# Detecting and Tracking People in Real-time

by
Amanjit Dulai

A Thesis submitted in fulfilment of requirements for
the degree of Doctor of Philosophy and Diploma of
Imperial College London

November 2013

Communications and Signal Processing Group
Department of Electrical and Electronic Engineering
Imperial College London

# Abstract

The problem of detecting and tracking people in images and video has been the subject of a great deal of research, but remains a challenging task. Being able to detect and track people would have an impact in a number of fields, such as driverless vehicles, automated surveillance, and human-computer interaction. The difficulties that must be overcome include coping with variations in appearance between different people, changes in lighting, and the ability to detect people across multiple scales. As well as having high accuracy, it is desirable for a technique to evaluate an image with low latency between receiving the image and producing a result.

This thesis explores methods for detecting and tracking people in images and video. Techniques are implemented on a desktop computer, with an emphasis on low latency. The problem of detection is examined first. The well established integral channel features detector is introduced and reimplemented, and various novelties are implemented in regards to the features used by the detector. Results are given to quantify the accuracy and the speed of the developed detectors on the INRIA person dataset. The method is further extended by examining the prospect of using multiple classifiers in conjunction. It is shown that using a classifier with a version of the same classifier reflected in the vertical axis can improve performance. A novel method for clustering images of people to find modes of appearance is also presented. This involves using boosting classifiers to map a set of images to vectors, to which $K$-means clustering is applied. Boosting classifiers are then trained on these clustered datasets to create sets of multiple classifiers, and it is demonstrated that these sets of classifiers can be evaluated on images with only a small increase in the running time over single classifiers.

The problem of single target tracking is addressed using the mean shift algorithm. Mean shift tracking works by finding the best colour match for a target from frame to

frame. A novel form of mean shift tracking through scale is developed, and the problem of multiple target tracking is addressed by using boosting classifiers in conjunction with Kalman filters. Tests are carried out on the CAVIAR dataset, which gives representative examples of surveillance scenarios, to show the performance of the proposed approaches.

# Acknowledgements

I would first like to thank my supervisor Dr. Tania Stathaki for her guidance and patience throughout the past four years. I am also grateful that she has allowed me to pursue research in an area that I have found difficult, but ultimately rewarding.

I would also like to thank Dr. Nelson Yung for hosting me at the University of Hong Kong during one summer, and Dr. Henry Ngan for helping me settle in.

I am thankful to the many friends I have made within the Communications and Signal Processing Group, particularly Dahir, Harry, Marc, Cyrus, Ali and Zaid.

I must also thank my mum for her continued support throughout my studies, and my sisters Mindy and Kiren.

I am very fortunate to have been supported by the University Defence Research Centre (UDRC) throughout my studies.

# Contents

# List of Figures

# List of Tables

# Statement of Originality

I declare that this thesis is my own work. Where other sources of information have been used, they have been acknowledged. This thesis was not and will not be submitted to any other institution to fulfil the requirements of a degree.

# Copyright Declaration

# List of Symbols

| | |
|---|---|
| $(\cdot)^{\mathsf{T}}$ | Transpose |
| $\|\cdot\|$ | Euclidean norm |
| $*$ | Convolution operator |
| $\mathbb{R}$ | The set of real numbers |
| $p(\cdot)$ | Probability distribution |
| $K$ | Kernel function in Chapters 2 and 5 |
| $\mathbf{H}$ | Bandwidth matrix in Chapters 2 and 5 |
| $k$ | Kernel profile function in Chapters 2 and 5 |
| $G$ | Shadow kernel in Chapters 2 and 5 |
| $\mathbf{m}_g(\mathbf{x})$ | Mean shift vector at $\mathbf{x}$ |
| $w$ | Mean shift weights |
| $\mathsf{z}_t$ | Measurement vector at time $t$ |
| $\mathsf{x}_t$ | State vector at time $t$ |
| $\mathsf{F}$ | Process model matrix |
| $\mathsf{H}$ | Observation model matrix |
| $\mathbb{E}[\cdot]$ | Expectation operator |
| $\mathbf{I}$ | A $m_1 \times m_2 \times 3$ RGB image |
| $\mathbf{I}^c$ | A $m_1 \times m_2$ RGB image channel |
| $\mathbf{I}(j,k,c), \mathbf{I}^c(j,k)$ | An image entry at column $j$, row $k$, and channel $c$ |
| $\mathbf{G}_x^c$ | A $m_1 \times m_2$ gradient image for the $x$-direction |
| $\mathbf{G}_y^c$ | A $m_1 \times m_2$ gradient image for the $y$-direction |
| $\mathbf{G}_{mag}(\mathbf{I})$ | A $m_1 \times m_2$ gradient magnitude image |

| | |
|---|---|
| $\mathbf{G}_\theta(\mathbf{I})$ | A $m_1 \times m_2$ gradient orientation image |
| $\mathbf{\Psi}^c(\mathbf{I}, j, k), \mathbf{\Psi}(\mathbf{I}, j, k, c)$ | Gradient histogram entry at column $j$, row $k$, and channel $c$ |
| $\mathbf{\Lambda}^c(\mathbf{I}, j, k), \mathbf{\Lambda}(\mathbf{I}, j, k, c)$ | Integral histogram entry at column $j$, row $k$ and channel $c$ |
| $\mathbf{\Phi}(\mathbf{I})$ | A $m_1 \times m_2 \times m_3$ feature map |
| $\mathbf{\Phi}^c(\mathbf{I})$ | A $m_1 \times m_2$ feature map channel |
| $\mathbf{\Phi}(\mathbf{I}, j, k, c), \mathbf{\Phi}^c(\mathbf{I}, j, k)$ | A feature map entry at column $j$, row $k$ and channel $c$ |
| $\boldsymbol{\rho}$ | A vector of parameters for a feature |
| $g(\mathbf{I}, \boldsymbol{\rho})$ | A feature value |
| $\mathbf{g}(\mathbf{I})$ | A vector of features from image $\mathbf{I}$ |
| $\gamma$ | A quantised feature value |
| $\mathbf{\Gamma}(\mathbf{I})$ | A vector of quantised features from image $\mathbf{I}$ |
| $y$ | Label |
| $\mathcal{S}$ | A training set |
| $D(i)$ | Weight for training example $i$ |
| $\epsilon$ | Error for weak classifier |
| $h$ | A weak classifier |
| $f$ | A boosting classifier |
| $\alpha$ | Weak classifier coefficient |
| $\Theta_u$ | Set of thresholds for feature $u$ |
| $\mathcal{S}_\mathbf{I}$ | Image training set |
| $\mathcal{S}_\mathbf{g}$ | Feature training set |
| $\mathcal{S}_\mathbf{\Gamma}$ | Quantised feature training set |
| $H(\mathbf{I})$ | Score from a classifier on an image $\mathbf{I}$ |
| $\boldsymbol{\alpha}(\mathbf{I})$ | Vector of weak classifier responses for $\mathbf{I}$ |
| $\hat{\mathbf{q}}$ | Reference colour histogram |
| $\hat{\mathbf{p}}(\mathbf{I}, \mathbf{y})$ | Colour histogram of $\mathbf{I}$ in region centred on $\mathbf{y}$ |
| BC | Bhattacharyya coefficient |

