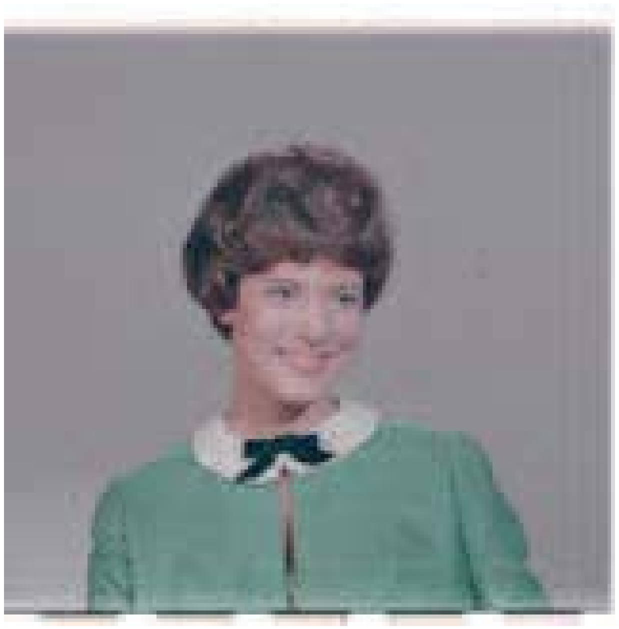


Figure 3.5: Original Image "Green Girl"

①: Apply the (3.3) on Fig 3.5, the result is Fig 3.6, it called D-Image of Original Image:

$$\begin{aligned} D(i,j) &= \omega_1|r(i,j)-g(i,j)| \\ &+ \omega_2|r(i,j)-b(i,j)| \\ &+ \omega_3|b(i,j)-g(i,j)| \end{aligned}$$

②: Apply the (3.5) on Fig 3.6 in order to calculate the ω_d:

$$\omega_d = 1.5 \times Range(D) + \sigma(D)$$

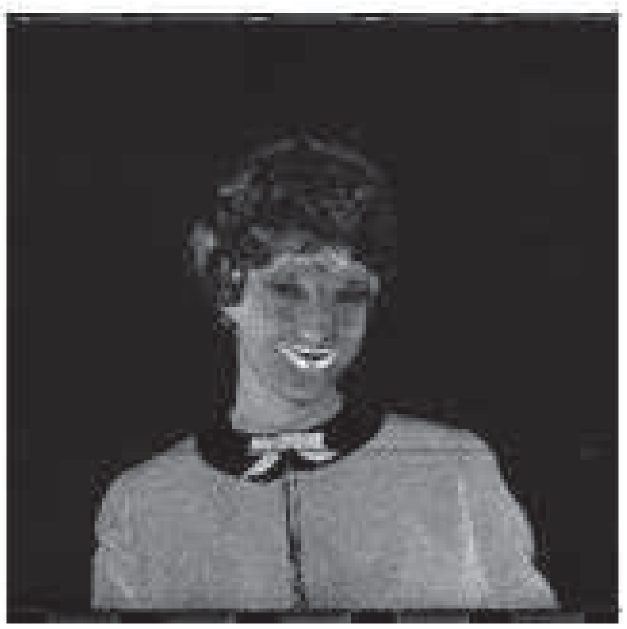


Figure 3.6: D-Image for Original Image

$\omega_d = 72.83$
Range : difference of the maximum and the minimum intensity values within the image
$Range(D) = 44$
σ : standard deviation
$\sigma(D) = 6.83$

③: Input value of ω_d to (3.4)

④: Input value of D-image to (3.4)

$$R(i,j) = k\frac{\omega_d G(i,j) + \omega_g D(i,j)}{\omega_d + \omega_g}$$

⑤: Obtain G-image from original image (RGB), the result shown in Fig

3.7

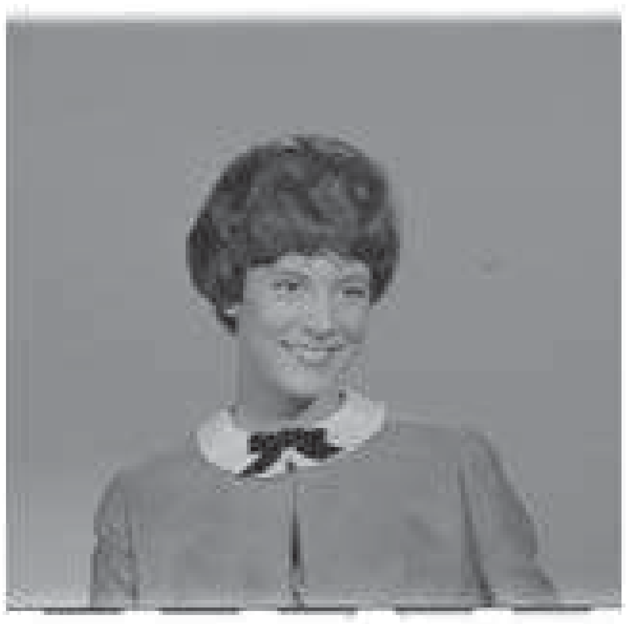


Figure 3.7: G-Image for Original Image

⑥: Apply (3.6) on Fig 3.7 in order to calculate the ω_g:

$$\omega_g = 1.5 \times Range(G) + \sigma(G)$$
$$\omega_g = 355.83$$
$$Range(G) = 226$$
$$\sigma(G) = 16.83$$

⑦: Input the value of G-Image to (3.4)

⑧: Input the value of ω_g to (3.4)

⑨: Obtain the R-image, it shown in Fig 3.8

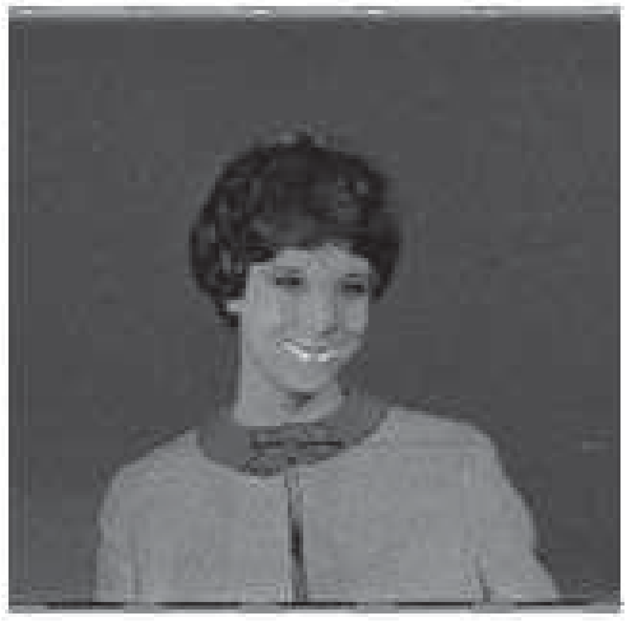


Figure 3.8: R-Image for Original Image

⑩: Apply (3.8) (3.9) in order to obtain T_h and T_l for R-image:

$$T_h = \mu + \max(\mu/2, \sigma)$$
$$T_l = |\mu - \sigma|/2$$

⑪: Input the results of ⑩ to Canny edge detector

⑫: Apply the edge detector that from step ⑪ on R-image

⑬: Apply (3.8) (3.9) in order to obtain T_h and T_l for G-image

$$T_h = 208$$
$$T_l = 55$$

⑭: Input the results of ⑬ to Canny edge detector

⑮: Apply the edge detector that from step ⑭ on G-image

⑯: Obtain the edge for R-image, result shown in Fig 3.9

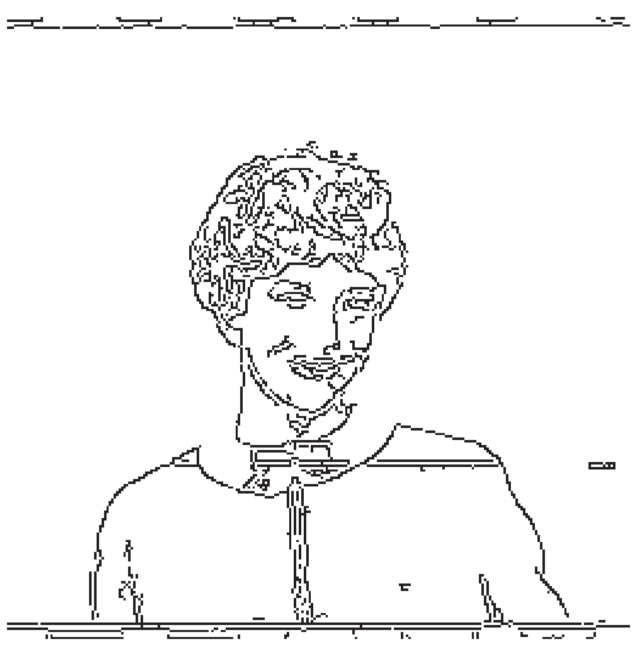


Figure 3.9: Edge for R-image

⑰: Obtain the edge for R-image, result shown in Fig 3.10

⑱: Combine the results from step ⑯ and ⑰, apply (3.10), result shown in Fig 3.11

$$E_{final} = E_R \bigcup E_G$$

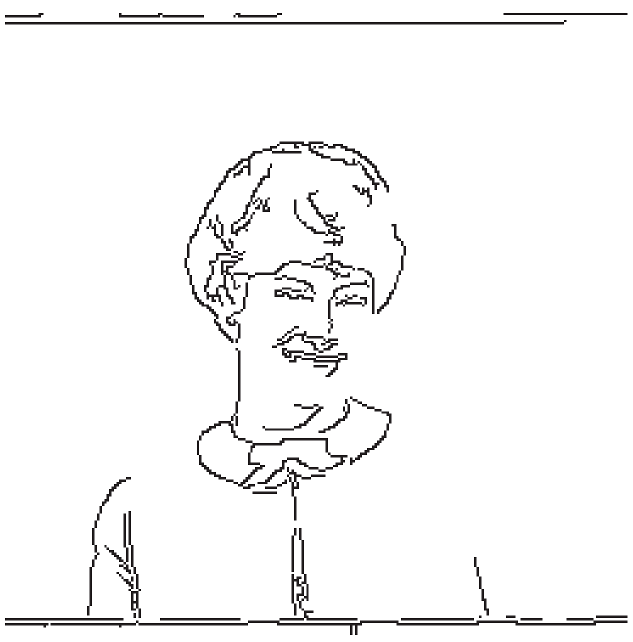


Figure 3.10: Edge for R-image

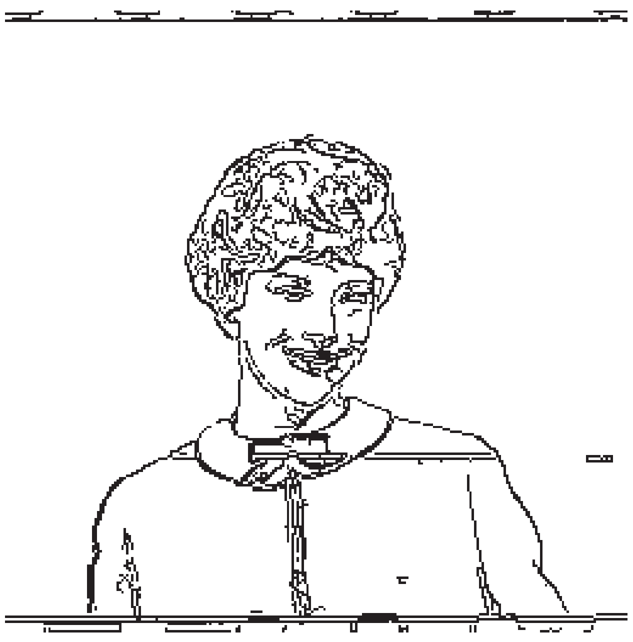


Figure 3.11: Final Result

In order to evaluate this fusion scheme in improved colour edge detection, the edges from RGB, YCbCr and HSV spaces will be taken for comparisons. RGB space has been selected because it is widely used for colour representation especially in computer graphics. On the other hand, YCbCr and HSV spaces for two reasons can be chosen:

1. both of them reflect certain human perceptions of colour;
2. either of them represents one group of similar colour spaces, such as YCbCr can represent YUV, and HSV can also represent HSL or HSI etc;

For the three colour spaces above, corresponding edges are extracted as follows: Firstly, all the colour components have their values normalized within [0,255]. Secondly, edges are extracted from each colour component using the Canny operator with automatically determined parameters. Thirdly, the final edge for each image is obtained as the union of edges extracted from each colour component.

For the original colour image "green girl" in Fig. 3.1, Fig. 3.12 illustrates edges extracted from RGB, YCbCr, HSV spaces and our fusion scheme for comparisons. As seen, our method has more continuous (see closed face contour) and accurate edges than edges from RGB and YCbCr spaces, and RGB edges is much better than YCbCr edges. In addition, edges extracted from HSV space have too much noise.

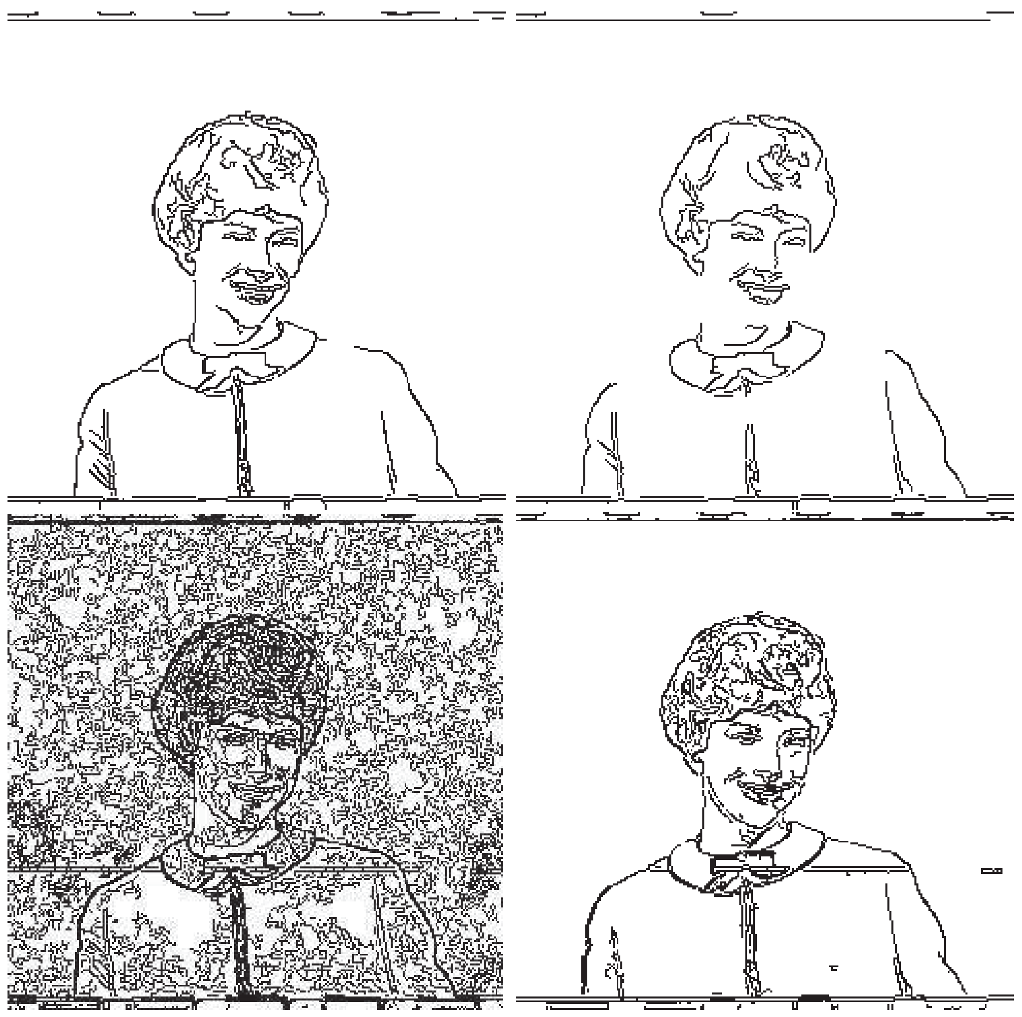


Figure 3.12: Comparing Extracted Edges (from top-left to bottom-right): the Images Are Edges Detected From RGB, YCbCr, HSV Spaces and Our Method From the Original Colour Image in Fig. 3.1

3.3.2.1 Effectiveness Evaluation

To further evaluate the effectiveness of our proposed algorithm, edges extracted from both synthetic and real images are further compared. Three standard test images, "lena", "pepper" and "house", as shown in Fig. 3.12, are used in this group of experiments. For quantitative evaluation, ground truth of edge images are defined as reference images. These ground truth images are produced in a semi-manual way containing two steps:

1. Extracting edges for each image using the logical OR of the results from RGB and YCbCR spaces;
2. manual refinement of the extracted results to remove ghost edges and noise, etc;

Basically, the precision rate η_p and recall rate detected η_r are defined for such evaluations as follows:

$$\eta_p = \frac{tp}{tp + fp} = \frac{|E_{ref} \bigcap E_{det}|}{|E_{det}|} \tag{3.11}$$

$$\eta_r = \frac{tp}{tp + fn} = \frac{|E_{ref} \bigcap E_{det}|}{|E_{ref}|} \tag{3.12}$$

where E_{ref} and E_{det} denote reference (as ground truth) and detected edge results; tp and fp refer respectively to true positive (correct detected) and false positive (false alarm) samples, and fn denotes false negative (missing in detection) samples. The samples are counted as number of edge pixels in the images accordingly.

For each test image, seven edge results are compared including those extracted from colour spaces of RGB, YCbCr and HSV, single component of the grey image, the green channel and a pseudo-grey channel [149] as well as our fusion scheme. Since this pseudo-grey component is attained via principal component analysis of the three colour channels, eigen edges simply have been named as its results.

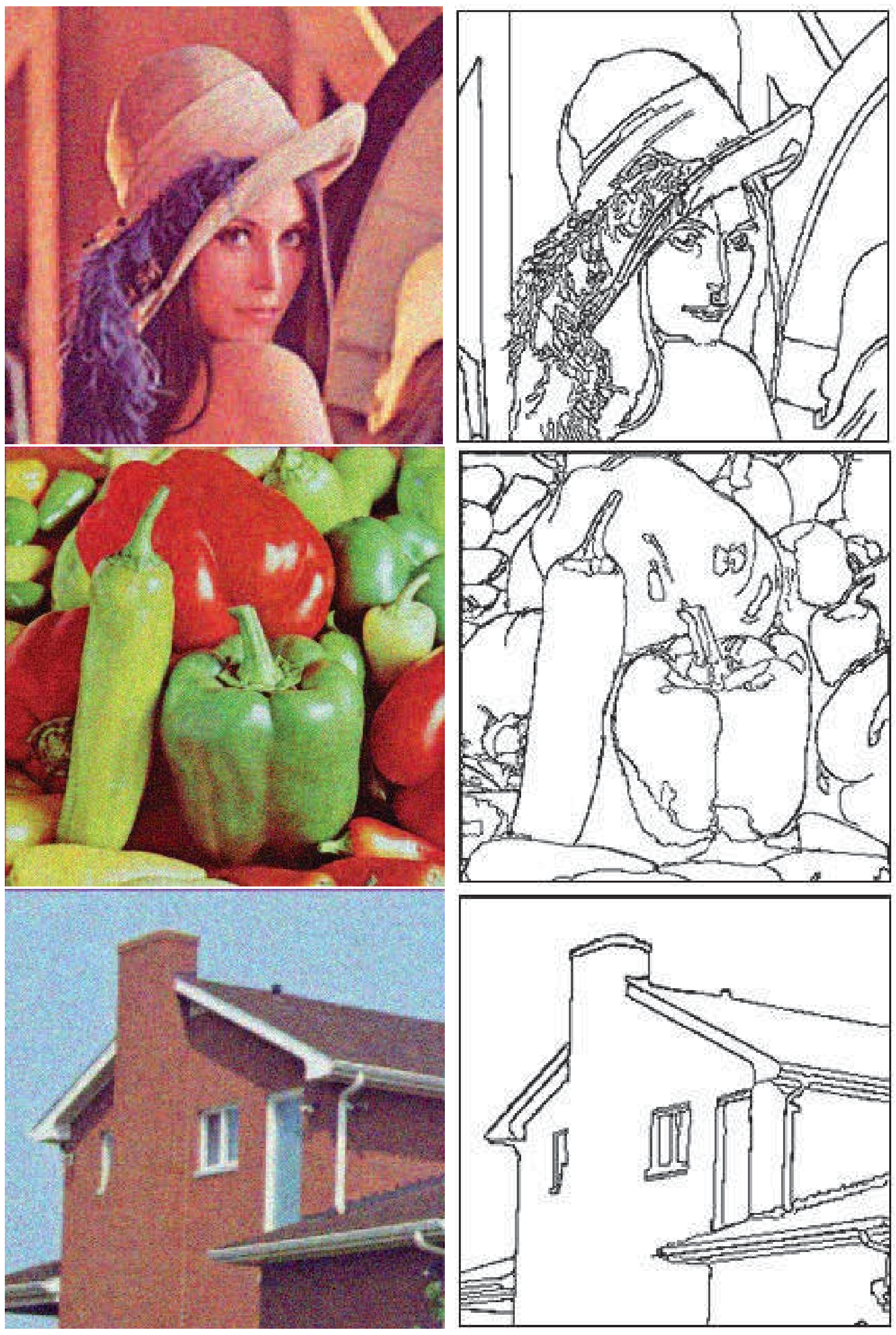
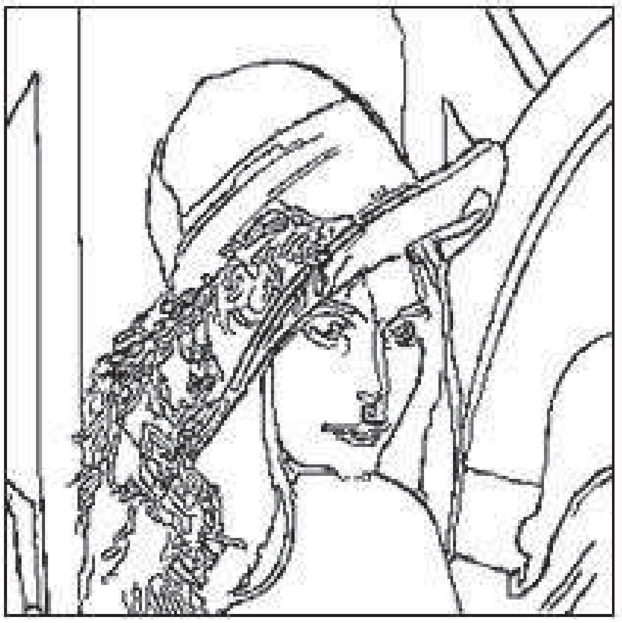


Figure 3.13: Three Test Images (left) and Associated Ground Truth Images of Edges (right)

From Fig 3.13-Fig 3.19, visual comparisons depend on 'Lena'.


Ground Truth Image

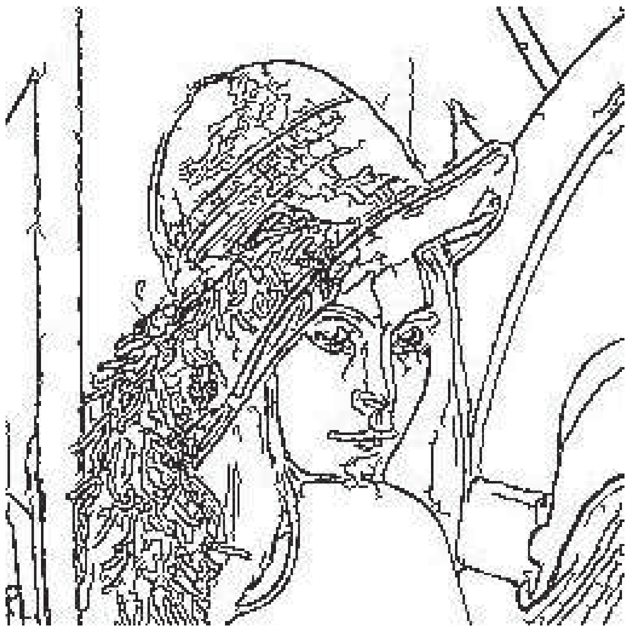
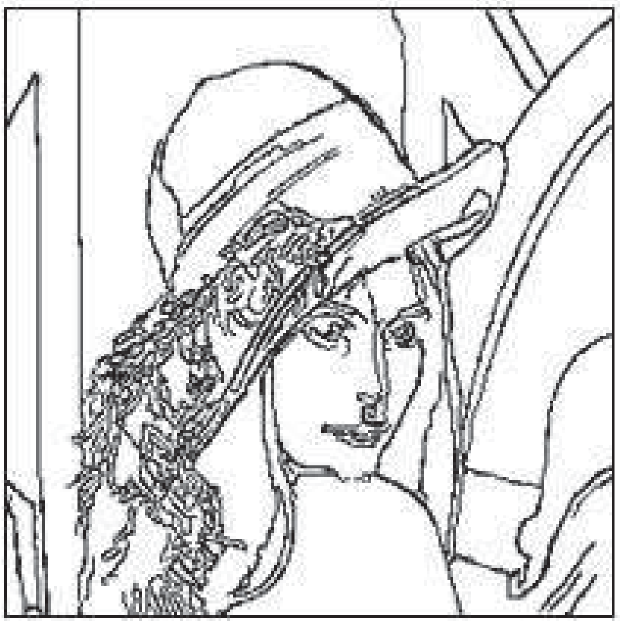

RGB

Figure 3.14: Compare Ground Truth and RGB, 'Lena'

Compare between Ground Truth Image and RGB, there are so many false alarm, especially on the hat and about the girl's nose.


Ground Truth Image

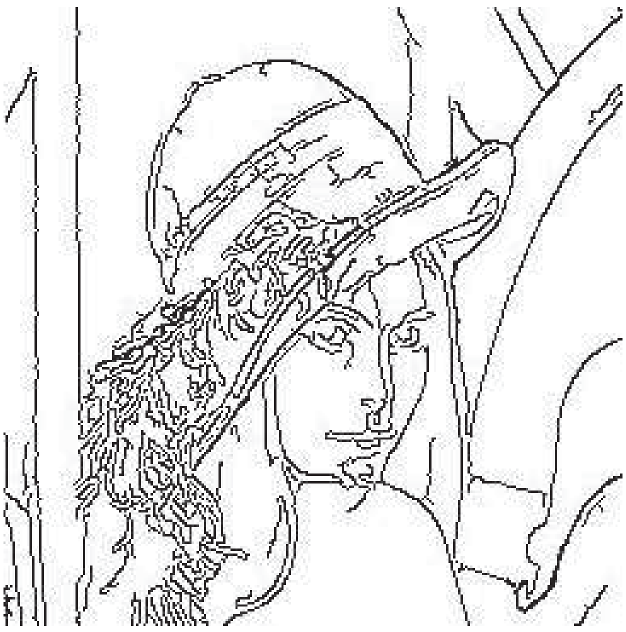
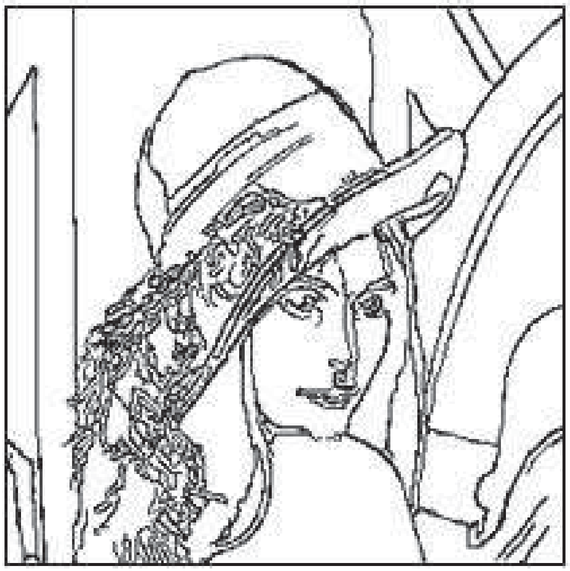

YCbCr

Figure 3.15: Compare Ground Truth and YCbCr, 'Lena'

Compare between Ground Truth Image and YCbCr, the edges extracted from YCbCr is better than the edge from extracted from RGB, especially on the hat. But there are still a few false alarms under the girl's nose and under the girl's mouth.


Ground Truth Image

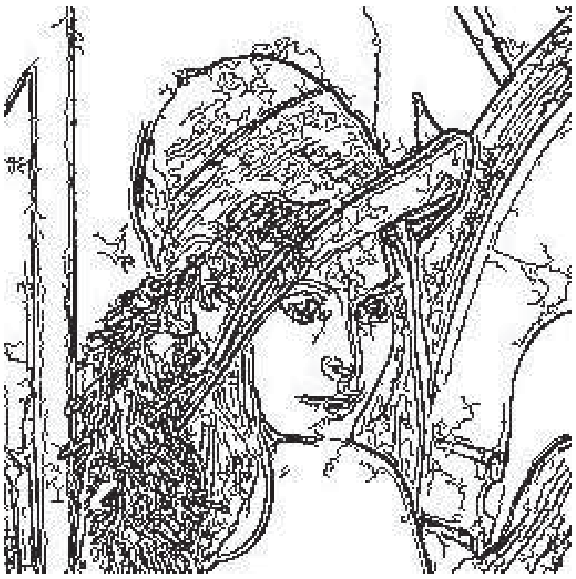

HSV

Figure 3.16: Compare Ground Truth and HSV, 'Lena'

Compare between Ground Truth Image and HSV, there are massive false alarms in everywhere, face, hat, arms, etc. The edges extracted from HSV even worse than from RGB.

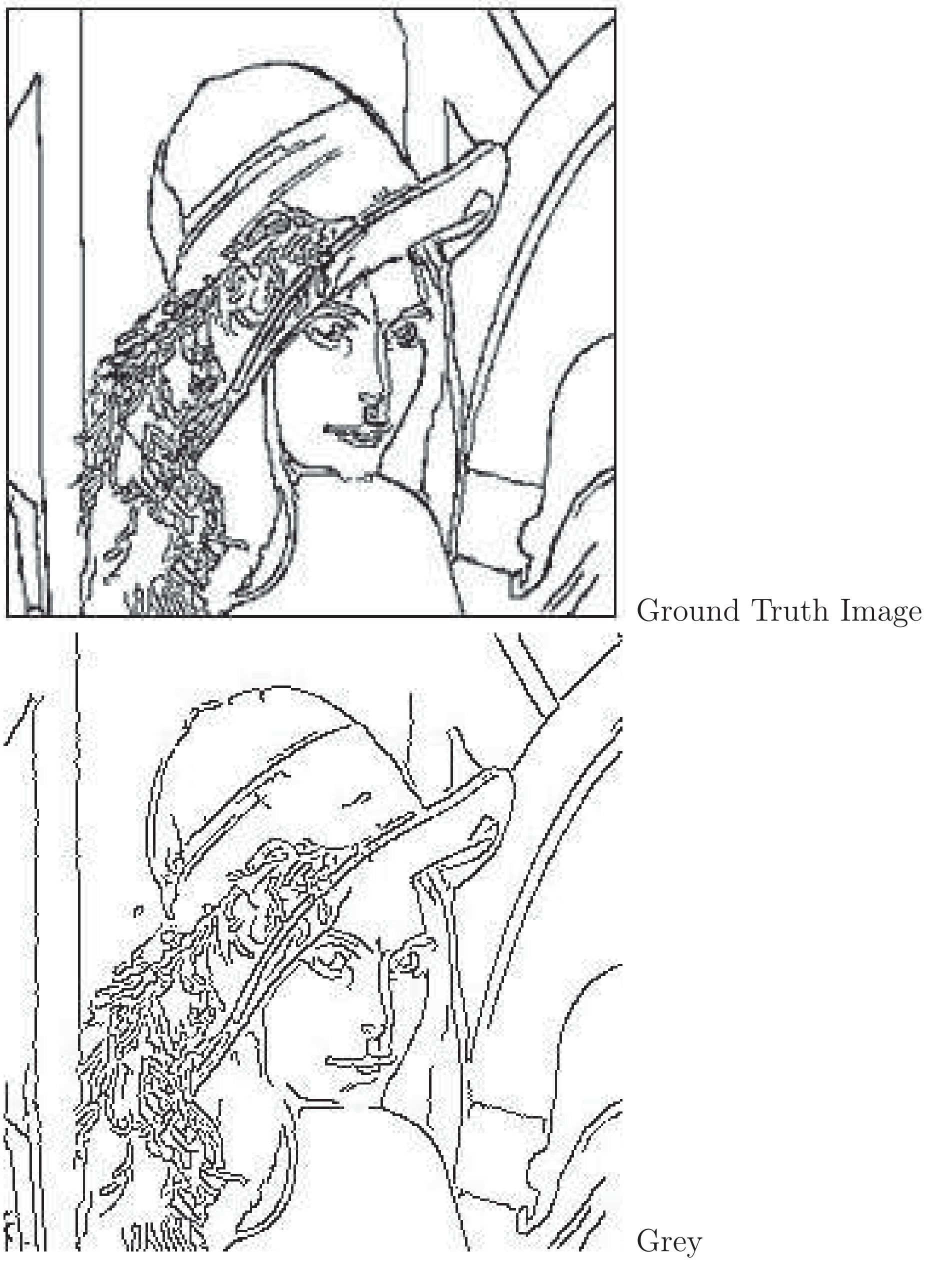


Figure 3.17: Compare Ground Truth and Grey, 'Lena'

Compare between Ground Truth Image and Grey, the edges extracted from grey are quite good, there is less false alarm, but got missing edges, like hat, hair, etc.

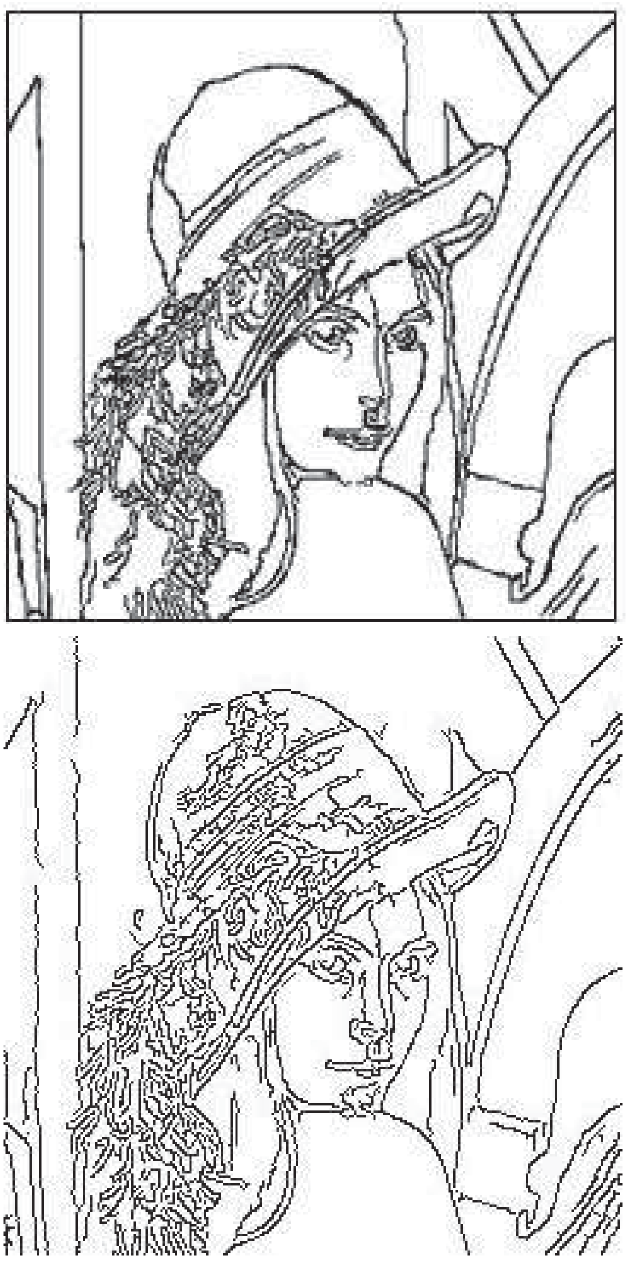
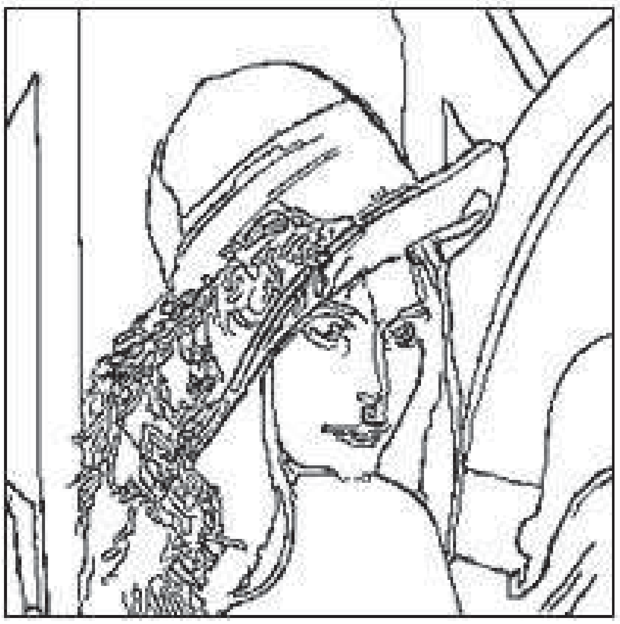


Figure 3.18: Compare Ground Truth and Green, 'Lena'

Compare between Ground Truth Image and Green, the false alarms in quite massive in the hat, face, etc.


Ground Truth Image

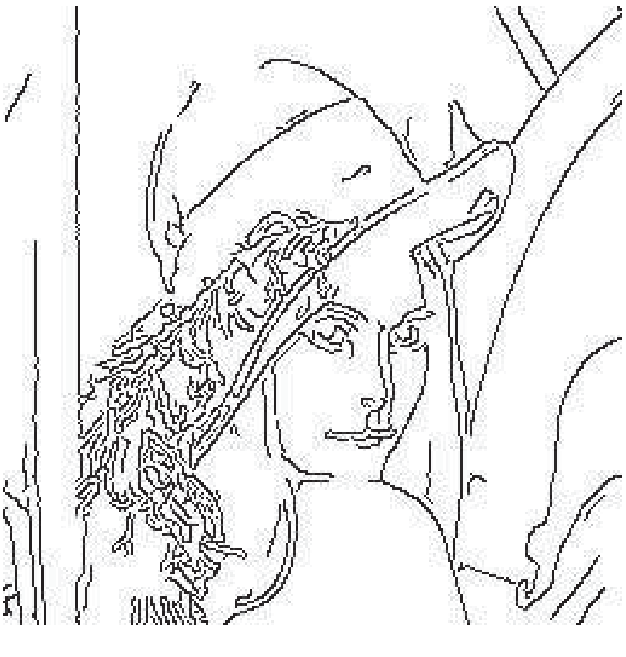
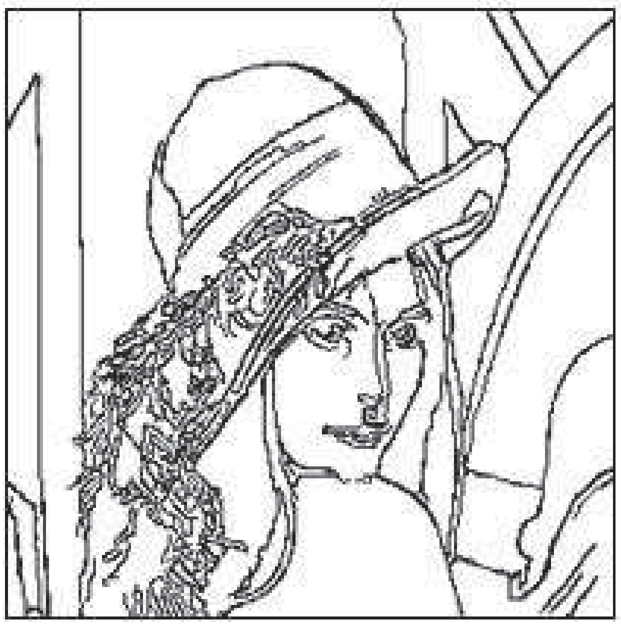

Eigen

Figure 3.19: Compare Ground Truth and Eigen, 'Lena'

Compare between Ground Truth Image and Eigen, the false alarms almost lost, but there are a quite lot of missing edges, like the boundary of the hat, the chin, hair, etc.


Ground Truth Image

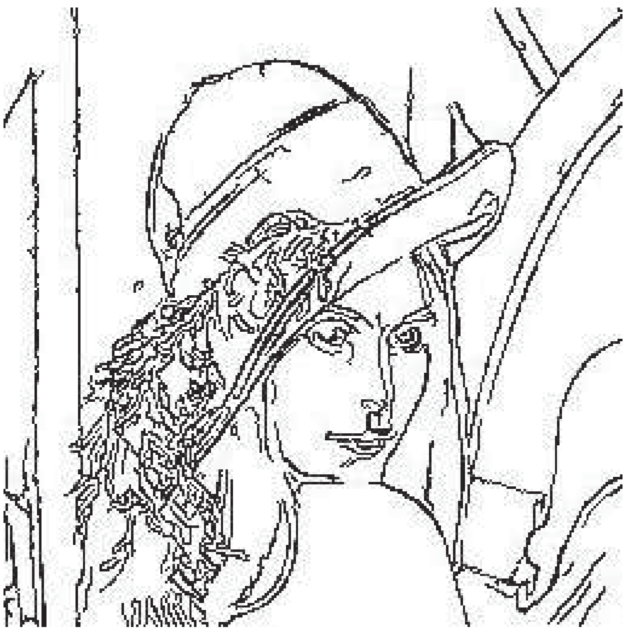
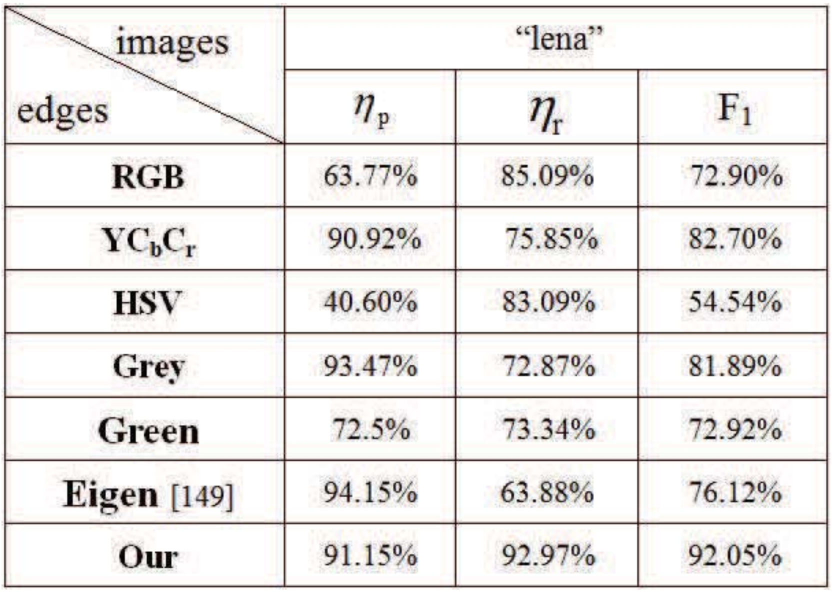

Our Approach

Figure 3.20: Compare Ground Truth and our approach, 'Lena'

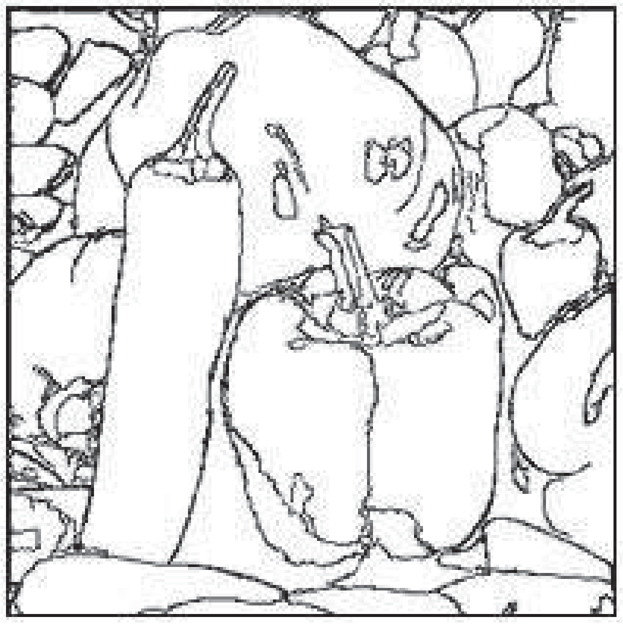
Compare between Ground Truth Image and our approach, there are missing edges in some places, like a little bit on the boundary of the face, the chin, and missing one line on the hat. But, for visual comparisons, our approach has the best performance in all these 7 ways. The precision rate, recall rate and F_1 score for these 7 methods are shown in the Table 3.1.

Table 3.1: Quantitative Evaluations of Edges Detected from the Test Image, 'Lena'



images / edges	"lena"		
	η_p	η_r	F_1
RGB	63.77%	85.09%	72.90%
YC_bC_r	90.92%	75.85%	82.70%
HSV	40.60%	83.09%	54.54%
Grey	93.47%	72.87%	81.89%
Green	72.5%	73.34%	72.92%
Eigen [149]	94.15%	63.88%	76.12%
Our	91.15%	92.97%	92.05%

From Fig 3.20-Fig 3.26, visual comparisons depend on 'Pepper':



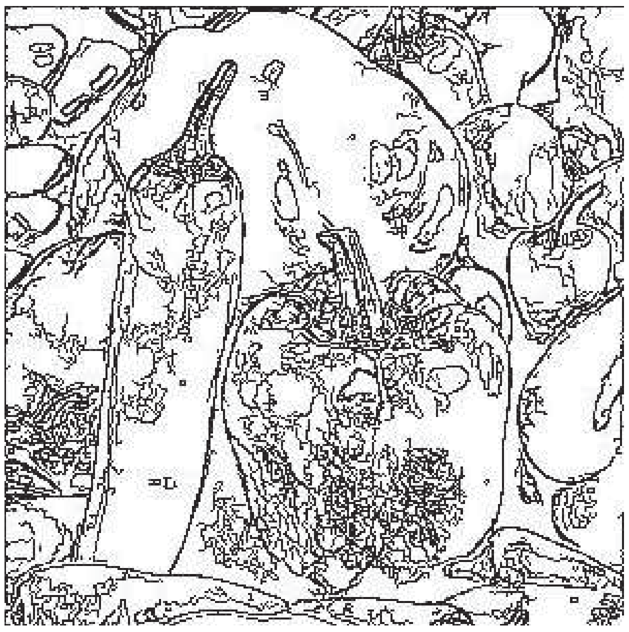
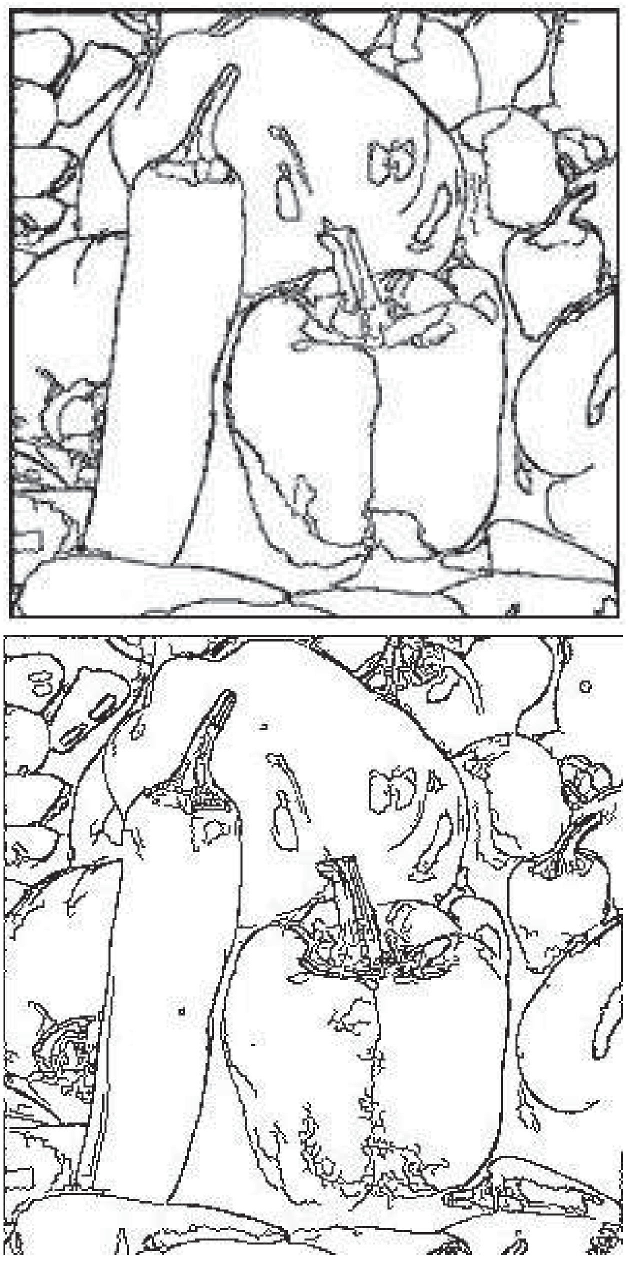


Figure 3.21: Compare Ground Truth and RGB, 'Pepper'

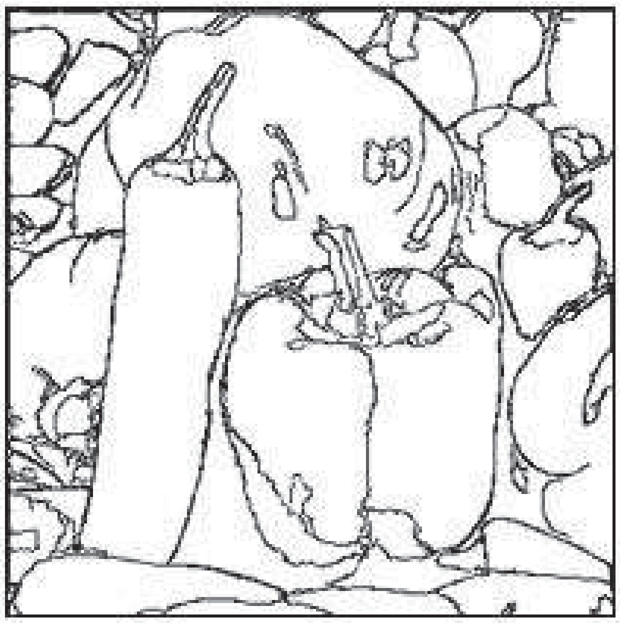
Compare Ground Truth and RGB, there are so many false alarms on the surface of the pepper.


Ground Truth Image

YCbCr

Figure 3.22: Compare Ground Truth and YCbCr, 'Pepper'

Compare Ground Truth and YCbCr, the edges extracted from YCbCr is better than extracted from RGB, but still have false alarms in some places, like in the surface of the big pepper and left side of long pepper.


Ground Truth Image

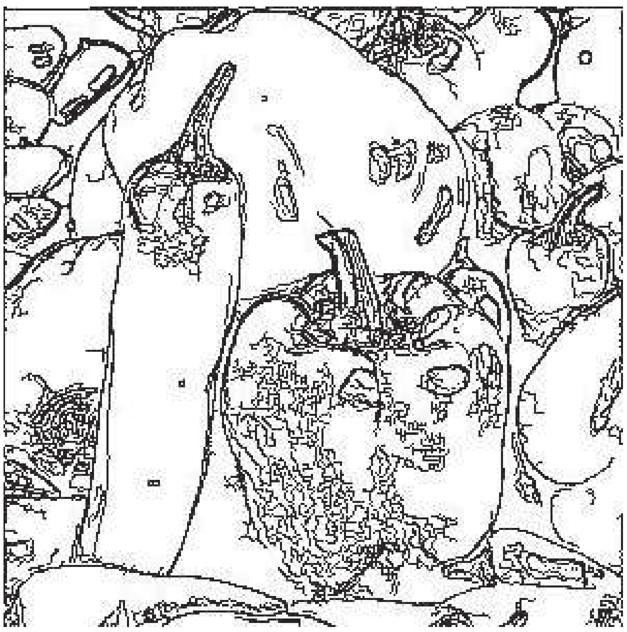
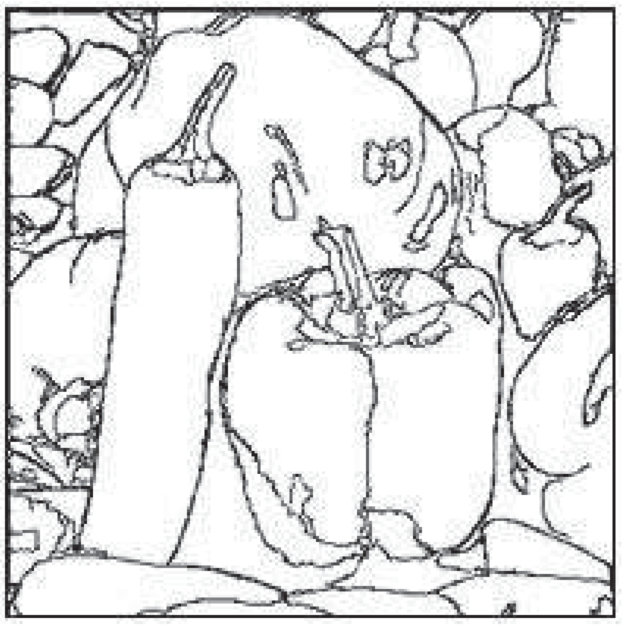

HSV

Figure 3.23: Compare Ground Truth and HSV, 'Pepper'

Compare Ground Truth and HSV, false alarms appear in the surface of the bigger pepper and the top of the long pepper.


Ground Truth Image

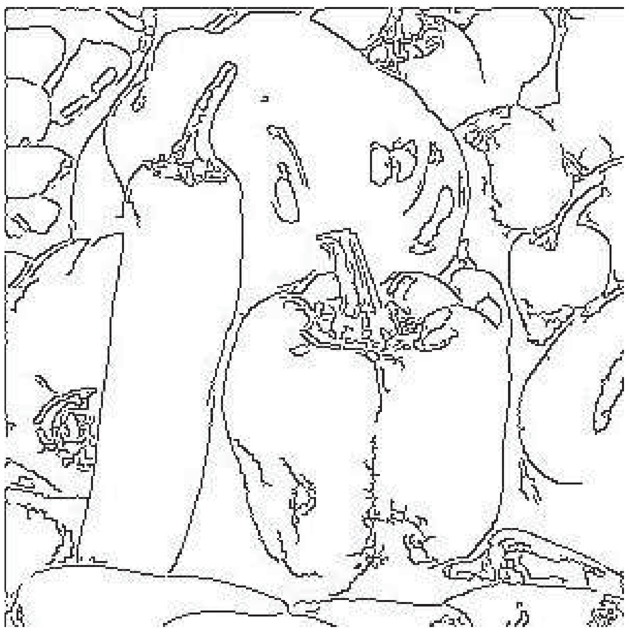
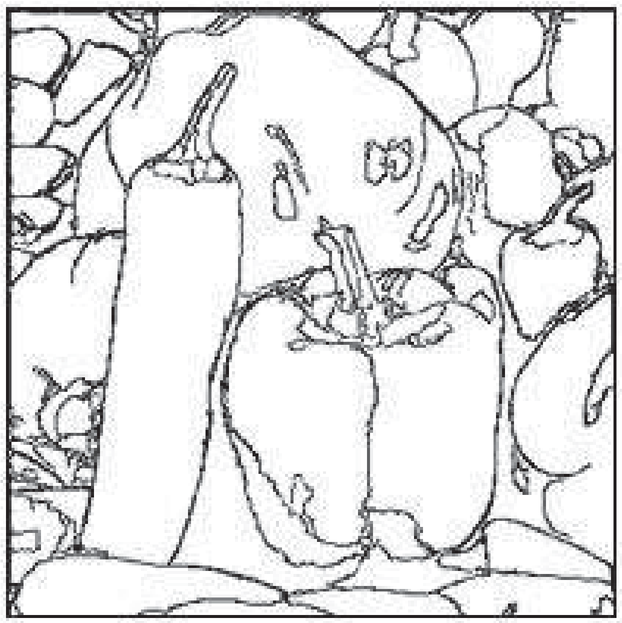
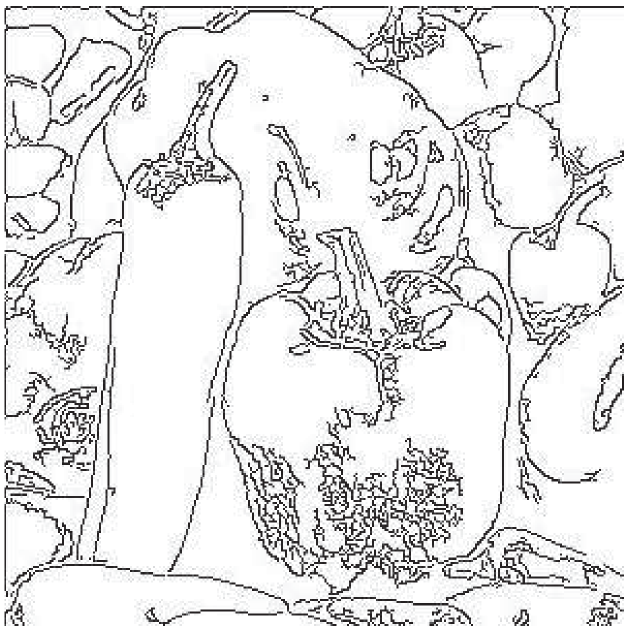

Grey

Figure 3.24: Compare Ground Truth and Grey, 'Pepper'

Compare Ground Truth and grey, the performance is much better, there are still have false alarm in the middle of the bigger pepper.



Ground Truth Image



Green

Figure 3.25: Compare Ground Truth and Green, 'Pepper'

Compare Ground Truth and Green, there are so many false alarms in the surface of the bigger pepper.

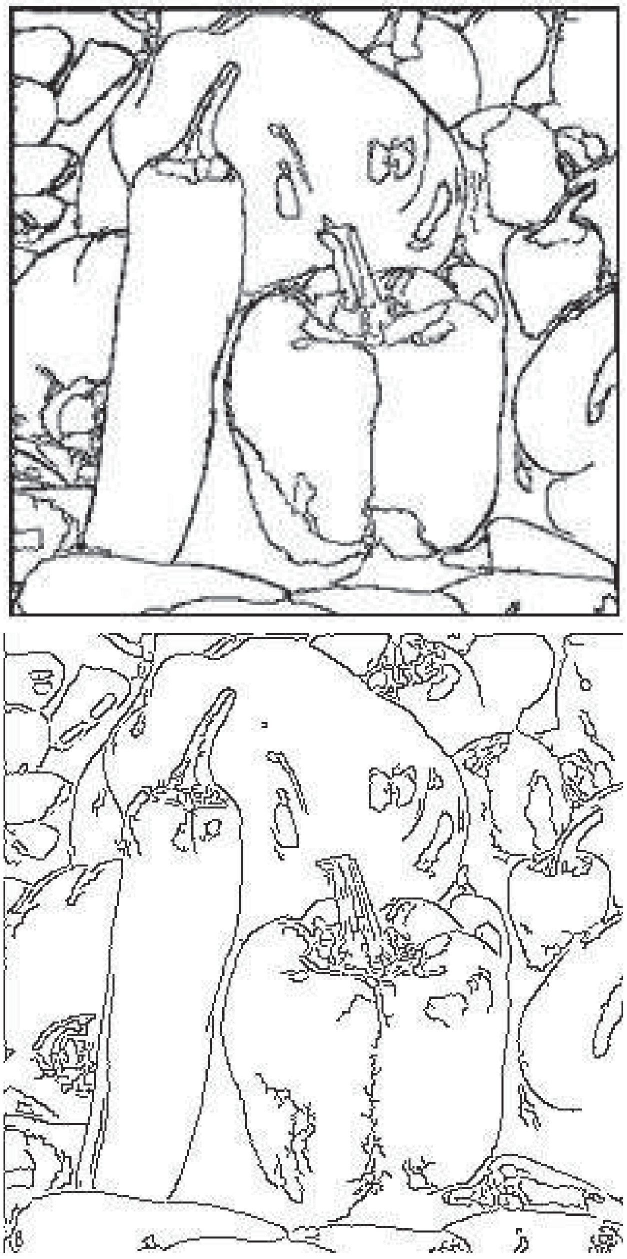
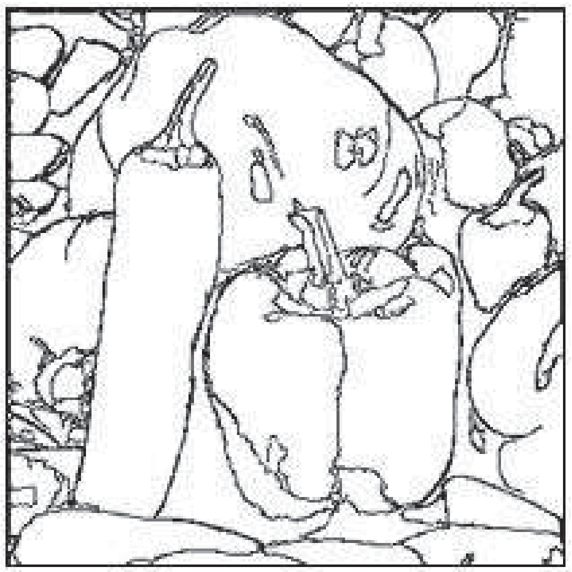


Figure 3.26: Compare Ground Truth and Eigen, 'Pepper'

Compare Ground Truth and Eigen, there are false alarms located in the left side of the long pepper, left side and top right side of the bigger pepper. Same time, the boundary of the bigger pepper could not connect, missing edges.


Ground Truth Image

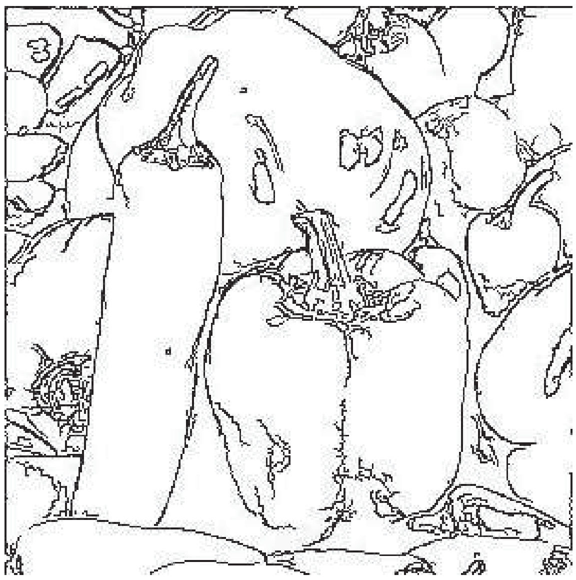

Our Approach

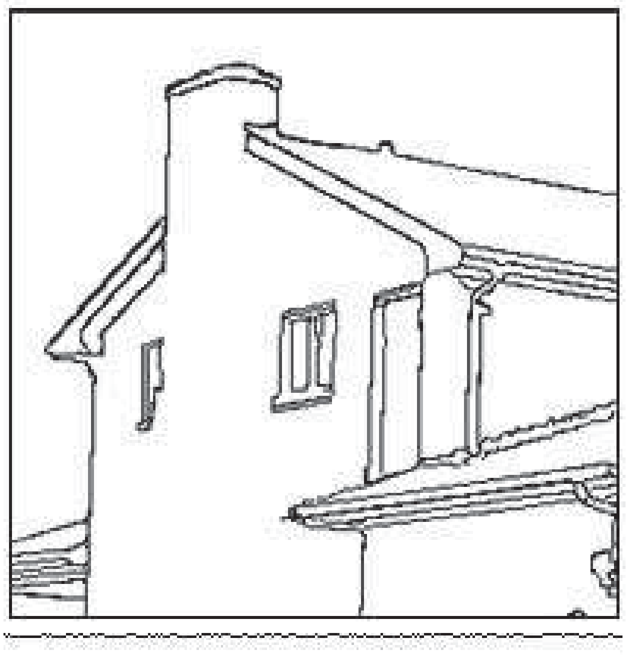
Figure 3.27: Compare Ground Truth and Our Approach, 'Pepper'

Compare Ground Truth and Our Approach, there are still having a few false alarms on the surface of the bigger pepper, and missing edges located on the bottom of the bigger pepper. But for long pepper, the performance is quite good. The precision rate, recall rate and score for these 7 methods are shown in the Table 3.2.

Table 3.2: Quantitative Evaluations of Edges Detected from the Test Images, 'Pepper'

images / edges	"pepper"		
	η_p	η_r	F_1
RGB	37.12%	84.44%	51.57%
YC_bC_r	66.75%	85.45%	74.95%
HSV	41.72%	82.27%	55.36%
Grey	78.75%	67.37%	72.61%
Green	58.22%	64.00%	60.97%
Eigen [149]	67.57%	64.01%	65.74%
Our	78.59%	84.98%	81.66%

From Fig 3.27-Fig 3.33, visual comparisons depend on 'house'.


Ground Truth Image

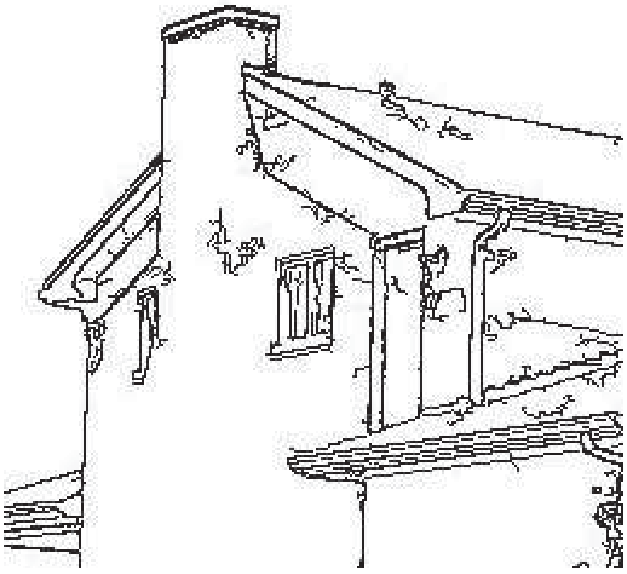
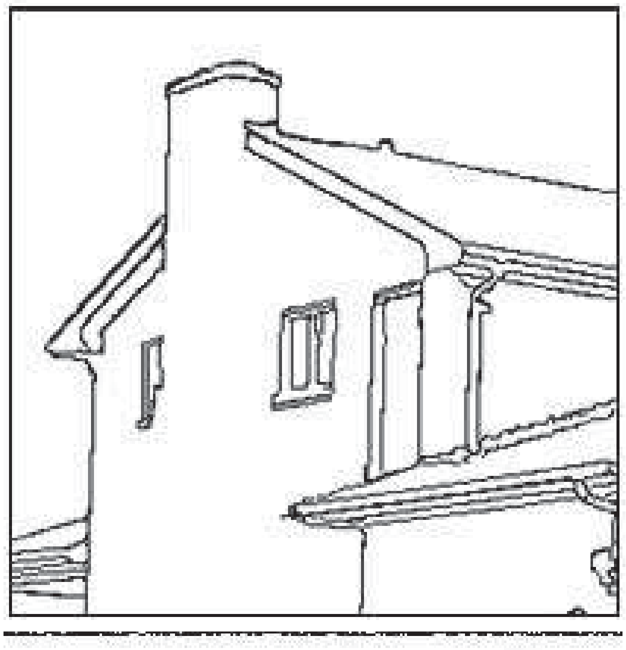

RGB

Figure 3.28: Compare Ground Truth and RGB, 'House'

Compare Ground Truth and RGB, there are so many false alarms appear on the roof, wall, and in the window's area, etc.


Ground Truth Image

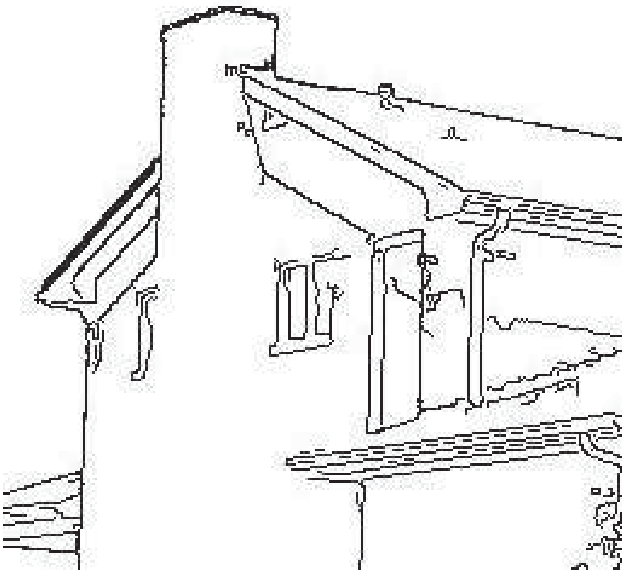
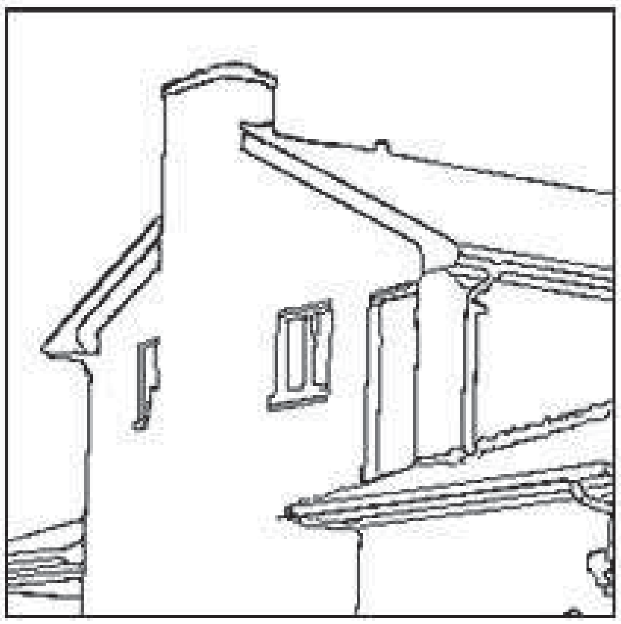

YCbCr

Figure 3.29: Compare Ground Truth and YCbCr, 'House'

Compare Ground Truth and YCbCr, the false alarms are almost lost, but missing edges appeared around the windows.


Ground Truth Image

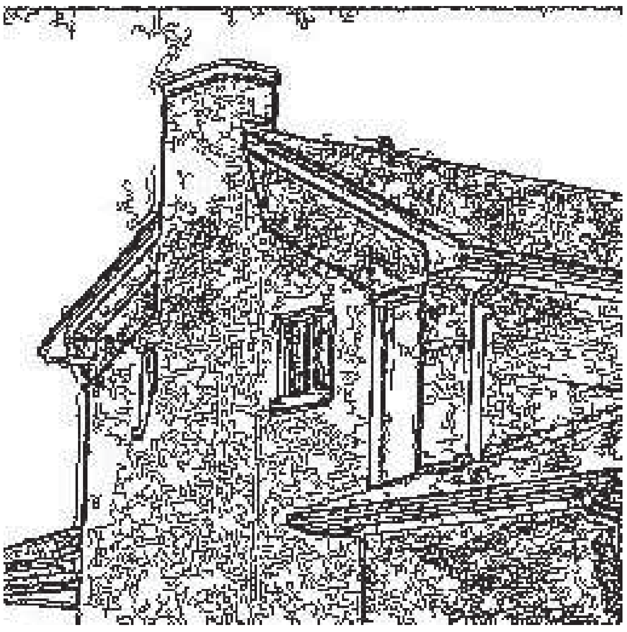
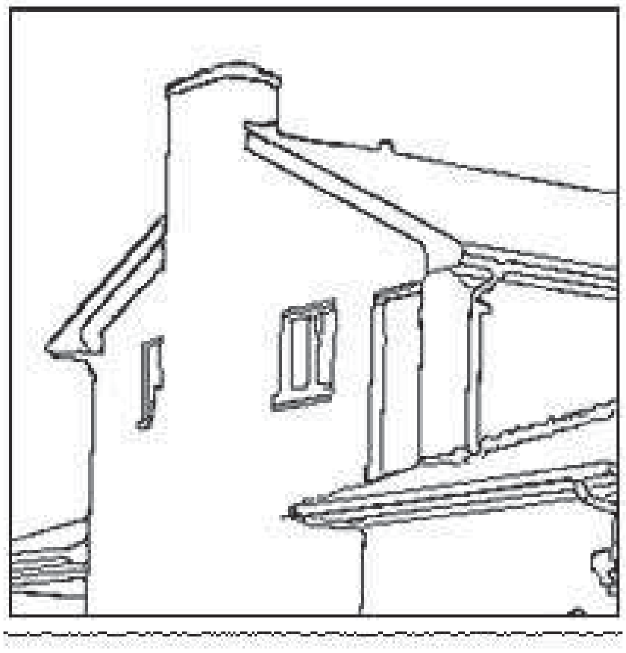
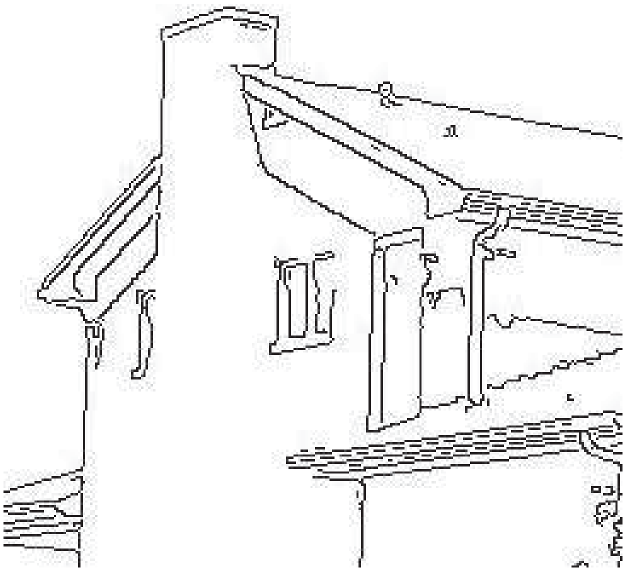

HSV

Figure 3.30: Compare Ground Truth and HSV, 'House'

Compare Ground Truth and HSV, the massive false alarms in everywhere.



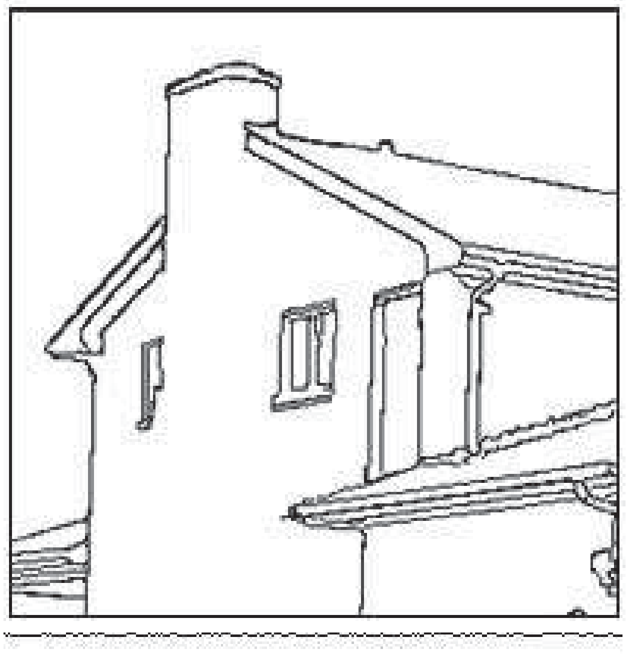
Ground Truth Image



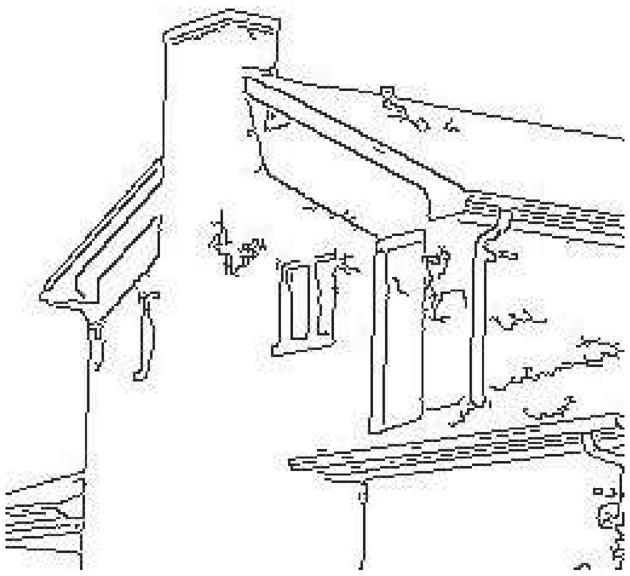
Grey

Figure 3.31: Compare Ground Truth and Grey, 'House'

Compare Ground Truth and Grey, the performance is quite good except there are missing edges on the top right corner of the window.



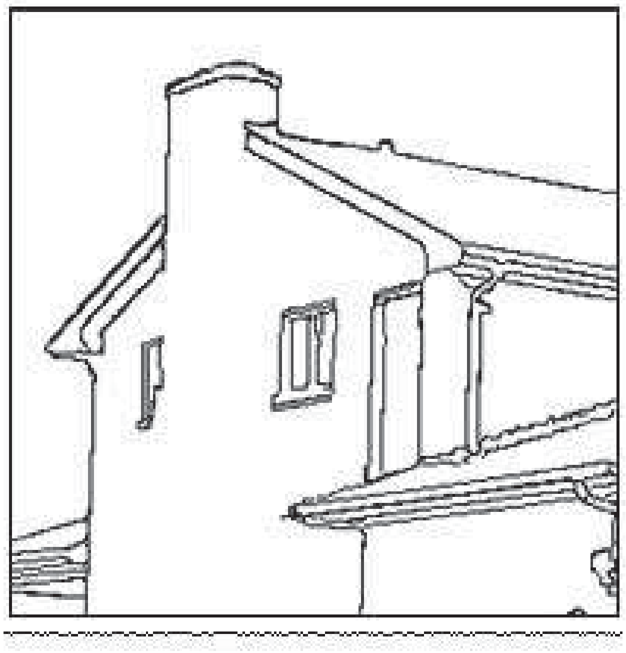
Ground Truth Image



Green

Figure 3.32: Compare Ground Truth and Green, 'House'

Compare Ground Truth and Green, the false alarms appeared on the wall, on the corner of the window.


Ground Truth Image

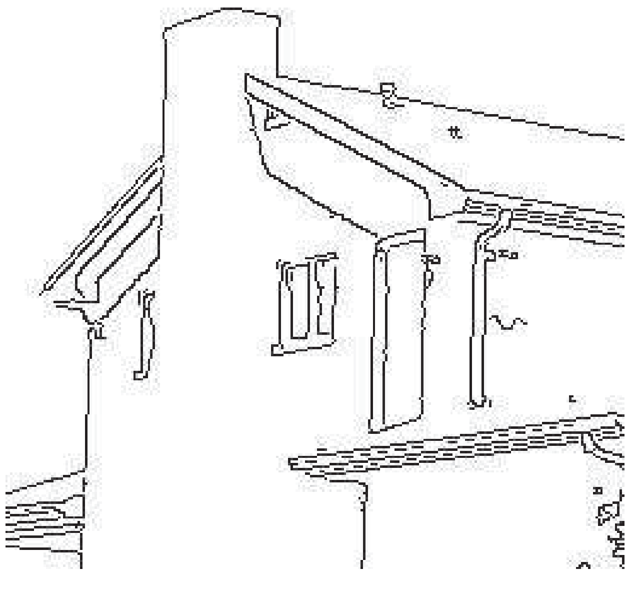
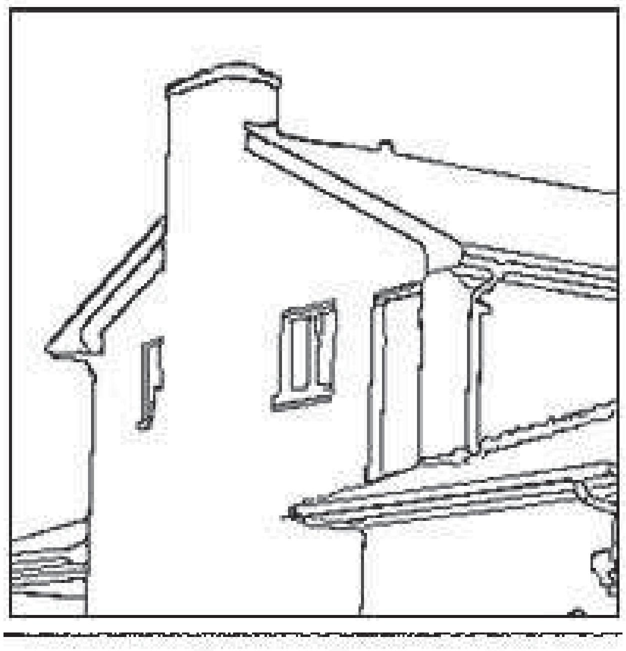

Eigen

Figure 3.33: Compare Ground Truth and Eigen, 'House'

Compare Ground Truth and Eigen, a few false alarms appeared on the wall and roof, but missing edges on the top of chimney.


Ground Truth Image

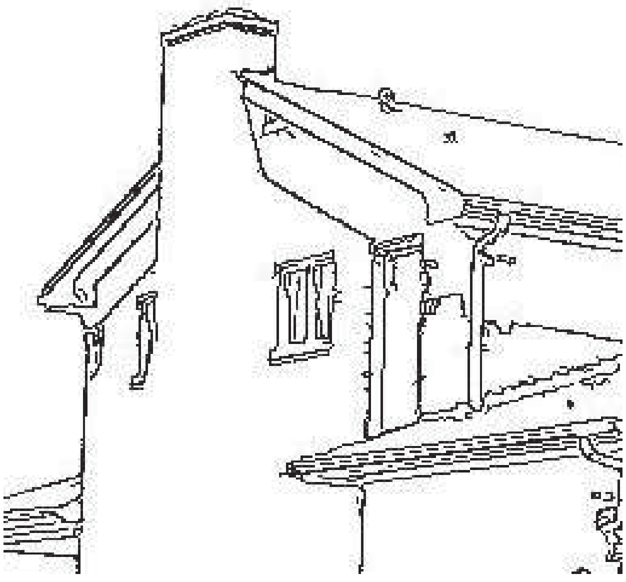

Our Approach

Figure 3.34: Compare Ground Truth and Our Approach, 'House'

Compare Ground Truth and Our Approach, false alarms appeared on the roof, and in the area of the window. The precision rate, recall rate and F_1 score for these 7 methods are shown in the Table 3.3.

Table 3.3: Quantitative Evaluations of Edges Detected from the Test Images, 'House'

images / edges	"house"		
	η_p	η_r	F_1
RGB	55.86%	85.41%	67.54%
YC_bC_r	71.92%	80.68%	76.05%
HSV	18.15%	89.17%	30.16%
Grey	73.51%	72.00%	72.75%
Green	65.95%	69.37%	67.61%
Eigen [149]	74.03%	65.21%	69.34%
Our	71.91%	91.53%	80.54%

Combine the Table 3.1, Table 3.2, Table 3.3, and provide one more item called 'average', the new table shown in Table 3.4. In Table 3.1, Tabel3.2, Table3.3 and Table 3.4, F_1 show in (3.13), according to the three test images in Fig. 3.13, the results of quantitative evaluations are given in Table 3.4.

Table 3.4: Quantitative Evaluations of Edges Detected from the Test Images.

images / edges	"lena"			"pepper"			"house"			Average		
	η_p	η_r	F_1	η_p	η_r	F_1	η_p	η_r	F_1	η_p	η_r	F_1
RGB	63.77%	85.09%	72.90%	37.12%	84.44%	51.57%	55.86%	85.41%	67.54%	52.25%	84.98%	64.71%
YC_bC_r	90.92%	75.85%	82.70%	66.75%	85.45%	74.95%	71.92%	80.68%	76.05%	76.53%	80.66%	78.54%
HSV	40.60%	83.09%	54.54%	41.72%	82.27%	55.36%	18.15%	89.17%	30.16%	33.49%	84.84%	48.02%
Grey	93.47%	72.87%	81.89%	78.75%	67.37%	72.61%	73.51%	72.00%	72.75%	81.91%	70.75%	75.92%
Green	72.5%	73.34%	72.92%	58.22%	64.00%	60.97%	65.95%	69.37%	67.61%	65.56%	68.90%	67.19%
Eigen [149]	94.15%	63.88%	76.12%	67.57%	64.01%	65.74%	74.03%	65.21%	69.34%	78.58%	64.37%	70.77%
Our	91.15%	92.97%	92.05%	78.59%	84.98%	81.66%	71.91%	91.53%	80.54%	80.55%	89.83%	84.94%

For visual comparisons, all the relevant edge images are also shown from Figure 3.14 to Figure 3.34. As can be seen, our method yields quite high values of precision and recall rate in all the test images, followed by the edges extracted from YCbCr space, and the similarity here is due to the fact that channel difference information has been successfully employed in the detection of edges.

While the other algorithms generate at least one poor value in terms of precision and recall measurements. Edges extracted from both RGB and HSV spaces suffer from massive false alarms, with HSV being the worst. Edges extracted from the grey image or the pseudo-grey channel is better than those from the green component, this is because grey image contains more information from other colour components which makes it more accurate in detecting edges.

However, the results from these three single components are worse than those from YCbCr space and our method. In addition, it is worth noting that the precision rate for the "house" image and the "pepper" image is less than that of the "lena" image. The reason behind is that there are fake edges in the previous two images caused by shadows or lighting, which has inevitably led to more false alarms and lower precision values.

3.3.2.2 Robustness Evaluation

For robustness evaluation, edges extracted from colour images with attached Gaussian noise are compared.

1. the intensity values from each colour component are normalized within $[0, 1]$;

2. zero-mean Gaussian noise is added to the normalized intensity values with the variance value of σ_v changes from 0.002 to 0.008 where a larger variance value indicates a higher level of noise;

3. the normalized image with noise is converted back so that its intensity in each component is within $[0, 255]$;

For the original test images in Fig. 3.35, their corresponding noisy test samples under the variance value of 0.006 are given in Fig. 3.34. Noting that before applying the Canny operator for colour edge detection, median filtering in a 3×3 rectangle window is employed to remove noise. In addition, a post-processing step is used to remove false edges whose length is below a given threshold T_e.

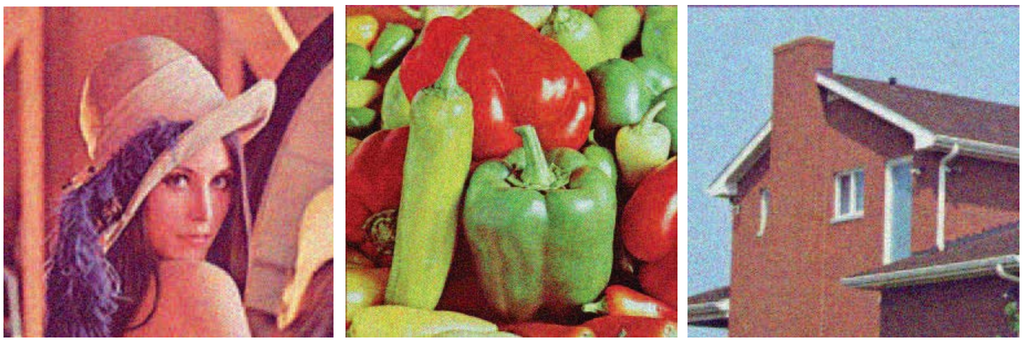


Figure 3.35: Three Noisy Test Samples with Additive Gaussian Noise Where the Variance Value is 0.006

To achieve an overall evaluation of both the precision rate and recall rate, a F_1 measurement is defined below as:

$$F_1 = \frac{2\eta_p\eta_r}{\eta_p + \eta_r} \tag{3.13}$$

In statistics, F_1 is also call F_1 score, it is a measure of a testing accuracy. In (3.13), η_p is precision rate and η_r is recall rate. η_p is the number of return correcting results divided the all return results and η_r is the number return of correcting results divided all the correcting results.

The F_1 can be interpreted as a weighted average of the precision and recall, normally, it is the best if the value of F_1 is 1 and it is worse if the value of F_1 is 0.

In the following, edges extracted from noisy images after length thresholding are compared. For simplicity, our method is only compared with the edges extracted from the pseudo-grey component [149] as well as RGB and Y-CbCr spaces, and all others are ignored for their poor performance according in Table 3.4.

Please note that only the additive Gaussian noise is tested, and more complex noise models like multiplicative noise are not considered as they are not popular in natural scenes. For each test image, three curves are plotted and shown in Fig. 3.35 to illustrate the change of F_1 value vs. the variance values of additive Gaussian noise. As can be seen, the F_1 values degrade significantly with increasing variance values of additive Gaussian noise, while our fusion scheme performs the best over all the test images. Edges extracted from RGB space are the worst which indicates that they are more sensitive to the added noise. Besides, it seems that edges extracted from the pseudo-grey component [149] are less sensitive to noise especially when the variance value is high.

Furthermore, in our implementation the threshold T_e is empirically determined as 12. As shown in Fig. 3.35, it is found that our results are insensitive to the threshold T_e which is used to remove short edges.

Comparison of F_1 values (y-axis) vs. the variance values of additive Gaussian noise (x-axis, the (a), (b), (c) three images and three plots respectively correspond to the test images "Lena", "Pepper" and "House") and in (d), various threshold values (x-axis) (right). The results labelled with "eigen" is extracted from the pseudo-grey image in [149].

In addition, visual comparisons over the edges extracted from noisy samples are also shown from Fig 3.37 to Fig 3.60, in which two groups of noisy samples are used and the variance values of the Gaussian noise are 0.002 and 0.006, respectively.

From Fig 3.37 - Fig 3.48 shown visual comparisons after put Gaussian noise on the original images, and the variance value of Gaussian noise is 000.2

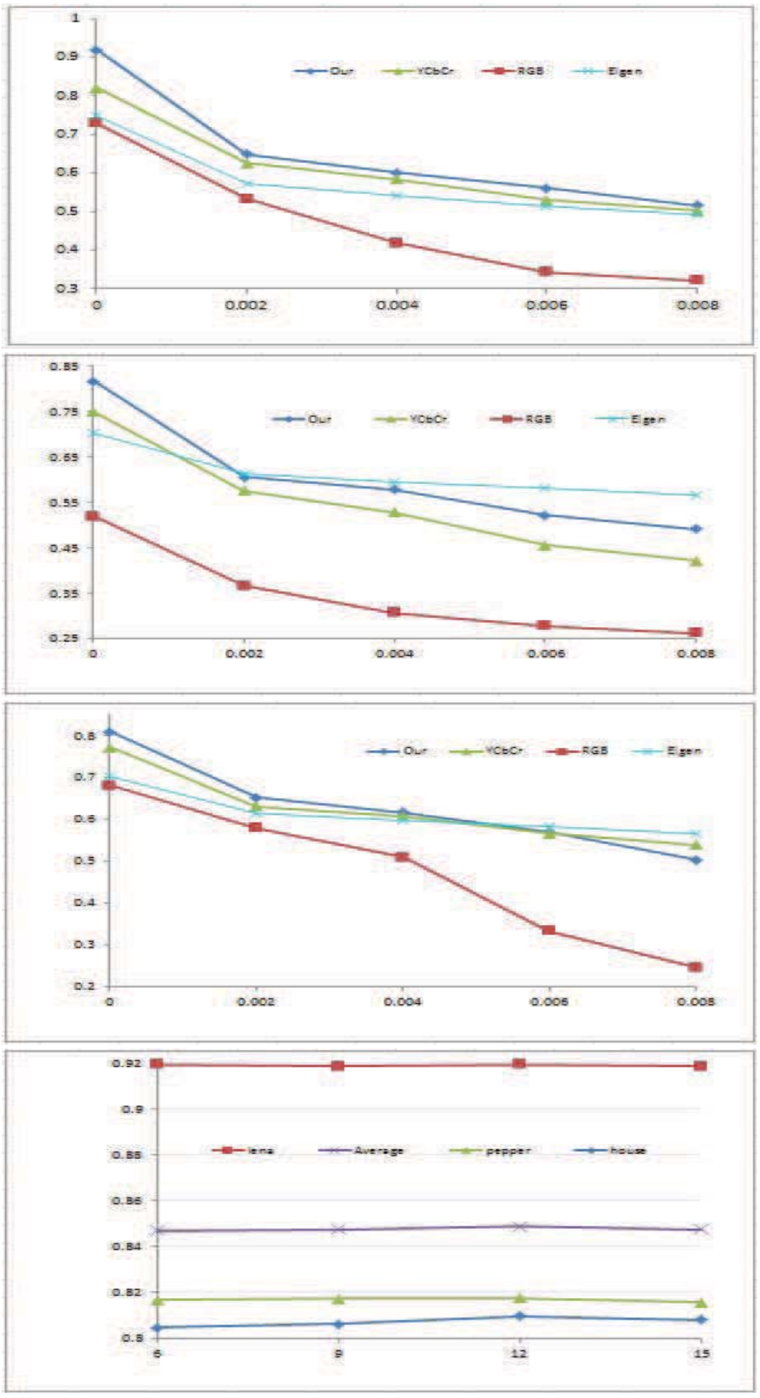
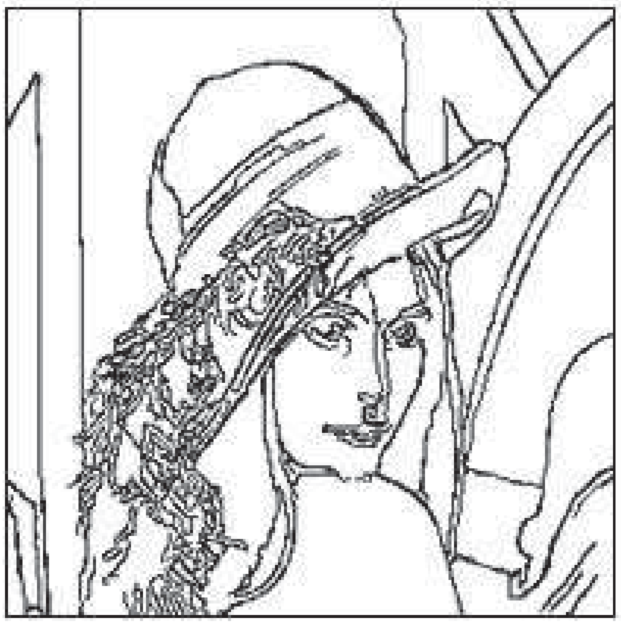


Figure 3.36: Comparison of F_1 Values

Compare Ground Truth Image and RGB, there are massive false alarms in RGB, even could not identify the face.


Ground Truth Image

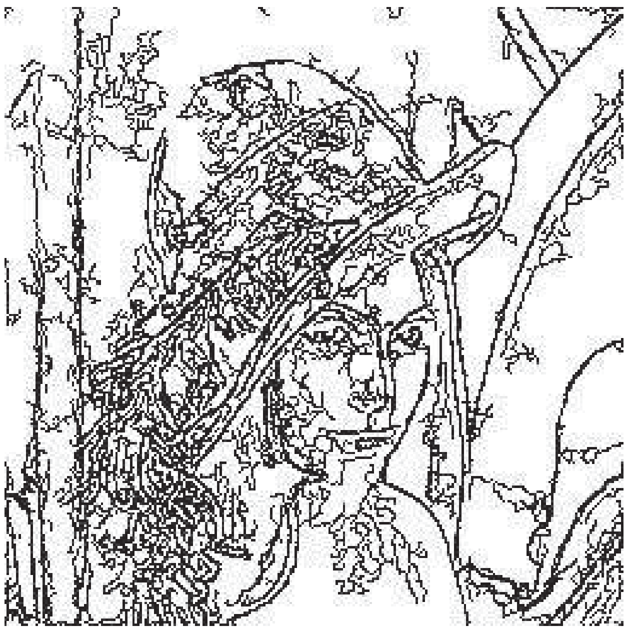

RGB

Figure 3.37: Compare Ground Truth and RGB, 'Lena'

Compare Ground Truth Image and YCbCr, the edges extracted from YCbCr are better than the edges extracted from RGB (Fig 3.36), but there are missing edges, like the brim of the hat, top of the hat, etc.

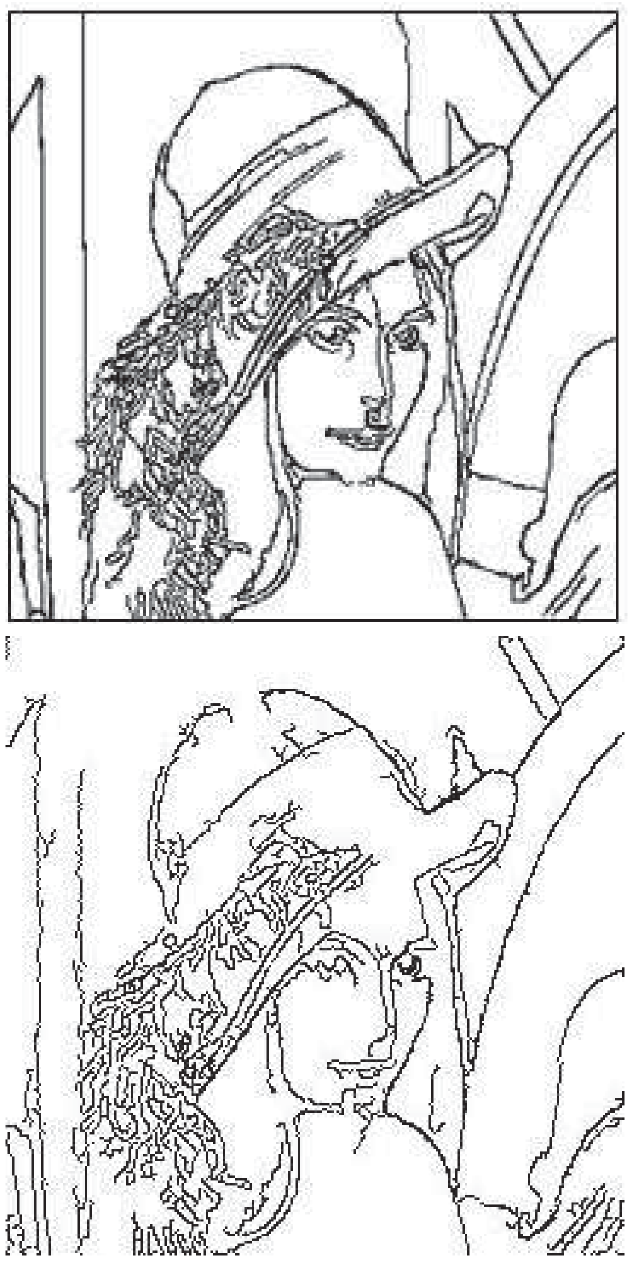
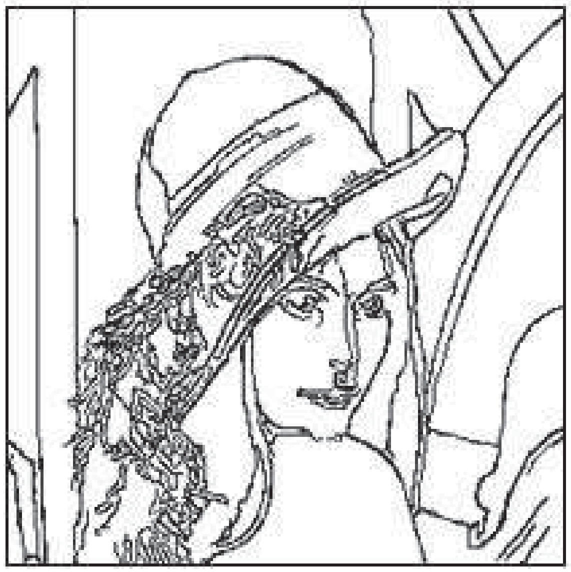
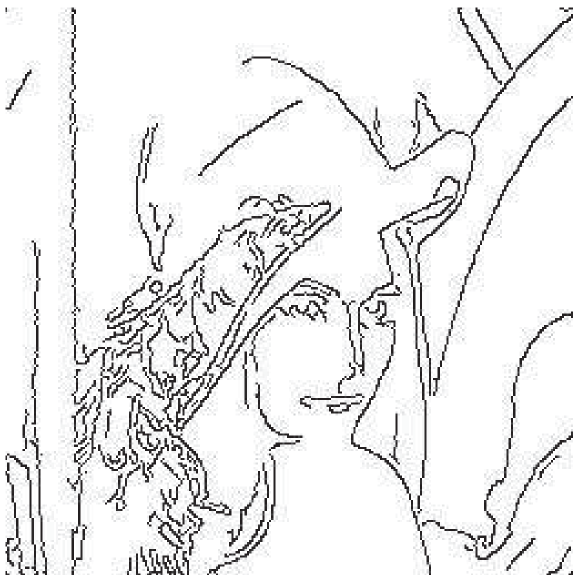


Figure 3.38: Compare Ground Truth and YCbCr, 'Lena'

Compare Ground Truth Image and Eigen, the false alarms nearly could not found, but the missing edges still in some places, like the brim of the hat, top of the hat and girl's chin, if compare with Fig 3.18, Eigen is clearly less sensitive to noise.



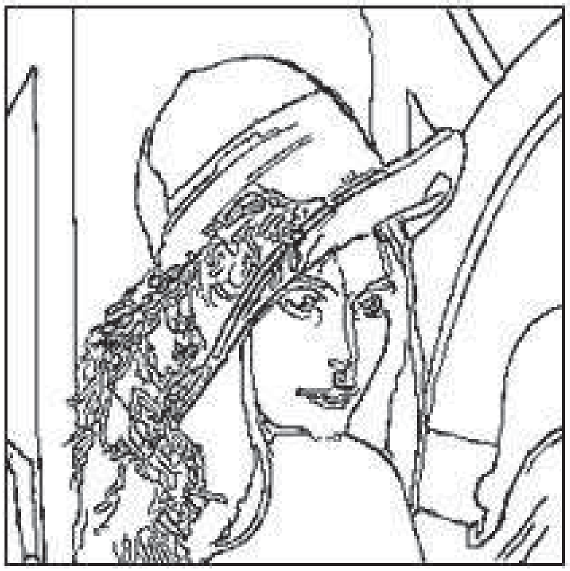
Ground Truth Image



Eigen

Figure 3.39: Compare Ground Truth and Eigen, 'Lena'

Compare Ground Truth Image and Our Approach, there are missing edges happened in the brim of the hat, on the top of hat, the girl's chin, hair, etc. For false alarms, right side of the hat, a little bit under the mouth. But compare with RGB and YCbCr, our approach's performance is better.


Ground Truth Image

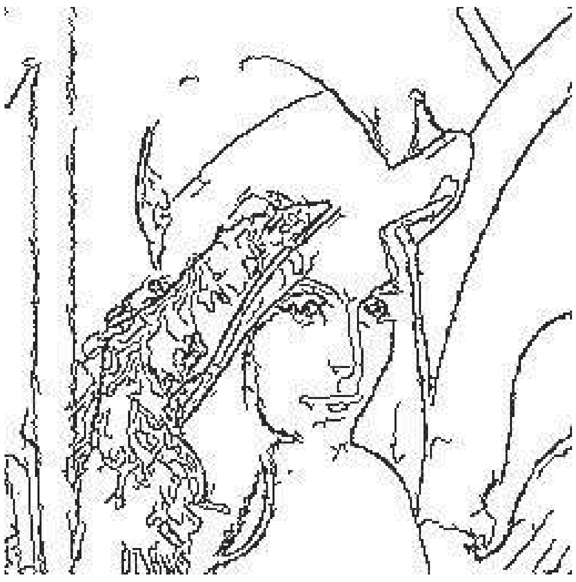
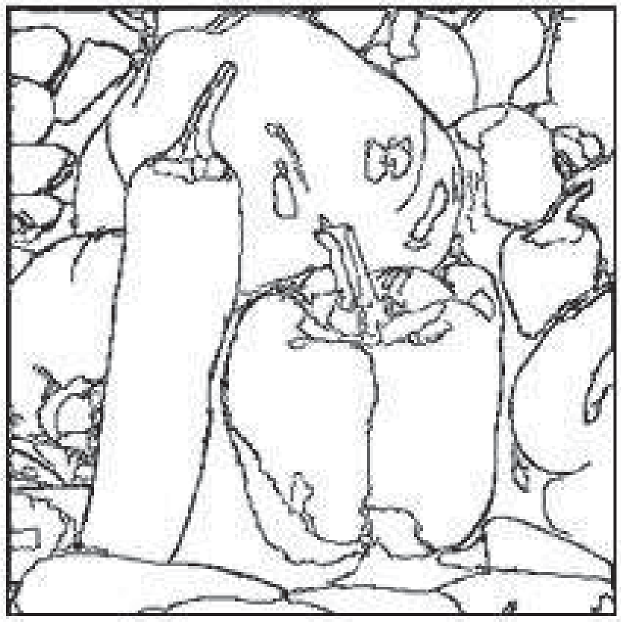

Our Approach

Figure 3.40: Compare Ground Truth and Our Approach, 'Lena'

Compare Ground Truth Image and RGB, massive false alarms appeared, RGB is sensitive to noise.


Ground Truth Image

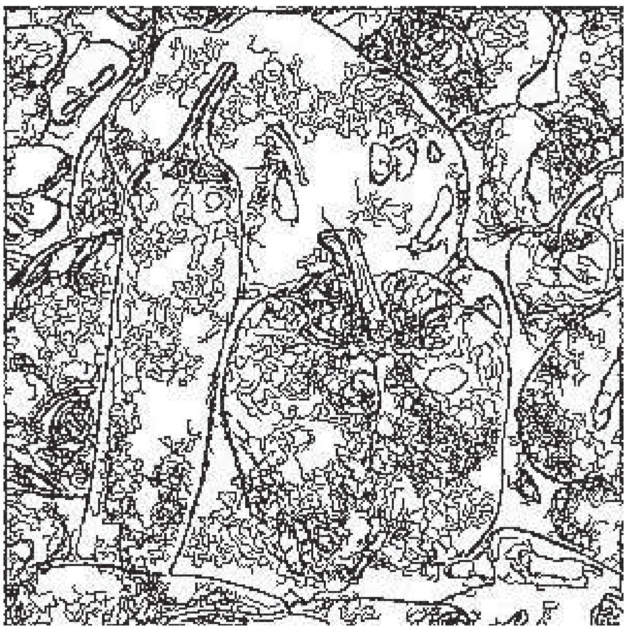
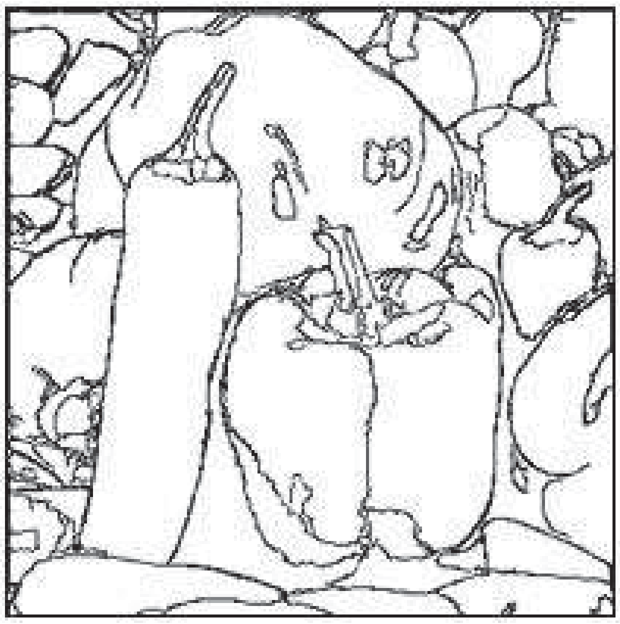

RGB

Figure 3.41: Compare Ground Truth and RGB, 'Pepper'

The edges extracted from YCbCr are better than RGB, but still lots of false alarms appeared and missing so many edges of the boundary of the peppers.


Ground Truth Image

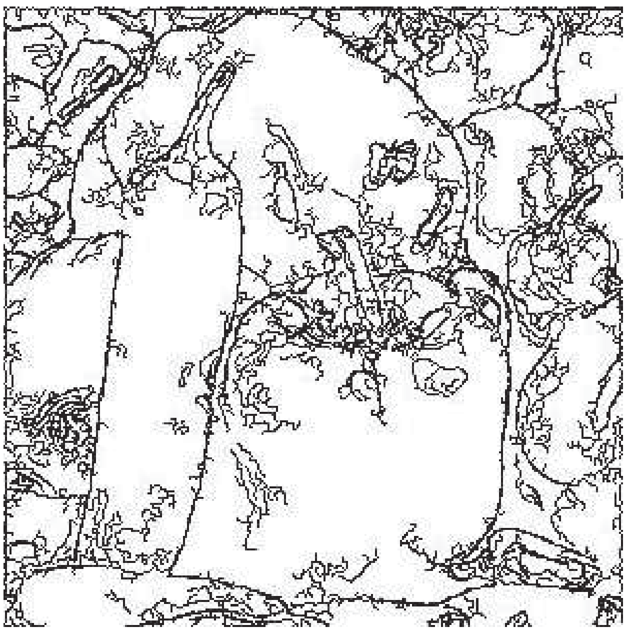

YCbCr

Figure 3.42: Compare Ground Truth and YCbCr, 'Pepper'

Compare with RGB and YCbCr, the false alarms in Eigen have reduced, but still in the suface of the pepper. The other side, the missing edges appeared in the boundary of the pepper.

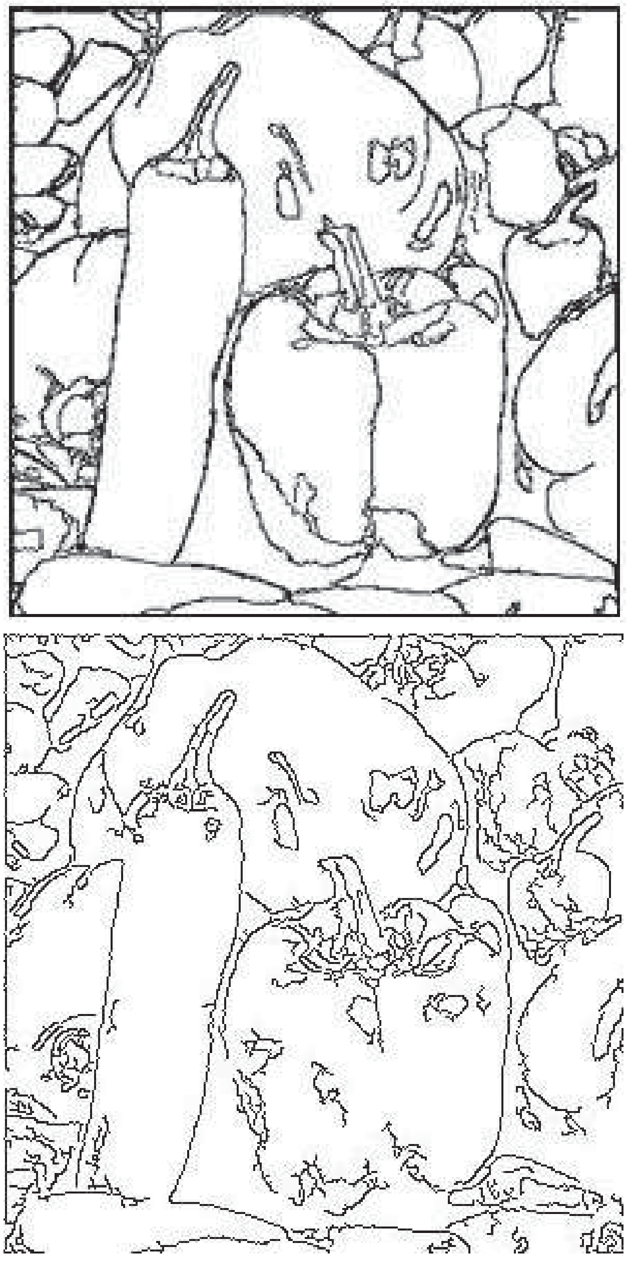
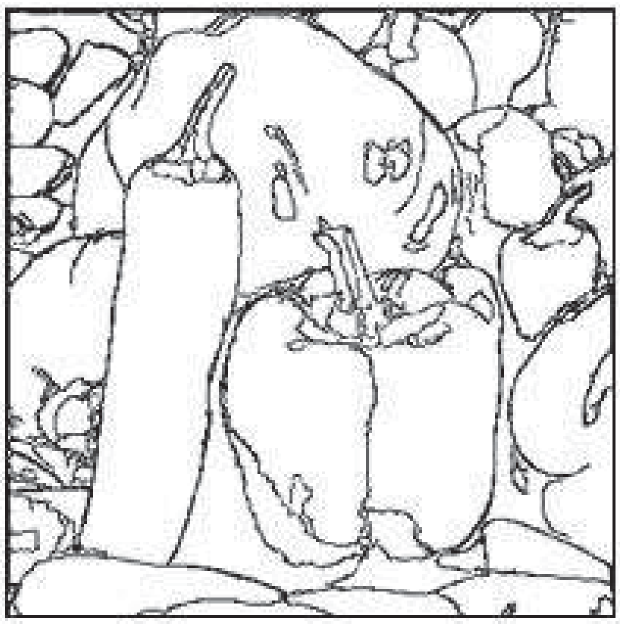


Figure 3.43: Compare Ground Truth and Eigen, 'Pepper'

Compare Ground Truth Image and Our Approach, our scheme can find almost edges, but missing edges in the bottom of the bigger pepper and still have false alarms on the surface.


Ground Truth Image

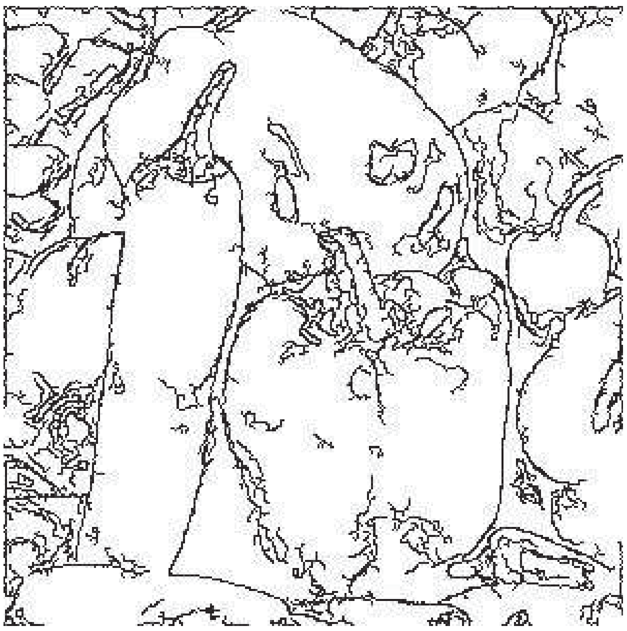
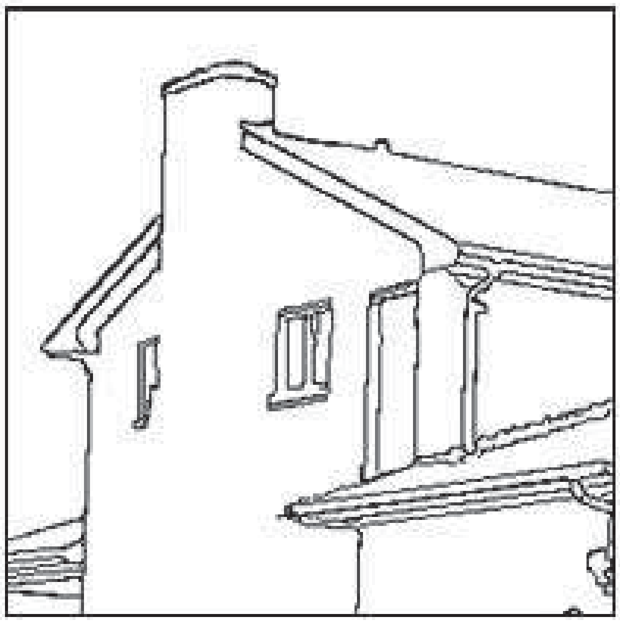
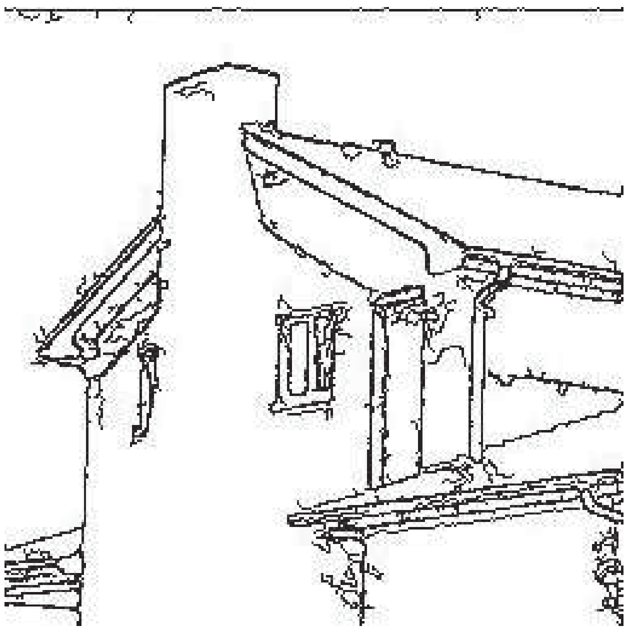

Our Approach

Figure 3.44: Compare Ground Truth and Our Approach, 'Pepper'

Compare Ground Truth Image and RGB, false alarms appeared on the roof, on the wall, and in the area of window.



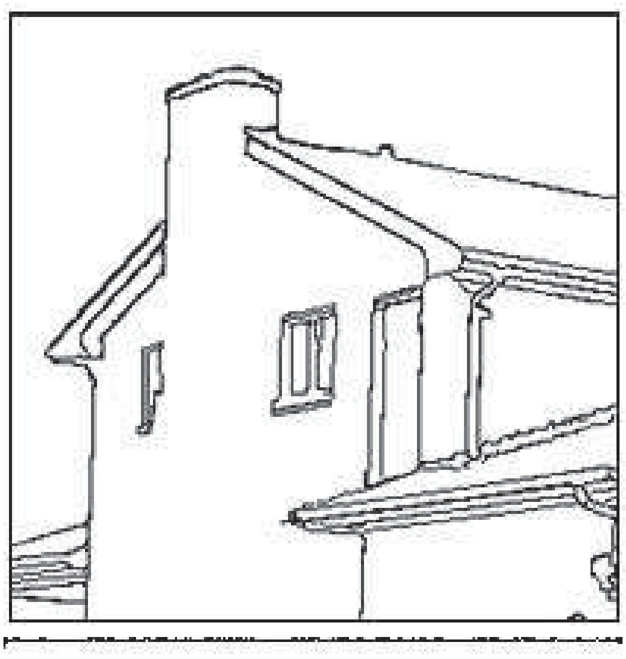
Ground Truth Image



RGB

Figure 3.45: Compare Ground Truth and RGB, 'House'

Compare the YCbCr and RGB, false alarms have been reduced, but missing edges appeared in the corner of the window, the wall, the chimney and the roof.


Ground Truth Image

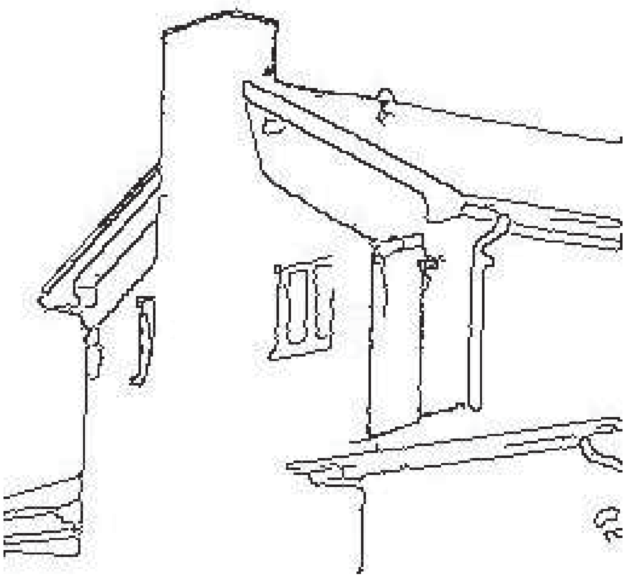
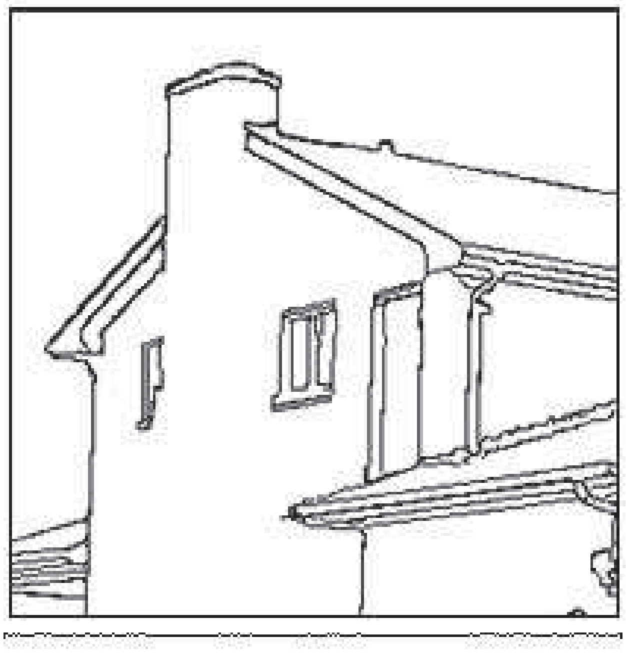

YCbCr

Figure 3.46: Compare Ground Truth and YCbCr, 'House'

Compare Ground Truth Image and Eigen, false alarms appeared under the eaves; missing edges appeared in the corner of the window


Ground Truth Image

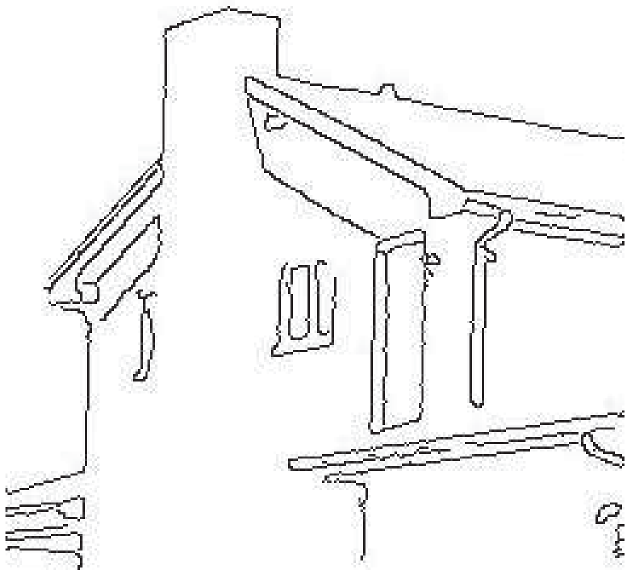
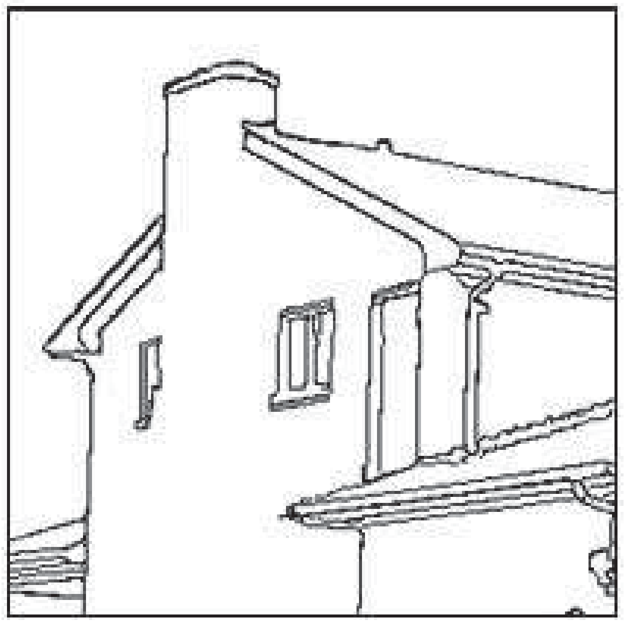

Eigen

Figure 3.47: Compare Ground Truth and Eigen, 'House'

Compare Ground Truth Image and Our Approach, false alarms appeared under the eaves, left side of the wall, etc.; missing edges appeared in the lower eaves and the chimney, etc.


Ground Truth Image

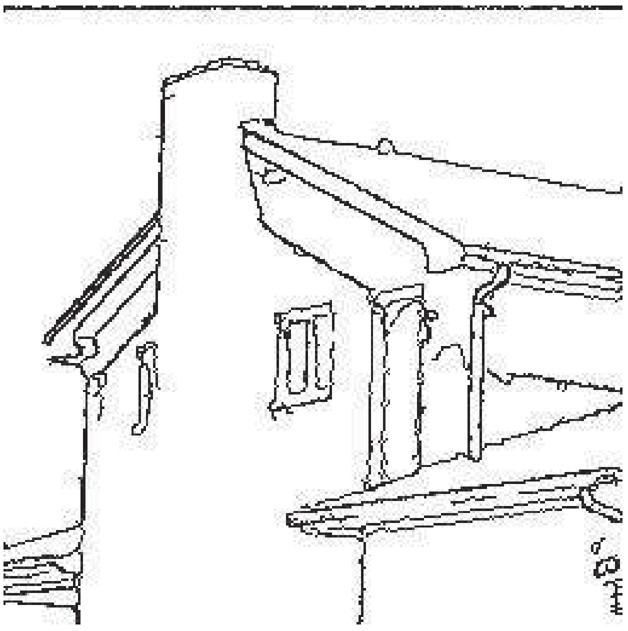

Our Approach

Figure 3.48: Compare Ground Truth and Our Approach, 'House'

From Fig 3.49 - Fig 3.60 shown visual comparisons after put Gaussian noise on the original images, and the variance value of Gaussian noise is 000.6. Compare Fig 3.49 and Fig 3.37, RGB is so sensitive to noise.

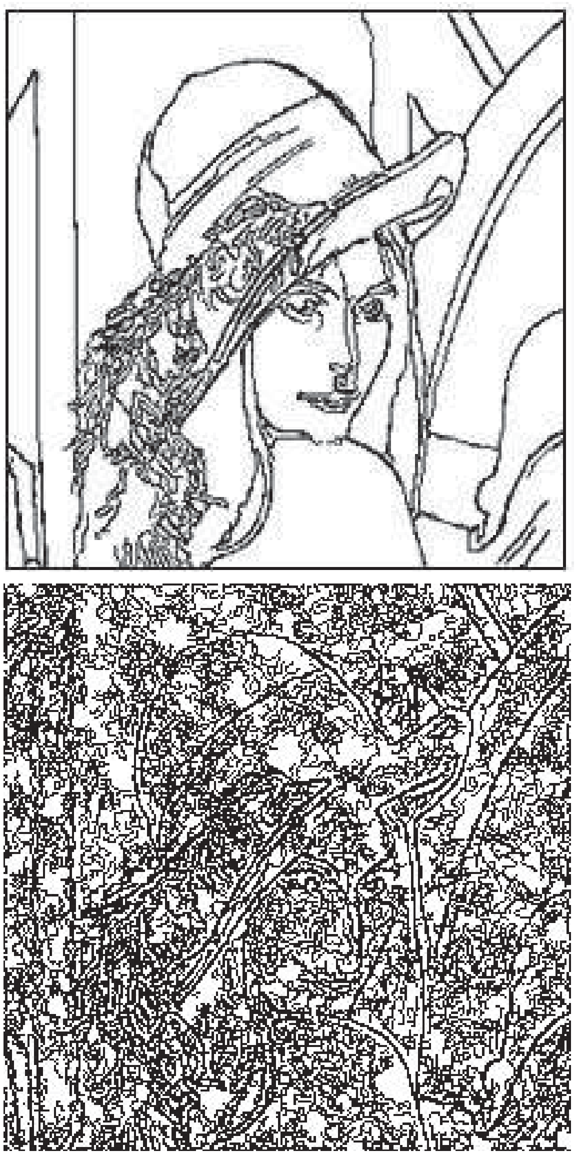
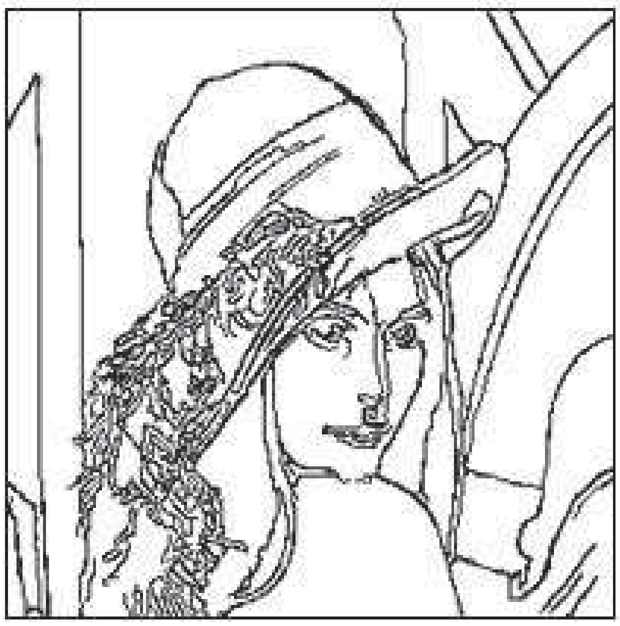


Figure 3.49: Compare Ground Truth Image and RGB, 'Lena'

Compare Ground Truth Image and YCbCr, false alarms appeared in the face, the boundary of the hat, the boundary of the hair, etc. Missing edges happened in the boundary of the hat, the brim of the hat and the chin, etc.


Ground Truth Image

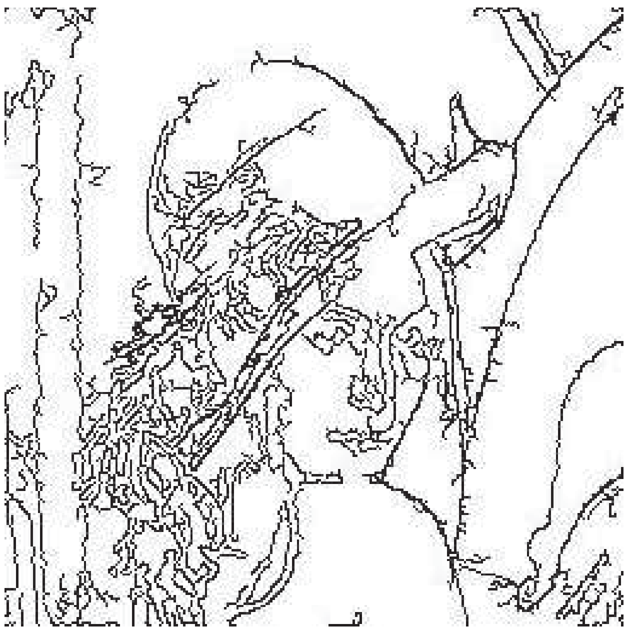

YCbCr

Figure 3.50: Compare Ground Truth Image and YCbCr, 'Lena'

Compare Ground Truth Image and Eigen, false alarms have been reduced, but missing edges appeared in the boundary of hat, the brim of the hat, the chin, and the hair, etc.

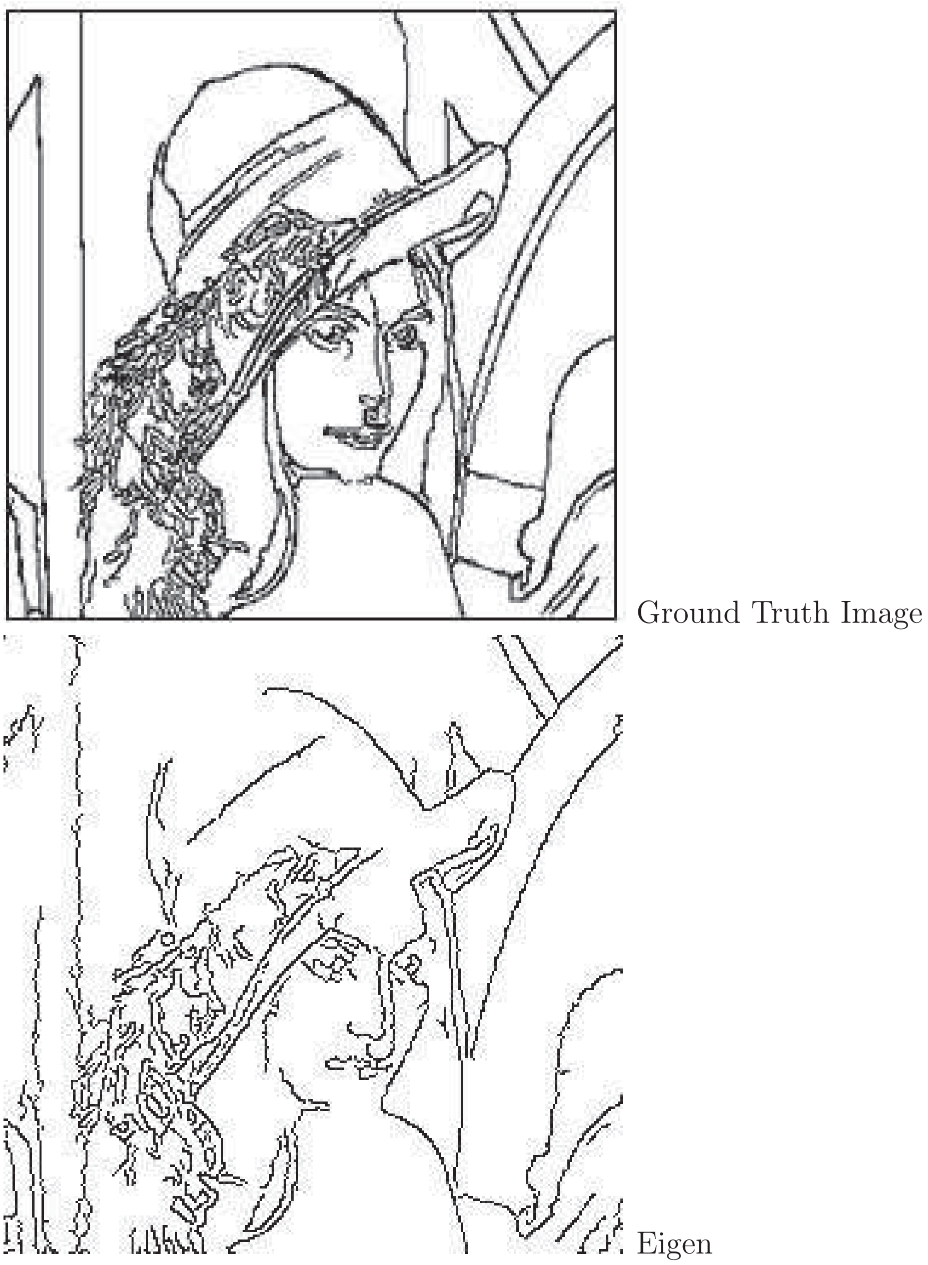


Figure 3.51: Compare Ground Truth Image and Eigen, 'Lena'

Compare Ground Truth Image and Our approach, still have false alarms in the face, and missing edges in the boundary of the hat, the brim of the hat. But the performance is better than RGB and YCbCr.

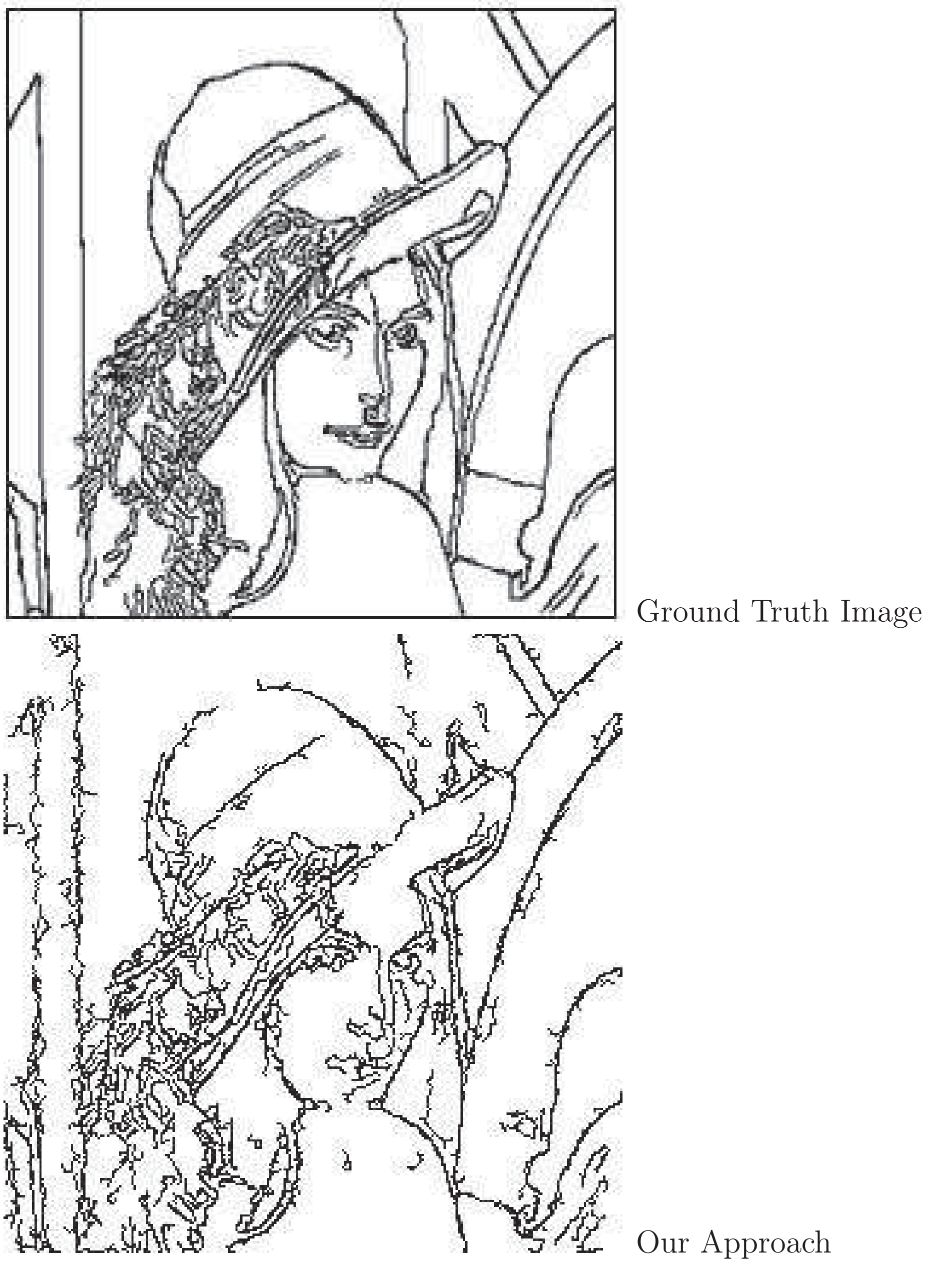
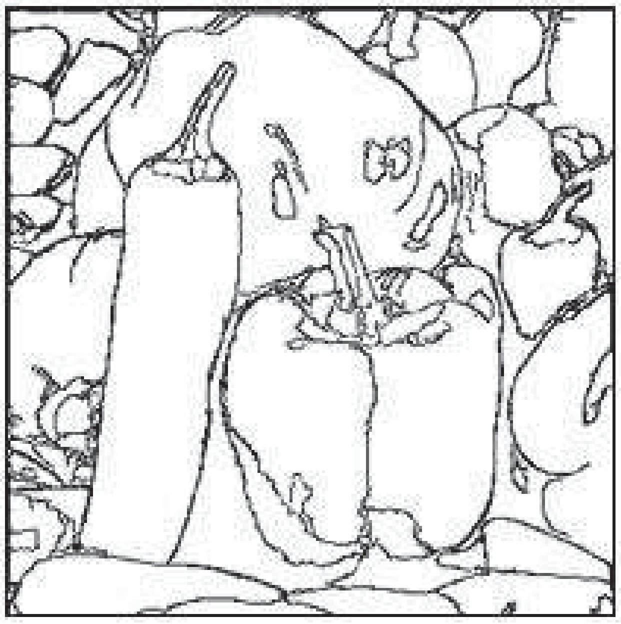


Figure 3.52: Compare Ground Truth Image and Our Approach, 'Lena'

In RGB, could not find out any edges, RGB sensitve to noise.


Ground Truth Image

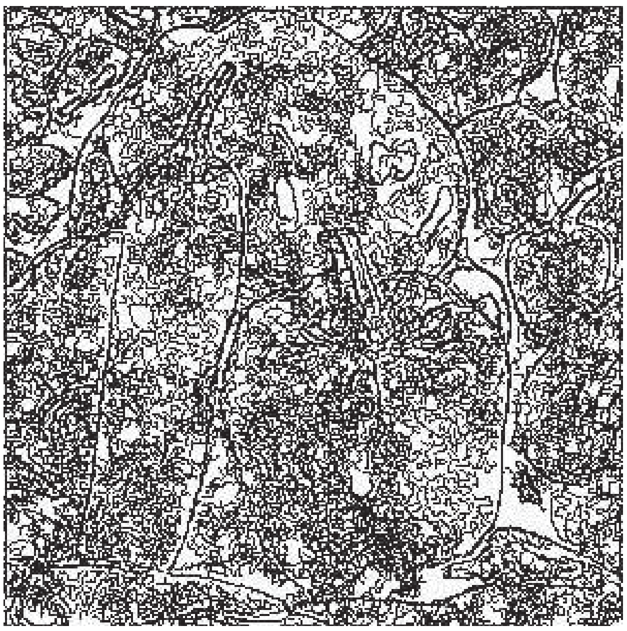
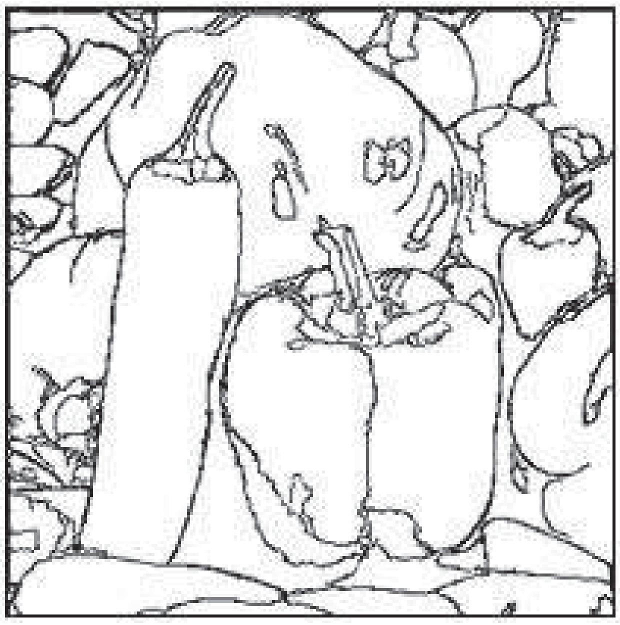

RGB

Figure 3.53: Compare Ground Truth Image and RGB, 'Pepper'

Compare Ground Truth Image and YCbCr, still a little bit false alarms in the surface of the bigger pepper, and missing edges appeared the bottom of the bigger pepper.


Ground Truth Image

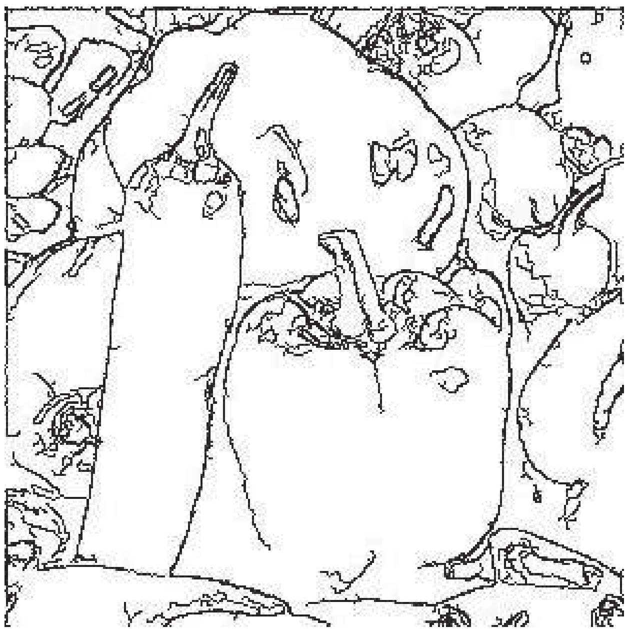
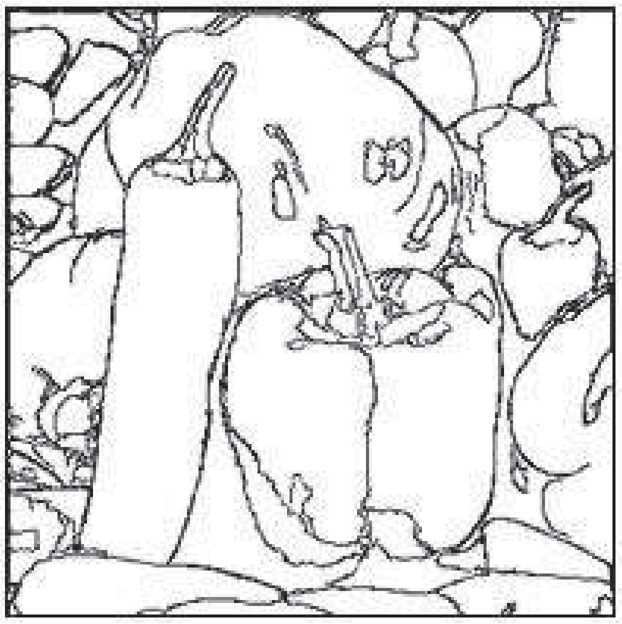
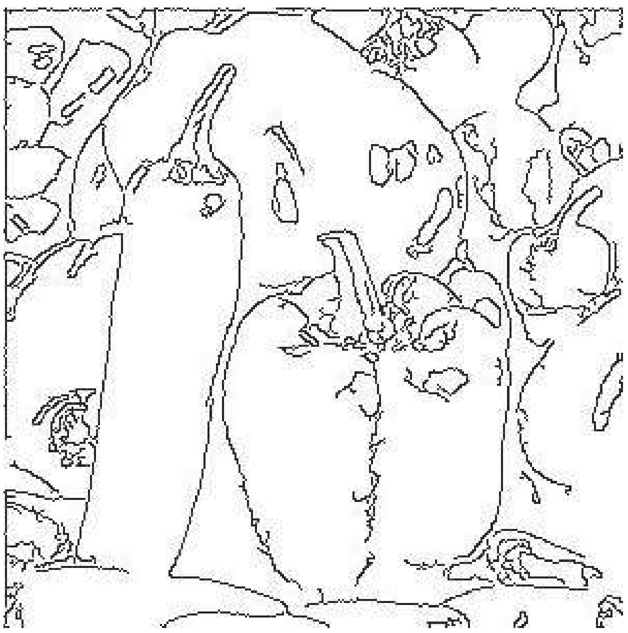

YCbCr

Figure 3.54: Compare Ground Truth Image and YCbCr, 'Pepper'

Compare Ground Truth Image and Eigen, false alarms appeared on the surface of the bigger pepper and miss detected edges on the bottom of the bigger pepper



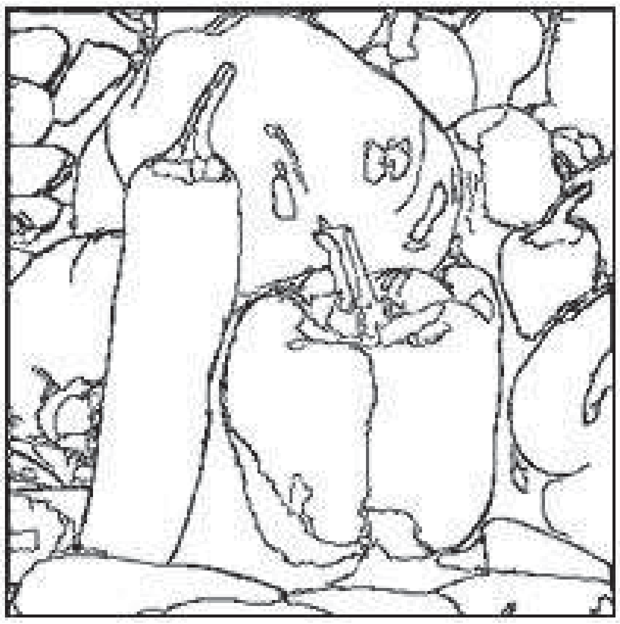
Ground Truth Image



Eigen

Figure 3.55: Compare Ground Truth Image and Eigen, 'Pepper'

Compare Ground Truth Image and Our Approach, the edge of the long pepper is quite good, but false alarms on the surface of the bigger pepper, and missing the edge of the bottom of the bigger pepper.


Ground Truth Image

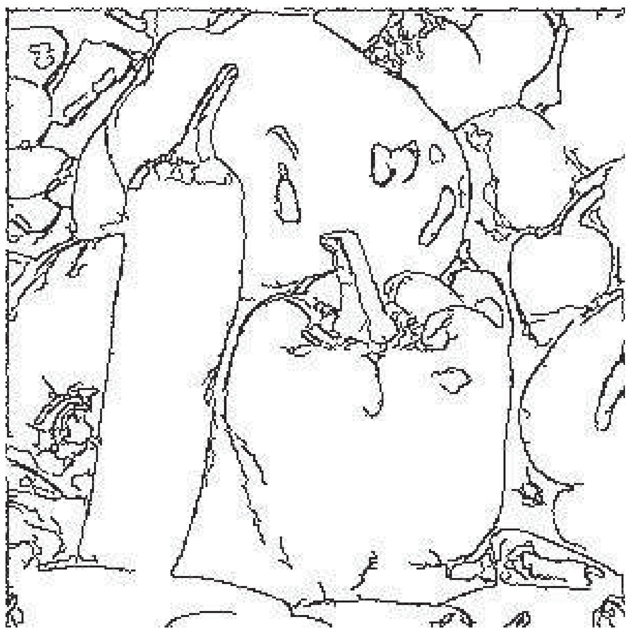
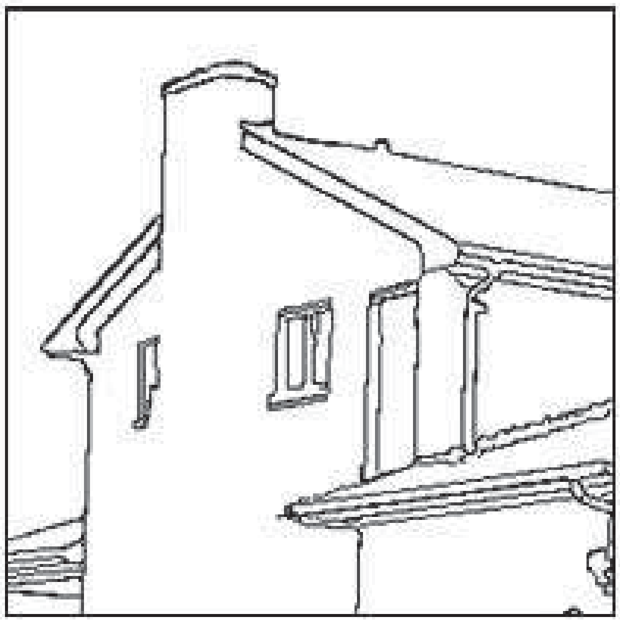
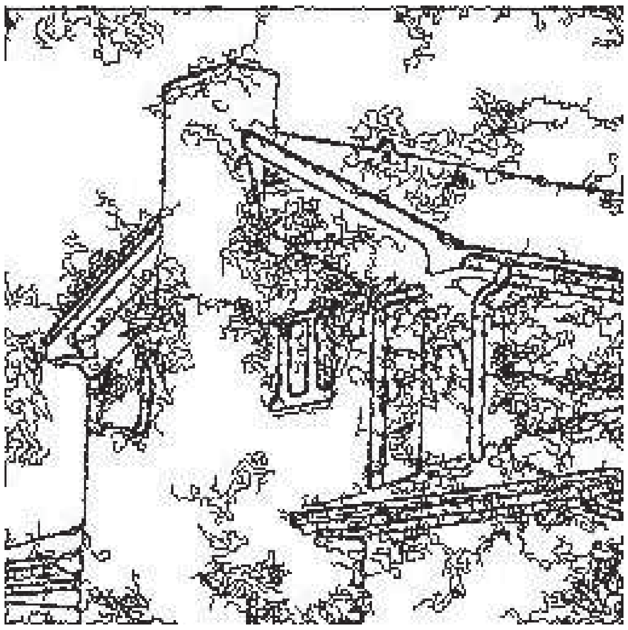

Our Approach

Figure 3.56: Compare Ground Truth Image and Our Approach, 'Pepper'

Compare Fig 3.57 and Fig 3.45, just can say RGB sensitive to noise.



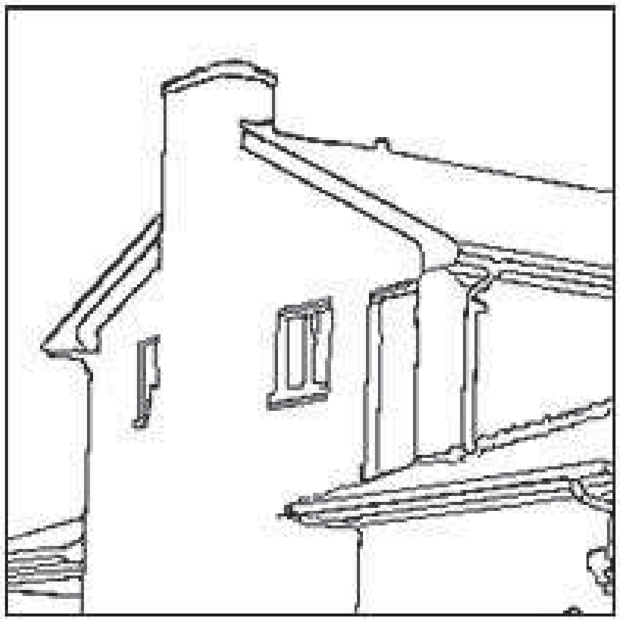
Ground Truth Image



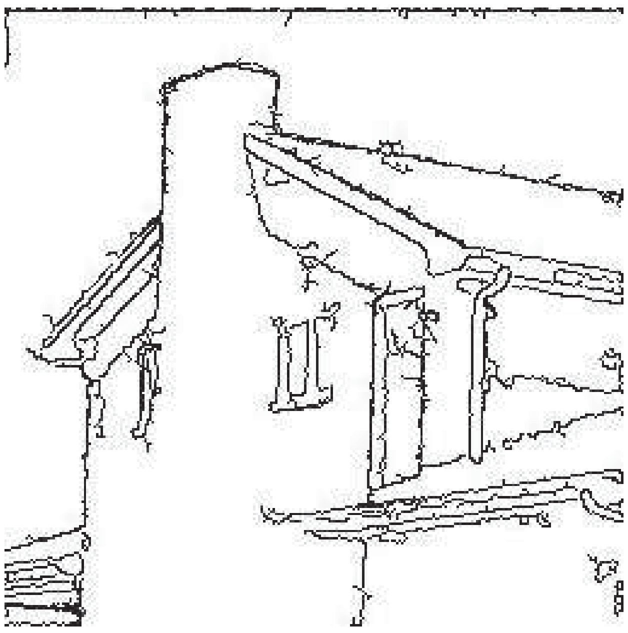
RGB

Figure 3.57: Compare Ground Truth Image and RGB, 'House'

Compare Ground Truth Image and YCbCr, false alarms appeared in the corner of the left side of the wall, the eaves. Missing edges happened the boundary of the window, the top of the chimney.



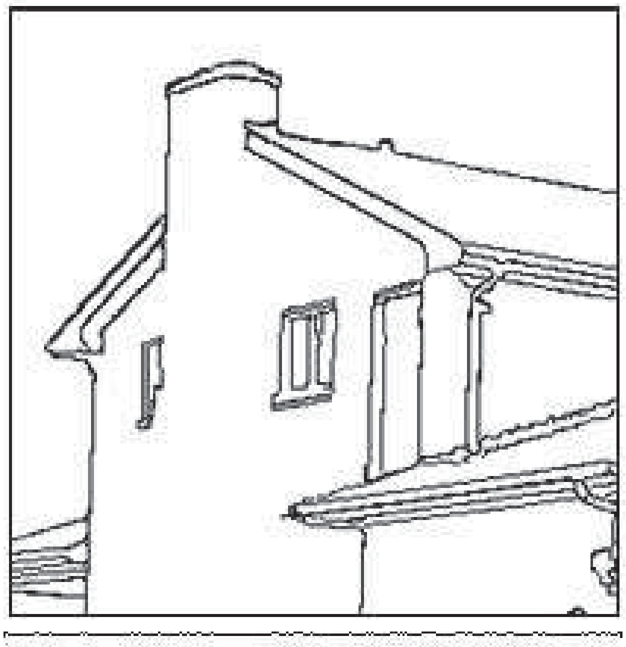
Ground Truth Image



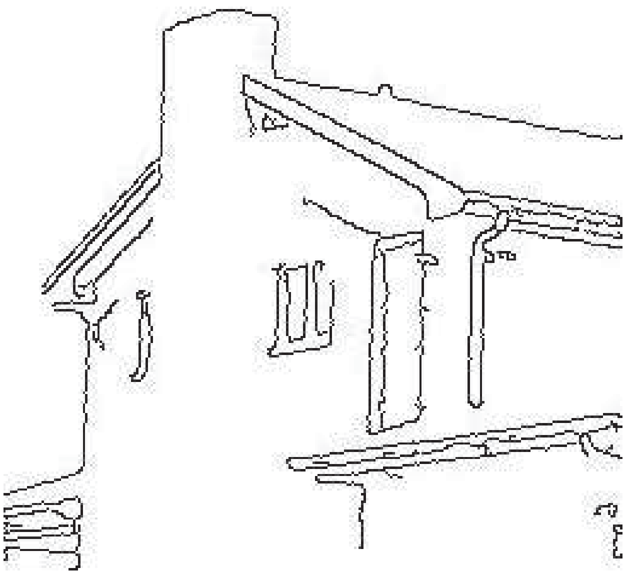
YCbCr

Figure 3.58: Compare Ground Truth Image and YCbCr, 'House'

In Eigen, false alarms almost disappeared except in the left corner of the wall and under the eaves, but missing edges happened in the top corner of the window, the chimney.



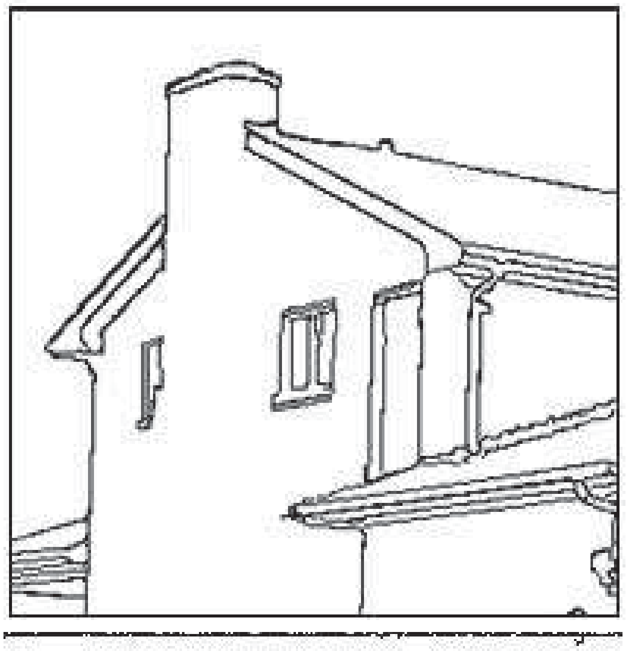
Ground Truth Image



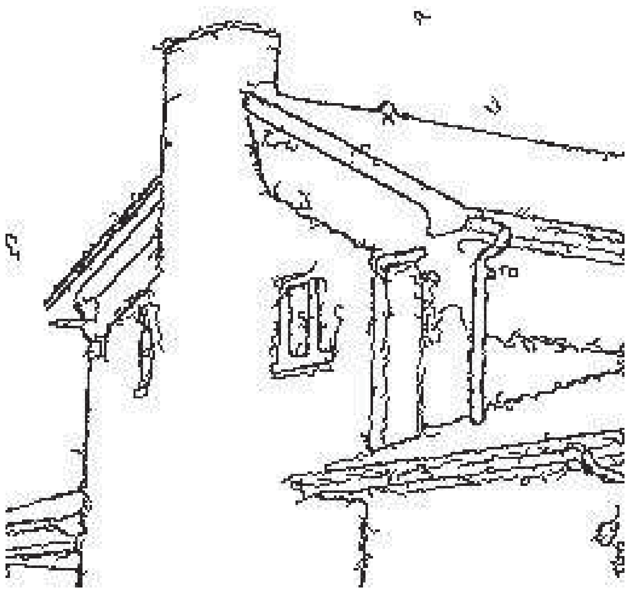
Eigen

Figure 3.59: Compare Ground Truth Image and Eigen, 'House'

Compare Ground Truth Image and Our Approach, our scheme produced a few false alarms, but it is better than RGB and YCbCr. Missing edges happened in the corner of the window, the chimney.



Ground Truth Image



Our Approach

Figure 3.60: Compare Ground Truth Image and Our Approach, 'House'

Again, RGB edges are extremely sensitive to noise. Although both our results and YCbCr edges exhibit some robustness to the noise, the false alarms caused in the corresponding images are different. In general, false alarms in YCbCr edges are adjacent to real edges, but in our results these false alarms are separated.

Therefore, it is possible to further improve the accuracy of our algorithm by introducing more powerful post-processing to reduce these separated fake edges. In addition, edges extracted from pseudo-grey component has fewer false alarms though more missing edges.

The computing complexity of the proposed algorithm contains three main parts including

1. extraction of G-image and R-image;
2. detection of edges using Canny detector on the two single-component images;
3. post-processing;

In general, the first step takes most of the running time, i.e. more than 60% of the total time of our method. Since the complexity of edge detection algorithms rely on image contents, especially for the Canny detector where tracing of edges is employed, the complexity of our method is compared with others in a relative way as follows. For one test image, different edge detection methods are applied for 100 times on the same machine and the executive times are taken as a good indicator to evaluate the corresponding complexity of the approaches. Executive times obtained in edge extraction from the three test images in Fig. 3.12, using RGB and YCbCr colour spaces, pseudo-grey component, and our scheme, are listed in Table 3.5 for comparisons.

As can be seen, the executive time for the "pepper" image is the longest as it contains the most edges. The overall complexity of YCbCr edges are the minimum, followed by RGB edges and eigen edges, and our scheme is the most complex one. However, our method is only 23% more complex than YCbCr edges, and the additional cost is quite limited for the good performance achieved in our tests.

Table 3.5: Comparison of Complexity by Executive Time (in seconds) via Running 100 Times of the Test in Extracting Edges from the Images in Fig. 3.13

images / methods	"lena"	"pepper"	"house"	Average	
				Time	Ratio
RGB edges	3.203	4.719	3.125	3.682	107%
YC_bC_r edges	3.188	4.188	2.985	3.454	100%
Eigen edges [149]	3.265	4.532	3.150	3.649	106%
Our edges	3.797	5.219	3.688	4.235	123%

3.4 Middle-Level Image Processing Approach — Face Detection

3.4.1 Skin Segmentation

In this part, how to extract human's face from an image, based on the theory of colour space transform, will be expressed, the skin detection was also discussed in 3.2.

Automatic detection of skin and face plays very important roles in many vision applications, such as face and gesture recognition in intelligent human-machine interaction and visual surveillance [150-152], naked adult image detection [153,154], video phone or sign language recognition [155,156] as well as content-based multimedia retrieval [157,158].

Since skin detection is a classification problem defined on colour similarity, supervised clustering is applied to achieve the exact rules for effective skin colour clustering and pixel classification. Through manually specifying representative skin and non-skin pixels, the linear relationships can be learned between different components in the new colour spaces. Finally, several main boundary conditions also can be obtained for skin pixels classification in different colour spaces.

Firstly, skin pixels are modelled by using the histogram-based approach, in which the probability or likelihood that each colour represents skin is estimated by checking its occurrence ratio in the training data. In (3.14),

V_{skin} indicates volumes or total occurrences of all skin colours in manual ground truth of training data.

$$p(\text{colour/skin}) = sum(\text{colour/skin})/V_{skin} \tag{3.14}$$

Then, boundary conditions in the skin model are extracted to allow more than 98% of skin pixels covered. Using the boundary conditions, test images are segmented into skin and non-skin regions accordingly. For different colour spaces, these boundary conditions are found as follows.

As for YUV space, the boundary conditions are found as:

$$\begin{cases} 148 \leq & V & \leq 185 \\ 189 \leq & U + 0.6V & \leq 215 \end{cases} \tag{3.15}$$

Considering the illumination intensity variation, the boundary conditions can be shown as:

$$\begin{cases} Y > 85 \qquad or \\ Y < 85, \quad U > 104, \quad Y + U - V > 2 \end{cases} \tag{3.16}$$

$$\begin{cases} S \leq 21 & , \ V \geq 2.5S \\ 158 \leq H + V \leq 400 & , \ H + V > 13S \\ H > 0.2V & , \ H > 4S \end{cases} \tag{3.17}$$

Fig 3.61 gives skin results from YUV and HSV spaces, where three people have been found in the background, which can be clearly found in the histogram-equalized image. However, the skin regions can be successfully found in HSV space from the original image while they cannot be found in YUV space.

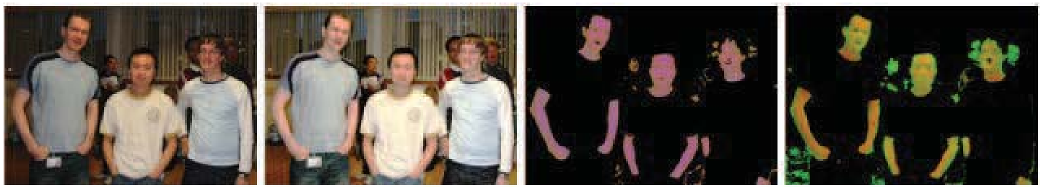

(a) Original image (b) Histogram (c) YUV Skin Results (d) HSV Skin Results

Figure 3.61: Comparison of Skin Regions Detected from YUV and HSV Colour Spaces

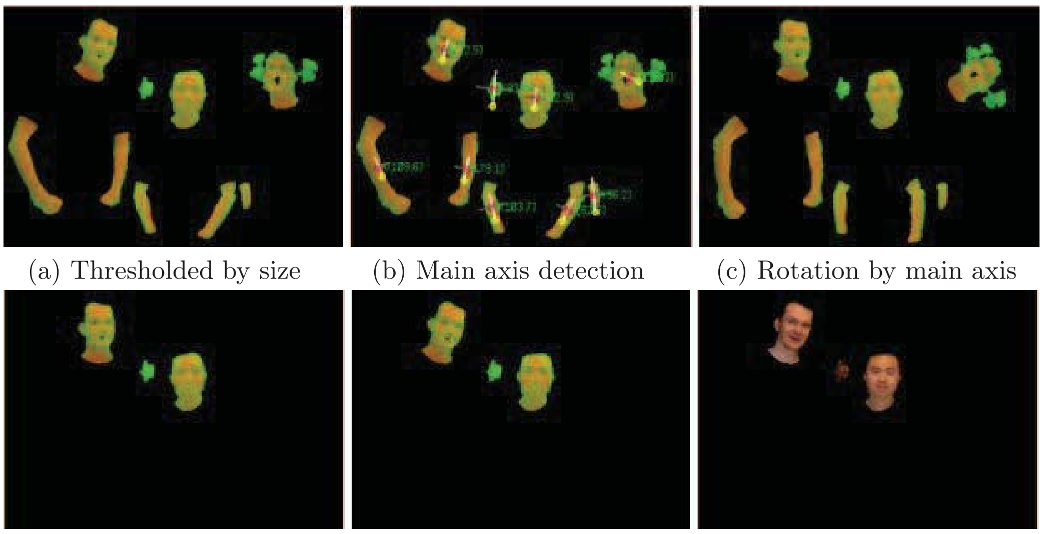
3.4.2 Knowledge-based Face Modelling and Detection

After skin detection, it needs to locate faces in candidate skin regions. Again faces of nearly frontal view have been detected, but there are no constraints on their leaning angles. Knowledge about the size, size ratio, locations of ears and mouth is used.

Firstly, the detected skin regions are labelled to obtain the outer boundary rectangle and pixel number of every region. Then, small regions that have pixels less than a given threshold, i.e. 300, will be removed. Finally, the skin regions are filtered by a SR parameter (Width/Height ratio) defined as,

$$SR = \begin{cases} width/height & \text{if} \quad width \leq height \\ height/width & \text{if} \quad width > height \end{cases} \tag{3.18}$$

In (3.18), the width and height of the regions are determined by the rectangle bounding box of each region, and the valid for candidate face regions should lie in [0.55,0.90]. To acquire more reasonable width and height of the regions, the main axis is extracted by moment calculation of each region. Then, the skin regions are rotated by the main axis angle to make the final main axis in vertical direction. Fig 3.62(a) is the filtering result by threshold-



(a) Thresholded by size (b) Main axis detection (c) Rotation by main axis

(d) Thresholded by W/H ratio (e) Face candidates (f) Face in original image

Figure 3.62: Face Filtering from Skin Regions in Fig 3.61(d) by Thresholding of Size and W/H Ratio

ing using the size of 300. In Fig 3.62(b), the main axis of each labelled region is marked with white line, and the angle and region number are also given. From Fig 3.62(d) to 3.62(e), the candidate face regions is given in rotated skin results and the skin regions before rotation in HSV space. Besides, Fig 3.62(f) gives the face candidates in RGB space for comparison. Three basic

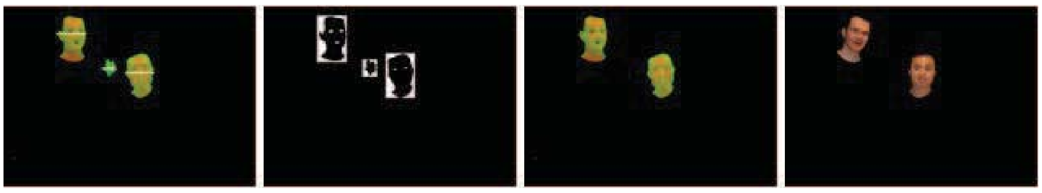

(a) Ears location (b) Feature holes (c) detected face (d) mapped back

Figure 3.63: Ears Location with White Line (a) and Feature Holes Detection (b) for Face Detection

rules are used in further face modelling and detection: First, there are one or two ears near the half height of every candidate face region which makes the width of the skin regions bigger than other lines. Second, there are one or two eyes over the height of the ear line which forms one or two dark holes. Third, an open mouth will form a dark hole near the middle of eyes below the ear line. Following is our algorithm for face detection and the results are given in Fig 3.63.

1. Detect the ear line by extracting of local maximum width near the centre of the candidate face regions, see Fig 3.62(a);

2. Detect the holes by the illumination intensity difference. Holes contain those pixels that have lower intensity than the average intensity of the candidate regions, say, less than 80% of the average intensity, see Fig 3.62(b);

3. Judge the relative positions of the holes and ear line and determine the candidate region is a valid face or not;

3.4.3 Implementation and Experimental Results

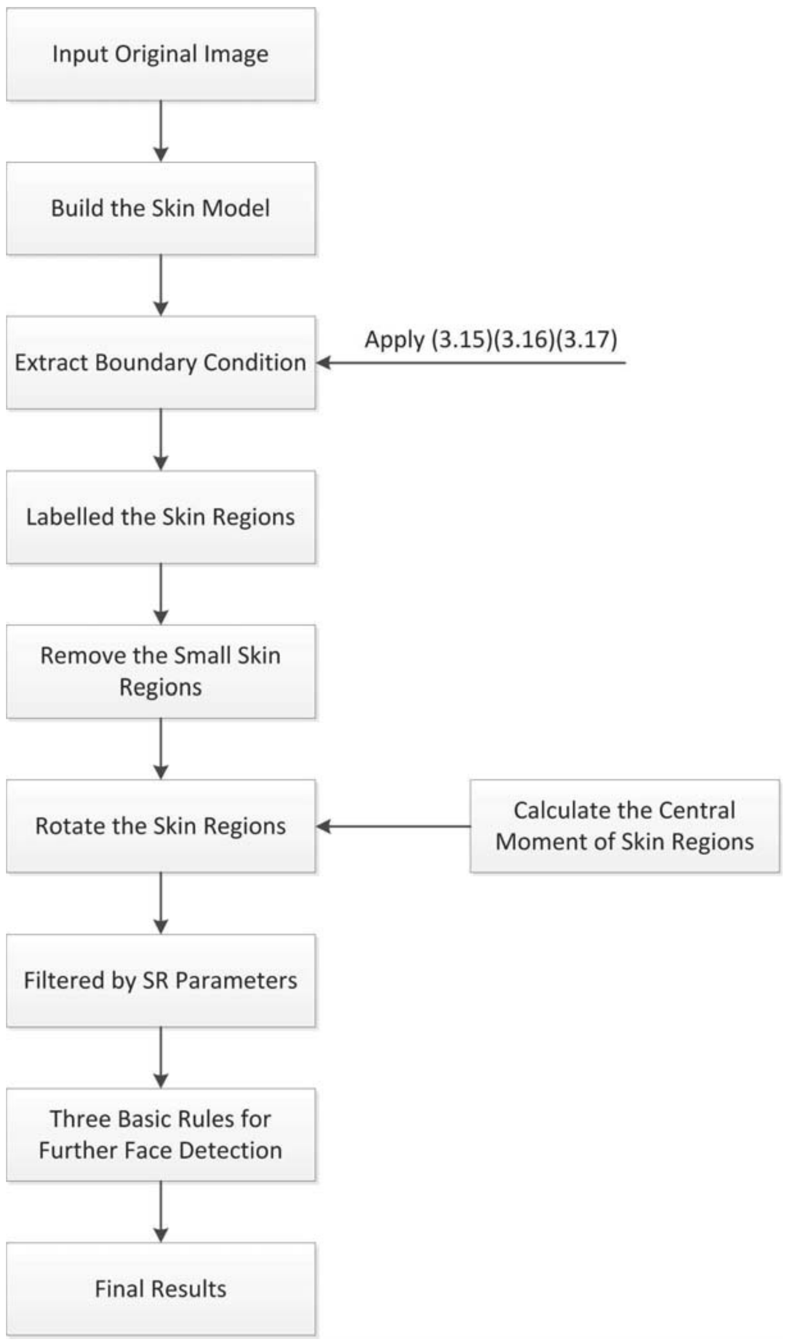

Input Original Image
Build the Skin Model
Extract Boundary Condition
Apply (3.15)(3.16)(3.17)
Labelled the Skin Regions
Remove the Small Skin Regions
Rotate the Skin Regions
Calculate the Central Moment of Skin Regions
Filtered by SR Parameters
Three Basic Rules for Further Face Detection
Final Results

Figure 3.64: Flow Chart for the Face Detection

Face detection always affected by internal factors like facial expression, beard and glass etc. and of course also by external factors, like scale, lighting condition, the orientation of face etc. On the other hand, in most cases the overall shape and size remain the same. So, in order to avoid the problems just described, in our approach, there are two main steps: skin detection and face detection. Through colour transform space, central moments, SR parameters setting to improve the performance.

Now, according the Fig3.61, the details of the experiment will be described as follows:

①: Our original image shown in Fig 3.65, it takes the office environment as background

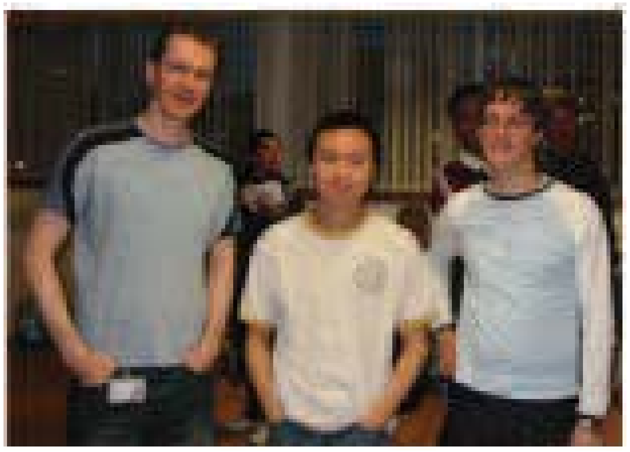


Figure 3.65: Original Image

In order to enhance the contrast of image, applying the histogram equalization, the result shown in Fig 3.66

Applying (3.14) on Fig 3.66 in order to build the skin model for each

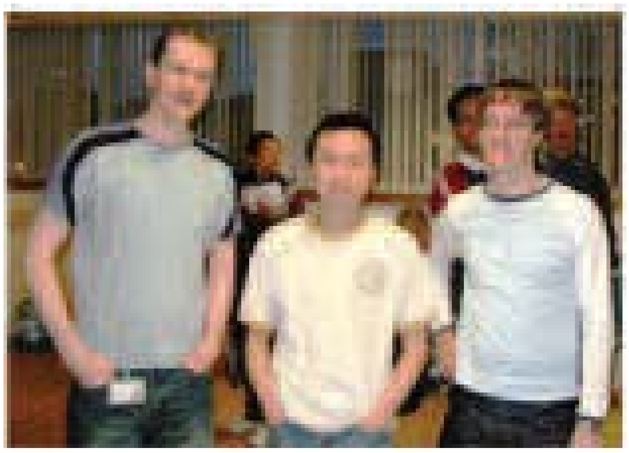


Figure 3.66: Histogram

colour component

$$p(\text{colour/skin}) = sum(\text{colour/skin})/V_{skin}$$

②③: Extract the boundary conditions in the skin model, according the (3.15)-(3.17), the boundary conditions can be obtained in different colour spaces:

$$\begin{cases} 148 \ \leq \quad V \quad \leq \ 185 \\ 189 \ \leq \ U + 0.6V \ \leq \ 215 \end{cases}$$

$$\begin{cases} Y > 85 \qquad or \\ Y < 85, \quad U > 104, \quad Y + U - V > 2 \end{cases}$$

$$\begin{cases} S \leq 21 & , \quad V \geq 2.5S \\ 158 \leq H + V \leq 400 & , \quad H + V > 13S \\ H > 0.2V & , \quad H > 4S \end{cases}$$

To determine these boundary conditions, for simplicity only linear constraints between colour components are considered. Take Eq. (3.15) as an example, first the colour distribution of skin pixels in U-V space is obtained as shown below Fig 3.67. Then, four lines are drawn to extract majority skin pixels in the model, where at least 98% of skin pixels are included in the extracted boundary conditions.

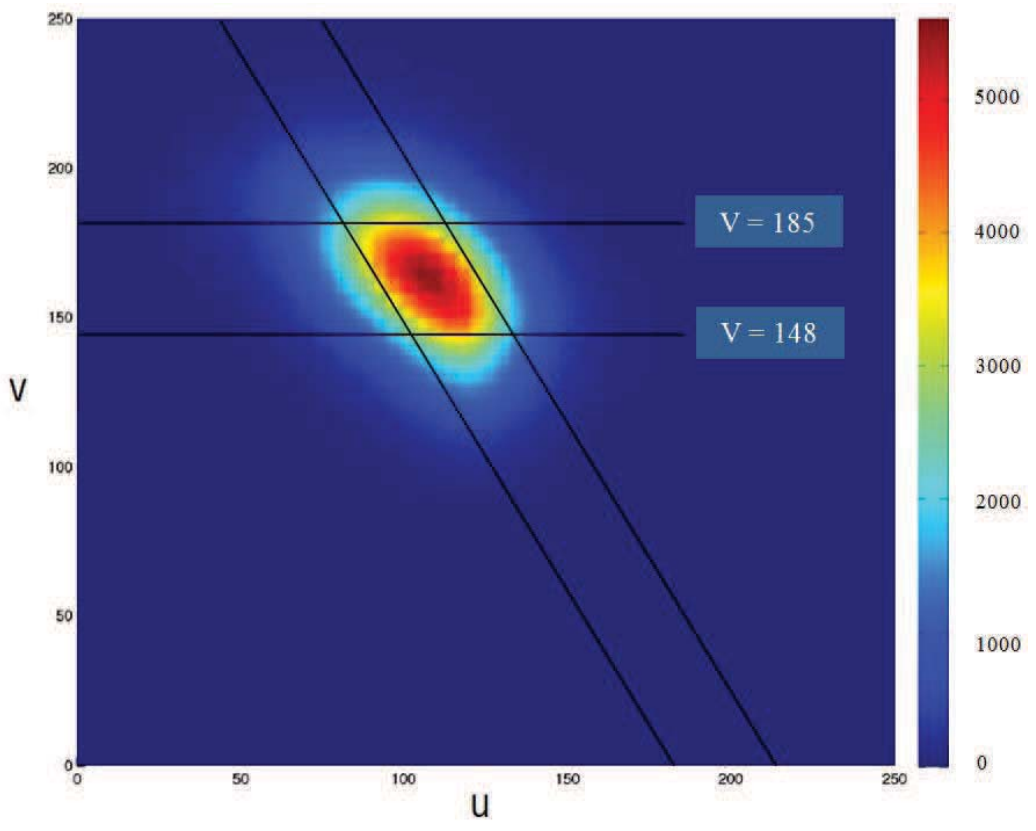


Figure 3.67: Determine the Boundary Conditions from Skin Distribution in UV Space

After extracted the boundary from skin model, the results shown in Fig 3.68 and Fig 3.69.

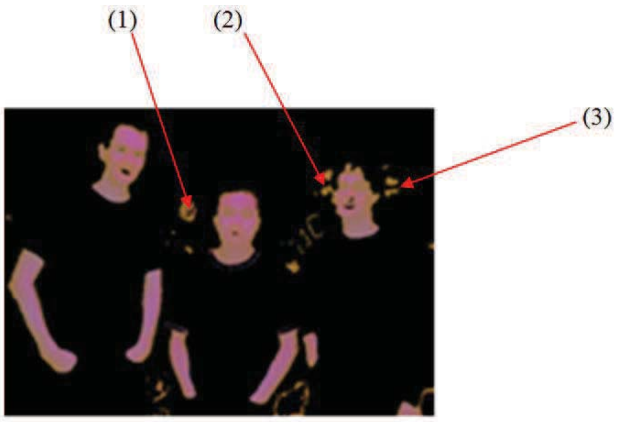


Figure 3.68: YUV Skin Results

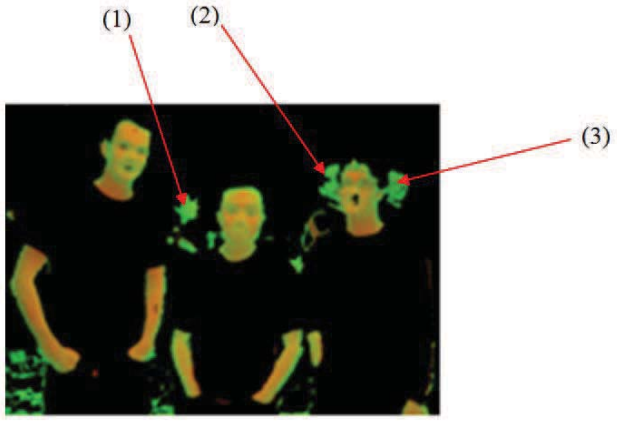


Figure 3.69: HSV Skin Results

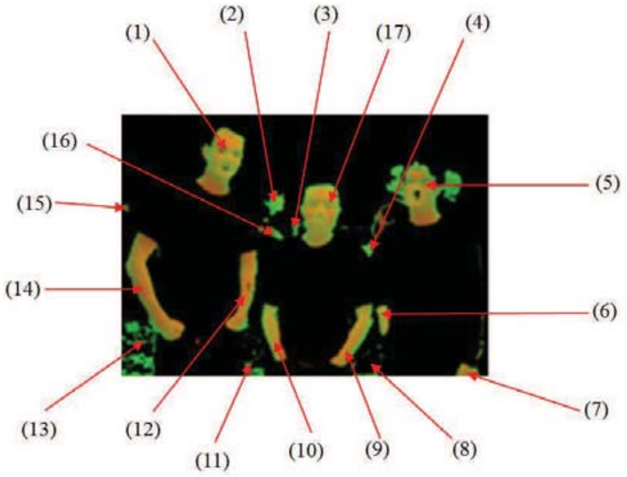


Figure 3.70: Histogram

In the Fig 3.70, 17 skin regions have been found totally, of course, they are not all faces, and even some places are false alarms, and it can be sorted out in next step.

④: Labelled the skin regions, the result shown in Fig 3.68.

⑤: Remove the small regions according the set threshold. The result shown in Fig 3.71, in here, threshold is set to 300, if the pixels in the skin region is smaller than 300, the region should be removed.

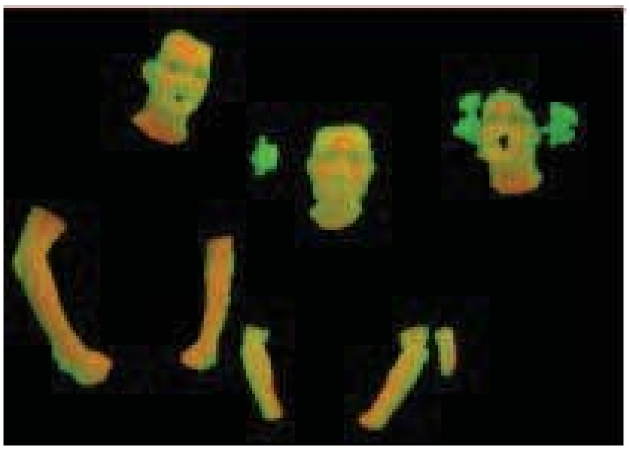


Figure 3.71: Labelled the Skin Regions

⑥⑦: In the Fig 3.71, though the small skin regions have been removed, but the arms still keep in the image and everyone skin region has different angle for the horizontal.

So, in this step, the main axis for everyone skin has been built, through calculate their central moment. After that, rotate everyone skin region according the main axis, and let them all have same angle for the horizontal. The results shown Fig 3.72 and Fig 3.73

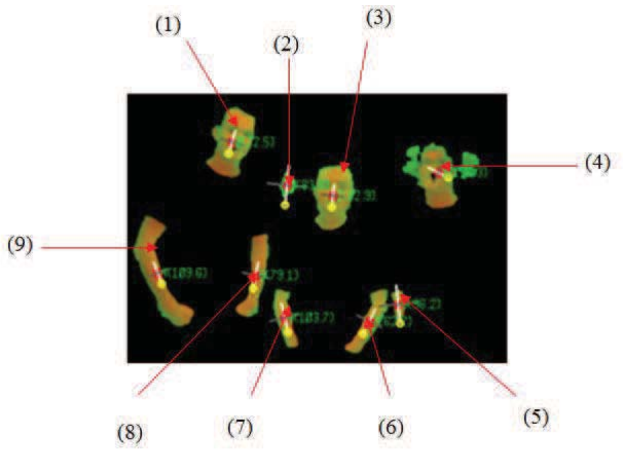


Figure 3.72: Main Axis Detection

In Fig 3.72, the white line is main axis and the angle for each of skin region is:

$$\begin{array}{lll} (1) = 72.5; & (2) = 83.6; & (3) = 82.3; \\ (4) = 156.3; & (5) = 86.2; & (6) = 62.7; \\ (7) = 133.7; & (8) = 79.1; & (9) = 133.6; \end{array}$$

So, according to the angles for every skin region, the skin regions is rotated and the result is shown in Fig 3.73

⑧: In Fig 3.73, everyone skin region has been rotated and has the same angle for the horizontal. Now in this step, SR(width/height ratio) parameters have been applied to remove the arms. The formula shown in (3.18).

$$SR = \begin{cases} width/height & \text{if} \quad width \leq height \\ height/width & \text{if} \quad width > height \end{cases}$$

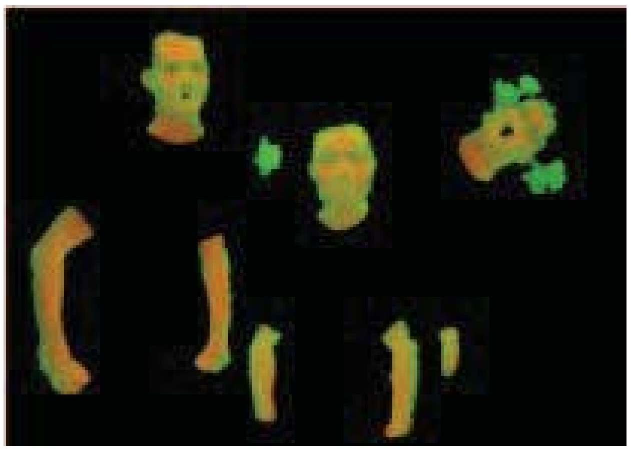


Figure 3.73: Rotation by Main Axis

After testing the images, the valid SR parameters is obtained as [0.550.90]. Apply SR parameters on the Fig 3.73 and removed the skin regions whose W/H ratio is outside of the range of SR parameters, the result shown in Fig 3.74.

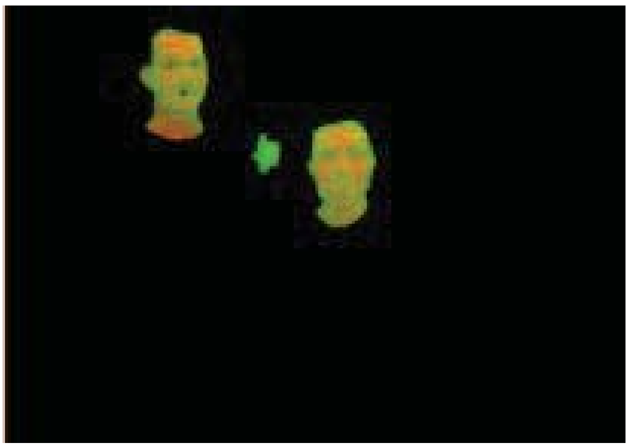


Figure 3.74: Filtered by SR Parameters

⑨: In the Fig 3.74, there one face belongs to background between two frontal faces, in this step through three basic rules: located ears, eyes and mouth to remove the middle face.

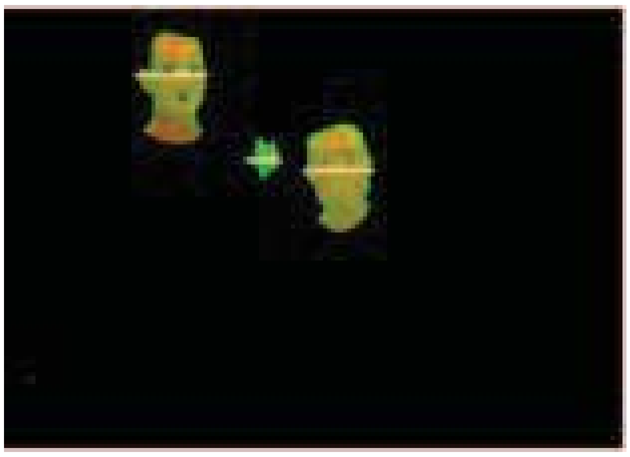


Figure 3.75: Ears Location

In Fig 3.75, ear line extracted according the local maximum width near the centre of the candidate face regions.

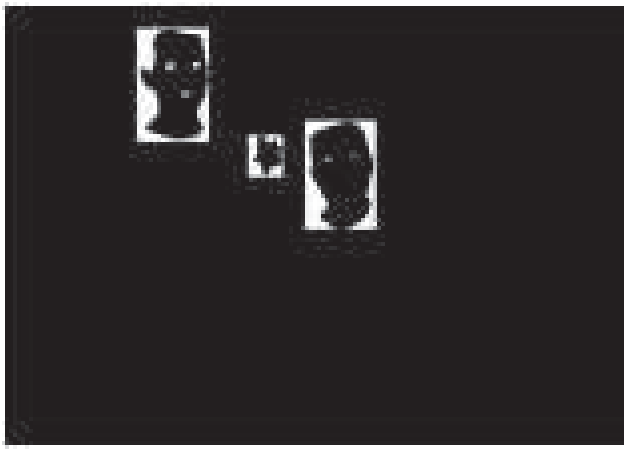


Figure 3.76: Located Open Mouths

Open mouths always have lower illumination intensity to compare the average intensity in the candidate regions. So, in this step, it assumes that if the intensity of the holes less than 80% of average intensity, this hole is mouth. The result is shown in Fig 3.76.

⑩: Through judging the results from step ⑨, the final result can be obtained, it is shown in Fig 3.77 and Fig 3.78.

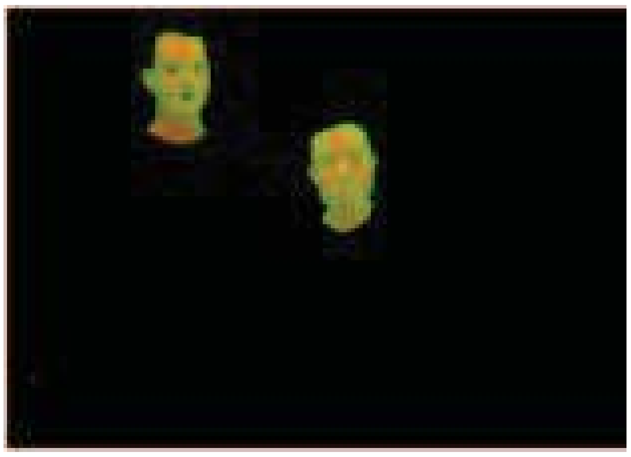


Figure 3.77: Candidate Face

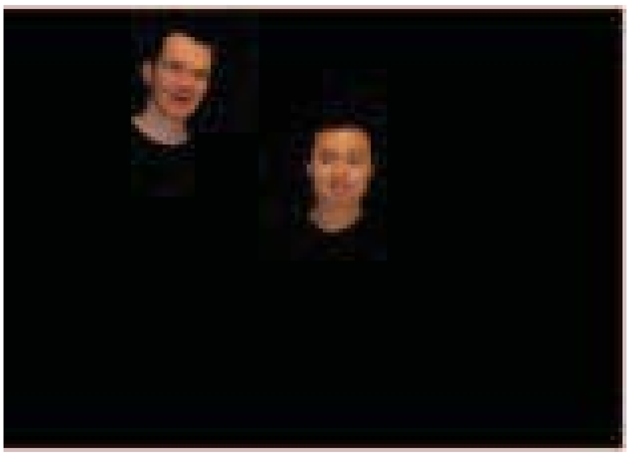


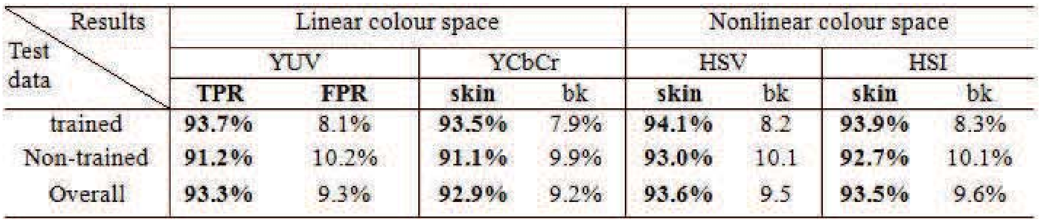
Figure 3.78: Mapped the Original Image

In our experiments, statistical models of skin colours are estimated through histogram based approach using a subset of ECU database [157], in which 500 images are used for training. Afterwards, 100 test images in the office environments for evaluation have been generated. Results on skin detection from both the training images and our own images are summarized in Table

3.6 below. As skin detection is a small portion of work in this chapter, results extracted from various colour spaces are compared, rather than using results from other approaches.

Although the results from different colour spaces are quite comparable, HSV and YUV seem yield slightly better performance in linear and nonlinear colour spaces, respectively. More results on skin and face detection are also given in Fig 3.76, along with discussions in details.

Table 3.6: Skin Detection Results from Different Colour Spaces



Results / Test data	Linear colour space				Nonlinear colour space			
	YUV		YCbCr		HSV		HSI	
	TPR	**FPR**	**skin**	bk	**skin**	bk	**skin**	bk
trained	**93.7%**	8.1%	**93.5%**	7.9%	**94.1%**	8.2	**93.9%**	8.3%
Non-trained	**91.2%**	10.2%	**91.1%**	9.9%	**93.0%**	10.1	**92.7%**	10.1%
Overall	**93.3%**	9.3%	**92.9%**	9.2%	**93.6%**	9.5	**93.5%**	9.6%

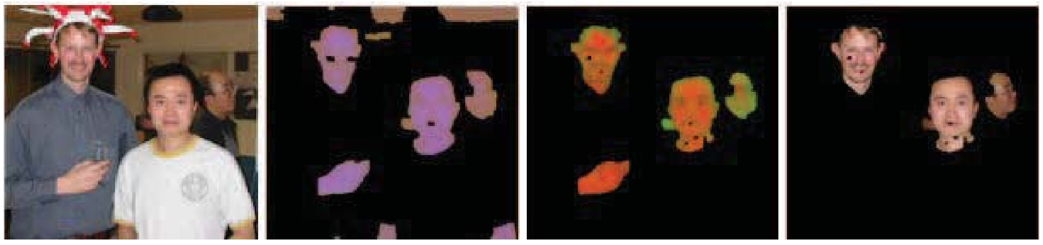


Figure 3.79: Skin and Face Detection Using Image of Peter and Tommy

As for skin detection, skin regions detected from HSV space are more accurate and robust than ones from YUV space, and the skin regions in background can also be detected easily in HSV spaces (see the face in Fig 3.60 and hand near the middle head in Fig 3.76, which means HSV space is less sensitive to variations of illumination intensity

Thresholding by size and ratio is very effective in non-face regions removal. Moreover, the face model composed by the rules on the relative positions of ears and holes of eyes or mouth is also very practical in face detection, as ears can be found in almost every face image, which are more robust for detection even when the face is rotated and eyes are difficult to be detected.

Though our face detection algorithm can achieve quite satisfied results even there are pose variations, there are several additional strategies can be further applied for more robust face detection in our model, such as how to obtain the W/H ratio more accurately if there are connected skin regions and holes, and how to detect eyes and mouth if there are no holes can be found, especially for the face in the background.

With detected regions of skin and face, semantic indexing and retrieval of images are achieved as follows:

1. According to whether skin and face regions can be detected, all the images are automatically annotated as with or without skin/face regions respectively;
2. For those with skin or face regions, size and number of regions are also recorded;
3. For images with face regions, the estimated positions of ears, etc. are also taken in semantic indexing, which can be further used to estimate pose of faces;

4. Finally, these indexes are utilized in semantic retrieval of images.

3.5 Summary

In this chapter, colour edge detection and face detection are discussed, where colour based edge detection and skin detection plays a key role in low- to middle- level image processing.

For colour edge detection, a novel fusion scheme is proposed to make use of inter-channel chromatic difference and intensity information for improved accuracy. Using the standard Canny edge detector with automatically determined parameters, the results have been compared with those extracted from several other colour spaces, including RGB, YCbCr, HSV et al. In addition, colour edge detection from images with manually added Gaussian noise is also investigated. With semi-manually derived ground truth, quantitative evaluation is achieved to validate the effectiveness of the proposed approach. Comprehensive results from several standard test images have fully verified both the effectiveness and robustness of the proposed approach, which is found outperforming edges extracted from RGB, YCbCr and HSV spaces.

For face detection, a straightforward strategy using skin detection followed by knowledge-based shape constraints is adopted. With the attained colour distribution of skin pixels, skin detection can be easily implemented using several extracted linear boundary conditions. By comparative study of skin detection from different colour spaces, it is found that nonlinear colour spaces, such as HSV, can obtain more accurate and robust skin result. Moreover, shape filtering and knowledge-based modelling proves to be very useful in face detection.

Chapter 4

Hand Gesture Recognition

4.1 Introduction

Interactive image processing is focus on the communication between the machine and human. Normally, it includes 5 steps: definition of object, analysis and training phase, recognition and feedback. The kernel of interactive image processing is how to locate the interesting objects and track them more accurately and more effectively.

In this chapter, a novel dynamic simulation scheme for interactive image processing is proposed. This scheme includes two steps: Hand Motion Detection and Hand Gesture Recognition.

In Hand Motion Detection, the movement of hand motion is identified by the difference between current image and previous image, if the difference is beyond the predefined threshold value, then the typical movement of hand motion is detected.

In Hand Gesture Recognition, some low-level features like colour, shape, etc. have been applied as well as one important feature: orientation histogram on identifying the specific hand gesture. In this processing, Gaussian Mixture Model (GMM) is used as our classifier and the Expectation-Maximization algorithm can calculate the maximum likelihood between the testing image and samples of each type of hand gestures in order to adjust the parameters of the GMM.

4.2 Motion Detection for Moving Object Segmentation

In this section, how to detect the change of hand gestures is focused. Obviously, there are two important steps when detecting the change of hand gestures, moving detection and moving object segmentation. Although, many approaches have been advised to apply on moving detection in a continuous video stream, the basic principle is to compare the current video frame with its previous one or against a fixed/dynamic background. This is useful application if marking the changed parts rather than the whole frame.

4.2.1 Frame Differencing

Frame differencing is a straightforward approach for motion detection, which uses the difference between two images as an indicator to show the changes caused by motion. Let $img(i)$ represent the i^{th} image in a sequence, the frame difference of this frame and its previous frame is defined as:

$$diff(i) = |img(i) - img(i-1)| \tag{4.1}$$

If the original image is a colour one containing three or more colour components, the difference above will also generate a colour image. For simplicity, the input image is usually converted to grey one before differencing. Consequently, the resulted difference $diff(i)$ will also be an 8-bit grey image.

Fig. 4.1 shows two colour input images and their associated grey images for motion detection.

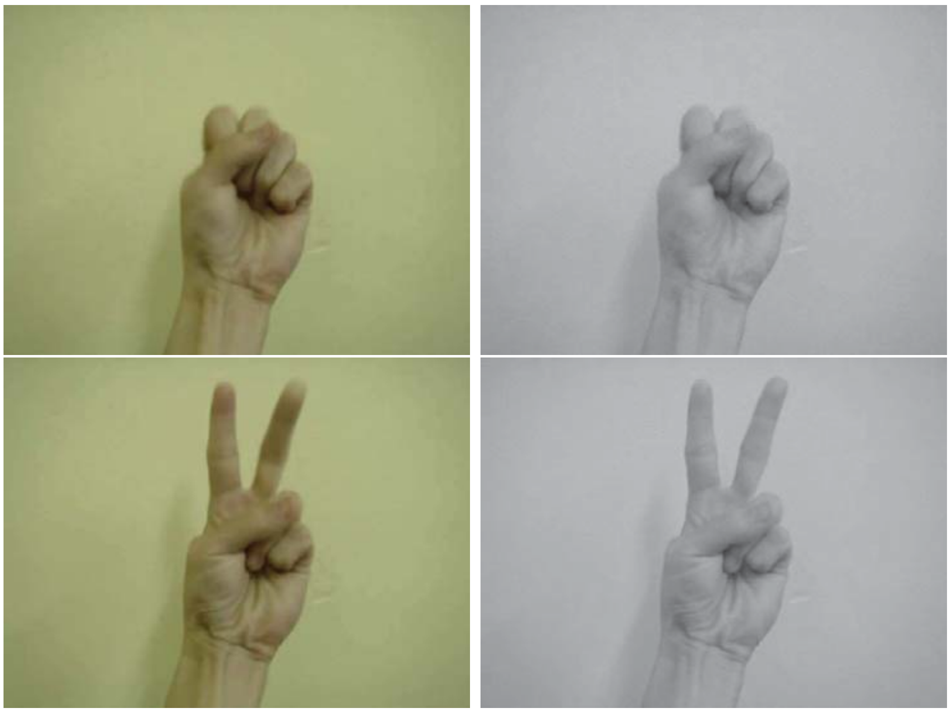


Figure 4.1: Two Colour Input Images (left) and Their Corresponding Grey Images (right)

With the extracted $diff(i)$, a simple thresholding th is applied to decide pixels changed or not. In here, a binary image mask $mask\_diff(i)$ can be obtained, in which white and black pixels represent those having been changed or remaining unchanged by motion.

$$mask_d iff(i) = \begin{cases} 1, & \text{if } diff(i) > th \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

One fixed threshold is not using in 4.2, one captured threshold is automatically applied for every $diff(i)$, that means, the threshold is obtained for one $diff(i)$ through one automatic way, the formula is shown as follows, where μ and σ denote respectively the mean and standard deviation of $diff(i)$.

$$th(i) = \mu(i) + \sigma(i) \tag{4.3}$$

$diff(i)$ in Fig. 4.2 has $\mu = 5.47$ and $\sigma = 10.02$, hence, $th(i) = 15.49$. The detected mask images under different thresholds are compared in Figure 4.3.

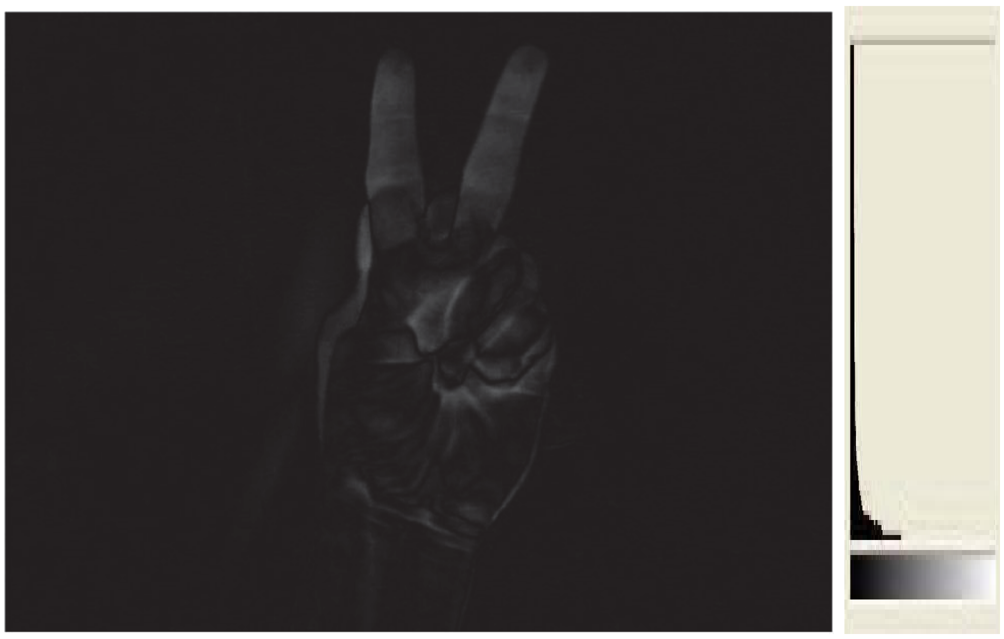
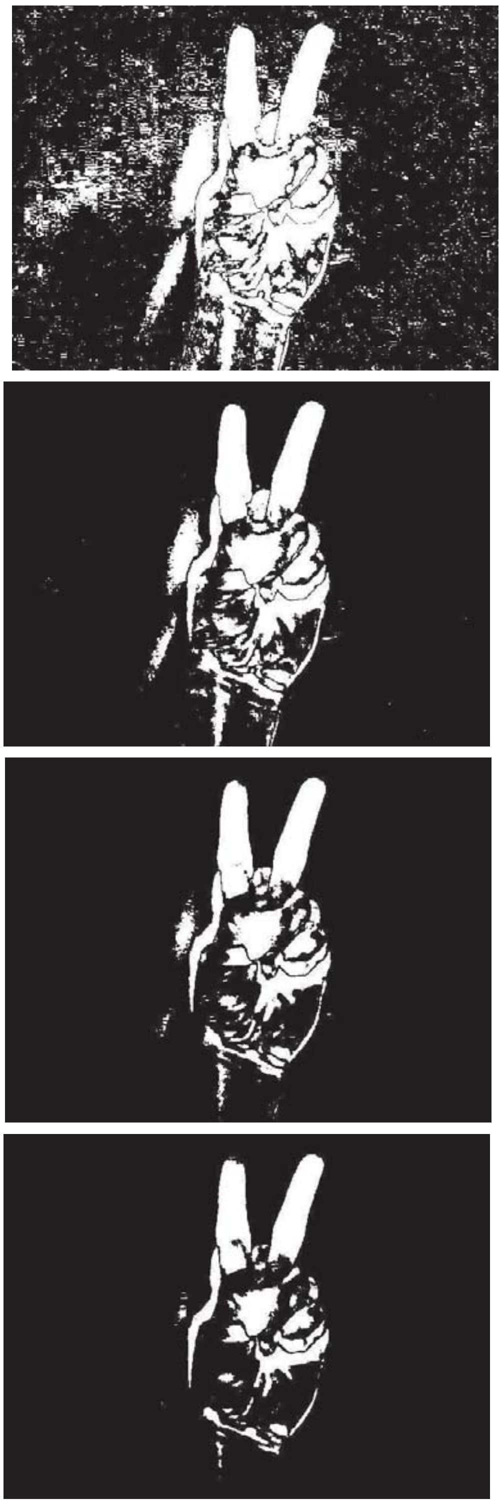


Figure 4.2: $diff(i)$ and Its Histogram


(a) $th = 5$

(b) $th = 10$

(c) $th = 15$

(d) $th = 20$

Figure 4.3: Extracted Binary Masks of Motion Using Different Thresholds

From Fig 4.3, the threshold is set to 5, there is less accurate, so with many white areas (false pixels) left in the picture. If the threshold is set to 20, obviously, the false pixels have been reduced, but it made holes in mask, such as the one on the left finger. So, after analysing the results of Fig 4.3, set the threshold as 15 is the more accurate and more robust.

To obtain more accurate and more clear photos, normally, the procedure and post-processing needs to run one more time. In here, the morphological filter as the post-processing has been applied. Firstly, apply erosion on detected binary mask $mask\_diff(i)$, followed by a dilation processing, both using 3×3 rectangle structure. If the new obtained mask image is represented as $mask\_diff1(i)$, like:

$$mask\_diff1(i) = dilation(erosion(mask\_diff(i))) \tag{4.4}$$

The result of processing is shown in Fig 4.4, where the binary mask $mask\_diff1(i)$ obtained after erosion filter and dilation processing is shown at the left, and the mask attached on the original image to illustrate the frame differencing clearly as shown at the right.