# List of Abbreviations

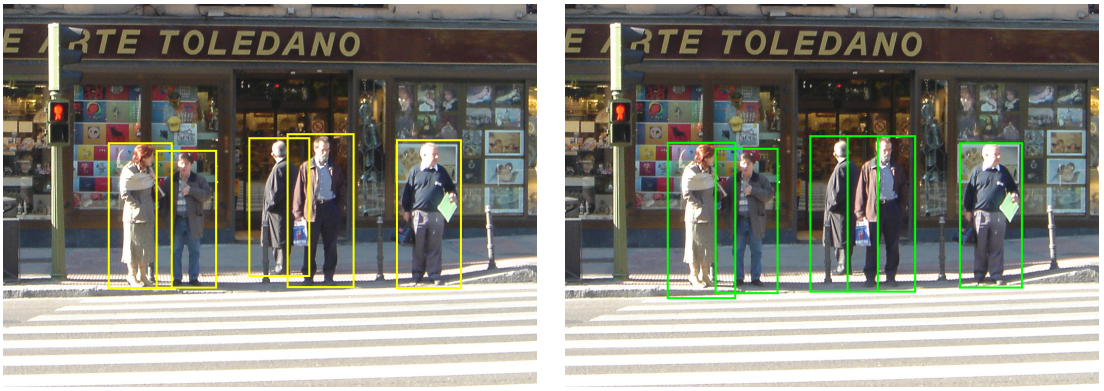
| | |
|---|---|
| AdaBoost | Adaptive Boosting |
| CCTV | Closed Circuit Television |
| CIE | International Commission on Illumination |
| CIELUV | CIE 1976 (L*, u*, v*) colour space |
| DET | Detection Error Trade-off |
| EKF | Extended Kalman Filter |
| HOG | Histogram of Oriented Gradients |
| NMS | Non-Maximum Suppression |
| RAM | Random Access Memory |
| RGB | Red-Green-Blue |
| SVM | Support Vector Machine |
| UKF | Unscented Kalman Filter |

# Chapter 1

# Introduction

The idea of creating machines that possess intelligence is one that has captured the imagination of a variety of thinkers, from philosophers to engineers. The widespread use of computers has revolutionised nearly every aspect of day to day life, and has led to the ability to capture, store and process more information than ever before. Computer vision is the field of research that concerns the intelligent processing of visual information, and has had considerable success in applications such as industrial inspection [1], character recognition [2] and even the adjudication of sporting events [3]. However, for many challenging problems, existing algorithms are not capable of achieving anything close to the accuracy that could be achieved by a typical person engaging in a similar task. Worse still, the best performing algorithms are usually slow to run, as improving the accuracy involves increasing the computational cost of a technique.

One task that has been the focus of a great deal of research is that which shall be referred to in this thesis as *object detection*. This is the problem of finding the location and extent of all instances of a particular object class within an image. An object's class in this context is a noun or other description that encompasses a variety of objects possessing some degree of visual similarity, such as "person", "car" or "bike". Different classes will display different degrees of internal similarity, and some will be easier to detect than others. The extent of an object instance is typically estimated with a *bounding box*, which is simply a rectangular region within an image containing an object instance. Figure 1.1a shows bounding boxes for the object class "person" that have been annotated manually for the purpose of creating a ground truth. Figure 1.1b shows




(a) (b)

Figure 1.1: The extent of an object is often represented by bounding boxes. (a) Ground truth annotations of bounding boxes are shown in yellow. (b) Bounding boxes output by one of the detectors developed in this thesis are shown in green.

bounding boxes output by one of the detectors developed in this thesis. As can be seen, an object detector must negotiate several difficulties, such as cluttered backgrounds, partially overlapping object instances and variations in object pose. It is important to note that object detection as defined here is different from the problem of finding a particular object within an image, as the former problem requires the detection of a broad class of objects, while the latter requires the detection of only a single specific object.

Another problem of great interest is that of object tracking. The precise definition of this problem varies depending on the scenario under consideration. In a situation where there is only a single target to be tracked, there are a number of methods that work by finding the best match for a target from frame to frame. However, the problem can also be formulated as a filtering problem, where the aim is to estimate a target's position over time given noisy measurements and some prior knowledge concerning the target's motion. Furthermore, in multi-target tracking, multiple measurements are generated per time instant, and measurements corresponding to a single target must be associated across time. It is important to point out that in both of the previously mentioned formulations of the tracking problem, it is assumed that a method for generating measurements is available. In other applications where tracking is used, such as radar and sonar, obtaining measurements is relatively straightforward. However, in computer vision, if we wished to obtain measurements for a specific class of objects (for example,

the position and extent of all people within an image), we need a solution to the object detection problem. Thus, the problem of object tracking is closely related to that of object detection.

## 1.1 Objectives

This thesis examines the problem of being able to detect and track people in images and video in real-time. The objective is to experiment with and develop techniques for detection and tracking with a particular emphasis on speed and accuracy.

For detection, the aim is to develop techniques that are capable of achieving results that are better than the widely used Histogram of Oriented Gradients (HOG) detector [4] as measured by the Caltech Pedestrian Detection Benchmark [5] on the INRIA Person Dataset, details of which are given in Sections 2.1.2 and 2.1.3. Another aim is to ensure that the proposed techniques are able to evaluate a $640 \times 480$ pixel image in less than a second on a desktop computer. While other platforms such as field programmable gate arrays or graphics processing units could be considered, desktop computers are cheaper and more widely available. The scope of this thesis is limited to the detection of people who are standing upright or are in near upright positions, and are mostly visible. Only instances of pedestrians that are larger than $32 \times 96$ pixels are considered, as instances at smaller resolutions are more difficult to detect due to the loss of discriminative information. A technique must be able to detect instances of people within an image at multiple scales and arbitrary translations, as shown in Figure 1.1, and must be able to cope with variation in appearance and background clutter. For this purpose, the integral channel features detector [6] is used as a starting point, as it has been demonstrated to achieve the latency requirements mentioned earlier. A number of novelties are introduced, and investigations into the effect of altering the features for the detector are presented, in terms of the impact on accuracy and speed. It is also shown that multiple detectors can be used to improve accuracy.