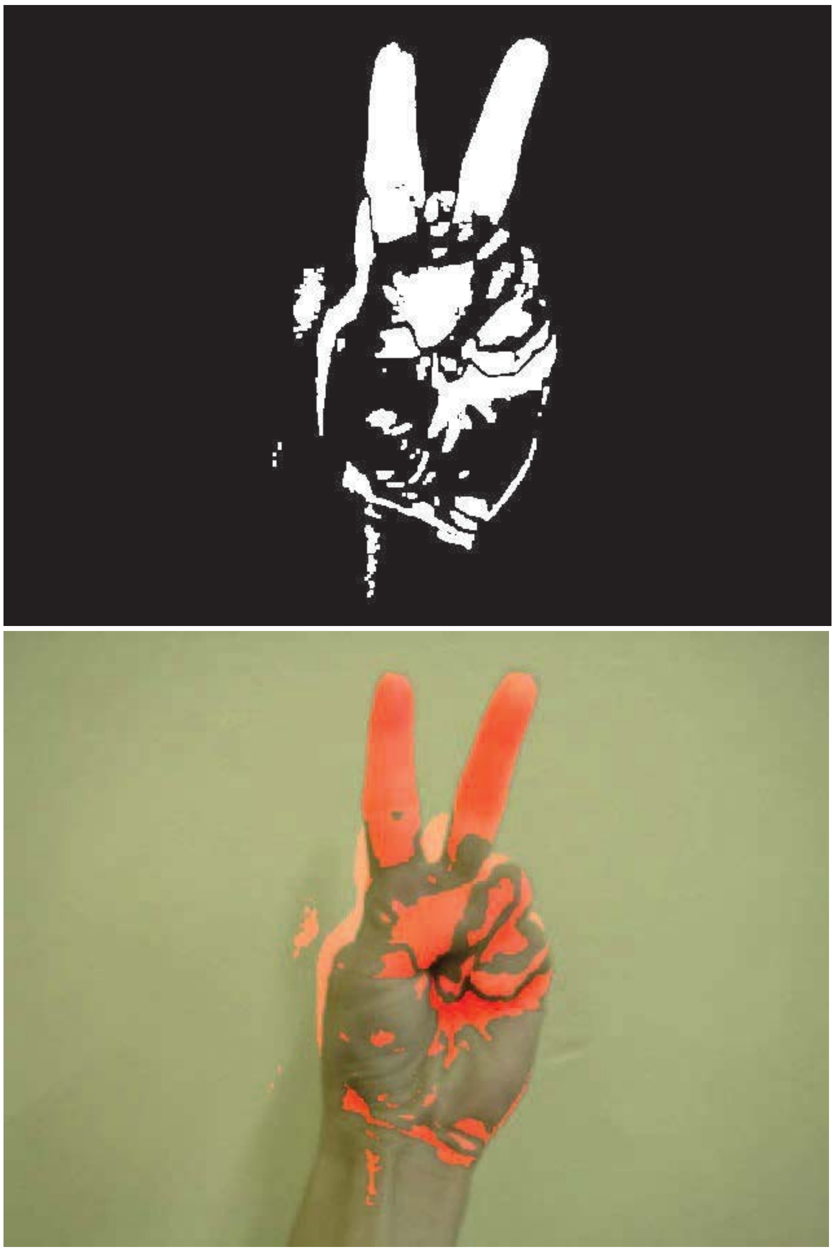


Figure 4.4: Extracted Binary Masks of Motion Using Different Thresholds

4.2.2 Background Modelling

Under some particular situations, like when the object moves smoothly or very slowly, it is difficult to track the whole moving changes, as shown in Fig 4.5. So, to accommodate this situation, one new approach can be applied to resolve it, that is background modelling.

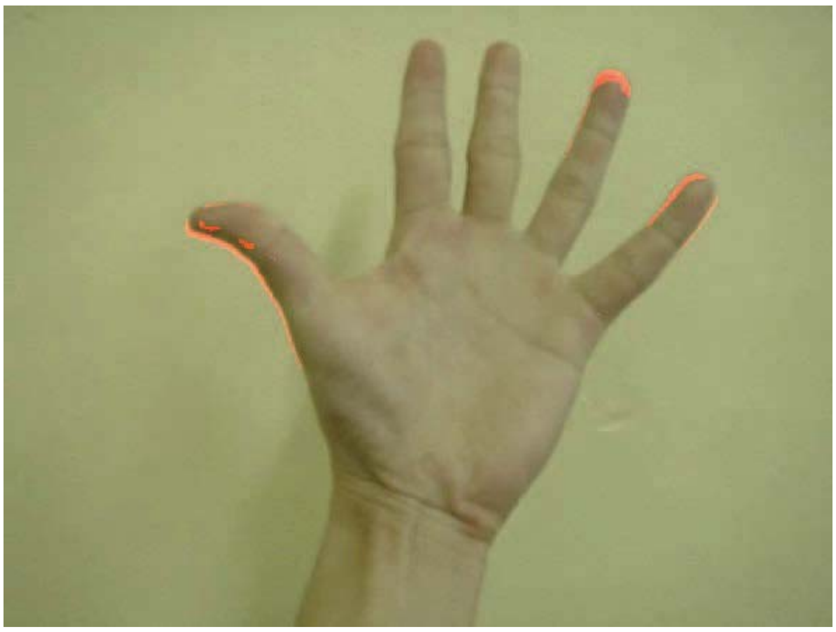


Figure 4.5: Smooth or Slow Movement Causes Motion Regions Inconspicuous

Method of background extraction during training sequence and updating it during input frame sequence is called background modelling. This method has been widely applied on computer vision. The kernel problem in moving object detection is how to extract one clean background and its updating.

The fastest and the most memory compact background modelling is the running average method. In this method, background extraction is done by arithmetic averaging on a training sequence. The method is introduced as following.

From the k^{th} image I_k in the image sequence, in total a temporal window is defined as $(I_{k-n_0}, I_{k-n_0+1}, \ldots, I_k, \ldots, I_{k+n_0-1}, I_{k+n_0})$, where there are $2n_0+1$ frames used, including the current frame, its previous and subsequent n_0 neighbouring frames, respectively. The average image over a temporal window is then extracted as follows:

$$av_k = \frac{1}{2n_0+1}\sum_{m=k-n_0}^{k+n_0} I_m \tag{4.5}$$

Consequently, each pixel value in represents the mean value of that pixel over the temporal window. Meanwhile, the standard deviation for each pixel over the temporal window is also attained as follows:

$$std(i,j) = \sqrt{\frac{1}{2n_0+1}\sum_{m=k-n_0}^{k+n_0} [I_m(i,j) - av_k(i,j)]^2} \tag{4.6}$$

For background pixels, limited changes in the temporal window are expected, i.e. smaller values in std_k. Therefore, a simple thresholding of std_k image can help to extract the background, where the threshold is also determined adaptively using the same strategy as introduced in (4.3).

$$bg_k(i,j) = \begin{cases} 1 & \text{if } std_k(i,j) < th_2 \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

With the extracted background, the simple frame differencing is modified as follows.

$$diff_k(i,j) = \begin{cases} |av_k(i,j) - I_k(i,j)| & \text{if } bg_k(i,j) = 1 \\ std_k(i,j) & \text{otherwise} \end{cases} \tag{4.8}$$

And the mask of changed pixels is then obtained using the same way as applied to the difference from frame differencing, i.e.

$$mask_d iff_k(i,j) = \begin{cases} 1 & \text{if } diff_k(i,j) > th_3 \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

where th_3 is a threshold determined from $diff_k$ using the same strategy in (4.3).

From a sequence of frames which begins with fist, then transforms into "Victory" sign and ends with fist, the original images are shown as following Fig 4.6.

After applying the method which described in Fig 4.6, each pixel is represented as a significant change or non-significant change. Now, the binary mask can be extracted from the frame difference shown in Fig 4.4 with background removed, the results is shown in Fig 4.7, the white pixel areas indicate the foreground or object, and the black pixel areas indicate background.

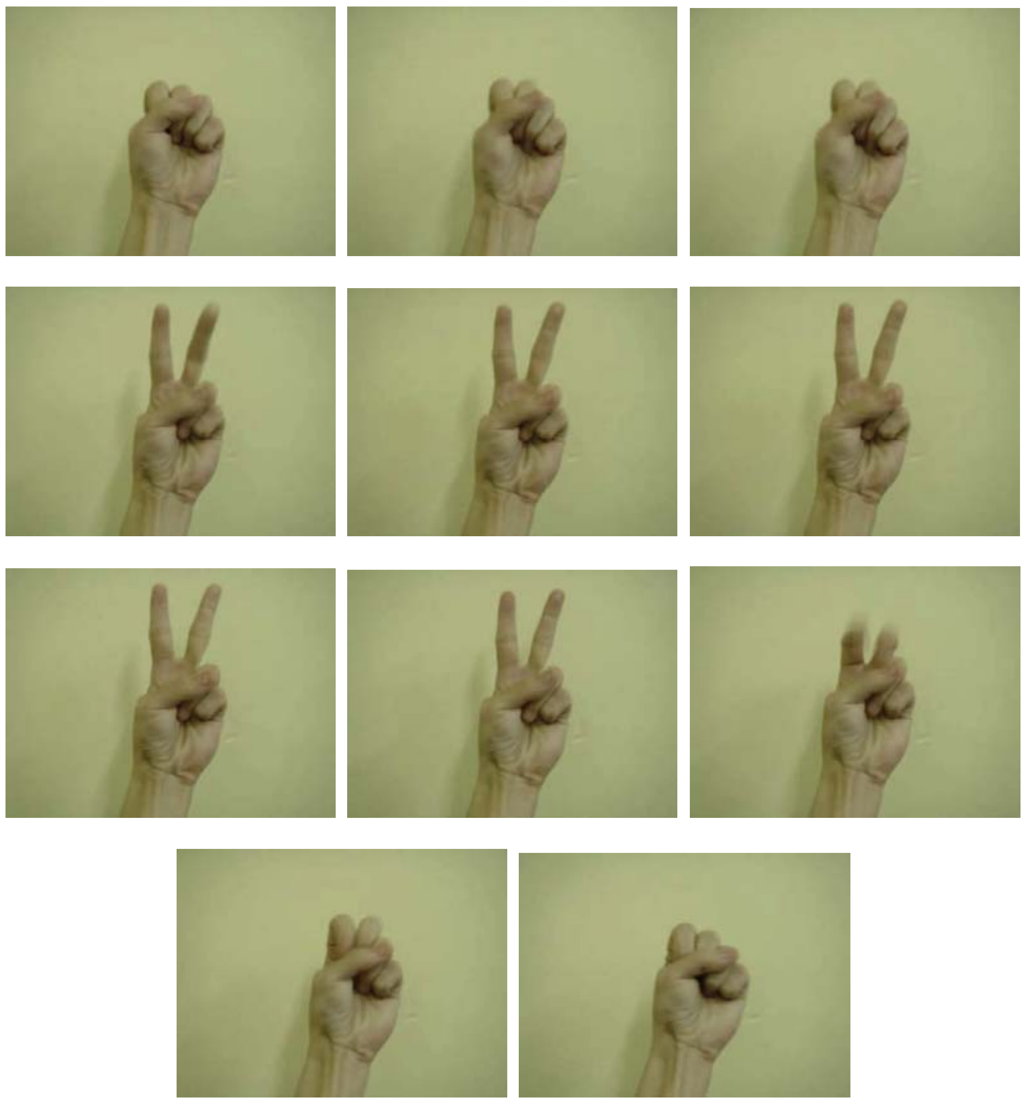


Figure 4.6: Images: from left to right, top to below, the hand gesture begins with fist, then turn to "Victory" sign and ends with fist.

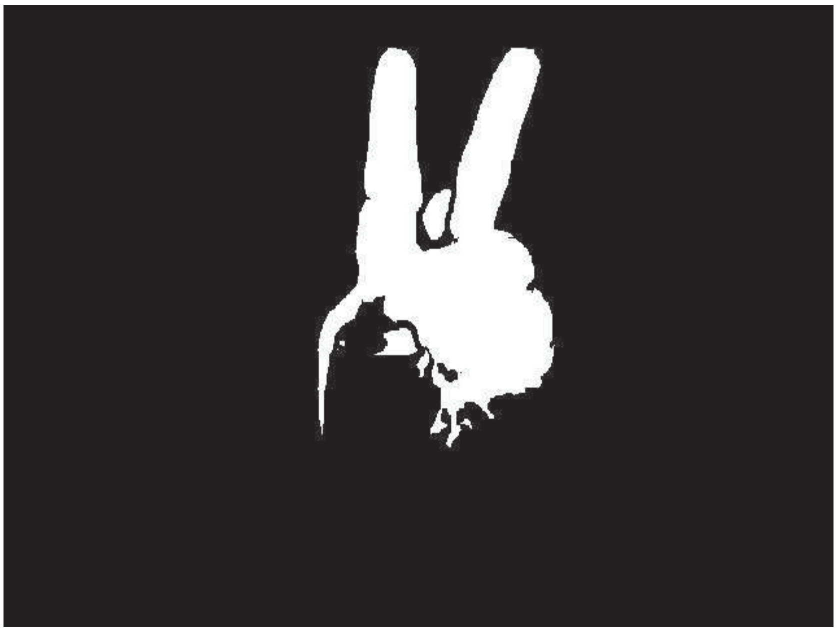


Figure 4.7: Binary Mask of Frame Difference Modified with Background Modelling

Obviously, after applying the background modelling on the frames, the result is better than just applying on frame difference. The real moving object can be obtained, and this is the bedrock for the recognition of hand gesture in the next part.

In this part, firstly, the frame difference to track one moving object is described, but this method cannot track the moving object precisely when the object moving smoothly or slowly. Under this situation, the background modelling to co-operate with frame difference has been introduced, and more accurate results can be obtained (see in Fig 4.7).

Before extracting the moving object, one new method will be applied, namely Temporal Window over Images Sequence to produce standard deviation for each pixel on the temporal window. This approach can reduce the noise and improve the accuracy rate for frame difference, base motion detection.

4.3 Hand Gesture Recognition with Gaussian Mixture Model

4.3.1 Generation of Proposed Approach

It is hard to track the hand gesture, because it is difficult to predict the hand's behaviours, it also means, it is difficult to extract the raw information from gesture images for 3D hand reconstruction.

Normally, one model needs to set up and adjust parameters for this model in order to match the hand gesture by tracking. The parameters in this model should provide essential information from captured pictures. So how to extract the essential information from lots of captured images becomes one big problem in our project.

In this part, one new approach will be discussed, which can recognise gesture effectively and extract feature robustly. It focuses on estimating the gesture contained in an image by analysing different complex features including shape, colour and orientation histogram quantized in Gaussian Mixture Model (GMM).

GMM is a widely used statistical model in many applications of pattern recognition, which is often regarded as a versatile modelling tool as it can be used to approximate any Probability Density Function (PDF) given a sufficient number of components, and impose only minimal assumptions about the modelled random variables.

The advantage includes a rigorous statistical basis, the possibility of encoding spatial, colour, texture and motion features in a unified system, and the ability to trade off accuracy of representation against data volume. Due to such advantages, our proposed technique builds upon the GMM to estimate the mutative meaning of human gestures in a compact and precise manner.

4.3.2 Introduction of Gaussian Mixture Model

Gaussian Mixture Model (GMM) is a parametric probability function, it is one of the most widely used mixture modelling techniques. It can produce accurate results when data are generated from a set of Gaussian distributions [99, 100]. If assume $X_i = \{x_t, 1 \leq t \leq T^i\}$ represent the feature vectors for data points belong to $i-th$ class, they can be modelled by:

$$P(X_i|\theta^i_{GMM}) = \prod_{t=1}^{T^i} \sum_{j=1}^{J} P(z_j) P_{z_j}(x_t|u_j, \Sigma_j) \tag{4.10}$$

in (4.10), J is total number of the data points; θ^i_{GMM} is model parameter, including $\{P(z_j), \mu_j, \Sigma_j, 1 \leq j \leq J\}$, so, $P_{z_j}(x_t|\mu_j, \Sigma_j)$ can be represent the

Gaussian distribution for the j^{th} class, μ_j is a mean vector, and covariance matrix Σ_j.

$$P_{z_j}(x_t|\mu_j, \Sigma_j) = \frac{1}{(2\pi)^{D/2}|\Sigma_j|^{1/2}} \exp\{-\frac{1}{2}(x_t - \mu_j)^T \Sigma_j^{-1}(x_t - \mu_j)\} \quad (4.11)$$

in (4.11) D is the dimension of the feature vector x_j, in order to reduce the size of parameter space, the Σ_j can transform to one a diagonal matrix as $diag\{\sigma_{jd}^2 : 1 \leq d \leq D\}$

Analysis the formula (4.10), when multiple Gaussian models generate the data points belong to one specific class, it produce one weight $p(z_j)$ same time. If define $\omega_j = P(z_j)$, the Gaussian Mixture model can be seen composed by three basic parameters: Mixture weight, Mean vector and Covariance matrix, which can be represented as λ:

$$\lambda = \{\omega_j, \mu_j, \Sigma_j\} \quad (4.12)$$

where ω_j is the mixture weight, μ_j is the mean vector, and Σ_j is the covariance matrix. The λ is used to stand for every single image. Additionally, it is:

$$\begin{cases} b_j(x) = P_{z_j}(x_t|\mu_j, \Sigma_j) \\ \sum_{j=1}^{J} \omega_j = 1 \end{cases} \quad (4.13)$$

in order to simplify the expression of $P_{z_j}(x_t|\mu_j, \Sigma_j)$ in the following training phase.

4.3.3 The Expectation Maximization Algorithm

The EM algorithm is an efficient iterative procedure to compute the Maximum Likelihood (ML) [140, 141, 102] estimate in the presence of missing or hidden data. In this thesis, after the GMM's results from training photos and testing photos respectively, applying EM to calculate the Maximum Likelihood between these two results and identify what is kind of hand gesture in testing photos. Now, the details about Expectation Maximization Algorithm will be described below.

Normally, Expectation Maximization Algorithm includes two steps: E-Step and M-Step:

- E-Step: depend on the current model to guess the probability distribution of missing or hidden data;
- M-Step: according the result from E-step, re-estimate the model parameters.

Example, if X is set to one random vector. The aim is find out the parameter θ, so that $P(X|\theta)$ is a maximum. This is called the Maximum Likelihood (ML) estimate for θ. Normally, the "log likelihood function" to estimate θ is used, it shown in (4.14).

$$L(\theta) = \ln P(X|\theta) \tag{4.14}$$

In (4.14), it is known $\ln(x)$ is one increase function, so, if θ maximizes the $P(X|\theta)$ same time the maximum of $L(\theta)$ is obtained.

The EM algorithm is an iteration procedure for maximize of $L(\theta)$. For example, after n^{th} iteration, one estimation value θ_n is obtained. Since the aim is maximize the $L(\theta)$, so the updated estimate θ make:

$$L(\theta) > L(\theta_n) \tag{4.15}$$

In the other words, the difference needs to be maximized:

$$L(\theta) - L(\theta_n) = \ln P(X|\theta) - \ln P(X|\theta_n) \tag{4.16}$$

4.3.4 Training Phase

In training phase, our aim is to obtain the mixture model λ, in here, λ represents one feature vector for every certain image. In our project, maximum likelihood means try to find the exactly λ from training images.

For example, it assumes one feature vector $X = \{x_1, x_2, \ldots, x_T\}$ is extracted from one image, T is number of features, the likelihood of GMM is shown as follow:

$$P(X|\lambda) = \prod_{t=1}^{T} p(x_i|\lambda) \tag{4.17}$$

Normally, the function $P(X|\lambda)$ is nonlinear, so ML is always applied on estimating the parameter of GMM when the $P(X|\lambda)$ convergent.

Firstly, it assumes one λ, estimation algorithm will use this λ to predict new $\overline{\lambda}$ for new model to satisfy the relationship of $P(X|\overline{\lambda}) > P(X|\lambda)$. Then, $\overline{\lambda}$ will replace λ in new model. This calculation will not stop until $P(X|\lambda)$ is convergent. During this procedure, in order to guarantee the approximation of GMM, the following estimation has been calculated:

$$P(i|x_i, \lambda) = \frac{\omega_i b_i(x_i)}{\sum_{k=1}^{T} \omega_k b_k(x_k)} \tag{4.18}$$

where the mixture weight is estimated as

$$\omega_i = \frac{1}{T} \sum_{t=1}^{T} p(i|x_t, \lambda) \tag{4.19}$$

The estimation of mean vector is:

$$\mu_i = \frac{\sum_{t=1}^{T} p(i|x_t, \lambda) x_t}{\sum_{t=1}^{T} p(i|x_t, \lambda)} \tag{4.20}$$

The estimation of covariance is:

$$\Sigma_i^2 = \frac{\sum_{t=1}^{T} p(i|x_t, \lambda) x_t^2}{\sum_{t=1}^{T} p(i|x_t, \lambda)} - \mu^2 \tag{4.21}$$

4.3.5 Methods of Testing

In this process, the maximum of a posterior criterion to classify all images has been applied. That is calculation of the likelihood for every different type between the testing images and pre-assigned images, in order to obtain one maximum value. Normally, the testing image is classified into one certain type, in this type, the testing image has maximum value of likelihood compare with the others images. The equation as follows:

$$\hat{S} = \arg \max_{1 \leq k \leq S} Pr(\lambda_k | X) \tag{4.22}$$

where S is the total of all pre-assigned different types, $\hat{S}$ is the certain type which the testing image is classified to, λk is the model of pre-assigned type K, and X is the vector of features of the testing image.

This equation can be transformed to another by the Bayesian rule below:

$$\hat{S} = \arg \max_{1 \leq k \leq S} \frac{p(X | \lambda k) Pr(\lambda_k)}{P(X)} \tag{4.23}$$

When $Pr(\lambda_k) = 1/S$, $\hat{X} = \arg\max_{1 \leq k \leq S} P(X|\lambda_k)$. By calculating their logarithms:

$$\hat{S} = \arg \max_{1 \leq k \leq S} \sum_{t=1}^{T} \log p(x_t | \lambda_k) \tag{4.24}$$

4.4 Extraction of Hand Features

After the processing steps described, the three the most important features from all images can be extracted, which include colour, shape, and the orientation histogram. Now, the details for these three features is described.

4.4.1 Colour Feature

Colour is one of significant features in objects identification. In the approach of this project, the skin colour detection is applied on hand detection, because the colour of hand has big differences compared with the colour of background. In here, the colour detection made the hand detection more quickly and more precisely.

After so many years researching, scientists found the colour of human being, especially the skin colour pixels have similar chrominance components and intensities [143, 144]. According the literature survey, the YCbCr colour space is more powerful and more accurate than the RGB colour space. The transformation formula from RGB to YCbCr is:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 65.481 & 128.553 & 24.966 \\ -37.797 & -74.203 & 112.000 \\ 112.000 & -93.786 & -18.214 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{4.25}$$

The chrominance values in the skin region are stable, that means the skin colour is fairly uniform [145].

4.4.2 Shape

In digital image processing, the shape of one object is represented by its edge. In our project, the extracted edge of one object through Canny edge operator, and apply histogram intersection technique to retrieve of corresponding histograms.

The Shape module as one type of module is also can be used as one way to classify one specific object. The shape which is extracted from a binary image is always represents the run-length values of the given image in each of 4 directions. In this thesis, *Hough Transformation* is applied to detect how many straight lines in the pre-assigned the images.

4.4.3 Gradient Orientation Histogram

The local orientation angle θ can be calculated as image gradients, is defined by horizontal and vertical image pixel differences as follows:

$$\theta(x, y) = \arctan[I(x, y) - I(x-1, y), I(x, y), I(x, y-1)] \tag{4.26}$$

For gesture recognition, shift-invariant is a basic principle. To achieve this, how often each orientation element occurred needs to measure in the histogram. Therefore, a vector Φ of N elements is formed, with the i^{th} element showing the number of orientation elements $\theta(x, y)$ between the angles $\frac{360^o}{N}(i - \frac{1}{2})$ and $\frac{360^o}{N}(i + \frac{1}{2})$:

$$\Phi(i) = \sum_{x,y} \begin{cases} 1, & \text{if } |\theta(x,y) - \frac{360^o}{N} i| < \frac{360^o}{N} \\ 0, & \text{otherwise} \end{cases} \tag{4.27}$$

4.5 Implementation and Experimental Results

In our scheme, the key point depends on three important features (colour, shape and gradient orientation histogram), which are extracted from re-assigned hand gestures image, to build one Gaussian Mixture Model. And apply EM to calculate the maximum likelihood between the training images and testing images in order to classify the type of hand gesture in testing images.

Combining GMM and EM, our approach can improve the performance over other exiting methods and not increase the computation complexity.

According the Fig 4.8, the details of experiment will be described as follows:

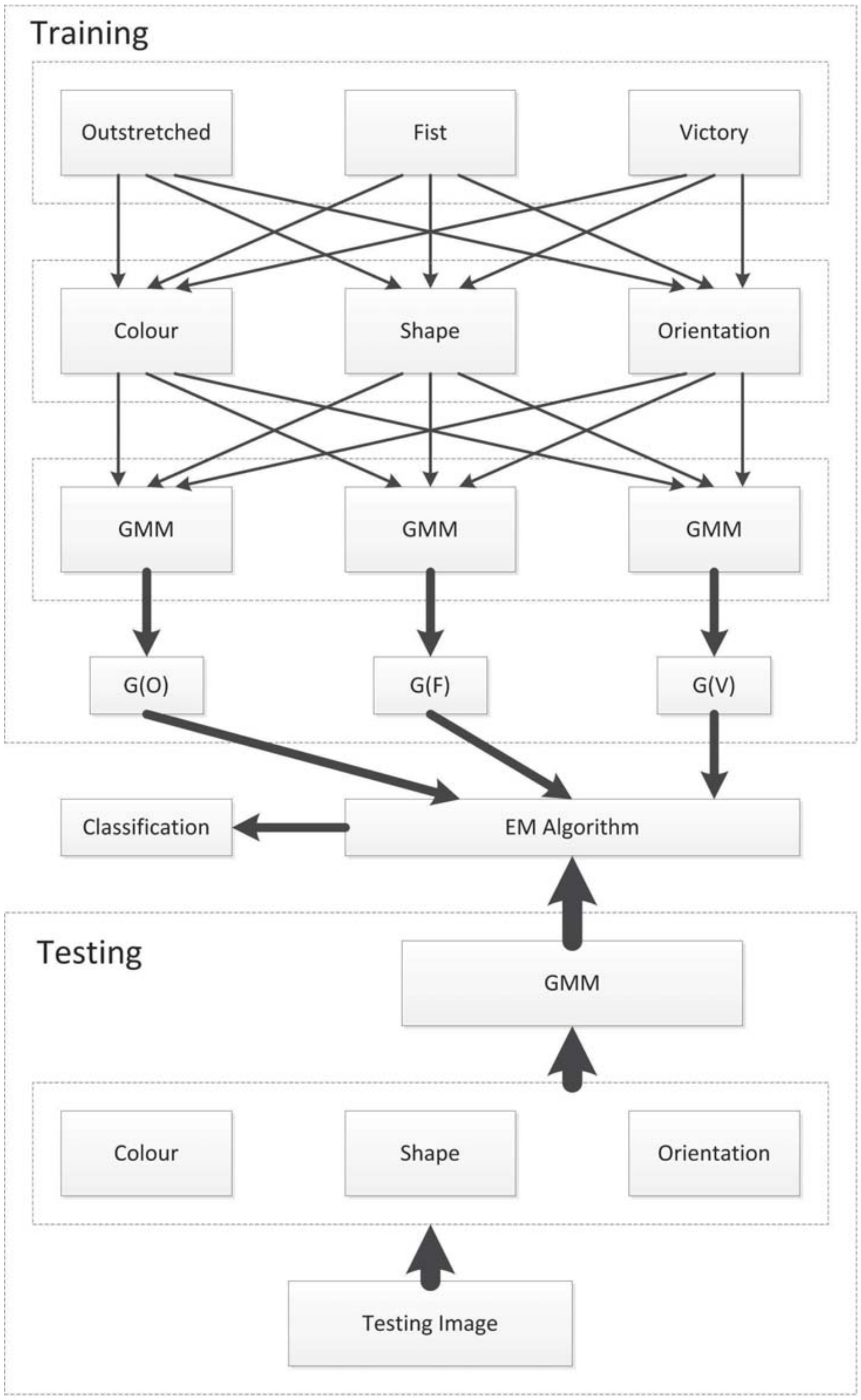


Figure 4.8: Flow Chart for Hand Gesture

①: Before starting the experiment, one group of hands gesture as testing data will be collected. The testing data includes 3 classes has been classified, according to the different gestures, shown in Fig 4.9. The first one is whole 5 fingers outstretched; the second one is one fist; the third one is just two fingers outstretched.

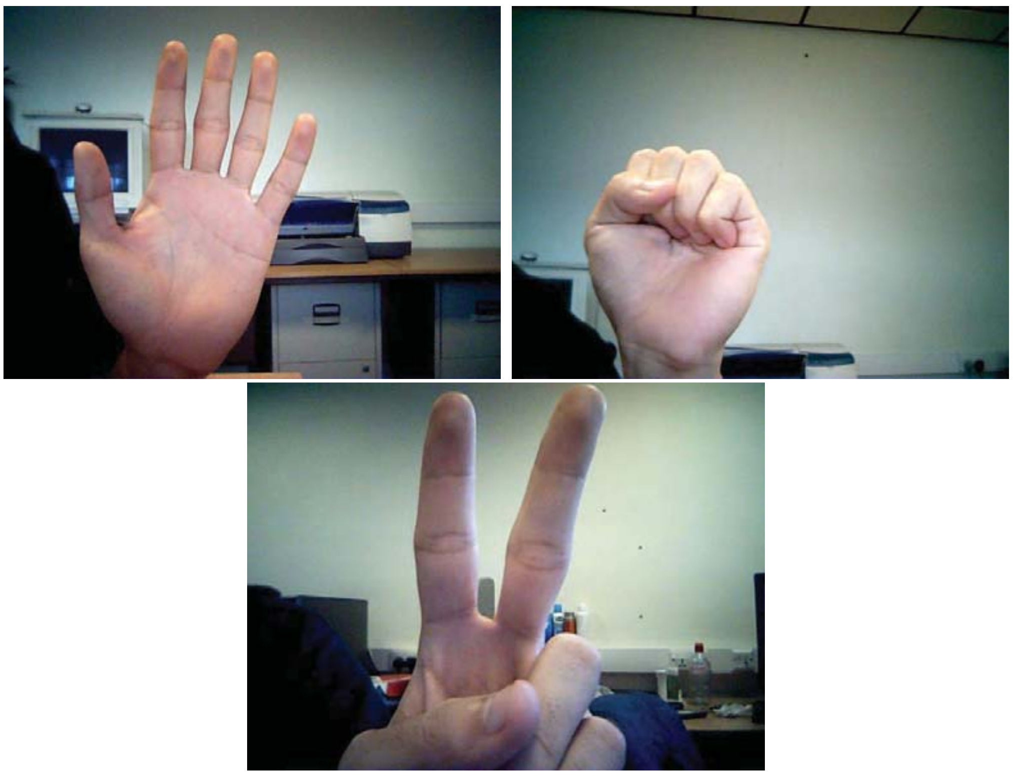


Figure 4.9: Original Images for Different Hand Gestures

②: According to the original images provided in Fig 4.9, the features needs to extracted for each of hand gesture, as described in section 4.5. In this experiment, three features is focused: colour, shape and gradient orientation histogram.

Colour feature: firstly, the original image from RGB domain is transformed into YCbCr domain. Because the YCbCr colour space is more powerful and more accurate than the RGB colour space.

And Recent research has also proved that the chrominance values in the skin region are narrowly distributed, which implies that the skin colour is fairly uniform [106]. According the reference, in this experiment, the value of Cb and Cr is [106]:

$$\begin{array}{lll} & Cb \in [77, 127] & \\ & Cr \in [133, 173] & \\ \text{Outstretched :} & Cb = 113.568, & Cr = 167.894 \\ \text{Fist :} & Cb = 92.145, & Cr = 154.373 \\ \text{Victory :} & Cb = 83.565, & Cr = 147.622 \end{array}$$

Shape feature: in this step, the RGB image is transformed to grey scale one in order to extract the edge for different hand gestures, in here, the Canny edge detector has been applied, and the results are shown in Fig 4.10 and Fig 4.11.

In here, the shape feature has been applied in order to find how many

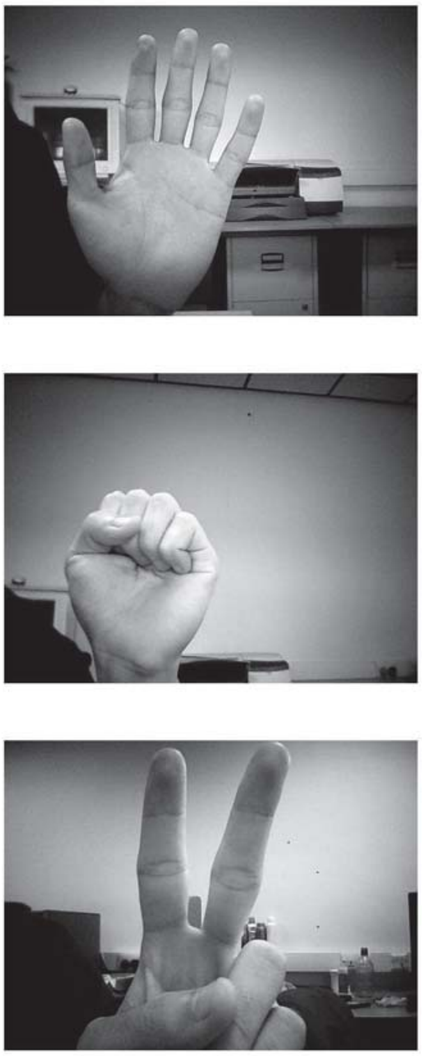


Figure 4.10: Grey scale for RGB

straight lines in each of image. For example, in the outstretched image, in the Fig 4.11 (a), there are 10 straight lines around the brim of the fingers, therefore, that 10 straight lines is one discriminate feature for outstretched can be defined. Of course, there are 0 straight line in Fist and 4 straight lines in Victory. In this experiment, Hough Transformation to detect straight lines in images has been applied. In here, the label SL is set to represent how many straight lines in the image.

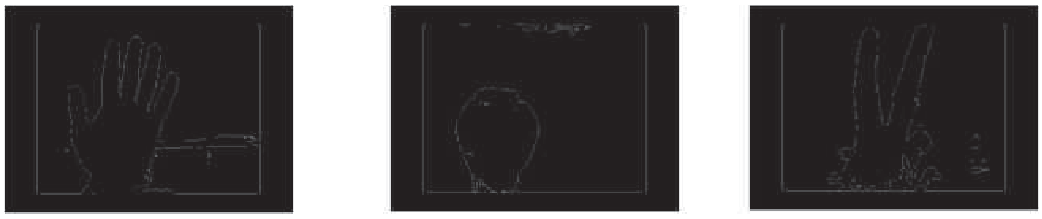


Figure 4.11: Edge for Different Hand Gesture

Gradient orientation histogram: The gradient orientation histogram can be considered as the probability distribution depends of local angles. In here, 2D situation is only considered. In the experiment, it assumes three reference angles: 80 degrees, 135 degrees and 270 degrees. So, the probability distributions (PD) for specific angle of pre-assigned hand gesture are shown in Table 4.1.

Table 4.1: Probability Distribution for Specific angle in Different Hand Gesture

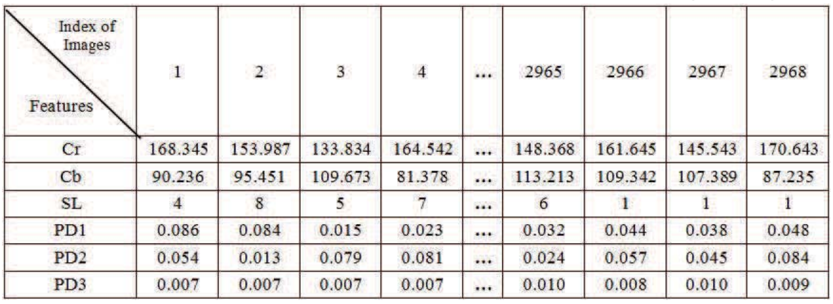
Gesture / Probability Distribution	Outstretched	Fist	Victory
PD1 = 80	0.07	0.01	0.07
PD2 = 135	0.1	0.01	0.1
PD3 = 270	0.003	0.01	0.003

③: GMM construction: after getting the three features of original images, these features are employed for GMM construction. Matlab code is (details in Appendices 2):

```
function [Weights, Mu, Variances] = GMM(Input, No_of_Clusters,Limit)
Input          = feature vectors
No_of_Clusters = 3
Limit          = 10
The results are:
Outstretched:   Weights = 0.001; Mean = 3.456.; Covariance = 13.154
Fist:           Weights = 0.002; Mean = 27.231; Covariance = 14.434
Victory:        Weights = 0.003; Mean = 63.247; Covariance = 15.235
```

④: Testing phase, 2968 hand gesture images have been collected (Sebastien Marcel Static Hand Posture Database" [100]) for our algorithm that has been described. Since the images were captured with different background, it is helpful to test the robustness of the algorithm just proposed. According to the algorithm described in training phase, the features for images can be obtained, shown from Table 4.2.

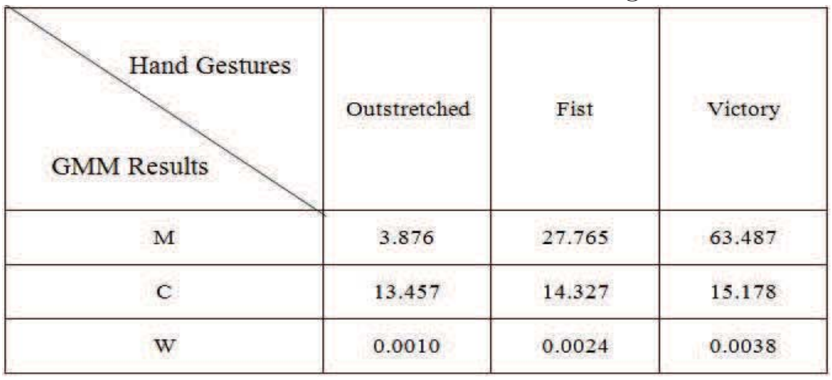
Table 4.2: Extract the Feature Value for Testing Images



Index of Images / Features	1	2	3	4	...	2965	2966	2967	2968
Cr	168.345	153.987	133.834	164.542	...	148.368	161.645	145.543	170.643
Cb	90.236	95.451	109.673	81.378	...	113.213	109.342	107.389	87.235
SL	4	8	5	7	...	6	1	1	1
PD1	0.086	0.084	0.015	0.023	...	0.032	0.044	0.038	0.048
PD2	0.054	0.013	0.079	0.081	...	0.024	0.057	0.045	0.084
PD3	0.007	0.007	0.007	0.007	...	0.010	0.008	0.010	0.009

After got the values of features for each testing image, choose 67 images as training data for outstretched, 54 images as training data for fist and 59 images as training data for victory. These three groups training data can build three feature vectors respectively. Apply these three features vectors on GMM contraction in step ③ one by one, the GMM results (training data) for each hand gesture can be obtained. The results are shown in Table 4.3.

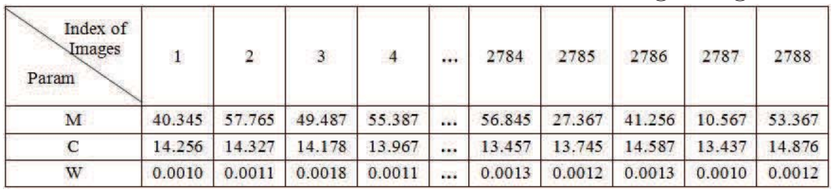
Table 4.3: GMM Results for Training Data



Hand Gestures / GMM Results	Outstretched	Fist	Victory
M	3.876	27.765	63.487
C	13.457	14.327	15.178
W	0.0010	0.0024	0.0038

In the left 2788 images, 776 images have been randomly choosen as testing images for outstretched, 1107 images have been randomly choosen as testing images for fist and the last 905 images as testing images for victory. In this testing database, each image also can get one GMM result for prepare in next step. The GMM results for each image in testing database is shown in Tabel 4.4. (Note: in Table 4.3, 4.4, 'Param' is parameters, M is mean, C is covariance, W is weight.)

Table 4.4: The Results of GMM for Testing Images



Index of Images / Param	1	2	3	4	...	2784	2785	2786	2787	2788
M	40.345	57.765	49.487	55.387	...	56.845	27.367	41.256	10.567	53.367
C	14.256	14.327	14.178	13.967	...	13.457	13.745	14.587	13.437	14.876
W	0.0010	0.0011	0.0018	0.0011	...	0.0013	0.0012	0.0013	0.0010	0.0012

⑤: Classification, Apply EM to classify, the Matlab code is (details in Appendices 2):

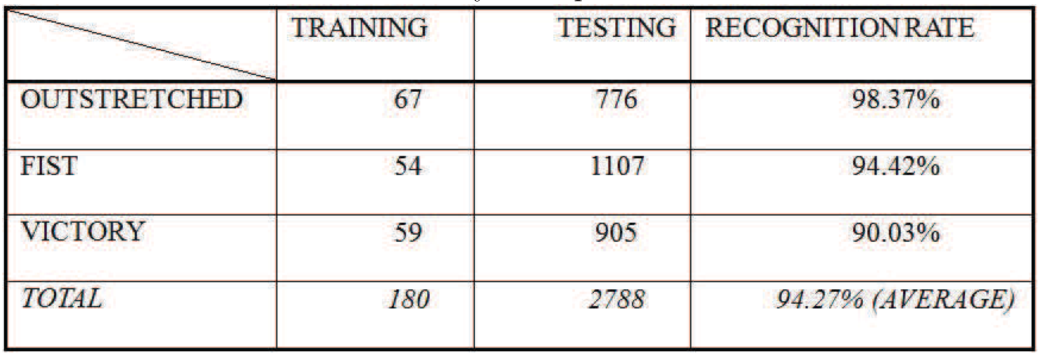
```
function [W,M,V,L] = EM_GM_fast(X,k,ltol,maxiter,pflag,Init)
X(n,d) - input data, n = number of observations, d = dimension of variable
k - maximum number of Gaussian components allowed
ltol - percentage of the log likelihood difference
           between 2 iterations ([] for none)
maxiter - maximum number of iteration allowed ([] for none)
pflag - 1 for plotting GM for 1D or 2D cases only,
        0 otherwise ([] for none)
Init - structure of initial W, M, V: Init.W, Init.M, Init.V ([] for none)
```

According this experiment, the label in EM function can be set as:

$$n = 3, d = 776 \text{ or } 1107 \text{ or } 905, k = 3,$$
$$Init = \text{results of GMM for training images}$$

The results are shown in Table 4.4

Table 4.5: Summary of Experimental Results



	TRAINING	TESTING	RECOGNITION RATE
OUTSTRETCHED	67	776	98.37%
FIST	54	1107	94.42%
VICTORY	59	905	90.03%
TOTAL	*180*	*2788*	*94.27% (AVERAGE)*

From Table 4.4, the recognition rate of victory is the lowest, because the shape of victory is between the outstretched and the fist, especially the bottom of victory, it is difficult for classifier to identify it. However, the outstretched and the fist have clearly characteristic, for example, the outstretched could be easily detected gradient orientation histogram in specific directions, and the fist could be seen no straight line in it. Therefore, it is necessary to find the others significant features to improve the performance of this experiment

4.6 Summary

In this chapter, a novel dynamic simulation scheme is presented for interactive image processing, which contains two main steps, i.e. hand motion detection and hand gesture recognition.

For hand motion detection, the movement of hand motion is detected by using image differencing. With a predefined threshold, typical movement of hand motion can be successfully identified with moving regions being segmented.

For hand gesture recognition, the combination of several low-level to middle-level features is adopted, using colour, shape and motion measurements. It is found that the gradient orientation histogram plays important roles for the recognition of the gestures. In addition, using the Gaussian mixture model as a classifier and the Expectation-Maximization algorithm to calculate the maximum likelihood between the testing image and samples of each type of hand gestures, the parameters of the GMM can be adaptively adjusted to achieve better performance.

Chapter 5

High Level Image Processing

5.1 Introduction

Machine learning is one core technique in high level image processing, only after this step, human can understand the information from raw images.

Unfortunately, machine learning is still a bottle neck, so many factors can impact the results output from machine learning algorithms.

Like described previously, in this chapter, computer-aided mammogram analysis will be used as the major application case to illustrate the effectiveness of our approaches in machine learning towards high-level image processing. In this project, two methods of machine learning are emphasized: Neural Network and SVM.

5.2 Neural Network

The concept of Neural Network (NN) was introduced when McCulloch and Pitts (MP) published their well-known thesis in 1943 [109] and at the same time started a completely new era within computer research and artificial intelligence [110]. It has been proposed that there exists a possible a way to construct a NN using mathematical functions. Their approach is based on an interconnecting set of binary decision units (BDNs), and it leads to a network which is capable of computer learning tasks. Later in 1962, Rosenblatt showed how to train a network of BDNs [111] and proposed that by changing the strengths of the connection between neurons, a NN may give a wrong answer. However, the network sequentially reaches the correct answer after training it for several times. This irony has been a demanding area of NN on which the researched today is focused.

The NNs have been utilized and applied on several areas such as pattern recognition and classification problems. A major NN application has been developed to distinguish the eleven vowels from each other using features [112]. Regarding the task as classification, the data set to be used involves more than ten thousands samples and is considered as a linearly non-separable data set. The efficiency of the NN is compared to former methods and more accurate results are obtained.

5.2.1 Neurons

There are two types of neurons (nodes) that can be utilized in NN structures. They are multilayer perceptron (MP) neuron and radial-basis function (RBF) neuron. In my work, MP neuron is utilized, and the reasons of that they are selected rather than RBF neurons will explained in details later of this chapter. The MP neuron is depicted in Figure 5.1 [113].

Let there exist $u_1, u_1, \cdots, u_J$ inputs and weights $\omega_1, \omega_2, \cdots, \omega_J$ for a given MP neuron. Each input is multiplied with the corresponding weight, the multiplications are summed and a bias term is added which leads to k (see Figure 5.1).

Finally, an activation function (AF) is applied to k. The aim of the AF

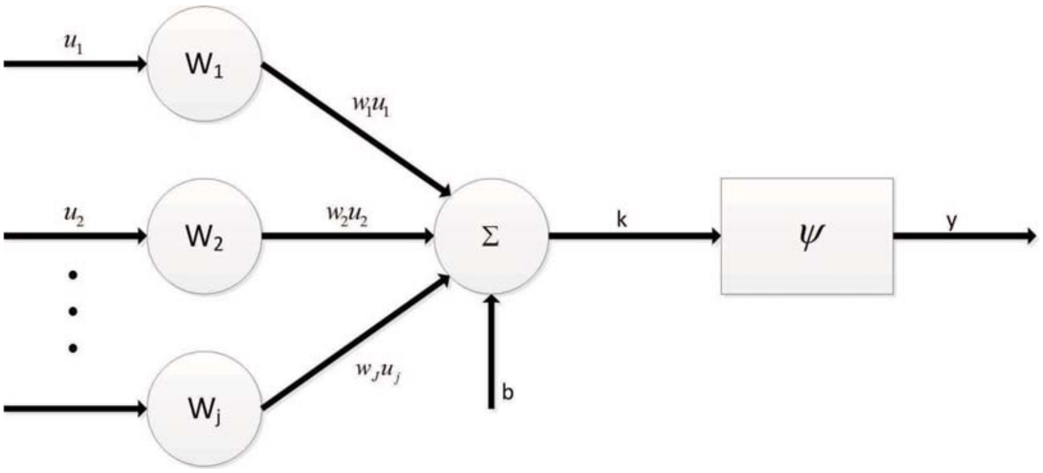


Figure 5.1: General Structure of MP Neurons

here is squashing unbounded inputs into a range. The most common AFs, are sigmoid and hyperbolic tangent and their ranges can respectively be written as follows:

$$\psi(k) = \frac{1}{1+\exp(-k)} \qquad 0 \leq \psi(k) \leq 1 \tag{5.1}$$

$$\psi(k) = \tanh(k) \qquad -1 \leq \psi(k) \leq 0 \tag{5.2}$$

Therefore, output of a neuron with J inputs and connections, a bias and an activation function $\psi(\bullet)$ can be written as follows:

$$y = \psi(\sum_{i=1}^{J} u_i\omega_i + b) \tag{5.3}$$

5.2.2 Network Architectures

There are three most common network architectures, i.e. single-layer (SL) N-N, multi-layer perception (MLP) NN and RBF NN [114]. A single-layer (SL) NN has only input and output layers and is only used for linearly separable cases where there is no statistical complexity.

RBF NNs involve the concept of transforming the input data to a high dimensional space in a nonlinear manner. In our project, MLP NN is utilized to deal with our data of more than 39 dimensions.

The MLP NN involves one or more layers between input and output nodes which are called hidden layers. The function of hidden neurons is to arbitrate between the input and output nodes in a non-linear manner.

In linearly inseparable cases, an MLP NN with sufficient hidden neuron-s can represent a continuous function (Haykin, 1999, [115]). A MLP NN constructed with m inputs, t_1 and t_2 neurons in the first and second hidden layer, respectively, and $m - t_1 - t_2 - c$ outputs is referred as a MLP NN.

For example, Figure 5.2 presents a 3-4-2-1 MLP NN with 3 inputs and 1 output respectively where f, g and h represent AFs and $w_{j,k}$ are the connect-ing weights between layers.

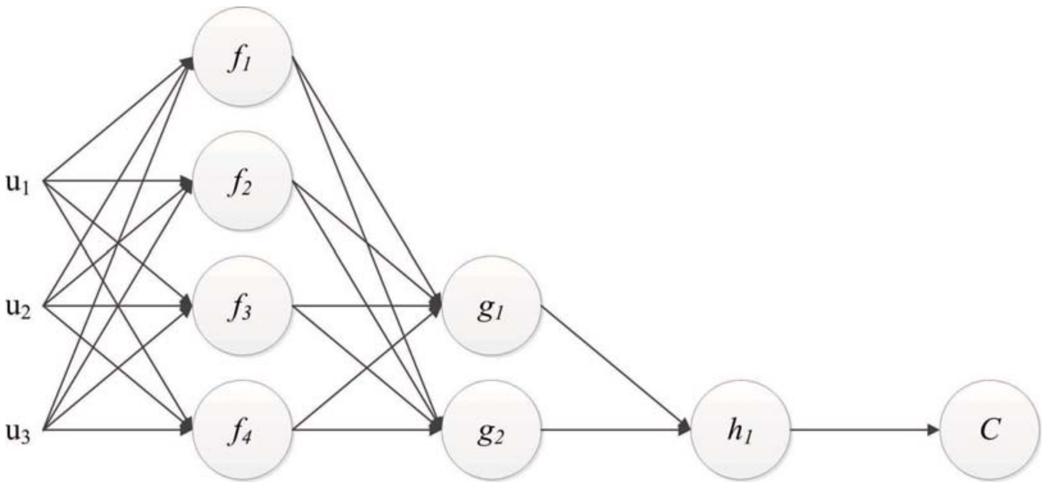


Figure 5.2: General Structure of MP Neurons

5.2.3 Supervised Learning

The concept of learning with a teacher can be referred as supervised learning [116]. The term 'teacher' means having knowledge about the environment represented by a set of input-output examples. Thus, training the NN with this set leads to accurate weights for testing the rest of the data.

To decide which weight is more accurate amongst others, a training error which is typically mean-squared error can be written as follows:

$$MSE = \frac{1}{J}\sum_{i=1}^{J}(t_i - y_i)^T(t_i - y_i) \tag{5.4}$$

where $t = [t_1, t_2, \ldots, t_J]$ is the desired output of the samples, also called target and $y = [y_1, y_2, \ldots, y_J]$ is the estimated output.

Optimized MSE is achieved by updating the weights of the NN in every iteration (epoch) until y cannot be reduced or the limit of the iteration count is reached which is determined by the analyzer.

5.2.4 Generalization Methods

Generalization methods aim to reduce the complexity of the NN by optimizing the weights and number of neurons while maximizing the accuracy of the model. Bayesian regularization is one of the most effective methods in NN optimization.

In MATLAB, it is referred as a training algorithm, namely trainbr. It is a modified version of performance function MSE. Using F to represent sum of squares of the weights in the network,

$$F = \frac{1}{n}\sum_{j=1}^{n} w_j^2$$

combined with MSE(5.4), then the regular MSE(MSE_{REG}) can be written as follows: [117]

$$MSE_{REG} = \gamma MSE + (1-\gamma)F \tag{5.5}$$

where γ is the learning rate.

Using this performance function forces the network to use smaller weights and biases which leads to smooth network response and small risk of overfitting. The selection of the γ value is a challenge for the bias rate approach. It is assumed in (MacKay, 1992, [118]) that the weights and the biases in a network are random variables with specified distributions, hence, selection of the γ value is handled with an automated approach. The details about it can be found in (Foresee and Hagan, 1997, [119])

5.2.5 Training Algorithms

The Training Algorithms, also well known as Supervised Learning Algorithms, are widely used for the great majority of MLP architectures, e.g. SL NN, MLP NN and RBF NN. The most decent training algorithm for SL NN is Rosenblatt's perceptron training algorithm (RP) which can converge to the global minimum when the data is linearly separable.

The algorithm can be described as follows [109]:

1. Consider a given training set $X = \{x_i, \theta_i\}, i = 1, \ldots, J$ where $x = \{x_1, x_2, \ldots, x_n\}$ and θ is the output;
2. Initialize the weights series, ω, (either to zero or to a small random values);
3. Pick a learning rate γ, $(0 < \gamma < 1)$;
4. For the step p, obtain y_p, in respect to (5.3), where $w = [w_1, w_2, \ldots, w_n]$ and $\psi = sign(\bullet)$;
5. Calculate output error, denoted as $e_p = \theta_p - y_p$. If $e_p = 0$ then terminate

the procedure, otherwise update w and b in a way as described below:

$$\omega_{p+1} = \omega_p + \nabla\omega_p, \quad \nabla\omega_p = \gamma e_p x_p \tag{5.6}$$

$$b_{p+1} = b_p + \nabla b_p, \qquad \nabla b_p = \gamma e_p \tag{5.7}$$

until $e_p = 0$

Various training algorithms have been devised for MLP NN. The nine most popular have been implemented with NN structure in the MATLAB environment with names as follows:

1. *traindg* — Batch Gradient Descent, standard back-propagation (BP) algorithm
2. *traindgm* — Batch Gradient Descent with momentum, BP with momentum
3. *traindga* — Variable Learning Rate, adaptive learning rate attempts to keep the learning step size as large as possible while keeping learning stable.
4. *traindgx* — Variable Learning rate with momentum
5. *trainrp* — Resilient BP, speeds up the BP process by focusing on the sign of the activation function derivatives
6. *traincgf* — Fletcher-Reeves Conjugate Gradient Algorithm, search by zig-zagging towards to minimum MSE. Usually, the method is much faster than above algorithms

7. *trainscg* — Moller Scaled Conjugate Gradient, combines the model-trust region approach with conjugate gradient algorithm

8. *trainbfg* — Broyden, Fletcher, Goldfarb, Shanno Algorithm, achieves faster convergence using quasi-Newton method. An approximate Hessian of size is stored, where is the number of the parameters in the network.

9. *trainlm* — Levenberg-Marquardt Algorithm, achieves faster convergence by approximating Newton's method by a Jacobian matrix. Memory requirement increases with parallel to the number of network parameters and the size of training data.

In this project, trainlm will be applied to train the neural network, and details of the training algorithm are given as follows. The LM (trainlm) training algorithm is a member of a class using hill-climbing optimization techniques (Yuret and Maza, 1993, [120]). The algorithm is an iterative method that locates the minimum value of a multivariable function that is expressed as the sum of squares of non-linear functions (Levenberg, 1944 [122], Marquardt, 1963 [121]). It has become a standard technique for non-linear function least squares problems. LM can be thought of as a combination of steepest descent and the Gauss-Newton method. When the current solution is far from the correct one, the algorithm behaves like a steepest descent method: slow, but guaranteed to converge. When the current solution is close to the correct solution, it becomes a Gauss-Newton method. The algorithm considers the NN as a linear separable case and begins the computation until MSE cannot be reduced. Then the algorithm converts itself to a non-linear solver and training terminates when the MSE cannot be reduced for the second time or by a limitation e.g. maximum iteration count, reaching an optimum MES that is defined beforehand. The method avoids calculating the Hessian matrix, the Jacobian matrix is used for this algorithm, instead. The proposed Levenberg and Marquardt iterative scheme is below:

$$\omega_{p+1} = \omega_p - (J^T J + \gamma I)^{-1} J^T MSE \tag{5.8}$$

where J is the Jacobian matrix and γ is the damping parameter (Nielsen, 1999 [123]). Thus, the complexity of the algorithm is dependent on the size of J which considers size of the network and number of training pairs as its elements. Therefore, algorithm speed reduces when the amount of data increases.

5.3 Feature Extraction and Selection

5.3.1 Data Set

The data set used in my project is from the University of South Florida where all images are of size 2048×2048 pixels. From these images, 748 typical samples are extracted including 633 benign (normal) cases and 115 malignant (abnormal) cases. Benign samples are represented by target 1, and malignant samples are represented by target 0.

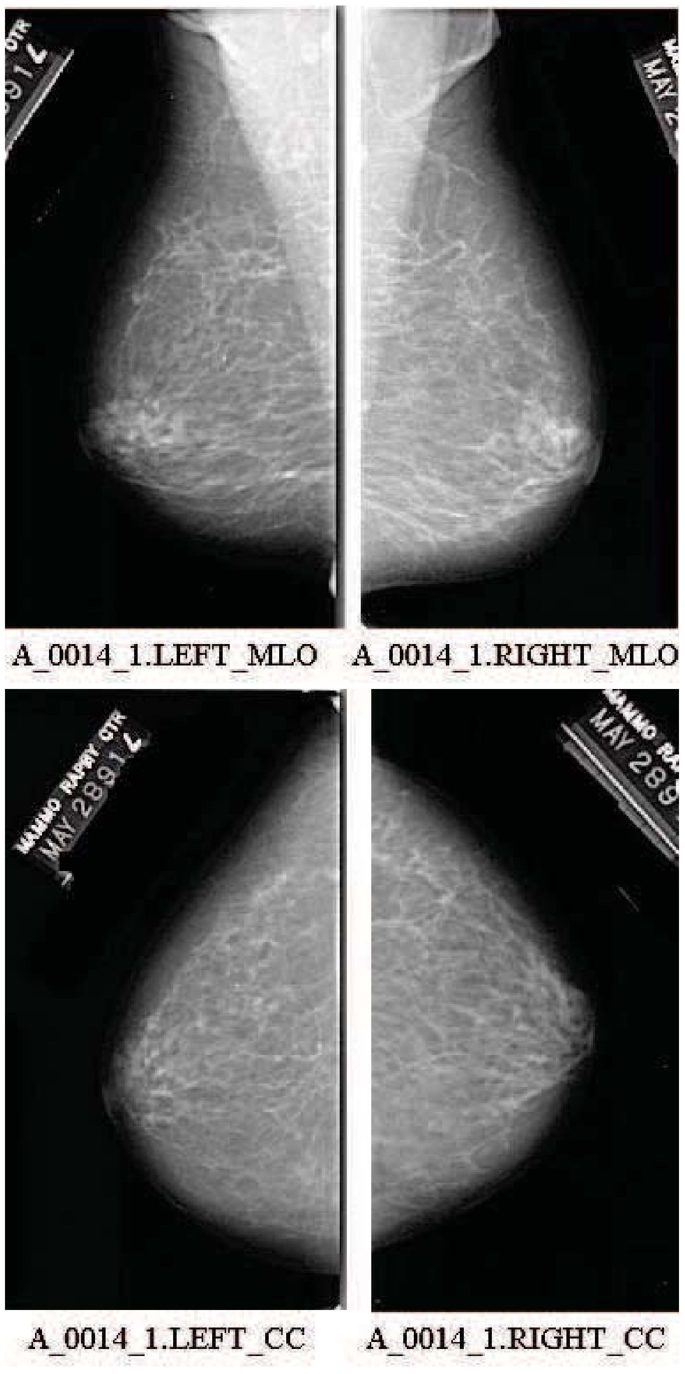


Figure 5.3: [part 1] Normal Case (case A_0014_1)

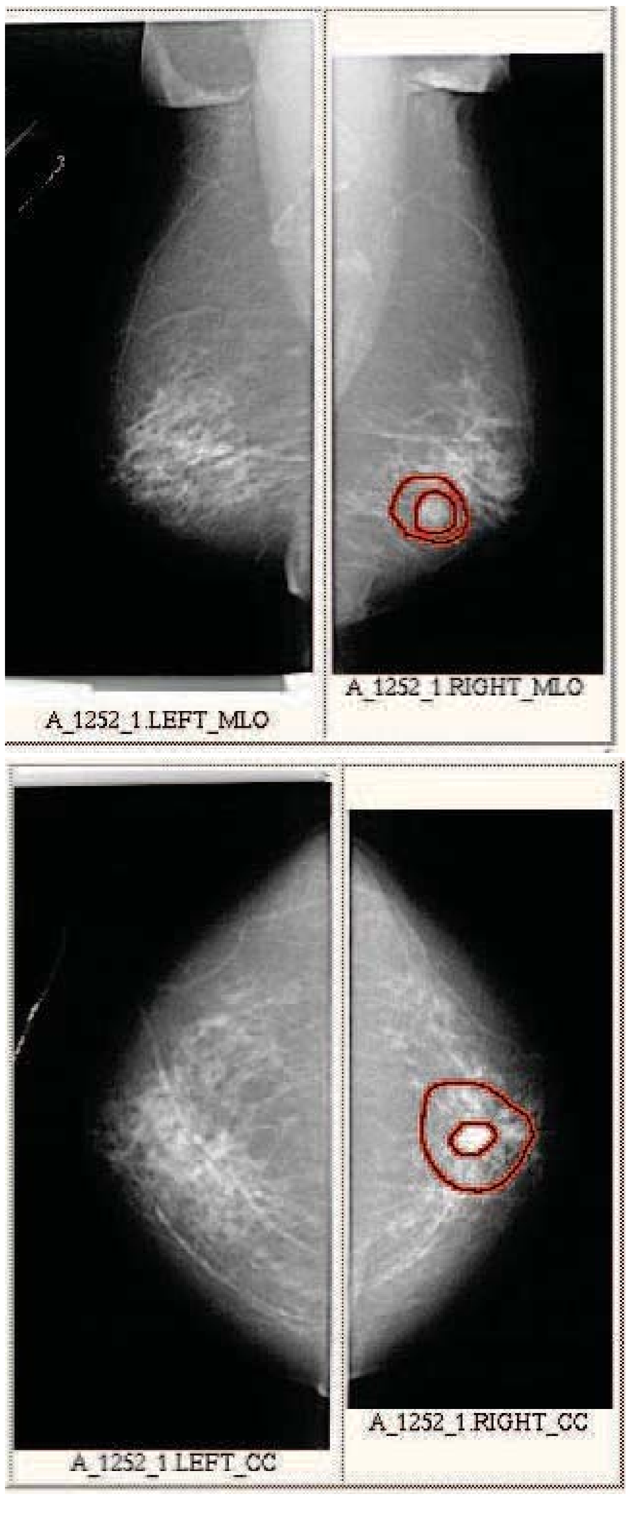


Figure 5.4: [part 2] Cancer Case (case A_1252_1)

Examples of both normal and abnormal samples are shown in Figure 5.3 and Figure 5.4 for information. In Figure 5.4, cancer regions are manually labelled using circular marks. It can be seen from the visual information, that the differences between these normal and abnormal cases are insignificant, especially in comparing the left image in Figure 5.3 with the images in Figure 5.4, which is the main challenge of the problem. Feature extraction is used to seek for representative measures to distinguish abnormal cases from normal ones.

5.3.2 Feature Extraction

In my project, in total there are 39 features employed for the detection of breast cancer, and the definitions and descriptions of them are given in Table 5.1. Except in the first, second and the last rows, the remaining 36 features correspond to 18 groups and each contains mean and standard deviation of one specified measurement [124, 125].

One of the primary tasks in my project is to select a group of most representative features for more effective classification. To achieve this, these 39 features need to be analyzed, and details of some of these features are presented below.

Table 5.1: Definitions and Descriptions of the 39 Features Used in This Thesis (*$L(\theta, l)$ be a string of pixels in direction θ and of length l)

No	Features for a Suspicious Region	Features for Cluster
1	The age of the patient	
2	The number of suspicious regions in a cluster	
3	The area of a suspicious region	Mean
4		Standard Mean
5	The compactness of a suspicious region	Mean
6		Standard Mean
7	The Measure Fourier descriptor FF of a suspicious region	Mean
8		Standard Mean
9	The Moment-based measure M of a suspicious region	Mean
10		Standard Mean
11	The Eccentricity of a suspicious region	Mean
12		Standard Mean
13	The Spread of a suspicious region	Mean
14		Standard Mean
15	The average minimum standard deviation of $L(\theta, l)^*$ in a suspicious	Mean
16	region	Standard Mean
17	The average standard deviation of the minimum standard deviation of	Mean
18	$L(\theta, l)^*$ at the different directions in a suspicious region	Standard Mean
19	The average standard deviation of the string of length l, starting from	Mean
20	each point in a suspicious region and at direction θ	Standard Mean
21	The average gradient of a suspicious region	Mean
22		Standard Mean
23	The average difference of a suspicious region	Mean
24		Standard Mean
25	The average brightness of a suspicious region	Mean
26		Standard Mean
27	The largest intensity in a suspicious region	Mean
28		Standard Mean
29	The average brightness in the context of a suspicious region	Mean
30		Standard Mean
31	The standard deviation of the brightness in the context of a suspicious	Mean
32	region	Standard Mean
33	The difference of the average brightness in a suspicious region and its	Mean
34	context	Standard Mean
35	The difference between the standard deviation of brightness in a	Mean
36	suspicious region and standard deviation of the brightness in its context	Standard Mean
37	The texture difference between microcalcifications and their context	Mean
38		Standard Mean
39	The average minimum distance between suspicious regions	

1). The Fourier descriptors were defined as the Fourier coefficients of boundary pixels. They are used for shape description of the objects extracted from input images [126-128]. Let the complex array $z_0, z_1, z_2, \ldots, z_{N-1}$ represents the boundary [129] belonging to the object whose shape needs to described. The k-th Fourier transform coefficient is calculated as

$$Z_k = \sum_{n=0}^{N-1} z_n e^{-2\pi k_n/N}, \quad k = 0, 1, \ldots, N-1$$

The Fourier descriptors are obtained from the sequence Z_k by truncating elements Z_0 and Z_1, then by taking the absolute value of the remaining elements and dividing every element of the thusly obtained array by $|Z_1|$. To summarize, the Fourier descriptors are

$$C_{k-2} = |Z_k|/|Z_1|, \quad k = 2, 3, \ldots, N-1$$

2). Moment-based measure [130] If the coordinates of the N pixels of a segmented calcification contour are described by an ordered set $z(i) = (x_i, y_i), i = 1, 2, \ldots, N$, the Euclidean distances $z(i)$ of the vectors connecting the centroid of the segmented object and the ordered set of contour pixels form a one-dimensional representation of the contour. The p^{th} moment can then be defined as [131]

$$m_p = \frac{1}{N}\sum_{i=1}^{N}[z(i)]^p$$

and p^{th} central moment as:

$$u_p = \frac{1}{N}\sum_{i=1}^{N}[z(i) - m_1]^p$$

3). Eccentricity (ε) measures the degree to which an object's mass is concentrated along a particular axis. The range of values for ε is [0-1] where 0 defines a circular object and 1 a liner object. It can be represented as [132]:

$$\varepsilon = \frac{(m_{2,0} - m_{0,2})^2 + 4m_{1,1}^2}{(m_{2,0} + m_{0,2})^2}$$

where, for an image $f(x, y)$, the moment of order $p + q$ was defined as:

$$m_{pq} = \sum_x \sum_y x^p y^q f(x, y)$$

4). Standard deviation In probability and statistics, the standard deviation is a measure of the dispersion of a collection of values [111].

The standard deviation of a probability distribution is the same as that of a random variable having that distribution. The standard deviation σ of a real-valued random variable X is defined as:

$$\begin{aligned} \sigma &= \sqrt{E((X - E(X))^2)} \\ &= \sqrt{E(X^2) - E^2(X)} \\ &= \sqrt{E(X^2) - \mu^2} \end{aligned} \tag{5.9}$$

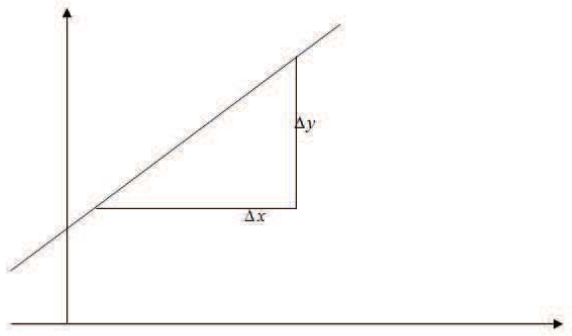
Where $\mu = E(X)$ refers to the expected value of X (another word for the mean).

Continuous distributions usually give a formula for calculating the standard deviation as a function of the parameters of the distribution. In general, the standard deviation of a continuous real-valued random variable X with probability density function $p(x)$ is:

$$\sigma^2 = \int (x-\mu)^2 p(x)dx$$

where $\mu = \int xp(x)dx$ and the integrals are definite integrals taken for x ranging over the range of X.

6). Average Gradient A measure of contrast in a photographic image, expressed as the slope of a straight line joining two density points on the sensitometric curve. The slope of a line in the plane containing the x and y



axes is generally represented by the letter m, and is defined as the change in the y coordinate divided by the corresponding change in the x coordinate, between two distinct points on the line. This is described by the following equation: $m = \Delta y/\Delta x$

5.3.3 Feature Analysis and Selection

Selection of the most representative features is based on the fact that these selected features should be most discriminative in distinguishing normal and abnormal cases.

In general, there are two general techniques for this purpose, i.e. PCA based and classifier-based. PCA, or principal component analysis, is a vector space transform used to reduce multidimensional data sets to lowers dimensions for easy analysis.

In PCA analysis, assumptions on linearity and statistical important of mean and covariance are emphasized, in which large variance usually indicates important dynamics. This has been implemented in many clustering methods where large inter-class distance is expected and taken as a kind of overall covariance, and details of which is described in Section 5.4.2.

As for classifier-based approaches, each separate feature is used as input for the classifier to compare their performance. Those with higher detection rates are then selected as more representative features [134]. In my project, the neural classifier is used to test the priority of all our 39 features, and details of this are discussed in Section 5.4.1.

5.3.4 Results and Discussion

5.3.4.1 Feature Selection Using a Neural Network

A back propagation neural network (BP) [134], which contains an input layer, one hidden layer and one output layer, as shown in Figure 5.5, is employed in my project. The features inputted can be single ones or combined ones and through changing hidden units, learning rates and momentums the best detection results can be achieved. The output value is normalized within [0, 1]. Those larger than 0.5 are taken as benign samples and those less than 0.5 are taken as malignant ones. For example, the output of 0.1 is very likely to be malignant, and 0.9 to be benign.

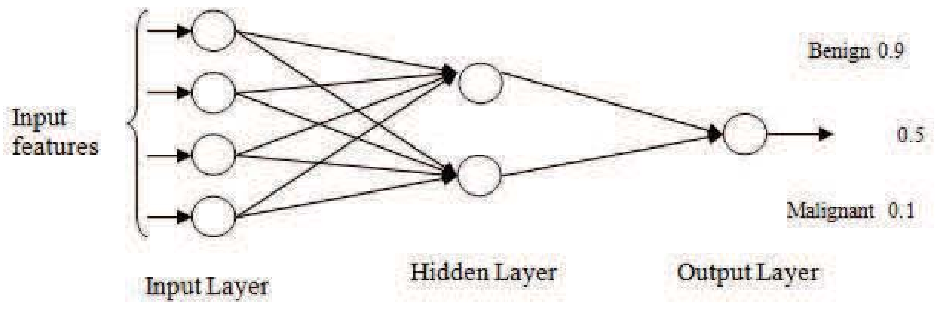


Figure 5.5: Outline of Neural Network Structure Used for Feature Selection

As mentioned above, each of the 39 features is applied as input to the designed neural classifier to determine its performance. At this stage, 90% of the test samples chosen to be training data and the testing data are the remaining 10%.

As mentioned above, in total there are 748 samples in my data set in which 633 are benign and 115 are malignant. As a result, 63 benign samples and 11 malignant samples are selected for testing as 10% of the whole data. The basic criterion here is to correctly detect as many malignant samples as possible, and of course, to reduce the occurrence of wrongly detected malignant samples.

Table 5.2: List of Features and Their Discriminative Ability in Classification Using the NN

Index of Features	Correct Rate	Index of Features	Correct Rate	Index of Features	Correct Rate
15	0.2333	20	0.0929	25	0.0491
1	0.1912	7	0.0877	24	0.0456
2	0.1877	29	0.0842	3	0.0456
16	0.1666	14	0.0842	4	0.0438
39	0.1526	12	0.0701	37	0.0438
18	0.1491	21	0.0666	26	0.0403
5	0.1473	30	0.0666	32	0.0385
17	0.1421	8	0.0631	36	0.0385
31	0.1403	35	0.0543	13	0.0385
34	0.1368	38	0.0543	10	0.0368
33	0.1280	22	0.0526	11	0.0350
23	0.1210	27	0.0526	9	0.0350
19	0.1035	28	0.0491	6	0.0315

Table 5.2 lists features and their discriminative ability in the neural classifier and these rates also shown in Figure 5.6 as a histogram for easy comparison.

From Table 5.2 and Figure 5.6, discriminative ability differs much over all features where the maximum and the minimum ones are 0.2333 and 0.0315,

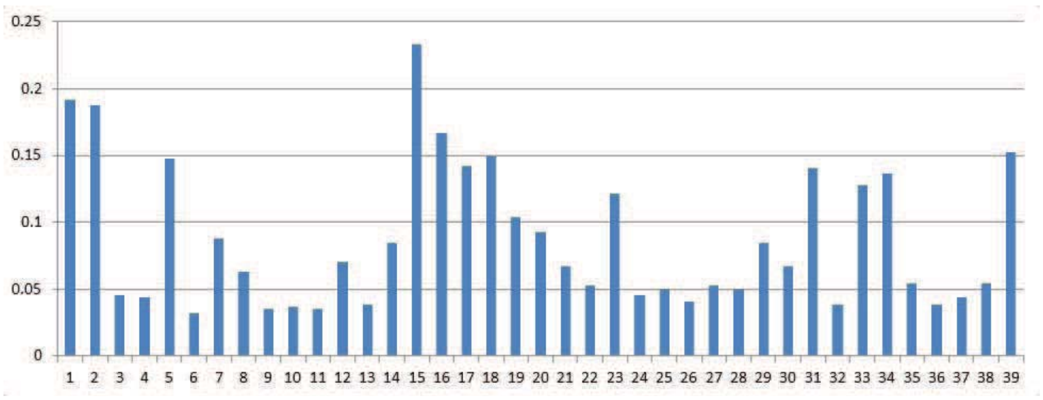


Figure 5.6: Histogram of Discriminative Ability of All Features from Neural Network

respectively. In my experiments, features of higher discriminative ability are selected with first priority for further training and testing. Under different numbers of selected features, the classification results are compared in Table 5.3 and also plotted in Figure 5.7.

Table 5.3: Classification Results Using Different Number of Selected Features From the Same Neural Classifier

Numbers of Selected Features	Correcting Rate of Class 1	Correcting Rate of Class 0	Overall Correcting rate
5	89.54%	76.19%	78.17%
10	85.90%	77.38%	78.64%
15	89.36%	80.15%	81.51%
20	82.45%	81.34%	81.50%
25	83.44%	75.43%	76.62%
30	85.78%	78.44%	79.53%
35	75.63%	77.45%	77.17%
39	78.13%	83.71%	82.88%

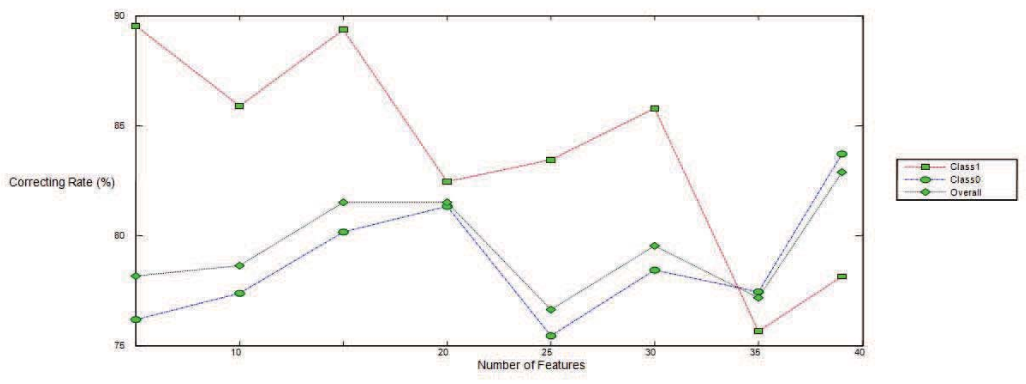


Figure 5.7: Plot of Table 5.3

The correct classification rates is defined as follows:

$$\text{Correct Rate of Class 1} = \frac{\text{Detected numbers of abnormal samples (DA)}}{\text{Sum of abnormal samples (SA)}}$$

$$\text{Correct Rate of Class 0} = \frac{\text{Detected numbers of normal samples (DN)}}{\text{Sum of normal samples (SN)}}$$

$$\text{Correct Rate of Overall} = \frac{\text{DA+DN}}{SA + SN}$$

From Table 5.3 and also Figure 5.7 it can be seen that increasing the number of features does not necessary improve the classification results of both normal and abnormal cases. In other words, there is no direct link to an optimal number of features used to achieve the best performance. Consequently, an improved classifier is necessary and this is discussed in the next section.

5.3.4.2 Feature Selection by using PCA based Clustering Rules

In probability and statistics, the standard deviation of a probability distribution, random variable, or population of values is a measure of the spread of its values. The standard deviation is usually denoted by the letter σ (lower case sigma), and is defined as the square root of the variance.

As for variance, as discussed in Section 5.3.3, large or small variance values refer to important or unimportant dynamics in PCA analysis. To this end, standard deviation can be also considered as a kind of measurement of dynamics.

To state it more formally, the standard deviation is the root mean square (RMS) deviation of values from their arithmetic mean. Consequently, mean and standard deviation is used for feature selection, especially for problem of classification. Before choosing appropriate features, several steps will be applied to measure the importance of each feature as follows.

1. The mean value of each feature is calculated as mean (μ) for every single feature. Instead of calculating the value over all samples, two mean values determined as μ_1 and μ_0, which correspond to samples in target 1 group and target 0 group, respectively. In other words, all the samples are divided into two groups and two mean values are extracted for the samples in each group.

2. For each feature, calculate σ_1 and σ_0 for two groups of its samples corresponding to target 1 and target 0;

3. Based on extracted measurements of mean and standard derivation, an overall rank of each feature is defined below:

$$rank(\bullet) = \frac{|\mu_1 - \mu_0|}{\max(\sigma_1, \sigma_0)} \tag{5.10}$$

The relationships among $\mu_1, \mu_0, \sigma_1, \sigma_0$ are illustrated in Figure 5.8 below, in which the two groups of samples are represented by hollow circles and circles enclosing crosses, respectively. The centroids in each group are denoted μ_1 and μ_0, and parameters σ_1 and σ_0 are used to denote the compactness of samples in each group. The basic criterion in ranking the features is to measure if it is easily to separate two groups of samples.

If the distance between two centriods are far enough, say larger than the larger item of σ_1 and σ_0, it will be ranked high as the samples can be more clearly split into two groups.

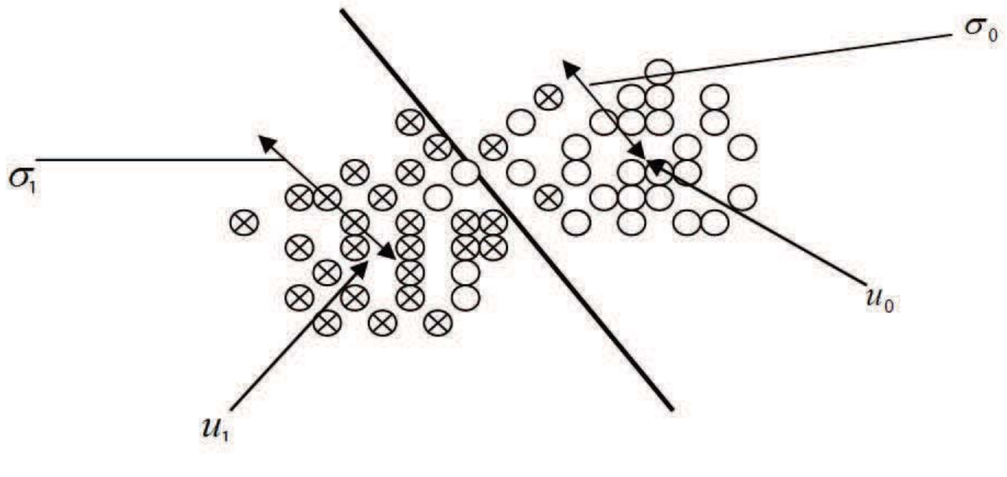


Figure 5.8: An illustration of the Simply Constructed Relationships Among μ_1, μ_0, σ_1 and σ_0

In definition, equation (5.9), $|\mu_1 - \mu_0|$ represents the distance between the centroids of two different groups. Therefore, larger $|\mu_1 - \mu_0|$ means the two groups can be more easily separated. On the other hand, smaller σ means the samples are more compact and also are more easily classified. Using this ranking function, the ranks for each of the 39 features are determined as given in Table 5.4. For easy comparison, the contents in Table 5.4 are also plotted in Figure 5.9 as a histogram to show the ranks of each feature.

Table 5.4: List of Ranks Corresponding to Each of the 39 Features

Index of Features	Rank $\frac{|\mu_1 - \mu_0|}{\max(\sigma_1 - \sigma_0)}$	Index of Features	Rank $\frac{|\mu_1 - \mu_0|}{\max(\sigma_1 - \sigma_0)}$	Index of Features	Rank $\frac{|\mu_1 - \mu_0|}{\max(\sigma_1 - \sigma_0)}$
39	0.9743	34	0.3797	35	0.1306
15	0.8506	22	0.3386	12	0.1073
2	0.7187	23	0.3384	14	0.1003
16	0.7026	20	0.3104	8	0.0765
17	0.6867	31	0.2931	1	0.0655
33	0.6627	30	0.2812	37	0.0652
27	0.6252	26	0.2623	11	0.0608
25	0.6109	28	0.2321	36	0.0573
21	0.5856	13	0.2072	32	0.0433
29	0.5286	5	0.1963	4	0.0405
7	0.4510	3	0.1765	10	0.0390
19	0.4192	38	0.1572	9	0.0287
18	0.3874	24	0.1501	6	0.0046

Comparing Figure 5.9 with Figure 5.6, it is easily seen that features assigned large ranks in Figure 5.9 usually have been found to have high discriminative ability derived from neural classifier. In other words, the two methods, NN and PCA, have generated consistent results in feature selection. According to the ranking results, again, some features have been chosen

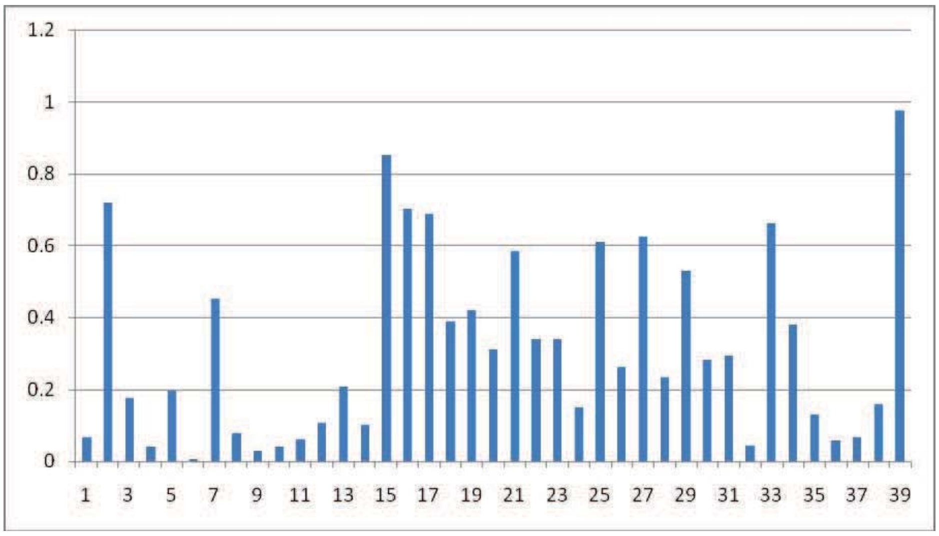


Figure 5.9: Plot of Table 5.4 as a Histogram to Show the Ranks of Each Feature

for classification with features of high rank values are to be selected first. The same neural classifier is applied in my test where 90% of the samples are used for training and 10% for testing. The testing results are reported in Table 5.5 and also plotted in Figure 5.10 for clear presentation and comparisons.

Table 5.5: Classification Results Using Feature Selected by PCA Analysis

Numbers of	Correct Rate		
Selected Features	Class 1	Class 0	Overall
5	77.11%	66.66%	68.33%
10	74.99%	74.99%	74.99%
15	61.35%	86.50%	82.52%
20	83.68%	81.34%	81.77%
25	45.45%	88.88%	82.02%
30	52.26%	85.71%	80.43%
35	36.36%	98.41%	88.61%
39	51.52%	88.71%	82.43%

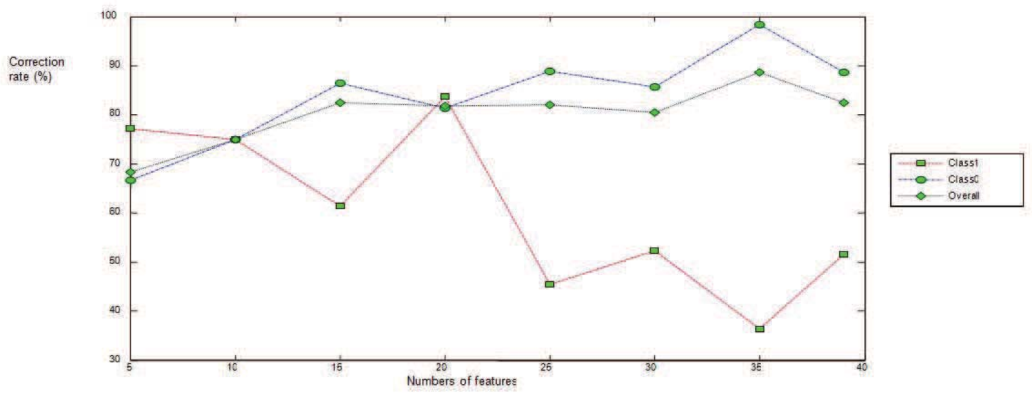


Figure 5.10: Plot of the results in Table 5.5

5.4 Compare the Performance Between ANN and SVM

5.4.1 Review of SVM and ANN Learning Techniques

In this chapter, the classification of MCCs is treated as a two-class pattern classification problem, and the two classes are referred to as "malignant" and "benign". If denote $x \in \Re^d$ as an input vector or pattern to be classified, and let scalar denote its class label, i.e. $y \in \{-1, 1\}$ for SVM and $y \in \{0, 1\}$ for ANN. The training set L contains M samples, i.e. $L = \{(x_i, y_i)\}$ and $i \in [1, M]$. The problem here is how to determine a classifier $f(x)$ which can make correct decision and classify the input pattern into suitable classes. In this section, brief introductions to SVM and ANN are presented, which forms the base of our proposed improved classifier as presented in the next section.

5.4.1.1 The SVM Classifier

In general, a SVM classifier can be formed as follows,

$$f_{SVM}(x) = w^T \phi(x) + b \tag{5.11}$$

where parameters w and b respectively denote a weight vector and a bias that can be determined in the training process through minimizing the cost function below, and $\phi(\bullet)$ refers to a nonlinear mapping to map the input vector x into a higher dimensional space for easily separated by a linear hyperplane as illustrated in Fig. 5.11.

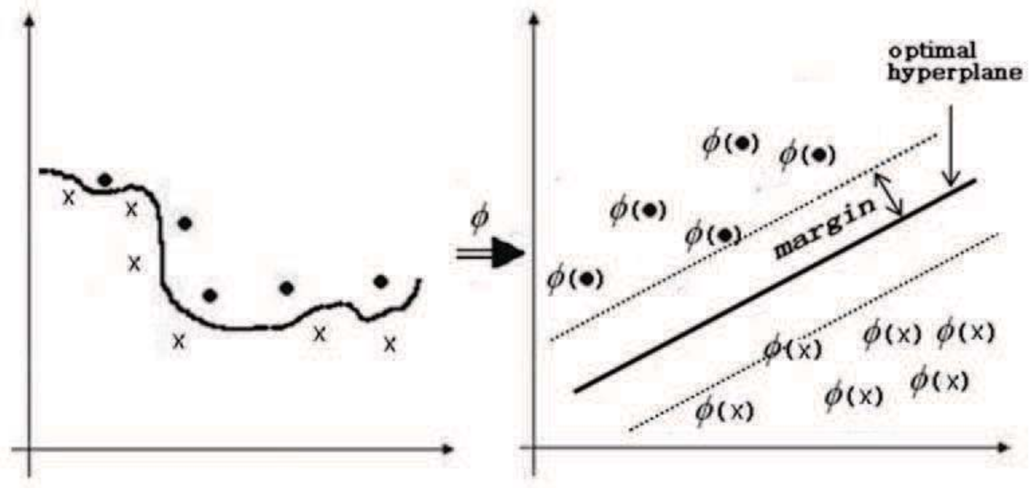


Figure 5.11: Illustration the Concept of SVM to Map a Nonlinear Problem to a Linear Separable One

A training sample (x_i, y_i) is a support vector if it holds $y_i f_{SVM}(x_i) \leq 1$. Let us denote s_k as extracted support vectors, $k \in [1, K]$, $\{s_k\} \subset L$ is a small subset of the training set. Hence, the SVM function becomes:

$$\begin{cases} f_{svm}(x) = \sum_{k=1}^{K} K(x, s_k) + b \\ K(x, s_k) = \phi^T(x)\phi(s_k) \end{cases} \tag{5.12}$$

where $K(\bullet, \bullet)$ is denoted as a kernel function to represent the effect of the nonlinear mapping $\phi(\bullet)$ in classification.

Some common used kernel functions are summarized below, including linear and two nonlinear functions. If the training samples are not linear separable, non-linear kernel functions are better choice. In addition, the associated parameters p and σ are determined automatically during the training process.

1. Linear kernel: $K(x_i, x_j) = x_i^T x_j$
2. Polynomial kernel: $K(x_i, x_j) = (x_i^T x_j + 1)^p$
3. RBF kernel: $K(x_i, x_j) = e^{-\|x_i - x_j\|^2/(2\sigma^2)}$

5.4.1.2 The ANN Classifier

Although there is no precise definition, ANN can be considered as an information processing system which is composed of a network of interconnected simple processing elements, i.e. neurons. Determined by the connections between these neurons and the associated parameters, ANN can exhibit complex global behavior to generate expected outputs via supervised or unsupervised learning.

Inspired by the biological nervous system, the learning process is to adjust the connection strength or weights between the neurons. Each neuron forms a node in the whole network and after training each node is assigned with a determined bias or threshold. For the each interconnection between two nodes, a weight is also assigned to represent the link-strength between the neurons.

For a given input vector $x = (x_1, x_2, \ldots, x_d)^T$ and weight vector $w = (w_1, w_2, \ldots, w_d)^T$, the output of a single neuron z in Fig 5.12 is determined as:

$$z = g(w^T x - b) = g(\sum_{i=1}^{d} w_i x_i - b) \tag{5.13}$$

where $g(\bullet)$ is namely an activation function to decide whether the perceptron should fire or not.

The sigmoid function $Sig(x) = (1 + e^{-x})^{-1}$ is the most popular used activation function, others include tanh and step functions, etc.

Using the same process as to compute the output of a single neuron, the output of the whole network can be also calculated in a topological manner. This means that for each neuron its inputs from other neurons need to be computed before determining its output.

As seen, the weight vector and the bias associated to each connection and each node will influence the output results, and they can be determined in training or learning process as follows.

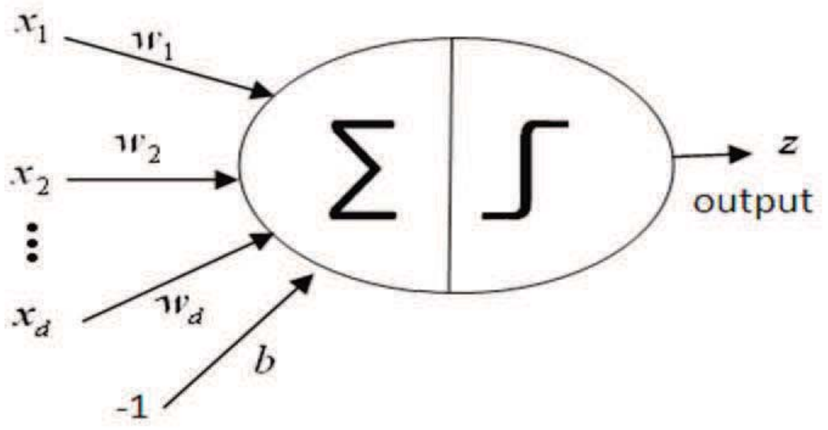


Figure 5.12: Illustration the Effect of a Single Neuron

First of all, the topology of the ANN needs to be specified, and feed-forward ANN is adopted as it has been widely applied for the classification of MCCs [40, 35, 42, 44]. A feed-forward ANN is a multi-layer perceptron (MLP) which contains three or more layers of neurons, i.e. one input layer, one output layer and at least one hidden layer.

With a given training set, a specified activation function and a learning ratio γ where $\gamma \in (0, 1)$, the learning process for supervised training using the well-known back-propagation algorithm can be described in the following three stages.

Firstly, the initial weights and bias are set randomly between $[-1, 1]$ to attain a group of outputs $z^{(t)}$ at $t = 1$ referring to the first round of iteration.

Then, an error function is decided as $\varepsilon(t) = \sum_{i=1}^{M}(y_i - z_i^{(t)})^2/2$ using the sum squared error between the estimated output z and the target output y. Finally, the error signal at the output units is propagated backwards through the whole network to update the weights using the gradient descent rule

$$\Delta w_{ij}(t) = -\gamma \frac{\partial \varepsilon(t)}{\partial w_{ij}} \tag{5.14}$$

where w_{ij} refers to a weight between the j^{th} node in a given layer and the i^{th} node in the following layer. With updated weights, $t = t + 1$ is set to start a new iteration until the network becomes convergence. This can be measured by using a small change ratio of $\varepsilon(\bullet)$ or a given number of iterations.

5.4.2 Comparisons Between SVM and ANN

As two different algorithms, SVM and ANN share the same concept using linear learning model for pattern recognition. The difference is mainly on how non-linear data is classified.

Basically, SVM utilizes nonlinear mapping to make the data linear separable, hence the kernel function is the key. However, ANN employs multi-layer connection and various activation functions to deal with nonlinear problems. In fact, single layer ANN can only generates linear boundary, and the 2nd layer can combine the linear boundary together; while at least three layers are required to produce boundary of arbitrary shapes.

Using the gradient descent learning algorithm, ANN tends to converge to local minima. As a result, it suffers from the over-fitting problem. On the other hand, SVM tends to find a global solution during the training as the model complexity has been taken into consideration as a structural risk in SVM training. In other words, ANN minimizes only the empirical risk learnt from the training samples, but SVM considers both this risk and the structural risk.

Consequently, the training results from SVM have better generalization capability than those from ANN. Therefore, SVM and ANN are two typical classifiers which are used to validate our balanced learning strategy as discussed in the next sections.

5.4.3 Balanced Learning

Despite the good generalization capability of SVM achieved for pattern recognition, the performance on classification of MCCs remains unsatisfied at around 80% in terms of ,the area under the ROC curve [43,45-47,]. This accuracy may degrade further if the distribution of the samples is severely imbalanced [46]. Unfortunately, such imbalance distribution is widely found for MCCs classification, as usually there are much more (>4 times) benign samples than malignant ones in the training sets [46, 39]. Therefore, the performance of a single classifier may bias to the majority class and fails for correct detection of MCCs. For this purpose, an improved strategy has been proposed, namely balanced learning, to overcome this problem.

5.4.3.1 Strategy in Balanced Learning

To achieve balanced learning, there are two main technical streams, i.e. data level and algorithm level methods [135, 136]. At the data level, the former refer to many re-sampling solutions to balance the training data [137].

On the other hand, algorithm level solutions intend to adjust the cost function, decision threshold or the learnt probability for refined learning, such as the work reported in [138,52]. Using Bayes optimal classifier theory, it is found that individual classifier has a fundamental performance limit which makes it little better than that of the majority class [135,137]. Consequently, data-level solutions are preferred for balanced training in this chapter.

Regarding data level solutions, there are two strategies in the data re-sampling, which include over-sampling of the minority class or under-sampling of majority class. Straightforward over-sampling and under-sampling refer to random replication in the minority class and discarding samples in the majority class.

Although under-sampling may reduce the size of the training set for efficiency, it may lead to serious problems in accurate modelling the majority class as most of data are ignored. On the contrary, random over-sampling seems to be a better solution despite of the increased training set.

Since random over-sampling may increase the likelihood of over-fitting in dealing with the duplicated samples, several smart sampling techniques have been presented such as synthetic over-sampling (SMOTE) [137]. In SMOTE, synthetic minority samples are generated via interpolation of one random sample and its nearest neighbours. Some other smart sampling techniques include one-sided selection, cluster-based over- sampling and Wilson's editing etc., and details of which can be referred to the work in [139].

5.4.3.2 Proposed Balanced Learning Strategy

According to the extensive experiments in [139], it is found that random sampling outperforms several smart sampling techniques and unaltered data set. However, the evaluation in [46] indicates that random over-sampling seems not improving the performance in classification of MCCs, and similar finding is concluded in detecting sentence boundaries in [136].

Besides, it is indicated that SMOTE may outperform down-sampling in certain cases [136]. These inconsistent results need to be further clarified before applying any sampling strategies to classify MCCs for improved performance.

Fig 5.13 illustrates a typical two-class classification problem which contains combined linear decision boundaries. This is very common in machine learning domain and the segment of the decision boundary can also be non-linear. For the two classes marked as circle and star shapes, two pairs of same-class samples are extracted satisfying minimum neighbouring distance and marked as A-B and C-D. According to the rules of smart sampling in SMOTE, synthetic samples can be generated for balanced learning.

Unfortunately, the generated samples in these cases are unreliable noisy ones which may inevitably degrade the performance of training and classification.

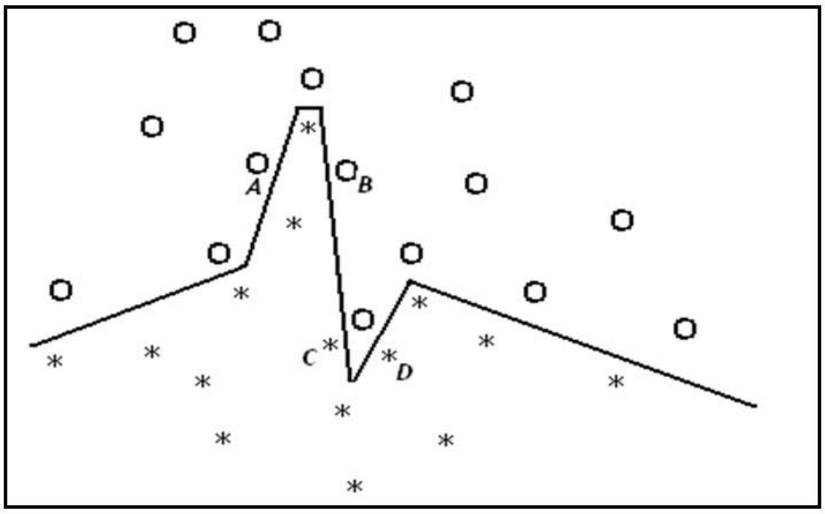


Figure 5.13: Illustrating a Two-class Problem with Combined Linear Decision Boundaries Where the Interpolation Using SMOTE May Fail for the Sample Pairs of A-B and C-D

To some degree, the analysis above can explain why smart sampling behaves well in some cases. The more complex the decision boundary is, the more noisy samples may be introduced via smart sampling, and hence the worse performance may be achieved.

On the other hand, smart sampling like SMOTE may work well in simpler cases such as the linear problem in detection of sentence boundary [136].

For the classification of MCCs, it is found that associated complexity is very high with the number of support vectors above 30% of the training samples. Consequently, random over-sampling is selected. Since there are much more negative samples than positive ones, the strategy here is for each positive sample in the training set to introduce additional samples. These newly introduced samples are almost replications of the original one with minor changes (increasing or decreasing at less than 1% after normalizing the range of the feature values within $[-1, 1]$) to one item of the feature values which is randomly determined. This helps to keep consistency between generated samples and the original ones for balanced learning and avoiding the problem caused by smart sampling as discussed above. Please note that it is assumed that the samples in our test set contain no noise instances thus the over-fitting caused by over-sampling in training can be avoided.

5.4.3.3 Optimized Decision Making

In our implemented ANN and SVM classifiers, the outputs are continuous values rather than binary symbols. Conventional methods use simple thresholding in decision making. If the outputs are larger than the chosen threshold, a positive sample is detected. Otherwise, it is decided as negative. However, this simple thresholding suffers uneven distribution of the training outputs and leads to poor performance. To overcome this drawback, on the contrary, optimized decision making using optimal thresholding is proposed and described as follows.

The optimal thresholding is achieved through statistical analysis of the output of the classifiers, where SVM is taken to show its principles. Let z_i denote the predicted output for a given input sample x_i with a target label $y_i, y_i \in \{-1, 1\}$, where $z_i \in \{a_0, a_1\}$ and the parameters a_0 and a_1

represent respectively the lowest and the highest boundary of the output from the classifier. Then, two conditional probabilities $p(z_i|y_i = 1)$ and $p(z_i|y_i = -1)$ are obtained. For a given threshold $T \in (a_0, a_1)$, the sum of error classification rate Err is determined as:

$$\begin{aligned} Err(T) &= \omega_1 \sum_i p(z_i|y_i = 1, z_i \leq T) \\ &+ \omega_{-1} \sum_i p(z_i|y_i = -1, z_i > T) \end{aligned} \tag{5.15}$$

where the weights w_i and w_{-1} are simply set as $1/2$. Then, an optimal threshold T_{SVM} can be determined when the minimum cost of error classification is achieved, i.e.

$$T_{SVM} = \arg\ \min(Err(T)) \tag{5.16}$$

Similarly, an optimal threshold T_{ANN} can be determined for the ANN classifier via statistical analysis of its outputs. Consequently, these two optimal thresholds can be used to obtain another group of the classification results.

The effectiveness of the proposed optimized decision making has been fully validated using the improved results as previous presented in the following section.

5.4.4 Implementation and Experimental Results

As discussed in Section 5.3, in total 748 typical samples are extracted from the well-known DDSM database including 633 benign (normal) cases and 115 malignant (abnormal) cases. All 748 MCC samples are randomly partitioned into two subsets for training and test, respectively. All the positive samples in the training set are over-sampled to enable balanced learning using SVM and ANN. The models determined are then used to classify samples in the test set. This process is repeated 10 times to overcome any bias in data partition. The average performance over these 10 times is taken as the final result for evaluations.

For a two-class problem, let us denote TP and TN as correctly classified positive and negative samples, FP and FN for incorrectly classified positive and negative samples, i.e. false alarms and missed positives. Several metrics can be determined for quantitative evaluations as follows.

$$TP_{rate} = Recall = TP/(TP + FN) \tag{5.17}$$

$$Precision = TP/(TP + FP) \tag{5.18}$$

$$FP_{rate} = 1 - Specificity = FP/(TN + FP) \tag{5.19}$$

To enable a single measure of performance, a F_1 measurement is also popularly used as defined below.

$$F_1 = \frac{2Recall * Precision}{Recall + Precision} \tag{5.20}$$

Receiver operating characteristic (ROC) analysis and its variants are commonly used for quantitative evaluations of classifiers, especially for the detection and classification of MCCs [103]. In ROC analysis, TP vs. FP rates are adopted. Under ROC analysis, the area under the ROC curve A_z is also used as an important evaluation criterion [103], where $A_z = 1$ indicates an ideal case with $TP_{rate} = 100\%$ and $FP_{rate} = 0$.

For the ANN classifier, the back-propagation network (BPN) is used. The number of nodes in the hidden layer is empirically set as 16 for the better results achieved. The training process stops when the training performance keeps unchanged over a long time, say more than 4000 iterations. The performance is measured using the F_1, and the parameters which yield the highest F_1 value is stored and used for testing.