For tracking, the aim is to develop methods that are capable of working in real-time, and coping with issues such as changes in scale, and variation in appearance. For this purpose, the mean shift algorithm is used. Mean shift tracking works by finding the best match for a target based on a colour histogram similarity metric. A novel method for

mean shift tracking through scale is developed. Multiple target tracking with boosting classifiers is also examined. Tests are carried out on the CAVIAR dataset, which gives a good representation of surveillance scenarios.

## 1.2 Motivation

Being able to detect and track people in real-time would have a variety of applications. It would open up a number of new possibilities for Closed Circuit Television (CCTV), such as being able to estimate the number of people passing through an area in a certain window of time, or collecting statistics on the trajectories travelled by people. Tracking people is often a prerequisite for other problems in visual surveillance, such as person identification [7], gait recognition [8] and behaviour analysis [9].

Another emerging area of application is in driverless vehicles and driver assistance systems. The use of computer vision in automotive applications has grown dramatically in recent times, and some pedestrian and cyclist detection systems are now incorporated into vehicles available to the public [10]. Driverless vehicles have now begun to emerge as a viable technology [11], [12], [13], and need to be able to detect where people are in relation to the vehicle's position in real-time to avoid collisions.

Being able to track people in real-time would also have applications in the field of human-computer interaction. Altering the methods that people use to interact with computers has been a strong trend in consumer electronics over the past few years [14]. Existing techniques have limited range or require expensive hardware [15]. The techniques explored in this thesis are designed to work over several scales, and use images captured by standard RGB cameras which are cheap and ubiquitous.

Finally, techniques that are able to detect and track people in real-time could be generalised to work on other object classes, such as vehicles. Different object classes will possess different aspects of visual variation, and this must be taken into account when implementing a technique. For example, the appearance of vehicles varies greatly depending on the angle from which they are viewed, and so vehicle detectors often incorporate multiple detectors for different view angle ranges [16], but the viewing angle is less of an issue when training a pedestrian detector. By extending object detection to other classes, it becomes possible to exploit the context of an object to aid detection [17].

For example, in a system that detects both pedestrians and vehicles, it is possible to exploit the spatial relationships that exist between these classes to improve the detection accuracy.

## 1.3 Original Contributions

The following original contributions are made in this thesis:

- An investigation into the effects of altering the layout of orientation bins for gradient features, and using contrast sensitive features in a variety of combinations with other features in boosting classifiers is presented in Chapter 3.
- The idea of using boosting classifiers in conjunction with their vertically reflected counterparts is presented in Chapter 4.
- The use of the weak classifiers within a boosting classifier to map images to vectors summarising visual information, and then clustering these vectors to find modes of appearance for people is presented in Chapter 4.
- The idea of combining multiple boosting classifiers trained on clustered image sets is outlined in Chapter 4.
- A novel method for mean shift tracking through scale is presented in Chapter 5, which works by using a novel kernel to interleave scale space mean shift iterations with spatial iterations.

## 1.4 Thesis Outline

An outline of the content of this thesis is now given. Chapter 2 covers the necessary background in the fields of detection and tracking. There is a large body of literature for both of these fields, and so there is a focus on reviewing research related to the approaches used in later chapters. For detection, techniques involving machine learning are reviewed, and for tracking, kernel based methods and recursive Bayesian filters are explained.

Chapter 3 explores the problem of detecting people in images. The integral channel features detector is introduced, and is used as a starting point to address the problem. This detector uses image gradients and colour features, and is trained using the AdaBoost algorithm. Practical issues surrounding the implementation of the detector are explored, and a detailed method for accelerating the training process is given. Results are presented on the INRIA person dataset, and the effect of a variety of novelties is explored. These include altering the layout of the orientation bins for gradient orientation features, normalising the gradient orientation features under different schemes, and using contrast sensitive features in a variety of combinations with other features.

Chapter 4 investigates the potential of using multiple boosting classifiers in conjunction. First, the idea of reflecting a boosting classifier in the vertical axis is presented, and it is shown that combining a reflected classifier with its original version can yield improvements in performance, without causing a significant reduction in speed. Next, it is shown that positive training images can be clustered by mapping them to vectors constructed from the outputs of the weak classifiers within a boosting classifier, and then applying the $K$-means algorithm. This creates clusters of images with similar visual characteristics. Boosting classifiers can then be trained on the images in these clusters and combined together to create sets of classifiers. Results for this approach are presented, and it is shown that combining classifiers can lead to improved accuracy over the single classifier approach, with only a small increase in the time taken to evaluate an image.

Chapter 5 examines the problem of tracking people in video. Mean shift tracking is introduced, and a novel method of mean shift tracking through scale is presented. This method relies on the use of a novel kernel, which can be used to calculate mean shift iterations in scale space in fewer operations than two other popular methods. Results are presented on the CAVIAR dataset. The problem of tracking multiple targets is also briefly addressed, by using boosting classifiers to detect targets and Kalman filters for smoothing.

In Chapter 6 the work that has been presented is summarised, and conclusions are drawn. Also, ideas for future research that could expand upon this thesis are suggested.

# Chapter 2

# Background

In this chapter, literature relevant to the material in this thesis is reviewed. First, the general problem of object detection is considered, and the role of machine learning in this problem is examined. Then, existing methods of person and pedestrian detection are reviewed. The next section covers evaluation methodologies for object detection, and this is followed by a section describing the data used for training. Literature on the problem of object tracking is covered next, and approaches based on the mean shift algorithm and recursive Bayesian filters are examined. The last section concludes the chapter by highlighting the limitations of existing techniques for detection and tracking, and suggesting directions for research.

## 2.1 Object Detection

To detect all instances of a specific object class within an image, it must be known what visual features indicate the presence of the object class. This can be deduced from images of the object class. One possible approach to this problem is to use template matching, where we measure the similarity of regions of an image to a template, or set of templates. While such methods have been used to detect object classes such as pedestrians [18], template based methods usually fail to capture the large variation in appearance of an object class, and suffer from the fact that their running time is linear in the number of templates.