The RBF kernel has been adopted in our SVM implementation as it can generate particular good results. Cross validation is employed to determine the optimal parameters such as the cost value as 20000.

With continuous values outputted from ANN and SVM, the ROC curve can be determined as follows. Firstly, the maximum and mimimum values of the output are extracted. Then, the threshold is set between these two values to obtain a group of TP, FP and F_1. These will form one point in the plot of the ROC curve. In fact, the threshold is selected evenly distributed between the maximum and the mimimum outputted values, and in total over 100 points are determined for high accuracy in the plotted ROC curve. These can be found clearly in the results as shown in the next subsection.

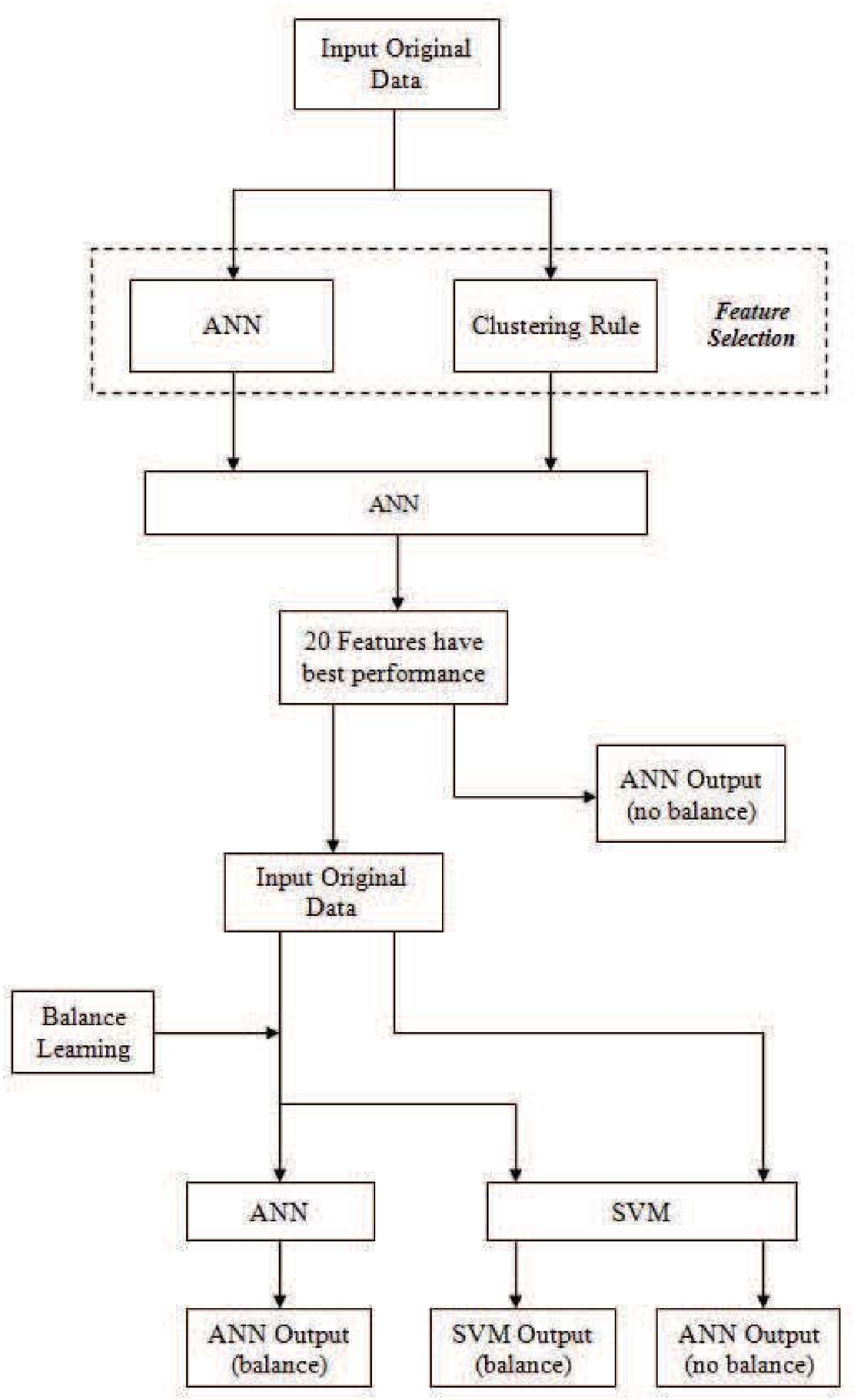
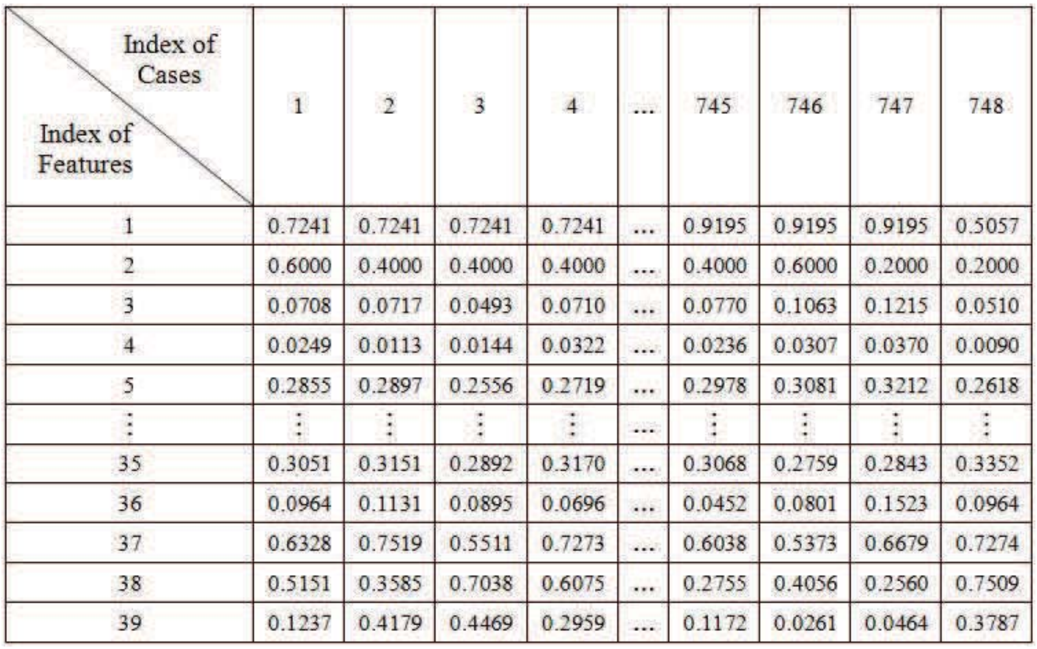


Figure 5.14: Flow Chart for ANN and SVM

①: Input data:

As discussed in Section 5.3, in total 748 typical samples are extracted from the well-known DDSM database including 633 benign (normal) cases and 115 malignant (abnormal) cases, everyone case has been extracted 39 features, that is mean, the original input data is one 39×748 matrix, it shown in Table 5.6. All 748 MCC samples are randomly partitioned into two subsets for training and test, respectively.

Table 5.6: The Original Input Data



Index of Features \ Index of Cases	1	2	3	4	...	745	746	747	748
1	0.7241	0.7241	0.7241	0.7241	...	0.9195	0.9195	0.9195	0.5057
2	0.6000	0.4000	0.4000	0.4000	...	0.4000	0.6000	0.2000	0.2000
3	0.0708	0.0717	0.0493	0.0710	...	0.0770	0.1063	0.1215	0.0510
4	0.0249	0.0113	0.0144	0.0322	...	0.0236	0.0307	0.0370	0.0090
5	0.2855	0.2897	0.2556	0.2719	...	0.2978	0.3081	0.3212	0.2618
⋮	⋮	⋮	⋮	⋮	...	⋮	⋮	⋮	⋮
35	0.3051	0.3151	0.2892	0.3170	...	0.3068	0.2759	0.2843	0.3352
36	0.0964	0.1131	0.0895	0.0696	...	0.0452	0.0801	0.1523	0.0964
37	0.6328	0.7519	0.5511	0.7273	...	0.6038	0.5373	0.6679	0.7274
38	0.5151	0.3585	0.7038	0.6075	...	0.2755	0.4056	0.2560	0.7509
39	0.1237	0.4179	0.4469	0.2959	...	0.1172	0.0261	0.0464	0.3787

As described in last paragraph, there are 115 cases are malignant and 633 cases are benign in totally 748 cases. As the results, before start feature selection, randomly choose 10% from malignant cases and 10% from benign cases in order to build the training database and left the cases as our testing database.

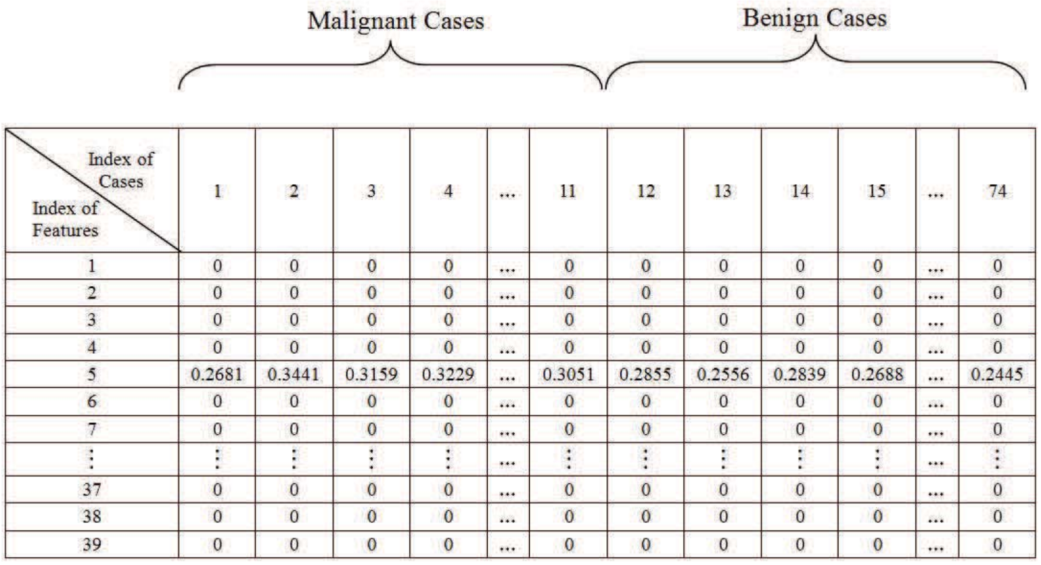
②: Methods:

As described in previous section 5.3.4 in this chapter, there are two methods applied to select features, one is Artificial Neural Network (ANN), and the other one is the Clustering Rules.

In Fig 5.3, has shown the structure of ANN to select the feature. In this method, every time, just bring one feature (the others features' value have been set 0) to training the ANN, and then applied this ANN to test the testing database. Because, in this step, 10% are training data and 90% are testing data, so, there are totally 84 cases are training data (11 cases belong to malignant and 63 cases belong to benign).

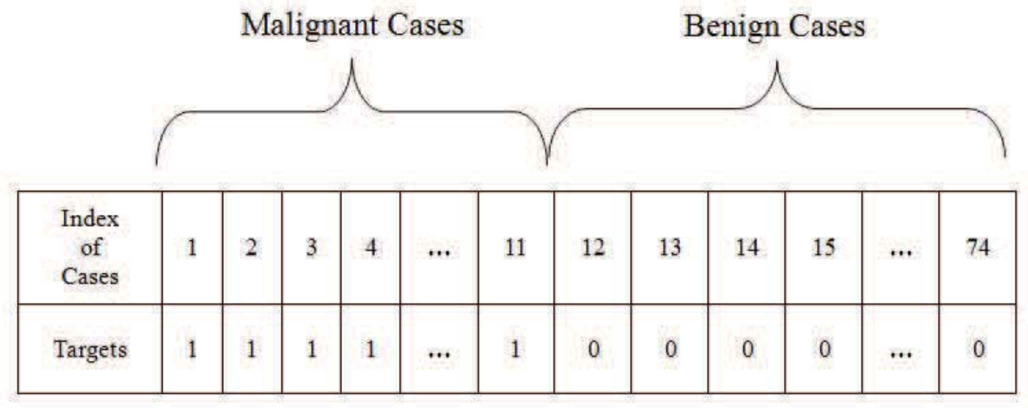
For example, now, test the 5th feature's discrimination, the training data is shown in Table 5.7, the target of ANN's output is shown in Table 5.8.

Table 5.7: Training Data if Test 5th feature's Discrimination



	Malignant Cases						Benign Cases					
Index of Cases / Index of Features	1	2	3	4	...	11	12	13	14	15	...	74
1	0	0	0	0	...	0	0	0	0	0	...	0
2	0	0	0	0	...	0	0	0	0	0	...	0
3	0	0	0	0	...	0	0	0	0	0	...	0
4	0	0	0	0	...	0	0	0	0	0	...	0
5	0.2681	0.3441	0.3159	0.3229	...	0.3051	0.2855	0.2556	0.2839	0.2688	...	0.2445
6	0	0	0	0	...	0	0	0	0	0	...	0
7	0	0	0	0	...	0	0	0	0	0	...	0
⋮	⋮	⋮	⋮	⋮	...	⋮	⋮	⋮	⋮	⋮	...	⋮
37	0	0	0	0	...	0	0	0	0	0	...	0
38	0	0	0	0	...	0	0	0	0	0	...	0
39	0	0	0	0	...	0	0	0	0	0	...	0

Table 5.8: the Targets of the Output


Malignant Cases
Benign Cases

Index of Cases	1	2	3	4	...	11	12	13	14	15	...	74
Targets	1	1	1	1	...	1	0	0	0	0	...	0

Now, according Fig 5.3 to build one ANN system, the Matlab code is:

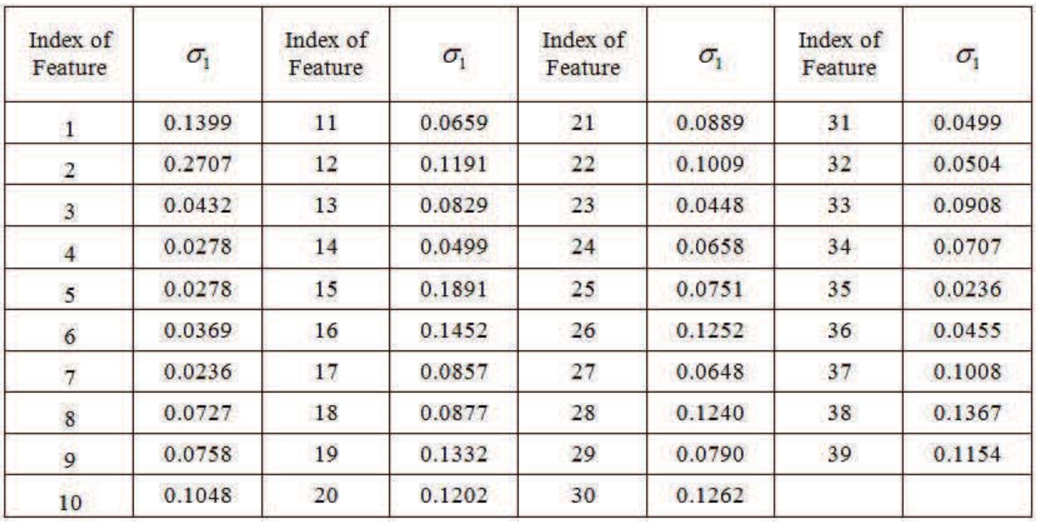
```
net1 = newff(minmax(A),
    [32 16 1],{'logsig' 'logsig' 'purelin'},'trainlm');
net1.inputConnect(2,1) = 1;
net1.biasConnect(1) = 1;
net1.trainParam.goal = 1e-10;
net1.trainParam.epochs = 100;
net1.trainParam.min_grad = 1e-15;
net1.trainParam.lr = 0.5;
```

There is one hidden layer in this ANN system, 'logsig' is transfer function for hidden layer, 'purelin' is transfer function for output layer, 'trainlm' is backprop network training function. In here, the neurons of hidden layer is 16, this is result after tried 10, 12, 14, 16, 18 and 20. Because when set the neurons of hidden layer is 16, the performance of ANN is better.

As known, there are 115 malignant cases in totally cases, therefore, the training data are chosen by 10 times randomly, and keep the results, after that, the Table 5.2 can be obtained:

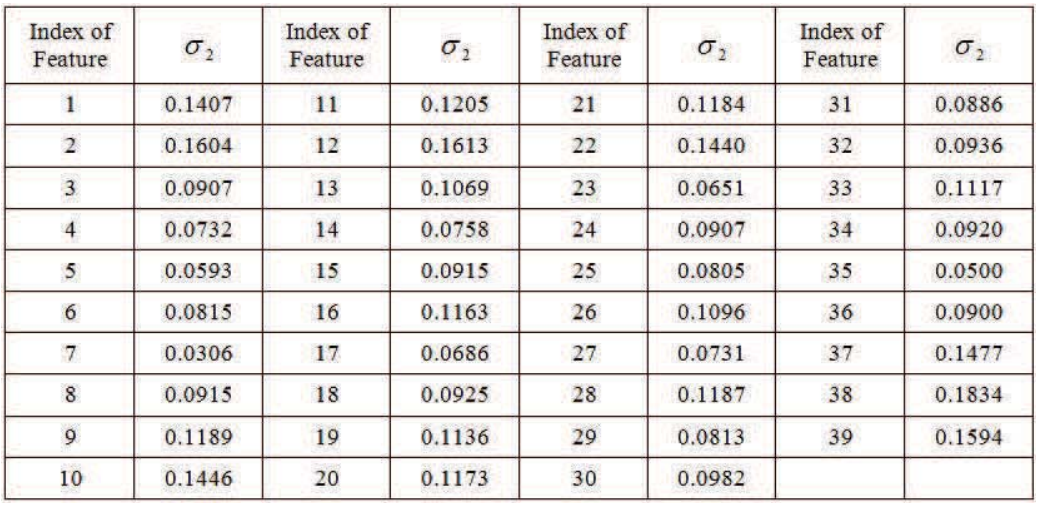
Fig 5.8 has explained the theory about clustering rule, the results for shown from Table 5.9 to Table 5.12.

Table 5.9: Results for σ_1



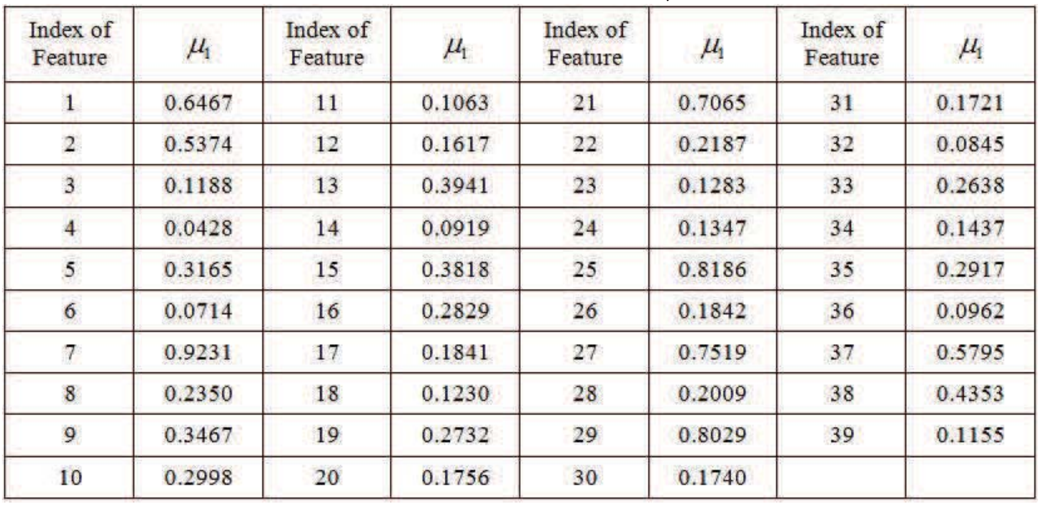
Index of Feature	σ_1	Index of Feature	σ_1	Index of Feature	σ_1	Index of Feature	σ_1
1	0.1399	11	0.0659	21	0.0889	31	0.0499
2	0.2707	12	0.1191	22	0.1009	32	0.0504
3	0.0432	13	0.0829	23	0.0448	33	0.0908
4	0.0278	14	0.0499	24	0.0658	34	0.0707
5	0.0278	15	0.1891	25	0.0751	35	0.0236
6	0.0369	16	0.1452	26	0.1252	36	0.0455
7	0.0236	17	0.0857	27	0.0648	37	0.1008
8	0.0727	18	0.0877	28	0.1240	38	0.1367
9	0.0758	19	0.1332	29	0.0790	39	0.1154
10	0.1048	20	0.1202	30	0.1262		

Table 5.10: Results for σ_2



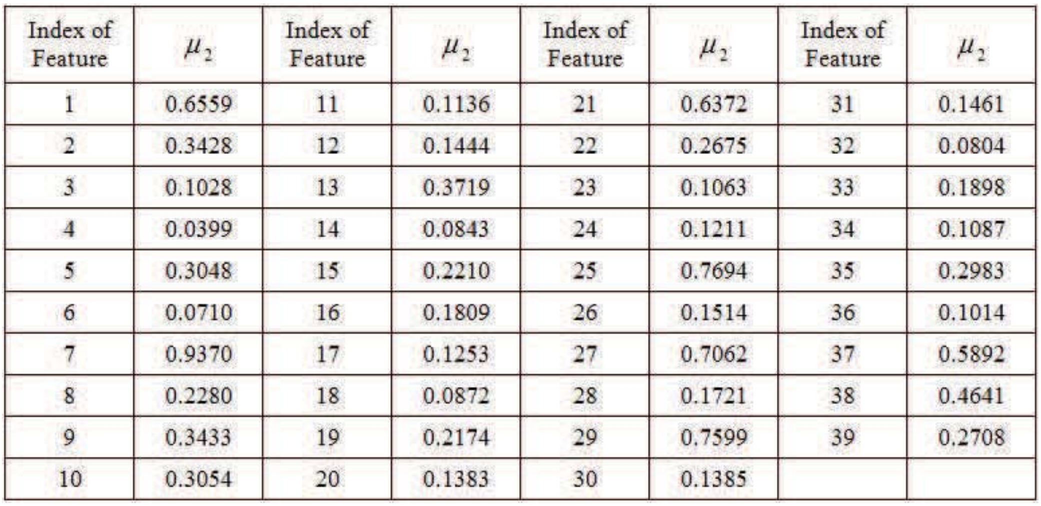
Index of Feature	σ_2	Index of Feature	σ_2	Index of Feature	σ_2	Index of Feature	σ_2
1	0.1407	11	0.1205	21	0.1184	31	0.0886
2	0.1604	12	0.1613	22	0.1440	32	0.0936
3	0.0907	13	0.1069	23	0.0651	33	0.1117
4	0.0732	14	0.0758	24	0.0907	34	0.0920
5	0.0593	15	0.0915	25	0.0805	35	0.0500
6	0.0815	16	0.1163	26	0.1096	36	0.0900
7	0.0306	17	0.0686	27	0.0731	37	0.1477
8	0.0915	18	0.0925	28	0.1187	38	0.1834
9	0.1189	19	0.1136	29	0.0813	39	0.1594
10	0.1446	20	0.1173	30	0.0982		

Table 5.11: Results for μ_1



Index of Feature	μ_1	Index of Feature	μ_1	Index of Feature	μ_1	Index of Feature	μ_1
1	0.6467	11	0.1063	21	0.7065	31	0.1721
2	0.5374	12	0.1617	22	0.2187	32	0.0845
3	0.1188	13	0.3941	23	0.1283	33	0.2638
4	0.0428	14	0.0919	24	0.1347	34	0.1437
5	0.3165	15	0.3818	25	0.8186	35	0.2917
6	0.0714	16	0.2829	26	0.1842	36	0.0962
7	0.9231	17	0.1841	27	0.7519	37	0.5795
8	0.2350	18	0.1230	28	0.2009	38	0.4353
9	0.3467	19	0.2732	29	0.8029	39	0.1155
10	0.2998	20	0.1756	30	0.1740		

Table 5.12: Results for μ_2



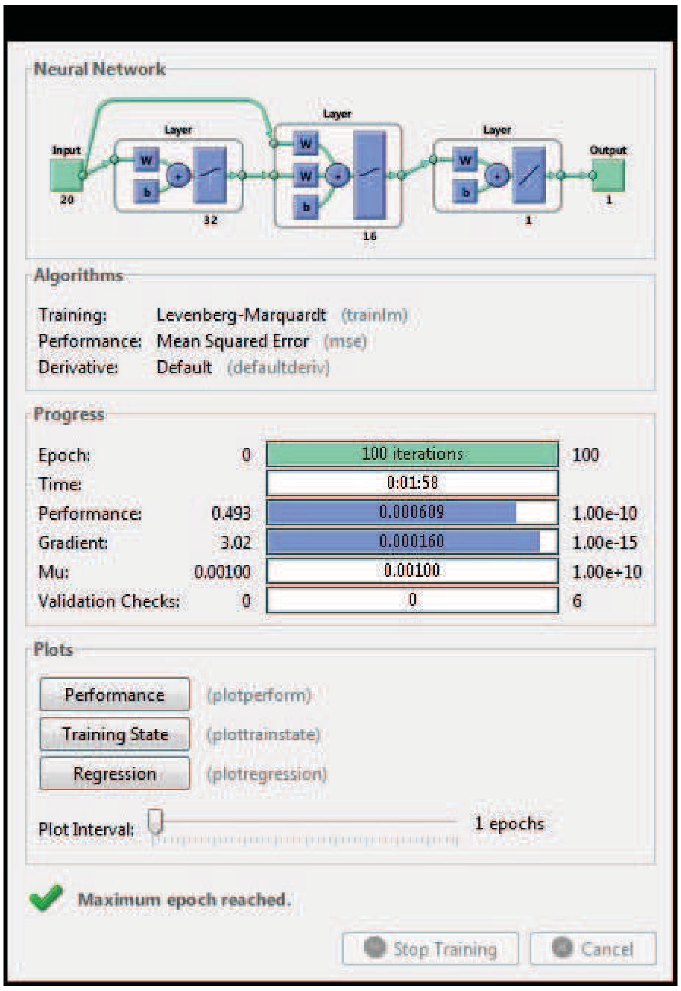
Index of Feature	μ_2	Index of Feature	μ_2	Index of Feature	μ_2	Index of Feature	μ_2
1	0.6559	11	0.1136	21	0.6372	31	0.1461
2	0.3428	12	0.1444	22	0.2675	32	0.0804
3	0.1028	13	0.3719	23	0.1063	33	0.1898
4	0.0399	14	0.0843	24	0.1211	34	0.1087
5	0.3048	15	0.2210	25	0.7694	35	0.2983
6	0.0710	16	0.1809	26	0.1514	36	0.1014
7	0.9370	17	0.1253	27	0.7062	37	0.5892
8	0.2280	18	0.0872	28	0.1721	38	0.4641
9	0.3433	19	0.2174	29	0.7599	39	0.2708
10	0.3054	20	0.1383	30	0.1385		

Now, applying (5.9) then can obtain the Table 5.4.

③: Classification:

In this step, applying the ANN system as the classifier which described in last step.

Firstly, according to the result of selection features from ANN (Table 5.2), 5, 10, 15, 20, 25, 30, 35, 39 features have input into ANN, separately. The number of neurons in hidden layer is still 16. After compare the results from different numbers of features input the ANN, 20 better performance features have been found. Table 5.13 is one example, it shown the results for training and testing if input 20 features. In this experiment, 20% of the totally cases are set as the training data and 80% of the totally cases are testing data.



Algorithms

Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Derivative: Default (defaultderiv)

Progress

Epoch:	0	100 iterations	100
Time:		0:01:58	
Performance:	0.493	0.000609	1.00e-10
Gradient:	3.02	0.000160	1.00e-15
Mu:	0.00100	0.00100	1.00e+10
Validation Checks:	0	0	6

Plots

Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)

Plot Interval: 1 epochs

Maximum epoch reached.

Figure 5.15: ANN model

Table 5.13: Results for Training (20 Features)

Times	Correcting rate of class 1	correcting rate of class 0
1	95.65%	98.46%
2	86.95%	97.67%
3	91.30%	100%
4	95.65%	98.78%
5	86.95%	97.65%
6	91.30%	98.54%
7	82.60%	97.45%
8	95.65%	98.89%
9	91.30%	96.43%
10	82.60%	98.76%

Table 5.14: Results for Texting (20 Features)

Times	Correcting rate of class 1	correcting rate of class 0
1	85.54%	78.67%
2	87.65%	84.78%
3	82.34%	80.34%
4	84.67%	79.65%
5	81.67%	81.84%
6	76.56%	78.90%
7	80.56%	83.45%
8	78.67%	80.15%
9	80.45%	81.45%
10	86.39%	84.17%

After finish all features testing, the Table 5.3 can be obtained.

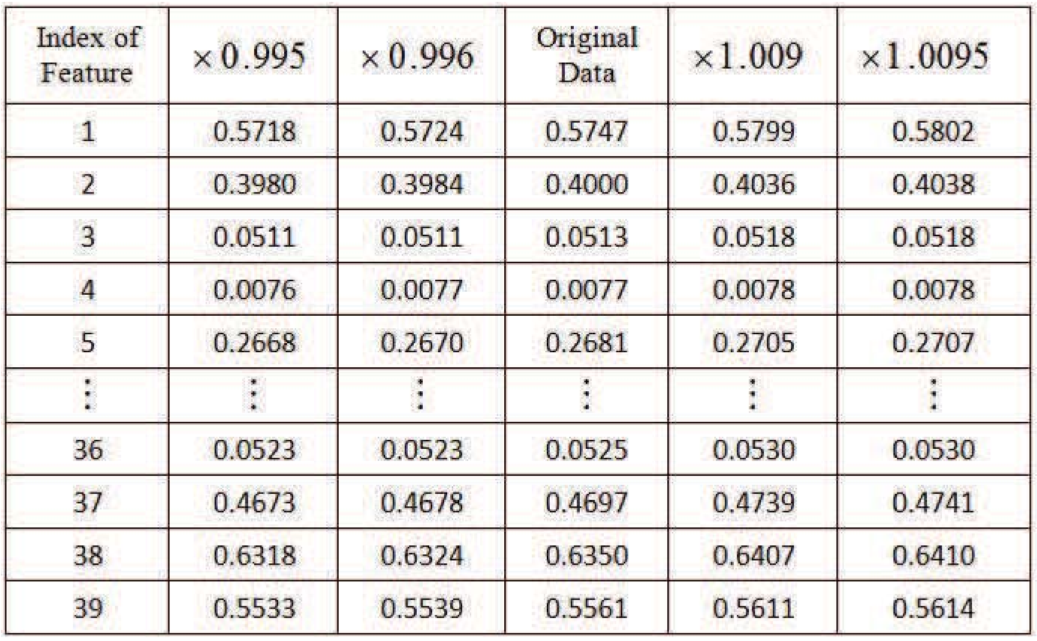
Secondly, same way on clustering rule, the results shown in Table 5.5 can be also obtained.

④: Balance Learning:

Just like describe in section, our database has 115 malignant cases and 633 benign cases, it is means the number of benign cases is 4 times over malignant cases.

According to the balance learning rules just described in 5.4.3.2, the malignant is over-sampled, increase or decrease less than 1% depends on the original feature value, the results shown in Table 5.15

Table 5.15: Over-sample the Malignant Cases



Index of Feature	$\times 0.995$	$\times 0.996$	Original Data	$\times 1.009$	$\times 1.0095$
1	0.5718	0.5724	0.5747	0.5799	0.5802
2	0.3980	0.3984	0.4000	0.4036	0.4038
3	0.0511	0.0511	0.0513	0.0518	0.0518
4	0.0076	0.0077	0.0077	0.0078	0.0078
5	0.2668	0.2670	0.2681	0.2705	0.2707
⋮	⋮	⋮	⋮	⋮	⋮
36	0.0523	0.0523	0.0525	0.0530	0.0530
37	0.4673	0.4678	0.4697	0.4739	0.4741
38	0.6318	0.6324	0.6350	0.6407	0.6410
39	0.5533	0.5539	0.5561	0.5611	0.5614

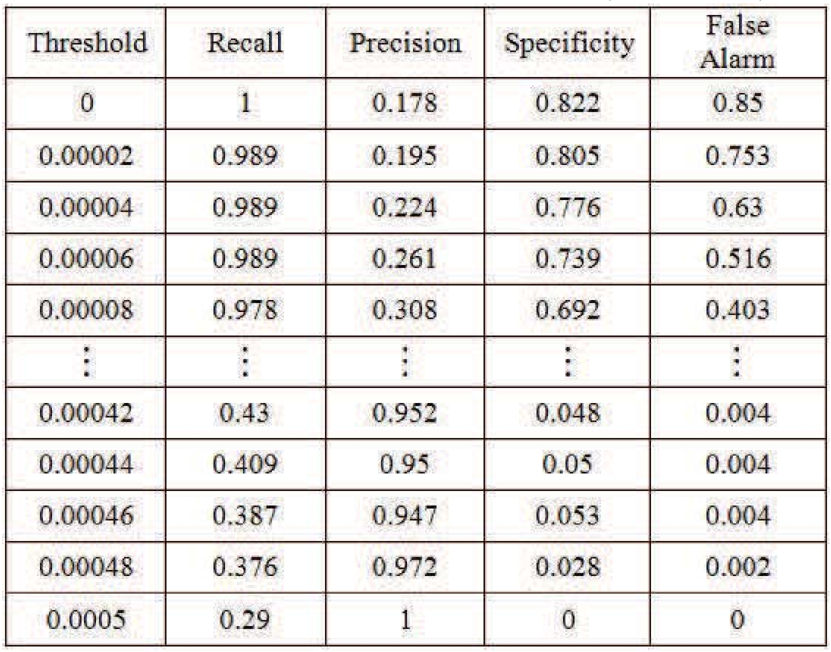
④: Build SVM:

In our SVM, the kernel function is RBF, and the bias is 0.654. The structure of SVM is:

```
svmstruct =

          SupportVectors: [588x25 double]

                   Alpha: [588x1 double]

                    Bias: 6.542257048272806e-001

          KernelFunction: @rbf_kernel

      KernelFunctionArgs: { }

              GroupNames: [599x1 double]

    SupportVectorIndices: [588x1 double]

               ScaleData: [1x1 struct]

           FigureHandles: [ ]
```

The training and testing results (no balance and balance) is shown from Table 5.16 and Table 5.19

Table 5.16: Training Results for SVM (no balance)



Threshold	Recall	Precision	Specificity	False Alarm
0	1	0.178	0.822	0.85
0.00002	0.989	0.195	0.805	0.753
0.00004	0.989	0.224	0.776	0.63
0.00006	0.989	0.261	0.739	0.516
0.00008	0.978	0.308	0.692	0.403
⋮	⋮	⋮	⋮	⋮
0.00042	0.43	0.952	0.048	0.004
0.00044	0.409	0.95	0.05	0.004
0.00046	0.387	0.947	0.053	0.004
0.00048	0.376	0.972	0.028	0.002
0.0005	0.29	1	0	0

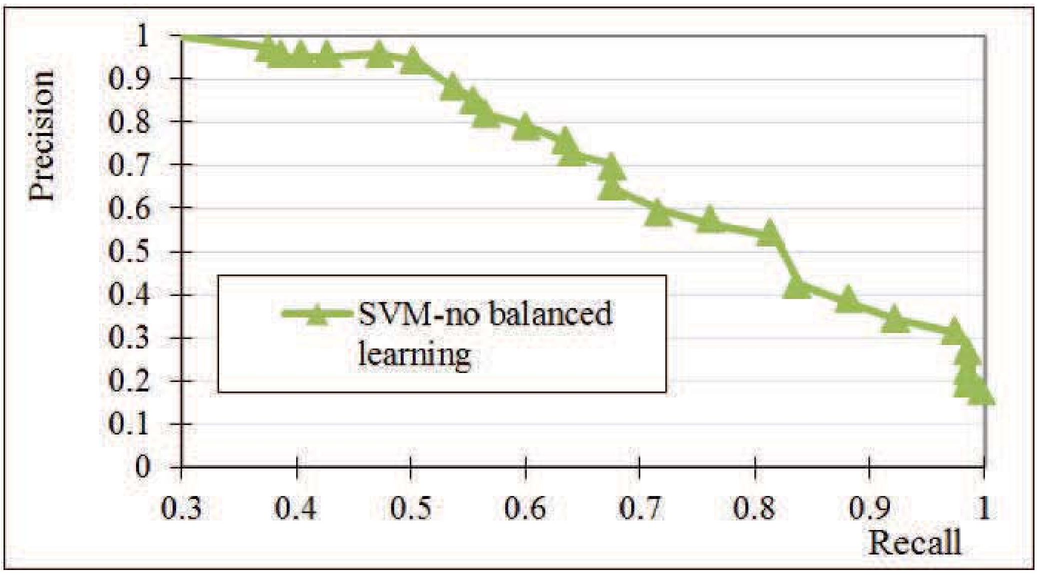
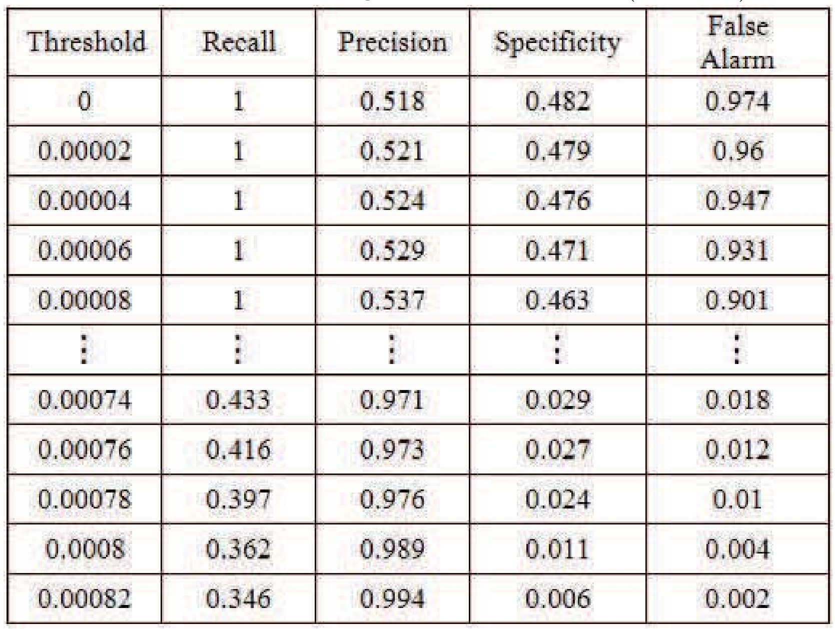


Figure 5.16: Training Results for SVM (no balance)

Table 5.17: Training Results for SVM (balance)



Threshold	Recall	Precision	Specificity	False Alarm
0	1	0.518	0.482	0.974
0.00002	1	0.521	0.479	0.96
0.00004	1	0.524	0.476	0.947
0.00006	1	0.529	0.471	0.931
0.00008	1	0.537	0.463	0.901
⋮	⋮	⋮	⋮	⋮
0.00074	0.433	0.971	0.029	0.018
0.00076	0.416	0.973	0.027	0.012
0.00078	0.397	0.976	0.024	0.01
0.0008	0.362	0.989	0.011	0.004
0.00082	0.346	0.994	0.006	0.002

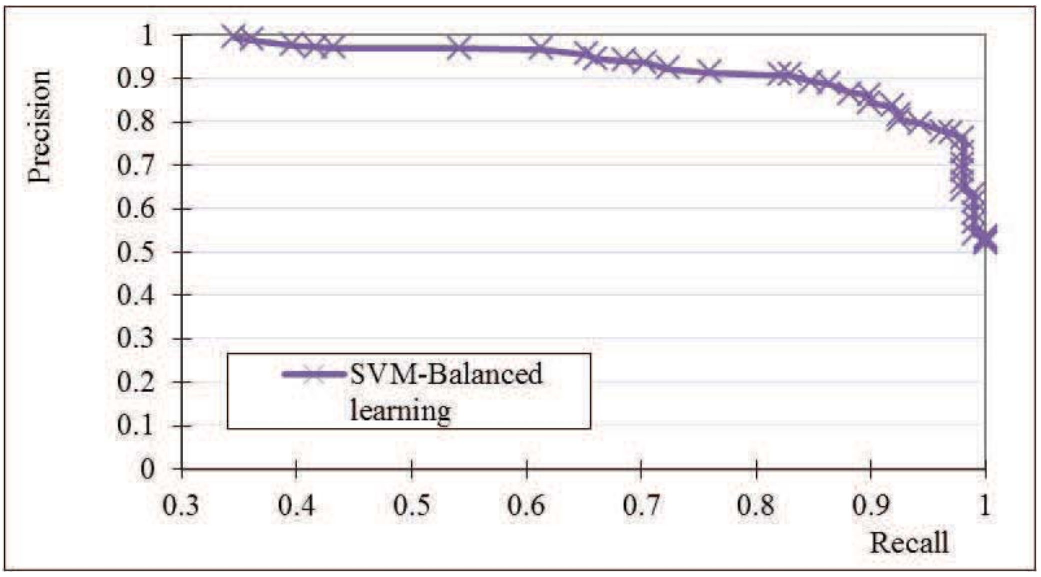


Figure 5.17: Training Results for SVM (balance)

Table 5.18: Testing Results for SVM (no balance)

Threshold	Recall	Precision	Specificity	False Alarm
0	0.955	0.221	0.779	0.583
0.00002	0.955	0.263	0.737	0.465
0.00004	0.955	0.3	0.7	0.386
0.00006	0.909	0.328	0.672	0.323
0.00008	0.818	0.353	0.647	0.26
⋮	⋮	⋮	⋮	⋮
0.00032	0.455	0.714	0.286	0.031
0.00034	0.409	0.75	0.25	0.024
0.00036	0.364	0.889	0.111	0.008
0.00038	0.364	1	0	0
0.0004	0.318	1	0	0

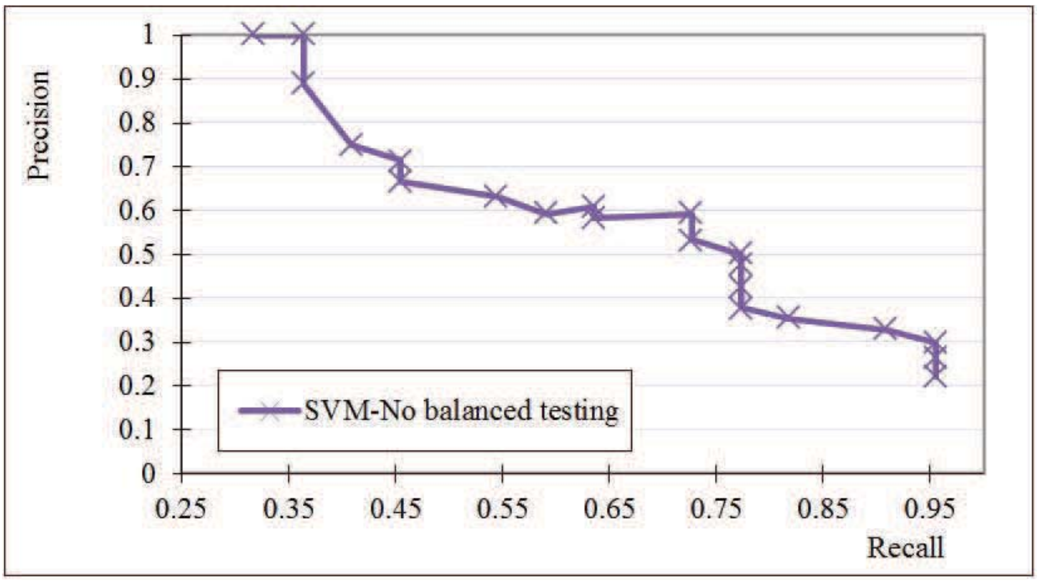
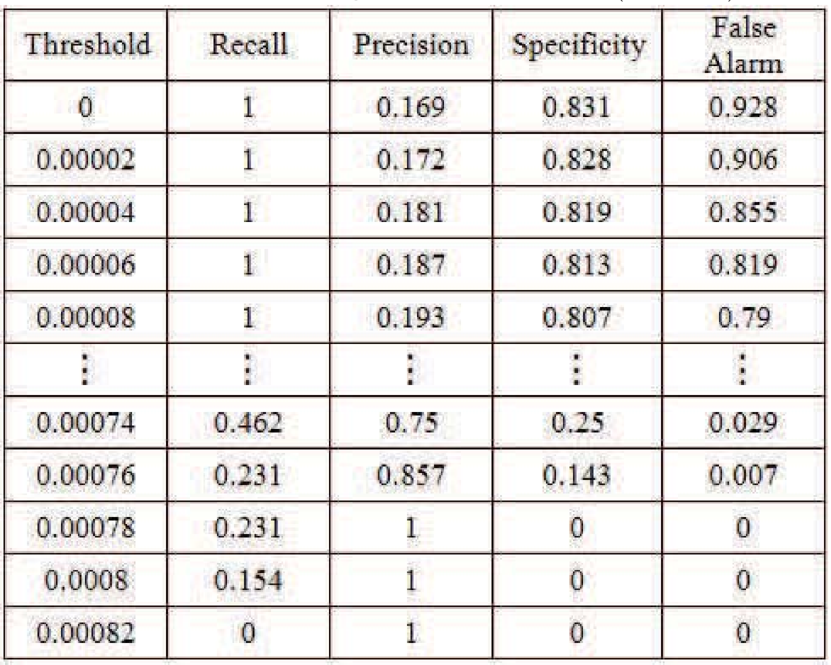


Figure 5.18: Testing Results for SVM (no balance)

Table 5.19: Testing Results for SVM (balance)



Threshold	Recall	Precision	Specificity	False Alarm
0	1	0.169	0.831	0.928
0.00002	1	0.172	0.828	0.906
0.00004	1	0.181	0.819	0.855
0.00006	1	0.187	0.813	0.819
0.00008	1	0.193	0.807	0.79
⋮	⋮	⋮	⋮	⋮
0.00074	0.462	0.75	0.25	0.029
0.00076	0.231	0.857	0.143	0.007
0.00078	0.231	1	0	0
0.0008	0.154	1	0	0
0.00082	0	1	0	0

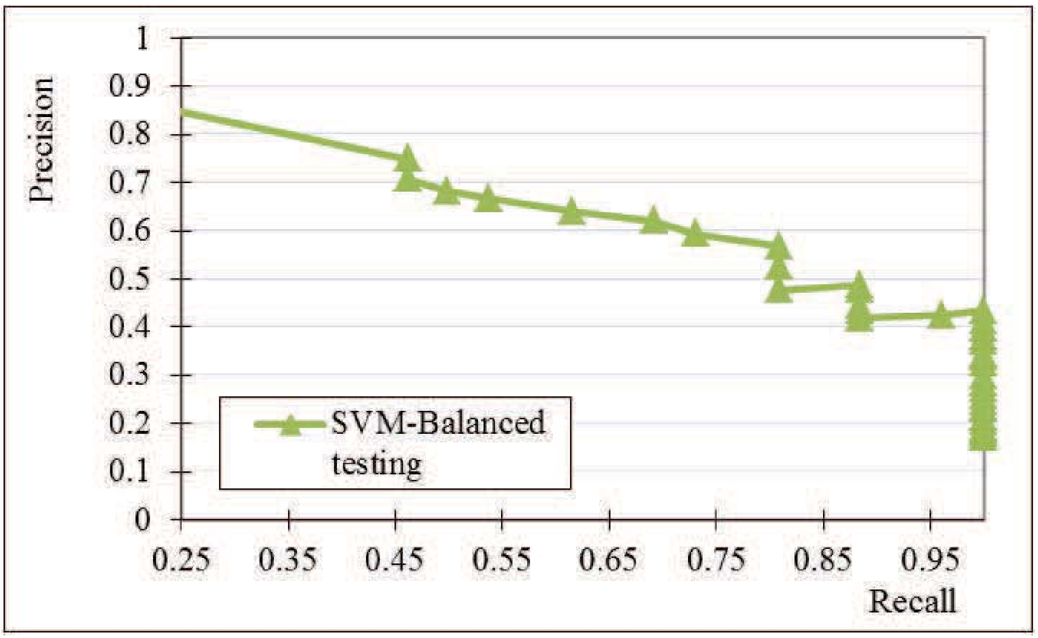


Figure 5.19: Testing Results for SVM (balance)

Same way, the training results and testing results can be obtained under applied the balance learning rule for ANN, the totally results will be shown in next 5.4.5

5.4.5 Results and Discussion

In this section, comprehensive experimental results from ANN and SVM classifiers are presented for the classification of benign and malignant MCCs. Quantitative evaluations are used to validate the effectiveness of our proposed method including balanced learning and optimal decision making.

5.4.5.1 Performance of Balanced Learning

First of all, the performance of balanced learning is compared with those training with the original data, and the training ratio is set as 80%, i.e. 80% of the samples for training and 20% for testing. The ROC curves are plotted in Fig 5.20 to show the performances in training and testing of SVM and ANN with or without balanced learning, respectively, where several facts can be summarized as follows:

Firstly, in general training results are much better than testing ones, especially for the results from ANN, which has validated our analysis that ANN tends to produce minimum errors.

Secondly, it is surprisingly to see that ANN outperforms SVM in both training and testing.

Thirdly, balanced learning indeed can yield better results despite of a little higher false positive rate. Regarding training, it has generated significant higher recall rate for SVM and slightly higher recall rate for ANN though its recall rate without balanced learning is already high enough.

For testing, balanced learning produces much improved results for ANN but limited improvements for SVM. Finally, it is worth noting that balanced learning seems inferior to unbalanced learning only if the false positive rate is less than 3%, although the testing results for the two classifiers are clearly different.

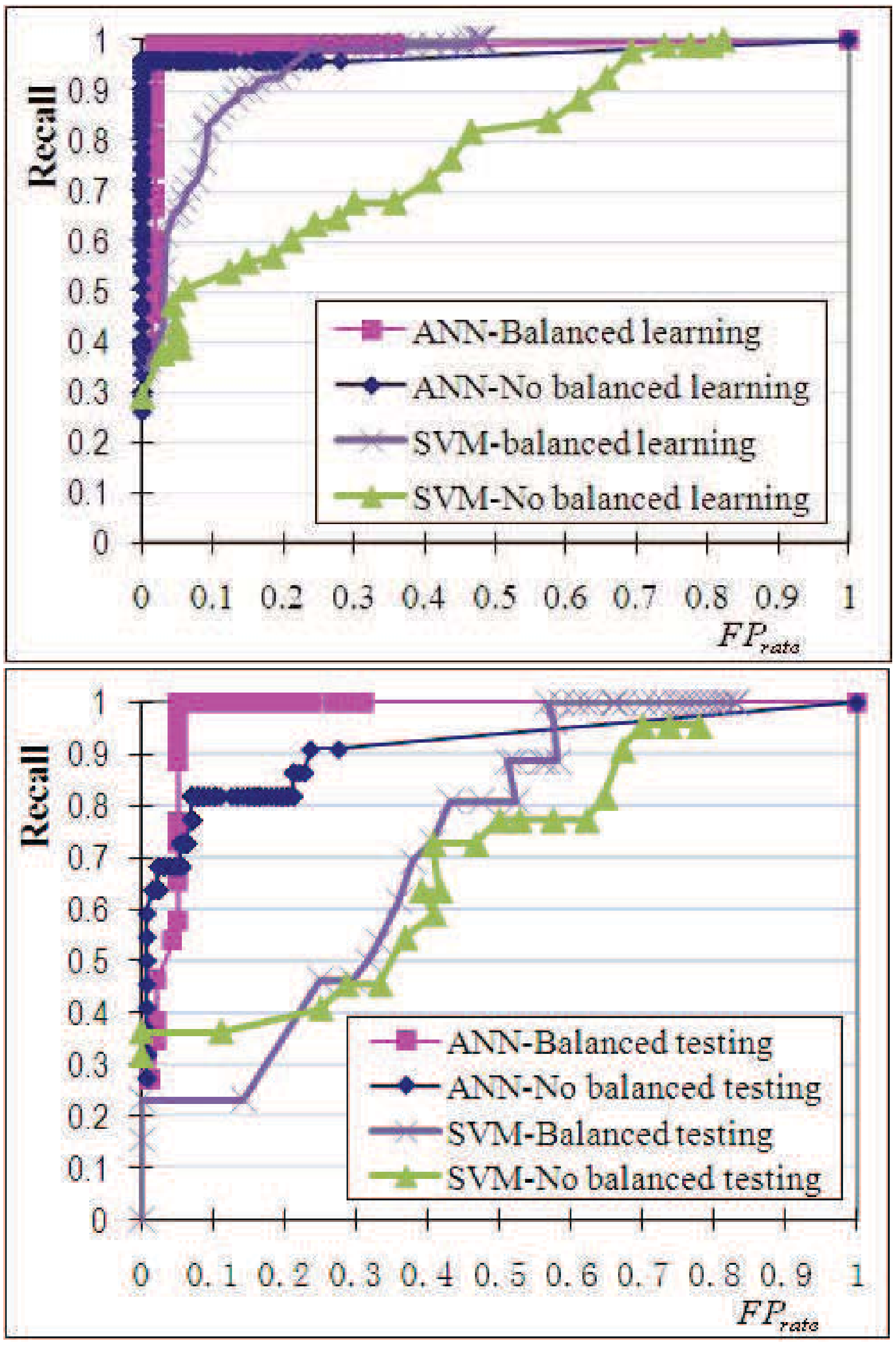
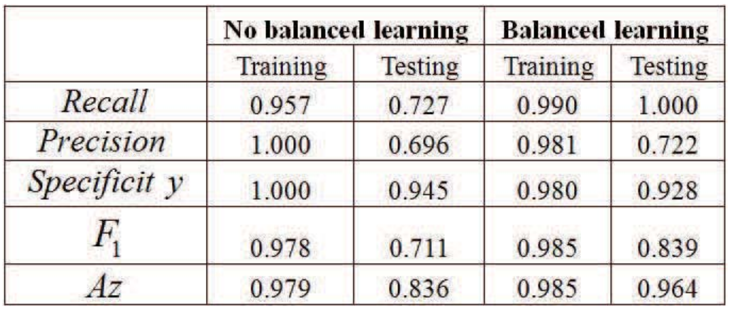


Figure 5.20: ROC Curves of Training and Testing Performances from SVM and ANN With or Without Balanced Learning.

In fact, the results for ANN have no much change, however, the false positive rate becomes more than 20% for ANN to enable balanced learning to achieve a better recall rate. As a higher recall rate is always desirable in such applications, the balanced learning still proves to be better than unbalanced ones.

Quantitative comparisons of the results from ANN and SVM are respectively reported in Table 5.20 and Table 5.21, and no optimized decision making is applied in the testing. First of all, with balanced learning, the testing performance in terms of F_1 and A_z can be significantly improved. This conclusion is different from the work in [46], which has validated the effectiveness of balanced learning.

In addition, it is worth noting that both the training and testing performance from ANN are much better than those from SVM, and further analysis is presented below.



	No balanced learning		Balanced learning	
	Training	Testing	Training	Testing
Recall	0.957	0.727	0.990	1.000
Precision	1.000	0.696	0.981	0.722
Specificit y	1.000	0.945	0.980	0.928
F_1	0.978	0.711	0.985	0.839
Az	0.979	0.836	0.985	0.964

Table 5.20: Training and Testing Results from ANN With or Without Balanced Learning.

In Table 5.20, it is found that the training performance is high in terms of all the five measures no matter balanced training is used or not. This has indicated that ANN is capable of model the problem accurately.

However, the testing performance under balanced training is much better, in which an improvement of 12.8% are achieved in both F_1 and A_z measurements. Although in training almost the same values of F_1 and A_z are obtained, smaller F_1 values in testing are yielded. This is caused by lower $Precision$ values due to false positives.

Due to the severe imbalancement of the data used for testing, a high $Specificity$ value is still achieved under these false alarms to yield a higher A_z measurement. In other words, the ratio between the number of false alarms to the number of malignant samples is much larger than the ratio between it to the number of benign samples, and this has led to lower F_1 but higher A_z values.

In Table 5.21, the results from SVM have some differences.

Firstly, the training results from balanced learning are much better than those without balanced learning, and the improvements in terms of F_1 and A_z are about 46% and 21%, respectively.

Secondly, the testing results from balanced learning are about 20% better in F_1 and A_z measurements than those without balanced learning.

Thirdly, balanced learning has improved the *Recall* rate by about 60%, which means massively reduction of missed detection although more false alarms are introduced to degrade the $Precision$ value from 1 to 0.479.

	No balanced learning		Balanced learning	
	Training	Testing	Training	Testing
Recall	0.301	0.273	0.899	0.885
Precision	0.966	1.000	0.947	0.479
Specificit y	0.998	1.000	0.830	0.819
F_1	0.459	0.429	0.922	0.622
A_z	0.650	0.637	0.865	0.852

Table 5.21: Training and Testing Results from SVM With or Without Balanced Learning

5.4.5.2 Performance of Optimized Decision Making

From Table 5.22 to Table 5.24, the results using our proposed optimized decision making in both ANN and SVM classifiers are given. Again, 80% of samples have been selected for training and 20% for testing. By comparing these results with those in Table 5.6 and Table 5.7, several facts are clearly found which are summarized as follows.

• Without balanced learning, optimized decision making the contributes for the ANN about 1% in F_1 and 3.4% in A_z measurements. For the SVM, the contributions are 22.4% and 18.3%, respectively. (See details in Table 5.22)

	ANN	ANN(ODM)	Contribution	SVM	SVM(ODM)	Contribution
F_1	0.711	0.720	1%	0.429	0.653	22.4%
A_z	0.836	0.870	3.4%	0.637	0.820	18.3%

Table 5.22: Without the Balance Learning, the Contribution of ODM (Optimized Decision Making)

• Regarding balanced learning, the improvements for ANN are 4.2% in

F_1 and 1.1% in A_z even the original A_z value is as high as 96.4%. However, SVM gains 4.5% in F_1 but 0.6% degradation in A_z measurements. (See in Table 5.23)

	ANN	ANN(ODM)	Contribution	SVM	SVM(ODM)	Contribution
F_1	0.839	0.881	4.2%	0.622	0.667	4.5%
A_z	0.964	0.975	1.1%	0.852	0.846	-0.6%

Table 5.23: With the Balance Learning, the Contribution of ODM (Optimized Decision Making)

This on one hand has fully validated the effectiveness of the proposed strategy for optimized decision making in terms of an improved F_1. On the other hand, significant improvements have achieved for the SVM classifier when balanced learning is not employed. Nevertheless, the results from ANN remain better than those from SVM in both F_1 and A_z measurements.

	No balanced learning		Balanced learning	
	Training	Testing	Training	Testing
Recall	0.818	0.727	1.000	0.808
Precision	0.643	0.593	0.788	0.568
Specificit y	0.921	0.913	0.949	0.884
F_1	0.720	0.653	0.881	0.667
A_z	0.870	0.820	0.975	0.846

Table 5.24: Testing Results from ANN and SVM under Optimized Decision Making With or Without Balanced Learning

5.4.5.3 Performance under Various Training Ratios

In the tests above, the training ratio is fixed at 80%. In this group of tests, the performance under various training ratios is compared. Under various training ratios, the training results and two testing results with or without optimized decision making are evaluated in terms of F_1 and Az measurements. These results are illustrated in Fig 5.21 and Fig 5.22, where Test2 denotes results using optimal decision making.

In total there are three pair of curves in each figure in which one is from training and the other two from testing without or with optimal decision making. Each pair of curves is plotted using the training ratio (changed from 50% to 90%) vs. performance of F_1 and A_z measurements and they are further discussed as follows.

Firstly, the F_1 and A_z measurements from training with or without balanced learning are very close to each other and appear insensitive to the training ratio, which again shows that ANN is capable in accurate modelling the problem.

Secondly, the testing results using balanced learning are much better that those without balanced learning.

Thirdly, in most cases optimized decision making produces better results in F_1 and A_z measurements when balanced learning is employed, except the result at the training ratio of 60%. When balanced learning is not used, however, better F_1 measurement can be only yielded when the training ratio is between 70% and 80%.

In addition, better A_z measurement can always be achieved from optimized decision making even the balanced learning is skipped.

In Fig 5.21, the training and testing rersults from ANN are illustrated. The results from SVM as illustrated in Fig 5.22 show some different facts.

Firstly, the training performance from SVM is not superior to the testing results as shown from ANN. One possible reason is the so-called high generalization capacity as it tends to avoid overfitting hence the relative poorer trainuing performance.

Secondly, balanced learning is useful in yielding better training and testing results, especially when the training ratio is around 75%.

Thirdly, under balanced learning optimized decision making can significantly improve F_1 measurement but such improvement on A_z measurement is quite limited and can only be found when the training ratio is between 55% and 70%.

In addition, optimized decision making helps to gain apparent improvements in both F_1 and A_z measurement when balanced learning is not used. This is because that balanced learning has reduced the diversity degree of the training samples. As a result, there is very limited space for optimized decision making for further improvement. On the contrary, there is much large space for optimized decision making in improving the results from high diverse data when balanced learning is not used.

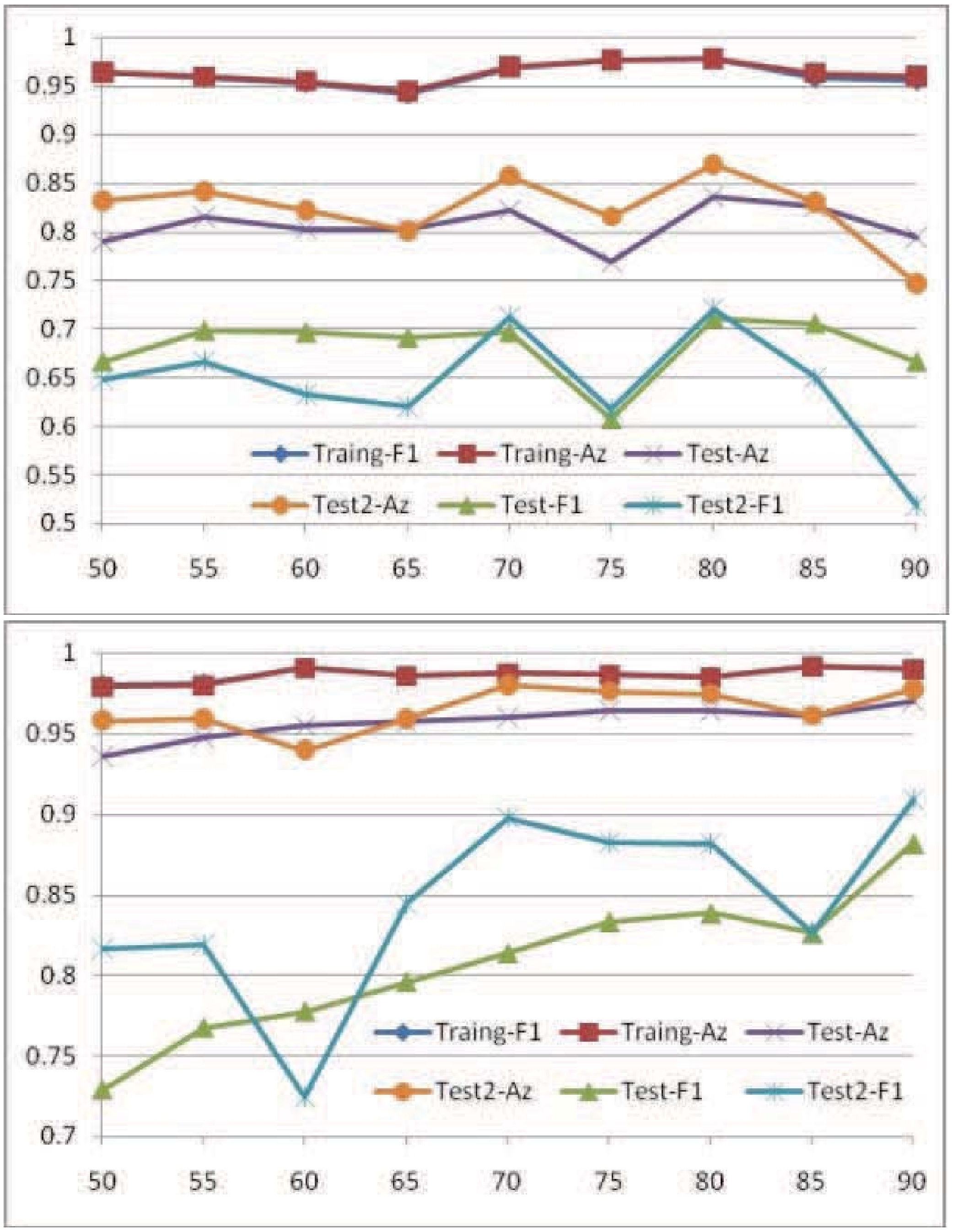


Figure 5.21: Training and Testing Results from ANN Using Plots of Training Ratio (x-axis) vs. F_1 and A_z Measurements, Where the Top and the Bottom Plots Refer Respectively to Results Without or With Balanced Learning.

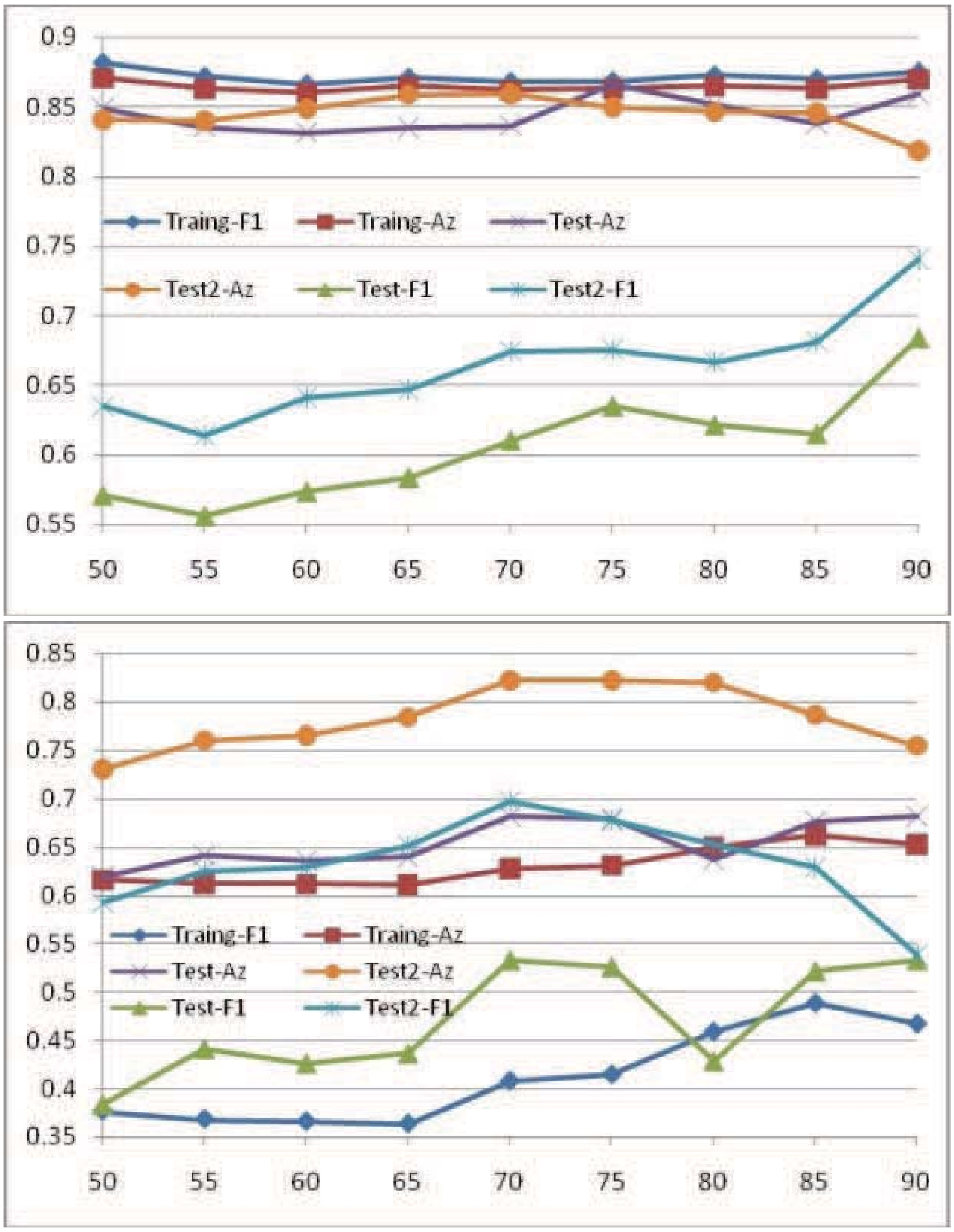


Figure 5.22: Training and Testing Results from SVM Using Plots of Training Ratio (x-axis) vs. F_1 and A_z Measurements, Where the Top and the Bottom Plots Refer Respectively to Results Without or With Balanced Learning.

5.4.5.4 Computational Complexity

In comparison with conventional ANN and SVM, the proposed balanced learning and optimized decision making do need additional computations. As optimized decision making does not involve in the training iterations, it can be simply ignored.

In the following, the effect of balanced learning in such a context is analysed.

Since more training samples are introduced in balanced learning, it costs more time in learning the model. Let $N = N_0 + N_1$ be the total samples, where N_0 and N_1 denote respectively the number of negative and positive samples satisfying $N_0 = KN_1$, $K > 2$.

After over-sampling, $(K-1)N_1$ new positive samples are produced, and in total $2N_0$ training samples. This equals to $2NK/(K+1)$ and less than $2N$, which has indicated that the number of samples under balanced learning is less than twice of the number of the original samples.

Under same number of iterations, the training time should not double one without balanced learning.

In addition, it is found that balanced learning needs less number of iterations to converge, which is about 77% of the one required for training without balanced learning. This fast converging might due to the improved distributions of training samples from our balanced learning. Consequently, the increased complexity under balanced learning is:

$$2K/(K+1) \times 77\% - 1 = 0.54 - 1.54/(K+1) \tag{5.21}$$

This indicates a maximum of 54% additional computing burden, which is totally acceptable for the benefit of much improved performance.

5.5 Summary

In this chapter, high-level image processing is presented, using detection of microclacification clusters from mammograms for computer-aided diagnosis for case study. The main techniques involved include feature selection and classification, and the main work can be summarised as follows.

Regarding feature selection, PCA based clustering and ANN is employed. For the extracted 39 features, each of them can then be assigned a rank to measure its discriminative ability. By extracting the most discriminative feature respectively, it is found that results from the two approaches are quite consistent.

To deal with classification of imbalanced data (in 748 samples there are 115 malignant ones and 633 benign ones), a novel approach is proposed for balanced learning with optimised decision making. By applying the same strategy to both ANN and SVM classifiers, it is found that the proposed approach can significantly improve the overall performance in terms of higher

F_1 and Az measurements for the two classifiers. Moreover, it seems that ANN can yield higher accuracy and outperforms SVM in the experiments.

Chapter 6

Conclusion

6.1 Thesis Contribution

After a comprehensive survey of existing techniques, four different themes are respectively discussed, they include: mammograms, colour edge detection, colour skin and face detection and hand gesture recognition. The contribution of this thesis can be summarised as follows:

- For colour edge detection, we have found that inter-component information in colour images is very important for accurate edge detection, though it has been ignored in many existing approaches. Through the fusion of intensity and chromatic difference, the proposed scheme is found to be very useful in generating better colour edges. In terms of effectiveness and robustness under Gaussian noise, both visual and quantitative evaluations were carried out. Comprehensive results from several standard test images have fully verified both the effectiveness and robustness of our proposed approach, which is found outperforming edges extracted from RGB, YCbCr and HSV spaces.
- For colour skin and face detection, by comparative study of skin detection from different colour spaces, we find nonlinear colour spaces, such as HSV, can obtain more accurate and robust skin result. Moreover, we find the shape filtering and knowledge-based modelling very useful in face detection. Besides, these detected skin and face regions can be further utilized for semantic indexing and retrieval of images.
- For hand gesture recognition, we described a proposed algorithm for

human gesture recognition and demonstrated its discriminative ability for recognition of gestures on a large database of images. By using Gaussian Mixture Model, we have shown that multiple features extracted from gesture images could be organised and controlled by GMM to formulate new discriminating vector for classification and recognition of human gestures. The application of Gaussian Mixture Model illustrates the advantage that it provides improved performance over other existing methods, yet requiring only modest computational cost to complete the gesture recognition.

- For high level image processing,

    1. Apply neural network and PCA on selection features respectively;
    2. Combine the results from the first step and find the best performance of the combination;
    3. Balanced learning with optimized decision making is proposed for classification of benign and malignant MCCs in mammograms;

The proposed methodology has been tested on two common used machine learning approaches including ANN and SVM. Firstly, balanced learning indeed has significantly improved the classification accuracy, and an average gain of more than 10% can be achieved for the two classifiers in terms of both and measurements. Secondly, optimized decision making produces improved results in F_1 and A_z for ANN no matter balanced learning is used or not. For SVM, however, more than 18% of improvements in F_1 and A_z can only be found without balanced learning. Otherwise, improved F_1 but slightly degraded A_z are produced. Thirdly, the overall results from ANN are much better than those from SVM, which is different from the work reported [43, 45-47]. Fourthly, a training ratio between 70% and 80% is suggested due to the various performances under different training ratios. Finally, it is found that the suggested balanced training will only bring up to 54% of additional computation load, a tolerable cost for the much improved performance.

6.2 Further Work

Although the work which has been presented in this thesis demonstrates a certain level of success in the relevant fields, there still exists some potential for improvement and further investigation. Accordingly, some suggestions are listed as follows.

- For colour skin and face detection, how to improve quantitative analysis of the shape filters and face modelling for more accurate and robust face detection, especially on separation of connected faces and detection of background faces.
- For colour edge detection, Further investigation will be to apply the fusion scheme on image segmentation and content-based retrieval applications as well as to introduce more powerful post-processing to remove separated false alarms.

- For hand gesture recognition, further research can be identified to focus on the issue of extendibility and selection of primary features as such that other pattern recognitions can be achieved, especially inside digital videos.
- For high level image processing, further investigations include introducing feature selection approaches for improved efficiency as well as reducing false alarms for more robustness.

References

[1] R. Gaughan, " New approaches to early detection of breast cancer makes small grains, " *Biophotonics Int., pp. 48-53, Sept/Oct, 1998*

[2] S. Shapiro, W. Venet, P. Strax, L. Venet, and R. Roeser, " *Ten-to-fourtenn-year effect of screening on breast mortality" JNCL, vol. 69, p. 349, 1982*

[3] R. G. Lester, " The contribution of radiology to the diagnosis, management, and cure of breast cancer," *Radiology, vol. 151, 1984*

[4] M. Moskowitz, Benefit and Risk, Breast Cancer detection: Mammography and other methods in breast Imaging, *2nd ed, L. W. Bassel and R. H. Gold, Eds. New York: Grune and Stratton, 1987*

[5] R. A. Smith, " Epidemiology of breast cancer categorical course in physics, " Tech. Aspects Breast *Imaging, Radiol. Soc. N. Amer., pp 21-33, 1993*

[6] N. Karssemeijer, "Computer-Assisted Reading Mammograms", *European Radiol, vol 7, pp 743-748, 1997*

[7] H. P. Chan, K. Doi, C. J. Vyborny, R. A. Schmidt, C. E. Metz, K. L. Lam, etc. "Improvement in radiologist's detection of clustered microcalcifications on mammograms, " *Investigative radiol, vol 25, pp. 1102-1100, 1990*

[8] S. Astley, I. Hutt, and S. Adamson et al., " Automation in mammography: Computer vision and human perception, " *Int, J. Pattern Recognit. Artifical Intell., vol 7, no. 6, pp. 1313-1338, 1993*

[9] Hunke, M., Waibel, A.: Face Locating and Tracking for Human-Computer Interaction. *IEEE Computer, (1996) 1277-1281*

[10] Cui, Y., Weng, J.: Appearance-Based Hand Gesture Sign Recognition from Intensity Image Sequences. *Computer Vision and Image Understanding, vol. 78, (2000) 157-176*

[11] Wren, C. R., Azarbayejani, A., Darrel, T., Pentland, A. P.: Pfinder: Real-time Tracking of The Human Body. *IEEE T-PAMI. 19(7), (1997) 780-785*

[12] Forsyth, D., Fleck, M.: Automatic Detection of Human Nudes. *Int. J. of Computer Vision, 32(1), (1999) 63-77*

[13] Jones, M. J., Rehg, J. M.: Statistical Colour Models with Application to Skin Detection. *Int. J. Computer Vision. 46(1) (2002) 81-96*

[14] Habili, N., Lim, C. C., Moini, A.: Segmentation of the Face and Hands in Sign language Video Sequences Using Colour and Motion Cues. *IEEE-TCSVT, 14(8) (2004) 1086-1097*

[15] Chai, D., Ngan, K. N.: Face Segmentation Using Skin-Colour Map in Videophone Applications. *IEEE. T-CSVT. 9(4) (1999) 551-564*

[16] Phung, S. L., Bouzerdoum, A., Chai, D.: Skin Segmentation Using Colour Pixel Classification: Analysis and Comparison. *IEEE T-PAMI. 27(1) (2005) 148-154*

[17] Kakumanu, P., Makrogiannis, S., Bourbakis, N.: A Survey of Skin-Colour Modeling and Detection Methods. *Pattern Recognition. 40 (2007) 1106-1122*

[18] V. Athitsos, S. Sclaroff: Estimating 3d hand pose from a cluttered image, *Proceeding of Conference on Computer Vision and Pattern Recognition (CVPR '03), vol. 1, Madison, WI, 2003*

[19] P. H. S. Torr, B. Stenger, A. Thayananthan and R. Cipolla,: Hand pose estimation using hierarchical detection, *Proceeding of International Workshop on human-computer interaction, 2004*

[20] H.D. Cheng, X. Cai, X. Chen, L. Hu, X. Lou: Computer-aided detection and classification of microcalcifications in mammograms: a survey, *Pattern Recog., 36(12) (2003) 2967-2991.*

[21] S. Sentelle, C. Sentelle, M.A. Sutton: Multiresolution-based segmentation of calcifications for the early detection of breast cancer, *Real-Time Imag., vol. 8, no. 3, pp. 237-252, June 2002.*

[22] A. Papadopoulos, D.I. Fotiadis, L. Costaridou, "Improvement of microcalcification cluster detection in mammography utilizing image en-

hancement techniques," *Computers in Biology and Medicine, 38(10) (2008) 1045-1055.*

[23] J. Grim, P. Somol, M. Haindl, J. Danes, "Computer-aided evaluation of screening mammograms based on local texture models," *IEEE Trans. Image Proc., 18(4) (2009) 765-773.*

[24] K. Thangavel, M. Karnan, A. Pethalakshmi, "Performance analysis of rough reduct algorithms in mammogram," *ICGST-GVIP Journal, 5(8) (2005) 13-21.*

[25] E.A. Rashed, I.A. Ismail, S.I. Zaki, "Multiresolution mammogram analysis in multilevel decomposition," *Pattern Recog. Letters, 28(2) (2007) 286-292.*

[26] H. Soltanian-Zadeha, F. Rafiee-Radc, S. Pourabdollah-Nejad, "Comparison of multiwavelet, wavelet, Haralick, and shape features for microcalcification classification in mammograms," *Pattern Recog., 37(10) (2004) 1973-1986.*

[27] P. Heinlein, J. Drexl, W. Schneider, "Integrated wavelets for enhancement of microcalcifications in digital mammography," *IEEE Trans. Med. Imag., 22(3) (2003) 402-413.*

[28] S. Sentelle, C. Sentelle, M.A. Sutton, "Multiresolution-based segmentation of calcifications for the early detection of breast cancer," *Real-Time Imag., vol. 8, no. 3, pp. 237-252, June 2002.*

[29] M. Kallergi, "Computer-aided diagnosis of mammographic microcalcification clusters," *Med. Phys., 31(2) (2004) 314-326.*

[30] K.Thangavel, M. Karnan, "Computer aided diagnosis in digital mammograms: detection of microcalcifications by meta heuristic algorithms," *ICGST-GVIP Journal, 5(7) (2005) 41-55.*

[31] A.E. Hassanien, "Fuzzy rough sets hybrid scheme for breast cancer detection," *Image and Vision Comput., 25(2) (2007) 172-183*

[32] H.D. Cheng, J. Wang, X. Shi, "Microcalcification detection using fuzzy logic and scale space approaches," *Pattern Recog., 37(2) (2004) 363-375.*

[33] R. Nakayama, Y. Uchiyama, K. Yamamoto, et al, "Computer- aided diagnosis scheme using a filter bank for detection of microcalcification clusters in mammograms," *IEEE Trans. Biomed. Eng., 53(2) (2006) 273-283.*

[34] L. Wei, Y. Yang, R.M. Nishikawa, et al, "Relevance vector machine for automatic detection of clustered microcalcifications," *IEEE Trans. Med. Imag., 24(10) (2005) 1278-1285.*

[35] P. Sajda, C. Spence, J. Pearson, "Learning contextual relationships in mammograms using a hierarchical pyramid neural network," *IEEE Trans. Med. Imag., 21(3) (2002) 239-250.*

[36] L. Bocchi, G. Coppini, J. Nori, G. Valli, "Detection of single and clustered microcalcifications in mammograms using fractals models and neural networks," *Med. Eng. & Phy., 26(4) (2004) 303-312.*

[37] R. Zwiggelaar, S.M. Astley, C.R.M. Boggis, C.J. Taylor, "Linear structures in mammographic images: detection and classification," *IEEE Trans. Med. Imag., 23(9) (2004) 1077-1086.*

[38] J. Ge, B. Sahiner, L.M. Hadjiiski, H.-P. Chan, J. Wei, M.A. Helvie, C. Zhou, "Computer aided detection of clusters of microcalcifications on full field digital mammograms," *Med. Phys., 33(8) (2006) 2975-2988.*

[39] Z.Q. Wu, J. Jiang, Y.H. Peng, "Effective features based on normal linear structures for detecting microcalcifications in mammograms," *in Proc. ICPR, pp. 1-4, 2008.*

[40] M. De Santo, M. Molinara, F. Tortorella, M. Vento, "Automatic classification of clustered microcalcifications by a multiple expert system," *Pattern Recog., 36(7) (2003) 1467-1477.*

[41] L. Hadjiiski, B. Sahiner, H.-P. Chan, et al, "Classification of malignant and benign masses based on hybrid ART2LDA approach," *IEEE Trans. Med. Imag., 18(12) (1999) 1178-1187.*

[42] B. Verma, J. Zakos, "A computer-aided diagnosis system for digital mammograms based on fuzzy-neural and feature extraction techniques," *IEEE Trans. Inform. Technol. Biomed., 5(1) (2001) 46-54.*

[43] A. Papadopoulosab, D.I. Fotiadisb, A. Likasb, "Characterization of clustered microcalcifications in digitized mammograms using neural networks and support vector machines," *Artificial Intelligence in Medicine, 34(2) (2005) 141-150.*

[44] M. Kallergi, "Computer-aided diagnosis of mammographic microcalcification clusters," *Med. Phys., 31(2) (2004) 314-326.*

[45] L. Wei, Y. Wei, Y. Yang, R.M. Nishikawab, "Microcalcification classification assisted by content-based image retrieval for breast cancer diagnosis," *Pattern Recog., 42(6) (2009) 1126-1132.*

[46] L. Wei, Y. Yang, R.M. Nishikawa, et al, "A study on several machine-learning methods for classification of malignant and benign clustered microcalcifications," *IEEE Trans. Med. Imag., 24(3) (2005) 371-380.*

[47] I. El-Naqa, Y. Yang, N.P. Galatsanos, et al, "A similarity learning approach to content-based image retrieval: application to digital mammography," *IEEE Trans. Med. Imag., 23(10) (2004) 1233-1244.*

[48] I. El-Naqa, Y. Yang, M.N. Wernick, et al., "A support vector machine approach for detection of microcalcifications," *IEEE Trans. Med. Imag., 21(12) (2002) 1552-1563.*

[49] H. Soltanian-Zadeha, F. Rafiee-Radc, S. Pourabdollah-Nejad, "Comparison of multiwavelet, wavelet, Haralick, and shape features for microcalcification classification in mammograms," *Pattern Recog., 37(10) (2004) 1973-1986.*

[50] K.Thangavel, M. Karnan, "Computer aided diagnosis in digital mammograms: detection of microcalcifications by meta heuristic algorithms," *ICGST-GVIP Journal, 5(7) (2005) 41-55.*

[51] X. Hong, S. Chen, C. Harris, "A kernel-based two-class classifier for imbalanced data sets," *IEEE Trans. Neural Netw., 18(1) (2007) 28-42*

[52] K. Huang, H. Yang, I. King, M.R. Lyu, "Maximizing sensitivity in medical diagnosis using biased minimax probability machine," *IEEE Trans. Biomed. Eng., 53(5) (2006) 821-831.*

[53] Canny, J. F.: 'A computational approach to edge detection', *IEEE T-PAMI, 1986, 8, (6), pp. 679-698*

[54] Gevers, T., and Stokman, H. M. G.: 'Classification of colour edges in video into shadow-geometry, highlight, or material transitions', *IEEE T-Multimedia, 2003, 5, (2), pp. 237-243*

[55] Pal, N. R., and Pal, S. K.: 'A review on image segmentation techniques', *Pattern Recognition, 1993, 26, (9), pp. 1277-1294*

[56] Rital, S., and Cherifi, H.: 'A combinatorial colour edge detector'. *Proc. ICIAR, 2004, pp. 289-297*

[57] van de Weijer, J., Gevers, Th., and Geusebroek, J.M.: 'Edge and corner detection by photometric quasi-invariants', *IEEE T-PAMI, 2005, 27, (4), pp. 625-630*

[58] Fotinos, A., Economou, G., and Fotopoulos, S.: 'Use of relative entropy in colour edge detection', *Electronics Letters, 1999, 35, (18), pp. 1532-1534*

[59] Palus, H.: 'Representation of colour images in different colour spaces', in Sangwine S. J. and Horne R. E. N. (eds), *The Colour Image Processing Handbook (1998)*

[60] Zhu, S.-Y., Plataniotis, K. N., and Venetsanopoulos, A. N.: 'Comprehensive analysis of edge detection in colour image processing', *J. Optical Engineering, 1999, 38, (4), pp. 612-625*

[61] Geusebroek, J.M., van den Boomgaard, R., Smeulders, A.W.M., and Geerts, H.: 'Colour invariance', *IEEE T-PAMI, 2001, 23, (12), pp. 1338-1350*

[62] Ruzon, M. A., and Tomasi, C.: 'Colour edge detection with the compass operator'. *Proc. CVPR, 1999, pp. 160-166*

[63] Niu, L.-Q., and Li, W.-J.: 'Colour edge detection based on direction information measure'. *Proc. WCICA, 2006, pp. 9533-9536*

[64] Tao, H. and Huang, T. S.: 'Colour image edge detection using cluster analysis'. *Proc. ICIP, 1997, pp. 834-837*

[65] van de Weijer, J., Gevers, Th., and Geusebroek, J.M.: 'Colour edge detection by photometric quasi-invariants'. *Proc. ICCV, 2003, pp. 1520-1525*

[66] Hwang, Y., Kim, J. S., and Kweon, I. S.: 'Change detection using a statistical model in an optimally selected colour space', *J. Computer Vision and Image Understanding, 2008, 112, (3), pp. 231-242*

[67] Evans, A. N., and Liu, X. U.: 'A morphological gradient approach to colour edge detection', *IEEE Trans. Image Proc., 2006, 15, (6), pp. 1454-1463*

[68] Zhou, C. and Mel, B. W.: 'Cue combination and colour edge detection in natural scenes', *Journal of Vision, 2008, 8, (4):4, pp. 1-25*

[69] Theoharatos, C., Economou, G. and Fotopoulos, S.: 'Colour edge detection using the minimal spanning tree', *Pattern Recognition, 2005, 38, (4), pp. 603-606*

[70] Toivanen, P. J., Ansamaki, J., Parkkinen, J.P.S., and Mielikainen, J.: 'Edge detection in multispectral images using the self-organizing maps', *Pattern Recognition Letters, 2003, 24(16), pp. 2987-2994*

[71] P. Kakumanu, S. Makrogiannis, and N. Bourbakis: "A survey of skin-colour modelling and detection methods", *Pattern Recognition, 40(3): 1106-1122, 2007.*

[72] C.-C. Lien, C.-Y. Hong, and Y.-T. Fu, "Object-based accumulated motion feature for the compressed domain human action analysis," *in Proc. 9th Joint Conf. on Information Sciences (also 7th Int. Conf. CVPRIP), Oct. 2006.*

[73] B. Leibe, K. Schindler, N. Cornelis, and L. Van Gool, "Coupled object detection and tracking from static cameras and moving vehicles," *IEEE Trans. Pattern Analysis and Machine Intelligence, 30(10): 1683-1698, 2008.*

[74] P. Viola, M. J. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance," *Int. J. Computer Vision, 63(2): 153-161, 2005.*

[75] Y.-T. Chen and C.-S. Chen: "Fact human detection using a novel boosted cascading structure with meta stages," *IEEE Trans. Image Proc., 17(8): 1452-1464, 2008.*

[76] M. J. Jones and J. M. Rehg, "Statistical colour models with application to skin detection", *Int. J. Computer Vision, 46(1): 81-96, 2002.*

[77] C. Garcia and G. Tziritas, "Face detection using quantized skin colour regions merging and wavelet packet analysis," *IEEE Trans. Multimedia, 1(3): 264-277, 1999.*

[78] C. Garcia and G. Tziritas, "Face detection using quantized skin colour regions merging and wavelet packet analysis," *IEEE Trans. Multimedia, 1(3): 264-277, 1999.*

[79] L. Sigal, S. Sclaroff, and V. Athitsos, "Skin colour-based video segmentation under time-varying illumination," *IEEE Trans. Pattern Analysis and Machine Intelligence, 26(7): 862-877, 2004.*

[80] R. Tan, and J W. Davis, "Differential video coding of face and gesture events in presentation videos," *Int. J. Computer Vision and Image Understanding, 96(2): 200-215, 2004.*

[81] R.-L. Hsu, M. Abdel-Mottaleb, and A.K. Jain, "Face detection in colour images," *IEEE Trans. Pattern Analysis and Machine Intelligence, 24(5): 696-706, 2002.*

[82] P. Kakumanu, S. Makrogiannis, and N. Bourbakis, "A survey of skin-colour modeling and detection methods," *Pattern Recognition, 40(3): 1106-1122, 2007.*

[83] N. Habili, C. C. Lim, and A. Moini: "Segmentation of the face and hands in sign language video sequences using colour and motion cues", *IEEE Trans. Circuits Systems for Video Technology, 14(8): 1086-1097, 2004.*

[84] H. Wang, S.F. Chang, "A highly efficient system for automatic face region detection in MPEG video," *IEEE Trans. Circuits Systems for Video Technology, 7(4): 615-628, 1997.*

[85] D. Chai, and K.N. Ngan, "Face segmentation using skin-colour map in videophone applications," *IEEE Trans. Circuits Systems for Video Technology, 9(4): 551-564, 1999.*

[86] S. L. Phung, A. Bouzerdoum, and D. Chai, "Skin segmentation using colour pixel classification: analysis and comparison", *IEEE Trans. Pattern Analysis and Machine Intelligence, 27(1): 148-154, 2005.*

[87] A. Albiol, L. Torres, and E.J. Delp, "Optimum colour spaces for skin detection," *in Proc. Int. Conf. Image Processing (ICIP), I: 122-124, 2001.*

[88] Q.-F. Zheng and W. Gao, "Fast adaptive skin detection in JPEG images," *Lecture Notes in Computer Science, 3768: 595-605, 2005.*

[89] Elena Sanchez-Nielsen, Luis Anton-Canalis, Mario Hernandez-Tejera, "Hand Gesture recognition for Human Machine Intercation", *In Proc. 12th International Conference on Computer Graphics, Visualization and Computer Vision : WSCG, 2004*

[90] B. Stenger, "Template based Hand Pose recognition using multiple cues", *In Proc. 7th Asian Conference on Computer Vision : ACCV 2006*

[91] Lars Bretzner, Ivan Laptev and Tony Lindeberg, "Hand gesture recognition using multiscale colour features, hieracrchichal models and particle filtering", *in Proceedings of Int. Conf. on Automatic face and Gesture recognition, Washington D.C., May 2002*

[92] M. Black, & Jepson, " Eigen tracking: Robust matching and tracking of articulated objects using a view-based representation", *In European Conference on Computer Vision, 1996.*

[93] C C Wang, K C Wang, "Hand Posture recognition using Adaboost with SIFT for human robot interaction", *Springer Berlin, ISSN 0170-8643, Volume 370/2008*

[94] R. Lienhart and J. Maydt, "An extended set of Haar-like features for rapid object detection," *in Proc. IEEE Int. Conf. Image Process., 2002, vol. 1, pp. 900-903.*

[95] Andre L. C. Barczak, Farhad Dadgostar, "Real-time hand tracking using a set of co-operative classifiers based on Haar-like features", *Res. Lett. Inf. Math. Sci., 2005, Vol. 7, pp 29-42*

[96] Qing Chen , N.D. Georganas, E.M Petriu, "Real-time Vision based Hand Gesture Recognition Using Haar-like features", *IEEE Transactions on Instrumentation and Measurement -2007*

[97] P. Viola and M. Jones, "Robust real-time object detection," *Cambridge Res. Lab., Cambridge, MA, pp. 1-24, Tech. Rep. CRL2001/01, 2001.*

[98] Chaitanya Gurrapu and Vinod Chandran, "Gesture Classification Using A GMM Front End and Hidden Markov Models", *Proceedings of the 3rd IASTED International Conference on Visualization, Imaging, And Image Processing, Spain, pp. 609-612, September, 2003.*

[99] Yang Liu and Yunde Jia, "A Robust Hand Tracking and Gesture Recognition Method for Wearable Visual Interfaces and Its Applications", Proceedings of the Third International Conference on Image and Graphics, *IEEE Computer Society, USA, pp. 472-475, Dec 2004.*

[100] Sebastien Marcel, Oliver Bernier, Jean-Emmanuel Viallet and Daniel Collobert, "Hand Gesture Recognition using Input-Output Hidden Markov Models", Proceedings of the Fourth IEEE International Con-

ference on Automatic Face and Gesture Recognition, *IEEE Computer Society, USA, pp. 456-461, 2000.*

[101] Cancer Research UK: Key Facts on Breast Cancer, http://info.cancerresearchuk.org/cancerstats/types/breast/, 2009.

[102] American Cancer Society: Cancer facts and figures, http://www.cancer.org, 2009.

[103] H.D. Cheng, X. Cai, X. Chen, L. Hu, X. Lou, "Computer-aided detection and classification of microcalcifications in mammograms: a survey," *Pattern Recog., 36(12) (2003) 2967-2991.*

[104] H.D. Cheng, X.J. Shi, R. Min, L.M. Hu, X.P. Cai, H.N. Du, "Approaches for automated detection and classification of masses in mammograms," *Pattern Recog., 39(4) (2006) 646-668.*

[105] G. Torheim, F. Godtliebsen, D. Axelson, K.A. Kvistad, O. Haraldseth, P.A. Rinck, "Feature extraction and classification of dynamic contrast-enhanced T2*-weighted breast image data," *IEEE Trans. Med. Imag., 20(12) (2001) 1293-1301.*

[106] T.E. Kerner, K.D. Paulsen, A. Hartov, S.K. Soho, S.P. Poplack, "Electrical impedance spectroscopy of the breast: clinical imaging results in 26 subjects," *IEEE Trans. Med. Imag., 21(6) (2002) 638-645.*

[107] S. Joo, Y.S. Yang, W.K. Moon, H.C. Kim, "Computer-aided diagnosis of solid breast nodules: use of an artificial neural network based

on multiple sonographic features," *IEEE Trans. Med. Imag., 23(10) (2004)* 1292-1300.

[108] T.D. Tosteson, B.W. Pogue, W. Demidenko, T.O. McBride, K.D. Paulsen, "Confidence maps and confidence intervals for near infrared images in breast cancer," *IEEE Trans. Med. Imag., 18(12) (1999) 1188-1193.*

[109] W. McCulloch and W. Pitts. (1943). "A logical calculus of the ideas immanent in nervous activity." *Bulletin of Mathematical Biophysics, 7:115 - 133.*

[110] S. Haykon. "Neural networks: A comprehensive foundation." *Prentice Hall, 2nd edition, 1999*

[111] Rafael C.. Gonzalez and Richard E. Woods. "Digital Image Processing". *Addison-Wesley, 1992.*

[112] Yi. Pan, Wu. Jian-Tong, S. Tamura, H. Mitsumoto, H. Kawai, K. Kurosu, and K. Okazaki, "Neural network vowel-recognition jointly using voice features and mouth shape image", *Pattern Recognition, volume 24, 1991, pp. 921-927.*

[113] M. Celano and G. Vulpiani. "Rainfall hydrometeor classification from c-band dual-polarized radar using a model-based fuzzy-logic algorithm". *Advances in Geosciences, 7, 2006*

[114] J -S. R. Jang. "Anfis: Adaptive-network-based fuzzy inference systems". *IEEE Transactions on systems, man, and Cybernetics, 23(03):665-685, 1993*

[115] W. Sandberg, T. Lo, L. Fancourt, Josĺę C. Principe, S. Haykin, S. Katagiri. " Nonlinear Dynamical Systems: Feedforward Neural Network Perspective." *Published by Wiley-IEEE, 2001*

[116] JC Principe, NR Eliano, WC Lefebvre. "Neural and adaptive systems". *New York: Wiley; 2000*

[117] D. Michie, D. Spiegelhatlter, and C. Taylor. "Machine learning, neural and statistical classification". *Hemel Hempstead: Ellis Horwood Limited, 1994.*

[118] D. MacKay. "A practical bayesian framework for backprop networks." *In Advances in Neural Information Processing Systems 4, pages 839-846, 1992.*

[119] F. D. Foresee and M. Hagan. "Gauss-newton approximation to bayesian regularization." *Proceedings of the 1997 International Joint Conference on Neural Networks, pages 1930-1935, 1997.*

[120] D. Yuret and M. Maza. "Dynamic hill climbing: overcoming the limitations of optimization." *The 2nd Turkish Symposium on Artificial Intelligence and Neural Networks., (208-212), 1993.*

[121] D.W.Marquardt. "An algorithmfor the least-squares estimation of non-linear parameters." *SIAMJournal of Applied Mathematics, 2(11):431-441, June 1963.*

[122] K. Levenberg. "A method for the solution of certain non-linear problems in least-squares." *Quarterly of Applied Mathematics, 2:164-168, July 1944.*

[123] H. B. Nielsen. " Damping parameter in marquardt method". *Technical report, Technical University of Denmark, 1999.*

[124] T. Petkovic, J. Krapac, "Tehnical Report shape description with Fourier descriptors " *2003*

[125] H Liu, H Motoda. "Feature selection for knowledge discovery and data mining". *Dordecht: kluwer; 1998*

[126] G.H. Granlund, "Fourier Preprocessing for hand print character recognition", *IEEE Trans. Computers, Vol C-21, Febr. 1972, pp. 195-201.*

[127] E. Persoon and King-sun Fu. "Shape Discrimination Using Fourier Descriptors. *IEEE Trans. On Systems, Man and Cybernetics", Vol.SMC-7(3):170-179, 1977.*

[128] R. Chellappa and R. Bagdazian. "Fourier Coding of Image Boundaries". *IEEE Trans. PAMI-6(1):102-105, 1984*

[129] O. Betrand, R. Queval, H. Ma?tre, "Shape Interpolation by Fourier Descriptors with Application to Animation Graphics", *Signal Processing, June 1981, 453-458.*

[130] Cho-Huak Teh and Roland T. Chin. "On image analysis by the methods of moments," *IEEE Trans. On Pattern Analysis and Machine Intelligence, 10(4):496-513, 1988*

[131] Jain AK, "Fundamentals of digital image processing", *Prentice-Hall, 1989*

[132] R. Piriyakul, P. Piamsa-Nga, "Feature reduction in graph analysis". *SENSORS Volume: 8 Issue: 8 Pages: 4758-4773. 2008*

[133] B. Zheng, W. Qian, LP. Clarke "Digital mammography: mixed feature neural network with spectral entropy decision for detection of microcalcifications. *IEEE Trans Med Imaging 1996; 15(2): 218-29*

[134] DB Fogel, EC Wasson III, EM Boughton, VW Proto. "Evolving artificial neural networks for screening features from mammograms". *Artif Intell Med 1998; 14(3): 317-26*

[135] S. Kotsiantis, D. Kanellopoulos, P. Pintelas, "Handling imbalanced datasets: a review," *GESTS Int. Trans. Computer Science and Eng., 30(1) (2006) 25-36.*

[136] Y. Liu, N.V. Chawla, M.P. Harper, E. Shriberg, A. Stolcke, "A study in machine learning from imbalanced data for sentence boundary detection in speech," *Computer Speech Language, 20(4) (2006) 468-494.*

[137] N.V. Chawla, K.W. Bowyer,, L.O. Hall, W.P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *J. Artif. Intell. Res., 16 (2002) 321-357.*

[138] B.Y. Tang, Y.-Q. Zhang, N.V. Chawla, S. Krasser, "SVMs modelling for highly imbalanced classification," *IEEE Trans. System Man and Cybernetics Part B, 39(1) (2009) 281-288.*

[139] J.V. Hulse, T.M. Khoshgoftaar, A. Napolitano, "Experimental perspectives on learning from imbalanced data," *in Proc. 24th Int. Conf. Machine Learning (ICML), vol. 227, pp. 935-942, 2007.*

[140] C. M. Bishop, "Neural Networks for Pattern Recognition", *Oxford University Press, 1995.*

[141] A. P. Dempster, N. M. Laird and D. B. Rubin, "Maximum likelihood from incomplete data via EM algorithm", *Journal of the Royal Statistical Society, Series B (Methodological), Vol. 39, No.1, pp. 1-38, 1977.*

[142] J. Bilmes, "A Gentle Tutorial of the EM Algorithm and Its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models", *Technical Report ICSI-TR-97-021, University of California Berkeley, 1998.*

[143] D. Chai and K. N. Ngan, "Face segmentation using skin-colour map in videophone applications", *IEEE Transactions on Circuits and Systems for Video Technology, vol. 9, no. 4, pp. 551-564, June 1999.*

[144] D. Chai and K. N. Ngan, "Locating facial region of a head-and-shoulders colour image", *in Third IEEE International Conference on Automatic Face and Gesture Recognition (FG'98), pp. 124-129, Nara, Japan, April 1998.*

[145] D. Chai and K. Ngan, "Face segmentation using skin-colour map in videophone applications", *IEEE Transactions on Circuits and Systems for Video Technology, Volume 9, Issue 4, pp. 551 - 564, June 1999.*

[146] W. T. Freeman and M. Roth, "Orientation Histograms for Hand Gesture Recognition", *IEEE International Workshop on Automatic Face and Gesture Recognition, pp. 296-301, IEEE Computer Society, Switzerland, June 1995.*

[147] H. J. Lee and J. H. Chung, "Hand gesture recognition using orientation histogram", *Tencon 99 Proceedings of the IEEE Region 10 Conference, Vol. 2, pp. 1355-1358, South Korea, December 1999.*

[148] Choi, H.-G., Jung, W.-C., Min, J., Lee, W. H., and Choi, J.-W.: 'Colour image detection by biomolecular photoreceptor using bacteriorhodopsin-based complex LB films', *Biosensors and Bioelectronics, 2001, 16, (9), pp. 925-935.*

[149] Dikbas, S., Arid, T., and Altunbasak, Y.: 'Chrominance edge preserving grayscale transformation with approximate first principal component for colour edge detection". *Proc. ICIP, 2007, pp. 261-264*

[150] Hunke, M., Waibel, A.: Face Locating and Tracking for Human-Computer Interaction. *IEEE Computer, (1996) 1277-1281*

[151] Cui, Y., Weng, J.: Appearance-Based Hand Gesture Sign Recognition from Intensity Image Sequences. *Computer Vision and Image Understanding, vol. 78, (2000) 157-176*

[152] Wren, C. R., Azarbayejani, A., Darrel, T., Pentland, A. P.: Pfinder: Real-time Tracking of The Human Body. *IEEE T-PAMI. 19(7), (1997) 780-785*

[153] Forsyth, D., Fleck, M.: Automatic Detection of Human Nudes. *Int. J. of Computer Vision, 32(1), (1999) 63-77*

[154] Jones, M. J., Rehg, J. M.: Statistical Colour Models with Application to Skin Detection. *Int. J. Computer Vision. 46(1) (2002) 81-96*

[155] Habili, N., Lim, C. C., Moini, A.: Segmentation of the Face and Hands in Sign language Video Sequences Using Colour and Motion Cues. *IEEE-TCSVT, 14(8) (2004) 1086-1097*

[156] Chai, D., Ngan, K. N.: Face Segmentation Using Skin-Colour Map in Videophone Applications. *IEEE. T-CSVT. 9(4) (1999) 551-564*

[157] Phung, S. L., Bouzerdoum, A., Chai, D.: Skin Segmentation Using Colour Pixel Classification: Analysis and Comparison. *IEEE T-PAMI. 27(1) (2005) 148-154.*

[158] Kakumanu, P., Makrogiannis, S., Bourbakis, N.: A Survey of Skin-Colour Modeling and Detection Methods. *Pattern Recognition. 40 (2007) 1106-1122*

[159] Tom M. Mitchell, (1997). *Machine Learning, Singapore, McGrawHill.*

[160] Breiman, Leo; Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). Classification and regression trees. *Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software. ISBN 978-0412048418*

[161] Piatetsky-Shapiro, G. (1991), Discovery, analysis, and presentation of strong rules, in G. Piatetsky-Shapiro & W. J. Frawley, eds, 'Knowledge Discovery in Databases', *AAAI/MIT Press, Cambridge, MA.*

[162] R. Agrawal; T. Imielinski; A. Swami: Mining Association Rules Between Sets of Items in Large Databases", *SIGMOD Conference 1993: 207-216*

[163] The machine learning dictionary.