An alternative approach is to use machine learning, which involves using image data to find a function that maps an image to a label that indicates the absence or presence of

an object class. Machine learning has been applied successfully to many problems where data is readily available for training, such as speech recognition [19], natural language processing [20], and user preference prediction [21]. In all of these areas, it would be difficult for a person to manually design a solution to the problem, and so it seems natural to use data to characterise what the solution should be. A key requirement is that a learned solution must be able to generalise beyond the data that was used to train it. For example, a person detector must be able to detect people that were not a part of its training set. Computational learning theory [22] analyses what kinds of guarantees can be made in regard to generalisation, but this topic will not be explored in this thesis.

Machine learning comprises a number of different algorithms and approaches for different problems. Typical problems include classification, regression, and clustering [23]. Object detection corresponds to the problem of classification, and if there is only one object class, the problem becomes binary classification. There are a variety of object detection methods that use machine learning, but many of these techniques conform to a general framework. Let $(\mathbf{I}, y)$ be a training example where $\mathbf{I}$ is an image and $y \in \{-1, +1\}$ is a binary label indicating the absence or presence of the object class of interest. A training example with the label $+1$ is referred to as a positive example, and the label $-1$ indicates a negative example. Typically, we begin with a training set $\mathcal{S} = \{(\mathbf{I}_1, y_1), \ldots, (\mathbf{I}_n, y_n)\}$, where all of the images in the set share the same dimensions. The two main aspects of most object detectors are a feature set and a learning algorithm.

Features are generated by applying a transformation $\mathbf{\Phi}$ to the input images $\mathbf{I}$. The purpose of using features is twofold. The first aim is to reduce the dimensionality of the input data to make training and testing tractable, and so the dimensionality of $\mathbf{\Phi}$ will typically be an order of magnitude less than that of $\mathbf{I}$. The second aim is to extract discriminative information from the input data. There are a variety of methods to accomplish this, such as image filters [24] and self-similarity measures [25], but the most widely used features rely on image gradients. Examples of popular features include the scale invariant feature transform [26] and the HOG feature transform [4].

Various different learning algorithms are available for training object detectors. Neural networks [27] were among some of the earliest techniques, but their use declined due

to problems with overfitting and the computational cost of training. Recently however, with the advent of deep learning [28], neural networks have become more popular, and have achieved some of the best results in the field of object detection [29]. However, they remain slow to train, and often require graphics processing units to accelerate the training process, which are outside the scope of this thesis. Support Vector Machines (SVMs) have been very popular in object detection [30]. Training an SVM corresponds to solving a convex optimisation problem, and such problems have been studied in great depth [31]. As a result, a range of efficient training algorithms exist, and there is always a global optimum solution to the training problem. SVMs are often used with kernels, which allow the classifier to be non-linear in the original feature space. However, kernel SVMs have a much greater computational cost when performing classification over linear SVMs, and so linear SVMs remain popular in computer vision problems. An important generalisation of SVMs are structural SVMs, which allow training for learning problems with structured output labels [32]. An example of such a problem is human pose estimation, where instead of binary output labels, an output label for a human would consist of the locations of various keypoints that estimate a person's pose [33]. Random forests [34] have become popular in recent times for object detection in depth imaging [15], and have been modified to create Hough forests, which have been used for pedestrian detection [35]. Random forests are fast to train, but various parameters such as the depth and number of trees must be set correctly to achieve good performance. Boosting algorithms have also been used for object detection [36]. These algorithms work by training a series of weak classifiers, with each individual weak classifier performing little better than chance on the classification task. However, the weak classifiers are trained sequentially, so that the next classifier in the sequence compensates for the deficiencies of the previous classifiers. Therefore, an additive combination of these weak classifiers results in a strong classifier. The actual type of weak classifier used for the algorithm is left as a choice for the user. As many weak classifiers are trained by a boosting algorithm, a simple form of weak classifier is often chosen to reduce the overall training time. Examples of classifiers that are often used include decision trees [37], look up tables [38], and linear discriminants [39]. Boosting classifiers perform well at classification tasks, but the training process is often slow, and can consume large amounts of memory.

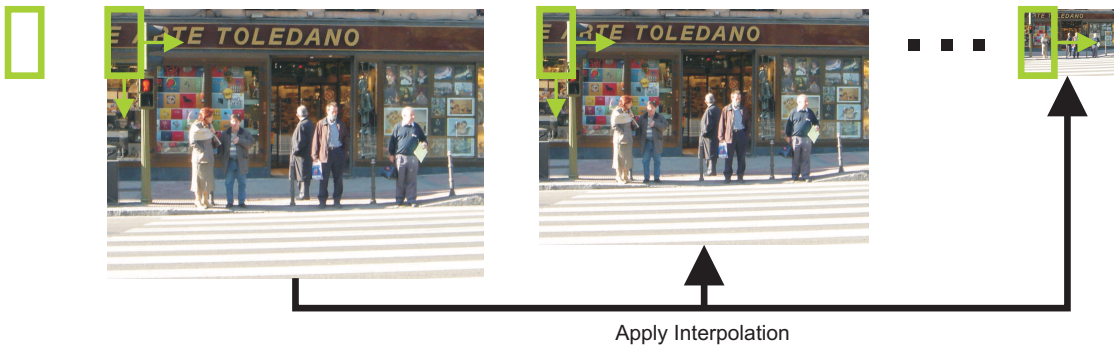



Figure 2.1: A sliding window classifier will have a fixed resolution, represented by the green rectangle on the left. To detect objects at multiple resolutions, the input image is rescaled multiple times with an interpolation algorithm, and the classifier is run at multiple overlapping positions on each rescaled image.

There are several popular boosting algorithms [37], and one of the most widely used is Adaptive Boosting (AdaBoost) [40].

When training an object detector in the manner that has been described, the detector will be able to classify images that are the same dimensions as those in the training set. To find instances within an image, it is necessary to apply the classifier at multiple scales and translations on the input image. This is achieved by rescaling the image several times, and applying the classifier at positions arranged on a two dimensional grid, as shown in Figure 2.1. Classifiers that work in this way are often referred to as *sliding window classifiers*. In this thesis, the term window will be used to refer to an image or a region within an image that has the same dimensions as the images in the training set of a detector. After running a sliding window classifier on an image, any object instance that has been detected will normally be enclosed by multiple bounding boxes. Each instance should only be marked by a single bounding box, as shown in Figure 1.1, and the process of reducing the multiple bounding boxes to a single bounding box per target is known as *non-maximum suppression*.