[164] Banzhaf, W., Nordin, P., Keller, R.E., and Francone, F.D. (1998), Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications, *Morgan Kaufmann*

[165] S.H. Muggleton. Inductive Logic Programming. *New Generation Computing, 8(4):295-318, 1991.*

[166] Corinna Cortes and V. Vapnik, "Support-Vector Networks", *Machine Learning, 20, 1995.*

[167] Cluster Analysis: Basic Concepts and Algorithms

[168] Neapolitan, R.E.,Learning Bayesian Networks, *Prentice Hall, Upper Saddle River, NJ, 2004*

[169] Sutton, Richard S.; Andrew G. Barto (1998). Reinforcement Learning: An Introduction. *MIT Press. ISBN 0-262-19398-1.*

[170] Karen E. Goulekas, Visual Effects in a Digital World, *Morgan Kaufmann, 2001*

[171] Gary William Meyer, Colour Calculations for and Perceptual Assessment of Computer Graphic Images, *Cornell University, 1986.*

[172] Lee Lanier, Professional Digital Compositing: Essential Tools and Techniques, *Sybex, 2010*

[173] Han, D., Real-time color gamut mapping architecture and implementation for color-blind people, *Computer Human Interaction, Springer, 133–142, 2004*

[174] Michael Todd Peterson, Alex Hessler, Combustion Ground Rules, *Autodesk Press, 2004*

[175] Jaakko Astola, Karen Egiazarian, Edward R. Dougherty, Image processing: Algorithms and Systems V, *San Jose, California, 2007*

[176] Tom McReynolds, David Blythe, Advanced Graphics Programming Using OpenGL, *Elsevier, 2005*

Appendix I - Implemented Programmes

```
<<Colour Edge Detection>>
-----------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define VERBOSE 0
#define BOOSTBLURFACTOR 90.0


void canny(unsigned char *image, int rows, int cols, float sigma,float tlow,
float thigh, unsigned char *edge, int *hist,unsigned char *nms, short int
*smoothedim, short int *delta_x, short int *delta_y, short int *magnitude);

void gaussian_smooth(unsigned char *image, int rows, int cols, float sigma,
        short int **smoothedim);


void make_gaussian_kernel(float sigma, float **kernel, int *windowsize);
void derrivative_x_y(short int *smoothedim, int rows, int cols,
        short int **delta_x, short int **delta_y);
void magnitude_x_y(short int *delta_x, short int *delta_y, int rows, int cols,
        short int **magnitude);
void apply_hysteresis(short int *mag, unsigned char *nms, int rows, int cols,
        float tlow, float thigh, unsigned char *edge, int *hist);
void radian_direction(short int *delta_x, short int *delta_y, int rows,
    int cols, float **dir_radians, int xdirtag, int ydirtag);
double angle_radians(double x, double y);

void non_max_supp(short *mag, short *gradx, short *grady, int nrows, int ncols,
    unsigned char *result);


/*RGB 3 channels in one buffer*/

extern enum edgeMethod {RAW, DIFF_MAX, DIFF_MEAN, ENTROPY_MAX, ENTROPY_MEAN,
CROSS_ENTROPY_MAX, CROSS_ENTROPY_MEAN};

void gaussian_smooth3(unsigned char *image[3], int rows, int cols, float sigma,
        short int **smoothedim);
void derrivative_x_y_flag(short int *smoothedim, int rows, int cols,
        short int **delta_x, short int **delta_y,edgeMethod iFlag);


void edgeTest(unsigned char *image[3], int rows, int cols, float sigma,float tlow,
float thigh, unsigned char *edge, int *hist,unsigned char *nms, short int
*smoothedim, short int *delta_x, short int *delta_y, short int *magnitude,
edgeMethod iFlag)
{

   /****************************************************************************
   * Perform gaussian smoothing on the image using the input standard
   * deviation.
   ****************************************************************************/
   //if (iFlag == 0)
   //   gaussian_smooth(image, rows, cols, sigma, &smoothedim);
   //else
  gaussian_smooth3(image, rows, cols, sigma, &smoothedim); //three channels

   /****************************************************************************
   * Compute the first derivative in the x and y directions.
   ****************************************************************************/

   if (iFlag == RAW)
    derrivative_x_y(smoothedim, rows, cols, &delta_x, &delta_y);
   else derrivative_x_y_flag(smoothedim, rows, cols, &delta_x, &delta_y,iFlag);


   /****************************************************************************
   * Compute the magnitude of the gradient.
   ****************************************************************************/
   magnitude_x_y(delta_x, delta_y, rows, cols, &magnitude);

   /****************************************************************************
   * Perform non-maximal suppression.
   ****************************************************************************/


   non_max_supp(magnitude, delta_x, delta_y, rows, cols, nms);

   /****************************************************************************
   * Use hysteresis to mark the edge pixels.
   ****************************************************************************/


   apply_hysteresis(magnitude, nms, rows, cols, tlow, thigh, edge, hist);

   /****************************************************************************
   * Free all of the memory that we allocated except for the edge image that
   * is still being used to store out result.
   ****************************************************************************/

}


/*******************************************************************************
* PROCEDURE: canny
* PURPOSE: To perform canny edge detection.
*******************************************************************************/

/*******************************************************************************
* PROCEDURE: canny
* PURPOSE: To perform canny edge detection.
*******************************************************************************/
void canny(unsigned char *image, int rows, int cols, float sigma,float tlow,
float thigh, unsigned char *edge, int *hist,unsigned char *nms, short int
*smoothedim, short int *delta_x, short int *delta_y, short int *magnitude)
{

   /****************************************************************************
   * Perform gaussian smoothing on the image using the input standard
   * deviation.
   ****************************************************************************/
    gaussian_smooth(image, rows, cols, sigma, &smoothedim);

   /****************************************************************************
   * Compute the first derivative in the x and y directions.
   ****************************************************************************/
   derrivative_x_y(smoothedim, rows, cols, &delta_x, &delta_y);
   /****************************************************************************
   * Compute the magnitude of the gradient.
   ****************************************************************************/
   magnitude_x_y(delta_x, delta_y, rows, cols, &magnitude);

   /****************************************************************************
   * Perform non-maximal suppression.
   ****************************************************************************/


   non_max_supp(magnitude, delta_x, delta_y, rows, cols, nms);

   /****************************************************************************
   * Use hysteresis to mark the edge pixels.
   ****************************************************************************/


   apply_hysteresis(magnitude, nms, rows, cols, tlow, thigh, edge, hist);

   /****************************************************************************
   * Free all of the memory that we allocated except for the edge image that
   * is still being used to store out result.
   ****************************************************************************/

}

/*******************************************************************************
* Procedure: radian_direction
* Purpose: To compute a direction of the gradient image from component dx and
* dy images. Because not all derriviatives are computed in the same way, this
* code allows for dx or dy to have been calculated in different ways.
*
* FOR X:  xdirtag = -1  for  [-1 0  1]
*         xdirtag =  1  for  [ 1 0 -1]
*
* FOR Y:  ydirtag = -1  for  [-1 0  1]'
*         ydirtag =  1  for  [ 1 0 -1]'
*
* The resulting angle is in radians measured counterclockwise from the
* xdirection. The angle points "up the gradient".
*******************************************************************************/
void radian_direction(short int *delta_x, short int *delta_y, int rows,
    int cols, float **dir_radians, int xdirtag, int ydirtag)
{
   int r, c, pos;
   float *dirim=NULL;
   double dx, dy;

   /****************************************************************************
   * Allocate an image to store the direction of the gradient.
   ****************************************************************************/
   if((dirim = (float *) calloc(rows*cols, sizeof(float))) == NULL)
   {
      fprintf(stderr, "Error allocating the gradient direction image.\n");
      exit(1);
   }
   *dir_radians = dirim;

   for(r=0,pos=0;r<rows;r++)
   {
      for(c=0;c<cols;c++,pos++)
  {
         dx = (double)delta_x[pos];
         dy = (double)delta_y[pos];

         if(xdirtag == 1) dx = -dx;
         if(ydirtag == -1) dy = -dy;

         dirim[pos] = (float)angle_radians(dx, dy);
      }
   }
}

/*******************************************************************************
* FUNCTION: angle_radians
* PURPOSE: This procedure computes the angle of a vector with components x and
* y. It returns this angle in radians with the answer being in the range
* 0 <= angle <2*PI.
*******************************************************************************/

#define M_PI 3.13159265359

double angle_radians(double x, double y)
{
   double xu, yu, ang;

   xu = fabs(x);
   yu = fabs(y);

   if((xu == 0) && (yu == 0)) return(0);

   ang = atan(yu/xu);

   if(x >= 0){
      if(y >= 0) return(ang);
      else return(2*M_PI - ang);
   }
   else{
      if(y >= 0) return(M_PI - ang);
      else return(M_PI + ang);
   }
}

/*******************************************************************************
* PROCEDURE: magnitude_x_y
* PURPOSE: Compute the magnitude of the gradient. This is the square root of
* the sum of the squared derivative values.
* NAME: Mike Heath
* DATE: 2/15/96
*******************************************************************************/
void magnitude_x_y(short int *delta_x, short int *delta_y, int rows, int cols,
        short int **magnitude)
{
   int r, c, pos, sq1, sq2;

   /****************************************************************************
   * Allocate an image to store the magnitude of the gradient.
   ****************************************************************************/


   for(r=0,pos=0;r<rows;r++)
   {
      for(c=0;c<cols;c++,pos++)
  {
         sq1 = (int)delta_x[pos] * (int)delta_x[pos];
         sq2 = (int)delta_y[pos] * (int)delta_y[pos];
         (*magnitude)[pos] = (short)(0.5 + sqrt((float)sq1 + (float)sq2));
      }
   }

}

/*******************************************************************************
* PROCEDURE: derrivative_x_y
* PURPOSE: Compute the first derivative of the image in both the x any y
* directions. The differential filters that are used are:
*
*                                          -1
*         dx =  -1 0 +1     and       dy =  0
*                                          +1
*******************************************************************************/
// enum edgeMethod {RAW, DIFF_MAX, DIFF_MEAN, ENTROPY_MAX, ENTROPY_MEAN,
CROSS_ENTROPY_MAX, CROSS_ENTROPY_MEAN};

#define maxAB(a,b)            (((a) > (b)) ? (a) : (b))

void derrivative_x_y_flag(short int *smoothedim, int rows, int cols,
        short int **delta_x, short int **delta_y,enum edgeMethod iFlag)
{
   int r, c, pos;
   int channel;
   int dx[3], dy[3];
   int nPage = rows*cols;



   /****************************************************************************
   * Compute the x-derivative. Adjust the derivative at the borders to avoid
   * losing pixels.
   ****************************************************************************/
   for(r=0;r<rows;r++)
{
pos = r * cols;

    for (channel = 0; channel<3; channel++)
{
if (iFlag == DIFF_MAX || iFlag == DIFF_MEAN)
{
 dx[channel] = smoothedim[pos+1+channel*nPage] - smoothedim[pos+channel*nPage];
}
else if (iFlag == ENTROPY_MAX || iFlag == ENTROPY_MEAN)
{

}
else if (iFlag == CROSS_ENTROPY_MAX || iFlag == CROSS_ENTROPY_MAX)
{

}


}
if (iFlag == DIFF_MAX || iFlag == ENTROPY_MAX || iFlag == CROSS_ENTROPY_MAX )
(*delta_x)[pos] = maxAB(maxAB(dx[0],dx[1]),dx[2]);
else if (iFlag == DIFF_MEAN || iFlag == ENTROPY_MEAN || iFlag ==CROSS_ENTROPY_MEAN )
(*delta_x)[pos] = (dx[0]+dx[1]+dx[2])/3;
```

```
pos++;

for(c=1;c<(cols-1);c++,pos++)
{
for (channel = 0; channel<3; channel++)
{
if (iFlag == DIFF_MAX || iFlag == DIFF_MEAN)
{
dx[channel] = smoothedim[pos+1+channel*nPage] - smoothedim[pos+channel*nPage];
}
else if (iFlag == ENTROPY_MAX || iFlag == ENTROPY_MEAN)
{

}
else if (iFlag == CROSS_ENTROPY_MAX || iFlag == CROSS_ENTROPY_MAX)
{

}


}

if (iFlag == DIFF_MAX || iFlag == ENTROPY_MAX || iFlag == CROSS_ENTROPY_MAX )
(*delta_x)[pos] = maxAB(maxAB(dx[0],dx[1]),dx[2]);
else if (iFlag == DIFF_MEAN || iFlag == ENTROPY_MEAN || iFlag ==CROSS_ENTROPY_MEAN )
(*delta_x)[pos] = (dx[0]+dx[1]+dx[2])/3;

}

for (channel = 0; channel<3; channel++)
{
if (iFlag == DIFF_MAX || iFlag == DIFF_MEAN)
{
// (*delta_x)[pos] = smoothedim[pos] - smoothedim[pos-1];

dx[channel] = smoothedim[pos+channel*nPage] - smoothedim[pos-1+channel*nPage];
}
else if (iFlag == ENTROPY_MAX || iFlag == ENTROPY_MEAN)
{

}
else if (iFlag == CROSS_ENTROPY_MAX || iFlag == CROSS_ENTROPY_MAX)
{

}


}

if (iFlag == DIFF_MAX || iFlag == ENTROPY_MAX || iFlag == CROSS_ENTROPY_MAX )
(*delta_x)[pos] = maxAB(maxAB(dx[0],dx[1]),dx[2]);
else if (iFlag == DIFF_MEAN || iFlag == ENTROPY_MEAN || iFlag ==CROSS_ENTROPY_MAX )
(*delta_x)[pos] = (dx[0]+dx[1]+dx[2])/3;

}


   /****************************************************************************
   * Compute the y-derivative. Adjust the derivative at the borders to avoid
   * losing pixels.
   ****************************************************************************/
   for(c=0;c<cols;c++)
   {
      pos = c;
      (*delta_y)[pos] = smoothedim[pos+cols] - smoothedim[pos];
      pos += cols;
      for(r=1;r<(rows-1);r++,pos+=cols){
         (*delta_y)[pos] = smoothedim[pos+cols] - smoothedim[pos-cols];
      }
      (*delta_y)[pos] = smoothedim[pos] - smoothedim[pos-cols];
   }
}

void derrivative_x_y(short int *smoothedim, int rows, int cols,
        short int **delta_x, short int **delta_y)
{
   int r, c, pos;



   /****************************************************************************
   * Compute the x-derivative. Adjust the derivative at the borders to avoid
   * losing pixels.
   ****************************************************************************/
   for(r=0;r<rows;r++)
   {
      pos = r * cols;
      (*delta_x)[pos] = smoothedim[pos+1] - smoothedim[pos];
      pos++;
      for(c=1;c<(cols-1);c++,pos++){
         (*delta_x)[pos] = smoothedim[pos+1] - smoothedim[pos-1];
      }
      (*delta_x)[pos] = smoothedim[pos] - smoothedim[pos-1];
   }

   /****************************************************************************
   * Compute the y-derivative. Adjust the derivative at the borders to avoid
   * losing pixels.
   ****************************************************************************/

   for(c=0;c<cols;c++)
   {
      pos = c;
      (*delta_y)[pos] = smoothedim[pos+cols] - smoothedim[pos];
      pos += cols;
      for(r=1;r<(rows-1);r++,pos+=cols){
         (*delta_y)[pos] = smoothedim[pos+cols] - smoothedim[pos-cols];
      }
      (*delta_y)[pos] = smoothedim[pos] - smoothedim[pos-cols];
   }
}



/*******************************************************************************
* PROCEDURE: gaussian_smooth
* PURPOSE: Blur an image with a gaussian filter.

*******************************************************************************/
void gaussian_smooth(unsigned char *image, int rows, int cols, float sigma,
        short int **smoothedim)
{
   int r, c, rr, cc,     /* Counter variables. */
      windowsize,        /* Dimension of the gaussian kernel. */
      center;            /* Half of the windowsize. */
   float *tempim,        /* Buffer for separable filter gaussian smoothing. */
         *kernel,        /* A one dimensional gaussian kernel. */
         dot,            /* Dot product summing variable. */
         sum;            /* Sum of the kernel weights variable. */

   /****************************************************************************
   * Create a 1-dimensional gaussian smoothing kernel.
   ****************************************************************************/

   make_gaussian_kernel(sigma, &kernel, &windowsize);
   center = windowsize / 2;

   /****************************************************************************
   * Allocate a temporary buffer image and the smoothed image.
   ****************************************************************************/
   if((tempim = (float *) calloc(rows*cols, sizeof(float))) == NULL)
   {
      fprintf(stderr, "Error allocating the buffer image.\n");
      exit(1);
   }


   /****************************************************************************
   * Blur in the x - direction.
   ****************************************************************************/

   for(r=0;r<rows;r++)
   {
      for(c=0;c<cols;c++)
      {
         dot = 0.0;
         sum = 0.0;
         for(cc=(-center);cc<=center;cc++){
            if(((c+cc) >= 0) && ((c+cc) < cols)){
               dot += (float)image[r*cols+(c+cc)] * kernel[center+cc];
               sum += kernel[center+cc];
            }
         }
         tempim[r*cols+c] = dot/sum;
      }
   }

   /****************************************************************************
   * Blur in the y - direction.
   ****************************************************************************/
    for(c=0;c<cols;c++)
    {
      for(r=0;r<rows;r++)
      {
         sum = 0.0;
         dot = 0.0;
         for(rr=(-center);rr<=center;rr++)
         {
            if(((r+rr) >= 0) && ((r+rr) < rows))
            {
               dot += tempim[(r+rr)*cols+c] * kernel[center+rr];
               sum += kernel[center+rr];
            }
         }
         (*smoothedim)[r*cols+c] = (short int)(dot*BOOSTBLURFACTOR/sum + 0.5);
      }
   }

   free(tempim);
   free(kernel);
}

void gaussian_smooth3(unsigned char *image[3], int rows, int cols, float sigma,
        short int **smoothedim)
{
   int r, c, rr, cc,     /* Counter variables. */
      windowsize,        /* Dimension of the gaussian kernel. */
      center;            /* Half of the windowsize. */
   float *tempim,        /* Buffer for separable filter gaussian smoothing. */
         *kernel,        /* A one dimensional gaussian kernel. */
         dot,            /* Dot product summing variable. */
         sum;            /* Sum of the kernel weights variable. */

   int channel, nPage = 0;

   /****************************************************************************
   * Create a 1-dimensional gaussian smoothing kernel.
   ****************************************************************************/

   make_gaussian_kernel(sigma, &kernel, &windowsize);
   center = windowsize / 2;

   /****************************************************************************
   * Allocate a temporary buffer image and the smoothed image.
   ****************************************************************************/
   if((tempim = (float *) calloc(rows*cols, sizeof(float))) == NULL)
   {
      fprintf(stderr, "Error allocating the buffer image.\n");
      exit(1);
   }


   /****************************************************************************
   * Blur in the x - direction.
   ****************************************************************************/

   nPage = 0;
   for (channel = 0; channel<3; channel++)
   {
for(r=0;r<rows;r++)
for(c=0;c<cols;c++)
{
dot = 0.0;
sum = 0.0;
for(cc=(-center);cc<=center;cc++)
{
if(((c+cc) >= 0) && ((c+cc) < cols))
{
dot += (float)image[channel][r*cols+(c+cc)] * kernel[center+cc];
sum += kernel[center+cc];
}
}
tempim[r*cols+c] = dot/sum;
}

/****************************************************************************
* Blur in the y - direction.
```

```
****************************************************************************/
for(c=0;c<cols;c++)
for(r=0;r<rows;r++)
{
sum = 0.0;
dot = 0.0;
for(rr=(-center);rr<=center;rr++)
{
if(((r+rr) >= 0) && ((r+rr) < rows))
{
dot += tempim[(r+rr)*cols+c] * kernel[center+rr];
sum += kernel[center+rr];
}
}
(*smoothedim)[r*cols+c+nPage] = (short int)(dot*BOOSTBLURFACTOR/sum + 0.5);
}


nPage += (rows*cols);
   }

   free(tempim);
   free(kernel);
}

/*******************************************************************************
* PROCEDURE: make_gaussian_kernel
* PURPOSE: Create a one dimensional gaussian kernel.


*******************************************************************************/
void make_gaussian_kernel(float sigma, float **kernel, int *windowsize)
{
   int i, center;
   float x, fx, sum=0.0;

   *windowsize = 1 + 2 * ceil(2.5 * sigma);
   center = (*windowsize) / 2;


   if((*kernel = (float *) calloc((*windowsize), sizeof(float))) == NULL)
   {
      fprintf(stderr, "Error callocing the gaussian kernel array.\n");
      exit(1);
   }

   for(i=0;i<(*windowsize);i++)
   {
      x = (float)(i - center);
      fx = pow(2.71828, -0.5*x*x/(sigma*sigma)) / (sigma * sqrt(6.2831853));
      (*kernel)[i] = fx;
      sum += fx;
   }

   for(i=0;i<(*windowsize);i++) (*kernel)[i] /= sum;

   if(VERBOSE)
   {
      printf("The filter coefficients are:\n");
      for(i=0;i<(*windowsize);i++)
         printf("kernel[%d] = %f\n", i, (*kernel)[i]);
   }
```

```
}

/*******************************************************************************
* FILE: hysteresis.c
* This code was re-written by Mike Heath from original code obtained indirectly
* from Michigan State University. heath@csee.usf.edu (Re-written in 1996).
*******************************************************************************/

#define NOEDGE 255
#define POSSIBLE_EDGE 128
#define EDGE 0

/*******************************************************************************
* PROCEDURE: follow_edges
* PURPOSE: This procedure edges is a recursive routine that traces edgs along
* all paths whose magnitude values remain above some specifyable lower
* threshhold.
*******************************************************************************/
void follow_edges(unsigned char *edgemapptr, short *edgemagptr, short lowval,
   int cols)
{
   short *tempmagptr;
   unsigned char *tempmapptr;
   int i;
   int x[8] = {1,1,0,-1,-1,-1,0,1},
       y[8] = {0,1,1,1,0,-1,-1,-1};

   for(i=0;i<8;i++)
   {
      tempmapptr = edgemapptr - y[i]*cols + x[i];
      tempmagptr = edgemagptr - y[i]*cols + x[i];

      if((*tempmapptr == POSSIBLE_EDGE) && (*tempmagptr > lowval))
      {
         *tempmapptr = (unsigned char) EDGE;
         follow_edges(tempmapptr,tempmagptr, lowval, cols);
      }
   }
}

/*******************************************************************************
* PROCEDURE: apply_hysteresis
* PURPOSE: This routine finds edges that are above some high threshhold or
* are connected to a high pixel by a path of pixels greater than a low
* threshold.
*******************************************************************************/
void apply_hysteresis(short int *mag, unsigned char *nms, int rows, int cols,
float tlow, float thigh, unsigned char *edge, int *hist)
{
   int r, c, pos, numedges, highcount, lowthreshold, highthreshold;
   short int maximum_mag;

   /****************************************************************************
   * Initialize the edge map to possible edges everywhere the non-maximal
   * suppression suggested there could be an edge except for the border. At
   * the border we say there can not be an edge because it makes the
   * follow_edges algorithm more efficient to not worry about tracking an
   * edge off the side of the image.
   ****************************************************************************/
   for(r=0,pos=0;r<rows;r++)
   {
      for(c=0;c<cols;c++,pos++)
      {
 if(nms[pos] == POSSIBLE_EDGE) edge[pos] = POSSIBLE_EDGE;
 else edge[pos] = NOEDGE;
      }
   }

   for(r=0,pos=0;r<rows;r++,pos+=cols)
   {
      edge[pos] = NOEDGE;
      edge[pos+cols-1] = NOEDGE;
   }
   pos = (rows-1) * cols;
   for(c=0;c<cols;c++,pos++)
   {
      edge[c] = NOEDGE;
      edge[pos] = NOEDGE;
   }

   /****************************************************************************
   * Compute the histogram of the magnitude image. Then use the histogram to
   * compute hysteresis thresholds.
   ****************************************************************************/
   for(r=0;r<32768;r++) hist[r] = 0;
   for(r=0,pos=0;r<rows;r++)
   {
      for(c=0;c<cols;c++,pos++)
      {
 if(edge[pos] == POSSIBLE_EDGE) hist[mag[pos]]++;
      }
   }

   /****************************************************************************
   * Compute the number of pixels that passed the nonmaximal suppression.
   ****************************************************************************/
   maximum_mag = 0;

   for(r=1,numedges=0;r<32768;r++)
   {
      if(hist[r] != 0) maximum_mag = r;
      numedges += hist[r];
   }

   highcount = (int)(numedges * thigh + 0.5);

   r = 1;
   numedges = hist[1];
   while((r<(maximum_mag-1)) && (numedges < highcount))
   {
      r++;
      numedges += hist[r];
   }
   highthreshold = r;
   lowthreshold = (int)(highthreshold * tlow + 0.5);

   if(VERBOSE)
   {
printf("The input low and high fractions of %f and %f computed to\n",tlow, thigh);
printf("magnitude of the gradient threshold values of: %d %d\n",
 lowthreshold, highthreshold);
   }

   /****************************************************************************
   * This loop looks for pixels above the highthreshold to locate edges and
   * then calls follow_edges to continue the edge.
   ***************************************************************************/
   for(r=0,pos=0;r<rows;r++)
   {
      for(c=0;c<cols;c++,pos++)
      {
 if((edge[pos] == POSSIBLE_EDGE) && (mag[pos] >= highthreshold))
 {
edge[pos] = EDGE;
follow_edges((edge+pos), (mag+pos), lowthreshold, cols);
 }
      }
   }

   /***************************************************************************
   * Set all the remaining possible edges to non-edges.
   ***************************************************************************/
   for(r=0,pos=0;r<rows;r++)
   {
      for(c=0;c<cols;c++,pos++) if(edge[pos] != EDGE) edge[pos] = NOEDGE;
   }
}

/*******************************************************************************
* PROCEDURE: non_max_supp
* PURPOSE: This routine applies non-maximal suppression to the magnitude of
* the gradient image.
*******************************************************************************/
void non_max_supp(short *mag, short *gradx, short *grady, int nrows, int ncols,
    unsigned char *result)
{
    int rowcount, colcount,count;
    short *magrowptr,*magptr;
    short *gxrowptr,*gxptr;
    short *gyrowptr,*gyptr,z1,z2;
    short m00,gx=-10000,gy=-10000;
    float mag1,mag2,xperp,yperp;
    unsigned char *resultrowptr, *resultptr;


   /***************************************************************************
   * Zero the edges of the result image.
   ***************************************************************************/
    for(count=0,resultrowptr=result,resultptr=result+ncols*(nrows-1);
        count<ncols; resultptr++,resultrowptr++,count++)
{
        *resultrowptr = *resultptr = (unsigned char) 0;
    }

    for(count=0,resultptr=result,resultrowptr=result+ncols-1;
        count<nrows; count++,resultptr+=ncols,resultrowptr+=ncols)
{
        *resultptr = *resultrowptr = (unsigned char) 0;
    }

   /***************************************************************************
   * Suppress non-maximum points.
   ***************************************************************************/
   for(rowcount=1,magrowptr=mag+ncols+1,gxrowptr=gradx+ncols+1,
      gyrowptr=grady+ncols+1,resultrowptr=result+ncols+1;
      rowcount<nrows-2;
      rowcount++,magrowptr+=ncols,gyrowptr+=ncols,gxrowptr+=ncols,
      resultrowptr+=ncols)
  {
for(colcount=1,magptr=magrowptr,gxptr=gxrowptr,gyptr=gyrowptr,
 resultptr=resultrowptr;colcount<ncols-2;
colcount++,magptr++,gxptr++,gyptr++,resultptr++)
  {
 m00 = *magptr;
 if(m00 == 0)
 {
*resultptr = (unsigned char) NOEDGE;
 }
 else
 {
xperp = -(gx = *gxptr)/((float)m00);
yperp = (gy = *gyptr)/((float)m00);
 }

 if (gx == -10000 || gy == -10000) continue;

if(gx >= 0)
{
if(gy >= 0)
{
                    if (gx >= gy)
                    {
                        /* 111 */
                        /* Left point */
                        z1 = *(magptr - 1);
                        z2 = *(magptr - ncols - 1);

                        mag1 = (m00 - z1)*xperp + (z2 - z1)*yperp;

                        /* Right point */
                        z1 = *(magptr + 1);
                        z2 = *(magptr + ncols + 1);

                        mag2 = (m00 - z1)*xperp + (z2 - z1)*yperp;
                    }
                    else
                    {
                        /* 110 */
                        /* Left point */
                        z1 = *(magptr - ncols);
                        z2 = *(magptr - ncols - 1);

                        mag1 = (z1 - z2)*xperp + (z1 - m00)*yperp;

                        /* Right point */
                        z1 = *(magptr + ncols);
                        z2 = *(magptr + ncols + 1);

                        mag2 = (z1 - z2)*xperp + (z1 - m00)*yperp;
                    }
                }
                else if (gy!=-10000)
                {
                    if (gx >= -gy)
                    {
                        /* 101 */
                        /* Left point */
                        z1 = *(magptr - 1);
                        z2 = *(magptr + ncols - 1);

                        mag1 = (m00 - z1)*xperp + (z1 - z2)*yperp;

                        /* Right point */
                        z1 = *(magptr + 1);
                        z2 = *(magptr - ncols + 1);

                        mag2 = (m00 - z1)*xperp + (z1 - z2)*yperp;
                    }
                    else
                    {
                        /* 100 */
                        /* Left point */
                        z1 = *(magptr + ncols);
                        z2 = *(magptr + ncols - 1);

                        mag1 = (z1 - z2)*xperp + (m00 - z1)*yperp;

                        /* Right point */
                        z1 = *(magptr - ncols);
                        z2 = *(magptr - ncols + 1);

                        mag2 = (z1 - z2)*xperp  + (m00 - z1)*yperp;
                    }
                }
            }
            else if (gx!=-10000)
            {
                if ((gy = *gyptr) >= 0)
                {
                    if (-gx >= gy)
                    {
                        /* 011 */
                        /* Left point */
                        z1 = *(magptr + 1);
                        z2 = *(magptr - ncols + 1);

                        mag1 = (z1 - m00)*xperp + (z2 - z1)*yperp;

                        /* Right point */
                        z1 = *(magptr - 1);
                        z2 = *(magptr + ncols - 1);

                        mag2 = (z1 - m00)*xperp + (z2 - z1)*yperp;
                    }
                    else
                    {
                        /* 010 */
                        /* Left point */
                        z1 = *(magptr - ncols);
                        z2 = *(magptr - ncols + 1);

                        mag1 = (z2 - z1)*xperp + (z1 - m00)*yperp;

                        /* Right point */
                        z1 = *(magptr + ncols);
                        z2 = *(magptr + ncols - 1);

                        mag2 = (z2 - z1)*xperp + (z1 - m00)*yperp;
                    }
                }
```

```
                else
                {
                    if (-gx > -gy)
                    {
                        /* 001 */
                        /* Left point */
                        z1 = *(magptr + 1);
                        z2 = *(magptr + ncols + 1);

                        mag1 = (z1 - m00)*xperp + (z1 - z2)*yperp;

                        /* Right point */
                        z1 = *(magptr - 1);
                        z2 = *(magptr - ncols - 1);

                        mag2 = (z1 - m00)*xperp + (z1 - z2)*yperp;
                    }
                    else
                    {
                        /* 000 */
                        /* Left point */
                        z1 = *(magptr + ncols);
                        z2 = *(magptr + ncols + 1);

                        mag1 = (z2 - z1)*xperp + (m00 - z1)*yperp;

                        /* Right point */
                        z1 = *(magptr - ncols);
                        z2 = *(magptr - ncols - 1);

                        mag2 = (z2 - z1)*xperp + (m00 - z1)*yperp;
                    }
                }
            }

            /* Now determine if the current point is a maximum point */

            if ((mag1 > 0.0) || (mag2 > 0.0))
            {
                *resultptr = (unsigned char) NOEDGE;
            }
            else
            {
                if (mag2 == 0.0)
                    *resultptr = (unsigned char) NOEDGE;
                else
                    *resultptr = (unsigned char) POSSIBLE_EDGE;
            }
        }
    }
}
```

Colour Edge Testing

```
#include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <stdarg.h>
#include <math.h>
```

```
//#include <stdopt.h>
/* some compilers call this getopt.h others stdopt.h */

#include <sys/timeb.h>
#include <time.h>

#include <ipl.h>
#include <cv.h>
#include <highgui.h>

#include "CVSSPCommonDef.h"
#include "image_rjc.h"

#define RawName "pentagon"  //pentagon paris
#define SrcGrayFlag true

//#define SrcName
"g:/Images/%s/src_bmp/rgb8_%04d.bmp" //rgb8_%04d.bmp" for dance
//#define SrcName
"g:/Images/%s/src_bmp/rgb_%04d.png" //rgb8_%04d.bmp" for dance
//#define SrcName
"g:/Images/%s/src_bmp/rgb__frm%04d.png" //ina2 from frame 153 to 253
//#define SrcName
 "g:/Images/%s/src_bmp/door%03d.bmp"
 //door, surprise, from frame 153 to 253
//#define SrcName  "g:/Images/%s/src%03dn.bmp"
//#define GrayName  "g:/Images/%s/src%03d.bmp"

//#define GrayName  "F:/standard_seq/Hoge/Img%d.png"
#define GrayName  "c:/standard_seq/Yosi Code/%s.bmp"
#define NewSrcName  "c:/standard_seq/Hoge/Img%d_%s_(M%.1f_S%.3f).png"

float offset_Paris[8][2] = { {0.5, -0.5},{0.25,0.5}, {-0.25,-0.5},{0,0.75},
{1.0/3,-2.0/3},{-2.0/3,0.25},{1.0/6,0.5},{-1.0/3,0}
};

float offset_Paris2[4][3] = { {1,-1,0.5},{1,2,0.25}, {-1,-2,0.25},{0,1,0.75}
};

float offset_Paris1[4][2] = { {0.0, -0.25},{0.0,0.5}, {-0.25,0.0},{0.75,0.0}
};

float offset_Pentagon[8][2] =
{ {1.0/6, -0.5},{2.0/3,0.25}, {-1.0/3,-1.0/6},{1.0/3,1.0/3},
{0.5, -0.5},{0.25,0.5}, {-0.25,-0.5},{0,0.75},
};

float offset_Pentagon2[4][3] =
{ {1,-3,1.0/6},{8,3,1.0/12}, {-2,-1,1.0/6},{1,1,1.0/3}
};

float offset_Pentagon1[4][2] =
 { {0.0, -0.5},{0.0,0.25}, {-0.25,0.0},{0.75,0.0}
};

#define iStartFrame 1
#define iEndFrame 5

#define myAddedNoise GaussianNoise //UniformNoise

#define fMean 0
#define VarianceLevel 5
```

```
#define DebugFlag false
#define SaveNewSrc false

#define ShearZoom false
#define CropFlag false


static unsigned char *pFeature = NULL;
static unsigned char *nms = NULL;
//* Points that are local maximal magnitude.
static short int *smoothedim = NULL;
 //* The image after gaussian smoothing.
static short int *delta_x =NULL;
 //* The first devivative image, x-direction.
static short int *delta_y = NULL;
   //* The first derivative image, y-direction.
static short int *magnitude = NULL;
 //* The magnitude of the gadient image.

extern void gaussian_smooth(unsigned char *image, int rows,
 int cols, float sigma,
        short int **smoothedim);
extern void derrivative_x_y(short int *smoothedim, int rows, int cols,
        short int **delta_x, short int **delta_y);
extern void magnitude_x_y(short int *delta_x, short int *delta_y,
int rows, int cols,
        short int **magnitude);
extern void apply_hysteresis(short int *mag, unsigned char *nms,
int rows, int cols,
        float tlow, float thigh, unsigned char *edge, int *hist);
extern void non_max_supp(short *mag, short *gradx, short *grady,
int nrows, int ncols,
    unsigned char *result);

/*****************************************************************
**
**
**
**
*****************************************************************/

// source from: http://en.wikipedia.org/wiki/Eigenvalue_algorithm
/*

% Given symmetric 3x3 matrix M, compute the eigenvalues
m = trace(M)/3;
K = M-m*eye(3); //eye return identify matrix of 3*3
q = det(K)/2;

p = 0
for i=1:3
    for j=1:3
        p = p + K(i,j)^2;
    end
end
p = p/6;

phi = 1/3*atan(sqrt(p^3-q^2)/q);
if(phi<0)
    phi=phi+pi/3;
```

```
end

eig1 = m + 2*sqrt(p)*cos(phi)
eig2 = m - sqrt(p)*(cos(phi) + sqrt(3)*sin(phi))
eig3 = m - sqrt(p)*(cos(phi) - sqrt(3)*sin(phi))

*/
void Eigen3x3symmetric(float fMatrix[3][3], float *fNamda, float **fVector)
{
double fM3[3][3];

double Trace = fMatrix[0][0]+fMatrix[1][1]+fMatrix[2][2];

double m = Trace/3;

int x, y;

double p = 0;

for (y=0; y<3; y++)
for (x=0; x<3; x++)
{
fM3[y][x] = fMatrix[y][x];

if (x==y)
fM3[y][x] -= m;

p += fM3[y][x]*fM3[y][x];
}

p = p/6;

double det = fM3[0][0]*fM3[1][1]*fM3[2][2]-fM3[0][0]*fM3[2][1]*fM3[1][2]-
fM3[1][0]*fM3[0][1]*fM3[2][2]+fM3[1][0]*fM3[0][2]*fM3[2][1]+
fM3[2][0]*fM3[0][1]*fM3[1][2]-fM3[2][0]*fM3[0][2]*fM3[1][1];

double q = det/2;

double phi = atan(sqrt(p*p*p-q*q)/q)/3;

if (phi<0) phi+= 3.1415927/3;

fNamda[0] = m + 2*sqrt(p)*cos(phi);
fNamda[1] = m - sqrt(p)*(cos(phi) + sqrt(3.0)*sin(phi));
fNamda[2] = m - sqrt(p)*(cos(phi) - sqrt(3.0)*sin(phi));

}

void Eigen3x3(float fMatrix[3][3], float *fNamda, float **fVector)
{
double a,b,c,d,x,y,z, i,j,k,m,n,p;

/*for a 3x3 matrix fMatrix, its characteristic polynomical is obtained below

a 3x3 is more complicated and requires several helper equations
to accomplish due to the ęË3 term.
These steps should help you calculate eigen values for a matris
that has 3 REAL eigen values .

  Eqn = -aęË3 + bęË2 + cęË + d = 0

*/
```

```
a = 1;
b = fMatrix[0][0]+fMatrix[1][1]+fMatrix[2][2];
c = fMatrix[0][1]*fMatrix[1][0]+fMatrix[0][2]*fMatrix[2][0]+
fMatrix[1][2]*fMatrix[2][1]-
fMatrix[0][0]*fMatrix[1][1]-fMatrix[0][0]*fMatrix[2][2]-
fMatrix[1][1]*fMatrix[2][2];
d = fMatrix[0][0]*fMatrix[1][1]*fMatrix[2][2]-fMatrix[0][0]*
fMatrix[2][1]*fMatrix[1][2]-
fMatrix[1][0]*fMatrix[0][1]*fMatrix[2][2]+fMatrix[1][0]*
fMatrix[0][2]*fMatrix[2][1]+
fMatrix[2][0]*fMatrix[0][1]*fMatrix[1][2]-fMatrix[2][0]*
fMatrix[0][2]*fMatrix[1][1];

/*
Define x,y,z //Use the equation from above to get your cubic equation
 and combine all constant terms possible to
//give you a reduced equation we will use a, b, c and d to denote
the coefficients of this equation.

x = ((3c/a) ÍC (b2/a2))/3
y = ((2b3/a3) ÍC (9bc/a2) + (27d/a))/27
z = y2/4 + x3/27
*/
x = (3*c/a-b*b/a/a)/3;
y = (2*b*b*b/a/a/a-9*b*c/a/a+27*d/a)/27;
z = y*y/4+x*x*x/27;

//Define I, j, k, m, n, p (so equations are not so cluttered)
/*i = sqrt(y2/4 - z)
j = -i^(1/3)
k = arccos(-(y/2i))
m = cos(k/3)
n = sqrt(3)*sin(k/3)
p = -(b/3a)
*/

i = sqrt(y*y/4 - z);

j = -pow( i,1.0/3);

//k = acos(-(y/2/i)); //acos
k = atan(-(y/2/i)); //acos
if (k<0) k+= 3.1415927/3;

//double phi = atan(sqrt(p*p*p-q*q)/q)/3;
//if (phi<0) phi+= 3.1415927/3;

m = cos(k/3);
n = sqrt(3.0)*sin(k/3);
p = -(b/3/a);

//Define Eig1, Eig2, Eig3

/* Eig1 = -2j*m + p
Eig2 = j *(m + n) + p
Eig3 = j*(m - n) + p
*/
fNamda[0] = -2*j*m + p;
fNamda[1] = j *(m + n) + p;
fNamda[2] = j*(m - n) + p;
```

```
}

void ChrominanceEdgePreserveGreyTransform(IplImage *OriginalImage,
IplImage *DestGreyImage)
{
unsigned char ch4[4];
float fImgMean[3],fMatrix[3][3], fM[3][3], fTemp[3][3];
float fSum1, fSum2, fSum3;
float fV0[3], fV1[3],fV2[3], fV3[3];
float fNamda[3],  fEigenVector[3][3];

float fTestVector[3][3] =
{
0.231, 0.022, -0.392,
        0.408, -0.265, -0.333,
0.361, -0.713, -0.275
};

int x,y,k, m, n;
int height,width, iImageSize;
/*
fMatrix[0][0] = 0;
fMatrix[0][1] = 1;
fMatrix[0][2] = -1;
fMatrix[1][0] = 1;
fMatrix[1][1] = 1;
fMatrix[1][2] = 0;
fMatrix[2][0] = -1;
fMatrix[2][1] = 0;
fMatrix[2][2] = 1;

printf("Matrix\n(%.3f, %.3f, %.3f)\n",fMatrix[0][0],fMatrix[0][1],
fMatrix[0][2]);
printf("(%.3f, %.3f, %.3f)\n",fMatrix[1][0],fMatrix[1][1],
fMatrix[1][2]);
printf("(%.3f, %.3f, %.3f)\n",fMatrix[2][0],fMatrix[2][1],
fMatrix[2][2]);

Eigen3x3(fMatrix,(float *)&fNamda,(float **)&fEigenVector);

printf("Eigen Values: (%.3f,%.3f,%.3f)\n",fNamda[0],fNamda[1],
fNamda[2]);


Eigen3x3symmetric(fMatrix,(float *)&fNamda,(float **)&fEigenVector);

printf("Eigen Values: (%.3f,%.3f,%.3f)\n",fNamda[0],fNamda[1],fNamda[2]);

*/

if (OriginalImage->nChannels==1)
{
iplCopy(OriginalImage,DestGreyImage);
}

fImgMean[0] = fImgMean[1] = fImgMean[2] = 0;

memset(fMatrix,0,9*sizeof(float));

height = OriginalImage->height;
width = OriginalImage->width;
```

```
iImageSize = width*height;

for (y=0; y<height; y++)
for (x=0; x<width; x++)
{
iplGetPixel(OriginalImage,x,y,ch4);

fImgMean[0] += ch4[0];
fImgMean[1] += ch4[1];
fImgMean[2] += ch4[2];
}

for (k=0; k<3; k++)
fImgMean[k] = fImgMean[k]/iImageSize;

for (y=0; y<height; y++)
for (x=0; x<width; x++)
{
iplGetPixel(OriginalImage,x,y,ch4);

for (k=0; k<3; k++)
fMatrix[k][k] += (ch4[k]-fImgMean[k])*(ch4[k]-fImgMean[k]);

fMatrix[0][1] += (ch4[0]-fImgMean[0])*(ch4[1]-fImgMean[1]);

fMatrix[0][2] += (ch4[0]-fImgMean[0])*(ch4[2]-fImgMean[2]);

fMatrix[1][2] += (ch4[1]-fImgMean[1])*(ch4[2]-fImgMean[2]);
}

for (k=0; k<3; k++)
fMatrix[k][k] = fMatrix[k][k]/iImageSize;

fMatrix[0][1] = fMatrix[0][1]/iImageSize;
fMatrix[0][2] = fMatrix[0][2]/iImageSize;
fMatrix[1][2] = fMatrix[1][2]/iImageSize;

fSum1 = fMatrix[0][1]*2+fMatrix[0][2]*2+fMatrix[1][2]*2+fMatrix[0][0]
+fMatrix[1][1]+fMatrix[2][2];

fMatrix[0][0] = fMatrix[0][0]/fSum1;
fMatrix[0][1] = fMatrix[0][1]/fSum1;
fMatrix[0][2] = fMatrix[0][2]/fSum1;
fMatrix[1][1] = fMatrix[1][1]/fSum1;
fMatrix[1][2] = fMatrix[1][2]/fSum1;
fMatrix[2][2] = fMatrix[2][2]/fSum1;

fMatrix[1][0] = fMatrix[0][1];
fMatrix[2][0] = fMatrix[0][2];
fMatrix[2][1] = fMatrix[1][2];

//printf("Mean Vector: (%.3f,%.3f,%.3f)\n",fImgMean[0],fImgMean[1],
fImgMean[2]);

/* printf("Matrix\n(%.3f, %.3f, %.3f)\n",fMatrix[0][0],fMatrix[0][1],
fMatrix[0][2]);
printf("(%.3f, %.3f, %.3f)\n",fMatrix[1][0],fMatrix[1][1],
fMatrix[1][2]);
printf("(%.3f, %.3f, %.3f)\n",fMatrix[2][0],fMatrix[2][1],
fMatrix[2][2]);
```

```
Eigen3x3symmetric(fMatrix,(float *)&fNamda,(float **)&fEigenVector);

printf("Eigen Values: (%.3f,%.3f,%.3f)\n",fNamda[0],fNamda[1],fNamda[2]);
*/
/*
for (m=0; m<3; m++)
for (n=0; n<3; n++)
{
fTemp[m][n] = 0;

for (k=0; k<3; k++)
fTemp[m][n] += fMatrix[m][k]*fTestVector[k][n];
}


printf("Test Matrix\n(%.3f, %.3f, %.3f)\n",fTestVector[0][0],
fTestVector[0][1],fTestVector[0][2]);
printf("(%.3f, %.3f, %.3f)\n",fTestVector[1][0],fTestVector[1][1],
fTestVector[1][2]);
printf("(%.3f, %.3f, %.3f)\n",fTestVector[2][0],fTestVector[2][1],
fTestVector[2][2]);

printf("Multiply Matrix\n(%.3f, %.3f, %.3f)\n",fTemp[0][0],fTemp[0][1],
fTemp[0][2]);
printf("(%.3f, %.3f, %.3f)\n",fTemp[1][0],fTemp[1][1],fTemp[1][2]);
printf("(%.3f, %.3f, %.3f)\n",fTemp[2][0],fTemp[2][1],fTemp[2][2]);

*/
/*
fMatrix[0][0] = 0;
fMatrix[0][1] = 1;
fMatrix[0][2] = -1;
fMatrix[1][0] = 1;
fMatrix[1][1] = 1;
fMatrix[1][2] = 0;
fMatrix[2][0] = -1;
fMatrix[2][1] = 0;
fMatrix[2][2] = 1;

printf("Matrix\n(%.3f, %.3f, %.3f)\n",fMatrix[0][0],fMatrix[0][1],
fMatrix[0][2]);
printf("(%.3f, %.3f, %.3f)\n",fMatrix[1][0],fMatrix[1][1],fMatrix[1][2]);
printf("(%.3f, %.3f, %.3f)\n",fMatrix[2][0],fMatrix[2][1],fMatrix[2][2]);

Eigen3x3symmetric(fMatrix,(float *)&fNamda,(float **)&fEigenVector);

printf("Eigen Values: (%.3f,%.3f,%.3f)\n",fNamda[0],fNamda[1],fNamda[2]);

*/



//get the eigenvector and eigenvalue

fV0[0] = 0.6; fV0[1] = 0.3; fV0[2] = 0.1;

//printf("Initial 1st Vector: (%.3f,%.3f,%.3f)\n",fV0[0],fV0[1],fV0[2]);

for (k=0; k<5; k++) //suggested three times of loop
{
fV1[0] = fV1[1] = fV1[2] = 0;
```

```
fSum1 = 0;


for (m=0; m<3; m++)
{

for (n=0; n<3; n++)
fV1[m] += fMatrix[m][n]*fV0[n];

fSum1 += fabs(fV1[m]);
}


fNamda[0] = (fV1[0]/fV0[0]+fV1[1]/fV0[1]+fV1[2]/fV0[2])/3;

for (x=0; x<3; x++)
fV0[x] = fV1[x]/fSum1;

// printf("New 1st Vector at Step %d: (%.3f,%.3f,%.3f) EigenValue:
%.3f\n",k, fV0[0],fV0[1],fV0[2],fNamda[0]);


}


//now, we get the eigenvalue below in fV1 and output it here:

iplSet(DestGreyImage,0);


for (y=0; y<height; y++)
for (x=0; x<width; x++)
{
iplGetPixel(OriginalImage,x,y,ch4);

fSum1 = 0;

for (k=0; k<3; k++)
fSum1 += ch4[k]*fV0[k];

k = round(fSum1);

DestGreyImage->imageData[y*DestGreyImage->widthStep+x] = (unsigned char) k;
}

return;
/*now try to recover the 2nd eigen vector*/


//get the 2nd eigenvector and eigenvalue
/*
fV0[0] = 0.6; fV0[1] = 0.3; fV0[2] = 0.1;


printf("Initial 2nd Vector: (%.3f,%.3f,%.3f)\n",fV0[0],fV0[1],fV0[2]);

for (k=0; k<10; k++) //suggested three times of loop
{
fV1[0] = fV1[1] = fV1[2] = 0;

fSum = 0;
```

```
for (m=0; m<3; m++)
{
for (n=0; n<3; n++)
{
fV1[m] += fMatrix[m][n]*(fV0[n]-fEigenVector[n][0]);
}


fSum += fabs(fV1[m]);
}

fNamda[1] = (fV1[0]/fV0[0]+fV1[1]/fV0[1]+fV1[2]/fV0[2])/3;

for (x=0; x<3; x++)
fV0[x] = fV1[x]/fSum;

printf("New 2nd Vector at Step %d: (%.3f,%.3f,%.3f) EigenValue:
%.3f\n",k, fV0[0],fV0[1],fV0[2],fNamda[1]);

}

*/

/**/

for (m=0; m<3; m++)
for (n=0; n<3; n++)
            fTemp[m][n] = fMatrix[m][n];

int iLevelK = 1;


//estimate a1 etc

for (k=0; k<10; k++)
{
iLevelK++;

for (m=0; m<3; m++)
for (n=0; n<3; n++)
{

fM[m][n] = 0;

for (int mn=0; mn<3; mn++)
fM[m][n] += fTemp[m][mn]*fMatrix[n][mn];

}

for (m=0; m<3; m++)
for (n=0; n<3; n++)
fTemp[m][n] = fM[m][n];



if (iLevelK <4)
{
// printf("Matrix level %d\n(%.3f, %.3f, %.3f)\n",iLevelK,
fM[0][0],fM[0][1],fM[0][2]);
// printf("(%.3f, %.3f, %.3f)\n",fM[1][0],fM[1][1],fM[1][2]);
// printf("(%.3f, %.3f, %.3f)\n",fM[2][0],fM[2][1],fM[2][2]);
}
```

```
//estimate the first
/*
fV0[0] = 0.6; fV0[1] = 0.3; fV0[2] = 0.1;

fV1[0] = fV1[1] = fV1[2] = 0;

fSum1 = fSum2 = fSum3 = 0;

for (m=0; m<3; m++)
{
for (n=0; n<3; n++)
fV1[m] += fM[m][n]*fV0[n];

fV2[m] = fV1[m] - fNamda[0]*fEigenVector[m][0];

fV3[m] = fV2[m] - fNamda[1]*fEigenVector[m][1];

fSum1 += fabs(fV1[m]);
fSum2 += fabs(fV2[m]);
fSum3 += fabs(fV3[m]);
}

fNamda[0] = fSum1;
fNamda[1] = fSum2;
fNamda[2] = fSum3;

printf("New 1st Vector at level %d: (%.3f,%.3f,%.3f) EigenValue: %.3f\n",iLevelK,fV1[0]/fSum1,fV1[1]/fSum1,fV1[2]/fSu
printf("New 2nd Vector at level %d: (%.3f,%.3f,%.3f) EigenValue: %.3f\n",iLevelK,fV2[0]/fSum2,fV2[1]/fSum2,fV2[2]/fSu
printf("New 3rd Vector at level %d: (%.3f,%.3f,%.3f) EigenValue: %.3f\n",iLevelK,fV3[0]/fSum3,fV3[1]/fSum3,fV3[2]/fSu

fEigenVector[0][0] = fV1[0]/fSum1;
fEigenVector[1][0] = fV1[1]/fSum1;
fEigenVector[2][0] = fV1[2]/fSum1;

fEigenVector[0][1] = fV2[0]/fSum2;
fEigenVector[1][1] = fV2[1]/fSum2;
fEigenVector[2][1] = fV2[2]/fSum2;

fEigenVector[0][2] = fV3[0]/fSum3;
fEigenVector[1][2] = fV3[1]/fSum3;
fEigenVector[2][2] = fV3[2]/fSum3;
*/


}

fV0[0] = 0.6; fV0[1] = 0.3; fV0[2] = 0.1;

fV1[0] = fV1[1] = fV1[2] = 0;

fSum1 = fSum2 = fSum3 = 0;

for (m=0; m<3; m++)
{
for (n=0; n<3; n++)
fV1[m] += fM[m][n]*fV0[n];


fSum1 += fabs(fV1[m]);
}
```

```
fNamda[0] = fSum1;

printf("New 1st Vector at level %d: (%.3f,%.3f,%.3f) EigenValue: %.3f\n",iLevelK,fV1[0]/fSum1,fV1[1]/fSum1,fV1[2]/fSu

fEigenVector[0][0] = fV1[0]/fSum1;
fEigenVector[1][0] = fV1[1]/fSum1;
fEigenVector[2][0] = fV1[2]/fSum1;


iLevelK *=2;

for (m=0; m<3; m++)
for (n=0; n<3; n++)
fTemp[m][n] = fM[m][n];

for (m=0; m<3; m++)
for (n=0; n<3; n++)
{

fM[m][n] = 0;

for (int mn=0; mn<3; mn++)
fM[m][n] += fTemp[m][mn]*fTemp[n][mn];

}


fV0[0] = 0.6; fV0[1] = 0.3; fV0[2] = 0.1;

fV1[0] = fV1[1] = fV1[2] = 0;

fSum1 = fSum2 = fSum3 = 0;

for (m=0; m<3; m++)
{
for (n=0; n<3; n++)
fV1[m] += fM[m][n]*fV0[n];

fV2[m] = fV1[m] - fNamda[0]*fEigenVector[m][0];


fSum1 += fabs(fV1[m]);
fSum2 += fabs(fV2[m]);
}

fNamda[1] = fSum2;

printf("New 2nd Vector at level %d: (%.3f,%.3f,%.3f) EigenValue: %.3f\n",iLevelK,fV2[0]/fSum2,fV2[1]/fSum2,fV2[2]/fSu

fEigenVector[0][1] = fV2[0]/fSum2;
fEigenVector[1][1] = fV2[1]/fSum2;
fEigenVector[2][1] = fV2[2]/fSum2;

//
fV0[0] = 0.6; fV0[1] = 0.3; fV0[2] = 0.1;

fV1[0] = fV1[1] = fV1[2] = 0;

fSum1 = fSum2 = fSum3 = 0;

for (m=0; m<3; m++)
{
```

```
for (n=0; n<3; n++)
fV1[m] += fM[m][n]*fV0[n];

fV2[m] = fV1[m] - fNamda[0]*fEigenVector[m][0];

fV3[m] = fV2[m] - fNamda[1]*fEigenVector[m][1];


fSum3 += fabs(fV3[m]);
}

fNamda[2] = fSum3;

printf("New 3rd Vector at level %d: (%.3f,%.3f,%.3f) EigenValue: %.3f\n",iLevelK,fV3[0]/fSum3,fV3[1]/fSum3,fV3[2]/fSu

fEigenVector[0][2] = fV3[0]/fSum3;
fEigenVector[1][2] = fV3[1]/fSum3;
fEigenVector[2][2] = fV3[2]/fSum3;



iLevelK *=2;

for (m=0; m<3; m++)
for (n=0; n<3; n++)
fTemp[m][n] = fM[m][n];

for (m=0; m<3; m++)
for (n=0; n<3; n++)
{

fM[m][n] = 0;

for (int mn=0; mn<3; mn++)
fM[m][n] += fTemp[m][mn]*fTemp[n][mn];

}

fV0[0] = 0.6; fV0[1] = 0.3; fV0[2] = 0.1;

fV1[0] = fV1[1] = fV1[2] = 0;

fSum1 = fSum2 = fSum3 = 0;

for (m=0; m<3; m++)
{
for (n=0; n<3; n++)
fV1[m] += fM[m][n]*fV0[n];

fV2[m] = fV1[m] - fNamda[0]*fEigenVector[m][0];


fSum1 += fabs(fV1[m]);
fSum2 += fabs(fV2[m]);
}

fNamda[1] = fSum2;

printf("New 2nd Vector at level %d: (%.3f,%.3f,%.3f) EigenValue: %.3f\n",iLevelK,fV2[0]/fSum2,fV2[1]/fSum2,fV2[2]/fSu

fEigenVector[0][1] = fV2[0]/fSum2;
fEigenVector[1][1] = fV2[1]/fSum2;
```

```
fEigenVector[2][1] = fV2[2]/fSum2;

//
fV0[0] = 0.6; fV0[1] = 0.3; fV0[2] = 0.1;

fV1[0] = fV1[1] = fV1[2] = 0;

fSum1 = fSum2 = fSum3 = 0;

for (m=0; m<3; m++)
{
for (n=0; n<3; n++)
fV1[m] += fM[m][n]*fV0[n];

fV2[m] = fV1[m] - fNamda[0]*fEigenVector[m][0];

fV3[m] = fV2[m] - fNamda[1]*fEigenVector[m][1];


fSum3 += fabs(fV3[m]);
}

fNamda[2] = fSum3;

printf("New 3rd Vector at level %d: (%.3f,%.3f,%.3f) EigenValue: %.3f\n",iLevelK,fV3[0]/fSum3,fV3[1]/fSum3,fV3[2]/fSu

fEigenVector[0][2] = fV3[0]/fSum3;
fEigenVector[1][2] = fV3[1]/fSum3;
fEigenVector[2][2] = fV3[2]/fSum3;

}




void BlackBoundaryImage(IplImage *img, int iThick)
{
if (img == NULL || iThick<1)
return;

int x,y;
unsigned char ch4[4];

ch4[0] = ch4[1] = ch4[2] = ch4[3] = 0;

for (y=0; y<iThick; y++)
for (x=0; x<img->width; x++)
{
iplPutPixel(img,x,y,ch4);
iplPutPixel(img,x,img->height-1-y,ch4);
}

for (x=0; x<iThick; x++)
for (y=0; y<img->height; y++)
{
iplPutPixel(img,x,y,ch4);
iplPutPixel(img,img->width-1-x,y,ch4);
}
}
void LargeImage(IplImage *imgSrc, char *sFileName,
int *width2 = NULL, int *height2 = NULL)
{
```

```
if (imgSrc == NULL )
return;


int width1 = imgSrc->width;
int height1 = imgSrc->height;

int width=1,height=1;

for (;;)
{
if (width>=width1)
break;

width *= 2;
}

for (;;)
{
if (height>=height1)
break;

height *= 2;
}

IplImage *imgDest = cvCreateImage(cvSize(width,height),
IPL_DEPTH_8U,imgSrc->nChannels);

iplSet(imgDest,0);

int xOffset = (width-width1)/2;
int yOffset = (height-height1)/2;

unsigned char ch4[4];
for (int y=0; y<height1; y++)
for (int x=0; x<width1; x++)
{
iplGetPixel(imgSrc,x,y,ch4);
iplPutPixel(imgDest,x+xOffset,y+yOffset,ch4);
}

if (sFileName!=NULL)
cvSaveImage(sFileName,imgDest);
else
cvSaveImage("large.png",imgDest);

cvReleaseImage(&imgDest);

if (width2 != NULL)
*width2 = width;
if (height2 != NULL)
*height2 = height;
}


void CropImage(IplImage *imgSrc, char *sFileName,
 int *w2 = NULL, int *h2 = NULL)
{

if (imgSrc == NULL )
return;
```

```
int width1 = imgSrc->width;
int height1 = imgSrc->height;

int width=1,height=1;

for (;;)
{
if (width>width1)
{
width = width/2;
break;
}
width *= 2;
}

for (;;)
{
if (height>height1)
{
height = height/2;
break;
}
height *= 2;
}

IplImage *imgDest1 = cvCreateImage(cvSize(width,height),
IPL_DEPTH_8U,imgSrc->nChannels);

int xOffset = (width1-width)/2;
int yOffset = (height1-height)/2;

unsigned char ch4[4];
for (int y=0; y<height; y++)
for (int x=0; x<width; x++)
{
iplGetPixel(imgSrc,x+xOffset,y+yOffset,ch4);
iplPutPixel(imgDest1,x,y,ch4);
}
if (sFileName!=NULL)
cvSaveImage(sFileName,imgDest1);
else
cvSaveImage("crop.png",imgDest1);

cvReleaseImage(&imgDest1);

if (w2 != NULL)
*w2 = width;
if (h2 != NULL)
*h2 = height;
}

void ZoomImage(IplImage *imgSrc, char *sFileName, float fScale,
int interpolate, IplImage **imgDest = NULL)
{

if (imgSrc == NULL )
return;


int width1 = imgSrc->width;
int height1 = imgSrc->height;

int width = (int)(width1*fScale+0.5);
int height = (int) (height1*fScale+0.5);

if (*imgDest == NULL)
{
*imgDest = cvCreateImage(cvSize(width,height),IPL_DEPTH_8U,
imgSrc->nChannels);

iplResize(imgSrc,*imgDest,width,width1,height,height1,interpolate);


if (sFileName!=NULL)
cvSaveImage(sFileName,*imgDest);
else
cvSaveImage("zoom.png",*imgDest);

//cvReleaseImage(&imgDest);
}
else
{
iplResize(imgSrc,*imgDest,width,width1,height,height1,interpolate);


if (sFileName!=NULL)
cvSaveImage(sFileName,*imgDest);
else
cvSaveImage("zoom.png",*imgDest);

}

}

int ColorMappingOld(IplImage* OriginalImage,IplImage* DestGreyImage,int w[3])
{
unsigned char ch[4];
short *image;
short *image2;
double ww[3];

if (OriginalImage->nChannels==1)
{
iplCopy(OriginalImage,DestGreyImage);
return 0;
}

image=new short[OriginalImage->height*OriginalImage->width];
if (image==NULL)
return -1;

image2=new short[OriginalImage->height*OriginalImage->width];

if (image2==NULL)
{
delete [] image;

return -1;
}

for (int n=0;n<4;n++)
ch[n]=0;
```

```
if (w==NULL)
{
for (n=0;n<3;n++)
ww[n]=1.0/3;
}
else
{
for (n=0;n<3;n++)
ww[n]=1.0*w[n]/(w[0]+w[1]+w[2]);
}

iplSet(DestGreyImage,0);

short maxv1=0;
short minv1=10000;

short maxv2=0;
short minv2=10000;

double sum1=0,sum2=0;

for( int j = 0; j < OriginalImage->height; j++ )
for( int i = 0; i < OriginalImage->width; i++ )
{
double count1,count2;

iplGetPixel(OriginalImage,i,j,ch);

count1=ch[0]*ww[0]+ch[1]*ww[1]+ch[2]*ww[2]+0.5;
maxv1=max((short)count1,maxv1);
minv1=min((short)count1,minv1);

sum1+=count1;

image[j*OriginalImage->width+i]=(short)count1;


count2=(abs(ch[0]-ch[1])/3+abs(ch[0]-ch[2])/3+abs(ch[2]-ch[1])/3);

//if (count2<count1/4) count2=count1;

maxv2=max((short)count2,maxv2);
minv2=min((short)count2,minv2);

sum2+=count2;

image2[j*OriginalImage->width+i]=(short)count2;

}

if (w!=NULL)
{
w[2]=maxv2*1000+minv2;
w[1]=maxv1*1000+minv1;
}

sum1=sum1/(OriginalImage->height*OriginalImage->width);
sum2=sum2/(OriginalImage->height*OriginalImage->width);

double std1=0,std2=0;
```

```
for( j = 0; j < OriginalImage->height; j++ )
for( int i = 0; i < OriginalImage->width; i++ )
{
double count1,count2;

count1=image[j*OriginalImage->width+i];
count2=image2[j*OriginalImage->width+i];

std1+=fabs(count1-sum1);
std2+=fabs(count2-sum2);

}

std1=std1/(OriginalImage->height*OriginalImage->width);
std2=std2/(OriginalImage->height*OriginalImage->width);

//sum1=sum1+std1+maxv1;
//sum2=sum2+std2+maxv2;

sum1=maxv1-minv1;
sum2=maxv2-minv2;

//sum1=1;
//sum2=0;
//sum1=sum1*1.5+std1+maxv1;
//sum2=sum2*1.5+std2+maxv2;

short maxv=0;
short minv=10000;



for( j = 0; j < OriginalImage->height; j++ )
for( int i = 0; i < OriginalImage->width; i++ )
{
double count,count1,count2;

count1=image[j*OriginalImage->width+i];
count2=image2[j*OriginalImage->width+i];

//count=(maxv2*count1+maxv1*count2)/(maxv2+maxv1);
count=(sum2*count1+sum1*count2)/(sum1+sum2);
//count=(sum1*count1+sum2*count2)/(sum1+sum2);


maxv=max((short)count,maxv);
minv=min((short)count,minv);

image[j*OriginalImage->width+i]=(short)count;

}

if (w!=NULL)
w[0]=maxv*1000+minv;

for(  j = 0; j < OriginalImage->height; j++ )
{
int offset=j*DestGreyImage->widthStep;

for( int i = 0; i < OriginalImage->width; i++ )
{
float ff=255.0f*(image[j*OriginalImage->width+i]-minv)/(maxv-minv);
DestGreyImage->imageData[offset+i]=unsigned char(ff);

}
}
delete[] image;
delete[] image2;

return 0;

}


int ColorMapping(IplImage* OriginalImage,IplImage* DestGreyImage,int w[3])
{
unsigned char ch[4];
short *image;
short *image2;
double ww[3];

if (OriginalImage->nChannels==1)
{
iplCopy(OriginalImage,DestGreyImage);
return 0;
}

image=new short[OriginalImage->height*OriginalImage->width];
if (image==NULL)
return -1;

image2=new short[OriginalImage->height*OriginalImage->width];

if (image2==NULL)
{
delete [] image;

return -1;
}

for (int n=0;n<4;n++)
ch[n]=0;

if (w==NULL)
{
for (n=0;n<3;n++)
ww[n]=1.0/3;
}
else
{
for (n=0;n<3;n++)
ww[n]=1.0*w[n]/(w[0]+w[1]+w[2]);
}

iplSet(DestGreyImage,0);

short maxv1=0;
short minv1=10000;

short maxv2=0;
short minv2=10000;

double sum1=0,sum2=0;
```

```
for( int j = 0; j < OriginalImage->height; j++ )
for( int i = 0; i < OriginalImage->width; i++ )
{
double count1,count2;

iplGetPixel(OriginalImage,i,j,ch);

count1=ch[0]*ww[0]+ch[1]*ww[1]+ch[2]*ww[2]+0.5;
maxv1=max((short)count1,maxv1);
minv1=min((short)count1,minv1);

sum1+=count1;

image[j*OriginalImage->width+i]=(short)count1;


count2=(abs(ch[0]-ch[1])/3+abs(ch[0]-ch[2])/3+abs(ch[2]-ch[1])/3);
//count2=(abs(ch[2]-ch[1])/2+abs(ch[0]-count1)/2);

//if (count2<count1/4) count2=count1;

maxv2=max((short)count2,maxv2);
minv2=min((short)count2,minv2);

sum2+=count2;

image2[j*OriginalImage->width+i]=(short)count2;

}

if (w!=NULL)
{
w[2]=maxv2*1000+minv2;
w[1]=maxv1*1000+minv1;
}

sum1=sum1/(OriginalImage->height*OriginalImage->width);
sum2=sum2/(OriginalImage->height*OriginalImage->width);

double std1=0,std2=0;

int index = 0;

for( j = 0; j < OriginalImage->height; j++ )
for( int i = 0; i < OriginalImage->width; i++ )
{
std1+=fabs(image[index]-sum1);
std2+=fabs(image2[index++]-sum2);

}

std1=std1/(OriginalImage->height*OriginalImage->width);
std2=std2/(OriginalImage->height*OriginalImage->width);

//sum1=sum1+std1+maxv1;
//sum2=sum2+std2+maxv2;

sum1=maxv1-minv1;
sum2=maxv2-minv2;

//sum1=1;
```

```
//sum2=0;
//sum1=sum1*1.5+std1+maxv1;
//sum2=sum2*1.5+std2+maxv2;
sum1=sum1*1.5+std1;
sum2=sum2*1.5+std2;

short maxv=0;
short minv=10000;

index = 0;

for( j = 0; j < OriginalImage->height; j++ )
for( int i = 0; i < OriginalImage->width; i++ )
{
double count,count1,count2;

count1=image[index];
count2=image2[index++];

//count=(maxv2*count1+maxv1*count2)/(maxv2+maxv1);
count=(sum2*count1+sum1*count2)/(sum1+sum2);
//count=(sum1*count1+sum2*count2)/(sum1+sum2);

//count=(sum2*count1/2+sum1*(count2/2+127))/(sum1+sum2);
//count=count2*255/sum2; //+count1*255/sum1/2;

maxv=max((short)count,maxv);
minv=min((short)count,minv);

image[j*OriginalImage->width+i]=(short)count;

}

/*
for( j = 1; j < OriginalImage->height-1; j++ )
for( int i = 1; i < OriginalImage->width-1; i++ )
{
double count,count1,count2;

count1=image[j*OriginalImage->width+i];
count2=image2[j*OriginalImage->width+i];

//count=(maxv2*count1+maxv1*count2)/(maxv2+maxv1);
count=(sum2*count1+sum1*count2)/(sum1+sum2);
//count=(sum1*count1+sum2*count2)/(sum1+sum2);

//count=(sum2*count1/2+sum1*(count2/2+127))/(sum1+sum2);
count=count2*255/sum2; //+count1*255/sum1/2;

maxv=max((short)count,maxv);
minv=min((short)count,minv);

image[j*OriginalImage->width+i]=(short)count;

}*/

if (w!=NULL)
w[0]=maxv*1000+minv;

for(  j = 0; j < OriginalImage->height; j++ )
{
int offset=j*DestGreyImage->widthStep;
for( int i = 0; i < OriginalImage->width; i++ )
{
float ff=255.0f*(image[j*OriginalImage->width+i]-minv)/(maxv-minv);

DestGreyImage->imageData[offset+i]=unsigned char(ff);

}
}
delete[] image;
delete[] image2;

return 0;

}


int ColorMappingFast(IplImage* OriginalImage,IplImage* DestGreyImage,
IplImage* GreyImage,int w[3], double stdGrey = -1)
{
unsigned char ch[4];
short *image;
short *image2;
double ww[3];

if (OriginalImage->nChannels==1)
{
iplCopy(OriginalImage,DestGreyImage);
return 0;
}

image=new short[OriginalImage->height*OriginalImage->width];
if (image==NULL)
return -1;

image2=new short[OriginalImage->height*OriginalImage->width];

if (image2==NULL)
{
delete [] image;

return -1;
}

for (int n=0;n<4;n++)
ch[n]=0;

if (w==NULL)
{
for (n=0;n<3;n++)
ww[n]=1.0/3;
}
else
{
for (n=0;n<3;n++)
ww[n]=1.0*w[n]/(w[0]+w[1]+w[2]);
}

iplSet(DestGreyImage,0);

short maxv1=0;
short minv1=10000;
short maxv2=0;
short minv2=10000;

double sum1=0,sum2=0;
int iOffset = 0;
int index = 0;

for( int j = 0; j < OriginalImage->height; j++ )
{
for( int i = 0; i < OriginalImage->width; i++ )
{
double count1,count2;

iplGetPixel(OriginalImage,i,j,ch);

count1=(unsigned char)GreyImage->imageData[iOffset+i];

if (count1>maxv1)
                maxv1 = count1;
else if (count1<minv1)
minv1 = count1;

//maxv1 = max((short)count1,maxv1);
//minv1=min((short)count1,minv1);

//sum1+=count1;

image[index]=(short)count1;


count2=(abs(ch[0]-ch[1])+abs(ch[0]-ch[2])+abs(ch[2]-ch[1]))*0.333333333333333;
//count2=(abs(ch[2]-ch[1])/2+abs(ch[0]-count1)/2);

//if (count2<count1/4) count2=count1;

if (count2>maxv2)
                maxv2 = count2;
else if (count2<minv2)
minv2 = count2;

//maxv2=max((short)count2,maxv2);
//minv2=min((short)count2,minv2);

sum2+=count2;

image2[index++]=(short)count2;

}
iOffset += DestGreyImage->widthStep;

}

if (w!=NULL)
{
w[2]=maxv2*1000+minv2;
w[1]=maxv1*1000+minv1;
}

// sum1=sum1/(OriginalImage->height*OriginalImage->width);
sum2=sum2/(OriginalImage->height*OriginalImage->width);
```

```
double std1=0,std2=0;


if (stdGrey>=0) std1 = stdGrey;

index = 0;

for( j = 0; j < OriginalImage->height; j++ )
for( int i = 0; i < OriginalImage->width; i++ )
{
double temp = image2[index++]-sum2;

if (temp>=0) std2 += temp;
else std2 -= temp;

//std1+=fabs(image[index]-sum1);
//std2+=fabs(-sum2);

}

//std1=std1/(OriginalImage->height*OriginalImage->width);
std2=std2/(OriginalImage->height*OriginalImage->width);

//sum1=sum1+std1+maxv1;
//sum2=sum2+std2+maxv2;

sum1=maxv1-minv1;
sum2=maxv2-minv2;

//sum1=1;
//sum2=0;
//sum1=sum1*1.5+std1+maxv1;
//sum2=sum2*1.5+std2+maxv2;

short maxv=0;
short minv=10000;
double sum = 1.0/(sum1+sum2);

index = 0;

for( j = 0; j < OriginalImage->height; j++ )
for( int i = 0; i < OriginalImage->width; i++ )
{
double count;

//count=(maxv2*count1+maxv1*count2)/(maxv2+maxv1);
count=(sum2*image[index]+sum1*image2[index])*sum;
//count=(sum1*count1+sum2*count2)/(sum1+sum2);

//count=(sum2*count1/2+sum1*(count2/2+127))/(sum1+sum2);
//count=count2*255/sum2; //+count1*255/sum1/2;


if (count>maxv)
                maxv = count;
else if (count<minv)
minv = count;


//maxv=max((short)count,maxv);
//minv=min((short)count,minv);
image[index++]=(short)count;

}

/*
for( j = 1; j < OriginalImage->height-1; j++ )
for( int i = 1; i < OriginalImage->width-1; i++ )
{
double count,count1,count2;

count1=image[j*OriginalImage->width+i];
count2=image2[j*OriginalImage->width+i];

//count=(maxv2*count1+maxv1*count2)/(maxv2+maxv1);
count=(sum2*count1+sum1*count2)/(sum1+sum2);
//count=(sum1*count1+sum2*count2)/(sum1+sum2);

//count=(sum2*count1/2+sum1*(count2/2+127))/(sum1+sum2);
count=count2*255/sum2; //+count1*255/sum1/2;

maxv=max((short)count,maxv);
minv=min((short)count,minv);

image[j*OriginalImage->width+i]=(short)count;

}*/

if (w!=NULL)
w[0]=maxv*1000+minv;

iOffset = 0;
float fScale = 255.0f/(maxv-minv);

for(  j = 0; j < OriginalImage->height; j++ )
{
for( int i = 0; i < OriginalImage->width; i++ )
{
//float ff=255.0f*(image[iOffset+i]-minv)*fScale;

DestGreyImage->imageData[iOffset+i]=unsigned char((image[iOffset+i]-minv)*fScale);

}
iOffset += DestGreyImage->widthStep;
}
delete[] image;
delete[] image2;

return 0;

}



int ExtractEntropy(unsigned char *cBuffer,IplImage* DestGreyImage)
{
double ww[3];
float f_e = log(2.718281828459);
float f_ni = log(9.0);
double sumEntropy;
int height = DestGreyImage->height,width=DestGreyImage->width;
float *fEntropyBuffer = new float[width*height];
unsigned char *cEntropyBuffer = new unsigned char[width*height];
```

```
int i,j,i1,j1, hist[32768];



if (cBuffer == NULL )
{
return -1;
}


float maxv1=0;
float minv1=10000;

double sum1=0,sum2=0;

sumEntropy = 0;

for(  j = 1; j < height-1; j++ )
for(  i = 1; i < width-1; i++ )
{
double count1,count2;
float fSum = 0;
int index = j*width+i;
float fEntropy;

fSum = cBuffer[index]+cBuffer[index-1]+cBuffer[index+1]+
cBuffer[index-width]+cBuffer[index+width]+
cBuffer[index-width+1]+cBuffer[index-width-1]+cBuffer[index+width-1]+
cBuffer[index+width+1];

if (fSum !=0)
{
fEntropy = 0;

for ( j1 = j-1; j1<=j+1; j1++)
for ( i1 = i-1; i1<=i+1; i1++)
{
int index2 = index+(j1-j)*width+i1-i;
float f1 = cBuffer[index2]/fSum;

if (cBuffer[index2]!=0)
fEntropy += (-f1*log(f1));
//check entropy in each channel

}

fEntropy = 100*(f_ni - fEntropy);
}
else
{
fEntropy = 0;
}

 minv1 = min(fEntropy,minv1);
 maxv1 = max(fEntropy,maxv1);

 fEntropyBuffer[index] = fEntropy;
 sumEntropy += fEntropy;
}

sumEntropy = sumEntropy/(height*width-height*2-width*2+4);
double sigma = 0;
```

```
for(  j = 1; j < height-1; j++ )
for(  i = 1; i < width-1; i++ )
{
float fEntropy =  fEntropyBuffer[j*width+i]-sumEntropy;

sigma += (fEntropy*fEntropy);
}

sigma = sqrt(sigma/(height*width-height*2-width*2+4));

printf("max  %.5f min %.5f mean %.5f sigma %.5f\n",maxv1,minv1, sumEntropy, sigma);

/*
for(  j = 1; j < height-1; j++ )
for(  i = 1; i < width-1; i++ )
{
float fEntropy =  fEntropyBuffer[j*width+i];

if (fEntropy<sumEntropy+sigma/200)
fEntropyBuffer[j*width+i] = 0;
}

*/
minv1 = log(1+minv1);
maxv1 = log(1+maxv1);
printf("log max  %.5f min %.5f mean %.5f\n",maxv1,minv1,log(1+sumEntropy));

for(  j = 1; j < height-1; j++ )
for(  i = 1; i < width-1; i++ )
{
float fEntropy =  fEntropyBuffer[j*width+i];

fEntropy = (log(1+fEntropy)-minv1)*255/(maxv1-minv1)+0.5;

//cEntropyBuffer[j*width+i] = (unsigned char)(fEntropy);
//magnitude[j*width+i] = (short) (fEntropy);

DestGreyImage->imageData[j*DestGreyImage->widthStep+i] = (unsigned char)(fEntropy);

}

/*gaussian_smooth(cEntropyBuffer, height, width, 0.7, &smoothedim);
derrivative_x_y(smoothedim, height, width, &delta_x, &delta_y);
magnitude_x_y(delta_x, delta_y, height, width, &magnitude);
non_max_supp(magnitude, delta_x, delta_y,height, width, nms);
apply_hysteresis(magnitude, nms, height, width, 0.2, 0.8, pFeature, hist);


   for(  j = 1; j < height-1; j++ )
for(  i = 1; i < width-1; i++ )
{
DestGreyImage->imageData[j*DestGreyImage->widthStep+i] =
(unsigned char)(pFeature[j*width+i]);

}
*/

delete[] fEntropyBuffer;
delete [] cEntropyBuffer;

return 0;
```

```
}



int ExtractCrossEntropy(unsigned char *cBuffer,IplImage* DestGreyImage)
{
double ww[3];
float f_e = log(2.718281828459);
float f_ni = log(9.0);
double sumEntropy;
int height = DestGreyImage->height,width=DestGreyImage->width;
float *fEntropyBuffer = new float[width*height];
int i,j,i1,j1;



if (cBuffer == NULL )
{
return -1;
}


float maxv1=0;
float minv1=10000;

double sum1=0,sum2=0;

sumEntropy = 0;

for(  j = 1; j < height-1; j++ )
for(  i = 1; i < width-1; i++ )
{
double count1,count2;
float fSum = 0;
int index = j*width+i;
float fEntropy, f0;

fSum = cBuffer[index]+cBuffer[index-1]+cBuffer[index+1]+cBuffer[index-width]+
cBuffer[index+width]+
cBuffer[index-width+1]+cBuffer[index-width-1]+cBuffer[index+width-1]+
cBuffer[index+width+1];

if (fSum!=0)
f0 = cBuffer[index]/fSum;
else f0 = 0;

if (fSum !=0 && f0 !=0)
{


fEntropy = 0;

for ( j1 = j-1; j1<=j+1; j1++)
for ( i1 = i-1; i1<=i+1; i1++)
{
int index2 = index+(j1-j)*width+i1-i;
float f1 = cBuffer[index2]/fSum;

if (cBuffer[index2]!=0)
{
if (f1>f0)
fEntropy += (f1*log(f1/f0));
else
fEntropy += (-f1*log(f1/f0));
}
//check entropy in each channel

}

fEntropy = 80* fEntropy;
}
else
{
fEntropy = 0;
}

 minv1 = min(fEntropy,minv1);
 maxv1 = max(fEntropy,maxv1);

 fEntropyBuffer[index] = fEntropy;
 sumEntropy += fEntropy;
}

sumEntropy = sumEntropy/(height*width-height*2-width*2+4);
double sigma = 0;

for(  j = 1; j < height-1; j++ )
for(  i = 1; i < width-1; i++ )
{
float fEntropy =  fEntropyBuffer[j*width+i]-sumEntropy;

sigma += (fEntropy*fEntropy);
}

sigma = sqrt(sigma/(height*width-height*2-width*2+4));

printf("max  %.5f min %.5f mean %.5f sigma %.5f\n",maxv1,minv1,
sumEntropy, sigma);

/*
for(  j = 1; j < height-1; j++ )
for(  i = 1; i < width-1; i++ )
{
float fEntropy =  fEntropyBuffer[j*width+i];

if (fEntropy<sumEntropy+sigma/200)
fEntropyBuffer[j*width+i] = 0;
}

*/
minv1 = log(1+minv1);
maxv1 = log(1+maxv1);
printf("log max  %.5f min %.5f mean %.5f\n",maxv1,minv1,log(1+sumEntropy));

for(  j = 1; j < height-1; j++ )
for(  i = 1; i < width-1; i++ )
{
float fEntropy =  fEntropyBuffer[j*width+i];


fEntropy = (log(1+fEntropy)-minv1)*255/(maxv1-minv1)+0.5;
// fEntropy = (fEntropy-minv1)*255/(maxv1-minv1)+0.5;
```

```
DestGreyImage->imageData[j*DestGreyImage->widthStep+i] = (unsigned char)(fEntropy);

}


delete[] fEntropyBuffer;

return 0;

}

int ColorMappingW3(unsigned char *cBuffer[3],IplImage* DestGreyImage)
{
float f_e = log(2.718281828459);
float f_ni = log(9.0);
int height = DestGreyImage->height,width=DestGreyImage->width;
float *entropyBuffer = new float[width*height];
int i,j, i1, j1;


if (cBuffer == NULL || cBuffer[0] == NULL || cBuffer[1] == NULL ||
cBuffer[2] == NULL)
{
return -1;
}


//iplSet(DestGreyImage,0);

float maxv1=0;
float minv1=10000;
double sum1 = 0;

for(  j = 1; j < height-1; j++ )
for(  i = 1; i < width-1; i++ )
{
double count1,count2;
double fEntropy[3];
float fSum = 0;
int index = j*width+i;

fSum = cBuffer[0][index]+cBuffer[0][index-1]+cBuffer[0][index+1]+
cBuffer[0][index-width]+cBuffer[0][index+width]+
cBuffer[0][index-width+1]+cBuffer[0][index-width-1]+cBuffer[0][index+width-1]+
cBuffer[0][index+width+1];

if (fSum !=0)
{
fEntropy[0] = 0;

for ( j1 = j-1; j1<=j+1; j1++)
for ( i1 = i-1; i1<=i+1; i1++)
{
int index2 = index+(j1-j)*width+i1-i;
float f1 = cBuffer[0][index2]/fSum;

if (cBuffer[0][index2]!=0)
fEntropy[0] += (-f1*log(f1));
//check entropy in each channel

}
fEntropy[0] = (f_ni - fEntropy[0])/f_e;
}
else
{
fEntropy[0] = 0;
}

fSum = cBuffer[1][index]+cBuffer[1][index-1]+cBuffer[1][index+1]+
cBuffer[1][index-width]+cBuffer[1][index+width]+
cBuffer[1][index-width+1]+cBuffer[1][index-width-1]+cBuffer[1][index+width-1]+
cBuffer[1][index+width+1];

if (fSum !=0)
{
fEntropy[1] = 0;

for ( j1 = j-1; j1<=j+1; j1++)
for ( i1 = i-1; i1<=i+1; i1++)
{
int index2 = index+(j1-j)*width+i1-i;
float f1 = cBuffer[1][index2]/fSum;

if (cBuffer[1][index2]!=0)
fEntropy[1] += (-f1*log(f1));
//check entropy in each channel

}

fEntropy[1] = (f_ni - fEntropy[1])/f_e;
}
else
{
fEntropy[1] = 0;
}

fSum = cBuffer[2][index]+cBuffer[2][index-1]+cBuffer[2][index+1]+
cBuffer[2][index-width]+cBuffer[2][index+width]+
cBuffer[2][index-width+1]+cBuffer[2][index-width-1]+cBuffer[2][index+width-1]+
cBuffer[2][index+width+1];

if (fSum !=0)
{
fEntropy[2] = 0;

for ( j1 = j-1; j1<=j+1; j1++)
for ( i1 = i-1; i1<=i+1; i1++)
{
int index2 = index+(j1-j)*width+i1-i;
float f1 = cBuffer[2][index2]/fSum;

if (cBuffer[2][index2]!=0)
fEntropy[2] += (-f1*log(f1));
//check entropy in each channel

}

fEntropy[2] = (f_ni - fEntropy[2])/f_e;
}
else
{
fEntropy[2] = 0;
}
```

```
//iplGetPixel(OriginalImage,i,j,ch);
entropyBuffer[index] = 100*max(max(fEntropy[0],fEntropy[1]),fEntropy[2]);

maxv1=max(entropyBuffer[index],maxv1);
minv1=min(entropyBuffer[index],minv1);

sum1+=entropyBuffer[index];
}


minv1 = log(1+minv1);
maxv1 = log(1+maxv1);

sum1 = sum1/(width*height-2*height-2*width+4);


for(  j = 1; j < height-1; j++ )
for(  i = 1; i < width-1; i++ )
{
float fEntropy = entropyBuffer[j*width+i];

DestGreyImage->imageData[j*DestGreyImage->widthStep+i]  =
(unsigned char)((log(1+fEntropy)-minv1)*255/(maxv1-minv1)+0.5);

}

delete [] entropyBuffer;

return 0;

}


int ColorMappingW3Cross(unsigned char *cBuffer[3],IplImage* DestGreyImage)
{
float f_e = log(2.718281828459);
float f_ni = log(9.0);
int height = DestGreyImage->height,width=DestGreyImage->width;
float *entropyBuffer = new float[width*height];
int i,j, i1, j1;


if (cBuffer == NULL || cBuffer[0] == NULL || cBuffer[1] == NULL || cBuffer[2] == NULL)
{
return -1;
}


//iplSet(DestGreyImage,0);

float maxv1=0;
float minv1=10000;
double sum1 = 0;

for(  j = 1; j < height-1; j++ )
for(  i = 1; i < width-1; i++ )
{
double count1,count2;
double fEntropy[3];
float fSum = 0;
int index = j*width+i;
```

```
float f0;

fSum = cBuffer[0][index]+cBuffer[0][index-1]+cBuffer[0][index+1]+
cBuffer[0][index-width]+cBuffer[0][index+width]+
cBuffer[0][index-width+1]+cBuffer[0][index-width-1]+
cBuffer[0][index+width-1]+cBuffer[0][index+width+1];

if (fSum !=0)
{
fEntropy[0] = 0;

f0 = cBuffer[0][index]/fSum;

if (f0!=0)
{

for ( j1 = j-1; j1<=j+1; j1++)
for ( i1 = i-1; i1<=i+1; i1++)
{
int index2 = index+(j1-j)*width+i1-i;
float f1 = cBuffer[0][index2]/fSum;

if (cBuffer[0][index2]!=0)
{
if (f1>f0) fEntropy[0] += (f1*log(f1/f0));
//else fEntropy[0] += (-f1*log(f1/f0));
}
//check entropy in each channel

}

fEntropy[0] = 80*fEntropy[0];
}
else fEntropy[0] = 0;

}
else
{
fEntropy[0] = 0;
}

fSum = cBuffer[1][index]+cBuffer[1][index-1]+cBuffer[1][index+1]+
cBuffer[1][index-width]+cBuffer[1][index+width]+
cBuffer[1][index-width+1]+cBuffer[1][index-width-1]+cBuffer[1][index+width-1]+
cBuffer[1][index+width+1];

if (fSum !=0)
{
fEntropy[1] = 0;

f0 = cBuffer[1][index]/fSum;

if (f0!=0)
{

for ( j1 = j-1; j1<=j+1; j1++)
for ( i1 = i-1; i1<=i+1; i1++)
{
int index2 = index+(j1-j)*width+i1-i;
float f1 = cBuffer[1][index2]/fSum;

if (cBuffer[1][index2]!=0)
{
if (f1>f0) fEntropy[1] += (f1*log(f1/f0));
//else fEntropy[1] += (-f1*log(f1/f0));
}
//check entropy in each channel

}

fEntropy[1] = 80* fEntropy[1];
}
else fEntropy[1] = 0;
}
else
{
fEntropy[1] = 0;
}

fSum = cBuffer[2][index]+cBuffer[2][index-1]+cBuffer[2][index+1]+
cBuffer[2][index-width]+cBuffer[2][index+width]+
cBuffer[2][index-width+1]+cBuffer[2][index-width-1]+cBuffer[2][index+width-1]+
cBuffer[2][index+width+1];

if (fSum !=0)
{
fEntropy[2] = 0;

f0 = cBuffer[2][index]/fSum;

if (f0!=0)
{

for ( j1 = j-1; j1<=j+1; j1++)
for ( i1 = i-1; i1<=i+1; i1++)
{
int index2 = index+(j1-j)*width+i1-i;
float f1 = cBuffer[2][index2]/fSum;

if (cBuffer[2][index2]!=0)
{
if (f1>f0) fEntropy[2] += (f1*log(f1/f0));
//else fEntropy[2] += (-f1*log(f1/f0));
}

//check entropy in each channel

}

fEntropy[2] = 80* fEntropy[2];
}
else fEntropy[2] = 0;
}
else
{
fEntropy[2] = 0;
}

//iplGetPixel(OriginalImage,i,j,ch);
//entropyBuffer[index] = max(max(fEntropy[0],fEntropy[1]),fEntropy[2]);
entropyBuffer[index] = (fEntropy[0]+fEntropy[1]+fEntropy[2])/3;

maxv1=max(entropyBuffer[index],maxv1);
minv1=min(entropyBuffer[index],minv1);
```

```
sum1+=entropyBuffer[index];
}


minv1 = log(1+minv1);
maxv1 = log(1+maxv1);

sum1 = sum1/(width*height-2*height-2*width+4);


for(  j = 1; j < height-1; j++ )
for(  i = 1; i < width-1; i++ )
{
float fEntropy = entropyBuffer[j*width+i];

DestGreyImage->imageData[j*DestGreyImage->widthStep+i]  = (unsigned char)((log(1+fEntropy)-minv1)*255/(maxv1-minv1)+(

}

delete [] entropyBuffer;

return 0;

}

int EvaluateEdgeResults(IplImage *ref, IplImage *test, double *recall,
double *precision)
{
int iTP = 0, iFP = 0, iMissing = 0, x,y;

if (ref == NULL || test == NULL)
return -1;

for (y=0; y<ref->height; y++)
for (x=0; x<ref->width; x++)
{
int index = y*ref->widthStep+x;

if ( ref->imageData[index] !=0 )
{
if (test->imageData[index] != 0) //correct
iTP ++;
else iMissing ++;
}
else
{
if (test->imageData[index] != 0) //correct
iFP ++;
}

}

*recall = 1.0*iTP/(iTP+iMissing);
*precision = 1.0*iTP/(iTP+iFP);

return 0;

}

char* cOriginalImageName[] = { "C:/standard_seq/images/pepper.bmp"};
//lena256.bmp green-girl.bmp pepper
```

```
char FirstName[]={"pepper"}; //green-girl lena256

char* cImageNameGT[] = { "C:/standard_seq/images/pepper-gt3.bmp"};
//"lena256.bmp" green-girl.bmp, house using Gt2 others using GT


enum edgeMethod {RAW, DIFF_MAX, DIFF_MEAN, ENTROPY_MAX, ENTROPY_MEAN,
CROSS_ENTROPY_MAX, CROSS_ENTROPY_MEAN};


extern void edgeTest(unsigned char *image[3], int rows, int cols,
float sigma,float tlow, float thigh, unsigned char *edge, int *hist,
 unsigned char *nms, short int *smoothedim, short int *delta_x,
short int *delta_y, short int *magnitude, edgeMethod iFlag);

enum ResultEdges {Grey, RGB, YCbCr, mapE, MapS, Green, Rjc, Final,
HSV, EigenGrey};

#define DebugText true

int mainTest() //TestEigenEdges
{
int iResult;

int width, height;

IplImage* OriginalImage;

IplImage* OriginalImageGray,  *EdgeImage;


OriginalImage= cvvLoadImage(cOriginalImageName[0]);


width = OriginalImage->width;
height = OriginalImage->height;

OriginalImageGray = cvCreateImage(cvSize(width,height),IPL_DEPTH_8U, 1);

EdgeImage = cvCreateImage(cvSize(width,height),IPL_DEPTH_8U, 1);


ChrominanceEdgePreserveGreyTransform(OriginalImage, OriginalImageGray);



// cvSaveImage("eigen.png",OriginalImageGray);

// ImageAdaptiveCanny(OriginalImageGray,EdgeImage);

// cvvSaveImage("eigen_edge.png",EdgeImage);


cvReleaseImage(&OriginalImage);
cvReleaseImage(&OriginalImageGray);
cvReleaseImage(&EdgeImage);



return 0;
```

```
}

int mainTestTime(); //test running time

int main()
{
int iResult;

IplImage* OriginalImage,*srcImgNew,*srcImgNew2;

IplImage* OriginalImageGray, *Edge0;

IplImage* OriginalImageMapping;

IplImage* OriginalImageMapping2;

IplImage* OriginalImageEdge[5];
int x,y,height,width, index1, index2;

float *srcArray, *destArray, fSigma;

char str[100];/*used for generate the file name using sprintf*/
unsigned char *cBuffer[3];
float sigma = 0.5;
int hist[32768];

double Mean=0,StdDev=0;

FILE *file = NULL;
double recall,precision;
char sFileName[40];
float Pre[VarianceLevel][10],Recall[VarianceLevel][10], f1PR[VarianceLevel][10];
//Grey, RGB, YCbCr, mapE, MapS, Green, Rjc, Final
float PreFiltered[VarianceLevel][4][10],RecallFiltered[VarianceLevel][4][10],
f1PRFiltered[VarianceLevel][4][10];



int iThresholdEdge = 6;


//return mainTestTime();



OriginalImage= cvvLoadImage(cOriginalImageName[0]);


width = OriginalImage->width;
height = OriginalImage->height;

OriginalImageGray=cvCreateImage(cvSize(width,height),IPL_DEPTH_8U, 1);

OriginalImageMapping=cvCreateImage(cvSize(width,height),IPL_DEPTH_8U, 1);

OriginalImageMapping2=cvCreateImage(cvSize(width,height),IPL_DEPTH_8U, 1);



pFeature = new unsigned char[(width+4)*(height+4)];
```

```
 nms = new BYTE[width*height];
 magnitude = new short[width*height];
 delta_x = new short[width*height];
 delta_y = new short[width*height];
 smoothedim = new short[width*height*3]; //enable RGB three channels

for (int i=0;i<5;i++)
OriginalImageEdge[i]=cvCreateImage(cvSize(width,height),IPL_DEPTH_8U, 1);

cBuffer[0] = new unsigned char[width*height];
cBuffer[1] = new unsigned char[width*height];
cBuffer[2] = new unsigned char[width*height];



//iResult=GetStandardImageByThreshold( OriginalImage,OriginalImage);
  /* gaussian_smooth(cBuffer[0], height, width, sigma, &smoothedim);
   for (y=0; y<height; y++)
{
for (x=0; x<width; x++)
cBuffer[0][y*width+x] = smoothedim[y*width+x];
}
*/


int iImageSize = width*height*OriginalImage->nChannels;

srcArray = new float[iImageSize];
destArray = new float[iImageSize];

Image2Array(OriginalImage,srcArray);

srcImgNew = cvCreateImage(cvSize(width,height),IPL_DEPTH_8U,
OriginalImage->nChannels);
srcImgNew2 = cvCreateImage(cvSize(width,height),IPL_DEPTH_8U,
OriginalImage->nChannels);

//Edge0 = cvCreateImage(cvSize(width,height),IPL_DEPTH_8U,1);


Edge0 = cvLoadImage(cImageNameGT[0],0);

sprintf(sFileName,"%s.txt",FirstName);

if (DebugText)
file = fopen(sFileName,"wt");
else file = NULL;

StartAgain:

if (file!=NULL)
{
fprintf(file,"\n\nInput image %s  EdgeThreshold %d\n\n",cOriginalImageName[0],
iThresholdEdge);
fprintf(file,"Noise Pre Recall Mean Stdev Method\n");
}


for (int m=0; m<VarianceLevel; m++)
{
printf("dealing with noise level %d\n",m);
//fSigma = 0.5f*(m+1)/VarianceLevel; //for uniform distribution
```

```
if (m!=0)
{
fSigma = 0.01f*m/VarianceLevel; //for Gaussian distribution
AddArrayNoise(srcArray, destArray,GaussianNoise, fMean, fSigma, width,
height, OriginalImage->nChannels, true); //normalised

Array2Image(destArray,srcImgNew2, 255.0); //scaled

sprintf(str,"%s_Src_n%d.bmp",FirstName,m);
cvvSaveImage( str, srcImgNew2);

iplMedianFilter(srcImgNew2,srcImgNew,3,3,1,1);

// iplCopy(srcImgNew2,srcImgNew);

//iplColorMedianFilter(srcImgNew2,srcImgNew,3,3,1,1);
}
else iplCopy(OriginalImage,srcImgNew);


index1 = 0;
index2 = 0;
for (y=0; y<height; y++)
{
for (x=0; x<width; x++)
{
cBuffer[0][index2] = (unsigned char) srcImgNew->imageData[index1+3*x+2];
cBuffer[1][index2] = (unsigned char) srcImgNew->imageData[index1+3*x+1];
cBuffer[2][index2] = (unsigned char) srcImgNew->imageData[index1+3*x];

index2++;
}

index1 += OriginalImage->widthStep;
}

/*
printf("dealing with buffer 0\n");
ExtractEntropy(cBuffer[0],OriginalImageEdge[0]);
//cvvNamedWindow( "Entropy Blue", 1);
//cvvShowImage( "Entropy Blue", OriginalImageEdge[0]);

sprintf(str,"%s_Entropy_n%dB.bmp",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[0]);
// SaveImageBMP(str,OriginalImageEdge[0],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[0], &recall, &precision);
fprintf(file,"%d %.4f %.4f %.3f %.3f Entropy-B Edge result\n",
m,precision,recall,Mean,StdDev);

printf("dealing with buffer 1\n");
ExtractEntropy(cBuffer[1],OriginalImageEdge[1]);
//cvvNamedWindow( "Entropy Green", 1);
//cvvShowImage( "Entropy Green", OriginalImageEdge[1]);
sprintf(str,"%s_Entropy_n%dG.bmp",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[1]);
// SaveImageBMP(str,OriginalImageEdge[1],NULL);
EvaluateEdgeResults(Edge0, OriginalImageEdge[1], &recall, &precision);
fprintf(file,"%d %.4f %.4f %.3f %.3f Entropy-G Edge result\
n",m,precision,recall,Mean,StdDev);
```

```
printf("dealing with buffer 2\n");
ExtractEntropy(cBuffer[2],OriginalImageEdge[2]);
//cvvNamedWindow( "Entropy Red", 1);
//cvvShowImage( "Entropy Red", OriginalImageEdge[2]);
sprintf(str,"%s_Entropy_n%dR.bmp",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[2]);
// SaveImageBMP(str,OriginalImageEdge[2],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[2], &recall, &precision);
fprintf(file,"%d %.4f %.4f %.3f %.3f Entropy-R Edge result\n",
m,precision,recall,Mean,StdDev);


//cross entropy

printf("dealing with buffer 0\n");
ExtractCrossEntropy(cBuffer[0],OriginalImageEdge[0]);
//cvvNamedWindow( "CrossEntropy Blue", 1);
//cvvShowImage( "CrossEntropy Blue", OriginalImageEdge[0]);

sprintf(str,"%s_EntropyC_n%dB.bmp",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[0]);
// SaveImageBMP(str,OriginalImageEdge[0],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[0], &recall, &precision);
fprintf(file,"%d %.4f %.4f %.3f %.3f CrossEntropy-B Edge result\n",
m,precision,recall,Mean,StdDev);



printf("dealing with buffer 1\n");
ExtractCrossEntropy(cBuffer[1],OriginalImageEdge[1]);
//cvvNamedWindow( "CrossEntropy Green", 1);
//cvvShowImage( "CrossEntropy Green", OriginalImageEdge[1]);
sprintf(str,"%s_EntropyC_n%dG.bmp",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[1]);
// SaveImageBMP(str,OriginalImageEdge[1],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[1], &recall, &precision);
fprintf(file,"%d %.4f %.4f %.3f %.3f CrossEntropy-G Edge result\n",
m,precision,recall,Mean,StdDev);




printf("dealing with buffer 2\n");
ExtractCrossEntropy(cBuffer[2],OriginalImageEdge[2]);
//cvvNamedWindow( "CrossEntropy Red", 1);
//cvvShowImage( "CrossEntropy Red", OriginalImageEdge[2]);
sprintf(str,"%s_EntropyC_n%dR.bmp",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[2]);
// SaveImageBMP(str,OriginalImageEdge[2],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[2], &recall, &precision);

fprintf(file,"%d %.4f %.4f %.3f %.3f CrossEntropy-R Edge result\n",
m,precision,recall,Mean,StdDev);


*/
//eigen-grey canny
```

```
ChrominanceEdgePreserveGreyTransform(srcImgNew, OriginalImageGray);

cvMean_StdDev(OriginalImageGray,&Mean,&StdDev,0);
printf("Original Gray Mean=%.3f, Std=%.3f\n",Mean,StdDev);

ImageAdaptiveCanny(OriginalImageGray,OriginalImageEdge[1]);
//cvvNamedWindow( "Original Edge G", 1);
//cvvShowImage( "Original Edge G", OriginalImageEdge[1]);

sprintf(str,"%s_Edge_eigeny%d.png",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[1]);
//SaveImageBMP(str,OriginalImageEdge[1],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[1], &recall, &precision);

Pre[m][EigenGrey] = precision;
Recall[m][EigenGrey] = recall;


if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f EigenGrey Edge result\n",
m,precision,recall,Mean,StdDev);

TrackingRemoveNoise(OriginalImageEdge[1],OriginalImageEdge[4],iThresholdEdge);

sprintf(str,"%s_Edge_eigen%df.png",FirstName,m);
SaveImageBMP(str,OriginalImageEdge[4],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[4], &recall, &precision);

PreFiltered[m][(iThresholdEdge-6)/3][EigenGrey] = precision;
RecallFiltered[m][(iThresholdEdge-6)/3][EigenGrey] = recall;

if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f Filtered EigenGrey Edge result\n"
,m,precision,recall,Mean,StdDev);



//gray canny

iplColorToGray(srcImgNew,OriginalImageGray);

//cvvNamedWindow( "Original Gray", 1);
//cvvShowImage( "Original Gray", OriginalImageGray);


cvMean_StdDev(OriginalImageGray,&Mean,&StdDev,0);
printf("Original Gray Mean=%.3f, Std=%.3f\n",Mean,StdDev);

ImageAdaptiveCanny(OriginalImageGray,OriginalImageEdge[1]);
//cvvNamedWindow( "Original Edge G", 1);
//cvvShowImage( "Original Edge G", OriginalImageEdge[1]);

sprintf(str,"%s_Edge_grey%d.png",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[1]);
//SaveImageBMP(str,OriginalImageEdge[1],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[1], &recall, &precision);

Pre[m][Grey] = precision;
Recall[m][Grey] = recall;
```

```
if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f Grey Edge result\n"
,m,precision,recall,Mean,StdDev);

TrackingRemoveNoise(OriginalImageEdge[1],OriginalImageEdge[4],iThresholdEdge);

sprintf(str,"%s_Edge_grey%df.png",FirstName,m);
SaveImageBMP(str,OriginalImageEdge[4],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[4], &recall, &precision);

PreFiltered[m][(iThresholdEdge-6)/3][Grey] = precision;
RecallFiltered[m][(iThresholdEdge-6)/3][Grey] = recall;

if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f FilteredGrey Edge result\n"
,m,precision,recall,Mean,StdDev);



//combined entropy maximum component

iResult=ColorMappingW3( cBuffer,OriginalImageEdge[3]);
if (iResult<0)
{
printf("Memory failure!\n");
goto finish;
}
sprintf(str,"%s_Entropy_RGB%d.png",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[3]);
//SaveImageBMP(str,OriginalImageEdge[3],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[3], &recall, &precision);
// fprintf(file,"%d %.4f %.4f %.3f %.3f Mapping-Entropy
Edge result\n",m,precision,recall,Mean,StdDev);


//combined cross-entropy maximum component


iResult=ColorMappingW3Cross( cBuffer,OriginalImageEdge[3]);
if (iResult<0)
{
printf("Memory failure!\n");
goto finish;
}
sprintf(str,"%s_EntropyC_RGB%d.png",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[3]);
//SaveImageBMP(str,OriginalImageEdge[3],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[3], &recall, &precision);
// fprintf(file,"%d %.4f %.4f %.3f %.3f EntropyRGB Edge result
\n",m,precision,recall,Mean,StdDev);




//enum edgeMethod {RAW, DIFF_MAX, DIFF_MEAN, ENTROPY_MAX, ENTROPY_MEAN,
CROSS_ENTROPY_MAX, CROSS_ENTROPY_MEAN};
/*
```

```
edgeTest(cBuffer, height, width, 0.7, 0.3, 0.8, pFeature, hist, nms,
smoothedim,delta_x,delta_y,magnitude,DIFF_MAX);
iplSet(OriginalImageEdge[3],0);

for (y=0; y<height; y++)
for (x=0; x<width; x++)
{
OriginalImageEdge[3]->imageData[y*OriginalImageEdge[3]->widthStep+x]
= pFeature[y*width+x];
}

cvvNamedWindow( "DiffMax", 1);
cvvShowImage( "DiffMax", OriginalImageEdge[3]);
sprintf(str,"%s_DiffMax.bmp",FirstName);
cvvSaveImage( str, OriginalImageEdge[3]);
SaveImageBMP(str,OriginalImageEdge[3],NULL);

edgeTest(cBuffer, height, width, 0.7, 0.3, 0.8, pFeature, hist, nms,
smoothedim,delta_x,delta_y,magnitude,DIFF_MEAN);
iplSet(OriginalImageEdge[3],0);

for (y=0; y<height; y++)
for (x=0; x<width; x++)
{
OriginalImageEdge[3]->imageData[y*OriginalImageEdge[3]->widthStep+x] =
pFeature[y*width+x];
}

cvvNamedWindow( "DiffMean", 1);
cvvShowImage( "DiffMean", OriginalImageEdge[3]);
sprintf(str,"%s_DiffMean.bmp",FirstName);
cvvSaveImage( str, OriginalImageEdge[3]);
SaveImageBMP(str,OriginalImageEdge[3],NULL);
*/
int w[3];

w[0]=1;
w[1]=1;
w[2]=1;

iResult=ColorMapping( srcImgNew,OriginalImageMapping,w);
if (iResult<0)
{
printf("Memory failure!\n");
goto finish;
}

printf("Map E, Gmaxv1=%d minv1=%d Dmaxv2=%d minv2=%d Tmaxv=%d minv=%d\n",w[1]/1000,w[1]%1000,w[2]/1000,w[2]%1000,w[0]

sprintf(str,"%s_Map_E%d.png",FirstName,m);
cvvSaveImage( str, OriginalImageMapping);
//SaveImageBMP(str,OriginalImageMapping,NULL);


//cvvNamedWindow( "Original Map E", 1);
//cvvShowImage( "Original Map E", OriginalImageMapping);

w[0]=72;
w[1]=715;
w[2]=213;
```

```
//w[0]=299;
//w[1]=587;
//w[2]=114;

iResult=ColorMapping( srcImgNew,OriginalImageMapping2,w);

if (iResult<0)
{
printf("Memory failure!\n");
goto finish;
}
printf("Map S, Gmaxv1=%d minv1=%d Dmaxv2=%d minv2=%d Tmaxv=%d minv=%d\n",w[1]/1000,w[1]%1000,w[2]/1000,w[2]%1000,w[0]

sprintf(str,"%s_Map_S%d.png",FirstName,m);
cvvSaveImage( str, OriginalImageMapping2);
//SaveImageBMP(str,OriginalImageMapping2,NULL);


//cvvNamedWindow( "Original Map S", 1);
//cvvShowImage( "Original Map S", OriginalImageMapping2);

printf("Color mapping Successfully!\n");


//edge
ImageAdaptiveCanny(srcImgNew,OriginalImageEdge[3]);

sprintf(str,"%s_Edge_C%d.png",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[3]);
//SaveImageBMP(str,OriginalImageEdge[0],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[3], &recall, &precision);

Pre[m][RGB] = precision;
Recall[m][RGB] = recall;

if (file!=NULL)

fprintf(file,"%d %.4f %.4f %.3f %.3f RGB Edge result\n"
,m,precision,recall,Mean,StdDev);

TrackingRemoveNoise(OriginalImageEdge[3],OriginalImageEdge[2],iThresholdEdge);
sprintf(str,"%s_Edge_C%df.png",FirstName,m);
SaveImageBMP(str,OriginalImageEdge[2],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[2], &recall, &precision);
if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f FilteredRGB Edge
result\n",m,precision,recall,Mean,StdDev);

PreFiltered[m][(iThresholdEdge-6)/3][RGB] = precision;
RecallFiltered[m][(iThresholdEdge-6)/3][RGB] = recall;


//edge
ImageAdaptiveCanny(srcImgNew,OriginalImageEdge[0],1);

//cvvNamedWindow( "Original Edge C", 1);
//cvvShowImage( "Original Edge C", OriginalImageEdge[0]);

sprintf(str,"%s_Edge_Green%d.png",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[0]);
```

```
//SaveImageBMP(str,OriginalImageEdge[0],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[0], &recall, &precision);
Pre[m][Green] = precision;
Recall[m][Green] = recall;

if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f Green Edge result\
n",m,precision,recall,Mean,StdDev);

TrackingRemoveNoise(OriginalImageEdge[0],OriginalImageEdge[2],iThresholdEdge);

sprintf(str,"%s_Edge_Green%df.png",FirstName,m);
SaveImageBMP(str,OriginalImageEdge[2],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[2], &recall, &precision);

PreFiltered[m][(iThresholdEdge-6)/3][Green] = precision;
RecallFiltered[m][(iThresholdEdge-6)/3][Green] = recall;

if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f FilteredG Edge result
\n",m,precision,recall,Mean,StdDev);




cvMean_StdDev(OriginalImageMapping,&Mean,&StdDev,0);
printf("Mapping E Gray Mean=%.3f, Std=%.3f\n",Mean,StdDev);

ImageAdaptiveCanny(OriginalImageMapping,OriginalImageEdge[2]);
//cvvNamedWindow( "Original Edge E", 1);
//cvvShowImage( "Original Edge E", OriginalImageEdge[2]);

sprintf(str,"%s_Edge_E%d.png",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[2]);
//SaveImageBMP(str,OriginalImageEdge[2],NULL);
EvaluateEdgeResults(Edge0, OriginalImageEdge[2], &recall, &precision);

Pre[m][mapE] = precision;
Recall[m][mapE] = recall;

if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f Mapping-Equal Edge result\
n",m,precision,recall,Mean,StdDev);

TrackingRemoveNoise(OriginalImageEdge[2],OriginalImageEdge[3],iThresholdEdge);

sprintf(str,"%s_Edge_E%df.png",FirstName,m);
SaveImageBMP(str,OriginalImageEdge[3],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[3], &recall, &precision);

PreFiltered[m][(iThresholdEdge-6)/3][mapE] = precision;
RecallFiltered[m][(iThresholdEdge-6)/3][mapE] = recall;

if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f FilteredMapE Edge result
\n",m,precision,recall,Mean,StdDev);
```

```
cvMean_StdDev(OriginalImageMapping2,&Mean,&StdDev,0);
printf("Mapping S Gray Mean=%.3f, Std=%.3f\n",Mean,StdDev);

ImageAdaptiveCanny(OriginalImageMapping2,OriginalImageEdge[0]);
//cvvNamedWindow( "Original Edge S", 1);
//cvvShowImage( "Original Edge S", OriginalImageEdge[3]);

sprintf(str,"%s_Edge_S%d.png",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[0]);
//SaveImageBMP(str,OriginalImageEdge[3],NULL);
EvaluateEdgeResults(Edge0, OriginalImageEdge[0], &recall, &precision);

Pre[m][MapS] = precision;
Recall[m][MapS] = recall;

if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f Mapping-Weight Edge
result\n",m,precision,recall,Mean,StdDev);

TrackingRemoveNoise(OriginalImageEdge[0],OriginalImageEdge[3],iThresholdEdge);

sprintf(str,"%s_Edge_S%df.png",FirstName,m);
SaveImageBMP(str,OriginalImageEdge[3],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[3], &recall, &precision);

PreFiltered[m][(iThresholdEdge-6)/3][MapS] = precision;
RecallFiltered[m][(iThresholdEdge-6)/3][MapS] = recall;

if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f FilteredMapS Edge
result\n",m,precision,recall,Mean,StdDev);




iplOr(OriginalImageEdge[2],OriginalImageEdge[1],OriginalImageEdge[3]);
sprintf(str,"%s_Edge_OR%d.png",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[3]);
//SaveImageBMP(str,OriginalImageEdge[3],NULL);
EvaluateEdgeResults(Edge0, OriginalImageEdge[3], &recall, &precision);

Pre[m][Rjc] = precision;
Recall[m][Rjc] = recall;

if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f Rjc Edge result\
n",m,precision,recall,Mean,StdDev);


TrackingRemoveNoise(OriginalImageEdge[3],OriginalImageEdge[0],iThresholdEdge);
sprintf(str,"%s_Edge_OR%df.png",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[0]);
//SaveImageBMP(str,OriginalImageEdge[3],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[0], &recall, &precision);

PreFiltered[m][(iThresholdEdge-6)/3][Rjc] = precision;
RecallFiltered[m][(iThresholdEdge-6)/3][Rjc] = recall;
```

```
if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f FilteredRjc Edge
result\n",m,precision,recall,Mean,StdDev);



iplOr(OriginalImageEdge[2],OriginalImageEdge[4],OriginalImageEdge[3]);
sprintf(str,"%s_Edge_OR%df2.png",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[3]);
//SaveImageBMP(str,OriginalImageEdge[3],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[3], &recall, &precision);

Pre[m][Final] = precision;
Recall[m][Final] = recall;

if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f Rjc-GreyFiltered
Edge result\n",m,precision,recall,Mean,StdDev);


TrackingRemoveNoise(OriginalImageEdge[3],OriginalImageEdge[0],iThresholdEdge);
sprintf(str,"%s_Edge_OR%dAll.png",FirstName,m);
cvvSaveImage( str, OriginalImageEdge[0]);

EvaluateEdgeResults(Edge0, OriginalImageEdge[0], &recall, &precision);

PreFiltered[m][(iThresholdEdge-6)/3][Final] = precision;
RecallFiltered[m][(iThresholdEdge-6)/3][Final] = recall;

if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f FilteredRjc=
GreyFiltred Edge result\n",m,precision,recall,Mean,StdDev);




//YCbCr edge

iplRGB2YCrCb(srcImgNew,srcImgNew2);

iResult=ImageAdaptiveCanny( srcImgNew2,OriginalImageEdge[3]);
if (iResult==0)
{

sprintf(str,"%s_Edge_YCbCr%d.png",FirstName,m);
SaveImageBMP(str,OriginalImageEdge[3],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[3], &recall, &precision);

Pre[m][YCbCr] = precision;
Recall[m][YCbCr] = recall;

if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f YCbCr Edge result\
n",m,precision,recall,Mean,StdDev);

TrackingRemoveNoise(OriginalImageEdge[3],OriginalImageEdge[0],iThresholdEdge);
sprintf(str,"%s_Edge_YCbCr%df.png",FirstName,m);
SaveImageBMP(str,OriginalImageEdge[0],NULL);
EvaluateEdgeResults(Edge0, OriginalImageEdge[0], &recall, &precision);
if (file!=NULL)
```

```
fprintf(file,"%d %.4f %.4f %.3f %.3f FilteredYCbCr Edge
result\n",m,precision,recall,Mean,StdDev);


PreFiltered[m][(iThresholdEdge-6)/3][YCbCr] = precision;
RecallFiltered[m][(iThresholdEdge-6)/3][YCbCr] = recall;

}

iplRGB2HSV(srcImgNew,srcImgNew2);
for ( i=0;i<srcImgNew->height;i++)
{
for (int j=0;j<srcImgNew->width;j++)
{
unsigned char ch[4];

iplGetPixel(srcImgNew2,j,i,ch);

if (ch[0]>128)
ch[0]=(unsigned char)(255-ch[0]);

iplPutPixel(srcImgNew2,j,i,ch);
}
}

iResult=ImageAdaptiveCanny( srcImgNew2,OriginalImageEdge[3]);
if (iResult==0)
{

sprintf(str,"%s_Edge_HSV%d.png",FirstName,m);
SaveImageBMP(str,OriginalImageEdge[3],NULL);

EvaluateEdgeResults(Edge0, OriginalImageEdge[3], &recall, &precision);

Pre[m][HSV] = precision;
Recall[m][HSV] = recall;

if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f HSV Edge result\
n",m,precision,recall,Mean,StdDev);

TrackingRemoveNoise(OriginalImageEdge[3],OriginalImageEdge[0],iThresholdEdge);
sprintf(str,"%s_Edge_HSV%df.png",FirstName,m);
SaveImageBMP(str,OriginalImageEdge[0],NULL);
EvaluateEdgeResults(Edge0, OriginalImageEdge[0], &recall, &precision);
if (file!=NULL)
fprintf(file,"%d %.4f %.4f %.3f %.3f FilteredHSV Edge
result\n",m,precision,recall,Mean,StdDev);


PreFiltered[m][(iThresholdEdge-6)/3][HSV] = precision;
RecallFiltered[m][(iThresholdEdge-6)/3][HSV] = recall;

}

for (iResult = 0; iResult<10; iResult++)
{
float f1 = Pre[m][iResult]+Recall[m][iResult];

if (f1 !=0)
f1PR[m][iResult] = 2*Pre[m][iResult]*Recall[m][iResult]/f1;
else f1PR[m][iResult] = 0;
```

```
f1 = PreFiltered[m][(iThresholdEdge-6)/3][iResult]+RecallFiltered[m]
[(iThresholdEdge-6)/3][iResult];

if (f1 !=0)
f1PRFiltered[m][(iThresholdEdge-6)/3][iResult] = 2*PreFiltered[m]
[(iThresholdEdge-6)/3][iResult]*RecallFiltered[m]

[(iThresholdEdge-6)/3][iResult]/f1;
else f1PRFiltered[m][(iThresholdEdge-6)/3][iResult] = 0;

}



}

if (iThresholdEdge<15)
{
iThresholdEdge +=3;

goto StartAgain;
}



finish:

cvvWaitKey(0);

if (file!=NULL)
{
int n;

fprintf(file,"\n%d Grey RGB YCbCr mapE Maps
Green Rjc Final  HSV  Eigen Threshold\n",m);

for (m=0; m<VarianceLevel; m++)
{
fprintf(file,"NoiseLevel %.3f\n",0.01f*m);

fprintf(file,"%d %.4f %.4f %.4f %.4f %.4f %.4f %.4f
%.4f %.4f %.4f %.4f  %.4f  %.4f %.4f %.4f %.4f %.4f
%.4f %.4f %.4f %d\n",m,
Pre[m][0],Recall[m][0],Pre[m][1],Recall[m][1],Pre[m][2],Recall[m][2],
Pre[m][3],Recall[m][3],
Pre[m][4],Recall[m][4],Pre[m][5],Recall[m][5],Pre[m][6],Recall[m][6],
Pre[m][7],Recall[m][7],Pre[m][8],Recall[m][8],
Pre[m][9],Recall[m][9],0);

for (n=0; n<4; n++) //threshold level
{
fprintf(file,"%d %.4f %.4f %.4f %.4f %.4f %.4f
%.4f %.4f %.4f %.4f  %.4f %.4f  %.4f %.4f %.4f
%.4f %.4f %.4f %.4f %.4f %d\n",m,

PreFiltered[m][n][0],RecallFiltered[m][n][0],PreFiltered[m][n][1],
RecallFiltered[m][n][1],PreFiltered[m][n][2],
RecallFiltered[m][n][2],PreFiltered[m][n][3],RecallFiltered[m][n][3],
PreFiltered[m][n][4],RecallFiltered[m][n][4],
PreFiltered[m][n][5],RecallFiltered[m][n][5],PreFiltered[m][n][6],
RecallFiltered[m][n][6],PreFiltered[m][n][7],
RecallFiltered[m][n][7],PreFiltered[m][n][8],RecallFiltered[m][n][8],
PreFiltered[m][n][9],RecallFiltered[m][n][9], n*3+6);
}

fprintf(file,"\n");
}

fprintf(file,"\n%d Grey RGB YCbCr mapE Maps Green Rjc Final
HSV Eigen (F1)\n");

for (m=0; m<VarianceLevel; m++)
{
fprintf(file,"%d %.4f %.4f %.4f %.4f %.4f %.4f
%.4f %.4f %.4f %.4f  f1-Original noise= %.3f\n",m,
f1PR[m][0],f1PR[m][1],f1PR[m][2],f1PR[m][3],f1PR[m][4],f1PR[m][5],f1PR[m]
[6],f1PR[m][7],f1PR[m][8], f1PR[m][9],0.01f*m);
}

for (n=0; n<4; n++) //threshold level
{
fprintf(file,"Threshold %d\n",n*3+6);
for (m=0; m<VarianceLevel; m++)
{
fprintf(file,"%d %.4f %.4f %.4f %.4f %.4f %.4f %.4f
%.4f %.4f %.4f  f1-filtered noise= %.3f\n",m,
f1PRFiltered[m][n][0],f1PRFiltered[m][n][1],f1PRFiltered[m][n][2],
f1PRFiltered[m][n][3],f1PRFiltered[m][n][4],
f1PRFiltered[m][n][5],f1PRFiltered[m][n][6],f1PRFiltered[m][n][7],
f1PRFiltered[m][n][8],f1PRFiltered[m][n][9],0.01f*m);
}

}


for (m=0; m<VarianceLevel; m++)
{

fprintf(file,"NoiseLevel %.3f\n",0.01f*m);


fprintf(file,"%d %.4f %.4f %.4f %.4f %.4f %.4f
%.4f %.4f %.4f %.4f  f1-Original th=%d\n",m,
f1PR[m][0],f1PR[m][1],f1PR[m][2],f1PR[m][3],f1PR[m][4],f1PR[m][5],
f1PR[m][6],f1PR[m][7],f1PR[m][8], f1PR[m][9],0);


for (n=0; n<4; n++) //threshold level
{
fprintf(file,"%d %.4f %.4f %.4f %.4f %.4f %.4f %.4f
%.4f %.4f %.4f  f1-filtered th=%d\n",m,
f1PRFiltered[m][n][0],f1PRFiltered[m][n][1],f1PRFiltered[m][n][2],
f1PRFiltered[m][n][3],f1PRFiltered[m][n][4],
f1PRFiltered[m][n][5],f1PRFiltered[m][n][6],f1PRFiltered[m][n][7],
f1PRFiltered[m][n][8],f1PRFiltered[m][n][9],n*3+6);
}
}



fclose(file);
}
```

```
delete [] smoothedim;
delete [] delta_x;
delete [] delta_y;
delete [] magnitude;
delete [] nms;
delete [] pFeature;


delete [] cBuffer[0];
delete [] cBuffer[1];
delete [] cBuffer[2];

delete [] srcArray;
delete [] destArray;


cvReleaseImage(&OriginalImage);
cvReleaseImage(&Edge0);
cvReleaseImage(&srcImgNew);
cvReleaseImage(&srcImgNew2);
cvReleaseImage(&OriginalImageGray);
cvReleaseImage(&OriginalImageMapping);
cvReleaseImage(&OriginalImageMapping2);
for (i=0;i<5;i++)
cvReleaseImage(&OriginalImageEdge[i]);

return 0;
}


int mainNoise(void)
{

int i;

IplImage *srcImg = NULL;
IplImage *srcImgNew = NULL;
IplImage *dstImg = NULL;



int width=0, height=0, w2, h2;
char fileName[_MAX_PATH];
char strNoiseType[20];


int iImageSize = 0;

int nChannels;
float *srcArray, *destArray, fSigma;

if (myAddedNoise == GaussianNoise)
strcpy(strNoiseType,"Gaussian");
else if (myAddedNoise == UniformNoise)
strcpy(strNoiseType,"Uniform");
else if (myAddedNoise == SaltPepperNoise)
strcpy(strNoiseType,"Salt && Pepper");
else if (myAddedNoise == SpeckleNoise)
strcpy(strNoiseType,"Speckle");

FILE *file = fopen("static.txt","wt");
```

```
// for ( i=iStartFrame; i<=iEndFrame;i++)
{
// printf("Deal with image %03d of %03d\n",i-iStartFrame,
iEndFrame-iStartFrame+1);

/*if (!SrcGrayFlag)
sprintf(fileName,SrcName,RawName,i+1);//ImageName
else
sprintf(fileName,GrayName,RawName,i+1);//ImageName
*/

//sprintf(fileName,GrayName,i);//ImageName
sprintf(fileName,GrayName,RawName);//ImageName

myLoadImage(&srcImg,fileName, -1);

if (srcImg == NULL)
goto terminate;

//continue;
if (CropFlag )
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_c0.pgm",RawName);
CropImage(srcImg, fileName, &w2, &h2);

sprintf(fileName,"c:/standard_seq/Yosi Code/%s_c0.png",RawName);
CropImage(srcImg, fileName, &w2, &h2);

}
else
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_big_c0.pgm",RawName);
LargeImage(srcImg, fileName,&w2,&h2);

sprintf(fileName,"c:/standard_seq/Yosi Code/%s_big_c0.png",RawName);
LargeImage(srcImg, fileName,&w2,&h2);

}

if (w2 == srcImg->width && h2 == srcImg->height)
{
BlackBoundaryImage(srcImg,2);
cvSaveImage(fileName,srcImg);

sprintf(fileName,"c:/standard_seq/Yosi Code/%s_c0.png",RawName);
cvSaveImage(fileName,srcImg);

}

myLoadImage(&srcImg,fileName, -1);


if (srcImgNew == NULL)
{
width = srcImg->width;
height = srcImg->height;

nChannels = srcImg->nChannels;

srcImgNew = cvCreateImage(cvSize(width,height),IPL_DEPTH_8U,nChannels);
```

```
iImageSize = width*height*nChannels;

srcArray = new float[iImageSize];
destArray = new float[iImageSize];

if ( srcImgNew == NULL|| srcArray == NULL || destArray == NULL )
return -CVSSP_OUT_OF_MEMORY;


}

/*
if (nChannels>1)
{
//iplColorToGray(srcImgNext,srcGrayNext);

//sprintf(fileName,GrayName,RawName,i+1);
//cvSaveImage(fileName,srcGray);
}

//continue;
if (CropFlag )
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_c0.pgm",RawName);
CropImage(srcImg, fileName);

}
else
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_big_c0.pgm",RawName);
LargeImage(srcImg, fileName);
}



for (int kk=0;kk<4; kk++)
{
//Image shear
float sx,sy,scale;

if (ShearZoom)
{
if (stricmp(RawName,"paris") == 0)
{

sx = offset_Paris2[kk][0];
sy = offset_Paris2[kk][1];
scale = offset_Paris2[kk][2];
}
else
{
sx = offset_Pentagon2[kk][0];
sy = offset_Pentagon2[kk][1];
scale = offset_Pentagon2[kk][2];

}
}
else
{
if (stricmp(RawName,"paris") == 0)
```

```
{

sx = offset_Paris2[kk][0]*offset_Paris2[kk][2];
sy = offset_Paris2[kk][1]*offset_Paris2[kk][2];
scale = 1.0;
}
else
{
sx = offset_Pentagon2[kk][0]*offset_Pentagon2[kk][2];
sy = offset_Pentagon2[kk][1]*offset_Pentagon2[kk][2];
scale = offset_Pentagon2[kk][2];

}

}

iplShear(srcImg,srcImgNew,0.0,0.0,sx,sy, IPL_INTER_LINEAR); //IPL_INTER_CUBIC

if (ShearZoom)
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_lz%d.png",RawName,kk+1);
ZoomImage(srcImgNew,fileName,scale,IPL_INTER_LINEAR,&dstImg);

if (CropFlag )
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_lz%d_1.pgm",RawName,kk+1);
CropImage(dstImg, fileName);
}
else
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_big_lz%d_1.pgm",RawName,kk+1);
LargeImage(dstImg, fileName);
}


ZoomImage(srcImg,NULL,scale,IPL_INTER_LINEAR,&dstImg);
if (CropFlag )
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_lz%d_0.pgm",RawName,kk+1);
CropImage(dstImg, fileName);
}
else
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_big_lz%d_0.pgm",RawName,kk+1);
LargeImage(dstImg, fileName);
}
}
else
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_l%d.png",RawName,kk+1);
cvSaveImage(fileName,srcImgNew);


if (CropFlag )
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_l%d.pgm",RawName,kk+1);
CropImage(srcImgNew, fileName);
}
else
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_big_l%d.pgm",RawName,kk+1);
LargeImage(srcImgNew, fileName);
}
}
//cvAbsDiff(srcImg,srcImgNew,srcImgNew);
//cvSaveImage("diff1.png",srcImgNew);

iplShear(srcImg,srcImgNew,0,0,sx,sy, IPL_INTER_CUBIC); //IPL_INTER_CUBIC

if (ShearZoom)
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_Cz%d.png",RawName,kk+1);
ZoomImage(srcImgNew,fileName,scale,IPL_INTER_CUBIC,&dstImg);

if (CropFlag )
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_Cz%d_1.pgm",RawName,kk+1);
CropImage(dstImg, fileName);
}
else
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_big_Cz%d_1.pgm",RawName,kk+1);
LargeImage(dstImg, fileName);

}

ZoomImage(srcImg,NULL,scale,IPL_INTER_CUBIC,&dstImg);
if (CropFlag )
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_Cz%d_0.pgm",RawName,kk+1);
CropImage(dstImg, fileName);
}
else
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_big_Cz%d_0.pgm",RawName,kk+1);
LargeImage(dstImg, fileName);

}

}
else
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_c%d.png",RawName,kk+1);
cvSaveImage(fileName,srcImgNew);


if (CropFlag )
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_C%d.pgm",RawName,kk+1);
CropImage(srcImgNew, fileName);
}
else
{
sprintf(fileName,"c:/standard_seq/Yosi Code/%s_big_C%d.pgm",RawName,kk+1);
LargeImage(srcImgNew, fileName);

}
}

cvReleaseImage(&dstImg);
dstImg = 0;

//cvAbsDiff(srcImg,srcImgNew,srcImgNew);
```

```
//cvSaveImage("diff2.png",srcImgNew);


}

*/



Image2Array(srcImg,srcArray);


for (int m=0; m<VarianceLevel; m++)
{
//fSigma = 0.5f*(m+1)/VarianceLevel; //for uniform distribution
fSigma = 0.05f*(m+1)/VarianceLevel; //for Gaussian distribution
AddArrayNoise(srcArray, destArray,GaussianNoise, fMean, fSigma, width,
  height, nChannels, true); //normalised

Array2Image(destArray,srcImgNew, 255.0); //scaled

//save result
//sprintf(fileName,NewSrcName,i,strNoiseType,fMean,fSigma);
//cvSaveImage(fileName,srcImgNew);

//sprintf(fileName,"test%d_%d.pgm",i,m);
sprintf(fileName,"pentagon_n%d.png",m+1);
cvSaveImage(fileName,srcImgNew);
sprintf(fileName,"pentagon_n%d.pgm",m+1);
cvSaveImage(fileName,srcImgNew);

fSigma = 0.8f*(m+1)/VarianceLevel; //for uniform distribution
//fSigma = 0.05f*(m+1)/VarianceLevel; //for Gaussian distribution
AddArrayNoise(srcArray, destArray,UniformNoise, fMean, fSigma, width,
  height, nChannels, true); //noamalised

Array2Image(destArray,srcImgNew, 255.0); //scaled

//save result
//sprintf(fileName,NewSrcName,i,strNoiseType,fMean,fSigma);
//cvSaveImage(fileName,srcImgNew);

//sprintf(fileName,"test%d_%d.pgm",i,m);
sprintf(fileName,"pentagon_u%d.png",m+1);
cvSaveImage(fileName,srcImgNew);
sprintf(fileName,"pentagon_u%d.pgm",m+1);
cvSaveImage(fileName,srcImgNew);
}


/*float mse, snr;


fprintf(file,"Source Image %i Address %s\n",i,fileName);

for (int m=0; m<VarianceLevel; m++)
{
mse = 0;
snr = 0;

sprintf(fileName,"F:/standard_seq/Hoge/Gaussian/test%d_%d.pgm",m+1,i);
```

```
myLoadImage(&srcImgNew,fileName, -1);

StatisticComparingImages(srcImg, srcImgNew, &mse , &snr );


fprintf(file," Noisy level %d Address %s \n",m+1,fileName);
fprintf(file," MSE %.5f SNR %.5f \n\n",mse, snr);

}*/



} /* for i*/

fclose(file);

terminate:

printf("End! \n Press any key for quit...\n");
cvvWaitKey(0);

cvDestroyAllWindows();

delete[] srcArray;
delete[] destArray;


cvReleaseImage( &srcImg );
cvReleaseImage( &srcImgNew );


return 0;


}



int mainTestTime() //test running time
{

struct _timeb timebuffer;
time_t timeStart, timeEnd;
    unsigned short millitmStart,millitmEnd, stime,mtime;

int iResult;

IplImage* OriginalImage,*srcImgNew,*srcImgNew2;

IplImage* OriginalImageGray, *Edge0;

IplImage* OriginalImageMapping;

IplImage* OriginalImageMapping2;

IplImage* OriginalImageEdge[5];
int x,y,height,width, index1, index2;

float *srcArray, *destArray, fSigma;

char str[100];/*used for generate the file name using sprintf*/
unsigned char *cBuffer[3];
```

```
float sigma = 0.5;
int hist[32768];

double Mean=0,StdDev=0;

FILE *file = NULL;
double recall,precision;
char sFileName[40];
float Pre[VarianceLevel][10],Recall[VarianceLevel][10], f1PR[VarianceLevel][10];
//Grey, RGB, YCbCr, mapE, MapS, Green, Rjc, Final
float PreFiltered[VarianceLevel][4][10],RecallFiltered[VarianceLevel][4][10],
f1PRFiltered[VarianceLevel][4][10];



int iThresholdEdge = 12;



OriginalImage= cvvLoadImage(cOriginalImageName[0]);


width = OriginalImage->width;
height = OriginalImage->height;

OriginalImageGray=cvCreateImage(cvSize(width,height),IPL_DEPTH_8U, 1);

OriginalImageMapping=cvCreateImage(cvSize(width,height),IPL_DEPTH_8U, 1);

OriginalImageMapping2=cvCreateImage(cvSize(width,height),IPL_DEPTH_8U, 1);



pFeature = new unsigned char[(width+4)*(height+4)];

 nms = new BYTE[width*height];
 magnitude = new short[width*height];
 delta_x = new short[width*height];
 delta_y = new short[width*height];
 smoothedim = new short[width*height*3]; //enable RGB three channels

for (int i=0;i<5;i++)
OriginalImageEdge[i]=cvCreateImage(cvSize(width,height),IPL_DEPTH_8U, 1);

cBuffer[0] = new unsigned char[width*height];
cBuffer[1] = new unsigned char[width*height];
cBuffer[2] = new unsigned char[width*height];



//iResult=GetStandardImageByThreshold( OriginalImage,OriginalImage);
/* gaussian_smooth(cBuffer[0], height, width, sigma, &smoothedim);
   for (y=0; y<height; y++)
{
for (x=0; x<width; x++)
cBuffer[0][y*width+x] = smoothedim[y*width+x];
}
*/


int iImageSize = width*height*OriginalImage->nChannels;
```

```
srcArray = new float[iImageSize];
destArray = new float[iImageSize];

Image2Array(OriginalImage,srcArray);

srcImgNew = cvCreateImage(cvSize(width,height),IPL_DEPTH_8U,
OriginalImage->nChannels);
srcImgNew2 = cvCreateImage(cvSize(width,height),IPL_DEPTH_8U,
OriginalImage->nChannels);

//Edge0 = cvCreateImage(cvSize(width,height),IPL_DEPTH_8U,1);


Edge0 = cvLoadImage(cImageNameGT[0],0);

sprintf(sFileName,"%s_time.txt",FirstName);

if (DebugText)
file = fopen(sFileName,"wt");
else file = NULL;

int testEdge = 0; //0: RGB 1: YCbCr 2: eigen 3: Our




StartAgain:


 _ftime( &timebuffer ); // C4996

 timeStart = timebuffer.time;
millitmStart = timebuffer.millitm;

for (int m=0; m<100; m++)
{

iplCopy(OriginalImage,srcImgNew);

index1 = 0;
index2 = 0;
for (y=0; y<height; y++)
{
for (x=0; x<width; x++)
{
cBuffer[0][index2] = (unsigned char) srcImgNew->imageData[index1+3*x+2];
cBuffer[1][index2] = (unsigned char) srcImgNew->imageData[index1+3*x+1];
cBuffer[2][index2] = (unsigned char) srcImgNew->imageData[index1+3*x];

index2++;
}

index1 += OriginalImage->widthStep;
}


if (testEdge == 2) //eigen edge
{
//eigen-grey canny
ChrominanceEdgePreserveGreyTransform(srcImgNew, OriginalImageGray);

cvMean_StdDev(OriginalImageGray,&Mean,&StdDev,0);
ImageAdaptiveCanny(OriginalImageGray,OriginalImageEdge[1]);

TrackingRemoveNoise(OriginalImageEdge[1],OriginalImageEdge[4],iThresholdEdge);

}
else if (testEdge == 0) //RGB
{
ImageAdaptiveCanny(srcImgNew,OriginalImageEdge[3]);


TrackingRemoveNoise(OriginalImageEdge[3],OriginalImageEdge[2],iThresholdEdge);

}
else if (testEdge == 1) //YCbCr
{
iplRGB2YCrCb(srcImgNew,srcImgNew2);

iResult=ImageAdaptiveCanny( srcImgNew2,OriginalImageEdge[3]);
if (iResult==0)
{

TrackingRemoveNoise(OriginalImageEdge[3],OriginalImageEdge[0],iThresholdEdge);

}

}
else if (testEdge == 3) //our method
{
int w[3];

iplColorToGray(srcImgNew,OriginalImageGray);

cvMean_StdDev(OriginalImageGray,&Mean,&StdDev,0);
ImageAdaptiveCanny(OriginalImageGray,OriginalImageEdge[1]);


w[0]=72;
w[1]=715;
w[2]=213;

//iResult=ColorMapping( srcImgNew,OriginalImageMapping,w);
iResult=ColorMappingFast( srcImgNew,OriginalImageMapping,OriginalImageGray,
w,StdDev);

cvMean_StdDev(OriginalImageMapping,&Mean,&StdDev,0);

ImageAdaptiveCanny(OriginalImageMapping,OriginalImageEdge[2]);


iplOr(OriginalImageEdge[2],OriginalImageEdge[1],OriginalImageEdge[3]);

    TrackingRemoveNoise(OriginalImageEdge[3],OriginalImageEdge[0],iThresholdEdge);


}





}
```

```
if (file!=NULL)
{

_ftime( &timebuffer ); // C4996
// Note: _ftime is deprecated; consider using _ftime_s instead
timeEnd = timebuffer.time;
millitmEnd = timebuffer.millitm;

if (millitmEnd>=millitmStart)
{
stime = (unsigned short)(timeEnd-timeStart);
mtime = millitmEnd-millitmStart;
}
else
{
stime = (unsigned short)(timeEnd-timeStart-1);
mtime = 1000+millitmEnd-millitmStart;

}


if (testEdge == 0)
fprintf(file,"\nProcessing time in running 100 times RGB edges %d.%03d
seconds\n",stime,mtime);
else if (testEdge == 1)
fprintf(file,"\nProcessing time in running 100 times YCbCr edges %d.%03d
seconds\n",stime,mtime);
else if (testEdge == 2)
fprintf(file,"\nProcessing time in running 100 times Eigen edges %d.%03d
seconds\n",stime,mtime);
else if (testEdge == 3)
fprintf(file,"\nProcessing time in running 100 times Our edges %d.%03d
seconds\n",stime,mtime);

}

testEdge ++;

if (testEdge<=3) goto StartAgain;



finish:

cvvWaitKey(0);

if (file!=NULL)
{


fclose(file);
}

delete [] smoothedim;
delete [] delta_x;
delete [] delta_y;
delete [] magnitude;
delete [] nms;
delete [] pFeature;
delete [] cBuffer[0];
delete [] cBuffer[1];
delete [] cBuffer[2];

delete [] srcArray;
delete [] destArray;


cvReleaseImage(&OriginalImage);
cvReleaseImage(&Edge0);
cvReleaseImage(&srcImgNew);
cvReleaseImage(&srcImgNew2);
cvReleaseImage(&OriginalImageGray);
cvReleaseImage(&OriginalImageMapping);
cvReleaseImage(&OriginalImageMapping2);
for (i=0;i<5;i++)
cvReleaseImage(&OriginalImageEdge[i]);

return 0;
}
```