### 2.1.1 Person and Pedestrian Detection

One of the earliest works on pedestrian detection used Haar wavelets in combination with a SVM classifier [41]. A major step forward was to design improved features based upon histograms of image gradients computed over a grid of cells, with the feature vectors from each cell being normalised with respect to neighbouring cells [4]. These features

were also combined with a SVM classifier. HOG features, as they are known, became widely adopted in the field of object detection. HOG features have also been used with other classification algorithms such as AdaBoost [42]. In other work, detection accuracy has been improved by combining HOG with other features such as local binary patterns [43] or colour similarity features [25]. A further development of the HOG-SVM approach uses an extended set of HOG features in conjunction with principal component analysis for reduced dimensionality, along with a new variant of the SVM known as the latent SVM [44]. The latent SVM allows a parts based detector to be trained, with the parts as latent variables. The parts are able to move relative to each other, greatly increasing the flexibility of the classifier. Whereas most classifiers score an example based on its appearance, the classifier presented in [44] computes a score based on appearance and spatial arrangement of parts. The parts based latent SVM framework has become very popular due to its consistently high performance in a series of object detection challenges [45]. It has been further extended to create the grammar model framework [46], where a set of rules can be used to create multiple deformable classifiers. In order to train these multiple deformable classifiers, a new type of classifier known as the weak label structural SVM, was developed.

While SVMs have been popular for person detection, another line of research has pursued the use of boosting classifiers. One approach is to combine an AdaBoost classifier, with decision trees as weak classifiers, with features computed from image gradients and the CIE 1976 (L*, u*, v*) colour space (CIELUV) [47]. The speed of such a classifier can be greatly increased by approximating the feature values over certain scales, so that features only have to be computed for a small number of rescaled images [48]. Further improvements in speed can be achieved by using a validation set to tune the parameters for a set of cascade classifiers that can communicate information from neighbouring regions of the input image [49], making it possible for the detector to process frames at a resolution of $640 \times 480$ at over 30 frames per second. Another approach to accelerating detection is to train classifiers for different scales, so that features are only computed for a single image [50].

### 2.1.2 Evaluation Methodology

An important issue is being able to gauge the performance of different object detection algorithms. The normal protocol for evaluating the performance of a detector trained on a data set is to have a separate test set which has a ground truth. The performance on this test set is then measured relative to the ground truth. If there is a common test set for a group of classifiers, it is possible to compare their performance. Ideally, we may wish the detectors to also share the same training set as well. In object detection, errors arise from failing to detect instances of an object (false negatives), or from detecting instances where none are present (false positives). Typically, we find that there is a trade-off between the two types of errors. If our detector produces real valued scores for each detection window, then it is possible to visualise and measure this trade-off using a precision-recall curve [45] or a Detection Error Trade-off (DET) curve [4].

The use of standardised testing and training sets is now widespread throughout the computer vision community. This approach was popularised for person detection by the public release of the INRIA person dataset [51], which accompanied the work presented in [4]. In this work, detectors were evaluated on the INRIA person test set using DET curves. These curves plot the miss rate on the $y$-axis against the number of false positives per window on the $x$-axis. The miss rate MR is defined as:

$$\mathrm{MR} = \frac{\text{false neg.}}{\text{true pos.} + \text{false neg.}} \tag{2.1}$$

where the quantities on the right hand side of the equation are the number of false negative windows and true positive windows. This methodology was adopted by a number of subsequent publications [52], [53], [43]. However, several issues have been raised with this approach. Plotting DET curves as described only evaluates the performance of a detector on the basis of windows, and not on the basis of entire images. This means that the evaluation takes place prior to non-maximum suppression. It has been shown that the performance measured per window does not necessarily reflect the performance measured per image [6]. Another issue is that there are ambiguities as to how the metrics should be calculated. Some authors only use a randomly selected subset of the negative test set when computing the number of false positive per window, but this obviously

| | Negative Images | Positive Images | Number Of Annotated People |
|---|---|---|---|
| Training Set | 1218 | 614 | 2416 |
| Test Set | 453 | 288 | 1126 |

Table 2.1: A table showing a breakdown of the INRIA training and test sets in terms of positive and negative images, and the number of annotated instances of people.

means results cannot be compared if different random subsets were used [51]. To address these issues, an alternative methodology has been suggested which involves plotting the miss rate against the number of false positives per image, along with standardised scripts to calculate these metrics and new ground truth annotations [6]. The authors of these scripts also evaluated the performance of more than a dozen methods for pedestrian detection using these metrics on the INRIA person dataset, the ETH pedestrian dataset [54], the TUD Brussels pedestrian dataset, the Daimler pedestrian dataset [55], and the Caltech pedestrian dataset [56]. Since then, more detectors have been added to the Caltech Pedestrian Detection Benchmark [5], and at the time of writing, over thirty detectors have been evaluated. In order to benchmark the detectors developed in this thesis, the scripts for this framework are used.

### 2.1.3 The INRIA Person Dataset

The INRIA person dataset [51] is a widely used set of images for training and testing person detection algorithms. It consists of fixed training and test sets, each of which are comprised of positive images which contain instances of people, and negative images which do not contain people. The instances vary in size and appearance. All images that contain instances of people are annotated with bounding boxes to denote the position and size of each person.

Table 2.1 gives a breakdown of the test and training sets in terms of positive and negative images, and the number of annotated instances in each set. It should be noted that the number of annotated people is larger than the number of positive images, as several people may appear in a single image. Also, the actual number of unique instances in the positive training set is 1208, but these are mirrored in the vertical axis to double this figure.