Skin and Face Detection

```
#include "skin.h"

char* cOriginalImageName = "C:/map_movie/skin3.bmp";
 //skin1 lena-color red-girl
char *FirstName="skin3";
char str[100];/*used for generate the file name using sprintf*/

#define PI 3.14159265359
//lena-color
//green-girl
//red-girl

//BABOON.bmp CAMERAM.BMP  GIRL.BMP map.bmp Temp0001.bmp
//band7.bmp

int MyRGB2IHS(IplImage * OriginalImage, IplImage * OriginalImage2)
{
assert(OriginalImage->width==OriginalImage2->width);
assert(OriginalImage->height==OriginalImage2->height);
assert(OriginalImage->nChannels==3 && OriginalImage2->nChannels==3);

for (int i=0;i<OriginalImage->height;i++)
{
for (int j=0;j<OriginalImage->width;j++)
{
unsigned char ch[4];

iplGetPixel(OriginalImage,j,i,ch);

int r=ch[2];
int g=ch[1];
int b=ch[0];

int sum=r+g+b;

int I=(sum+2)/3;
int min_rgb=min(r,min(g,b));
double d_s=1-3.0*min_rgb/sum;
int S=(int)(255*d_s+0.5);
double H0=acos(0.5*(2*r-g-b)/sqrt(((r-g)*(r-g)+(r-b)*(g-b))));
```

```
//if (b>g) H0=2*PI-H0;
if (H0<0) H0=2*PI+H0;
else H0=2*PI-H0;

int H=(int)(H0/(2*PI)*255);

//int H=(int)(127*cos(H0)+127+0.5);

ch[0]=(unsigned char)S;
ch[1]=(unsigned char)H;
ch[2]=(unsigned char)I;
iplPutPixel(OriginalImage2,j,i,ch);
}
}

return 0;
}
void GetSkinMaskYCrCb(IplImage * srcImage, IplImage * mask_BW, int erosions,
int dilations)
{

//assert(srcImage->nChannels==3 && mask_BW->nChannels==3);

CvSize sz = cvSize( srcImage->width & -2, srcImage->height & -2);

//get the size of input_image (src_RGB)
IplImage* pyr = cvCreateImage( cvSize(sz.width/2, sz.height/2), 8, 3 );
//create 2 temp-images
IplImage* src = cvCreateImage(  cvSize(srcImage->width,
srcImage->height),IPL_DEPTH_8U, 3);
IplImage* src2 = cvCreateImage(  cvSize(srcImage->width,
srcImage->height),IPL_DEPTH_8U, 3);

cvCopyImage(srcImage, src);
cvPyrDown( src, pyr, 7 );

//remove noise from input
cvPyrUp( pyr, src2, 7 );

//cvCvtColor(src2 ,src , CV_RGB2YCrCb);//color conversion

// cvCvtColor(srcImage ,src , CV_RGB2YCrCb);//color conversion
//iplRGB2YCrCb(srcImage,src);
iplRGB2YCrCb(src2,src);

uchar Y;
uchar Cr;
uchar Cb;

iplSet(mask_BW,0);

for( int y=0;y<srcImage-> height; y++)
{

for (int x=0; x<srcImage->width; x++)
{
unsigned char ch[4];

iplGetPixel(src,x,y,ch);
Y = ch[0];
Cb= ch[1];
```

```
Cr= ch[2];

//iplGetPixel(srcImage,x,y,ch);

if( Cr > 138 && Cr < 178 &&Cb + 0.6 * Cr >189 && Cb + 0.6 * Cr <215)
{
//if ( Y>78 && Y<190 &&Cr/1.3>Cb && (Cr+Cb)/1.5>Y ) //1.345 //
if (Cr/1.3>Cb)
{


iplPutPixel(mask_BW,x,y,ch);

/*if ((Y>=Cb)||(Y>=Cr))
iplPutPixel(mask_BW,x,y,ch);
else if (Y*1.9>Cr)
iplPutPixel(mask_BW,x,y,ch);*/

}
//else if (Y>60 && Y<120 && Cr/1.45>Cb && (Cb+Cr)/3.5>Y)
// iplPutPixel(mask_BW,x,y,ch);
}

}
}

sprintf(str,"%s_YCbCr_Skin2.bmp",FirstName);
SaveImageBMP(str,mask_BW,NULL);


if(erosions>0)
cvErode(mask_BW,mask_BW,0,erosions);

if(dilations>0)
cvDilate(mask_BW,mask_BW,0,dilations);

if(dilations-erosions>0)
cvErode(mask_BW,mask_BW,0,dilations-erosions-1);

cvReleaseImage( &pyr );
cvReleaseImage( &src );
cvReleaseImage( &src2 );

}
void GetSkinMaskYCrCbNew(IplImage * srcImage, IplImage * mask_BW,
int erosions, int dilations)
{

//assert(srcImage->nChannels==3 && mask_BW->nChannels==3);

CvSize sz = cvSize( srcImage->width & -2, srcImage->height & -2);

//get the size of input_image (src_RGB)
IplImage* pyr = cvCreateImage( cvSize(sz.width/2, sz.height/2), 8, 3 );
//create 2 temp-images
IplImage* src = cvCreateImage(  cvSize(srcImage->width,
srcImage->height),IPL_DEPTH_8U, 3);
IplImage* src2 = cvCreateImage(  cvSize(srcImage->width,
srcImage->height),IPL_DEPTH_8U, 3);

cvCopyImage(srcImage, src);
cvPyrDown( src, pyr, 7 );
```

```
//remove noise from input
cvPyrUp( pyr, src2, 7 );

//cvCvtColor(src2 ,src , CV_RGB2YCrCb);//color conversion

// cvCvtColor(srcImage ,src , CV_RGB2YCrCb);//color conversion
//iplRGB2YCrCb(srcImage,src);
iplRGB2YCrCb(src2,src);

uchar Y;
char Cr;
char Cb;

iplSet(mask_BW,0);

for( int y=0;y<srcImage-> height; y++)
{

for (int x=0; x<srcImage->width; x++)
{
unsigned char ch[4];

iplGetPixel(src,x,y,ch);
Y = ch[0];
Cb= ch[1]-128;
Cr= ch[2]-128;

double th1,th2,th3,th4;

if (Y>128)
{
th1=-2+1.0*(256-Y)/16;
th2=-th1+18;
th3=6;
th4=-8;
}
else
{
th1=6;
th2=12;
th3=2+1.0*Y/32;
th4=-16+1.0*Y/16;
}

if (Cr<-2*(Cb+24)) continue;
if (Cr<-(Cb+17)) continue;
if (Cr<-4*(Cb+32)) continue;
if (Cr<2.5*(Cb+th1)) continue;
if (Cr<th3) continue;
if (Cr<0.5*(th4-Cb)) continue;
if (Cr>(220-Cb)/6) continue;
if (Cr>4*(th2-Cb)/3) continue;

iplPutPixel(mask_BW,x,y,ch);

}
}

if(erosions>0)
cvErode(mask_BW,mask_BW,0,erosions);
```

```
if(dilations>0)
cvDilate(mask_BW,mask_BW,0,dilations);

if(dilations-erosions>0)
cvErode(mask_BW,mask_BW,0,dilations-erosions-1);

cvReleaseImage( &pyr );
cvReleaseImage( &src );
cvReleaseImage( &src2 );

}

void GetSkinMaskIHS(IplImage * srcImage, IplImage * mask_BW, int erosions,
int dilations)
{

//assert(srcImage->nChannels==3 && mask_BW->nChannels==3);

CvSize sz = cvSize( srcImage->width & -2, srcImage->height & -2);

//get the size of input_image (src_RGB)
IplImage* pyr = cvCreateImage( cvSize(sz.width/2, sz.height/2), 8, 3 );
//create 2 temp-images
IplImage* src = cvCreateImage(  cvSize(srcImage->width,
srcImage->height),IPL_DEPTH_8U, 3);
IplImage* src2 = cvCreateImage(  cvSize(srcImage->width,
srcImage->height),IPL_DEPTH_8U, 3);

cvCopyImage(srcImage, src);
cvPyrDown( src, pyr, 7 );

//remove noise from input
cvPyrUp( pyr, src2, 7 );

MyRGB2IHS(src2,src);
//MyRGB2IHS(srcImage,src);
sprintf(str,"%s_IHS2.bmp",FirstName);
SaveImageBMP(str,src,NULL);

uchar I;
uchar H;
uchar S;

iplSet(mask_BW,0);

for( int y=0;y<srcImage-> height; y++)
{

for (int x=0; x<srcImage->width; x++)
{
unsigned char ch[4];

iplGetPixel(src,x,y,ch);
S = ch[0];
H= ch[1];
I= ch[2];

// iplGetPixel(srcImage,x,y,ch);

if (x==287 && y==167)
printf("Enter! S=%uc H=%uc I=%uc \n",S,H,I);
```

```
double kk=H-240; //H-240

if (kk<0) kk=0;

if (H<230 || S<H/10 || S+I<100) continue;

if( H+S<400 && S*5>I  && S+I>105) //S>35 or S*5>I//&&  H>380-S H+I<380
400 //&& I>S/(4.1+sqrt(kk)/3) S>H/6
{
//iplPutPixel(mask_BW,x,y,ch);
if (S+I> 180 && H>230 && S+I+H>460 && S+I+45>H+abs(S-I))
iplPutPixel(mask_BW,x,y,ch);
else if (S+I>135 && H>=236 && H+I+S>390 && S<120 && S<I*3)
iplPutPixel(mask_BW,x,y,ch);
else if ( H>=239 && S<I*5 && S+15>I) //S<(I+4*(250-H))*3 //H>=238 234
{
if (H<250)
iplPutPixel(mask_BW,x,y,ch);
else if (I*2>S)
iplPutPixel(mask_BW,x,y,ch);
}
}
else if (H+S>400 && S>H/6  && H>=237 && S+I>=H  && S<180) //S<173 S+I-8>H
&& I>S/4.3 && I>S/3.6 //&& S<I*(4.1+sqrt(kk)/3) //
{

if (S+I> 180 && H>230 && S+I+H>460 )
iplPutPixel(mask_BW,x,y,ch);
else if (S+I>135 && H>=236 && H+I+S>390 && S<I*3 && S<120)
iplPutPixel(mask_BW,x,y,ch);
else if (H>240)
iplPutPixel(mask_BW,x,y,ch);
}

}
}

sprintf(str,"%s_IHS_Skin2.bmp",FirstName);
SaveImageBMP(str,mask_BW,NULL);

cvDilate(mask_BW,mask_BW,0,1);

if(erosions>0)
cvErode(mask_BW,mask_BW,0,erosions+1);

if(dilations>0)
cvDilate(mask_BW,mask_BW,0,dilations);

if(dilations-erosions>0)
cvErode(mask_BW,mask_BW,0,dilations-erosions);

for ( int i=0;i<mask_BW->height;i++)
{
for (int j=0;j<mask_BW->width;j++)
{
unsigned char ch[4];

iplGetPixel(mask_BW,j,i,ch);

int k=10*(ch[1]-248);
if (k<0) k=0;
if (ch[0]+ch[2]<106+k)
{
ch[0]=0;
ch[1]=0;
ch[2]=0;
iplPutPixel(mask_BW,j,i,ch);
}
}
}
cvErode(mask_BW,mask_BW,0,1);
cvDilate(mask_BW,mask_BW,0,1);



cvReleaseImage( &pyr );
cvReleaseImage( &src );
cvReleaseImage( &src2 );

}

int MyRGB2HSV(IplImage * OriginalImage, IplImage * OriginalImage2)
{
assert(OriginalImage->width==OriginalImage2->width);
assert(OriginalImage->height==OriginalImage2->height);
assert(OriginalImage->nChannels==3 && OriginalImage2->nChannels==3);

iplRGB2HSV(OriginalImage,OriginalImage2);

for (int  i=0;i<OriginalImage2->height;i++)
{
for (int j=0;j<OriginalImage2->width;j++)
{
unsigned char ch[4];

iplGetPixel(OriginalImage2,j,i,ch);

if (ch[0]>128)
ch[0]=(unsigned char)(255-ch[0]);

iplPutPixel(OriginalImage2,j,i,ch);
}
}

return 0;
}
void GetSkinMaskHSV(IplImage * srcImage, IplImage * mask_BW, int erosions,
 int dilations)
{

//assert(srcImage->nChannels==3 && mask_BW->nChannels==3);

CvSize sz = cvSize( srcImage->width & -2, srcImage->height & -2);

//get the size of input_image (src_RGB)
IplImage* pyr = cvCreateImage( cvSize(sz.width/2, sz.height/2), 8, 3 );
 //create 2 temp-images
IplImage* src = cvCreateImage(  cvSize(srcImage->width,
srcImage->height),IPL_DEPTH_8U, 3);
IplImage* src2 = cvCreateImage(  cvSize(srcImage->width,
srcImage->height),IPL_DEPTH_8U, 3);
```

```
cvCopyImage(srcImage, src);
cvPyrDown( src, pyr, 7 );

//remove noise from input
cvPyrUp( pyr, src2, 7 );

MyRGB2HSV(src2,src);
//MyRGB2IHS(srcImage,src);
sprintf(str,"%s_HSV2.bmp",FirstName);
SaveImageBMP(str,src,NULL);

uchar V;
uchar H;
uchar S;

iplSet(mask_BW,0);

for( int y=0;y<srcImage-> height; y++)
{

for (int x=0; x<srcImage->width; x++)
{
unsigned char ch[4];

iplGetPixel(src,x,y,ch);
S = ch[0];
H= ch[1];
V= ch[2];
// iplGetPixel(srcImage,x,y,ch);

if (x==330 && y==99)
printf("Entered!\n");

if( S<=21 && V>=S*2.5 && (S+1)*V>H && (H+V)>S*13 && S<25 && H+V>=158 &&
H+V<=400 && H*5>V && H>S*4 && 8.0*(2*S*S+abs(V-H))>V+H) //  H>S*5 (H+V)>S*15
{

int kk=0;

//examples for hair, VHS=(53,149,19) or (48,164,17)

/*if ((H>200)&&(S<19)) kk=230-H;
else  kk=208-H;

int kk2=kk/5; //kk2=0;
*/
if (x==38 && y==71)
kk=0;

if (S<9 && abs(V-H)>9*S && V+H>320)
continue;

if (S>=18 && V<H && V<S*7.2)
continue;

if (S>=17 && abs(H-V)>S*4.5 && H<200) //if (S>=16 && abs(H-V)>S*4) //18
continue;


if (H<200 ) kk=208-H;
else if (S<19 )
{
```

```
if (V<75 ) //&& S>=15
kk=230-H;
else
kk=213-H; //210-

}

int kk2=kk/5; //kk2=0;


if (H<180) //210 //160 //215
iplPutPixel(mask_BW,x,y,ch);
else if ( V+kk*5>=S*(6+kk2) && V+kk*5<S*30) //( kk*4>S) //( V+kk*(4-kk2)>=S*6)
iplPutPixel(mask_BW,x,y,ch);

/*if (V>40 && V< 228)
iplPutPixel(mask_BW,x,y,ch);
else if (V>228 && V<H*4.2)
iplPutPixel(mask_BW,x,y,ch);*/

}

}
}

sprintf(str,"%s_HSV_Skin2.bmp",FirstName);
SaveImageBMP(str,mask_BW,NULL);

if(erosions>0)
cvErode(mask_BW,mask_BW,0,erosions);

if(dilations>0)
cvDilate(mask_BW,mask_BW,0,dilations);

if(dilations-erosions>0)
cvErode(mask_BW,mask_BW,0,dilations-erosions);

cvReleaseImage( &pyr );
cvReleaseImage( &src );
cvReleaseImage( &src2 );

}


int MyRGB2HLS(IplImage * OriginalImage, IplImage * OriginalImage2)
{
assert(OriginalImage->width==OriginalImage2->width);
assert(OriginalImage->height==OriginalImage2->height);
assert(OriginalImage->nChannels==3 && OriginalImage2->nChannels==3);

iplRGB2HLS(OriginalImage,OriginalImage2);

for (int  i=0;i<OriginalImage2->height;i++)
{
for (int j=0;j<OriginalImage2->width;j++)
{
unsigned char ch[4];

iplGetPixel(OriginalImage2,j,i,ch);

if (ch[0]>128)
ch[0]=(unsigned char)(255-ch[0]);
```

```
iplPutPixel(OriginalImage2,j,i,ch);
}
}

return 0;
}
void GetSkinMaskHLS(IplImage * srcImage, IplImage * mask_BW, int erosions,
int dilations)
{

//assert(srcImage->nChannels==3 && mask_BW->nChannels==3);

CvSize sz = cvSize( srcImage->width & -2, srcImage->height & -2);

//get the size of input_image (src_RGB)
IplImage* pyr = cvCreateImage( cvSize(sz.width/2, sz.height/2), 8, 3 );
//create 2 temp-images
IplImage* src = cvCreateImage(  cvSize(srcImage->width,
srcImage->height),IPL_DEPTH_8U, 3);
IplImage* src2 = cvCreateImage(  cvSize(srcImage->width,
srcImage->height),IPL_DEPTH_8U, 3);

cvCopyImage(srcImage, src);
cvPyrDown( src, pyr, 7 );

//remove noise from input
cvPyrUp( pyr, src2, 7 );

MyRGB2HLS(src2,src);
//MyRGB2IHS(srcImage,src);
sprintf(str,"%s_HLS2.bmp",FirstName);
SaveImageBMP(str,src,NULL);

uchar L,V;
uchar H;
uchar S;

iplSet(mask_BW,0);

for( int y=0;y<srcImage-> height; y++)
{

for (int x=0; x<srcImage->width; x++)
{
unsigned char ch[4];

iplGetPixel(src,x,y,ch);
S = ch[0];
H= (unsigned char)(ch[2]*1.35);
L= ch[1];
V=L;
if (S<13) V=(unsigned char)(1.4*V);

// iplGetPixel(srcImage,x,y,ch);


if( L>=S*3 && (S+1)*V>H && (H+V)>S*13 && S<25 && H+V>=158 && H+V<=400
&& H*5>V && H>S*4 && 8.0*(2*S*S+abs(V-H))>V+H) //  H>S*5 (H+V)>S*15
{

int kk=0;
/*if ((H>200)&&(S<19)) kk=230-H;
else  kk=208-H;

int kk2=kk/5; //kk2=0;
*/

if (H<200 ) kk=208-H;
else if (S<19 && H>=200)
{
if (V<75 ) //&& S>=15
kk=230-H;
else
kk=210-H;

}

int kk2=kk/5; //kk2=0;


if (H<180) //210 //160 //215
iplPutPixel(mask_BW,x,y,ch);
else if ( V+kk*5>=S*(6+kk2) && V+kk*5<S*30) //( kk*4>S) //( V+kk*(4-kk2)>=S*6)
iplPutPixel(mask_BW,x,y,ch);

/*if (V>40 && V< 228)
iplPutPixel(mask_BW,x,y,ch);
else if (V>228 && V<H*4.2)
iplPutPixel(mask_BW,x,y,ch);*/

}

}
}

sprintf(str,"%s_HLS_Skin2.bmp",FirstName);
SaveImageBMP(str,mask_BW,NULL);

if(erosions>0)
cvErode(mask_BW,mask_BW,0,erosions);

if(dilations>0)
cvDilate(mask_BW,mask_BW,0,dilations);

if(dilations-erosions>0)
cvErode(mask_BW,mask_BW,0,dilations-erosions);

cvReleaseImage( &pyr );
cvReleaseImage( &src );
cvReleaseImage( &src2 );

}
void GetSkinMaskYUV(IplImage * srcImage, IplImage * mask_BW, int erosions,
int dilations)
{

//assert(srcImage->nChannels==3 && mask_BW->nChannels==3);

CvSize sz = cvSize( srcImage->width & -2, srcImage->height & -2);

//get the size of input_image (src_RGB)
IplImage* pyr = cvCreateImage( cvSize(sz.width/2, sz.height/2), 8, 3 );
```

```
//create 2 temp-images
IplImage* src = cvCreateImage(  cvSize(srcImage->width,
srcImage->height),IPL_DEPTH_8U, 3);
IplImage* src2 = cvCreateImage(  cvSize(srcImage->width,
srcImage->height),IPL_DEPTH_8U, 3);

cvCopyImage(srcImage, src);
cvPyrDown( src, pyr, 7 );

//remove noise from input
cvPyrUp( pyr, src2, 7 );

//cvCvtColor(src2 ,src , CV_RGB2YCrCb);//color conversion

// cvCvtColor(srcImage ,src , CV_RGB2YCrCb);//color conversion
//iplRGB2YCrCb(srcImage,src);
iplRGB2YUV(src2,src);
sprintf(str,"%s_YUV2.bmp",FirstName);
SaveImageBMP(str,src,NULL);


uchar Y;
uchar U;
uchar V;

iplSet(mask_BW,0);

for( int y=0;y<srcImage-> height; y++)
{

for (int x=0; x<srcImage->width; x++)
{
unsigned char ch[4];

iplGetPixel(src,x,y,ch);
Y = ch[0];
U= ch[1];
V= ch[2];

//iplGetPixel(srcImage,x,y,ch);


if (x==455 && y==283)
printf("Enter! Y=%uc U=%uc V=%uc \n",Y,U,V);

if ((V<148)||(V>185)) continue;


if(U + 0.6 * V >189 && U + 0.6 * V <215 )
{
if (Y>85)
iplPutPixel(mask_BW,x,y,ch);
else if (U>104 && Y+U-V>2)
iplPutPixel(mask_BW,x,y,ch);
}

}
}
sprintf(str,"%s_YUV_Skin2.bmp",FirstName);
SaveImageBMP(str,mask_BW,NULL);

if(erosions>0)
```

```
cvErode(mask_BW,mask_BW,0,erosions);

if(dilations>0)
cvDilate(mask_BW,mask_BW,0,dilations);

if(dilations-erosions>0)
cvErode(mask_BW,mask_BW,0,dilations-erosions-1);

cvReleaseImage( &pyr );
cvReleaseImage( &src );
cvReleaseImage( &src2 );

}

int FindShapeResults(IplImage* OriginalImage,IplImage* ResultImage8,IplImage*
ResultImage24,char *col_str)
{
int i,j,k;
IplImage* OriginalImage8;
IplImage* OriginalImage9;
IplImage* OriginalImage10;
IplImage* TempImage;
IplImage* TempImage24;
IplImage* RotatedImage;
IplImage* RotatedImage24;
IplImage* OriginalImage24;


OriginalImage8=cvCreateImage(  cvSize(OriginalImage->width,
OriginalImage->height),IPL_DEPTH_8U, 1);

OriginalImage9=cvCreateImage(  cvSize(OriginalImage->width,
OriginalImage->height),IPL_DEPTH_8U, 1);

OriginalImage10=cvCreateImage(  cvSize(OriginalImage->width,
OriginalImage->height),IPL_DEPTH_8U, 1);

TempImage=cvCreateImage(  cvSize(OriginalImage->width,
OriginalImage->height),IPL_DEPTH_8U, 1);

TempImage24=cvCreateImage(  cvSize(OriginalImage->width,
OriginalImage->height),IPL_DEPTH_8U, 3);

RotatedImage=cvCreateImage(  cvSize(OriginalImage->width,
OriginalImage->height),IPL_DEPTH_8U, 1);

RotatedImage24=cvCreateImage(  cvSize(OriginalImage->width,
OriginalImage->height),IPL_DEPTH_8U, 3);

OriginalImage24=cvCreateImage(  cvSize(OriginalImage->width,
OriginalImage->height),IPL_DEPTH_8U, 3);

//get the grey mask
unsigned char ch[4];
for (i=0;i<OriginalImage8->height;i++)
{
int nOffset=i*OriginalImage8->widthStep;
for (j=0;j<OriginalImage8->width;j++)
{
iplGetPixel(OriginalImage,j,i,ch);

if (ch[0]!=0 && ch[1]!=0 && ch[2]!=0)
```

```
OriginalImage8->imageData[nOffset+j]=(unsigned char)255;
}
}
//labeling and remove small regions
DetectedObjs objs1[100],objs2[100];

cvvNamedWindow( "Grey Image", 1);
cvvShowImage( "Grey Image", OriginalImage8); /*the original map image*/

int iCount1=FastImageLabeling(OriginalImage8,OriginalImage9,300,objs1,0);
//for (i=0;i<iCount1;i++)
// printf("Lable first, No %d x=%d y=%d\n",i,objs1[i].x,objs1[i].y);
printf("Labeled objects %d\n",iCount1);

//original momemnt
iplCopy(OriginalImage,OriginalImage24);

ch[0]=0;
ch[1]=0;
ch[2]=0;
ch[3]=0;

for (i=0;i<OriginalImage9->height;i++)
{
int nOffset=i*OriginalImage9->widthStep;
for (j=0;j<OriginalImage9->width;j++)
{
if (OriginalImage9->imageData[nOffset+j]==0)
iplPutPixel(OriginalImage24,j,i,ch);
}
}
   //the original image without rotation
iplCopy(OriginalImage9,ResultImage8);
iplCopy(OriginalImage24,ResultImage24);


cvvNamedWindow( "Grey Image Label 300", 1);
cvvShowImage( "Grey Image Label 300", OriginalImage24); /*the original map image*/
sprintf(str,"%s_skin_th300%s.bmp",FirstName,col_str);
SaveImageBMP(str,OriginalImage24,NULL);

DrawObjMoments(objs1, OriginalImage24, iCount1);

cvvNamedWindow( "Moment", 1);
cvvShowImage( "Moment", OriginalImage24); /*the original map image*/
sprintf(str,"%s_skin_moment300%s.bmp",FirstName,col_str);
SaveImageBMP(str,OriginalImage24,NULL);

//rotate each regions

iplSet(RotatedImage,0);
iplSet(RotatedImage24,0);


for ( k=0;k<iCount1;k++)
{
iplSet(OriginalImage10,0);
for (i=0;i<OriginalImage9->height;i++)
{
int nOffset=i*OriginalImage9->widthStep;

for (j=0;j<OriginalImage9->width;j++)
```

```
{
if (OriginalImage9->imageData[nOffset+j]==k+1)
OriginalImage10->imageData[nOffset+j]=(unsigned char)255;
}
}

//rotate iplRotate()
iplRotateCenter(OriginalImage10,TempImage,objs1[k].angle/PI*180-90,
   objs1[k].x,objs1[k].y,IPL_INTER_LINEAR);//311,217

iplRotateCenter(OriginalImage,OriginalImage24,objs1[k].angle/PI*180-90,
   objs1[k].x,objs1[k].y,IPL_INTER_LINEAR);//311,217

for (i=0;i<TempImage->height;i++)
{
int nOffset=i*TempImage->widthStep;

for (j=0;j<TempImage->width;j++)
{
if ((unsigned char)TempImage->imageData[nOffset+j]>128)
{
RotatedImage->imageData[nOffset+j]=(unsigned char)255;

iplGetPixel(OriginalImage24,j,i,ch);
iplPutPixel(RotatedImage24,j,i,ch);
}
}
}
}

cvvNamedWindow( "Rotate Label", 1);
cvvShowImage( "Rotate Label", RotatedImage); /*the original map image*/

sprintf(str,"%s_skin_rotate_%s.bmp",FirstName,col_str);
SaveImageBMP(str,RotatedImage,NULL);

cvvNamedWindow( "Rotate24", 1);
cvvShowImage( "Rotate24", RotatedImage24); /*the original map image*/

sprintf(str,"%s_skin_rotate24_%s.bmp",FirstName,col_str);
SaveImageBMP(str,RotatedImage24,NULL);

//after rotation, find the ratio
int iCount2=FastImageLabeling(RotatedImage,OriginalImage10,300,objs2,0);
printf("Labeled objects after rotation%d\n",iCount2);
for (i=0;i<iCount2;i++)
printf("Lable second, No %d x=%d y=%d\n",i,objs2[i].x,objs2[i].y);

ch[0]=0;
ch[1]=0;
ch[2]=0;
ch[3]=0;
for (k=0;k<iCount2;k++)
{
double ratio;
int h=objs2[k].Lowright.y-objs2[k].Upleft.y;
int w=objs2[k].Lowright.x-objs2[k].Upleft.x;

ratio=1.0*h/w;
if (ratio>1) ratio=1/ratio;
//printf("Obj %d height=%d width=%d Ratio=%.3f\n",k,h,w,ratio);
```

```
if (ratio>0.55 && ratio<0.9) continue;

objs2[k].Pixels=-objs2[k].Pixels;

for (int m=0;m<iCount1;m++)
if (objs1[m].x==objs2[k].x && objs1[m].y==objs2[k].y) break;

for (i=0;i<OriginalImage10->height;i++)
{
int nOffset=i*OriginalImage10->widthStep;

for (j=0;j<OriginalImage10->width;j++)
{
if (OriginalImage10->imageData[nOffset+j]==k+1)
{
RotatedImage->imageData[nOffset+j]=(unsigned char)0;
iplPutPixel(RotatedImage24,j,i,ch);
}

if (OriginalImage9->imageData[nOffset+j]==m+1) //before rotation
{
ResultImage8->imageData[nOffset+j]=(unsigned char)0;
iplPutPixel(ResultImage24,j,i,ch);
}

}
}
}
cvvNamedWindow( "Rotate Label ratio", 1);
cvvShowImage( "Rotate Label ratio", RotatedImage); /*the original map image*/

sprintf(str,"%s_skin_rotate_ratio_%s.bmp",FirstName,col_str);
SaveImageBMP(str,RotatedImage,NULL);

cvvNamedWindow( "Rotate24 ratio", 1);
cvvShowImage( "Rotate24 ratio", RotatedImage24); /*the original map image*/

sprintf(str,"%s_skin_rotate24_ratio_%s.bmp",FirstName,col_str);
SaveImageBMP(str,RotatedImage24,NULL);

//recover the images before rotation
cvvNamedWindow( "Label ratio", 1);
cvvShowImage( "Label ratio", ResultImage8); /*the original map image*/

sprintf(str,"%s_skin_ratio_%s.bmp",FirstName,col_str);
SaveImageBMP(str,ResultImage8,NULL);

cvvNamedWindow( "ratio24", 1);
cvvShowImage( "ratio24", ResultImage24); /*the original map image*/

sprintf(str,"%s_skin_ratio24_%s.bmp",FirstName,col_str);
SaveImageBMP(str,ResultImage24,NULL);

//mark the ears and eyes

int hist[300]; //for the continuous width of each object

double MeanHSV[4];
for (int m=0;m<3;m++)
MeanHSV[m]=0;

iplSet(TempImage,0);
```

```
for (k=0;k<iCount2;k++)
{
if (objs2[k].Pixels<0) continue;

int h=objs2[k].Lowright.y-objs2[k].Upleft.y;
int w=objs2[k].Lowright.x-objs2[k].Upleft.x;

for (i=0;i<h;i++)
hist[i]=0;

for (i=objs2[k].Upleft.y;i<=objs2[k].Lowright.y;i++)
{
int nOffset=i*RotatedImage->widthStep;
for (j=objs2[k].Upleft.x;j<=objs2[k].Lowright.x;j++)
{
//find the mean HSV
iplGetPixel(RotatedImage24,j,i,ch);

for (int m=0;m<3;m++)
MeanHSV[m]+=(unsigned char)ch[m];

if (RotatedImage->imageData[nOffset+j]!=0)
hist[i-objs2[k].Upleft.y]++;
}
}

for (int m=0;m<3;m++)
MeanHSV[m]=MeanHSV[m]/objs2[k].Pixels;

double sum=MeanHSV[0]+MeanHSV[1]+MeanHSV[2];

printf("Mean =%.1f %.1f %.1f sum=%.1f\n",MeanHSV[0],MeanHSV[1],MeanHSV[2],sum);


for (i=objs2[k].Upleft.y;i<=objs2[k].Lowright.y;i++)
{
int nOffset=i*RotatedImage->widthStep;
for (j=objs2[k].Upleft.x;j<=objs2[k].Lowright.x;j++)
{
//find the mean HSV
iplGetPixel(RotatedImage24,j,i,ch);

int sum1=ch[0]+ch[1]+ch[2];

if (sum1<sum*0.85)
TempImage->imageData[nOffset+j]=(unsigned char)255; //see the holes
}
}

int ear_h=0,ear_hist=0;
for (i=h/4;i<h*3/4;i++)
if (hist[i]>ear_hist)
{
ear_h=i+objs2[k].Upleft.y;
ear_hist=hist[i];
}
printf("Obj %d Center=(%d,%d) ear_h=%d h=%d w=%d\n",k,objs2[k].x,objs2[k].
y,ear_h,h,w);

CvPoint p1= {objs2[k].Upleft.x,ear_h};
CvPoint p2= {objs2[k].Lowright.x,ear_h};
cvLine ( RotatedImage24, p1, p2, CV_RGB(255,255,255), 2 );
}

sprintf(str,"%s_hole_%s.bmp",FirstName,col_str);
cvvNamedWindow( str, 1);
cvvShowImage( str,TempImage); /*the original map image*/
SaveImageBMP(str,TempImage,NULL);

cvvNamedWindow( "Rotate24 ratio draw", 1);
cvvShowImage( "Rotate24 ratio draw", RotatedImage24); /*the original map image*/

sprintf(str,"%s_skin_rotate24_ratio_draw_%s.bmp",FirstName,col_str);
SaveImageBMP(str,RotatedImage24,NULL);


cvReleaseImage(&OriginalImage8);
cvReleaseImage(&OriginalImage9);
cvReleaseImage(&OriginalImage10);
cvReleaseImage(&TempImage);
cvReleaseImage(&TempImage24);
cvReleaseImage(&RotatedImage);
cvReleaseImage(&RotatedImage24);
cvReleaseImage(&OriginalImage24);

return 0;
}
int FindRGBSkinRegions(IplImage* RGBImage,IplImage* RefImage,IplImage* ResultImage)
{
unsigned char chRGB[4];
unsigned char chRef[4];

iplSet(ResultImage,0);
for ( int i=0;i<ResultImage->height;i++)
{
for (int j=0;j<ResultImage->width;j++)
{
iplGetPixel(RefImage,j,i,chRef);
iplGetPixel(RGBImage,j,i,chRGB);

if (chRef[0]!=0 || chRef[1]!=0 ||chRef[2]!=0)
iplPutPixel(ResultImage,j,i,chRGB);
}
}

return 0;
}
int main()
{
int iResult;

int i=0;

IplImage* OriginalImage;
IplImage* OriginalImage2; //converted image
IplImage* OriginalImage8;
IplImage* OriginalImage24;


OriginalImage= cvvLoadImage(cOriginalImageName);

cvvNamedWindow( "RGB Image", 1);
cvvShowImage( "RGB Image", OriginalImage); /*the original map image*/
```

```
OriginalImage2=cvCreateImage(  cvSize(OriginalImage->width,
OriginalImage->height),IPL_DEPTH_8U, 3);

OriginalImage8=cvCreateImage(  cvSize(OriginalImage->width,
OriginalImage->height),IPL_DEPTH_8U, 1);

OriginalImage24=cvCreateImage(  cvSize(OriginalImage->width,
OriginalImage->height),IPL_DEPTH_8U, 3);

GetSkinMaskYCrCb(OriginalImage,OriginalImage2,2,3);
cvvNamedWindow( "YCbCr Skin", 2);
cvvShowImage( "YCbCr Skin", OriginalImage2);
sprintf(str,"%s_YCbCr_skin.bmp",FirstName);
SaveImageBMP(str,OriginalImage2,NULL);

iplCopy(OriginalImage2,OriginalImage24);
FindRGBSkinRegions(OriginalImage,OriginalImage24,OriginalImage2);
cvvNamedWindow( "YCbCr Skin RGB", 2);
cvvShowImage( "YCbCr Skin RGB", OriginalImage2);
sprintf(str,"%s_YCbCr_skin_RGB.bmp",FirstName);
SaveImageBMP(str,OriginalImage2,NULL);

/*GetSkinMaskYCrCbNew(OriginalImage,OriginalImage2,2,6);
cvvNamedWindow( "YCbCr Skin2", 2);
cvvShowImage( "YCbCr Skin2", OriginalImage2);
sprintf(str,"%s_YCbCr_skin_new.bmp",FirstName);
SaveImageBMP(str,OriginalImage2,NULL);*/


    GetSkinMaskIHS(OriginalImage,OriginalImage2,3,4);
cvvNamedWindow( "IHS Skin", 2);
cvvShowImage( "IHS Skin", OriginalImage2);
sprintf(str,"%s_IHS_skin.bmp",FirstName);
SaveImageBMP(str,OriginalImage2,NULL);

iplCopy(OriginalImage2,OriginalImage24);
FindRGBSkinRegions(OriginalImage,OriginalImage24,OriginalImage2);
cvvNamedWindow( "HIS Skin RGB", 2);
cvvShowImage( "HIS Skin RGB", OriginalImage2);
sprintf(str,"%s_HIS_skin_RGB.bmp",FirstName);
SaveImageBMP(str,OriginalImage2,NULL);


//iplRGB2HSV(OriginalImage,OriginalImage2);

GetSkinMaskHSV(OriginalImage,OriginalImage2,2,4);
cvvNamedWindow( "HSV Skin", 2);
cvvShowImage( "HSV Skin", OriginalImage2);
sprintf(str,"%s_HSV_skin.bmp",FirstName);
SaveImageBMP(str,OriginalImage2,NULL);

//FindShapeResults(OriginalImage2,OriginalImage8,OriginalImage24,"HSV");
iplCopy(OriginalImage2,OriginalImage24);

//find the RGB face candidates
FindRGBSkinRegions(OriginalImage,OriginalImage24,OriginalImage2);
cvvNamedWindow( "HSV Skin RGB", 2);
cvvShowImage( "HSV Skin RGB", OriginalImage2);
sprintf(str,"%s_HSV_skin_RGB.bmp",FirstName);
SaveImageBMP(str,OriginalImage2,NULL);
```

```
//iplRGB2HLS(OriginalImage,OriginalImage2);

/*GetSkinMaskHLS(OriginalImage,OriginalImage2,3,5);
cvvNamedWindow( "HLS Skin", 2);
cvvShowImage( "HLS Skin", OriginalImage2);
sprintf(str,"%s_HLS_skin.bmp",FirstName);
SaveImageBMP(str,OriginalImage2,NULL);

*/
GetSkinMaskYUV(OriginalImage,OriginalImage2,2,4);
cvvNamedWindow( "YUV Skin", 2);
cvvShowImage( "YUV Skin", OriginalImage2);
sprintf(str,"%s_YUV_skin.bmp",FirstName);
SaveImageBMP(str,OriginalImage2,NULL);

iplCopy(OriginalImage2,OriginalImage24);

//FindShapeResults(OriginalImage2,OriginalImage8,OriginalImage24,"YUV");

//find the RGB face candidates
FindRGBSkinRegions(OriginalImage,OriginalImage24,OriginalImage2);
cvvNamedWindow( "YUV Skin RGB", 2);
cvvShowImage( "YUV Skin RGB", OriginalImage2);
sprintf(str,"%s_YUV_skin_RGB.bmp",FirstName);
SaveImageBMP(str,OriginalImage2,NULL);

  cvvWaitKey(0);

cvReleaseImage(&OriginalImage);
cvReleaseImage(&OriginalImage2);
cvReleaseImage(&OriginalImage8);
cvReleaseImage(&OriginalImage24);

return iResult;

}
```

Gaussian Mixture Model

```
function [Weights, Mu, Variances] = GMM(Input, No_of_Clusters,Limit)

% Initialize_the_Cluster_Centroid
[IDX, Initial_Centroids] = kmeans(Input',No_of_Clusters);

Mu = Initial_Centroids';
Limit = 10;
for Iterations = 1:Limit
[No_of_Features_within_Data,No_of_Data_Points] = size(Input);
Probability_of_Cluster_given_Point(1:No_of_Clusters,1:No_of_Data_Points) = 0.0;
[PC,Weights] = Cluster_Probability(Input,Mu);

%Initialize Cluster Covariances
COVAR(1:No_of_Features_within_Data,1:No_of_Clusters) = 0.0;
for i=1:No_of_Clusters
  COVAR(:,i) = Cluster_Covariance(Input(:,IDX==i));
end

%Initialize the probability matrix P(Cluster/Point)
Variances = COVAR;
for i=1:No_of_Clusters
for j=1:No_of_Data_Points
Probability_of_Cluster_given_Point(i,j) = Probability_of_Cluster_given_X(Input(:,j),
Mu,Variances,PC,i);
end;
end;

% New Means
Mu1(1:No_of_Clusters,1:No_of_Features_within_Data) = 0.0;
for i=1:No_of_Clusters
Mu1(i,:) = Compute_Mean_for_Cluster(Input,Mu,Variances,PC,i);
end;
%disp(Iterations);
%disp(Mu1);
Mu = Mu1';
end;

%Co-variance Computation for a Cluster
function [COVAR] = Cluster_Covariance(Data)
[r,c] = size(Data);
for i=1:r
    COVAR(i) = var(Data(i,:));
end;
end;

% Function to compute the Cluster Probablity
function [PC,INDEX] = Cluster_Probability(Data,Mu)
[No_of_Features_within_Data,No_of_Data_Points] = size(Data);
[No_of_Features_within_Mu,No_of_Mu_Points] = size(Mu);
PC(1:No_of_Mu_Points) = 0;
INDEX(1:No_of_Data_Points) = 0;
Distance(1:No_of_Data_Points,1:No_of_Mu_Points) = 0.0;

for i=1:No_of_Data_Points
    for j = 1:No_of_Mu_Points
    Distance(i,j) = sqrt(dot(Data(:,i)-Mu(:,j),Data(:,i)-Mu(:,j)));
    end
end

for i=1:No_of_Data_Points
    [value,idx] = min(Distance(i,:));
    PC(idx) = PC(idx)+1;
    INDEX(i) = idx;
end

PC = PC/No_of_Data_Points;

function mu = Compute_Mean_for_Cluster(X,Means,Variances,PC,Label_of_Cluster)
[r,c] = size(X);
mu = 0.0;
Numerator = 0.0;
Denominator = 0.0;
for i=1:c
    Numerator = Numerator+ Probability_of_Cluster_given_X(X(:,i),Means,
    Variances,PC,Label_of_Cluster)*X(:,i);
    Denominator = Denominator + Probability_of_Cluster_given_X(X(:,i),
    Means,Variances,PC,Label_of_Cluster);
end;
mu = Numerator/Denominator;

function MC = Mixing_Coefficient(X,MU,SIGMA)
M = normpdf(X,MU,SIGMA);
[r,c] = size(M);
```

```
MC = 0.0;
for i=1:c
    MC = MC + M(i);
end
MC = MC/c;


function PY = Probability_of_Cluster_given_X(X,Means,Variances,PC,
Label_of_Cluster)
PY = PC(Label_of_Cluster)*Mixing_Coefficient(X,Means(:,Label_of_Cluster),
Variances(:,Label_of_Cluster));
PY = PY/Probability_of_X(X,Means,Variances,PC);



function [res]=gather_point(xx,yy,zz)
[m,n]=size(xx);
res=ones(n,3);
for i=1:1:n
    for i1=1:1:3
        if i1==1
            res(i,i1)=xx(i);
        elseif i1==2
            res(i,i1)=yy(i);
        else
            res(i,i1)=zz(i);
        end
    end
end


clear all; close all; clc;
i=imread('E:\MATLAB6p5\work\ÖśÏßÔÚÆäÖÐ.bmp');
[row,range]=size(i);
p=row*range;a=0;Y=zeros(1,p);Q=zeros(1,p);
for k1=1:1:row
    for k2=1:1:range
        if i(k1,k2)==0
            a=a+1;
            XXX=k2-1;YYY=row+1-k1;
            Y(a)=YYY;Q(a)=YYY-XXX;
        end
    end
end
% for t=1:1:a
%     o=[Y(t) Q(t)]
% end
X1=[0 1];Y1=[Y(1) Q(1)];
plot(X1,Y1,'r');hold on;
YY=zeros(1,a-1);QQ=zeros(1,a-1);
for t=2:1:a
    X2=[0 1];Y2=[Y(t) Q(t)];
    plot(X2,Y2);
    [YY(t-1),QQ(t-1)]=pll(X1,Y1,X2,Y2);
%     yyy=YY(t-1)
%     qqq=QQ(t-1)%£ť¡żţã
    plot(YY(t-1),QQ(t-1),'ko');
end



% [res]=gather_point(YY,QQ,QQ);nn=0;res
% [m,n3]=size(YY);
% for no1=1:1:n3
%     n=0;
%     for no2=(no1+1):1:n3
%         if (abs(res(no2,1)-res(no1,1))<=0.0001&abs(res(no2,2)-res(no1,2))
<=0.0001&abs(res(no2,2)-res(no1,2))<=0.0001)
%            n=n+1;hao=res(no1,:);
%         end
%     end
%     if nn>=n3/5
%         rhao=hao;break;
%     elseif nn<n
%         rhao=hao;nn=n;
%     end
% end
[rhao]=the_max1(YY,QQ,QQ);


% yyy=YY
% qqq=QQ
% t2=0;t3=0;
% for t1=2:1:t
%     t2=YY(t1)+t2;
%     t3=t3+QQ(t1);
% end
%     t2=t2/(t1-1)
%     t3=t3/(t1-1)%t2=m,t3=b
%     i1=ones(row,range);
%    t2=2;t3=0;

i1=ones(row,range);
rhao
t2=rhao(1);t3=rhao(2);
t2=round(t2);t3=round(t3);
% t2=1.5;t3=1.5;
    for k1=1:1:row
        for k2=1:1:range
            XXX=k2-1;YYY=row+1-k1;
            if abs(YYY-(t2*XXX+t3))<=1%YYY==t2*XXX+t3
                i1(k1,k2)=0;
            end
        end
    end
    imwrite(i1,'E:\MATLAB6p5\work\ÕÒşöţÄÖśÏSS.bmp','bmp');
    i2=imread('E:\MATLAB6p5\work\ÕÒşöţÄÖśÏSS.bmp');
    figure;
    subplot(2,1,1);
    imshow(i);
    subplot(2,1,2);
    imshow(i2);


RGB=imread('001.gif');
YCbCr=rgb2ycbcr(RGB);
Y=YCbCr(:,:,1);
Cb=YCbCr(:,:,2);
Cr=YCbCr(:,:,3);
imshow(RGB);title('RGB');
figure,imshow(YCbCr);title('YCbCr');
I=RGB;
rows=size(YCbCr,1);
columns=size(YCbCr,2);
```

```
k=(2.53/180)*pi;
m=sin(k);n=cos(k);
cx=109.38;cy=152.02;ecx=1.60;ecy=2.41;a=25.39;b=14.03;
for i=1:rows
    for j=1:columns
         if Y(i,j)<80
           I(i,j,:)=0;
       elseif (Y(i,j)<=230&&Y(i,j)>=80)
         x=(double(Cb(i,j))-cx)*n+(double(Cr(i,j))-cy)*m;
         y=(double(Cr(i,j))-cy)*n-(double(Cb(i,j))-cx)*m;
          if((x-ecx)^2/a^2+(y-ecy)^2/b^2)<=1
             I(i,j,:)=255;
         else I(i,j,:)=0;
         end
     elseif Y(i,j)>230
         x=(double(Cb(i,j))-cx)*n+(double(Cr(i,j))-cy)*m;
         y=(double(Cr(i,j))-cy)*n-(double(Cb(i,j))-cx)*m;
         if ((x-ecx)^2/(1.1*a)^2+(y-ecy)^2/(1.1*b)^2)<=1
             I(i,j,:)=255;
         else  I(i,j,:)=0;
         end
     end
 end
end
figure,imshow(I);


function [W,M,V,L] = EM_GM_fast(X,k,ltol,maxiter,pflag,Init)

%%%% Validate inputs %%%%
if nargin <= 1,
    disp('EM_GM must have at least 2 inputs: X,k!/n')
    return
elseif nargin == 2,
    ltol = 0.1; maxiter = 1000; pflag = 0; Init = [];
    err_X = Verify_X(X);
    err_k = Verify_k(k);
    if err_X | err_k, return; end
elseif nargin == 3,
    maxiter = 1000; pflag = 0; Init = [];
    err_X = Verify_X(X);
    err_k = Verify_k(k);
    [ltol,err_ltol] = Verify_ltol(ltol);
    if err_X | err_k | err_ltol, return; end
elseif nargin == 4,
    pflag = 0;  Init = [];
    err_X = Verify_X(X);
    err_k = Verify_k(k);
    [ltol,err_ltol] = Verify_ltol(ltol);
    [maxiter,err_maxiter] = Verify_maxiter(maxiter);
    if err_X | err_k | err_ltol | err_maxiter, return; end
elseif nargin == 5,
     Init = [];
    err_X = Verify_X(X);
    err_k = Verify_k(k);
    [ltol,err_ltol] = Verify_ltol(ltol);
    [maxiter,err_maxiter] = Verify_maxiter(maxiter);
    [pflag,err_pflag] = Verify_pflag(pflag);
    if err_X | err_k | err_ltol | err_maxiter | err_pflag, return; end
elseif nargin == 6,
    err_X = Verify_X(X);
    err_k = Verify_k(k);
    [ltol,err_ltol] = Verify_ltol(ltol);
    [maxiter,err_maxiter] = Verify_maxiter(maxiter);
    [pflag,err_pflag] = Verify_pflag(pflag);
    [Init,err_Init]=Verify_Init(Init);
    if err_X | err_k | err_ltol | err_maxiter | err_pflag | err_Init, return; end
else
    disp('EM_GM must have 2 to 6 inputs!');
    return
end

%%%% Initialize W, M, V,L %%%%
t = cputime;
if isempty(Init),
    [W,M,V] = Init_EM(X,k); L = 0;
else
    W = Init.W;
    M = Init.M;
    V = Init.V;
end
Ln = Likelihood(X,k,W,M,V); % Initialize log likelihood
Lo = 2*Ln;

%%%% EM algorithm %%%%
niter = 0;
while (abs(100*(Ln-Lo)/Lo)>ltol) & (niter<=maxiter),
    E = Expectation(X,k,W,M,V);      % E-step
    [W,M,V] = Maximization(X,k,E);  % M-step
    Lo = Ln;
    Ln = Likelihood(X,k,W,M,V);
    niter = niter + 1;
end
L = Ln;

%%%% Plot 1D or 2D %%%%
if pflag==1,
    [n,d] = size(X);
    if d>2,
        disp('Can only plot 1 or 2 dimensional applications!/n');
    else
        Plot_GM(X,k,W,M,V);
    end
    elapsed_time = sprintf('CPU time used for EM_GM: %5.2fs',cputime-t);
    disp(elapsed_time);
    disp(sprintf('Number of iterations: %d',niter-1));
end
%%%%%%%%%%%%%%%%%%%%%%
%%%% End of EM_GM %%%%
%%%%%%%%%%%%%%%%%%%%%%

function E = Expectation(X,k,W,M,V)
% This function is the modification of 'Expectation' in EM_GM made by
% Mr. Michael Boedigheimer to enchance computational speed.
% Note: this modification requires more memory to execute.
%       If EM_GM_fast does not provide any speed gain or is slower than EM_GM,
%       more memory is needed or EM_GM should be used instead.
[n,d] = size(X);
E = zeros(n,k);
for j = 1:k,
    if V(:,:,j)==zeros(d,d), V(:,:,j)=ones(d,d)*eps; end
    E(:,j) = W(j).*mvnpdf( X, M(:,j)', V(:,:,j));
end
total = repmat(sum(E,2),1,j);
```

```
E = E./total;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Expectation %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [W,M,V] = Maximization(X,k,E)
% This function is the modification of 'Maximization' in EM_GM made by
% Mr. Michael Boedigheimer to enchance computational speed.
% Note: this modification requires more memory to execute.
%       If EM_GM_fast does not provide any speed gain or is slower than EM_GM,
%       more memory is needed or EM_GM should be used instead.
[n,d] = size(X);
W = sum(E);
M = X'*E./repmat(W,d,1);
for i=1:k,
    dXM = X - repmat(M(:,i)',n,1);
    Wsp = spdiags(E(:,i),0,n,n);
    V(:,:,i) = dXM'*Wsp*dXM/W(i);
end
W = W/n;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Maximization %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function L = Likelihood(X,k,W,M,V)
% Compute L based on K. V. Mardia, "Multivariate Analysis", Academic
% Press, 1979, PP. 96-97
% to enchance computational speed
[n,d] = size(X);
U = mean(X)';
S = cov(X);
L = 0;
for i=1:k,
    iV = inv(V(:,:,i));
    L = L + W(i)*(-0.5*n*log(det(2*pi*V(:,:,i))) ...
        -0.5*(n-1)*(trace(iV*S)+(U-M(:,i))'*iV*(U-M(:,i))));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Likelihood %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

function err_X = Verify_X(X)
err_X = 1;
[n,d] = size(X);
if n<d,
    disp('Input data must be n x d!/n');
    return
end
err_X = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Verify_X %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%

function err_k = Verify_k(k)
err_k = 1;
if ~isnumeric(k) | ~isreal(k) | k<1,
    disp('k must be a real integer >= 1!/n');
    return
end
err_k = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Verify_k %%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%

function [ltol,err_ltol] = Verify_ltol(ltol)
err_ltol = 1;
if isempty(ltol),
    ltol = 0.1;
elseif ~isreal(ltol) | ltol<=0,
    disp('ltol must be a positive real number!');
    return
end
err_ltol = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Verify_ltol %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [maxiter,err_maxiter] = Verify_maxiter(maxiter)
err_maxiter = 1;
if isempty(maxiter),
    maxiter = 1000;
elseif ~isreal(maxiter) | maxiter<=0,
    disp('ltol must be a positive real number!');
    return
end
err_maxiter = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Verify_maxiter %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [pflag,err_pflag] = Verify_pflag(pflag)
err_pflag = 1;
if isempty(pflag),
    pflag = 0;
elseif pflag~=0 & pflag~=1,
    disp('Plot flag must be either 0 or 1!/n');
    return
end
err_pflag = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Verify_pflag %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Init,err_Init] = Verify_Init(Init)
err_Init = 1;
if isempty(Init),
    % Do nothing;
elseif isstruct(Init),
    [Wd,Wk] = size(Init.W);
    [Md,Mk] = size(Init.M);
    [Vd1,Vd2,Vk] = size(Init.V);
    if Wk~=Mk | Wk~=Vk | Mk~=Vk,
        disp('k in Init.W(1,k), Init.M(d,k) and Init.V(d,d,k) must equal!/n')
        return
    end
    if Md~=Vd1 | Md~=Vd2 | Vd1~=Vd2,
        disp('d in Init.W(1,k), Init.M(d,k) and Init.V(d,d,k) must equal!/n')
        return
    end
else
    disp('Init must be a structure: W(1,k), M(d,k), V(d,d,k) or []!');
    return
end
err_Init = 0;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Verify_Init %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [W,M,V] = Init_EM(X,k)
[n,d] = size(X);
[Ci,C] = kmeans(X,k,'Start','cluster', ...
    'Maxiter',100, ...
    'EmptyAction','drop', ...
    'Display','off'); % Ci(nx1) - cluster indeices; C(k,d) - cluster centroid
    (i.e. mean)
while sum(isnan(C))>0,
    [Ci,C] = kmeans(X,k,'Start','cluster', ...
        'Maxiter',100, ...
        'EmptyAction','drop', ...
        'Display','off');
end
M = C';
Vp = repmat(struct('count',0,'X',zeros(n,d)),1,k);
for i=1:n, % Separate cluster points
    Vp(Ci(i)).count = Vp(Ci(i)).count + 1;
    Vp(Ci(i)).X(Vp(Ci(i)).count,:) = X(i,:);
end
V = zeros(d,d,k);
for i=1:k,
    W(i) = Vp(i).count/n;
    V(:,:,i) = cov(Vp(i).X(1:Vp(i).count,:));
end
%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Init_EM %%%%
%%%%%%%%%%%%%%%%%%%%%%%%

function Plot_GM(X,k,W,M,V)
[n,d] = size(X);
if d>2,
    disp('Can only plot 1 or 2 dimensional applications!/n');
    return
end
S = zeros(d,k);
R1 = zeros(d,k);
R2 = zeros(d,k);
for i=1:k,  % Determine plot range as 4 x standard deviations
    S(:,i) = sqrt(diag(V(:,:,i)));
    R1(:,i) = M(:,i)-4*S(:,i);
    R2(:,i) = M(:,i)+4*S(:,i);
end
Rmin = min(min(R1));
Rmax = max(max(R2));
R = [Rmin:0.001*(Rmax-Rmin):Rmax];
clf, hold on
if d==1,
    Q = zeros(size(R));
    for i=1:k,
        P = W(i)*normpdf(R,M(:,i),sqrt(V(:,:,i)));
        Q = Q + P;
        plot(R,P,'r-'); grid on,
    end
    plot(R,Q,'k-');
    xlabel('X');
    ylabel('Probability density');
else % d==2
    plot(X(:,1),X(:,2),'r.');
    for i=1:k,
        Plot_Std_Ellipse(M(:,i),V(:,:,i));
    end
    xlabel('1^{st} dimension');
    ylabel('2^{nd} dimension');
    axis([Rmin Rmax Rmin Rmax])
end
title('Gaussian Mixture estimated by EM');
%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Plot_GM %%%%
%%%%%%%%%%%%%%%%%%%%%%%%

function Plot_Std_Ellipse(M,V)
[Ev,D] = eig(V);
d = length(M);
if V(:,:)==zeros(d,d),
    V(:,:) = ones(d,d)*eps;
end
iV = inv(V);
% Find the larger projection
P = [1,0;0,0];  % X-axis projection operator
P1 = P * 2*sqrt(D(1,1)) * Ev(:,1);
P2 = P * 2*sqrt(D(2,2)) * Ev(:,2);
if abs(P1(1)) >= abs(P2(1)),
    Plen = P1(1);
else
    Plen = P2(1);
end
count = 1;
step = 0.001*Plen;
Contour1 = zeros(2001,2);
Contour2 = zeros(2001,2);
for x = -Plen:step:Plen,
    a = iV(2,2);
    b = x * (iV(1,2)+iV(2,1));
    c = (x^2) * iV(1,1) - 1;
    Root1 = (-b + sqrt(b^2 - 4*a*c))/(2*a);
    Root2 = (-b - sqrt(b^2 - 4*a*c))/(2*a);
    if isreal(Root1),
        Contour1(count,:) = [x,Root1] + M';
        Contour2(count,:) = [x,Root2] + M';
        count = count + 1;
    end
end
Contour1 = Contour1(1:count-1,:);
Contour2 = [Contour1(1,:);Contour2(1:count-1,:);Contour1(count-1,:)];
plot(M(1),M(2),'k+');
plot(Contour1(:,1),Contour1(:,2),'k-');
plot(Contour2(:,1),Contour2(:,2),'k-');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% End of Plot_Std_Ellipse %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

ANN and SVM

```
clear
format long e;
load I;
% I = radbas(I);

% make the original data between 0 and 1
I1 = I';
n = 1;
```

```
while n <= 39
M1(1,n) = min(I1(:,n));
n = n+1;
end

for n = 1:39
    if M1(1,n) < 0
        I1(:,n) = I1(:,n) + (-M1(1,n));
        n = n+1;
    end
end

n = 1;
while n <= 39
    M2(1,n) = max(I1(:,n));
    I1(:,n) = I1(:,n)/M2(1,n);
    n = n+1;
end
I = I1'


% Mine

I2(1,:) = I(15,:);
I2(2,:) = I(2,:);
I2(3,:) = I(39,:);
I2(4,:) = I(16,:);
I2(5,:) = I(17,:);
I2(6,:) = I(33,:);
I2(7,:) = I(18,:);
I2(8,:) = I(34,:);
I2(9,:) = I(19,:);
I2(10,:) = I(29,:);
I2(11,:) = I(7,:);
I2(12,:) = I(31,:);
I2(13,:) = I(21,:);
I2(14,:) = I(23,:);
I2(15,:) = I(5,:);
I2(16,:) = I(20,:);
I2(17,:) = I(27,:);
I2(18,:) = I(1,:);
I2(19,:) = I(25,:);
I2(20,:) = I(30,:);
I2(21,:) = I(22,:);
I2(22,:) = I(12,:);
I2(23,:) = I(14,:);
I2(24,:) = I(28,:);
I2(25,:) = I(38,:);
% I2(26,:) = I(35,:);
% I2(27,:) = I(8,:);
% I2(28,:) = I(26,:);
% I2(29,:) = I(24,:);
% I2(30,:) = I(3,:);
% I2(31,:) = I(13,:);
% I2(32,:) = I(37,:);
% I2(33,:) = I(4,:);
% I2(34,:) = I(32,:);
% I2(35,:) = I(36,:);
% I2(36,:) = I(11,:);
% I2(37,:) = I(10,:);
% I2(38,:) = I(9,:);
% I2(39,:) = I(6,:);
% search the target 1 and 0

[r,w] = find(T == 1);
[r1,w1] = find(T == 0);
[r2,w2] = size(w);
[r3,w3] = size(w1);

n = 1;          % Target is 1
while n <= w2
    Input1(:,n) = I2(:,w(n));
    Target1(:,n) = T(:,w(n));
    n = n+1;
end

n = 1           % Target is 0
while n<= w3
    Input2(:,n) = I2(:,w1(n));
    Target2(:,n) = T(:,w1(n));
    n = n+1;
end

%  Divided data
[trainV1,valV1,testV1] = dividevec(Input1,Target1,0,0.2);  % target is 1
[trainV2,valV2,testV2] = dividevec(Input2,Target2,0,0.2);  % target is 0

a1 = trainV1.P;
a2 = trainV2.P;
a3 = trainV1.indices;
a4 = trainV2.indices;
[ra1,wa1] = size(a1);
[ra2,wa2] = size(a2);
[RA1,WA1] = size(Input1);
[RA2,WA2] = size(Input2);
W1 = wa1/WA1;
W2 = wa2/WA2;
A = [a1 a2];   % A = trainV.P

b1 = trainV1.T;
b2 = trainV2.T;
[rb1,wb1] = size(b1);
[rb2,wb2] = size(b2);
[RB1,WB1] = size(Input1);
[RB2,WB2] = size(Input2);
W3 = wb1/WB1;
W4 = wb2/WB2;
B = [b1 b2];   % B = trainV.T

% Build the structure of testV
f1 = testV1.P;  % Target is 1
f2 = testV2.P;  % Target is 0
g1 = testV1.T;
g2 = testV2.T;
h1 = testV1.indices;
h2 = testV2.indices;
F = [f1 f2];
G = [g1 g2];
H = [h1 h2];
testV = struct('P',{F},'T',{G},'indices',{H});

b3 = zeros(1,104);
b4 = ones(1,570);
```

```
B1 = [b3 b4];
net1 = newff(minmax(A),[30 16 1],{'logsig' 'logsig' 'purelin'},'trainlm');
% net1 = newff(minmax(A),[30 1],{'logsig' 'purelin'},'trainlm');
% net1 = newff(minmax(I2),[25 1],{'radbas' 'logsig'},'trainlm');
% net1 = newff(minmax(I2),[25 1],{'logsig' 'satlin'},'trainlm');
% net1.IW{1,1} = net1.IW{1,1}*0.5;
% net1.LW{2,1} = net1.LW{2,1}*0.01;
% net1.inputConnect(3,1) = 0;
net1.inputConnect(2,1) = 1;
net1.biasConnect(1) = 1;
% net1.biasConnect(3) = 0;
net1.trainParam.goal = 1e-10;
net1.trainParam.epochs = 100;
net1.trainParam.mu_max = 1e400;
net1.trainParam.min_grad = 1e-15;
net1.trainParam.lr = 0.5;
[net1,tr1,Y1,E1,Pf1,Af1] = train(net1,A,B);

% net2 = newff(minmax(Y1),[30 1],{'logsig' 'purelin'},'trainlm');
% net2.trainParam.goal = 1e-2;
% net2.trainParam.epochs = 1000;
% net2.trainParam.min_grad = 1e-20;
% [net2,tr2,Y2,E2,Pf2,Af2] = train(net2,Y1,B);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CHECKING THE RESULT OF TRAINING DATA ARE 90% %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% y1 = Y1(:,1:104);   % Target is 1, result of training data
% y2 = Y1(:,105:674); % Target is 0, reuslt of training data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CHECKING THE RESULT OF TRAINING DATA ARE 80% %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% y1 = Y1(:,1:92);   % Target is 1, result of training data
% y2 = Y1(:,93:599); % Target is 0, reuslt of training data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CHECKING THE RESULT OF TRAINING DATA ARE 70% %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% y1 = Y1(:,1:81);   % Target is 1, result of training data
% y2 = Y1(:,82:525); % Target is 0, reuslt of training data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CHECKING THE RESULT OF TRAINING DATA ARE 60% %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% y1 = Y1(:,1:69);   % Target is 1, result of training data
% y2 = Y1(:,70:449); % Target is 0, result of training data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CHECKING THE RESULT OF TRAINING DATA ARE 50% %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% y1 = Y1(:,1:57);   % Target is 1, result of training data
% y2 = Y1(:,58:373); % Target is 0, result of training data

% s1 = min(y1);
% s2 = max(y1);
% s3 = min(y2);
% s4 = max(y2);

save net1


clear
format long e
load I

% make the original data between 0 and 1
I1 = I'
n = 1;
while n <= 39
M1(1,n) = min(I1(:,n));
n = n+1;
end

for n = 1:39
    if M1(1,n) < 0
        I1(:,n) = I1(:,n) + (-M1(1,n));
        n = n+1;
    end
end

n = 1;
while n <= 39
    M2(1,n) = max(I1(:,n));
    I1(:,n) = I1(:,n)/M2(1,n);
    n = n+1;
end
I = I1'

% Extract features
I2(1,:) = I(15,:);
I2(2,:) = I(2,:);
I2(3,:) = I(39,:);
I2(4,:) = I(16,:);
I2(5,:) = I(17,:);
I2(6,:) = I(33,:);
I2(7,:) = I(18,:);
I2(8,:) = I(34,:);
I2(9,:) = I(19,:);
I2(10,:) = I(29,:);
I2(11,:) = I(7,:);
I2(12,:) = I(31,:);
I2(13,:) = I(21,:);
I2(14,:) = I(23,:);
I2(15,:) = I(5,:);
I2(16,:) = I(20,:);
I2(17,:) = I(27,:);
I2(18,:) = I(1,:);
I2(19,:) = I(25,:);
I2(20,:) = I(30,:);
I2(21,:) = I(22,:);
I2(22,:) = I(12,:);
I2(23,:) = I(14,:);
I2(24,:) = I(28,:);
I2(25,:) = I(38,:);
% I2(26,:) = I(35,:);
% I2(27,:) = I(8,:);
% I2(28,:) = I(26,:);
% I2(29,:) = I(24,:);
% I2(30,:) = I(3,:);
% I2(31,:) = I(13,:);
% I2(32,:) = I(37,:);
% I2(33,:) = I(4,:);
% I2(34,:) = I(32,:);
% I2(35,:) = I(36,:);
% I2(36,:) = I(11,:);
% I2(37,:) = I(10,:);
% I2(38,:) = I(9,:);
% I2(39,:) = I(6,:);