As well as including annotations for all instances of people in the test and training

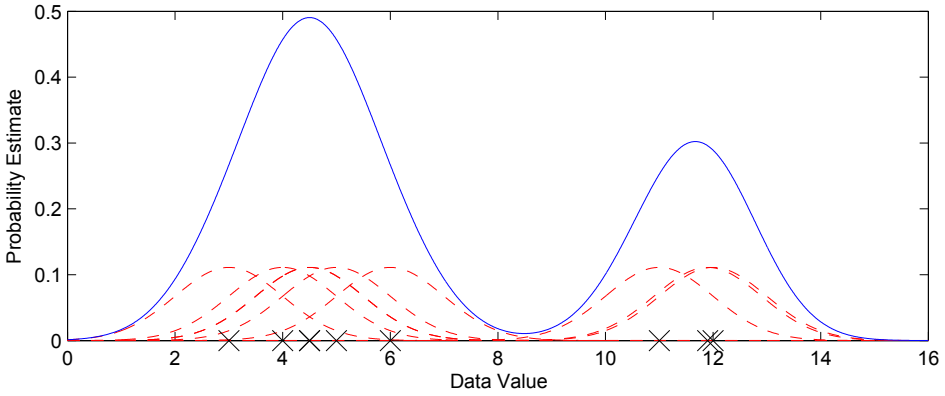



Figure 2.2: The Parzen window is a simple kernel density estimator for one dimensional data. In this figure, the data points that have been observed are plotted along the bottom axis as crosses. Gaussian kernels, shown by the dotted red lines, have been placed at each data point. The sum of these kernels, shown in blue, gives an estimate of the probability distribution that generated the points.

sets, the INRIA person dataset also includes cropped and normalised images of all the annotated instances of people. These cropped images are $64 \times 128$ pixels, where each person is $32 \times 96$ pixels with a border around them. There are no cropped negative images, but these can be easily produced from the provided negative images.

Throughout this thesis, person detectors will be trained using the cropped positive images from the INRIA training set, along with randomly cropped negative images from the INRIA training set. Testing will use the 288 uncropped positive images in the INRIA test set, in accordance with the Caltech evaluation methodology outlined in Section 2.1.2.

## 2.2 Tracking

An extensive review of tracking techniques in computer vision is given in [57]. For the purposes of this thesis, two broad approaches to the problem are examined. These approaches are kernel methods (specifically the mean shift algorithm) and recursive Bayesian filters.

### 2.2.1 Kernel-Based Tracking

Kernel-based tracking methods are derived from the field of kernel density estimation, which deals with the problem of estimating a probability distribution from observations

drawn from the distribution. The simplest such estimator is the Parzen window [58], where the distribution is approximated by placing a kernel function (typically a Gaussian function) at the position of each observation, as shown in Figure 2.2. Formally, this gives the kernel density estimate for a multivariate distribution as

$$\hat{p}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} K_{\mathbf{H}}(\mathbf{x} - \mathbf{x}_i), \tag{2.2}$$

where $N$ is the number of observations drawn from the $d$ dimensional distribution, and $\mathbf{x}_i \in \mathbb{R}^d, i = 1, \ldots, N$ are the observations. This is simply a result of placing kernels centered on the data points, and $K_{\mathbf{H}}$ is given by

$$K_{\mathbf{H}}(\mathbf{x}) = |\mathbf{H}|^{-1/2} K(\mathbf{H}^{-1/2}\mathbf{x}), \tag{2.3}$$

where $\mathbf{H} \in \mathbb{R}^{d\times d}$ is a bandwidth matrix which is used to apply affine transformations (scaling, rotation, and/or skew) to the kernel, and $K$ is a kernel function satisfying the conditions [59]

$$\begin{aligned} \int_{\mathbb{R}^d} K(\mathbf{x})\mathrm{d}\mathbf{x} &= 1 & \lim_{\|\mathbf{x}\|\to\infty} \|\mathbf{x}\|^d K(\mathbf{x}) &= 0 \\ \int_{\mathbb{R}^d} \mathbf{x}K(\mathbf{x})\mathrm{d}\mathbf{x} &= 0 & \int_{\mathbb{R}^d} \mathbf{x}\mathbf{x}^{\mathsf{T}} K(\mathbf{x})\mathrm{d}\mathbf{x} &= c_K\mathbf{I} \end{aligned} \tag{2.4}$$

where $c_K$ is a constant. In many situations, it is adequate for the bandwidth matrix $\mathbf{H}$ to be the identity matrix multiplied by a constant $h$ squared. Also, the kernel function can take a number of forms [60], but a popular approach is to make it radially symmetric by relating it to a profile function $k$ through the equation $K(\mathbf{x}) = c_k k(\|\mathbf{x}\|^2)$ where $c_k$ is a normalisation constant. These two conditions result in Equation 2.2 being simplified to

$$\hat{p_k}(\mathbf{x}) = \frac{c_k}{Nh^d}\sum_{i=1}^{N} k\left(\left\|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right\|^2\right), \tag{2.5}$$

Under the conditions that lead to Equation 2.5, it is possible to derive an algorithm that allows us to iteratively find the nearest mode of the density estimation from a given point $\mathbf{x}$. This algorithm is known as the mean shift procedure, and it is a method of gradient ascent. The derivation begins by calculating the gradient of the density estimation,

where $g(x) = k'(x)$ (assuming $k$ is differentiable)

$$\begin{aligned}\nabla\hat{p_k}(\mathbf{x}) &= \frac{2c_k}{Nh^{d+2}}\sum_{i=1}^{N}(\mathbf{x}-\mathbf{x}_i)g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right) \\ &= \underbrace{\frac{2c_k}{Nh^{d+2}}\left(\sum_{i=1}^{N}g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)\right)}_{\hat{p}_g(\mathbf{x})\frac{2c_k}{c_g h^2}}\underbrace{\left(\frac{\sum_{i=1}^{N}\mathbf{x}_i g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)}{\sum_{i=1}^{N}g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)}-\mathbf{x}\right)}_{\mathbf{m}_g(\mathbf{x})},\end{aligned} \tag{2.6}$$

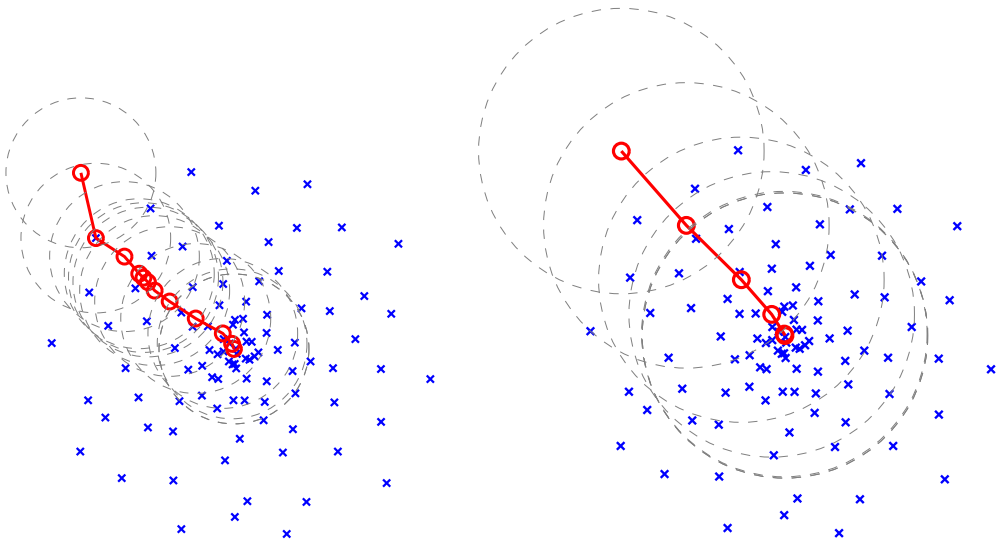
where $\mathbf{m}_g(\mathbf{x}) \in \mathbb{R}^d$ is the mean shift vector and $\hat{p}_g(\mathbf{x})$ is kernel density estimation with the kernel $G$ (described by profile function $g$). The kernel $G$ is referred to as the shadow kernel of $K$ [61] (the relation between these kernels is explored later). Equation 2.6 has no closed form solution for $\mathbf{x}$, but by setting the left hand side to zero, we obtain the following iterative method of calculation

$$\mathbf{y}_{j+1} = \frac{\sum_{i=1}^{N}\mathbf{x}_i g\left(\left\|\frac{\mathbf{y}_j-\mathbf{x}_i}{h}\right\|^2\right)}{\sum_{i=1}^{N}g\left(\left\|\frac{\mathbf{y}_j-\mathbf{x}_i}{h}\right\|^2\right)} \tag{2.7}$$

where $\mathbf{x}$ has been replaced by $\mathbf{y}_{j+1}$ on the left hand side and $\mathbf{y}_j$ on the right hand side to emphasise that this is an iterative procedure. This equation tells us that the next position in the gradient ascent procedure can be calculated at the previous position. The difference between two consecutive position estimates is the mean shift vector $\mathbf{m}_g(\mathbf{x})$. Equation 2.6 can be rearranged to show that the mean shift vector is given by

$$\mathbf{m}_g(\mathbf{x}) = \frac{\sum_{i=1}^{N}\mathbf{x}_i g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)}{\sum_{i=1}^{N}g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)} - \mathbf{x} = \frac{h^2 c_g \nabla\hat{p_k}(\mathbf{x})}{2c_k\hat{p}_g(\mathbf{x})} \tag{2.8}$$

which clearly shows that the mean shift vector is proportional to the density gradient estimate with kernel $K$. Hence, the mean shift vector always points towards the nearest mode of the distribution, and it can be used iteratively to travel to this mode using Equation 2.7, which is iterated until $\mathbf{m}_g(\mathbf{y}_j)$ is less than some small threshold. Details on the convergence of the algorithm are given in [60]. An illustrative example is shown in Figure 2.3. We can now reinterpret the kernel density estimate in Equation 2.8



(a) $h = 10$ (b) $h = 18$

Figure 2.3: An example of using mean shift to find the nearest mode. Here, the points $\mathbf{x}_i \in \mathbb{R}^2$ are shown by blue crosses. The starting point for the mean-shift procedure, $\mathbf{y}_0$ is at the top left, and Equation 2.7 is iterated until the mean-shift vector $\mathbf{m}_g(\mathbf{y})$ is less than 0.01. The red circles show the values of $\mathbf{y}$, and the red lines show $\mathbf{m}_g(\mathbf{y})$. The uniform kernel given by Equation 2.11 has been used for $g$, and its position at each iteration is shown by the dotted gray lines. Results for two different values for the bandwidth $h$ are shown.

as representing a single kernel evaluated at many points (rather than multiple kernels evaluated at a single point), and the position of the kernel is moved with each mean shift iteration. It is also noteworthy that the kernel density estimate can be viewed as a convolution, in which case the mean shift procedure gives an efficient way of finding the local mode of a certain type of convolution surface. Typical kernels include the normal kernel, the Epanechnikov kernel and the uniform kernel, which are respectively given by the following equations

$$K_{N,h}(\mathbf{x}) = c_{N,h}\exp\left(-\frac{1}{2}\left\|\frac{\mathbf{x}}{h}\right\|^2\right), \tag{2.9}$$

$$K_{E,h}(\mathbf{x}) = \begin{cases} c_{E,h}\left(1 - \left\|\frac{\mathbf{x}}{h}\right\|^2\right) & \|\mathbf{x}\| \leq h \\ 0 & \text{otherwise} \end{cases}, \tag{2.10}$$

$$K_{U,h}(\mathbf{x}) = \begin{cases} c_{U,h} & \|\mathbf{x}\| \leq h \\ 0 & \text{otherwise} \end{cases}, \tag{2.11}$$

where $c_{N,h}$, $c_{E,h}$ and $c_{U,h}$ denote normalisation constants. It can be easily shown that the Gaussian kernel is its own shadow kernel. The uniform kernel is the shadow of the Epanechnikov kernel. Thus, finding the local mode of a kernel density estimate where Epanechnikov kernels are used is done by performing mean-shift with the uniform kernel, which simply results in a normalised mean.

The equation for mean shift given earlier is for the general case where the points $\mathbf{x}_i$ are randomly drawn from a distribution. The mean shift procedure looks for where these random points are most dense. For image processing, these points will actually be the pixels, which are uniformly distributed on a grid (which would result in the mean shift vector being zero), so the equation is modified to introduce a weighting function

$$\mathbf{y}_{j+1} = \frac{\sum_{\mathbf{x}_i \in X} \mathbf{x}_i w(\mathbf{x}_i) g\left(\left\| \frac{\mathbf{y}_j - \mathbf{x}_i}{h} \right\|^2\right)}{\sum_{\mathbf{x}_i \in X} w(\mathbf{x}_i) g\left(\left\| \frac{\mathbf{y}_j - \mathbf{x}_i}{h} \right\|^2\right)}, \tag{2.12}$$

where the nature of the weighting function $w(\mathbf{x}_i)$ is dependent upon the particular algorithm, and $X$ is the support of the kernel $g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)$.