% find out how many Target is 1 and Target is 0
[r,w] = find(T == 1);
[r1,w1] = find(T == 0);
[r2,w2] = size(w);
[r3,w3] = size(w1);


n = 1;         % Target is 1
while n <= w2
    Input1(:,n) = I2(:,w(n));
    Target1(:,n) = T(:,w(n));
    n = n+1;
end

n = 1          % Target is 0
while n<= w3
    Input0(:,n) = I2(:,w1(n));
    Target0(:,n) = T(:,w1(n));
    n = n+1;
end

Input = [Input1 Input0];
Input = Input';

Target = [Target1 Target0];
Target = Target';



%%%%%%%
% SVM %
%%%%%%%

[train, test] = crossvalind('holdOut',Target,0.2); % 0.1 is test data
cp = classperf(Target);
options = optimset('MaxIter',10000);
svmstruct = svmtrain(Input(train,:),Target(train),'kernel_function','rbf');
classes = svmclassify(svmstruct,Input(test,:));
classperf(cp,classes,test);

IT = Input(test,:);
It = Input(train,:);
TT = Target(test,:);
[r4,w4] = find(TT == 1);

[r5,w5] = size(r4);

n = 1;
while n <= r5
    IT1(n,:) = IT(r4(n),:);
    n = n+1;
end


save IT1
```

Compare ANN and SVM

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>


typedef int           BOOL;
typedef int           INT;
typedef double        REAL;

#define FALSE         0
#define TRUE          1
#define NOT           !
#define AND           &&
#define OR            ||

#define MIN_REAL      -HUGE_VAL
#define MAX_REAL      +HUGE_VAL
#define MIN(x,y)      ((x)<(y) ? (x) : (y))
#define MAX(x,y)      ((x)>(y) ? (x) : (y))

#define LO            0.1
#define HI            0.9
#define BIAS          1

#define sqr(x)        ((x)*(x))


typedef struct {                     /* A LAYER OF A NET:                     */
        INT           Units;         /* - number of units in this layer       */
        REAL*         Output;        /* - output of ith unit                  */
        REAL*         Error;         /* - error term of ith unit              */
        REAL**        Weight;        /* - connection weights to ith unit      */
        REAL**        WeightSave;    /* - saved weights for stopped training  */
        REAL**        dWeight;       /* - last weight deltas for momentum     */
} LAYER;

typedef struct {                     /* A NET:                                */
        LAYER**       Layer;         /* - layers of this net                  */
        LAYER*        InputLayer;    /* - input layer                         */
        LAYER*        OutputLayer;   /* - output layer                        */
        REAL          Alpha;         /* - momentum factor                     */
        REAL          Eta;           /* - learning rate                       */
        REAL          Gain;          /* - gain of sigmoid function            */
        REAL          Error;         /* - total net error                     */
} NET;


/******************************************************************************
        R A N D O M S   D R A W N   F R O M   D I S T R I B U T I O N S
```

```
 ******************************************************************************/

INT  iRandomStart;
REAL  ThOptimal = 0.5;

void InitializeRandoms()
{
time_t time1;

time(&time1);

iRandomStart = time1%4712;

iRandomStart = 4237; //4133; 4711; 4237 1082 3721

srand(iRandomStart);


}


INT RandomEqualINT(INT Low, INT High)
{
  return rand() % (High-Low+1) + Low;
}


REAL RandomEqualREAL(REAL Low, REAL High)
{
  return ((REAL) rand() / RAND_MAX) * (High-Low) + Low;
}


/******************************************************************************
               A P P L I C A T I O N - S P E C I F I C   C O D E
 ******************************************************************************/

#define UsingOriginalSamples 1

#define BALANCED_SAMPLES 1265
#define UNBALANCED_SAMPLES 748

#define NUM_SAMPLES   ((UsingOriginalSamples == 0)? BALANCED_SAMPLES:
UNBALANCED_SAMPLES)

#define TrainingRate 0.8

#define FEATURE_DIM   23

int FeatureList[FEATURE_DIM] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,
18,19,20,25,26,39};

REAL InputFeatures[NUM_SAMPLES][FEATURE_DIM];

REAL LabelResult[NUM_SAMPLES]; // 0 or 1 to indicate negative or positive samples
REAL OutputResults[NUM_SAMPLES]; //0-1 as predictable output
unsigned char Duplicated[NUM_SAMPLES]; // 0 or 1 to indicate if the sample
                is duplicated or not

unsigned char TrainingLabel[NUM_SAMPLES]; //0 or 1 to indicate if the
sample is used for training or not
```

```
#define NUM_LAYERS    3
#define N             FEATURE_DIM //input layer
#define H   15
#define M             1  //output layer
INT                   Units[NUM_LAYERS] = {N, H, M}; //intermediate
layer with 10 nodes, 15 nodes seems the best


REAL                  Mean;
REAL                  TrainError;
REAL                  TrainErrorPredictingMean=0;
REAL                  TestError;
REAL                  TestErrorPredictingMean=0;

FILE*                 f;
INT   iIndexTraining;

void NormalizeFeature(REAL *feature, int num)
{
  int i;
  REAL Min, Max;

  Min = MAX_REAL;
  Max = MIN_REAL;
  for (i=0; i<num; i++) {
    Min = MIN(Min, feature[i]);
    Max = MAX(Max, feature[i]);
  }

  Mean = 0;
  for (i=0; i<num; i++)
  {
    feature[i] = ((feature[i]-Min) / (Max-Min)) * (HI-LO) + LO;
    Mean += feature[i] / num;
  }
}


void InitializeApplication(NET* Net)
{
  int  i,j;
  REAL Out, Err;
  char str[100];

  Net->Alpha = 0.5;
  Net->Eta   = 0.05;
  Net->Gain  = 1;

  iIndexTraining = 0;


  //NormalizeSunspots();
  TrainErrorPredictingMean = 0;

  for (j=0; j<=NUM_SAMPLES; j++)
  {
  if (TrainingLabel[j]!=0)
  {
for (i=0; i<M; i++)
```

```
{
Out = LabelResult[j+i];
Err = Mean - Out;
TrainErrorPredictingMean += 0.5 * sqr(Err);
}
  }
  }


  TestErrorPredictingMean = 0;

  for (j=0; j<=NUM_SAMPLES; j++)
  {
  if (TrainingLabel[j]==0)
  {
    for (i=0; i<M; i++)
{
Out = LabelResult[j+i];
Err = Mean - Out;
TestErrorPredictingMean += 0.5 * sqr(Err);
}
  }
  }

  sprintf(str,"BPN_h%d_t%dc_b%d_r%dblc.txt",H,(int)(100*TrainingRate),
  1-UsingOriginalSamples,iRandomStart);

  f = fopen(str, "wt");
}


void FinalizeApplication(NET* Net)
{
  fclose(f);
}


/******************************************************************************
                          I N I T I A L I Z A T I O N
 ******************************************************************************/


void GenerateNetwork(NET* Net)
{
  INT l,i;

  Net->Layer = (LAYER**) calloc(NUM_LAYERS, sizeof(LAYER*));

  for (l=0; l<NUM_LAYERS; l++) {
    Net->Layer[l] = (LAYER*) malloc(sizeof(LAYER));

    Net->Layer[l]->Units      = Units[l];
    Net->Layer[l]->Output     = (REAL*)  calloc(Units[l]+1, sizeof(REAL));
    Net->Layer[l]->Error      = (REAL*)  calloc(Units[l]+1, sizeof(REAL));
    Net->Layer[l]->Weight     = (REAL**) calloc(Units[l]+1, sizeof(REAL*));
    Net->Layer[l]->WeightSave = (REAL**) calloc(Units[l]+1, sizeof(REAL*));
    Net->Layer[l]->dWeight    = (REAL**) calloc(Units[l]+1, sizeof(REAL*));
    Net->Layer[l]->Output[0]  = BIAS;

    if (l != 0) {
      for (i=1; i<=Units[l]; i++) {
Net->Layer[l]->Weight[i]     = (REAL*) calloc(Units[l-1]+1, sizeof(REAL));
Net->Layer[l]->WeightSave[i] = (REAL*) calloc(Units[l-1]+1, sizeof(REAL));
Net->Layer[l]->dWeight[i]    = (REAL*) calloc(Units[l-1]+1, sizeof(REAL));
      }
    }
  }
  Net->InputLayer  = Net->Layer[0];
  Net->OutputLayer = Net->Layer[NUM_LAYERS - 1];
  Net->Alpha       = 0.9;
  Net->Eta         = 0.25;
  Net->Gain        = 1;
}


void RandomWeights(NET* Net)
{
  INT l,i,j;

  for (l=1; l<NUM_LAYERS; l++) {
    for (i=1; i<=Net->Layer[l]->Units; i++) {
      for (j=0; j<=Net->Layer[l-1]->Units; j++) {
        Net->Layer[l]->Weight[i][j] = RandomEqualREAL(-0.5, 0.5);
      }
    }
  }
}


void SetInput(NET* Net, REAL* Input)
{
  INT i;

  for (i=1; i<=Net->InputLayer->Units; i++)
  {
    Net->InputLayer->Output[i] = Input[i-1];
  }
}


void GetOutput(NET* Net, REAL* Output)
{
  INT i;

  for (i=1; i<=Net->OutputLayer->Units; i++)
  {
    Output[i-1] = Net->OutputLayer->Output[i];
  }
}


/******************************************************************************
            S U P P O R T   F O R   S T O P P E D   T R A I N I N G
 ******************************************************************************/


void SaveWeights(NET* Net)
{
  INT l,i,j;

  for (l=1; l<NUM_LAYERS; l++)
  {
    for (i=1; i<=Net->Layer[l]->Units; i++)
{
      for (j=0; j<=Net->Layer[l-1]->Units; j++)
  {
        Net->Layer[l]->WeightSave[i][j] = Net->Layer[l]->Weight[i][j];
      }
    }
  }
}


void RestoreWeights(NET* Net)
{
  INT l,i,j;

  for (l=1; l<NUM_LAYERS; l++) {
    for (i=1; i<=Net->Layer[l]->Units; i++) {
      for (j=0; j<=Net->Layer[l-1]->Units; j++) {
        Net->Layer[l]->Weight[i][j] = Net->Layer[l]->WeightSave[i][j];
      }
    }
  }
}


/******************************************************************************
                     P R O P A G A T I N G   S I G N A L S
 ******************************************************************************/


void PropagateLayer(NET* Net, LAYER* Lower, LAYER* Upper)
{
  INT  i,j;
  REAL Sum;

  for (i=1; i<=Upper->Units; i++) {
    Sum = 0;
    for (j=0; j<=Lower->Units; j++) {
      Sum += Upper->Weight[i][j] * Lower->Output[j];
    }
    Upper->Output[i] = 1 / (1 + exp(-Net->Gain * Sum));
  }
}


void PropagateNet(NET* Net)
{
  INT l;

  for (l=0; l<NUM_LAYERS-1; l++) {
    PropagateLayer(Net, Net->Layer[l], Net->Layer[l+1]);
  }
}


/******************************************************************************
                  B A C K P R O P A G A T I N G   E R R O R S
 ******************************************************************************/


void ComputeOutputError(NET* Net, REAL* Target)
{
  INT  i;
  REAL Out, Err;
  Net->Error = 0;
  for (i=1; i<=Net->OutputLayer->Units; i++) {
    Out = Net->OutputLayer->Output[i];
    Err = Target[i-1]-Out;
    Net->OutputLayer->Error[i] = Net->Gain * Out * (1-Out) * Err;
    Net->Error += 0.5 * sqr(Err);
  }
}


void BackpropagateLayer(NET* Net, LAYER* Upper, LAYER* Lower)
{
  INT  i,j;
  REAL Out, Err;

  for (i=1; i<=Lower->Units; i++) {
    Out = Lower->Output[i];
    Err = 0;
    for (j=1; j<=Upper->Units; j++) {
      Err += Upper->Weight[j][i] * Upper->Error[j];
    }
    Lower->Error[i] = Net->Gain * Out * (1-Out) * Err;
  }
}


void BackpropagateNet(NET* Net)
{
  INT l;

  for (l=NUM_LAYERS-1; l>1; l--) {
    BackpropagateLayer(Net, Net->Layer[l], Net->Layer[l-1]);
  }
}


void AdjustWeights(NET* Net)
{
  INT  l,i,j;
  REAL Out, Err, dWeight;

  for (l=1; l<NUM_LAYERS; l++) {
    for (i=1; i<=Net->Layer[l]->Units; i++) {
      for (j=0; j<=Net->Layer[l-1]->Units; j++) {
        Out = Net->Layer[l-1]->Output[j];
        Err = Net->Layer[l]->Error[i];
        dWeight = Net->Layer[l]->dWeight[i][j];
Net->Layer[l]->Weight[i][j] += Net->Eta * Err * Out + Net->Alpha * dWeight;
        Net->Layer[l]->dWeight[i][j] = Net->Eta * Err * Out;
      }
    }
  }
}


/******************************************************************************
                      S I M U L A T I N G   T H E   N E T
 ******************************************************************************/


void SimulateNet(NET* Net, REAL* Input, REAL* Output, REAL* Target, BOOL Training)
```

```
{
  SetInput(Net, Input);
  PropagateNet(Net);
  GetOutput(Net, Output);

  ComputeOutputError(Net, Target);
  if (Training)
  {
    BackpropagateNet(Net);
    AdjustWeights(Net);
  }
}


void TrainNet(NET* Net, INT Epochs)
{
  INT  p, n,k,m;
  REAL Output[M];

  for (n=0; n<Epochs; n++) //n<Epochs*NUM_SAMPLES
  {
  m = -1;

  for (p=0; p<NUM_SAMPLES; p++)
  {
  if (TrainingLabel[p]!=0)
  {
 // m++;

 //if (m>NUM_SAMPLES*TrainingRate*0.7) continue;

  //   Year = RandomEqualINT(TRAIN_LWB, TRAIN_UPB);
//SimulateNet(Net, &(Sunspots[Year-N]), Output, &(Sunspots[Year]), TRUE);
SimulateNet(Net, &(InputFeatures[p][0]),Output, &(LabelResult[p]), TRUE);

  }
  }
  }
}


void TestNet(NET* Net,INT flag, REAL *fTraining, REAL *fTest)
{
  INT  n,p,kk,m;
  REAL Output[M];

  INT Hist0[1002],Hist1[1002]; //first 0/1 is GT, next 0/1 is detected
  REAL SUM1=0,SUM0=0, DIFF=1, fTr, fTe;

  float threshold;

  int iGT_pos = 0, iGT_neg = 0;
  int iTp =0, iFp = 0, iMissed = 0, iTn = 0;


  //Need determine a threshold for class 1, i.e. large than a threshold
  is 1 otherwise is 0, certainly the threshold
  //should more than 0.5 to remove false alarms or less than 0.5 to
  recover missed samples. Finally, three thresholds
  //need to be set using the training data hence better statistics!!!

  //statistics as histogram: 0.25-0.75 500 levels each level represent 0.001
```

```
   memset(Hist0,0,1002*sizeof(INT));
   memset(Hist1,0,1002*sizeof(INT));

   if (flag!=0)
   {
   n = 0;
   }

  TrainError = 0;

  fprintf(f, "\n\nTraining Result index %d\n",iIndexTraining);

  m = -1;

  for (p=0; p<NUM_SAMPLES; p++)
  {
    if (TrainingLabel[p]!=0)
{
//m++;

//if (m>NUM_SAMPLES*TrainingRate*0.7)
//  continue;

SimulateNet(Net, &(InputFeatures[p][0]),Output, &(LabelResult[p]), FALSE);
//SimulateNet(Net, &(Sunspots[Year-N]), Output, &(Sunspots[Year]), FALSE);


    TrainError += Net->Error;


if (LabelResult[p] == 1)
{
iGT_pos++;

 //memset(Hist00,0,1000*sizeof(INT));
  //memset(Hist01,0,1000*sizeof(INT));
  //memset(Hist10,0,1000*sizeof(INT));
  //memset(Hist11,0,1000*sizeof(INT));

kk= (int)(Output[0]*1000+0.5);

//for (;kk<1000;kk++)  // //accumulated histogram?
Hist1[kk]++;


SUM1+=Output [0];

if (Output [0]>0.5)
{
iTp ++;

if (flag!=0)
fprintf(f, "%d   %0.3f %0.3 %s\n",p, LabelResult[p], Output [0],"1");
}
else
{
iMissed++;

if (flag!=0)
fprintf(f, "%d   %0.3f  %0.3f %s\n",p, LabelResult[p], Output [0],"1==>0");
}
}
else
{
iGT_neg++;

kk= (int)(Output[0]*1000+0.5);

//for (;kk<1000;kk++)  // //accumulated histogram?

Hist0[kk]++;

SUM0+=Output [0];

if (Output [0]>0.5)
{
iFp++;

if (flag!=0)
fprintf(f, "%d   %0.3f  %0.3f %s\n",p, LabelResult[p], Output [0],"0==>1");
}
else
{
if (flag!=0)
fprintf(f, "%d %0.3f %0.3f %s\n",p, LabelResult[p],  Output [0],"0");

iTn++;
}

}

}
  }

  if (flag!=0)
  {
  //output the histogram

     fprintf(f, "\n\n Histogram of output\n");
fprintf(f, "\nIndex Th Pos-s Neg-s Pos-a Neg-a\n");

SUM1=0; SUM0=1;

  for (kk=0; kk<1000; kk++)
  {
  fprintf(f, "%d %.3f %.5f %.5f %.5f %.5f\n",
            kk, 0.001*kk, 1.0*Hist1[kk]/iGT_pos,
            1.0*Hist0[kk]/iGT_neg,SUM1,SUM0);

  SUM1 += 1.0*Hist1[kk]/iGT_pos;
  SUM0 -= 1.0*Hist0[kk]/iGT_neg;

  if (fabs(SUM1-SUM0)<=DIFF)
  {
  ThOptimal = 0.001*(kk+1);
  DIFF = fabs(SUM1-SUM0);
  }
  }


    fprintf(f,"Optimal Threshold %.5f\n",ThOptimal);
  }

    fprintf(f, "\n GT-Pos GT-Neg Sum\n");
fprintf(f, "Pos %d %d %d\n",iTp,iFp,iTp+iFp);
fprintf(f, "Neg %d %d %d\n",iMissed,iTn,iMissed+iTn);
fprintf(f, "Sum %d %d %d\n",iTp+iMissed,iFp+iTn,iMissed+iTn+iTp+iFp);

fprintf(f, "Overall accuracy: %.3f\n",1.0*(iTp+iTn)/(iGT_pos+iGT_neg));

if (iTp+iFp !=0)
fprintf(f, "Sensitivity: %.3f specificity: %.3f False Alarm Rate: %.3f\n",
            1.0*iTp/iGT_pos,1.0*iTp/(iTp+iFp),1.0*iFp/iGT_neg);
else
        fprintf(f, "Sensitivity: %.3f specificity: %.3f False Alarm Rate: %.3f\n",
            1.0*iTp/iGT_pos,0,1.0*iFp/iGT_neg)

if (fTraining!=NULL)
*fTraining = 2*1.0*iTp/iGT_pos*1.0*iTn/iGT_neg/(1.0*iTp/iGT_pos+1.0*iTn/iGT_neg);

if (flag!=0)
fprintf(f, "\n Test Result Th=0.5\n");
else fprintf(f, "\n Test Result\n");


  TestError = 0;

  iGT_pos = 0;   iGT_neg = 0;
  iTp =0; iFp = 0; iMissed = 0; iTn = 0;


  m = -1;

   for (p=0; p<NUM_SAMPLES; p++)
  {
  /*
   if (TrainingLabel[p]!=0 && flag == 0)
   {
   m++;

  if (m<=NUM_SAMPLES*TrainingRate*0.7)
     continue;

   if (Duplicated[p] == 1 ) continue;

SimulateNet(Net, &(InputFeatures[p][0]),Output, &(LabelResult[p]), FALSE);//
 SimulateNet(Net, &(Sunspots[Year-N]), Output, &(Sunspots[Year]), FALSE);
     TestError += Net->Error;

if (LabelResult[p] == 1)
{
iGT_pos++;

if (Output [0]>0.5)
{
iTp ++;
// fprintf(f, "%d  %0.3f    %0.3f %s\n",p, LabelResult[p], Output [0],"True Positive");
}
else
{
iMissed++;
// fprintf(f, "%d   %0.3f %0.3f %s\n",p, LabelResult[p], Output [0],"Missed Positive");
}
```

```
}
else
{
iGT_neg++;

if (Output [0]>0.5)
{
iFp ++;
// fprintf(f, "%d  %0.3f  %0.3f %s\n",p, LabelResult[p], Output [0],"False Positive");
}
else
{
// fprintf(f, "%d %0.3f %0.3f %s\n",p, LabelResult[p], Output [0],"True Negative");
iTn++;
}

}


   }*/


    if (TrainingLabel[p]==0)
{

if (Duplicated[p] == 1 && flag ==1) continue;

SimulateNet(Net, &(InputFeatures[p][0]),Output, &(LabelResult[p]),
FALSE);// SimulateNet(Net, &(Sunspots[Year-N]), Output, &(Sunspots[Year]), FALSE);
     TestError += Net->Error;

if (LabelResult[p] == 1)
{
iGT_pos++;

if (Output [0]>0.5)
{
iTp ++;
// fprintf(f, "%d %0.3f %0.3f %s\n",p, LabelResult[p],Output [0],"True Positive");
}
else
{
iMissed++;
// fprintf(f, "%d %0.3f %0.3f %s\n",p, LabelResult[p],Output [0],"Missed Positive");
}

}
else
{
iGT_neg++;

if (Output [0]>0.5)
{
iFp ++;
// fprintf(f, "%d %0.3f %0.3f %s\n",p, LabelResult[p], Output [0],"False Positive");
}
else
{
// fprintf(f, "%d %0.3f %0.3 %s\n",p, LabelResult[p], Output [0],"True Negative");
iTn++;
}
```

```
}

}

  }

    fprintf(f, "\n GT-Pos GT-Neg Sum\n");
fprintf(f, "Pos %d %d %d\n",iTp,iFp,iTp+iFp);
fprintf(f, "Neg %d %d %d\n",iMissed,iTn,iMissed+iTn);
fprintf(f, "Sum %d %d %d\n",iTp+iMissed,iFp+iTn,iMissed+iTn+iTp+iFp);

fprintf(f, "Overall accuracy: %.3f\n",1.0*(iTp+iTn)/(iGT_pos+iGT_neg));

if (iTp+iFp != 0)
fprintf(f, "Sensitivity: %.3f specificity: %.3f False Alarm Rate: %.3f\n",
            1.0*iTp/iGT_pos,1.0*iTp/(iTp+iFp),1.0*iFp/iGT_neg);
else
        fprintf(f, "Sensitivity: %.3f specificity: %.3f False Alarm Rate: %.3f\n",
            1.0*iTp/iGT_pos,0,1.0*iFp/iGT_neg);

        fprintf(f, "\nNMSE is %0.3f on Training Set and %0.3f on Test Set",
            TrainError / TrainErrorPredictingMean,
            TestError / TestErrorPredictingMean);

 if (fTest!=NULL)
*fTest = 2*1.0*iTp/iGT_pos*1.0*iFp/iGT_neg/(1.0*iTp/iGT_pos+1.0*iFp/iGT_neg);

 iIndexTraining++;



}


void EvaluateNet(NET* Net)
{
  INT  p;
  REAL Output [M];
  REAL Output_[M];
  INT index = 1;

  int iGT_pos = 0, iGT_neg = 0;
  int iTp =0, iFp = 0, iMissed = 0, iTn = 0;
  float threshold;

  fprintf(f, "\n\n\n");
  fprintf(f, "index Sample Class Output Result\n");
  for (p=0; p<NUM_SAMPLES; p++)
  {
    if (TrainingLabel[p]==0 && Duplicated[p] == 0)
{
//SimulateNet(Net, &(Sunspots [Year-N]), Output,  &(Sunspots [Year]), FALSE);
SimulateNet(Net, &(InputFeatures[p][0]),Output, &(LabelResult[p]), FALSE);

/*SimulateNet(Net, &(Sunspots_[Year-N]), Output_, &(Sunspots_[Year]), FALSE);
Sunspots_[Year] = Output_[0];
fprintf(f, "%d  %0.3f %0.3f %0.3f\n",
FIRST_YEAR + Year,
Sunspots[Year],
Output [0],
Output_[0]);
```

```
*/
if (LabelResult[p] == 1)
{
iGT_pos++;

if (Output [0]>ThOptimal)
{
iTp ++;
fprintf(f, "%d %d %0.3f %0.3f %s\n",index, p, LabelResult[p],Output [0],"1");

}
else
{
iMissed++;
fprintf(f, "%d %d %0.3f %0.3f %s\n",index,p, LabelResult[p],  Output [0],"1-->0");


}

}
else
{
iGT_neg++;

if (Output [0]>ThOptimal)
{
iFp ++;
fprintf(f, "%d %d %0.3f %0.3f %s\n",index, p, LabelResult[p],Output [0],"0-->1");
}
else
{

fprintf(f, "%d %d %0.3f %0.3f %s\n",index, p, LabelResult[p],Output [0],"0");
iTn++;
}

}

index++;
}
  }

  /*output confusion matrix and overall correction rate*/

    fprintf(f, "\n GT-Pos GT-Neg Sum\n");
fprintf(f, "Pos %d %d %d\n",iTp,iFp,iTp+iFp);
fprintf(f, "Neg %d %d %d\n",iMissed,iTn,iMissed+iTn);
fprintf(f, "Sum %d %d %d\n",iTp+iMissed,iFp+iTn,iMissed+iTn+iTp+iFp);

fprintf(f, "Overall accuracy: %.3f\n",1.0*(iTp+iTn)/(iGT_pos+iGT_neg));

if (iGT_pos == 0)
iGT_pos = 1;


if (iTp+iFp!=0)
fprintf(f, "Sensitivity: %.3f specificity: %.3f False Alarm Rate: %.3f\n",
            1.0*iTp/iGT_pos,1.0*iTp/(iTp+iFp),1.0*iFp/iGT_neg);
else
        fprintf(f, "Sensitivity: %.3f specificity: %.3f False Alarm Rate: %.3f\n",
            1.0*iTp/iGT_pos,0,1.0*iFp/iGT_neg);
```

```
/*To test ROC results*/

   threshold = 0;


   fprintf(f, "\n\nTesting ROC\n");
   fprintf(f, "Thresh Recall Prec FAR Accuracy\n");

loopROC:


   iGT_pos = 0; iGT_neg = 0;
   iTp =0;
   iFp = 0;
   iMissed = 0;
   iTn = 0;

  for (p=0; p<NUM_SAMPLES; p++)
  {
if (TrainingLabel[p]==0 && Duplicated[p] == 0) //if (TrainingLabel[p]==0
 && Duplicated[p] == 0) this refers to testing performance!
{
//SimulateNet(Net, &(Sunspots [Year-N]), Output,  &(Sunspots [Year]), FALSE);
SimulateNet(Net, &(InputFeatures[p][0]),Output, &(LabelResult[p]), FALSE);

/*SimulateNet(Net, &(Sunspots_[Year-N]), Output_, &(Sunspots_[Year]), FALSE);
Sunspots_[Year] = Output_[0];
fprintf(f, "%d    %0.3f   %0.3f      %0.3f\n",
FIRST_YEAR + Year,
Sunspots[Year],
Output [0],
Output_[0]);
*/
if (LabelResult[p] == 1)
{
iGT_pos++;

if (Output [0]>=threshold)
iTp ++;
else iMissed++;

}
else
{
iGT_neg++;

if (Output [0]>=threshold)
iFp ++;
else iTn++;

}

index++;
}
  }

  /*output confusion matrix and overall correction rate*/


if (iGT_pos == 0)
iGT_pos = 1;
```

```
if (iTp+iFp!=0)
fprintf(f, "%.5f %.5f %.5f %.5f %.5f\n",threshold,
        1.0*iTp/iGT_pos,1.0*iTp/(iTp+iFp),
        1.0*iFp/iGT_neg,1.0*(iTp+iTn)/(iGT_pos+iGT_neg));
else
fprintf(f, "%.5f %.5f %.5f %.5f %.5f\n",threshold,
            1.0*iTp/iGT_pos,0,1.0*iFp/iGT_neg,1.0*(iTp+iTn)/(iGT_pos+iGT_neg));



if (threshold<0.001 || threshold>0.999)
threshold += 0.00002;
else
threshold += 0.001;

if (threshold<1) goto loopROC;


/*To test training ROC results*/

   threshold = 0;


   fprintf(f, "\n\nTraning ROC\n");
   fprintf(f, "index Sample Class Output Result\n");
   fprintf(f, "Thresh Recall Prec FAR Accuracy\n");


loopROC2:


   iGT_pos = 0; iGT_neg = 0;
   iTp =0;
   iFp = 0;
   iMissed = 0;
   iTn = 0;

  for (p=0; p<NUM_SAMPLES; p++)
  {
if (TrainingLabel[p]!=0) //if (TrainingLabel[p]==0 && Duplicated[p] == 0)
this refers to testing performance!
{
//SimulateNet(Net, &(Sunspots [Year-N]), Output,  &(Sunspots [Year]), FALSE);
SimulateNet(Net, &(InputFeatures[p][0]),Output, &(LabelResult[p]), FALSE);

/*SimulateNet(Net, &(Sunspots_[Year-N]), Output_, &(Sunspots_[Year]), FALSE);
Sunspots_[Year] = Output_[0];
fprintf(f, "%d %0.3f %0.3f  %0.3f\n",
FIRST_YEAR + Year,
Sunspots[Year],
Output [0],
Output_[0]);
*/
if (LabelResult[p] == 1)
{
iGT_pos++;

if (Output [0]>=threshold)
iTp ++;
else iMissed++;
}
else
{
iGT_neg++;

if (Output [0]>=threshold)
iFp ++;
else iTn++;

}

index++;
}
  }

  /*output confusion matrix and overall correction rate*/


if (iGT_pos == 0)
iGT_pos = 1;


if (iTp+iFp!=0)
fprintf(f, "%.5f %.5f %.5f %.5f %.5f\n",threshold,
        1.0*iTp/iGT_pos,1.0*iTp/(iTp+iFp),
        1.0*iFp/iGT_neg,1.0*(iTp+iTn)/(iGT_pos+iGT_neg));
else
fprintf(f, "%.5f %.5f %.5f %.5f %.5f\n",threshold,
        1.0*iTp/iGT_pos,0,1.0*iFp/iGT_neg,1.0*(iTp+iTn)/(iGT_pos+iGT_neg));

if (threshold<0.001 || threshold>0.999)
threshold += 0.00002;
else
threshold += 0.001;

if (threshold<1) goto loopROC2;

}


#define MaxFeature 1024


void my_load_dataNew(char *oldFileName) //feature will be loaded to InputFeatures
// loads the same format as LIBSVM
{
#define IgnoreComments 1
double fMaxRemain=0; /*maximum remain when fLabels is changed as an integer*/

double fSamples[NUM_SAMPLES][40]; /*full matrix in vector format, size: m*max_index  */
double fLables[NUM_SAMPLES]; /*label vector size: m   */


    int index;
double value;
int elements, i,lineSize, indexSample, j, k;

FILE *fp = NULL;

fp = fopen(oldFileName,"rt"); //
```

```
//FILE *fNewBin = NULL;
int splitpos=0;
char lineBuffer[4096],str[300];
long int fpos1,fpos2;

int FullCommentsOn, CommentsOn;
double label;

int msz = 0;
int max_index = 0;
int m; //


float fMaxFeature[MaxFeature]; /*40960 features*/


for (k=0; k<MaxFeature; k++)
fMaxFeature[k] = 0;

if(fp == NULL)
{
fprintf(stderr,"Can't open input file \"%s\"\n",oldFileName);
return;
}
    else
    printf("\"%s\"..  ",oldFileName);

elements = 0;

lineSize = 0;

FullCommentsOn = 0;
CommentsOn = 0;

while(1)
{
int c = fgetc(fp);
switch(c)
{
case '#':
if (lineSize==0)
{
FullCommentsOn = 1;
break;
}
else
{
CommentsOn = 1;
lineSize ++;
}
break;
case '\n':
if (FullCommentsOn !=1)
++msz;
//printf("%d\n",m);
elements=0;

FullCommentsOn = 0;
CommentsOn = 0;
lineSize = 0;

break;
case ':':
if (CommentsOn!=1)
++elements;
break;
case EOF:
goto out;
default:
if (FullCommentsOn+CommentsOn!=0)
break;

fscanf(fp,"%lf",&label);
while(1)
{
int c;
do {
c = getc(fp);
if(c=='\n') goto out2;
} while((char)(c)==' ');  //(isspace(c));

ungetc(c,fp);
fscanf(fp,"%d:%lf",&index,&value);

if (index>max_index) max_index=index;

if (index<MaxFeature)
fMaxFeature[index] = MAX(fMaxFeature[index],fabs(value));
else
{
printf("Feature vector too large than the buffer size %d!\n",MaxFeature);
}
}
out2:
if (FullCommentsOn !=1)
++msz;
//printf("%d\n",m);
elements=0;

FullCommentsOn = 0;
CommentsOn = 0;
lineSize = 0;

break;

}
}
out:
rewind(fp);

m = msz;

printf("examples: %d   features: %d\n",msz,max_index);

//fSamples = InputFeatures;  /*full matrix in vector format, size: m*max_index  */
//fLables = LabelResult; //new double[msz]; /*label vector size: m   */

memset(fSamples,0,msz*max_index*sizeof(float));

FullCommentsOn = 0;
CommentsOn = 0;
lineSize = 0;
indexSample = 0;
```

```
while(1)
{
int c = fgetc(fp);
switch(c)
{
case '#':
if (lineSize==0)
FullCommentsOn = 1;
else
CommentsOn = 1;

if (IgnoreComments == 0)
lineBuffer[lineSize++] = c;
break;
case '\n':
lineBuffer[lineSize++] = c;
lineBuffer[lineSize] = 0;

elements=0;
FullCommentsOn = 0;
CommentsOn = 0;
lineSize = 0;

break;
case EOF:
if (lineSize>0)
{
// lineBuffer[lineSize++] = c;
lineBuffer[lineSize] = 0;

}
goto out4;
default:
if (FullCommentsOn+CommentsOn!=0)
{
lineBuffer[lineSize++] = c;
break;
}

ungetc(c,fp);

fpos1 = ftell(fp);
fscanf(fp,"%lf",&label);

fMaxRemain = MAX(fMaxRemain,fabs(label-(int)(label)));

if (label<0)
{
k = 1;
}
fpos2 = ftell(fp);
fseek(fp,fpos1,SEEK_SET);

for (j=fpos1;j<fpos2;j++)
lineBuffer[lineSize++] = getc(fp);

if (label<0) label = 1; /*added for ANN, Jinchang 02/06/09*/
else label = 0;

fLables[indexSample] = label;
while(1)
{
int c;
do {
c = getc(fp);

lineBuffer[lineSize++] = (char)(c);

if(c=='\n') goto out3;
} while((char)(c)==' ');  //(isspace(c));

lineSize--;
ungetc(c,fp);
fscanf(fp,"%d:%lf",&index,&value);

if (index>max_index) max_index=index;

if (index<MaxFeature && fMaxFeature[index]!=0)
{
value = value/fMaxFeature[index]; // = max(fMaxFeature[index],fabs(value));

fSamples[indexSample][index-1] = value;
}
else
{
printf("Feature %d underflow in normalisation for sample %d!\n",index,indexSample);
}

sprintf(str,"%d:%lf",index,value);

for (k=0; k<strlen(str); k++)
lineBuffer[lineSize++] = str[k];

}
out3:
lineBuffer[lineSize] = 0;

//printf("%d\n",m);
elements=0;

FullCommentsOn = 0;
CommentsOn = 0;
lineSize = 0;
indexSample++;



break;

}
}




out4:

fclose(fp);

//copy the features to our buffer
```

```
for (j=0; j<NUM_SAMPLES; j++)
{

if (fLables[j]>0.5)
LabelResult[j] = 1;
else
LabelResult[j] = 0;

k = 0;
for (m=0; m<39; m++)
{
if (m<20 || m==24 || m==25 || m==38)
{
InputFeatures[j][k] = fSamples[j][m];

k++;

}

}
}





}

/******************************************************************************
                                    M A I N
 ******************************************************************************/



void main()
{
  NET  Net;
  BOOL Stop;
  REAL MinTestError, Error;
  int m,n,k,pp;
  FILE *fileTraining = NULL;
  FILE *fileTesting = NULL;
  //FILE *fileTesting = NULL;
  char str[40];

  REAL fSTopCriteria=1000, fTrain,fTest;
  int  iLoopIndex=0;

  int  iLoopApps = 0;


startapp:

  fSTopCriteria=0;
  fileTraining = NULL;
  fileTesting = NULL;
  iLoopIndex=0;
  ThOptimal = 0.5;
//initialize the apps
```

```
  InitializeRandoms();
  GenerateNetwork(&Net);
  RandomWeights(&Net);
  InitializeApplication(&Net);

   // Read input data

  if (UsingOriginalSamples==1)
   my_load_dataNew("Samples.dat");
  else
  my_load_dataNew("SamplesBalanced.dat");


  memset(Duplicated,0 ,NUM_SAMPLES);

  n = -1;
  for (m=0; m<NUM_SAMPLES; m++)
  {
  if (LabelResult[m]<0.5)
  {
  n = -1;
  continue;
  }

  if (n<0) n = m;
  else
  {
  pp = 0;

  for (k=0; k<FEATURE_DIM; k++)
if (100000*InputFeatures[m][k]!=100000*InputFeatures[n][k]) pp++;

  if (pp<=1)
  Duplicated[m] = 1;
  else n = m;
  }


  }

    //select training data

  m = 0;

  memset(TrainingLabel, 0,NUM_SAMPLES);

  for (;;)
  {
  n = rand() % NUM_SAMPLES;

  if (TrainingLabel[n] == 0)
  {
  m++;

  TrainingLabel[n] = 1;

  if (m>=NUM_SAMPLES*TrainingRate)
  break;
  }
  }

   /*save index of training samples*/
  sprintf(str,"training%db%d_t%d.dat",iRandomStart,1-UsingOriginalSamples,
  (int)(TrainingRate*100));
  fileTraining = fopen(str,"wt");

  sprintf(str,"testing%db%d_t%d.dat",iRandomStart,1-UsingOriginalSamples,
  (int)(TrainingRate*100));
  fileTesting = fopen(str,"wt");

  for (n=0; n<NUM_SAMPLES; n++)
  {
  m = TrainingLabel[n];

  if (m==1)
  {
  //output to training file


if (LabelResult[n] >0)
fprintf(fileTraining,"+1");
else
fprintf(fileTraining,"-1");


for (k=0; k<FEATURE_DIM; k++)
{
fprintf(fileTraining," %d:%f",k+1,InputFeatures[n][k]);
}

fprintf(fileTraining," #%d\n",n+1);

  }
  else
  {
  if (Duplicated[n] == 1) continue;

  //output to test file
  if (LabelResult[n] >0)
fprintf(fileTesting,"+1");
else
fprintf(fileTesting,"-1");

for (k=0; k<FEATURE_DIM; k++)
{
fprintf(fileTesting," %d:%f",k+1,InputFeatures[n][k]);
}

fprintf(fileTesting," #%d\n",n+1);

  }
  }

  fclose(fileTraining);
  fclose(fileTesting);

  /*end of save index of training samples*/


//goto end;

  //training the data
  Stop = FALSE;
  MinTestError = MAX_REAL;
  Error = MAX_REAL;
  do {
    TrainNet(&Net, 10);
    TestNet(&Net,FALSE,&fTrain,&fTest);


if (TrainError>= Error*1.2) //from Jichang 0.9999
{
      fprintf(f, " - stopping Training and restoring Weights ...");
      Stop = TRUE;
      RestoreWeights(&Net);
}
else
{
  fprintf(f, " - saving Weights ...");
      MinTestError = TestError;

  Error = TrainError;

  if (fSTopCriteria<fTrain) // && iIndexTraining>200
  {
fSTopCriteria = fTrain;
iLoopIndex = iIndexTraining;

SaveWeights(&Net);

  }
else if (iIndexTraining-iLoopIndex>200+300*(1+UsingOriginalSamples))
  {
fprintf(f, " - stopping Training and restoring Weights ...");
  Stop = TRUE;
  SaveWeights(&Net);
          //RestoreWeights(&Net);

  }
  //
}



/*
    if (TestError < MinTestError)
{
      fprintf(f, " - saving Weights ...");
      MinTestError = TestError;
      SaveWeights(&Net);

 }
    else if (TestError > 1.2 * MinTestError)
{
//if (TestError<TrainError*1.2)
{
fprintf(f, " - stopping Training and restoring Weights ...");
Stop = TRUE;
RestoreWeights(&Net);
}
    }

*/
```

```
  } while (NOT Stop);


  //test the network


  TestNet(&Net,TRUE,NULL,NULL);
  EvaluateNet(&Net);

end:

  FinalizeApplication(&Net);

  iLoopApps++;

 // if (iLoopApps<10)
 //   goto startapp;
}
```

Appendix II - Published Papers

Skin Detection from Different Color Spaces for Model-based Face Detection

Dong Wang[1], Jinchang Ren[1], Jianmin Jiang[1], Stan S. Ipson[1]

[1] School of Informatics, University of Bradford, BD7 1DP, U. K.
{ D.Wang6,J.Ren,J.Jiang1, S.S.Ipson}@bradford.ac.uk

Abstract. Skin and face detection has many important applications in intelligent human-machine interfaces, reliable video surveillance and visual understanding of human activities. In this paper, we propose an efficient and effective method for frontal-view face detection based on skin detection and knowledge-based modeling. Firstly, skin pixels are modeled by using supervised training, and boundary conditions are then extracted for skin segmentation. Faces are further detected by shape filtering and knowledge-based modeling. Skin results from different color spaces are compared. In addition, experimental results have demonstrated our method robust in successful detection of skin and face regions even with variant lighting conditions and poses.

Keywords: Skin detection, face detection, performance evaluation, semantic image indexing and retrieval.

1 Introduction

Automatic detection of skin and face plays very important roles in many vision applications, such as face and gesture recognition in intelligent human-machine intelligence and visual surveillance [1,2,4], naked adult image detection [3,8], video phone or sign language recognition [14, 15] as well as content-based multimedia retrieval [5, 13].

Usually, skins regions are segmented or detected by histogram matching, statistical classification and pixel-based thresholding or clustering. In [8], Jones and Rehg developed a general color model from color histograms in R, G and B channels and adopted supervised learning by manually labeling skin pixels in 4675 images to acquire the probability that a given color belonged a skin and non-skin classes. Then, they tested the method in another 8965 images to detect skins and judge naked images. Saber and Tekalp employed a YES model to detect skins from color images by linear weighting of R, G and B values like YUV space did [9]. In [10], Hsu etc used YCbCr space for skin detection in their face detection system and found after lighting compensation their algorithm could detect more accurate skin pixels. In [11], Garcia and Tziritas compared skin detection results obtained from color clustering, and found the results in YCbCr and HSV spaces are quite equivalent. They also concluded that the cluster of skin colors is less compact in HSV space than in YCbCr space, and

HSV space is more sensitive to lighting variations. In addition, some adaptive models are also proposed for skin detection [16-19].

After skin detection, faces can be detected by shape constraints, template matching, knowledge conducting and statistical classifications by Neural Network (NN), Hidden-Markov Model (HMM) or Support Vector Machine (SVM) [6, 9-11]. As statistical or neural classification is always implemented by supervised or unsupervised learning, in which many face features are applied internally, therefore, face features and knowledge of facial distribution is more important in face detection. Saber and Tekalp introduced an interesting algorithm to detect face by locating the eyes, nose and mouth [9]. At the same time, they took an ellipse to simulate the shape of a face just as many other people adopted [6, 10]. In [10], Hsu etc also detected faces from skin regions based on the spatial arrangement of skin patches like eyes, mouth and boundary maps and could attain the face ellipse and triangle feature points of eyes and mouth. In [11], Garcia and Tziritas presented another face detection method based on skin detection, region merging and constraints of shape, size and intensity texture analyzed by wavelet-packet decomposition, and they utilized rectangle shapes to mark the detected faces. In [12], face is detected by using a fuzzy pattern matching scheme. In [20], merging skin regions for face detection by using wavelet analysis is proposed.

From skin detection to face detection, we have quite a few algorithms with different results and conclusions [8-12], and some of them are quite different and even disagreed with each other. Therefore, we will comparatively investigate skin detection and employ same methodology in different color spaces during the comparisons before our knowledge-based face modeling and detection.

2 Color Space Transform

Both linear and nonlinear color spaces are examined in our paper for comparisons. To provide some direct information about these color spaces, we summarize below how typical color space transforms, including linear and nonlinear ones, are defined.

From RGB to YCbCr or YUV color spaces are linear transforms, in which the three components in both new spaces are defined simply by linear weighting of R, G and B values, and Y refers to illumination intensity defined in (1),

$$Y = \sum \lambda w_\lambda, \lambda = R, G, B \qquad \sum w_\lambda = 1, w_\lambda \geq 0 \qquad (1)$$
$$U = B - Y, V = R - Y.$$

Generally, these linear transforms can be defined as

$$\begin{bmatrix} Y \\ A \\ B \end{bmatrix} = \begin{bmatrix} w_r & w_g & w_b \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \qquad (2)$$

As hue is more effective in distinguishing different colors than illumination intensity, HSV (Hue, Saturation, Value) and HIS (Hue, Intensity, Saturation) transforms are taken as suitable color spaces that correspond to human visual perceptions and have been widely utilized in color clustering for image segmentation and coding.

The RGB to HSV transform can be defined as [7]:

$$V = \max(R, G, B); \qquad S = V'/V \qquad V' = V - M, M = \min(R, G, B) \tag{3}$$

Let $r' = (V - R)/V'$, $g' = (V - G)/V'$ and $b' = (V - B)/V'$, then H is given by

$$H = \frac{1}{6}\begin{cases} 5 + b' & if \quad R = V \,\&\, G = M \\ 1 - g' & if \quad R = V \,\&\, G \neq M \\ 1 + r' & if \quad G = V \,\&\, B = M \\ 3 - b' & if \quad G = V \,\&\, B \neq M \\ 3 + g' & if \quad B = V \,\&\, R = M \\ 5 + r' & \text{otherwise} \end{cases} \tag{4}$$

3 Skin Segmentation

Since skin detection is a classification problem defined on color similarity, supervised clustering is applied to achieve the exact rules for effective skin color clustering and pixel classification. Through manually specifying representative skin and non-skin pixels, we can learn linear relationships between different components in the new color spaces. Finally, we obtain several main boundary conditions for skin pixels classification in different color spaces.

Firstly, skin pixels are modeled by using the histogram-based approach, in which the probability or likelihood that each color represents skin is estimated by checking its occurrence ratio in the training data. In (5), V_{skin} indicates volumes or total occurrences of all skin colors in manual ground truth of training data.

$$p(color \,/\, skin) = sum(color \,/\, skin) / V_{skin}. \tag{5}$$

Then, boundary conditions in the skin model are extracted to allow more than 98% of skin pixels covered. Using the boundary conditions, test images are segmented into skin and nonskin regions accordingly. For different color spaces, these boundary conditions are found as follows.

As for YUV space, the boundary conditions are found as:

$$\begin{cases} 148 \le V \le 185 \\ 189 \le U + 0.6V \le 215 \end{cases}. \tag{6}$$

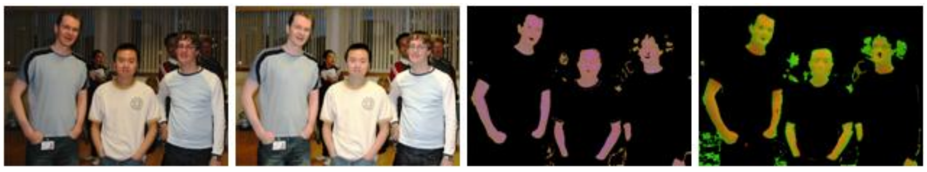
Considering the illumination intensity variation, we have boundary conditions as

$$\begin{cases} Y > 85 \quad or \\ Y < 85, U > 104, Y + U - V > 2 \end{cases}. \tag{7}$$

In HSV space, we scale the H into [0,255] and let $H = 255 - H$ if $H > 128$. We also find several boundary conditions for skin pixels in HSV space and given in (8):

$$\begin{cases} S \le 21, V \ge 2.5S \\ 158 \le H + V \le 400, H + V > 13S \\ H > 0.2V, H > 4S \end{cases}. \tag{8}$$

Fig 1 gives skin results from YUV and HSV spaces, where we can find three people in the background, which can be clearly found in the histogram-equalized image. However, the skin regions can be successfully in HSV space from the original image while they cannot be found in YUV space.



(a) Original image (b) histogram equation (c) YUV skin results (d) HSV skin results

Fig. 1. Comparison of skin regions detected from YUV and HSV color spaces.

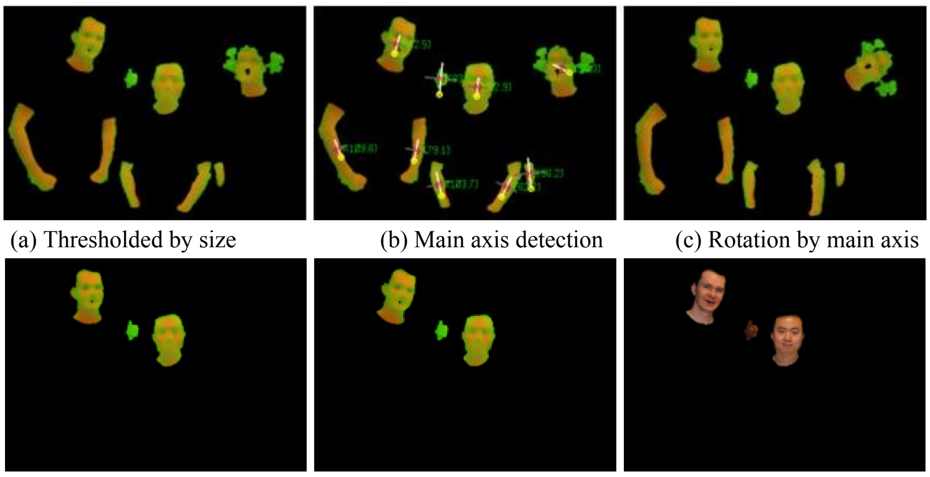
4 Knowledge-based Face Modeling and Detection

After skin detection, we need to locate faces in candidate skin regions. Again we detect faces of nearly frontal view, but there are no constraints on their leaning angles. Knowledge about the size, size ratio, locations of ears and mouth is used.

Firstly, the detected skin regions are labeled to obtain the outer boundary rectangle and pixel number of every region. Then, small regions that have pixels less than a given threshold, i.e. 300, will be removed. Finally, the skin regions are filtered by a *SR* parameter (Width/Height ratio) defined as,

$$SR = \begin{cases} width / height & if \quad width \le height \\ height / width & if \quad width > height \end{cases}. \tag{9}$$

In (10), the width and height of the regions are determined by the rectangle bounding box of each region, and we find the valid SR for candidate face regions should lie in [0.55,0.90]. To acquire more reasonable width and height of the regions, the main axis is extracted by moment calculation of each region. Then, the skin regions are rotated by the main axis angle to make the final main axis in vertical direction.

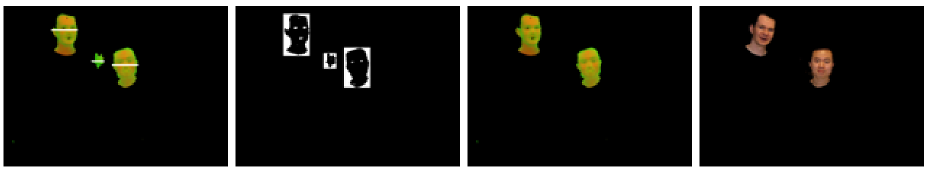


(a) Thresholded by size (b) Main axis detection (c) Rotation by main axis

(d) Thresholded by W/H ratio (e) Face candidates (f) Face in original image

Fig. 2. Face filtering from skin regions in Fig 2(b) by thresholding of size and W/H ratio.

Fig 2(a) is the filtering result by thresholding using the size of 300. In Fig 2(b), the main axis of each labeled region is marked with white line, and the angle and region number are also given. From Fig 2(d) to 2(e), we give the candidate face regions in rotated skin results and the skin regions before rotation in HSV space. Besides, Fig 2(f) gives the face candidates in RGB space for comparison.



(a) Ears location (b) Feature holes (c) detected face (d) mapped back

Fig. 3. Ears location with white line (a) and feature holes detection (b) for face detection

Three basic rules are used in further face modeling and detection: First, there are one or two ears near the half height of every candidate face region which makes the width of the skin regions bigger than other lines. Second, there are one or two eyes over the height of the ear line which forms one or two dark holes. Third, an open mouth will form a dark hole near the middle of eyes below the ear line. Following is our algorithm for face detection and the results are given in Fig 3.

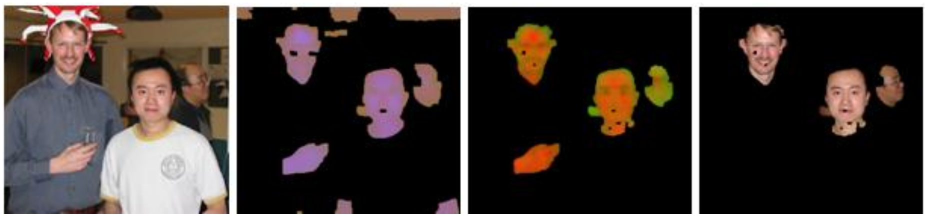
1) Detect the ear line by extracting of local maximum width near the center of the candidate face regions, see Fig 3(a);
2) Detect the holes by the illumination intensity difference. Holes contain those pixels that have lower intensity than the average intensity of the candidate regions, say, less than 80% of the average intensity, see Fig 3(b);
3) Judge the relative positions of the holes and ear line and determine the candidate region is a valid face or not.

5 Results and Discussions

In our experiments, statistical models of skin colors are estimated through histogram based approach using a subset of ECU database [5], in which 500 images are used for training. Afterwards, we generate 100 test images in the office environments for evaluation. Results on skin detection from both the training images and our own images are summarized in Table 1 below. Although the results from different color spaces are very comparable, HSV and YUV seem yield slightly better performance in linear and nonlinear color spaces, respectively. More results on skin and face detection are also given in Fig. 4, along with discussions in details.

Table 1. Skin detection results from different color spaces.

Results / Test data	Linear color space				Nonlinear color space			
	YUV		YCbCr		HSV		HSI	
	TPR	**FPR**	**skin**	bk	**skin**	bk	**skin**	bk
trained	**93.7%**	8.1%	**93.5%**	7.9%	**94.1%**	8.2	**93.9%**	8.3%
Non-trained	**91.2%**	10.2%	**91.1%**	9.9%	**93.0%**	10.1	**92.7%**	10.1%
Overall	**93.3%**	9.3%	**92.9%**	9.2%	**93.6%**	9.5	**93.5%**	9.6%



(a) Original image (b) YUV skins (c) HSV skins
(d) Final face detected

Fig. 4. Skin and face detection using image of Peter and Tommy.

As for skin detection, we can still find that skin regions detected from HSV space are more accurate and robust than that from YUV space, and the skin regions in background can also be detected easily in HSV spaces (see the face in Fig 1 and hand near the middle head in Fig 4, which means HSV space is less sensitive to variations of illumination intensity.

Thresholding by size and ratio is very effective in non-face regions removal. Moreover, the face model composed by the rules on the relative positions of ears and holes of eyes or mouth is also very practical in face detection, as ears can be found in almost every face image, which are more robust for detection even when the face is rotated and eyes are difficult to be detected.

Though our face detection algorithm can achieve quite satisfied results even there are pose variations, there are several additional strategies can be further applied for more robust face detection in our model, such as how to obtain the W/H ratio more accurately if there are connected skin regions and holes, and how to detect eyes and mouth if there are no holes can be found, especially for the face in the background.

With detected regions of skin and face, semantic indexing and retrieval of images are achieved as follows:

1) According to whether skin and face regions can be detected, all the images are automatically annotated as with or without skin/face regions respectively;
2) For those with skin or face regions, size and number of regions are also recorded;
3) For images with face regions, the estimated positions of ears, etc. are also taken in semantic indexing, which can be further used to estimate pose of faces;
4) Finally, these indexes are utilized in semantic retrieval of images.

6 Conclusions

By comparative study of skin detection from different color spaces, we find nonlinear color spaces, such as HSV, can obtain more accurate and robust skin results, especially in detecting of background faces. Moreover, we find the shape filtering and knowledge-based modeling very useful in face detection. Besides, these detected skin and face regions can be further utilized for semantic indexing and retrieval of images. How to improve quantitative analysis of the shape filters and face modeling for more accurate and robust face detection, especially on separation of connected faces and detection of background faces, will be investigated as the next step in the near further.

7 Acknowledgement

Finally, the authors wish to acknowledge the financial support under EU IST FP-6 Research Programme with the integrated project: LIVE (Contract No. IST-4-027312).

References

1. Hunke, M., Waibel, A.: Face Locating and Tracking for Human–Computer Interaction. IEEE Computer, (1996) 1277-1281
2. Cui, Y., Weng, J.: Appearance-Based Hand Gesture Sign Recognition from Intensity Image Sequences. Computer Vision and Image Understanding, vol. 78, (2000) 157-176
3. Forsyth, D., Fleck, M.: Automatic Detection of Human Nudes. Int. J. of Computer Vision, 32(1), (1999) 63–77
4. Wren, C. R., Azarbayejani, A., Darrel, T., Pentland, A. P.: Pfinder: Real-time Tracking of The Human Body. IEEE T-PAMI. 19(7), (1997) 780–785
5. Phung, S. L., Bouzerdoum, A., Chai, D.: Skin Segmentation Using Color Pixel Classification: Analysis and Comparison. IEEE T-PAMI. 27(1) (2005) 148-154.
6. Yang, M.-H., Kriegman, D., Ahuja, N.: Detecting Faces in Images: A Survey. IEEE T-PAMI, 24(1), (2002) 34–58.
7. Palus, H.: Representation of Color Images in Different Color Spaces. The Color Image Processing Handbook. S. J. Sangwine and R. E. N. Horne (eds), London (1998)
8. Jones, M. J., Rehg, J. M.: Statistical Color Models with Application to Skin Detection. Int. J. Computer Vision. 46(1) (2002) 81-96
9. Saber, E., Tekalp, A. M.: Frontal-View Face Detection and Facial Feature Extraction Using Color, Shape and Symmetry Based Cost Functions. Pattern Recognition Letters, 19(8), (1998) 669-680
10. Hsu, R.-L., Abdel-Mottaleb, M., Jain, A. K.: Face Detection in Color Images. IEEE T-PAMI. 24(5) (2002) 696-706
11. Garcia, C., Tziritas, G.: Face Detection Using Quantized Skin Color Regions Merging and Wavelet Packet Analysis. IEEE T-Multimedia. 1(3) (1999) 264-277
12. Wu, H., Chen, Q., Yachida, M.: Face Detection from Color Images Using a Fuzzy pattern Matching Model. IEEE T-PAMI. 21(6) (1999) 557-563
13. Kakumanu, P., Makrogiannis, S., Bourbakis, N.: A Survey of Skin-Color Modeling and Detection Methods. Pattern Recognition. 40 (2007) 1106-1122
14. Habili, N., Lim, C. C., Moini, A.: Segmentation of the Face and Hands in Sign language Video Sequences Using Color and Motion Cues. IEEE-TCSVT, 14(8) (2004) 1086-1097
15. Chai, D., Ngan, K. N.: Face Segmentation Using Skin-Color Map in Videophone Applications. IEEE. T-CSVT. 9(4) (1999) 551-564
16. Cho, K.-M., Jang, J.-H., Hong, K.-S.: Adaptive Skin-Color Filter. Pattern Recognition. 34 (2001) 1067-1073
17. Zheng, Q.-F., Gao, W.: Fast Adaptive Skin Detection in JPEG Images. Lecture Notes in Computer Science, Vol. 3768. (2005) 595-605
18. Zhu, Q., Cheng, K.-T., Wu, C.-T., Wu, Y.-L.: Adaptive Learning of an Accurate Skin-Color Model. (2004) 37-42
19. Zhang, M.-J., Gao, W.: An Adaptive Skin Color Detection Algorithm with Confusing Background Elimination. Proc. ICIP. II (2005) 390-393
20. Garcia, C., Tziritas, G.: Face Detection Using Quantized Skin Color Regions Merging and Wavelet Packet Analysis. IEEE T-Multimedia. 1(3) (1999) 264-277
21. Albiol, A., Torres, L., Delp, E. J.: Optimum Color Spaces for Skin Detection. Proc. ICIP. I (2001) 122-124

Fusion of Intensity and Channel Difference for Improved Colour Edge Detection

Jinchang Ren, Dong Wang, Jianmin Jiang and Stanley S. Ipson

School of Informatics, University of Bradford, BD7 1DP, United Kingdom
{J.Ren,dwang6,J.Jiang1,S.S.Ipson}@bradford.ac.uk

Keywords: edge detection, image segmentation, colour space transform, colour channel difference.

Abstract

Edge detection, especially from colour images, plays very important roles in many applications for image analysis, segmentation and recognition. In this paper, a new colour-gray mapping method for effective colour edge detection is proposed. From any given colour image *C*, a gray image *D* is defined as the accumulative differences between each of its two colour channels, and another gray image *R* is then obtained by weighting of *D* and gray intensity image *G*. Fusion of edges extracted from *R* and *G* forms the final results. Comparing with edges detected from traditional colour spaces like RGB, YCbCr and HSV, all using same Canny operator, it seems the proposed method can achieve more effective results from different test images.

1 Introduction

Since physical edges usually correspond to apparent variations in the illumination and colors, edge detection is very useful and important in many low-level vision applications as to provide essential visual information for feature extraction, segmentation and scene understanding [1, 5, 7, 10, 13]. In general, edge detection usually has three main stages, namely preprocessing or smoothing, image difference and gradient detection for edge pixel judgment, and continuous edge extraction. Gradient-based methods are almost the earliest edge detector which only uses convolution templates to obtain local difference for edge detection, and then Canny introduced a well defined edge detector with good performance, high precision and unique response [1].

From convolution templates to Canny edge detector, traditional edge detection methods are usually defined on grey images, and some improvements or new methods are necessary for edge detection from colour images according to human colour perceptions. A simple idea is to convert <r,g,b> colour image to its luminance intensity image G, from which traditional edge detectors are applied to extract colour edges. As the conversion from colour to grey is multiple to one mapping, edges detected from G-image are less accurate and usually edge pixels with obvious colour difference but less intensity variation are missing.

Another simple idea for colour edge detection is to apply the edge detectors to each colour channel and the final edges will be the combined results of the edges from different channel images. Although the combined edges have more accuracy and detail information than the edges from the intensity image, they are still not accurate enough for effective object detection and image segmentation because of missing of certain edges.

Colour vision is a synthetic perception of R, G and B channels, people can determine edge pixels easily from 3D <r,g,b> space. The combined edges extracted from multiple single channels have intrinsic limitations according to human visual perceptions. To acquire more reasonable and accurate edges, people have tried a lot of colour spaces and colour models [3, 8, 14], such as HLS (hue-lightness-saturation), HSV (hue-saturation-value), YUV, XYZ, YC_bC_r, etc. When <r,g,b> colour images are converted to specified colour space, edges will be extracted from the components of the new space. Since different components are independent each other, the final edges are also combination of the edges from each component including colour and luminance information [4]. Alternatively, some other efforts have been make on edge detection from colour images, such as the compass operator in [9], direction information measurement in [6], cluster analysis in [11], and invariance analysis [4, 12]. However, choosing a suitable colour space is still a very fundamental task in such a context on which the above operators or processing can then be applied.

In this paper, we present a colour edge detection method based on a new colour space transform. At first, amplitude differences between different two colour channels are accumulated to generate a grey D-image. Then, another grey R-image is obtained by weighting of D-image and luminance intensity image G. Final edges will be combination of edges extracted from the R-image and G-image. Compared with edges from three colour channels and YC_bC_r model, the proposed method can achieve more effective and robust edges from different standard colour images.

2 Colour Space Transform

Comparing to texture and shape, colour is the chief discrimination attribute in human visual system [2, 4]. Therefore, colour edge detection is more important in scene analysis and understanding. Although many colour transforms and colour space models have been developed, they can be converted between each other by mapping from and to RGB space. In general, luminance and the differences or proportions between luminance and different colour components from RGB space compose the new components in the transformed space. Following is an example transform from RGB space to YUV space:

$$y = w_r r + w_g g + w_b b \qquad (1)$$

$$u = b - y \quad and \quad v = r - y \qquad (2)$$

In (1) and (2), y is the luminance and over all measurement of the three colour channels, and u, v are the chrominance and detail representation by colour channel differences.

YIQ and YC_bC_b have similar transforms like YUV in (1), and HSV and HLS spaces have more complex transform formulas [8]. Although some of these transforms can achieve coherent distance measurement with human perceptions, they are not effective in accurate edge detection, because they have not taken full consideration of the colour differences between different colour channels.

Suppose f is the original 3 channel colour image (see Fig. 1), and we define the one channel D-image as:

$$\begin{aligned} D(i,j) = \omega_1 |r(i,j) - g(i,j)| + \omega_2 |r(i,j) - b(i,j)| \\ + \omega_3 |b(i,j) - g(i,j)| \end{aligned} \qquad (3)$$

where D-image gives the total colour differences between different colour channels. Currently we simply have same weights for three channels.

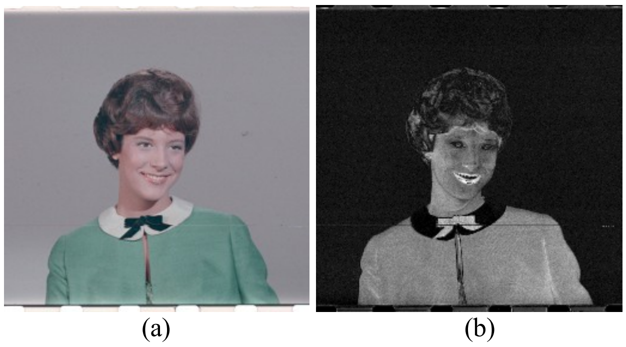

(a) (b)

Fig. 1. Original colour image (left) and its D-image (right).

For grey pixels in image f, they have same values in three colour channels, and we cannot distinguish them from D-image. Therefore, another one channel image R is obtained by weighting of D image and luminance intensity image G as follows:

$$R(i,j) = k \frac{w_d G(i,j) + w_g D(i,j)}{w_d + w_g} \qquad (4)$$

where w_d and w_g are the weights and determined by the statistical properties of D-image and G-image given in (5) and (6), and parameter k is taken to scale the final image within 256 grey levels.

$$w_d = 1.5 * Range(D) + \sigma(D) \qquad (5)$$

$$w_g = 1.5 * Range(G) + \sigma(G) \qquad (6)$$

$Range$ is a function to get the intensity range of the given image, and σ is the standard derivation. Generally, we have $Range(G) \leq 255$ but $Range(D) \geq 255$. Therefore, (4) is used to make the weighted values, $w_d G(i,j)$ and $w_g D(i,j)$, more comparable to get more robust edges.

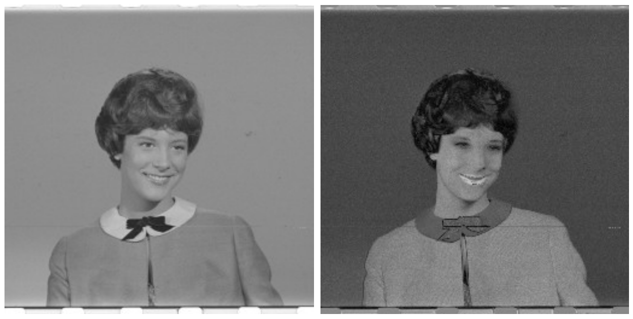


Fig. 2. *G*-image (left) and *R*-image (right) of the colour image in Fig. 1(a).

3 Colour Edge Detection

Canny operator is a typical edge detection method with good performance. Taken $f(x)$ and $[-\omega,\omega]$ as impulse response and bandwidth of Canny operator, the corresponded filter should make formula (7) maximum:

$$\sum = \frac{\left|\int_{-\omega}^{0} f(x)dx\right|}{\sqrt{\int_{-\omega}^{\omega} f^2(x)dx}} \cdot \frac{|f'(0)|}{\sqrt{\int_{-\omega}^{\omega} f'^2(x)dx}} \qquad (7)$$

Edges detected by Canny operator are all local extremums to ensure the detection precision. Firstly, the image is smoothed by a Gauss function. Then, normalized gradient image is obtained from the smoothed image to detect edge pixels. Two thresholds T_h and T_l are used to get more continuous and robust results. If a pixel has a gradient more than T_h, it belongs to the edge. If the gradient is less than T_l, it is not edge pixels. Otherwise, the pixel will be carefully examined and determined as edge pixels if there are edge pixels in its neighbourhood and the determination helps to find more continuous edges.

T_h and T_l are two very important parameters in Canny operator, because different thresholds will lead to quite different edge results. For the G-image in Fig 2, we give the detected edges using different thresholds in Fig 3. For a given

T_h, small T_l can achieve more detail edges, but too small T_l may cause noise (see the girl's face in Fig 3). Therefore, how to automatic select suitable thresholds is a basic problem for effective edge detection of Canny operator.

In our experiments, we use the OpenCV package for Canny edge detection. Assume μ and σ are the mean and standard derivation of a given gray image g, then we determine T_h and T_l as follows to cope with some adaptivity:

$$T_h = \mu + \max(\mu/2, \sigma) \quad (8)$$

$$T_l = (\mu - \sigma)/2 \quad (9)$$

For G-image in Fig. 2, it has $\mu = 138.97$ and $\sigma = 28.75$, then we can find $T_h = 208$ and $T_l = 55$, and the detected edges is given in Fig 3b, which is better than Fig 3a and Fig 3c.

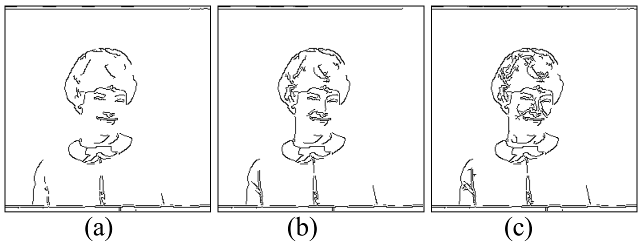


Fig. 3. Extracted edges using Canny with $T_h = 208$ and T_l changes from 100, 55 to 10, respectively.

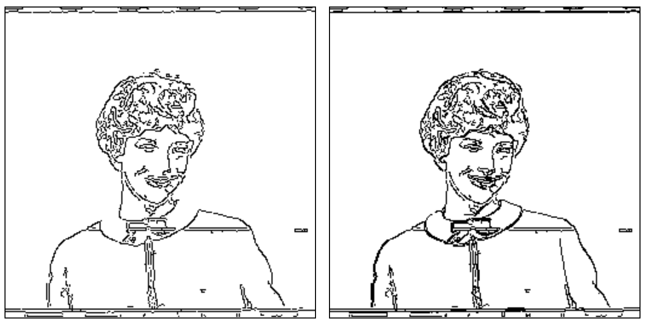


Fig. 4. E_R (left) and E_{final} (right) of colour image in Fig 1.

For a given colour image, we can firstly get its G-image and R-image. Then, edges E_G and E_R can be extracted from G-image and R-image using the adaptive Canny given above, respectively. The final edges E_{final} can be obtained by combination of edges E_G and E_R as:

$$E_{final} = E_R \cup E_G \quad (10)$$

As for the colour image in Fig 1 and E_G in Fig 3, we give its E_R and E_{final} in Fig 4. Note that E_G is useful to recover edges from grey part (no channel difference) of images.

4 Results and Discussions

In order to evaluate our method, we take edges from RGB and YC_bC_r space for comparison. The reason for choosing YC_bC_r space is that YC_bC_r is very close to human colour perceptions and has direct applications in image and video representation, compression, analysis and segmentation.

Edges from RGB space are acquired as follows: Firstly, we extract edges from single channels of R, G and B by self-adaptive Canny; then, the final edge image will be combination of the three channels edges.

Edges from YC_bC_r space is extracted similarly with the edges from RGB space, which means the final edges are also combination of edges from each of the components. After transform from RGB to YUV (see section 2), C_b and C_r can be further determined by:

$$C_b = (U+1)/2\,,\ C_r = round(V/1.6) \quad (11)$$

Finally, we linearly normalize the range of Y, C_b and C_r within 0 to 255 before edge detection.

In Fig 5, we give edge comparisons from RGB, YC_bC_r and combined results by our colour-grey mapping, and the original colour image is shown in Fig 1. All the edges are extracted by the adaptive Canny in section 3, and edges from multiple colour channels will be combined to obtain the final edge results. From Fig 5 we can see, our method has more continuous (see closed face contour) and accurate edges than edges from RGB and YC_bC_r space, and edges from RGB is much better than those from YC_bC_r space.

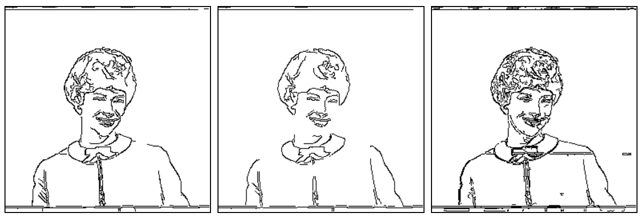


Fig. 5. Edges extracted from RGB space (left), YC_bC_r and our method (right).

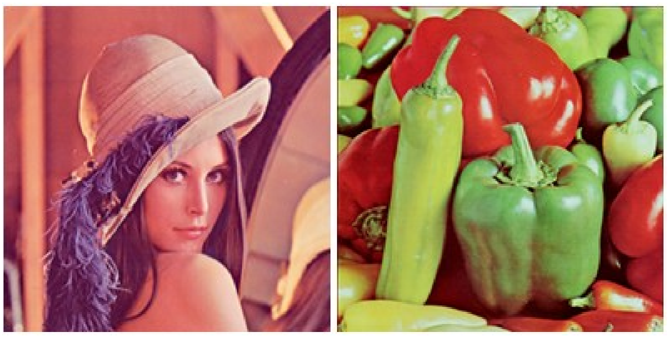


Fig. 6. Two well-known test images (lena & pepper).

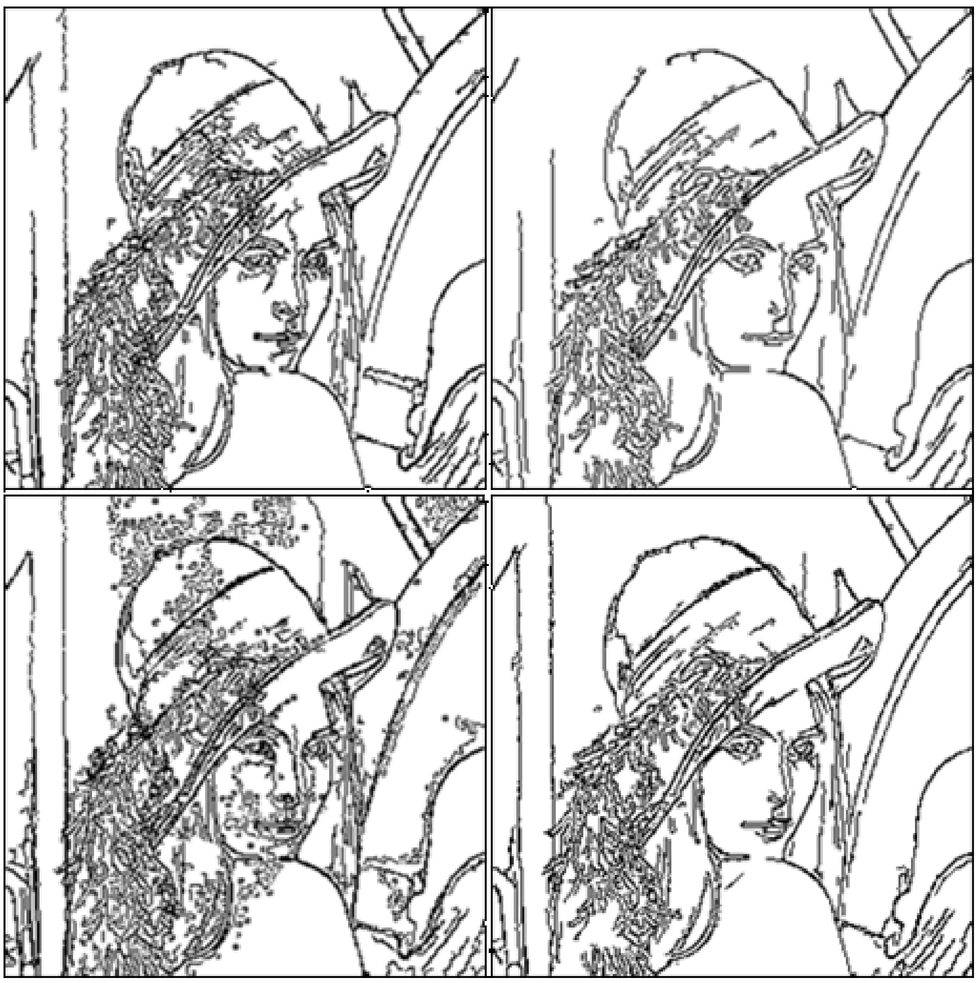


Fig. 7. Edges extracted from colour spaces of RGB (top left), YCbCr (top right), HSV (bottom left) and our method (bottom right) using the well-known lena image.

In order to evaluate our method, two well-known test images, the lena and the pepper(as showed in Fig 6), are adopted, and the comparative results extracted from RGB, YCbCr, HSV and our method are given in Fig 7 and Fig 8, respectively.

In Fig 5 and Fig 7, our method can produce much better results, though few edges are found missing in Fig 8. From Fig 7 and Fig 8 we can see, edges from YCbCr spaces have least detail compared with edges from other spaces. Although edge images from HSV space have more detail, they also contain substantial noise. Therefore, edges from RGB space and our method are more comparable.

In Fig 7, our method produces more continuous and meaningful edges (see lines in the left of the image, and also edges in Lena's face). However, we lost part of edges of the big green pepper in Fig 8 due to limited channel difference and low intensity, though our result has less noise than the edges from RGB space. Hence, we can say, our method has less noise and comparable edge results with those from RGB space.

5 Conclusions

A new colour-gray mapping is proposed for effective edge detection from colour images. Compared with those results from RGB and other typical colour spaces, it seems combined edges extracted from the mapped image can well compensate the edges from the intensity image, especially when the colour distribution is limited, for more effective and robust colour edge detection. The proposed scheme on colour to grey transform and colour edge detection will be further investigated on image segmentation and content-based retrieval applications.

Acknowledgements

The original colour image in Fig 1 comes from web (http://sipi.usc.edu/services/database/Database.html). Finally, the authors wish to acknowledge the financial support under EU IST FP-6 Research Programme with the integrated project: LIVE (Contract No. IST-4-027312).

References

[1] J. F. Canny: "A computational approach to edge detection", *IEEE T-PAMI*, **8(6)**, pp. 679-698, (1986).

[2] H.-G. Choi, W.-C. Jung, J. Min, W. H. Lee, J.-W. Choi: "Colour image detection by biomolecular photoreceptor using bacteriorhodopsin-based complex LB films", *Biosensors and Bioelectronics*, **16(9)**, pp. 925-935, (2001).

[3] A. Fotinos, G. Economou, S. Fotopoulos: "Use of relative entropy in colour edge detection", *Electronics Letters*, **35(18)**, pp. 1532-1534, (1999).

[4] J.M. Geusebroek, R. van den Boomgaard, A.W.M. Smeulders, H. Geerts: "Colour invariance", *IEEE T-PAMI*, **23(12)**, pp. 1338-1350, (2001).

[5] T. Gevers, H. M. G. Stokman: "Classification of colour edges in video into shadow-geometry, highlight, or material transitions", *IEEE T-Multimedia*, **5(2)**, pp. 237-243, (2003).

[6] L.-Q. Niu, W.-J. Li: "Colour edge detection based on direction information measure", In: *Proc. WCICA*, **2**, pp. 9533-9536, (2006).

[7]. N. R. Pal, S. K. Pal: "A review on image segmentation techniques", *Pattern Recognition*, **26(9)**, pp. 1277-1294 (1993).

[8] H. Palus: "Representation of colour images in different colour spaces", *The Colour Image Processing Handbook*, Sangwine S. J. and Horne R. E. N. (eds), Chapman and Hall, London (1998).

[9] M. A. Ruzon, C. Tomasi: "Colour edge detection with the compass operator", In: *Proc. CVPR*, pp. 160-166, (1999).

[10] H.M.G. Stokman, Th. Gevers: "Automatic thresholding and classification of colour edges", *J. Electronic Imaging*, **9**, (2000).

[11] H. Tao, T. S. Huang: "Colour image edge detection using cluster analysis", In: *Proc. ICIP*, **1**, pp. 834-837, (1997).

[12] J. van de Weijer, T. Gevers, J.M. Geusebroek: "Colour edge detection by photometric quasi-invariants", In: *Proc. IJCV*, **5(2)**, pp. 1520-1526, (2003).

[13] J. van de Weijer, T. Gevers, J.M. Geusebroek: "Edge and corner detection by photometric quasi-invariants", *IEEE T-PAMI*, **27(4)**, pp. 625-630, (2005).

[14] S.-Y. Zhu, K. N. Plataniotis, A. N. Venetsanopoulos: "Comprehensive analysis of edge detection in colour image processing", *J. Optical Engineering*, **38(4)**, pp. 612-625, (1999).

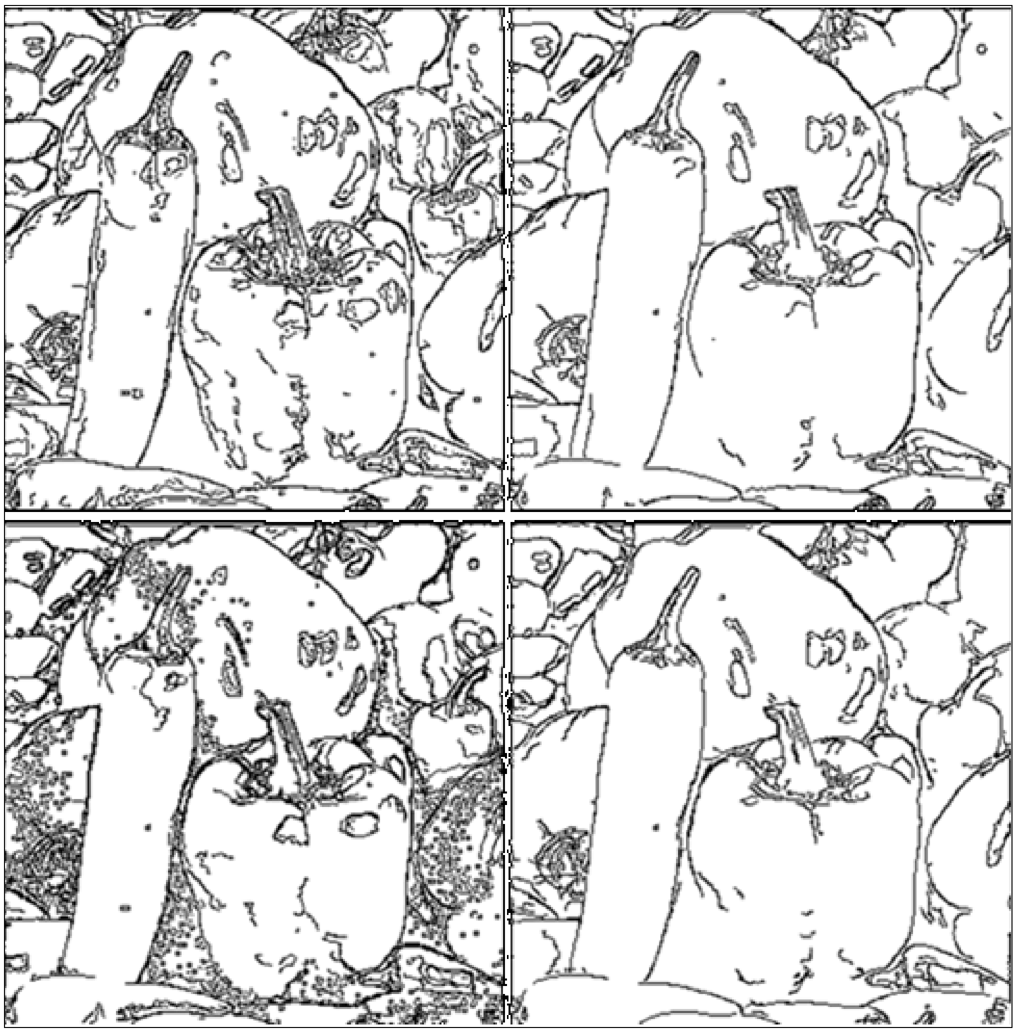


Fig. 8. Edges from colours spaces of RGB (top left), YCbCr (top right), HSV (bottom left) and our method (bottom right) from the pepper image.

RECOGNITION OF HAND GESTURE BASED ON GAUSSIAN MIXTURE MODEL

Jia Jia, Jianmin Jiang, Dong Wang

EIMC, School of Informatics, University of Bradford, West Yorkshire BD7 1DP, UK
jjia@bradford.ac.uk, j.jiang1@bradford.ac.uk, D.Wang6@brad.ac.uk

ABSTRACT

This paper presents a new method for gesture recognition of Human beings' hand. This method integrates the features of shape, color and orientation histograms, which are extracted from images, and estimate the comparability with all the different types of gestures by a proposed Expectation-Maximization algorithm in Gaussian Mixture Model. The classification results were presented based on the values of likelihood compared with all the types of pre-assigned images, and the performance of this approach in an experiment is shown that the proposed method works well.

1. INTRODUCTION

Human gesture has its specific meanings and is widely used for communications between deaf people. It is considered as a very important function in many practical communication applications. Recently, hand gesture recognition has gained a lot of interests, which plays a crucial role in a wide range of applications, such as automatic sign language understanding, entertainment, and human computer interaction (HCI). Because hand gestures are natural and intuitive in providing rich information to computers without extra cumbersome devices, they can offer a great potential for next generation user interfaces, being especially suitable for large scale displays, 3D volumetric displays or wearable devices.

In addition, there has recently been a growing interest in gesture-recognition systems by a number of researchers, providing some novel approaches since early nineties. Chaitanya Gurrapu [1] adopts GMM to classify the human body's gestures and track its changes through time using HMM. The key features extracted from body images are polygonal vertices obtained from body shapes. Accordingly, GMM is trained on the vertices feature and the relationship between vertices which is represented in the form of gradient of the line joining two vertices. The final accuracy is close to 98%. Yang Liu [2] used a method integrating shape and depth information for robust hand tracking. Shape is the primary measurement which builds an important function describing areas of state-space and contains critical information about the posterior. Recognition rate is between 70% and 92% under various numbers of samples. Sebastien Marcel [3] presented a hand gesture recognition algorithm based on input/output Hidden Markov Models. This approach achieves a recognition rate between 90% and 100% of the sequence.

In our system, we addressed the core issue of gesture recognition in extracting robust features, leading to a more accurate estimation. The new approach we propose is different from existing efforts reported in the literature. It focuses on estimating the gesture contained in an image by analyzing different complex features including shape, color and orientation histogram quantized in Gaussian Mixture Model (GMM).

GMM is a widely used statistical model in many applications of pattern recognition, which is often regarded as a versatile modeling tool as it can be used to approximate any probability density function (PDF) given a sufficient number of components, and impose only minimal assumptions about the modeled random variables. The advantage is including a rigorous statistical basis, the possibility of encoding spatial, color, texture and motion features in a unified system, and the ability to trade off accuracy of representation against data volume. Due to such advantages, our proposed technique builds upon the GMM to estimate the mutative meaning of human gestures in a compact and precise manner.

The rest of this paper is organized as follows. An overview of principle of Gaussian Mixture Model is described in Section 2. The method of training and parameter estimation is covered in Section 3. While Section 4 is dedicated to description of how images are tested, Section 5 is devoted to the experiments. Finally, concluding remarks and further area of research are provided in Section 6.

2. DESCRIPTION OF PRINCIPLE OF GAUSSIAN MIXTURE MODEL (GMM)

GMM is one of the most widely used mixture modeling techniques. It's a simple model and is reasonably accurate when data are generated from a set of Gaussian distributions [4, 5]. Let $X_i = \{x_t, 1 \le t \le T^i\}$, denote the feature vectors for data points from the *i*-th class. They are modeled by a total number of J Gaussians as follows:

$$P\left(X_i \middle| \theta^i_{GMM}\right) = \prod_{t=1}^{T^i} \sum_{j=1}^{J} P(z_j) P_{z_j}\left(x_t \middle| u_j, \Sigma_j\right) \quad (1)$$

Where θ^i_{GMM} includes all the model parameters, i.e. $\{P(z_j), \mu_j, \Sigma_j, 1 \le j \le J\}$. $P_{z_j}\left(x_t \middle| \mu_j, \Sigma_j\right)$ is the Gaussian distribution for the *j*-th class, with a mean vector μ_j and a covariance matrix $\sum_j$ as:

$$P_{z_j}\left(x_t \middle| \mu_j, \Sigma_j\right) = \frac{1}{(2\pi)^{D/2} \left|\Sigma_j\right|^{1/2}} \exp\left\{-\frac{1}{2}\left(x_t - \mu_j\right)^T \sum\nolimits_j^{-1}\left(x_t - \mu_j\right)\right\} \quad (2)$$

where D is the dimension of the feature vector x_t. Usually, Σ_j is set to be a diagonal matrix as $diag\{\sigma^2_{jd} : 1 \le d \le D\}$ in order to reduce the size of parameter space.

It can be seen from Equation (1) that the data points of a specific class are generated from multiple Gaussian models with an identical weight $P(z_j)$. We define $\omega_j = P(z_j)$.

In other words, an integrated Gaussian mixture model contains three basic parameters: Mixture weight, Mean vector and Covariance matrix, which can be represented as:

$$\lambda = \{\omega_j, \mu_j, \Sigma_j\} \quad (3)$$

where ω_j is the mixture weight, μ_j is the mean vector, and Σ_j is the covariance matrix. We use λ to stand for every single image. Additionally, we use $b_j(x) = P_{z_j}\left(x_t \middle| \mu_j, \Sigma_j\right)$ and $\sum_{j=1}^{J} \omega_j = 1$.

3. TRAINING METHODS AND PARAMETER ESTIMATION

For training purposes, our primary work is to find the parameter λ which can stand for the feature vector of every certain image. A normal method is the maximum likelihood (ML) estimation via expectation-maximization algorithm [4, 6, 7]. The ML means attempting to find the certain λ from the image which is used for training purposes in order to get the maximum likelihood.

For example, we extract the feature vectors $X = \{x_1, \cdots, x_T\}$ from an image by selecting the feature where the T is the number of features and the likelihood of GMM is defined as:

$$P\left(X \middle| \lambda\right) = \prod_{t=1}^{T} p\left(x_t \middle| \lambda\right), \quad (4)$$

According to the fact that the $P\left(X \middle| \lambda\right)$ is nonlinear function, we should use the way of ML to estimate the parameter of GMM until the $P\left(X \middle| \lambda\right)$ is convergent.

The method of algorithm estimation starts from an initial guess λ for the new model parameters $\overline{\lambda}$ to be estimated, in order to get a relationship of $P\left(X \middle| \overline{\lambda}\right) \ge P\left(X \middle| \lambda\right)$. Then transform the $\overline{\lambda}$ into the initial model parameter λ . This step will be repeated until $P\left(X \middle| \lambda\right)$ is convergent. During the iteration, the following estimation ensures that the approximation of GMM is achieved via the nature of monotonic increase:

$$p\left(i \middle| x_i, \lambda\right) = \frac{\omega_i b_i(x_t)}{\sum_{k=1}^{T} \omega_k b_k(x_k)} \quad (5)$$

Where the estimation of mixture weight is

$$\omega_i = \frac{1}{T} \sum_{t=1}^{T} p\left(i \middle| x_t, \lambda\right) \quad (6)$$

The estimation of mean vector is:

$$\mu_i = \frac{\sum_{t=1}^{T} p\left(i \middle| x_t, \lambda\right) x_t}{\sum_{t=1}^{T} p\left(i \middle| x_t, \lambda\right)} \quad (7)$$

The estimation of covariance is:

$$\Sigma_i^2 = \frac{\sum_{t=1}^{T} p\left(i \middle| x_t, \lambda\right) x_t^2}{\sum_{t=1}^{T} p\left(i \middle| x_t, \lambda\right)} - \mu^2 \quad (8)$$

4. METHODS OF TESTING

We use the maximum of a posterior criterion to differentiate all images, which means that the likelihood between testing images and pre-assigned images of each different type are calculated in order to compare the results and select the maximum numerical value. Accordingly, the testing image is ranged to a certain type, in which it has the maximum numerical value of likelihood compared with other images. In this way, we can use the equation below to describe the proposed process:

$$\hat{S} = \arg \max_{1 \le k \le S} \Pr(\lambda_k | X) \quad (9)$$

where S is the total of all pre-assigned different types, $\hat{S}$ is the certain type which the testing image is classified to, λ_k is the model of pre-assigned type *K*, and *X* is the vector of features of the testing image.

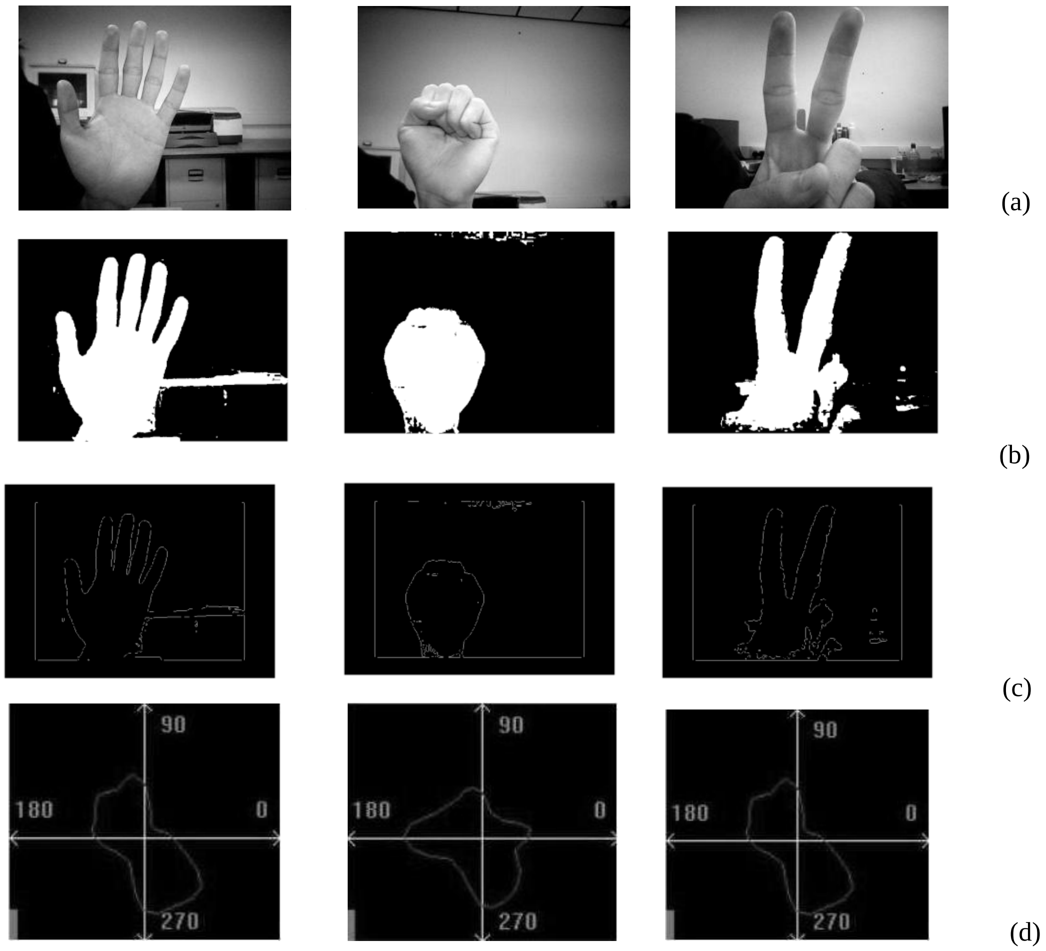


Fig.1. Classification of Hand Gesture: (a) Three basic classes of hand gesture; (b) colour extraction of each class of hand gesture; (c) shape extraction of each class of hand gesture; (d) orientation histogram of each class of hand gesture

This equation can be transformed to another by Bayesian rule:

$$\hat{S} = \arg\max_{1\le k\le S} \frac{p(X|\lambda_k)\Pr(\lambda_k)}{P(X)} \qquad (10)$$

for $\Pr(\lambda_k) = 1/S$, we have: $\hat{S} = \arg\max_{1\le k\le S} P(X|\lambda_k)$. By calculating their logarithms, we have:

$$\hat{S} = \arg\max_{1\le k\le S} \sum_{t=1}^{T} \log p(x_t|\lambda_k) \qquad (11)$$

5. EXPERIMENTS

To evaluate the proposed algorithm, we collected a group of Hand-gestures as the test data set, and then classified all of them into 3 basic classes as shown in Fig. 1 (a). The first one is a hand with all fingers outstretched. The second one is considered as a fist, and the third one involves only 2 outstretched fingers (forefinger and middle finger) symbolizing a victory.

Following the described algorithm, three significant features are extracted, which include color, shape, and orientation histogram from all the image. The feature of color is extracted in the YCbCr Color Space. Fig.1 (b) shows the results of the color extraction of every image in Fig.1. The feature of shape is extracted by Canny Edge Detector. The results are shown in Fig.1 (c). Orientation histogram [8, 9] is one of useful features which offers robustness to lighting changes and give translational invariance. The orientation histograms for each gesture are illustrated in Fig.1 (d).

We evaluate the performance by using a total number of 450 gesture images, which were derived from the "Sebastien Marcel Static Hand Posture Database" [10]. As these images are captured against different backgrounds, it would help to test the robustness of the proposed algorithm if the order of images is randomized in the data set. Trained by 8 images from the database for each gesture, the proposed technique is evaluated by extensive experiments and their results are measured by error rate, which are summarized in Table-I:

Table-I: Summary of Experimental Results

Total Number	450
Images for Training	24
Amount of testing images	426
Error rate (%)	5.37%

6. CONCLUSION AND SUMMARY

In this paper, we described a proposed algorithm for human gesture recognition and demonstrated its discriminative ability for recognition of gestures on a large database of images. By using Gaussian Mixture Model, we have shown that multiple features extracted from gesture images could be organized and controlled by GMM to formulate new discriminating vector for classification and recognition of human gestures. The application of Gaussian Mixture Model illustrates the advantage that it provides improved performance over other existing methods, yet requiring only modest computational cost to complete the gesture recognition. Further research can be identified to focus on the issue of extendibility and selection of primary features as such that other pattern recognitions can be achieved, especially inside digital videos.

7. ACKNOWLESGMENT

The GMM code used here is based in part on Netlab Toolbox, and the work is part of the LIVE project initiatives funded under FP6 programme.

8. REFERENCES

[1] Chaitanya Gurrapu and Vinod Chandran, "Gesture Classification Using A GMM Front End and Hidden Markov Models", Proceedings of the 3rd IASTED International Conference on Visualization, Imaging, And Image Processing, Spain, pp. 609-612, September, 2003.

[2] Yang Liu and Yunde Jia, "A Robust Hand Tracking and Gesture Recognition Method for Wearable Visual Interfaces and Its Applications", Proceedings of the Third International Conference on Image and Graphics, IEEE Computer Society, USA, pp. 472-475, Dec 2004.

[3] Sebastien Marcel, Oliver Bernier, Jean-Emmanuel Viallet and Daniel Collobert, "Hand Gesture Recognition using Input-Output Hidden Markov Models", Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition, IEEE Computer Society, USA, pp. 456-461, 2000.

[4] C.M.Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[5] R.O.Duda, P.E.Hart, and D.G.Stork, *Pattern Classification*, volume 2, Wiley-Interscience Publication, 2000.

[6] A.P.Dempster, N.M.Laird and D.B.Rubin, "Maximum likelihood from incomplete data via EM algorithm", Journal of the Royal Statistical Society, Series B (Methodological), Vol. 39, No. 1, pp. 1-38, 1977.

[7] J.Bilmes, "A Gentle Tutorial of the EM Algorithm and Its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models", Technical Report ICSI-TR-97-021, University of California Berkeley, 1998.

[8] William T. Freeman and Michal Roth, "Orientation Histograms for Hand Gesture Recognition", IEEE International Workshop on Automatic Face and Gesture Recognition, IEEE Computer Society, Switzerland, pp. 296-301, June 1995.

[9] Hyung-Ji Lee and Jae-Ho Chung, "Hand gesture recognition using orientation histogram", Tencon 99. Proceedings of the IEEE Region 10 Conference, South Korea, pp. 1355-1358 Vol. 2, Dec 1999.

[10] S. Marcel, "Hand posture recognition in a body-face centered space", Proceedings of the Conference on Human Factors in Computer Systems, ACM, USA, pp. 302-303, 1999.

Fusion of Intensity and Inter-component Chromatic Difference for Effective and Robust Colour Edge Detection

Jinchang Ren, Dong Wang, Jianmin Jiang and Stanley S. Ipson

School of Informatics, University of Bradford, BD7 1DP, United Kingdom
{J.Ren,J.Jiang1,dwang6,S.S.Ipson}@bradford.ac.uk

Abstract

Edge detection, especially from colour images, plays very important roles in many applications for image analysis, segmentation and recognition. Most existing methods extract colour edges via fusing edges detected from each colour components or detecting from the intensity image where inter-component information is ignored. In this paper, an improved method on colour edge detection is proposed in which the significant advantage is the use of inter-component difference information for effective colour edge detection. For any given colour image *C*, a grey *D*-image is defined as the accumulative differences between each of its two colour components, and another grey *R*-image is then obtained by weighting of *D*-image and the grey intensity image *G*. The final edges are determined through fusion of edges extracted from *R*-image and *G*-image. Quantitative evaluations under various levels of Gaussian noise are achieved for further comparisons. Comprehensive results from different test images have proved that our approach outperforms edges detected from traditional colour spaces like RGB, YCbCr and HSV in terms of effectiveness and robustness.

Keywords: edge detection, image segmentation, colour space transform, inter-component difference.

1 Introduction

Since physical edges usually correspond to apparent variations in the illumination and colours, edge detection is very useful and important in many low-level vision applications as to provide essential visual information for feature extraction, segmentation and scene understanding [1, 5, 7, 10, 13]. In general, edge detection contains three main stages, namely preprocessing or smoothing, image difference and gradient detection for edge pixel judgment, and continuous edge extraction. Gradient-based methods are almost the earliest edge detector which only uses convolution templates to obtain local difference for edge detection, and then Canny introduced a well defined edge detector with good performance, high precision and unique response [1].

From convolution templates to Canny edge detector, traditional edge detection methods are usually defined on grey images, and some improvements or new methods are necessary for edge detection from colour images according to human colour perceptions. A simple idea is to convert <r,g,b> colour image to its luminance intensity image G, from which traditional edge detectors are applied to extract colour edges. As the conversion from colour to grey is multiple to one mapping, edges detected from G-image are less accurate and usually edge pixels with obvious colour difference but less intensity variation are missing. Another simple idea for colour edge detection is to apply the edge detectors to each colour component and the final edges will be the combined results of the edges from different component images. Although the combined edges have more accuracy and detail information than the edges from G-image, they are still not accurate enough and have missing edges due to the fact that inter-component information is ignored in the process of edge detection.

Since colour vision is synthetic perception of R, G and B components, the combined edges extracted from multiple single components have intrinsic limitations according to human visual perception. To acquire more reasonable and accurate edges, quite a few colour spaces and colour models have been investigated [3, 8, 14], such as HLS (hue-lightness-saturation), HSV (hue-saturation-value), YUV, XYZ, YC_bC_r, etc. When <r,g,b> colour images are converted to specified colour space, edges will be extracted from the components of the new space. Since different components are independent each other, the final edges are also combination of the edges from each component including colour and luminance information [4]. Alternatively, some other efforts have been made on edge detection from colour images, such as the compass operator in [9], direction information measurement in [6], cluster analysis in [11], and invariance analysis [4, 12]. However, choosing a suitable colour space is still a very fundamental task in such a context on which the above operators or processing can then be applied [16]. In addition, quite a few combined approaches have been proposed for colour edge detection, such as morphological gradient followed by outlier rejection [17], statistical analysis of R-G and B-Y colour components [18], clustering of pixels using the minimal spanning tree [19], combination of self-organising map (SOM) and a grayscale edge detector [20], and neighbourhood hypergraph and validation of hyperedge [10].

In this paper, we present an improved method for colour edge detection via fusion of intensity and chromatic difference. At first, amplitude differences between different two colour components are accumulated to generate a grey D-image. Then, another grey R-image is obtained by weighting of D-image and G-image of luminance intensity. Final edges will be combination of edges extracted from the R-image and G-image. In comparison with the edges extracted from three colour components and other colour spaces, the proposed method can achieve more effective and robust results from different standard colour images even with attached Gaussian noise.

2 Colour Space Transform

Comparing to texture and shape, colour is the chief discrimination attribute in human visual system [2, 4]. Therefore, colour edge detection is more important in scene analysis and understanding. Although many colour transforms and colour space models have been developed, they can be converted between each other by mapping from and to RGB space. In general, luminance and the differences or proportions between luminance and different colour components from RGB space compose the new components in the transformed space. Following is an example transform from RGB space to YUV space:

$$y = w_r r + w_g g + w_b b \tag{1}$$

$$u = b - y, \quad v = r - y \tag{2}$$

In (1) and (2), y refers to the luminance as an overall measurement of the three colour components, and u, v are the chrominance and detail representation by colour component differences.

YIQ and YC$_b$C$_b$ have similar transforms like YUV above, and HSV and HLS spaces have more complex transform formulas [8]. Although some of these transforms can achieve coherent distance measurement with human perceptions, they are not effective in accurate edge detection due to the fact that inter-component information has not been fully considered.

Let f be the original three component colour image, and we define the one component D-image as:

$$D(i,j) = \omega_1 |r(i,j) - g(i,j)| + \omega_2 |r(i,j) - b(i,j)| + \omega_3 |b(i,j) - g(i,j)| \tag{3}$$

where D-image gives the total colour differences between different colour components. Currently we simply set same weight of one-third for three components.

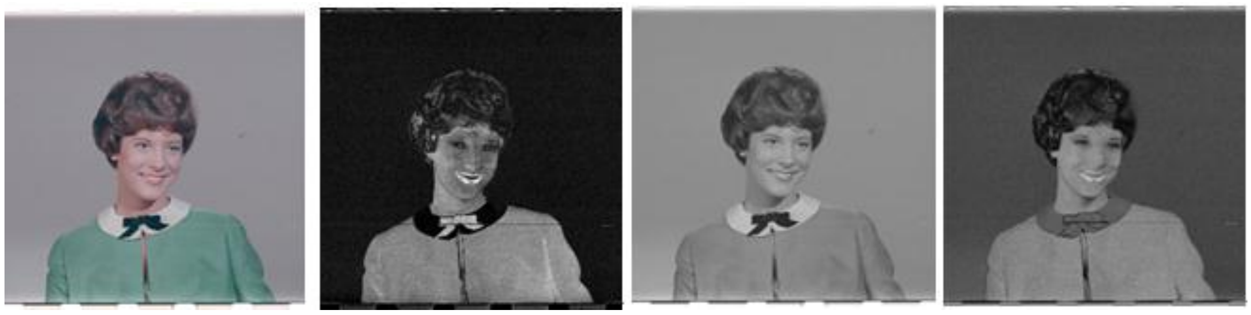


Figure 1. One original colour image (left) and its corresponding three single channel images including D-image, *G*-image and *R*-image, respectively.

Since the D-image has too much noise, it is smoothed by weighting with G-image of luminance intensity to generate a new one component R-image as follows:

$$R(i,j) = k \frac{w_d G(i,j) + w_g D(i,j)}{w_d + w_g} \tag{4}$$

where w_d and w_g are the weights and determined in (5) and (6) by the statistical properties of D-image and G-image, and the parameter k is used to scale the determined values of R-image within [0,255].

$$w_d = 1.5 * Range(D) + \sigma(D) \tag{5}$$

$$w_g = 1.5 * Range(G) + \sigma(G) \tag{6}$$

$Range$ is a function to get the intensity range of the given image, i.e. difference of the maximum and the minimum intensity values, and σ is the standard derivation. In comparison with [15], our solution can obtain a new pseudo-grey image without PCA analysis. For a given colour image, its corresponding D-image, *G*-image and *R*-image are all illustrated in Fig. 1 for comparisons.

3 Colour Edge Detection

Since our main focus is the fusion scheme for improved colour edge detection, we intend to use standard edge detectors for consistent measurements and evaluations. To this end, the well-known Canny operator is used for its relative good performance. Taken $f(x)$ and $[-\omega, \omega]$ as impulse response and bandwidth of the Canny operator, the corresponded filter should make formula (7) maximum:

$$\sum = \frac{\left|\int_{-\omega}^{0} f(x)dx\right|}{\sqrt{\int_{-\omega}^{\omega} f^2(x)dx}} \cdot \frac{\left|f^{'}(0)\right|}{\sqrt{\int_{-\omega}^{\omega} f^{'2}(x)dx}} \tag{7}$$

Edges detected by the Canny operator are all local extrema to ensure the precision of detection. Firstly, the grey image is smoothed by a Gauss function. Then, normalized gradient image is obtained from the smoothed image to determine possible edge pixels. Two thresholds T_h and T_l are used to get continuous and robust results. If a pixel has a gradient more than T_h, it belongs to the edge. If the gradient is less than T_l, it is not edge pixel. Otherwise, the pixel will be determined as edge pixels if there are edge pixels in its neighbourhood and this process helps to improve the continuity of detected edges.

In our experiments, the implementation of the Canny operator in the OpenCV package is adopted for edge detection. As seen, T_h and T_l are two important parameters in the Canny operator and different thresholds will lead to quite different edge results. For the G-image in Fig 1, we give the detected edges using different thresholds in Fig 2. For a given T_h, small T_l can achieve more detail edges, but too small T_l may cause noise (see the girl’s face in Fig 2). Therefore, how to automatic select suitable thresholds is a basic problem for effective edge detection of Canny operator.

For consistency in evaluations, in our method T_h and T_l are automatically determined as follows:

$$T_h = \mu + \max(\mu / 2, \sigma) \tag{8}$$

$$T_l = |\mu - \sigma| / 2 \tag{9}$$

where μ and σ are the mean and standard derivation of any given grey image. For G-image in Fig. 1, we can find $T_h = 208$ and $T_l = 55$, and the corresponding edges is given in Fig 2(b), which is better than Fig 2(a) and Fig 2(c).

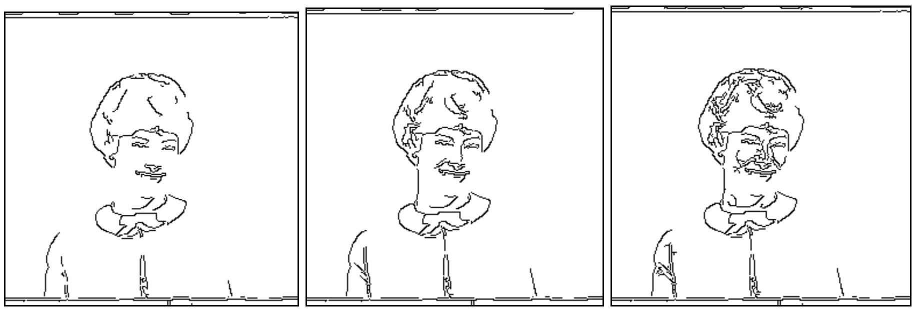

(a) $T_l = 100$ (b) $T_l = 55$ (c) $T_l = 10$

Figure 2. Extracted edges using the Canny edge detector with $T_h = 208$ and T_l changes from 100, 55 to 10, respectively.

For a given colour image, its edges are then extracted as follows. Firstly, the associate G-image and R-image are obtained. Secondly, using the Canny operator with automatically determined parameters, edges are detected from these two images as E_G and E_R, respectively. Finally, edges for the colour image E_{final} are determined as follows.

$$E_{final} = E_R \bigcup E_G \tag{10}$$

As for the colour image in Fig. 1 and E_G in Fig. 2, we give its E_R and E_{final} in Fig. 3. Note that E_G is useful to recover edges from grey part (no component difference) of images, as grey pixels in image f with same values in three colour components will appear as zero and cannot be distinguished from both the D-image and R-image.

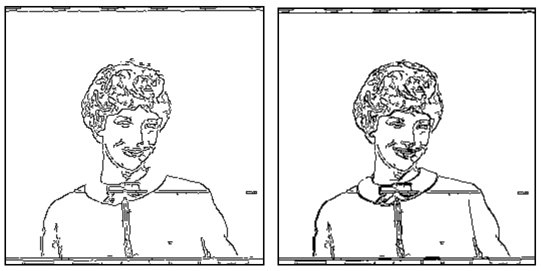


Fig. 3. E_R (left) and E_{final} (right) of colour image in Fig 1.

4 Results and Discussions

In order to evaluate our fusion scheme in improved colour edge detection, we take edges from RGB, YC_bC_r and HSV spaces for comparisons. RGB space has been selected because it is widely used for colour representation especially in computer graphics. On the other hand, we choose YC_bC_r and HSV spaces for two reasons: i) both of them reflect certain human perceptions of colour, ii) either of them represents one group of similar colour spaces, such as YC_bC_r can represent YUV, and HSV can also represent HSL or HSI etc.

For the three colour spaces above, corresponding edges are extracted as follows: Firstly, all the colour components have their values normalized within [0,255]. Secondly, edges are extracted from each colour component using the Canny operator with automatically determined parameters. Thirdly, the final edge for each image is obtained as the union of edges extracted from each colour component. For the original colour image "green girl" in Fig. 1, Fig. 4 illustrates edges extracted from RGB, YC_bC_r, HSV spaces and our fusion scheme for comparisons. As seen, our method has more continuous (see closed face contour) and accurate edges than edges from RGB and YC_bC_r spaces, and RGB edges is much better than YC_bC_r edges. In addition, edges extracted from HSV space have too much noise.

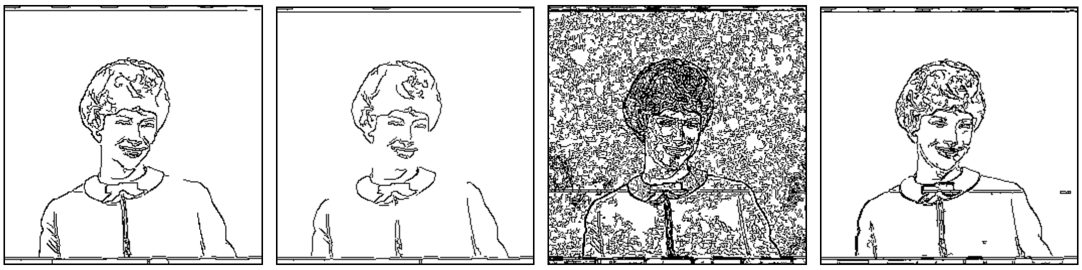


Figure 4. Comparing extracted edges: from left to right, the images are edges detected from RGB, YCbCr, HSV spaces and our method from the original colour image in Fig. 1.

4.1 Effectiveness Evaluation

To further evaluate the effectiveness of our proposed algorithm, edges extracted from both synthetic and real images are further compared. Three standard test images, "lena", "pepper" and "house", as shown in Fig. 5, are used in this group of experiments. For quantitative evaluation, ground truth of edge images are defined as reference images. These ground truth images are produced in a semi-manual way containing two steps: i) Extracting edges for each image using the logical OR of the results from RGB and YCbCR spaces, and ii) manual refinement of the extracted results to remove ghost edges and noise, etc. Basically, the precision rate η_{p} and recall rate detected η_{r} are defined for such evaluations as follows.

$$\eta_{\mathrm{p}} = \frac{tp}{tp+fp} = \frac{|E_{\mathrm{ref}} \cap E_{\mathrm{det}}|}{|E_{\mathrm{det}}|} \tag{11}$$

$$\eta_{\mathrm{r}} = \frac{tp}{tp+fn} = \frac{|E_{\mathrm{ref}} \cap E_{\mathrm{det}}|}{|E_{\mathrm{ref}}|} \tag{12}$$

where E_{ref} and E_{det} denote reference (as ground truth) and detected edge results; tp and fp refer respectively to true positive (correct detected) and false positive (false alarm) samples, and fn denotes false negative (missing in detection) samples. The samples are counted as number of edge pixels in the images accordingly.

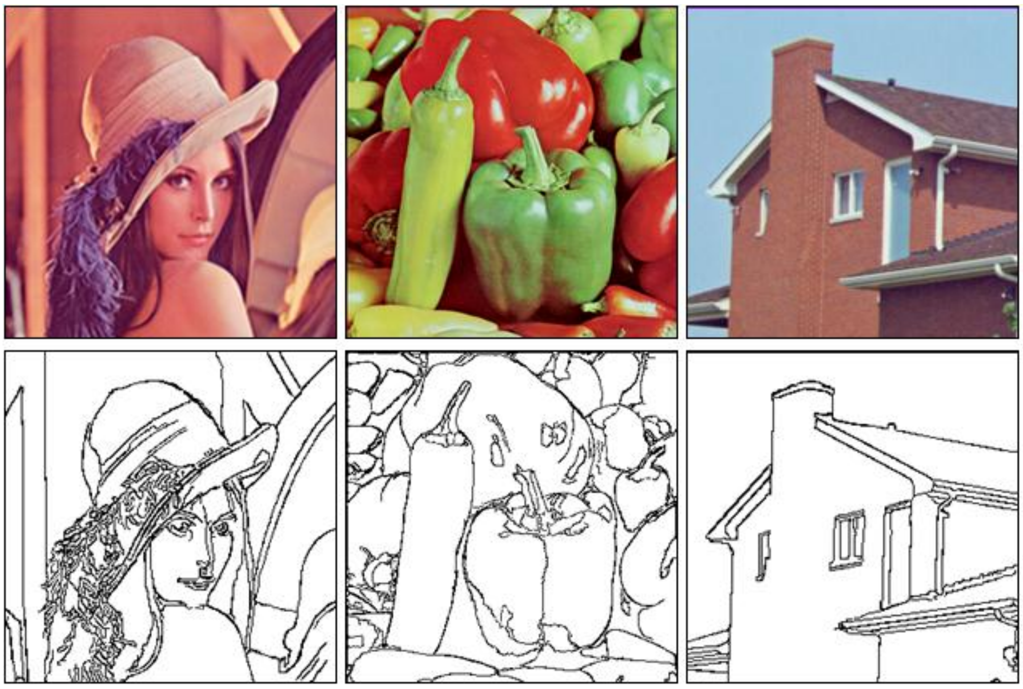


Figure 5. Three test images (top) and associated ground truth images of edges (bottom).

Table 1. Quantitative evaluations of edges detected from the test images.

images / edges	"lena"			"pepper"			"house"			**Average**		
	η_p	η_r	F_1	η_p	η_r	F_1	η_p	η_r	F_1	η_p	η_r	F_1
RGB	63.77%	85.09%	72.90%	37.12%	84.44%	51.57%	55.86%	85.41%	67.54%	73.92%	63.96%	51.75%
YC_bC_r	90.92%	75.85%	82.70%	66.75%	85.45%	74.95%	71.92%	80.68%	76.05%	83.15%	77.44%	73.06%
HSV	40.60%	83.09%	54.54%	41.72%	82.27%	55.36%	18.15%	89.17%	30.16%	59.41%	51.93%	56.84%
Grey	93.47%	72.87%	81.89%	78.75%	67.37%	72.61%	73.51%	72.00%	72.75%	82.74%	71.17%	75.22%
Green	72.5%	73.34%	72.92%	58.22%	64.00%	60.97%	65.95%	69.37%	67.61%	72.92%	63.64%	63.31%
Eigen [15]	94.15%	63.88%	76.12%	67.57%	64.01%	65.74%	74.03%	65.21%	69.34%	78.05%	67.93%	66.37%
Our	91.15%	92.97%	92.05%	78.59%	84.98%	81.66%	71.91%	91.53%	80.54%	92.06%	79.52%	84.57%

According to the three test images in Fig. 5, the results of quantitative evaluations are given in Table 1. For each test image, seven edge results are compared including those extracted from colour spaces of RGB, YCbCr and HSV, single component of the grey image, the green channel and a pseudo-grey channel [15] as well as our fusion scheme. Since this pseudo-grey component is attained via principle component analysis of the three colour channels, we simply namely its results as eigen edges. For visual comparisons, all the relevant edge images are also shown in Figure 6. As can be seen, our method yield quite high values of precision and recall rate in all the test images, followed by the edges extracted from YCbCr space, and the similarity here is due to the fact that channel difference information has been successfully employed in the detection of edges. While the other algorithms generate at least one poor value in terms of precision and recall measurements. Edges extracted from both RGB and HSV spaces suffer from massive false alarms, and HSV edge is even worse. Edges extracted from the grey image or the pseudo-grey channel is better than those from the green component, this is because grey image contains more information from other colour components which makes it more accurate in detection edges. However, the results from these three single components are worse than those from YCbCr space and our method. In addition, it is worth noting that the precision rate for the "house" image and the "pepper" image is less than that of the "lena" image. The reason behind is that there are fake edges in the previous two images caused by shadows or lighting, which has inevitably led to more false alarms and lower precision values.

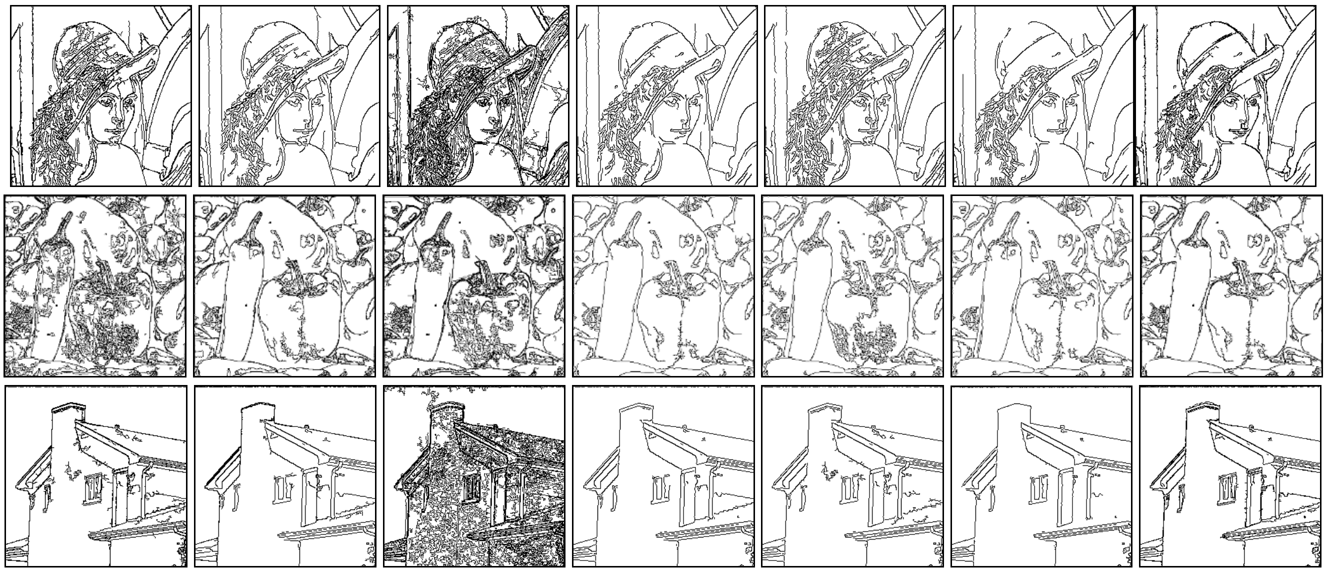


Figure 6. Comparing edges extracted from the three test images in Fig. 5. From left to right, the edges images are detected from RGB, YCbCr, HSV spaces, grey image, green component, eigen grey [15] and our method, respectively.

4.2 Robustness Evaluation

For robustness evaluation, edges extracted from colour images with attached Gaussian noise are compared. Firstly, the intensity values from each colour component are normalized within [0, 1]. Secondly, zero-mean Gaussian noise is added to the normalized intensity values with the variance value of σ_{v} changes from 0.002 to 0.008 where larger variance value indicates higher level of noise. Thirdly, the normalized image with attached noise is converted back so that its intensity in each component is within [0, 255]. For the original test images in Fig. 5, their corresponding noisy test samples under the variance value of 0.006 are given in Fig. 7. Noting that before applying the Canny operator for colour edge detection, median filtering in a 3*3 rectangle window is employed to remove noise. In addition, a post-processing step is used to remove false edges whose length is below a given threshold T_{e}.

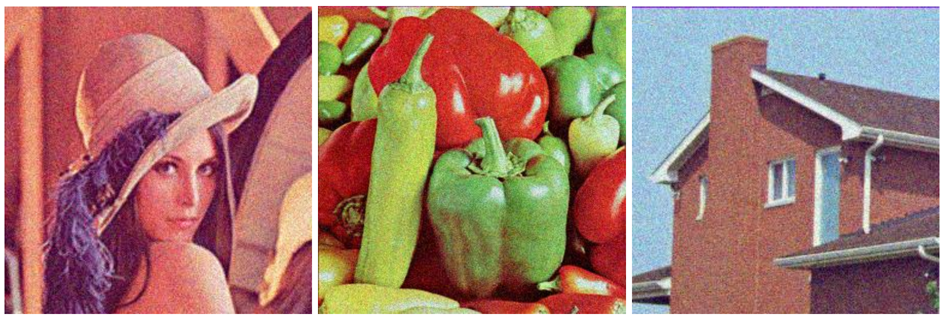


Figure 7. Three noisy test samples with additive Gaussian noise where the variance value is 0.006.

To achieve an overall evaluation of both the precision rate and evaluation rate, a F_1 measurement is defined below as

$$F_1 = \frac{2\eta_\mathrm{p}\eta_\mathrm{r}}{\eta_\mathrm{p}+\eta_\mathrm{r}} \tag{13}$$

In the following, edges extracted from noisy images after length thresholding are compared. For simplicity, our method is only compared with the edges extracted from the pseudo-grey component [15] as well as RGB and YCbCr spaces, and all others are ignored for their poor performance in Table 1. Please note that only the additive Gaussian noise is tested, and more complex noise models like multiplicative noise are not considered as they are not popular in natural scenes. For each test image, three curves are plotted and shown in Fig. 8 to illustrate the change of F_1 value vs. the variance values of additive Gaussian noise. As can be seen, the F_1 values degrade significantly with increasing variance values of additive Gaussian noise, while our fusion scheme performs the best over all the test images. Edges extracted from RGB space are the worst which indicates that they are more sensitive to the added noise. Besides, it seems that edges extracted from the pseudo-grey component [15] are less sensitive to noise especially when the variance value is high. Furthermore, in our implementation the threshold T_e is empirically determined as 12. As shown in Fig. 8, it is found that our results are insensitive to the threshold T_e which is used to remove short edges.

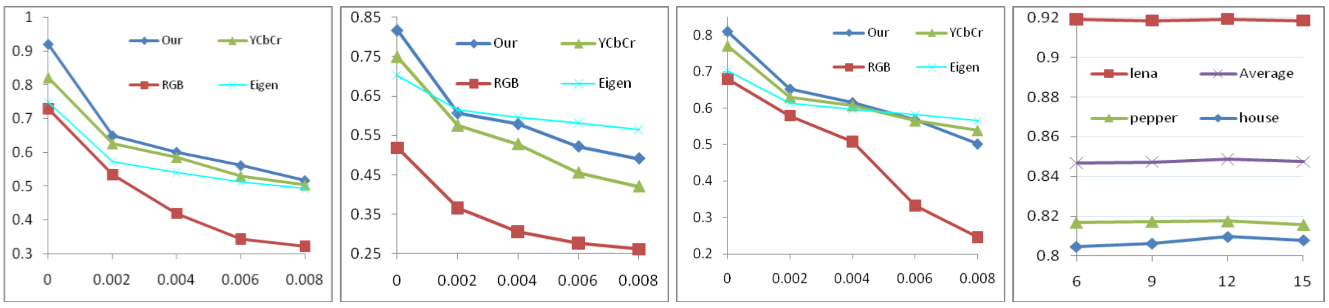


Figure 8. Comparison of F_1 values (y-axis) vs. the variance values of additive Gaussian noise (x-axis, the left three images and three plots respectively correspond to the test images “lena”, “pepper” and “house”) and various threshold values (x-axis) (right). The results labelled with “eigen” is extracted from the pseudo-grey image in [15].

In addition, visual comparisons over the edges extracted from noisy samples are also shown in Fig. 9, in which two groups of noisy samples are used and the variance values of the Gaussian noise are 0.002 and 0.006, respectively. Again, we can easily find that RGB edges are extremely sensitive to noise. Although both our results and YCbCr edges exhibit some robustness to the noise, the false alarms caused in the corresponding images are different. In general, false alarms in YCbCr edges are adjacent to real edges, but in our results these false alarms are separated. Therefore, it is possible to further improve the accuracy of our algorithm by introducing more powerful post-processing to reduce these separated fake edges. In addition, edges extracted from pseudo-grey component has fewer false alarms though more missing edges.

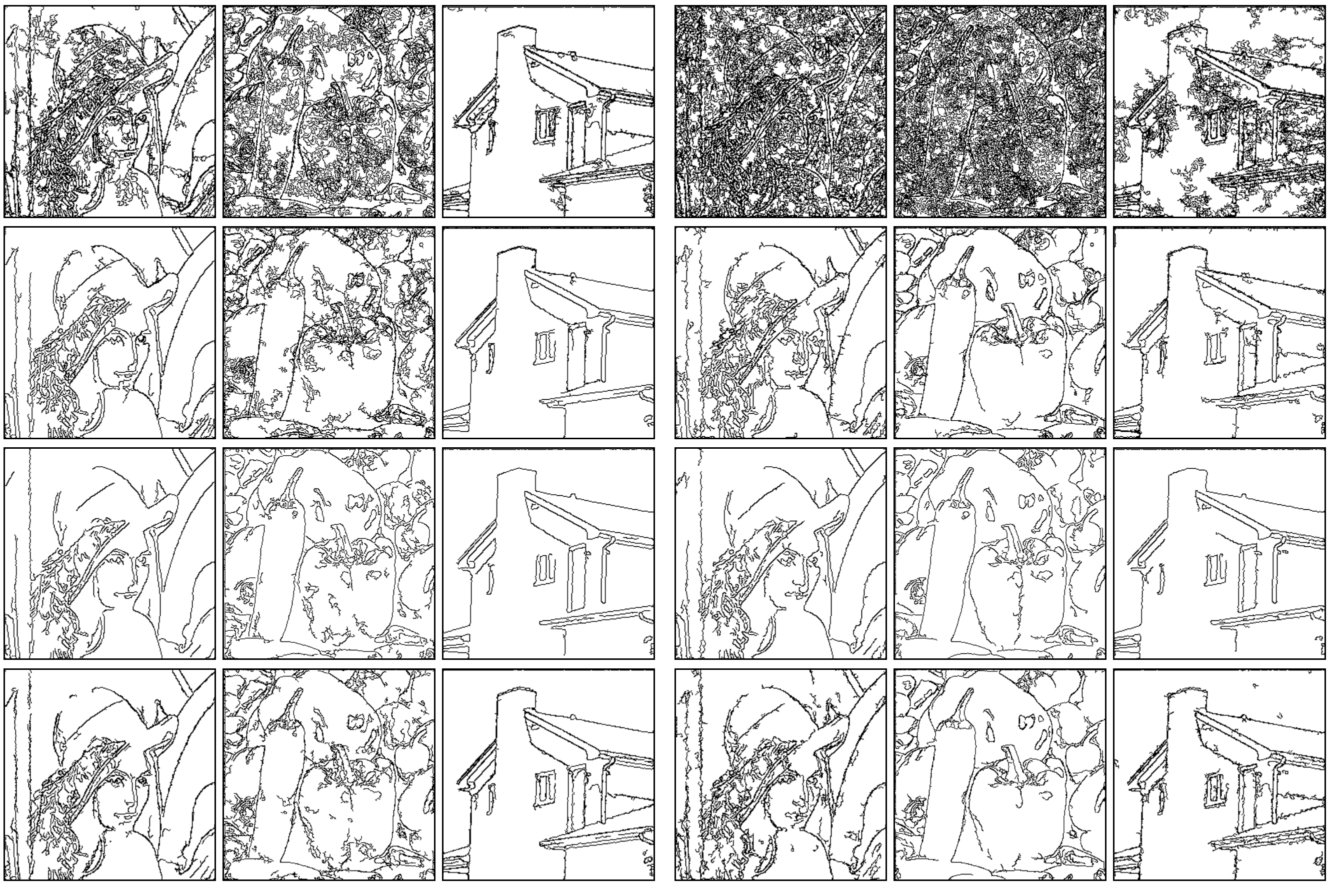


Figure 9. Visual comparison of edges extracted from noisy samples. In each row, two groups of edges images are displayed which are results from noisy samples with the variance values at 0.002 (the left) and 0.006 (the right), respectively. From top to bottom, the four rows correspond to edge results detected from RGB, YCbCr, pseudo-grey[15] and our scheme.

The computing complexity of the proposed algorithm contains three main parts including i) extraction of G-image and R-image, ii) detection edges using Canny detector on the two single-component images, and iii) post-processing. In general, the first step takes most of the running time, i.e. more than 60% of the total time of our method. Since the complexity of edge detection algorithms rely on image contents, especially for the Canny detector where tracing of edges is employed, the complexity of our method is compared with others in a relative way as follows. For one test image, different edge detection methods are applied for 100 times on the same machine and the executive times are taken as a good indicator to evaluate the corresponding complexity of the approaches. Executive times obtained in edge extraction from the three test images in Fig. 5, using RGB and YCbCr colour spaces, pseudo-grey component, and our scheme, are listed in Table II for comparisons. As can be seen, the executive time for the "pepper" image is the longest as it contains the most edges. The overall complexity of YCbCr edges are the minimum, followed by RGB edges and eigen edges, and our scheme is the most complex one. However, our method is only 23% more complex than YCbCr edges, and the additional cost is quite limited for the good performance achieved in our tests.

Table 2. Comparison of complexity by executive time (in seconds) via running 100 times of the test in extracting edges from the images in Fig. 5.

images / methods	"lena"	"pepper"	"house"	**Average**	
				Time	Ratio
RGB edges	3.203	4.719	3.125	3.682	107%
YC_bC_r edges	3.188	4.188	2.985	3.454	100%
Eigen [15] **edges**	3.265	4.532	3.150	3.649	106%
Our edges	3.797	5.219	3.688	4.235	123%

5 Conclusions

We have proposed an improved method on colour edge detection. We have found that inter-component information in colour images is very important for accurate edge detection, though it has been ignored in many existing approaches. Through the fusion of intensity and chromatic difference, the proposed scheme is found very useful in generate better colour edges. In terms of effectiveness and robustness under attached Gaussian noise, both visual and quantitative evaluations are achieved. Comprehensive results from several standard test images have fully verified both the effectiveness and robustness of our proposed approach, which is found outperforming edges extracted from RGB, YCbCr and HSV spaces. Further investigation will be to apply the fusion scheme on image segmentation and content-based retrieval applications as well as to introduce more powerful post-processing to remove separated false alarms.

Acknowledgements

The original colour image in Fig. 1 comes from web (http://sipi.usc.edu/services/database/Database.html). Finally, the authors wish to acknowledge the partial financial support under EU IST FP-6 Research Programme with the integrated project: HERMES (Contract No IST-216709).

References

[1] Canny, J. F.: ‘A computational approach to edge detection’, IEEE T-PAMI, 1986, 8, (6), pp. 679-698

[2] Choi, H.-G., Jung, W.-C., Min, J., Lee, W. H., and Choi, J.-W.: ‘Colour image detection by biomolecular photoreceptor using bacteriorhodopsin-based complex LB films’, Biosensors and Bioelectronics, 2001, 16, (9), pp. 925-935.

[3] Fotinos, A., Economou, G., and Fotopoulos, S.: ‘Use of relative entropy in colour edge detection’, Electronics Letters, 1999, 35, (18), pp. 1532-1534

[4] Geusebroek, J.M., van den Boomgaard, R., Smeulders, A.W.M., and Geerts, H.: ‘Colour invariance’, IEEE T-PAMI, 2001, 23, (12), pp. 1338-1350

[5] Gevers, T., and Stokman, H. M. G.: ‘Classification of colour edges in video into shadow-geometry, highlight, or material transitions’, IEEE T-Multimedia, 2003, 5, (2), pp. 237-243

[6] Niu, L.-Q., and Li, W.-J.: ‘Colour edge detection based on direction information measure’. Proc. WCICA, 2006, pp. 9533-9536

[7]. Pal, N. R., and Pal, S. K.: ‘A review on image segmentation techniques’, Pattern Recognition, 1993, 26, (9), pp. 1277-1294

[8] Palus, H.: ‘Representation of colour images in different colour spaces’, in Sangwine S. J. and Horne R. E. N. (eds), The Colour Image Processing Handbook (1998)

[9] Ruzon, M. A., and Tomasi, C.: ‘Colour edge detection with the compass operator’. Proc. CVPR, 1999, pp. 160-166

[10] Rital, S., and Cherifi, H.: ’A combinatorial color edge detector’. Proc. ICIAR, 2004, pp. 289-297

[11] Tao, H. and Huang, T. S.: ‘Colour image edge detection using cluster analysis’. Proc. ICIP, 1997, pp. 834-837

[12] van de Weijer, J., Gevers, Th., and Geusebroek, J.M.: ‘Colour edge detection by photometric quasi-invariants’. Proc. ICCV, 2003, pp. 1520-1525

[13] van de Weijer, J., Gevers, Th., and Geusebroek, J.M.: ‘Edge and corner detection by photometric quasi-invariants’, IEEE T-PAMI, 2005, 27, (4), pp. 625-630

[14] Zhu, S.-Y., Plataniotis, K. N., and Venetsanopoulos, A. N.: ‘Comprehensive analysis of edge detection in colour image processing’, J. Optical Engineering, 1999, 38, (4), pp. 612-625

[15] Dikbas, S., Arid, T., and Altunbasak, Y.: ‘Chrominance edge preserving grayscale transformation with approximate first principal component for color edge detection”. Proc. ICIP, 2007, pp. 261-264

[16] Hwang, Y., Kim, J. S., and Kweon, I. S.: ‘Change detection using a statistical model in an optimally selected color space’, J. Computer Vision and Image Understanding, 2008, 112, (3), pp. 231-242

[17] Evans, A. N., and Liu, X. U.: ‘A morphological gradient approach to color edge detection’, IEEE Trans. Image Proc., 2006, 15, (6), pp. 1454-1463

[18] Zhou, C. and Mel, B. W.: ‘Cue combination and color edge detection in natural scenes’, Journal of Vision, 2008, 8, (4):4, pp. 1-25

[19] Theoharatos, C., Economou, G. and Fotopoulos, S.: ‘Color edge detection using the minimal spanning tree’, Pattern Recognition, 2005, 38, (4), pp. 603-606

[20] Toivanen, P. J., Ansamaki, J., Parkkinen, J.P.S., and Mielikainen, J.: ‘Edge detection in multispectral images using the self-organizing maps’, Pattern Recognition Letters, 2003, 24(16), pp. 2987-2994

Applying Feature Selection for Effective Classification of Microcalcification Clusters in Mammograms

Dong Wang, Jinchang Ren, Jianmin Jiang, and Stan S. Ipson

School of Informatics (EIMC), University of Bradford, BD7 1DP, UK.
{d.wang6, j.ren, j.jiang1, s.s.ipson}@bradford.ac.uk

Abstract. Classification of benign and malignant microcalcification clusters (MCC) in mammograms plays an essential role for early detection of breast cancer in computer aided diagnosis (CAD) systems, where feature selection is desirable to improve both the efficiency and robustness of the classifiers. In this paper, three approaches are applied for this task, including feature selection using a neural classifier, a clustering criterion and a combined scheme. To evaluate the performance of these feature selection approaches, a same neural classifier is then applied using the selected features and the classification results are then compared. In our dataset in total 748 MCC samples are detained from the well-known DDSM database, where 39 features are extracted for each sample. Comprehensive experiments with quantitative evaluations have demonstrated that the best classification rate can be achieved using 15-20 selected features. Also it is found that applying features selected from clustering rules can yield better performance in separate and combined scheme.

1 Introduction

Breast cancer comprises over 10% of all cancer incidences among women, which is the second most common type of non-skin cancer (after lung cancer) and one of the most common causes of cancer deaths. In every year, about 1 million women worldwide have been diagnosed with breast cancer [1]. According to a report from National Cancer Institute, about 1 in 8 women have breast cancer in US, whilst this ratio becomes 1 in 11 in Australia and one in nine in UK and Canada. In 2007, there were about 1.15 million women had found having breast cancer in the world and about 36% of them died because of it.

As breast cancer becomes the most threat to women's health, many scientists and researchers have been focusing on how to solve it efficiently [11]. It is found that early detection of breast cancer will help reducing the death rate. Since digital mammogram has the major role for early detection of breast cancer, detection of benign and malignant samples in mammogram images become a bottleneck for radiologists and researchers towards automatic diagnosis. The main problem that makes it difficult here is that there is no unique pattern for individual benign and malignant samples, i.e. variety in their appearances [6]. In the work reported in [7][9], the CAD systems can increase the detection rate of breast cancer at about 5-15%. The flowchart of a typical CAD system is shown in Figure 1, where feature extraction and feature selection is the core of the whole system.

After decades of research, many relevant works have been reported in the detection and classification of breast abnormalities of mammograms in the literatures. The most popular methods used in this context include the artificial neural network (ANN) [6, 8], fuzzy logic [2, 9], statistical approaches [4] and several combined solutions [3, 12]. It is our aim to apply feature selection approaches in breast cancer detection, especially in classification of MCCs in mammograms, where neural classifier and statistical clustering principles are employed for their well-known good performance in this field [1, 4, 11].

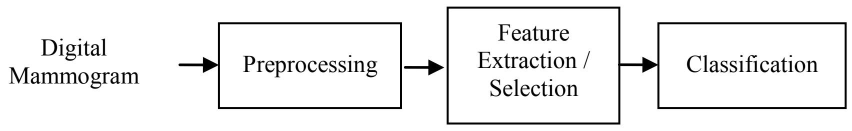


Fig 1: Flowchart of a typical CAD system.

In our dataset, there are 748 MCC samples in the database, where 633 are benign and 115 are malignant. All the sample data are from the University of South Florida's DDSM database (The Digital Database for Screening Mammography) [13, 14]. Before applying classification, in total 39 different features have been extracted for each sample and indexed from 1 to 39 [15]. For more effective and robust classification, feature selection approaches are employed in order to reduce data redundancy and improve the classification performance. Further details are discussed in the next sections.

2 Feature Selection Schemes

Feature selection is to choose the most representative components among the feature set, and the principle is based on the fact that these selected features should be most discriminative in classifying the training samples. This discriminative ability is determined using two basic classifiers, including neural network and statistical clustering.

2.1. Feature Selection using Neural Network

The basic criterion here is to correctly detect as many malignant samples as possible, and of course, to reduce the occurrence of wrongly detected malignant samples. A back propagation (BP) neural network, which contains an input layer, one hidden layer and one output layer, as shown in Fig. 2, is employed. The number of nodes for input layer and hidden layer is 32 and 16, respectively. The features inputted can be single ones or combined ones and through changing hidden units, learning rates and momentums the best detection results can be achieved. The output value is normalized within [0.1, 0.9]. Those lying within the range (0.5, 0.9] are taken as benign and those lying within the range [0.1, 0.5] are taken as malignant.

Each of the 39 features is applied as input to the designed neural classifier to determine its performance. At this stage, 90% of the test samples are used for training and the remaining 10% for testing. Then, each separate feature is ranked according to its recognition rate and taken as discriminative ability.

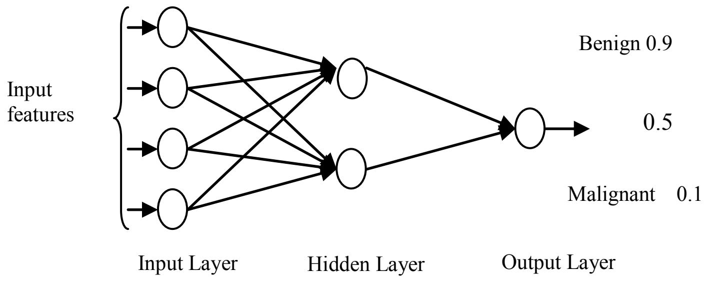


Figure 2: Outline of Neural Network structure used for feature selection

2.2. Feature Selection using Clustering Rules

In probability and statistics, the standard deviation of a probability distribution, random variable, or population of values is a measure of the spread of its values. The standard deviation is usually denoted by the letter σ (lower case sigma), and is defined as the square root of the variance. As for variance, large or small variance values refer to important or unimportant dynamics in Clustering rules analysis. To this end, standard deviation can be also considered as a kind of measurement of dynamics. As a result, we intend to use the standard deviation to rank each feature in several steps as follows.

1) For each feature, the mean value of all its samples in target 1 (malignant) group and target 0 (benign) group are respectively determined as u_1 and u_0.
2) Accordingly, σ_1 and σ_0 are also obtained as standard deviation for all samples in two groups corresponding to target 1 and target 0;
3) Based on extracted mean and standard derivation, an overall rank of each feature is defined below:

$$rank(.) = |u_1 - u_0| / (\sigma_1 + \sigma_0) \tag{1}$$

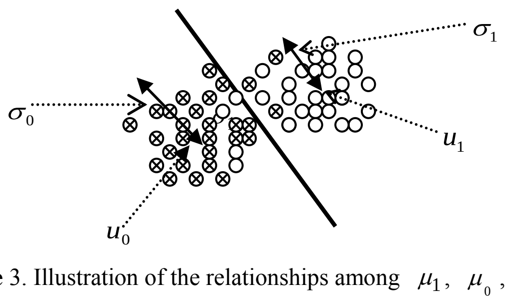


Figure 3. Illustration of the relationships among μ_1, μ_0, σ_1 and σ_0.

The relationships among μ_1, μ_0, σ_1 and σ_0 are illustrated in Fig. 3, in which two groups of samples are represented by hollow circle and circle inside cross, respectively. The centroids in each group are denoted as μ_1 and μ_0, and parameters σ_1 and σ_0 are used to denote the compactness of samples in each group. The basic criterion in ranking the features is to measure if it is easily to separate two groups of samples. If the distance between the two centriods is far enough, say larger than $\sigma_1 + \sigma_0$, it will be ranked high as more likely the samples can be split into two groups.

2.3. Feature Selection using Combined Approach

Based on the results of feature selection from neural network and clustering rules, a combined scheme is then proposed as follows. Firstly, for the features selected using neural network and clustering rules, we sequentially assign a score within [1, 39] to each feature according to its decreasing discriminative ability. Then, for each feature an overall rank is obtained as the sum of two scores assigned, and the feature of lower overall rank will be selected of first priority in further classification.

3. RESULTS AND DISCUSSIONS

In this section, results in feature selection using neural network, clustering rules and combined scheme are compared and analyzed. For quantitative evaluation, three recognition rate C_1, C_0 and C_{all} are defined as follows.

$$C_1 = DA/SA \tag{2}$$

$$C_0 = DN/SN \tag{3}$$

$$C_{all} = (DN + DA)/(SN + SA) \tag{4}$$

where DA and SA respectively refer to corrected detected abnormal samples and the sum of abnormal samples; DN and SN denote corrected detected normal samples and the sum of normal samples.

In our experiments, we set the number of iterations in the neural classifier as 10000. Each separate experiment is tested for 20 times in which data used for training and testing can be randomly selected for a wide coverage of all cases. Finally, the quantitative evaluations are achieved as an average measurement over all the 20 tests. Details of the results and analysis are presented below.

3.1. Results of Feature Selection via NN

In Table 1, features are listed with their associated discriminative ability in decreasing order which is determined using the neural classifier.

Table 1. List of features indexed from 1 to 39 and their discriminative ability (rank) decided using the NN.

Index	Rank	Index	Rank	Index	Rank	Index	Rank
#15	0.2333	#33	0.1280	#8	0.0631	#37	0.0438
#1	0.1912	#23	0.1210	#35	0.0543	#26	0.0403
#2	0.1877	#19	0.1035	#38	0.0543	#32	0.0385
#16	0.1666	#20	0.0929	#22	0.0526	#36	0.0385
#39	0.1526	#7	0.0877	#27	0.0526	#13	0.0385
#18	0.1491	#29	0.0842	#28	0.0491	#10	0.0368
#5	0.1473	#14	0.0842	#25	0.0491	#11	0.0350
#17	0.1421	#12	0.0701	#24	0.0456	#9	0.0350
#31	0.1403	#21	0.0666	#3	0.0456	#6	0.0315
#34	0.1368	#30	0.0666	#4	0.0438		

From Table 1 we can see, discriminative ability differs much over all features where the maximum and the minimum ones are 0.2333 and 0.0315, respectively, and this has clearly demonstrates the potential for feature selection. In our experiments, features of higher discriminative ability are selected with first priority for further training and testing. Under different numbers of selected features, the classification results are compared in Figure 4.

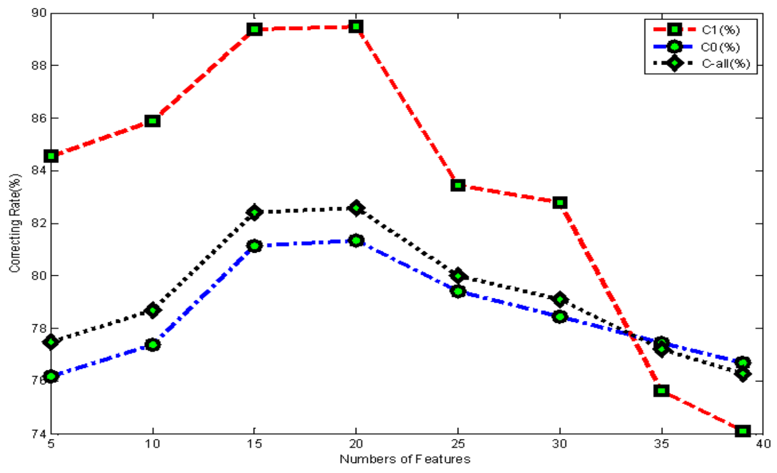


Figure 4. Classification results with the same NN using different number of features selected via neural network.

In Figure 4, some facts can be summarized below: i) When the number of selected features changes from 5 to 20, the correcting rates of C_1, C_0 and C_{all} stably increase; ii) When the increasing number of selected features becomes more than 20, the correcting rate begin to decrease; iii) It is suggested that the first 15 to 20 features will yield particular good performance with highest correcting rate in terms of C_1, C_0 and C_{all} .

3.2. Results of Feature Selection via Clustering

Using our ranking function in (1), we can determine ranks for each of the 39 features on the basis of clustering rules and the results are given in Table 3.

According to the ranking results in Table 3, again we choose some features for classification whilst features of high rank values are to be selected first. The same neural classifier is applied in our test where 90% of the samples are used for training and 10% for testing. The testing results are reported in Figure 5.

Table 3. List of features indexed from 1 to 39 and their discriminative rank decided using clustering rules.

Index	Rank	Index	Rank	Index	Rank	Index	Rank
#39	0.9743	#7	0.4510	#28	0.2321	#1	0.065
#15	0.8506	#19	0.4192	#13	0.2072	#37	0.0652
#2	0.7187	#18	0.3874	#5	0.1963	#11	0.0608
#16	0.7026	#34	0.3797	#3	0.1765	#36	0.0573
#17	0.6867	#22	0.3386	#38	0.1572	#32	0.0433
#33	0.6627	#23	0.3384	#24	0.1501	#4	0.0405
#27	0.6252	#20	0.3104	#35	0.1306	#10	0.0390
#25	0.6109	#31	0.2931	#12	0.1073	#9	0.0287
#21	0.5856	#30	0.2812	#14	0.1003	#6	0.0046
#29	0.5286	#26	0.2623	#8	0.0765		

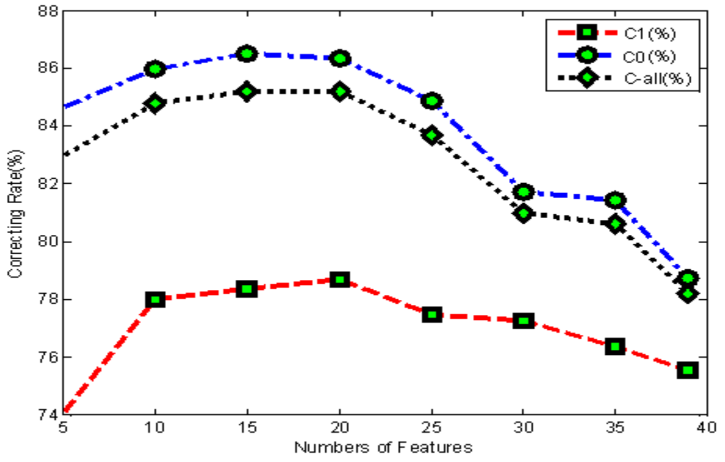


Figure 5. Classification results with the same NN using different number of features selected via clustering rules.

In Figure 5, we can see that with the increasing number of selected features used in classification the correcting rate will firstly increase and then decrease. Again the correcting rates achieve their maximum (peaks) when the number of selected features lies between 15 and 20.

If we compare the results in Figure 4 and Figure 5, we can easily find that the results in Figure 5 yield an overall better correcting rate in terms of C_{all} and also much higher correcting rate of C_0 than that of Figure 4. On the contrary, the correcting rate C_1 in Figure 5 is much lower than that of Figure 4.

3.3. Results of Combined Feature Selection

Using our combined feature selection scheme, the final rank determined for each of the 39 features is listed in Table 5. Again, the correcting rate under various numbers of selected features are compared and shown in Figure 6.

Table 5. Combined rank for each feature while lower rank refers to high priority.

Index	Rank	Index	Rank	Index	Rank	Index	Rank
#15	3	#7	26	#22	39	#13	57
#2	6	#31	27	#12	46	#37	63
#39	6	#21	28	#14	46	#4	66
#16	8	#23	28	#28	47	#32	68
#17	13	#5	30	#38	48	#36	68
#33	17	#20	31	#35	49	#11	70
#18	19	#27	32	#8	51	#10	73
#34	24	#1	33	#26	52	#9	76
#19	25	#25	35	#24	54	#6	78
#29	26	#30	39	#3	53		

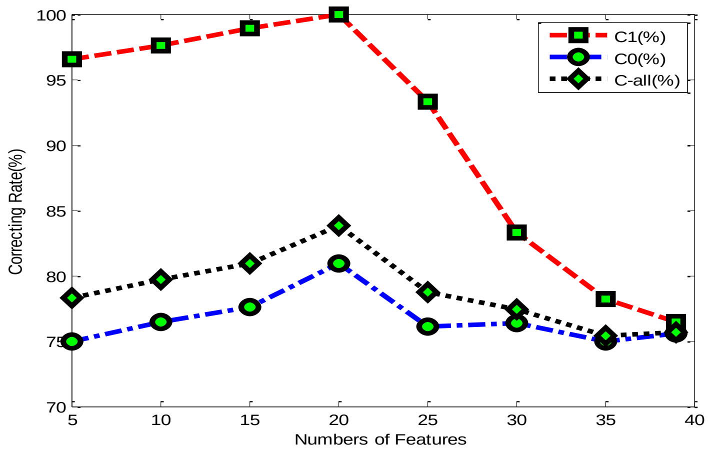


Figure 6. Classification results with the same NN using different features selected via combined approach.

From Figure 6, we can see that the correcting rate of class 1 (malignant) is quite impressive, especially when 15 to 20 features are selected and the C_1 value achieved is nearly 100%. Meanwhile, the correcting rate of class 0 (benign), C_0, becomes relative low in a range of 76-80%. This means that quite a few false alarms are detected this it needs to be further improved.

4. CONCLUSIONS

In this paper, we have employed three approaches for feature selection in classification of MCCs in mammograms for breast cancer detection and their performances are compared using quantitative evaluations. Based on our results, it is found that the combined approach is useful in successfully detecting malignant samples. Also it is found that only about half of the all 39 features are useful in generate better results. This has not only reduced the complexity of the problem, it also reveals a potential for more robust classification. Further investigations will be to apply combined classifiers to further reduce false alarms, plus a strategy to deal with the problem of unbalanced data.

5. REFERENCES

[1] M Parkin, F Freddie Bray, J Ferlay, P Pisani. Global Cancer Statistics, *A Cancer Journal for Clinicians*, vol. 55, American Cancer Society; p. 74-108. 2005.

[2] H Cheng, YM Lui, RI Freimanis. *A novel approach to microcalcification detection using fuzzy logic technique*. IEEE Trans Med Imag; vol 17 p442-50.1998

[3] CA Pena-Reyes, M Sipper. *A fuzzy-genetic approach to breast cancer diagnosis.* Artif Intell Med; vol.17 p131-551999

[4] P Zhang, B Verma, K Kumar. *Neural Vs. Statistical classifier in conjunction with genetic algorithm feature selection in digital mammography*. Pattern Recogn Lett; vol 26(7) p909-19. 2005

[5] Pir, R iyakul; P Piamsa-Nga, *Feature reduction in graph analysis.* SENSORS Volume: 8 Issue: 8 Pages: 4758-4773. 2008.

[6] B Verma, *Novel Network Architecture and Learning Algorithm For The Classification of Mass Abnormalities in Digitized Mammograms*, Vol. 42, Artificial Intelligence in Medical, p.67-79 2008.

[7] ED Pisano, C Gatsonis, et al. *Diagnostic performance of digital versus film mammography for breast cancer screening*. N Engl J Med; vol 353:1773-83. 2005

[8]. A Papadopoulos, DI Fotiadis, A Likas. *An automatic microcalcification detection system based on a hybrid neural network classifier.* Artif Intell Med: vol 25:149-67. 2002;

[9] C. E. Metz and J. H. Shen, *Gains in accuracy from replicated readings of diagnostic images: prediction and assessment in terms of ROC analysis*, Med. Decision Making, vol. 12, p241-244, 1994

[10] M Markey, J Lo, G Tourassi, C Floyd. *Self-organizing map for cluster analysis of a breast cancer database.* Artif Intell Med; vol. 27 p113-27, 2003.

[11] HD Cheng, X Cai, X Chen, L Hu, X Lou. *Computer-aided detection and classification of microcalcification in mammograms: a survey*. Pattern Recogn: vol. 36 p2967-9, 2003.

[12] B Verma, J Zakos. A *computer-aided diagnosis system for digital mammograms based on fuzzy-neural and feature extraction techniques.* IEEE Trans Inform Technol Biomed vol. 5 p46-54, 2001.

[13] M. Heath, K. Bowyer, D. Kopans, R. Moore, W.P. Kegelmeyer, "The digital database for screening mammography," in Proc. *the 5th Int. Workshop on Digital Mammography*, pp. 212-218, 2001.

[14] M. Heath, K. Bowyer, D. Kopans, W.P. Kegelmeyer, R. Moore, K. Chang, S. MunishKumaran, "Current status of the digital database for screening mammography," in Proc. *the 4th Int. Workshop on Digital Mammography*, pp. 457-460, 1998.

[15] ZQ Wu, J Jiang, YH Peng. *A filter-based approach towards automatic detection of microcalcification.* LNCS, vol. 4046, pp. 424-432, 2006.

Effective Recognition of MCCs in Mammograms Using an Improved Neural Classifier

Jinchang Ren, Dong Wang, Jianmin Jiang,

School of Computing, Informatics & Media, University of Bradford, Bradford, U. K.

Email: j.ren@bradford.ac.uk, j.jiang1@bradford.ac.uk, d.wang6@bradford.ac.uk

Corresponding Author:

Dr. Jinchang REN

Email: j.ren@bradford.ac.uk
Tel: +44-1274-233993
Fax: +44-1274-233727

Abstract

Classification of microcalcification clusters from mammograms plays essential roles in computer-aided diagnosis for early detection of breast cancer, where support vector machine (SVM) and artificial neural network (ANN) are two commonly used techniques. Although SVM is found performing better than ANN, the average accuracy achieved is only around 80% in terms of the area under the receiver operating characteristic curve A_z. This performance may become much worse when the training samples are imbalanced. As a result, a new strategy namely balanced learning with optimized decision making is proposed to enable effective learning from imbalanced samples. When the proposed learning strategy is applied to individual classifiers, the results on the DDSM database have demonstrated that the performance from both ANN and SVM has been significantly improved. Under the improved learning scheme, ANN surprisingly outperforms SVM in both training and testing. These are in contrast to the results in reported [20], which suggested that similar strategy could not improve the classification performance. An average improvement of more than 10% in the measurements of F1 and Az has fully validated the effectiveness of our proposed method for the successful classification of clustered microcalcifications.

***Index Terms*—microcalification clusters (MCC), balanced learning, optimized decision making, neural network, support vector machine, mammography, computer-aided diagnosis.**

I. Introduction

Breast cancer is the most common diagnosed cancer among woman. In the United Kingdom, every year there are about 45000 cases are diagnosed, and more than 1100 women die from this cancer every month [1]. In the United States, about 182500 cases were diagnosed in 2008, and nearly 40500 women die from this disease annually [2]. Since the reasons behind are still uncertain, early detection and diagnosis is the key for improving breast cancer prognosis [3, 4]. Among many available techniques, x-ray mammogram has been one of the most reliable methods for early detection of such disease [11]. Generally, it can increase the survival ratio by 20% to about 80% for patients. In England alone, around 1400 lives are saved each year via the NHS breast screening [1]. Other popular means for breast cancer detection include magnetic resonance imaging (MRI) [5], electrical impedance spectroscopy (EIS) [6], ultrasound [7], and infrared imaging [8].

Although mammogram contains useful information for the early detection of breast cancer, it is difficult for radiologists to make accurate and consistent judgments due to the huge amount of data and widespread screening. Consequently, about 10-30% cases are missed during the routine check [3]. With the assistance of computer-aided diagnosis (CAD), the overall sensitivity from human observers can be improved by 10% on average, which provides a promising solution in such a context.

Detection and classification of microclacification clusters (MCCs) from mammograms plays important roles in early diagnosis of breast cancer. In early detected cases, MCCs can be found in 30-50% of the screened mammograms. This will increase to 60-80% if histological examinations of cancer cases are considered. The difficulty for the detection of MCCs is due to i) small size but various shapes, ii) low contrast and unclear boundary from surrounding normal tissue, etc. [3, 23].

To solve such problems, a typical CAD system contains at least four stages including preprocessing, feature-based extraction of regions of interest (ROI), detection of MCCs, and classification. The preprocessing covers noise suppression and contrast enhancement, including histogram equalization etc. [35], which is useful for robust extraction of features and ROIs. The features include local statistics and texture modeling [10, 13], wavelets [12, 15, 18, 23], and morphological features [34]. From the segmented ROIs, MCCs can be detected using heuristics [14,23], fuzzy sets [11,16], sub-image decomposition and filtering [27], and machine learning algorithms [19, 25, 36], where shape features such as linear structure is widely used [24, 27, 29, 32].

Regarding classification of MCCs, a number of techniques have been presented using machine learning approaches to classify samples as malignant and benign, and this is also the focus of this paper. Among these techniques, two main streams are those using artificial neural networks (ANN) [17, 25-26, 28-30, 34, 36] and support vector machines (SVM) [9, 20-22, 30], along with other approaches like linear discriminant analysis (LDA) [29], Bayes classifiers [27], *K*-nearest-neighbor (KNN) clustering [15], genetic algorithms (GA) [14, 15] and decision-rules [11, 30]. According to the evaluation work in [20], SVM and other kernel based approaches including relevance vector machine and kernel Fisher discriminant (KFD) outperform ANN classifier in classification of MCCs. However, the area under the ROC curve A_z achieved by SVM is only 0.85 in comparison with 0.80 from ANN, which apparently has space for further improvement.

The reasons for the classification accuracy in terms of A_z above is not only the complexity of the problem, i.e. containing cases that cannot be judged even by radiologists as analyzed in [20], but also the shortcomings of single classifiers, especially the difficulty in dealing with imbalanced training set in machine learning. The imbalance here refers to the fact that one class is more heavily represented than the other. This is a common problem in real-world domains in detecting rare but important cases from large suspiciously normal samples [41]. Most existing machine learning algorithms fail in dealing with imbalanced data set as their predictions are biased to the class of majority samples [42].

In this paper, an improved over-sampling based balanced learning strategy is proposed for the classification of MCCs, which can avoid drawbacks of existing techniques. The performance along with the proposed optimized decision making has been fully validated using two individual classifiers including SVM and ANN. The proposed method is found effective in improving both the sensitivity and specificity rate while maintaining the computing complexity of the classifier.

The remaining part of this paper is organised as follows. Section II contains introductory concepts related to the SVM and ANN classifiers. In Section III, the proposed balanced learning and optimized decision making is presented. Section IV discusses the evaluation criteria and implementation details including the data set and extracted features. Experimental results are given and analyzed in Section V to fully validate the proposed methodology. Finally, brief conclusions are drawn in Section VI.

II. Review of SVM and ANN Learning Techniques

In this paper, the classification of MCCs is treated as a two-class pattern classification problem, and the two classes are referred to as "malignant" and "benign". If we denote $\mathbf{x} \in \Re^d$ as an input vector or pattern to be classified, and let scalar y denote its class label, i.e. $y \in \{-1,1\}$ for SVM and $y \in \{0,1\}$ for ANN. The training set $\mathbf{L}$ contains M samples, i.e. $\mathbf{L} = \{(\mathbf{x}_i, y_i)\}$ and $i \in [1, M]$. The problem here is how to determine a classifier $f(\mathbf{x})$ which can make correct decision and classify the input pattern into suitable classes. In this section, brief introductions to SVM and ANN are presented, which forms the base of our proposed improved classifier as presented in the next section.

A. The SVM Classifier

In general, a SVM classifier can be formed as follows,

$$f_{SVM}(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}) + b \tag{1}$$

where parameters $\mathbf{w}$ and b respectively denote a weight vector and a bias that can be determined in the training process through minimizing the cost function below, and $\phi(\cdot)$ refers to a nonlinear mapping to map the input vector $\mathbf{x}$ into a higher dimensional space for easily separated by a linear hyperplane as illustrated in Fig. 1.

A training sample $(\mathbf{x}_i, y_i)$ is a support vector if it holds $y_i f_{SVM}(\mathbf{x}_i) \le 1$. Let us denote $\mathbf{s}_k$ as extracted support vectors, $k \in [1, K]$, $\{\mathbf{s}_k\} \subset \mathbf{L}$ is a small subset of the training set. Hence, the SVM function becomes

$$\begin{cases} f_{SVM}(\mathbf{x}) = \sum_{k=1}^{K} K(\mathbf{x}, \mathbf{s}_k) + b \\ K(\mathbf{x}, \mathbf{s}_k) = \phi^{\mathrm{T}}(\mathbf{x})\phi(\mathbf{s}_k) \end{cases} \tag{2}$$

where $K(\cdot,\cdot)$ is denoted as a kernel function to represent the effect of the nonlinear mapping $\phi(\cdot)$ in classification.

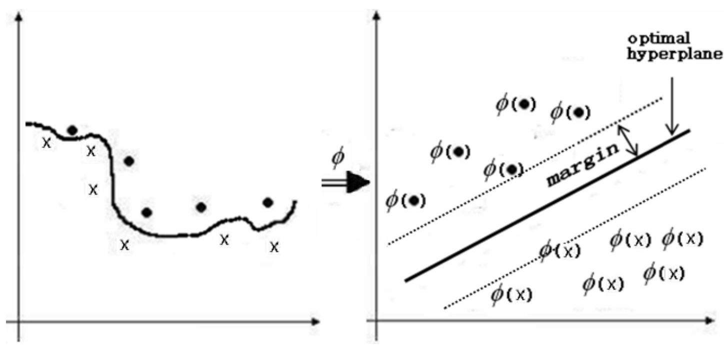


Figure 1. Illustration the concept of SVM to map a nonlinear problem to a linear separable one.

Some common used kernel functions are summarized below, including linear and two nonlinear functions. If the training samples are not linear separable, non-linear kernel functions are better choice. In addition, the associated parameters p and σ are determined automatically during the training process.

1) *Linear kernel* $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j$

2) *Polynomial kernel* $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j + 1)^p$

3) *RBF kernel* $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|^2/(2\sigma^2)}$

B. The ANN Classifier

Although there is no precise definition, ANN can be considered as an information processing system which is composed of a network of interconnected simple processing elements, i.e. neurons. Determined by the connections between these neurons and the associated parameters, ANN can exhibit complex global behavior to generate expected outputs via supervised or unsupervised learning. Inspired by the biological nervous system, the learning process is to adjust the connection strength or weights between the neurons. Each neuron forms a node in the whole network and after training each node is assigned with a determined bias or threshold. For each interconnection between two nodes, a weight is also assigned to represent the link-strength between the neurons.

For a given input vector $\mathbf{x} = (x_1, x_2, ..., x_d)^{\mathrm{T}}$ and weight vector $\mathbf{w} = (w_1, w_2, ..., w_d)^{\mathrm{T}}$, the output of a single neuron z in Fig. 2 is determined as

$$z = g(\mathbf{w}^{\mathrm{T}}\mathbf{x} - b) = g(\sum_{i=1}^{d} w_i x_i - b) \tag{3}$$

where $g(\cdot)$ is namely an activation function to decide whether the perceptron should fire or not. The sigmoid function $Sig(x) = (1+e^{-x})^{-1}$ is the most popular used activation function, others include tanh and step functions, etc.

Using the same process as to compute the output of a single neuron, the output of the whole network can be also calculated in a topological manner. This means that for each neuron its inputs from other neurons need to be computed before determining its output. As seen, the weight vector and the bias associated to each connection and each node will influence the outputted results, and they can be determined in training or learning process as follows.

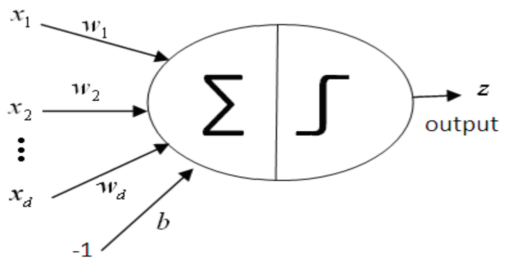


Figure 2. Illustration the effect of a single neuron.

First of all, the topology of the ANN needs to be specified, and feed-forward ANN is adopted as it has been widely applied for the classification of MCCs [17, 25, 28, 34]. A feed-forward ANN is a multi-layer perceptron (MLP) which contains three or more layers of neurons, i.e. one input layer, one output layer and at least one hidden layer. With a given training set, a specified activation function and a learning ratio γ where $\gamma \in (0,1)$, the learning process for supervised training using the well-known back-propagation algorithm can be described in the following three stages.

Firstly, the initial weights and bias are set randomly between $[-1,1]$ to attain a group of outputs $\mathbf{z}^{(t)}$ at $t=1$ referring to the first round of iteration. Then, an error function is decided as $\varepsilon(t) = \sum_{i=1}^{M}(y_i - z_i^{(t)})^2/2$ using the sum squared error between the estimated output z and the target output y. Finally, the error signal at the output units is propagated backwards through the whole network to update the weights using the gradient descent rule

$$\Delta w_{ij}(t) = -\gamma \frac{\partial \varepsilon(t)}{\partial w_{ij}} \tag{4}$$

where w_{ij} refers to a weight between the j^{th} node in a given layer and the i^{th} node in the following layer. With updated weights, we can set $t = t + 1$ to start a new iteration until the network becomes convergence. This can be measured by using a small change ratio of $\varepsilon(\cdot)$ or a given number of iterations.

C. Comparisons between SVM and ANN

As two different algorithms, SVM and ANN share the same concept using linear learning model for pattern recognition. The difference is mainly on how non-linear data is classified. Basically, SVM utilizes nonlinear mapping to make the data linear separable, hence the kernel function is the key. However, ANN employs multi-layer connection and various activation functions to deal with nonlinear problems. In fact, single layer ANN can only generates linear boundary, and the 2nd layer can combine the linear boundary together; while at least three layers are required to produce boundary of arbitrary shapes.

Using the gradient descent learning algorithm, ANN intends to converge to local minima. As a result, it suffers from the over-fitting problem. On the other hand, SVM tends to find a global solution during the training as the model complexity has been taken into consideration as a structural risk in SVM training. In other words, ANN minimizes only the empirical risk learnt from the training samples, but SVM considers both this risk and the structural risk. Consequently, the training results from SVM have better generalization capability than those from ANN. Therefore, SVM and ANN are two typical classifiers which are used to validate our balanced learning strategy as discussed in the next sections.

III. Balanced Learning

Despite the good generalization capability of SVM achieved for pattern recognition, the performance on classification of MCCs remains unsatisfied at around 80% in terms of A_z [9, 20-21, 30]. This accuracy may degrade further if the distribution of the samples is severely imbalanced [20]. Unfortunately, such imbalance distribution is widely found for MCCs classification, as usually there are much more (>4 times) benign samples than malignant ones in the training sets [20, 32]. Therefore, the performance of a single classifier may bias to the majority class and fails for correct detection of MCCs. For this purpose, we have proposed an improved strategy, namely balanced learning, to overcome this problem.

A. Strategy in Balanced Learning

To achieve balanced learning, there are two main technical streams, i.e. data level and algorithm level methods [37, 44]. At the data level, the former refer to many re-sampling solutions to balance the training data [39]. On the other hand, algorithm level solutions intend to adjust the cost function, decision threshold or the learnt probability for refined learning, such as the work reported in [40-42]. Using Bayes optimal classifier theory, it is found that individual classifier has a fundamental performance limit which makes it little better than that of the majority class [37-39]. Consequently, data-level solutions are preferred for balanced training in our paper.

Regarding data level solutions, there are two strategies in data re-sampling which include over-sampling of the minority class or under-sampling of majority class. Straightforward over- and under- sampling refer to random replication in the minority class and discarding samples in the majority class. Although under-sampling may reduce the size of the training set for efficiency, it may lead to serious problems in accurate modeling the majority class as most of data are ignored. On the contrary, random over-sampling seems to be a better solution despite of the increased training set.

Since random over-sampling may increase the likelihood of over-fitting in dealing with the duplicated samples, several smart sampling techniques have been presented such as synthetic over-sampling (SMOTE) [39]. In SMOTE, synthetic minority samples are generated via interpolation of one random sample and its nearest neighbors. Some other smart sampling techniques include one-sided selection, cluster-based over- sampling and Wilson's editing etc., and details of which can be referred to the work in [43].

B. Proposed Balanced Learning Strategy

According to the extensive experiments in [43], it is found that random sampling outperforms several smart sampling techniques and unaltered data set. However, the evaluation in [20] indicates that random over-sampling seems not improving the performance in classification of MCCs, and similar finding is concluded in detecting sentence boundaries in [44]. Besides, it is indicated that SMOTE may outperform down-sampling in certain cases [44]. These inconsistent results need to be further clarified before applying any sampling strategies to classify MCCs for improved performance.

Fig. 3 illustrates a typical two-class classification problem which contains combined linear decision boundaries. This is very common in machine learning domain and the segment of the decision boundary can also be nonlinear. For the two classes marked as circle and star shapes, two pairs of same-class samples are extracted satisfying minimum neighboring distance and marked as A-B and C-D. According to the rules of smart sampling in SMOTE, synthetic samples can be generated for balanced learning. Unfortunately, the generated samples in these cases are unreliable noisy ones which may inevitably degrade the performance of training and classification.

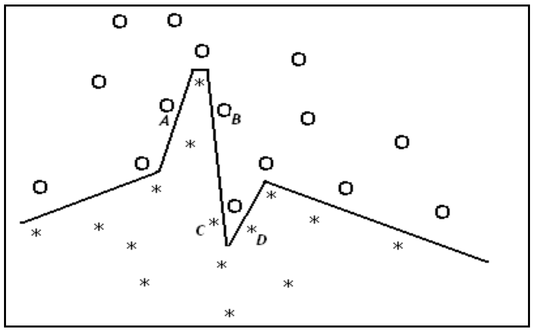


Figure 3. Illustrating a two-class problem with combined linear decision boundaries where the interpolation using SMOTE may fail for the sample pairs of A-B and C-D.

To some degree, the analysis above can explain why smart sampling behaves well in some cases. The more complex the decision boundary is, the more noisy samples may be introduced via smart sampling, and hence the worse performance may be achieved. On the other hand, smart sampling like SMOTE may work well in simpler cases such as the linear problem in detection of sentence boundary in [44].

For the classification of MCCs, it is found that associated complexity is very high with the number of support vectors above 30% of the training samples. Consequently, random over-sampling is selected. Since there are much more negative samples than positive ones, the strategy here is for each positive sample in the training set to introduce additional samples. These newly introduced samples are almost replications of the original one with minor changes (increasing or decreasing at less than 1% after normalizing the range of the feature values within [-1,1]) to one item of the feature values which is randomly determined. This helps to keep consistency between generated samples and the original ones for balanced learning and avoiding the problem caused by smart sampling as discussed above. Please note that it is assumed that the samples in our test set contain no noise instances thus the over-fitting caused by over-sampling in training can be avoided.

C. Optimized Decision Making

In our implemented ANN and SVM classifiers, the outputs are continuous values rather than binary symbols. Conventional methods use simple thresholding in decision making. If the outputs are larger than the chosen threshold, a positive sample is detected. Otherwise, it is decided as negative. However, this simple thresholding suffers uneven distribution of the training outputs and leads to poor performance. To overcome this drawback, on the contrary, optimized decision making using optimal thresholding is proposed and described as follows.

The optimal thresholding is achieved through statistical analysis of the output of the classifiers, where SVM is taken to show its principles. Let z_i denote the predicted output for a given input sample $\mathbf{x}_i$ with a target label y_i , $y_i \in \{-1,1\}$, where $z_i \in (a_0, a_1)$ and the parameters a_0 and a_1 represent respectively the lowest and the highest boundary of the output from the classifier. Then, two conditional probabilities $p(z_i \mid y_i = 1)$ and $p(z_i \mid y_i = -1)$ are obtained. For a given threshold $T \in (a_0, a_1)$, the sum of error classification rate Err is determined as

$$Err(T) = w_1 \sum_i p(z_i \mid y_i = 1, z_i \le T) + w_{-1} \sum_i p(z_i \mid y_i = -1, z_i > T) \tag{5}$$

where the weights w_1 and w_{-1} are simply set as ½. Then, an optimal threshold T_{svm} can be determined when the minimum cost of error classification is achieved, i.e.

$$T_{svm} = \arg\min(Err(T)) \tag{6}$$

Similarly, an optimal threshold T_{ANN} can be determined for the ANN classifier via statistical analysis of its outputs. Consequently, these two optimal thresholds can be used to obtain another group of classification results. The effectiveness of the proposed optimized decision making has been fully validated using the improved results as presented in Section V.

IV. Implementation and Evaluation Study

The data set and feature set as well as evaluation strategy are discussed in this section, along with some implementation details. These are essential for consistent evaluation of our proposed methodology to compare with others.

A. Data Set

To evaluation the performance of the classifiers, in total 748 suspicious MCCs are collected, which contain 633 benign and 115 malignant samples where the ratio between them is nearly 5.5. These MCCs are extracted from 295 full-field mammograms in the well-known DDSM database, where the mammography data from more than 2600 patients are scanned at 50 microns using LUMISYS [45-46]. The collected MCCs are then randomly divided into two dataset for training and testing purposes, receptively.

To detect suspicious MCC regions, optimal filtering using texture measurements is employed [32, 47]. Firstly, some pre-processing is applied to remove the influence of background and several artefacts like white/black spots and scratches. Then, optimal filtering is employed using local frequencies in terms of energy distribution extracted from mammograms. Finally, adaptive thresholding is utilized as post-processing for further robustness. Relevant details can be found in [47].

B. Feature Set

Breast microcalcifications appear as small white specks in various patterns on the mammogram [3]. Whether their clusters are malignant or benign depends on the size, shape and geographic distribution of all microcalcification regions in a cluster, i.e. if they are tightly clustered and has certain linear structure, etc. Therefore, the extracted features need to measure these properties accordingly which include the area, the scattered degree and brightness of the regions in the cluster.

In total 23 features are extracted from each of the segmented microcalcification clusters, and a list of them is summarized in Table 1. As seen, except the three single measures #1 to #3, the other 20 features in the feature set are composed of the mean and standard deviation values of ten measures. Among these 20 features, they can be categorized into three classes including i) intensity statistics (#4-#5), ii) shape features (#6-#17), and iii) linear structure features (#18-#23). Introductions to most of these features can be found in [3, 29-36].

Since about 80% of the diagnosed breast cancer cases are for women over 50 years old [1], age is a good indicator and has been widely used in the classification of MCCs [3, 32, 34]. A MCC is defined as a group of at least three microcalcifications within 1 cm^2, and the number of microcalcifications in a cluster is also an important feature [9, 32, 34-35]. The mean of the least distance of all regions in a cluster refers to the average value of inter-distance between each region and its neighboring ones [9], which can be also used to measure the scattered degree of the distribution of the microcalcifications in a cluster. In addition, the intensity measures are also useful as high intensity is expected for the white specks in MCCs [3, 30].

Table 1. List of features used for the classification of MCCs where std. means standard deviation.

No.	Meanings	Notes
1	Patients' age [3, 32, 34]	
2	The number of regions in a cluster	
3	Least inter-distance of all regions in a cluster	mean
4	The average intensities of all regions in a cluster	mean
5		std.
6	The areas of all regions in a cluster	mean
7		std.
8	The compactness of all regions in a cluster	mean
9		std.
10	The measure Fourier description FF of all regions in a cluster	mean
11		std.
12	The moment-based measure M of al regions in a cluster	mean
13		std.
14	The eccentricity of all regions in a cluster	mean
15		std.
16	The spread of all regions in a cluster	mean
17		std.
18	The average minimum std. of $r(\theta, l)$ of all regions in a cluster	mean
19		std.
20	Average std. of the minimum std. of $r(\theta, l)$ at various directions in all regions in a cluster	mean
21		std.
22	Average std. of the string of length l, starting from each point in a region at direction θ	mean
23		std.

Shape features are very important indicators in this field, in which the area (size), the compactness, Fourier descriptors, moments, eccentricity, and the spread are commonly used. The definitions of these shape features can be referred to [3, 30, 32, 34, 35]. Please note that these measures can be extracted from each microclacification region within a candidate MCC, and the mean and standard derivation values over all regions are then determined for classification purpose.

Linear structure features form another important feature set which has been widely used in detection and classification of MCCs [24, 27, 29, 32]. Linear structure here means a string of pixels (representing a line) with similar intensity along a certain direction, which can be denoted as $r(\theta, l)$ where θ and l refer respectively to the direction and the length in the linear structure. In addition, the pixel intensities on the line are higher than that of their surrounding pixels, and also the length of the line should be larger than its width. To measure the consistency of the intensities along the linear structures in a MCC, six features are extracted as summarized in Table 1 using the mean and standard deviation values of three measurements [32].

C. Evaluation Criteria

As mentioned before, all 748 MCC samples are randomly partitioned into two subsets for training and test, respectively. All the positive samples in the training set are over-sampled to enable balanced learning using SVM and ANN. The models determined are then used to classify samples in the test set. This process is repeated 10 times to overcome any bias in data partition. The average performance over these 10 times is taken as a final result for evaluations.

For a two-class problem, let us denote TP and TN as correctly classified positive and negative samples, FP and FN for incorrectly classified positive and negative samples, i.e. false alarms and missed positives. Several metrics can be determined for quantitative evaluations as follows.

$$TP_{rate} = Recall = TP/(TP + FN) \tag{7}$$

$$Precision = TP/(TP + FP) \tag{8}$$

$$FP_{rate} = 1 - Specificity = FP/(TN + FP) \tag{9}$$

To enable a single measure of performance, a F_1 measurement is also popularly used as defined below.

$$F_1 = \frac{2Recall * Precision}{Recall + Precision} \tag{10}$$

Receiver operating characteristic (ROC) analysis and its variants are commonly used for quantitative evaluations of classifiers, especially for the detection and classification of MCCs [3]. In ROC analysis, TP vs. FP rates are adopted. Under ROC analysis, the area under the ROC curve Az is also used as an important evaluation criterion [3], where $Az = 1$ indicates an ideal case with $TP_{rate} = 100\%$ and $FP_{rate} = 0$.

For the ANN classifier, the number of nodes in the hidden layer is empirically set as 15 for the better results achieved. The training process stops when the training performance keeps unchanged over a long time, say more than 4000 iterations. The performance is measured using the F_1, and the parameters which yield the highest F_1 value is stored and used for testing. In addition, the RBF kernel has been adopted in our SVM implementation as it can generate particular good results.

V. Results and Discussions

In this section, comprehensive experimental results from ANN and SVM classifiers are presented for the classification of benign and malignant MCCs. Quantitative evaluations are used to validate the effectiveness of our proposed method including balanced learning and optimal decision making.

A. Performance of Balanced Learning

First of all, the performance of balanced learning is compared with those training with the original data, and we set the training ratio as 80%, i.e. 80% of the samples for training and 20% for testing. The ROC curves are plotted in Fig. 4 to show the performances in training and testing of SVM and ANN with or without balanced learning, respectively, where several facts can be summarized as follows.

Firstly, in general training results are much better than testing ones, especially for the results from ANN, which has validated our analysis that ANN tends to produce minimum errors. Secondly, it is surprisingly to see that ANN outperforms SVM in both training and testing. Thirdly, balanced learning indeed can yield better results despite of a little higher false positive rate. Regarding training, it has generated significant higher recall rate for SVM and slightly higher recall rate for ANN though its recall rate without balanced learning is already high enough. For testing, balanced learning produces much improved results for ANN but limited improvements for SVM. Finally, it is worth noting that balanced learning seems inferior to unbalanced learning only if the false positive rate is less than 3%, although the testing results for the two classifiers are different. In fact, the results for ANN have no much change, but the false positive rate becomes more than 20% for ANN to enable balanced learning to achieve a better recall rate. As a higher recall rate is always desirable in such applications, balanced learning still proves to be better than unbalanced ones.

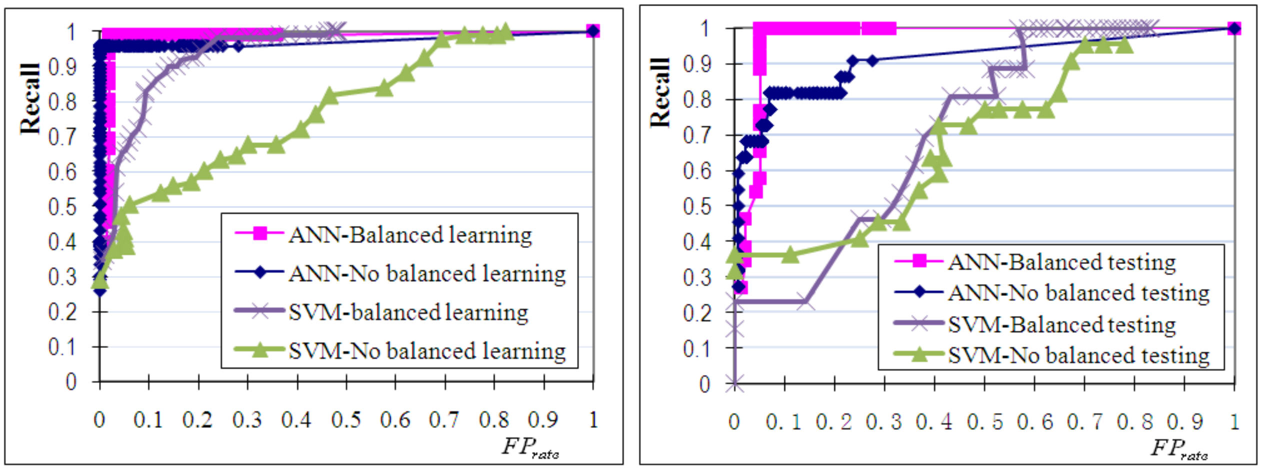


Figure 4. ROC curves of training and testing performances from SVM and ANN with or without balanced learning.

Quantitative comparisons of the results from ANN and SVM are respectively reported in Table 2 and Table 3, and no optimized decision making is applied in the testing. First of all, with balanced learning, the testing performance in terms of F_1 and Az can be significantly improved. This conclusion is different from the work in [20], which has validated the effectiveness of balanced learning. In addition, it is worth noting that both the training and testing performance from ANN are much better than those from SVM, and further analysis is presented below.

Table 2. Training and testing results from ANN with or without balanced learning.

	No balanced learning		**Balanced learning**	
	Training	Testing	Training	Testing
$Recall$	0.957	0.727	0.990	1.000
$Precision$	1.000	0.696	0.981	0.722
$Specificity$	1.000	0.945	0.980	0.928
F_1	0.978	0.711	0.985	0.839
Az	0.979	0.836	0.985	0.964

In Table 2, it is found that the training performance is high in terms of all the five measures no matter balanced training is used or not. This has indicated that ANN is capable of model the problem accurately. However, the testing performance under balanced training is much better, in which an improvement of 12.8% are achieved in both F_1 and Az measurements. Although in training almost the same values of F_1 and Az are obtained, smaller F_1 values in testing are yielded. This is caused by lower $Precision$ values due to false positives. Due to the severe imbalancement of the data used for testing, a high $Specificity$ value is still achieved under these false alarms to yield a higher Az measurement. In other words, the ratio between the number of false alarms to the number of malignant samples is much larger than the ratio between it to the number of benign samples, and this has led to lower F_1 but higher Az values.

In Table 3, the results from SVM is some different. Firstly, the training results from balanced learning are much better than those without balanced learning, and the improvements in terms of F_1 and Az are about 46% and 21%, respectively. Secondly, the testing results from balanced learning are about 20% better in F_1 and Az measurements than those without balanced learning. Thirdly, balanced learning has improved the $Recall$ rate by about 60%, which means massively reduction of missing detection although more false alarms are introduced to degrade the $Precision$ value from 1 to 0.479.

Table 3. Training and testing results from SVM with or without balanced learning.

	No balanced learning		**Balanced learning**	
	Training	Testing	Training	Testing
$Recall$	0.301	0.273	0.899	0.885
$Precision$	0.966	1.000	0.947	0.479
$Specificity$	0.998	1.000	0.830	0.819
F_1	0.459	0.429	0.922	0.622
Az	0.650	0.637	0.865	0.852

B. Performance of Optimized Decision Making

In Table 4, the results using our proposed optimized decision making in both ANN and SVM classifiers are given. Again, we select 80% of samples for training and 20% for testing. By comparing these results with those in Table 2 and Table 3, we can clearly find several facts which are summarized as follows.

- Without balanced learning, optimized decision making contributes for the ANN about 1% in F_1 and 3.4% in Az measurements. For the SVM, the contributions are 22.4% and 18.3%, respectively.
- Regarding balanced learning, the improvements for ANN are 4.2% in F_1 and 1.1% in Az even the original Az value is as high as 96.4%. However, SVM gains 4.5% in F_1 but 0.6% degradation in Az measurements.

This on one hand has fully validated the effectiveness of the proposed strategy for optimized decision making in terms of an improved F_1. On the other hand, significant improvements have achieved for the SVM classifier when balanced learning is not employed. Nevertheless, the results from ANN remain better than those from SVM in both F_1 and Az measurements.

Table 4. Testing results from ANN and SVM under optimized decision making with or without balanced learning.

	No balanced learning		**Balanced learning**	
	ANN	SVM	ANN	SVM
Recall	0.818	0.727	1.000	0.808
Precision	0.643	0.593	0.788	0.568
Specificity	0.921	0.913	0.949	0.884
F_1	0.720	0.653	0.881	0.667
Az	0.870	0.820	0.975	0.846

C. Performance under Various Training Ratios

In the tests above, the training ratio is fixed at 80%. In this group of tests, the performance under various training ratios is compared. Under various training ratios, the training results and two testing results with or without optimized decision making are evaluated in terms of F_1 and Az measurements. These results are illustrated in Fig. 5 and Fig. 6, where Test2 denotes results using optimal decision making. In total there are three pair of curves in each figure in which one is from training and the other two from testing without or with optimal decision making. Each pair of curves is plotted using the training ratio (changed from 50% to 90%) vs. performance of F_1 and Az measurements and they are further discussed as follows.

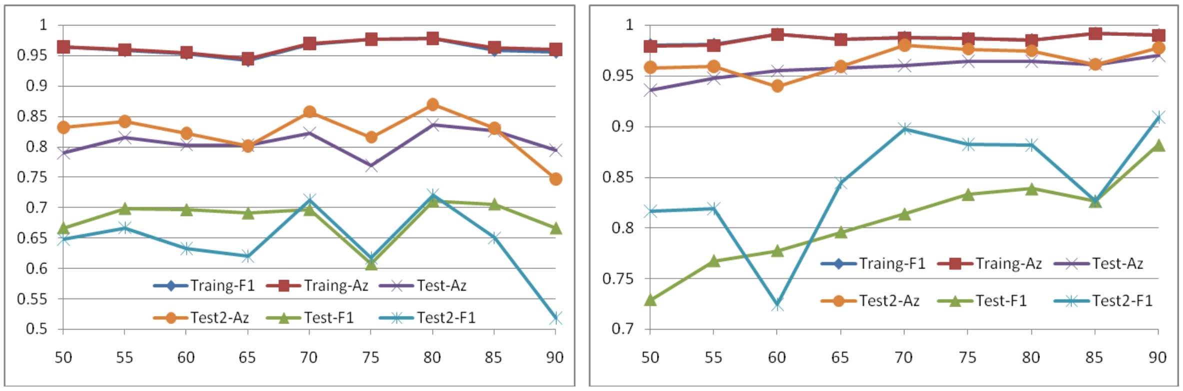


Figure 5. Training and testing results from ANN using plots of training ratio (x-axis) vs. F_1 and Az measurements, where the top and the bottom plots refer respectively to results without or with balanced learning.

In Fig. 5, the training and testing rersults from ANN are illustrated. Firstly, the F_1 and Az measurements from training with or without balanced learning are very close to each other and appear insensitive to the training ratio, which again shows that ANN is capable in accurate modelling the problem. Secondly, the testing results using balanced learning are much better that those without balanced learning. Thirdly, in most cases optimized decision making produces better results in F_1 and Az measurements when balanced learning is employed, except the result at the training ratio of 60%. When balanced learning is not used, however, better F_1 measurement can be only yielded when the training ratio is between 70% and 80%. In addition, better Az measurement can always be achieved from optimized decision making even the balanced learning is skipped.

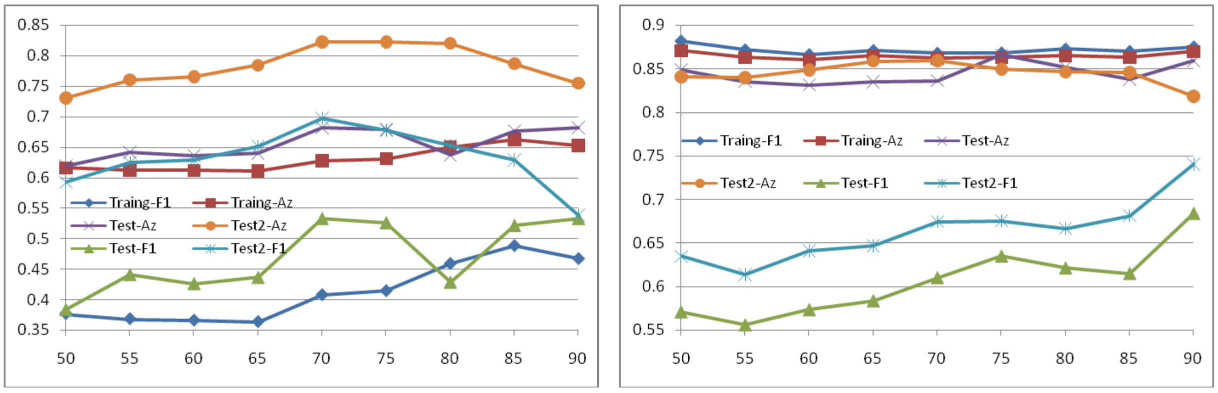


Figure 6. Training and testing results from SVM using plots of training ratio (x-axis) vs. F_1 and Az measurements, where the top and the bottom plots refer respectively to results without or with balanced learning.

The results from SVM as illustrated in Fig. 6 show some different facts. Firstly, the training performance from SVM is not superior to the testing results as shown from ANN. One possible reason is the so-called high generalization capacity as it tends to avoid overfitting hence the relative poorer trainuing performance. Secondly, balanced learning is useful in yielding better training and testing results, especially when the training ratio is around 75%. Thirdly, under balanced learning optimized decision making can significantly improve F_1 measurement but such improvement on Az measurement is quite limited and can only be found when the training ratio is between 55% and 70%. In addition, optimized decision making helps to gain apparent improvements in both F_1 and Az measurement when balanced learning is not used. This is because that balanced learning has reduced the diversity degree of the training samples. As a result, there is very limited space for optimized decision making for further improvement. On the contrary, there is much large space for optimized decision making in improving the results from high diverse data when balanced learning is not used.

D. Computational Complexity

In comparison with conventional ANN and SVM, the proposed balanced learning and optimized decision making do need additional computations. As optimized decision making does not involve in the training iterations, it can be simply ignored. In the following, we will analyze the effect of balanced learning in such a context.

Since more training samples are introduced in balanced learning, it costs more time in learning the model. Let $N = N_0 + N_1$ be the total samples, where N_0 and N_1 denote respectively the number of negative and positive samples satisfying $N_0 = KN_1, K > 2$. After oversampling, $(K-1)N_1$ new positive samples are produced and in total we have $2N_0$ training samples. This equals to $2NK/(K+1)$ and less than $2N$, which has indicated that the number of samples under balanced learning is less than twice of the number of the original samples. Under same number of iterations, the training time should no more double of the one without balanced learning.

In addition, it is found that balanced learning needs less number of iterations to converge, which is about 77% of the one required for training without balanced learning. This fast converging might due to the improved distributions of training samples from our balanced learning. Consequently, the increased complexity under balanced learning is

$$2K/(K+1)*77\%-1=0.54-1.54/(K+1) \tag{11}$$

This indicates a maximum of 54% additional computing burden, which is totally acceptable for the benefit of much improved performance.

VI. CONCLUSIONS

In this paper, balanced learning with optimized decision making is proposed for classification of benign and malignant MCCs in mammograms. The proposed methodology has been tested on two common used machine learning approaches including ANN and SVM. The experiments are conducted on 748 samples extracted from the well-known DDSM database, and the main findings can be summarised as follows.

Firstly, balanced learning indeed has significantly improved the classification accuracy, and an average gain of more than 10% can be achieved for the two classifiers in terms of both F_1 and Az measurements. Secondly, optimized decision making produces improved results in F_1 and Az for ANN no matter balanced learning is used or not. For SVM, however, more than 18% of improvements in F_1 and Az can only be found without balanced learning. Otherwise, improved F_1 but slightly degraded Az are produced. Thirdly, the overall results from ANN are much better than those from SVM, which is some different from the work reported [9, 20-22, 30]. Fourthly, a training ratio between 70% and 80% is suggested due to the various performances under different training ratios. Finally, it is found that the suggested balanced training will only bring up to 54% of additional computation load, a tolerable cost for the much improved performance. Further investigations include introducing feature selection approaches for improved efficiency as well as reducing false alarms for more robustness.

ACKNOWLEDGMENT

The authors would like to thank Dr. Z.-Q. Wu for his helps in data collection and feature extraction in this work.

REFERENCES

[1] Cancer Research UK: Key Facts on Breast Cancer, http://info.cancerresearchuk.org/cancerstats/types/breast/, 2009.

[2] American Cancer Society: Cancer facts and figures, http://www.cancer.org, 2009.

[3] H.D. Cheng, X. Cai, X. Chen, L. Hu, X. Lou, "Computer-aided detection and classification of microcalcifications in mammograms: a survey," *Pattern Recog.*, 36(12) (2003) 2967-2991.

[4] H.D. Cheng, X.J. Shi, R. Min, L.M. Hu, X.P. Cai, H.N. Du, "Approaches for automated detection and classification of masses in mammograms," *Pattern Recog.*, 39(4) (2006) 646-668.

[5] G. Torheim, F. Godtliebsen, D. Axelson, K.A. Kvistad, O. Haraldseth, P.A. Rinck, "Feature extraction and classification of dynamic contrast-enhanced T2*-weighted breast image data," *IEEE Trans. Med. Imag.*, 20(12) (2001) 1293-1301.

[6] T.E. Kerner, K.D. Paulsen, A. Hartov, S.K. Soho, S.P. Poplack, "Electrical impedance spectroscopy of the breast: clinical imaging results in 26 subjects," *IEEE Trans. Med. Imag.*, 21(6) (2002) 638-645.

[7] S. Joo, Y.S. Yang, W.K. Moon, H.C. Kim, "Computer-aided diagnosis of solid breast nodules: use of an artificial neural network based on multiple sonographic features," *IEEE Trans. Med. Imag.*, 23(10) (2004) 1292-1300.

[8] T.D. Tosteson, B.W. Pogue, W. Demidenko, T.O. McBride, K.D. Paulsen, "Confidence maps and confidence intervals for near infrared images in breast cancer," *IEEE Trans. Med. Imag.*, 18(12) (1999) 1188-1193.

[9] L. Wei, Y. Wei, Y. Yang, R.M. Nishikawab, "Microcalcification classification assisted by content-based image retrieval for breast cancer diagnosis," Pattern Recog., 42(6) (2009) 1126-1132. [45]

[10] J. Grim, P. Somol, M. Haindl, J. Danes, "Computer-aided evaluation of screening mammograms based on local texture models," IEEE Trans. Image Proc., 18(4) (2009) 765-773.

[11] A.E. Hassanien, "Fuzzy rough sets hybrid scheme for breast cancer detection," Image and Vision Comput., 25(2) (2007) 172-183. [31]

[12] E.A. Rashed, I.A. Ismail, S.I. Zaki, "Multiresolution mammogram analysis in multilevel decomposition," Pattern Recog. Letters, 28(2) (2007) 286-292.

[13] K. Thangavel, M. Karnan, A. Pethalakshmi, "Performance analysis of rough reduct algorithms in mammogram," ICGST-GVIP Journal, 5(8) (2005) 13-21.

[14] K.Thangavel, M. Karnan, "Computer aided diagnosis in digital mammograms: detection of microcalcifications by meta heuristic algorithms," ICGST-GVIP Journal, 5(7) (2005) 41-55. [50]

[15] H. Soltanian-Zadeha, F. Rafiee-Radc, S. Pourabdollah-Nejad, "Comparison of multiwavelet, wavelet, Haralick, and shape features for microcalcification classification in mammograms," Pattern Recog., 37(10) (2004) 1973-1986. [49]

[16] H.D. Cheng, J. Wang, X. Shi, "Microcalcification detection using fuzzy logic and scale space approaches," Pattern Recog., 37(2) (2004) 363-375.

[17] M. De Santo, M. Molinara, F. Tortorella, M. Vento, "Automatic classification of clustered microcalcifications by a multiple expert system," Pattern Recog., 36(7) (2003) 1467-1477. [40]

[18] P. Heinlein, J. Drexl, W. Schneider, "Integrated wavelets for enhancement of microcalcifications in digital mammography," IEEE Trans. Med. Imag., 22(3) (2003) 402-413.

[19] L. Wei, Y. Yang, R.M. Nishikawa, et al, "Relevance vector machine for automatic detection of clustered microcalcifications," IEEE Trans. Med. Imag., 24(10) (2005) 1278-1285.

[20] L. Wei, Y. Yang, R.M. Nishikawa, et al, “A study on several machine-learning methods for classification of malignant and benign clustered microcalcifications,” IEEE Trans. Med. Imag., 24(3) (2005) 371-380. [46]

[21] I. El-Naqa, Y. Yang, N.P. Galatsanos, et al, “A similarity learning approach to content-based image retrieval: application to digital mammography,” IEEE Trans. Med. Imag., 23(10) (2004) 1233-1244. [47]

[22] I. El-Naqa, Y. Yang, M.N. Wernick, et al., “A support vector machine approach for detection of microcalcifications,” IEEE Trans. Med. Imag., 21(12) (2002) 1552-1563. [47]

[23] S. Sentelle, C. Sentelle, M.A. Sutton, “Multiresolution-based segmentation of calcifications for the early detection of breast cancer,” Real-Time Imag., vol. 8, no. 3, pp. 237 – 252, June 2002.

[24] R. Zwiggelaar, S.M. Astley, C.R.M. Boggis, C.J. Taylor, “Linear structures in mammographic images: detection and classification,” IEEE Trans. Med. Imag., 23(9) (2004) 1077-1086. [37]

[25]P. Sajda, C. Spence, J. Pearson, “Learning contextual relationships in mammograms using a hierarchical pyramid neural network,” IEEE Trans. Med. Imag., 21(3) (2002) 239-250. [35]

[26] L. Hadjiiski, B. Sahiner, H.-P. Chan, et al, “Classification of malignant and benign masses based on hybrid ART2LDA approach,” IEEE Trans. Med. Imag., 18(12) (1999) 1178-1187. [41]

[27] R. Nakayama, Y. Uchiyama, K. Yamamoto, et al, “Computer- aided diagnosis scheme using a filter bank for detection of microcalcification clusters in mammograms,” IEEE Trans. Biomed. Eng., 53(2) (2006) 273-283. [33]

[28] B. Verma, J. Zakos, “A computer-aided diagnosis system for digital mammograms based on fuzzy-neural and feature extraction techniques,” IEEE Trans. Inform. Technol. Biomed., 5(1) (2001) 46-54. [42]

[29] J. Ge, B. Sahiner, L.M. Hadjiiski, H.-P. Chan, J. Wei, M.A. Helvie, C. Zhou, “Computer aided detection of clusters of microcalcifications on full field digital mammograms,” Med. Phys., 33(8) (2006) 2975-2988. [38]

[30] A. Papadopoulosab, D.I. Fotiadisb, A. Likasb, “Characterization of clustered microcalcifications in digitized mammograms using neural networks and support vector machines,” Artificial Intelligence in Medicine, 34(2) (2005) 141-150. [43]

[31] M. Nemoto, A. Shimizu, Y. Hagihara, et al, “Improvement of tumor detection performance in mammograms by feature selection from a large number of features and proposal of fast feature selection method,” Systems and Computers in Japan, 37(12) (2006) 56-68.

[32] Z.Q. Wu, J. Jiang, Y.H. Peng, “Effective features based on normal linear structures for detecting microcalcifications in mammograms,” in Proc. ICPR, pp. 1-4, 2008. [39]

[33] K. Thangavel, M. Karnan, R. Sivakumar, A.K. Mohideen, “Automatic detection of microcalcification in mammograms– a review,” ICGST-GVIP Journal, 5(5) (2005) 31-61.

[34] M. Kallergi, “Computer-aided diagnosis of mammographic microcalcification clusters,” Med. Phys., 31(2) (2004) 314-326. [44]

[35] A. Papadopoulos, D.I. Fotiadis, L. Costaridou, “Improvement of microcalcification cluster detection in mammography utilizing image enhancement techniques,” Computers in Biology and Medicine, 38(10) (2008) 1045-1055.

[36] L. Bocchi, G. Coppini, J. Nori, G. Valli, “Detection of single and clustered microcalcifications in mammograms using fractals models and neural networks,” Med. Eng. & Phy., 26(4) (2004) 303-312. [36]

[37] S. Kotsiantis, D. Kanellopoulos, P. Pintelas, “Handling imbalanced datasets: a review,” GESTS Int. Trans. Computer Science and Eng., 30(1) (2006) 25-36.

[38]. C. Drummond, R.C. Holte, “Severe class imbalance: why better algorithms aren’t the answer,” LNCS, vol. 3655, pp. 539-546, 2005.

[39] N.V. Chawla, K.W. Bowyer,, L.O. Hall, W.P. Kegelmeyer, “SMOTE: synthetic minority over-sampling technique,” J. Artif. Intell. Res., 16 (2002) 321-357.

[40] B.Y. Tang, Y.-Q. Zhang, N.V. Chawla, S. Krasser, “SVMs modelling for highly imbalanced classification,” IEEE Trans. System Man and Cybernetics Part B, 39(1) (2009) 281-288.

[41] X. Hong, S. Chen, C. Harris, “A kernel-based two-class classifier for imbalanced data sets,” IEEE Trans. Neural Netw., 18(1) (2007) 28-42. [51]

[42] K. Huang, H. Yang, I. King, M.R. Lyu, “Maximizing sensitivity in medical diagnosis using biased minimax probability machine,” IEEE Trans. Biomed. Eng., 53(5) (2006) 821–831. [52]

[43] J.V. Hulse, T.M. Khoshgoftaar, A. Napolitano, “Experimental perspectives on learning from imbalanced data,” in Proc. 24th Int. Conf. Machine Learning (ICML), vol. 227, pp. 935-942, 2007.

[44] Y. Liu, N.V. Chawla, M.P. Harper, E. Shriberg, A. Stolcke, “A study in machine learning from imbalanced data for sentence boundary detection in speech,” Computer Speech Language, 20(4) (2006) 468-494.

[45] M. Heath, K. Bowyer, D. Kopans, R. Moore, W.P. Kegelmeyer, “The digital database for screening mammography,” in Proc. the 5th Int. Workshop on Digital Mammography, pp. 212-218, 2001.

[46] M. Heath, K. Bowyer, D. Kopans, W.P. Kegelmeyer, R. Moore, K. Chang, S. MunishKumaran, “Current status of the digital database for screening mammography,” in Proc. the 4th Int. Workshop on Digital Mammography, pp. 457-460, 1998.

[47] Z.Q. Wu, J. Jiang, Y.H. Peng, T.O. Gulsrud, “A filter-based approach towards automatic detection of microcalcification,” LNCS, vol. 4046, pp. 424-432, 2006.