The first mean shift based tracking technique was the Continuously Adaptive Mean Shift (CAMSHIFT) algorithm [62]. Given the position and scale of a target in the first frame of a video sequence, this algorithm works by first taking a colour histogram of the target in the Hue, Saturation and Value (HSV) colour space. In subsequent frames, pixels are replaced with their corresponding values from the colour histogram (this is the weighting function for Equation 2.12 in this instance). This leads to pixels that are likely to be part of the target being assigned high values, while other pixels are assigned low or zero values. Then, mean shift is performed in the spatial domain using a uniform rectangular kernel.

A very popular mean shift tracking algorithm was derived in [63]. In this paper, it is proposed that a target is represented by a colour histogram (the target model) that is calculated in the first frame given the initial target position and scale. In subsequent frames, the best target candidate is found by maximising the Bhattacharyya coefficient which measures the similarity between the target model and a candidate model. It is shown that the linear Taylor series representation of the Bhattacharyya coefficient takes

the form of a convolution surface, and so the mean shift procedure can be used to find the local mode of this surface (which will probably correspond to the target position, assuming that it is not moving extremely quickly). Tracking through scale is achieved by running the tracker at three different scales (the current scale, a smaller scale, and a larger scale), and then selecting the solution with the highest Bhattacharyya coefficient.

The previous two mean shift tracking algorithms are fairly typical in that they use only colour information and do not take into account the spatial distribution of colour. This can be considered both an advantage and disadvantage, as it enables mean shift trackers to deal with deformations of the target, which can cause other trackers to break down (for example, consider the many articulations of the human body, and the difficulty associated with creating a tracker capable of handling all possible poses). However, this also means that these trackers will fail in situations where the target's colours are not distinct from their surroundings (for example, consider tracking any target in greyscale or black and white). A very general framework for mean shift tracking is derived in [64], where it is shown that spatial kernels can be introduced to enforce soft constraints on the spatial arrangement of colours. This is done by creating a kernel estimate that takes the form of a product of two kernels, one for colour, and another for space. Adjusting the bandwidth of either kernel allows for a trade off between considering spatial information and allowing for deformations of the target. Interestingly, the authors prove that some popular algorithms are in fact special cases of this framework, such as the sum of squares tracker (which has zero spatial kernel bandwidth) and the algorithm presented in [63] (which has infinite spatial kernel bandwidth). However, the spatial kernels result in a large increase in the number of computational operations needed over the mean-shift algorithm from [63].

The problem of tracking through scale for mean shift style trackers is non-trivial, and has been explored in more depth in [65], where a method is developed based on Lindberg's theory of feature scale selection [66]. However, the method requires the generation of multiple normal kernels for each frame, which once again requires many more operations by an order of magnitude than the normal mean-shift algorithm.

There has been much research into the relation between mean shift and Newton style iterative methods. In [67] an alternative to mean shift is derived using Newton style

iterations, and it is shown that these iterations can converge in a single step. In [68], it is proved that mean shift can be viewed as a form of quadratic bound optimisation, and that performing mean shift with a piece-wise constant kernel is equivalent to Newton's method. Other work focuses on recovering other types of motion using kernels. It is shown in [69] that kernel based methods can be used to track the articulated motion of limbs.

### 2.2.2 Recursive Bayesian Filters

Recursive Bayesian filters include methods such as the Kalman filter [70] and particle filters [71]. These techniques are also often referred to as state space methods.

The formulation of the tracking problem in the recursive Bayesian framework is as follows. It is assumed that we receive a sequence of noisy measurements $\mathbf{z}_t$ $(t = 0, ..., n)$ over time of a target's state, from which we wish to infer the true state $\mathbf{x}_t$. In the cases considered here, the observation $\mathbf{z}_t$ is a vector which represents a noisy estimate of the target's position, and the state $\mathbf{x}_t$ is usually an estimate of the true position and velocity. The solution is to compute the posterior distribution $p(\mathbf{x}_t|\mathbf{z}_{1:t})$ where $\mathbf{z}_{1:t} = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_t\}$, and to then use this distribution to estimate the true state, usually by taking the conditional expectation $\mathbb{E}[\mathbf{x}_t|\mathbf{z}_{1:t}]$.

In this dynamic system, we have a process model, which dictates how the state evolves over time, and a measurement model, which explains how the measurements are related to the state. These are given by

$$\mathbf{x}_t = \mathbf{f}(\mathbf{x}_{t-1}, \mathbf{v}_t), \tag{2.13}$$

$$\mathbf{z}_t = \mathbf{h}(\mathbf{x}_t, \mathbf{n}_t), \tag{2.14}$$

where $\mathbf{v}_t$ is the process noise, and $\mathbf{n}_t$ is the observation noise. For example, if an object that is falling under the force of gravity is being tracked, then $\mathbf{f}$ would apply Newtonian dynamics to calculate $\mathbf{x}_t$ from $\mathbf{x}_{t-1}$. If the observations $\mathbf{z}_t$ are the target's position, and the state vector contains both the target's position and velocity, then $\mathbf{h}$ may be a function that projects $\mathbf{x}_t$ to a vector of lower dimensionality. In the special case where $\mathbf{f}$ and/or $\mathbf{h}$ are linear functions, they are represented by matrices $\mathbf{F}$ and $\mathbf{H}$.

The following assumptions are made:

1. The states follow a first order Markov process. This means

$$p(\mathbf{x}_t|\mathbf{x}_{1:t-1}) = p(\mathbf{x}_t|\mathbf{x}_{t-1}). \tag{2.15}$$

This assumption explains the form of the process model, Equation 2.13.

2. The observations are independent given the states. This means

$$p(\mathbf{z}_t|\mathbf{x}_t, A) = p(\mathbf{z}_t|\mathbf{x}_t), \tag{2.16}$$

where $A$ is any set of random variables that does not include $\mathbf{z}_t$ and $\mathbf{x}_t$.

These assumptions can be used to derive an expression for the posterior distribution as shown below

$$\begin{aligned}
p(\mathbf{x}_t|\mathbf{z}_{1:t}) &= \frac{p(\mathbf{z}_{1:t}|\mathbf{x}_t)p(\mathbf{x}_t)}{p(\mathbf{z}_{1:t})} \\
&= \frac{p(\mathbf{z}_t, \mathbf{z}_{1:t-1}|\mathbf{x}_t)p(\mathbf{x}_t)}{p(\mathbf{z}_t, \mathbf{z}_{1:t-1})} \\
&= \frac{p(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_t)p(\mathbf{z}_{1:t-1}|\mathbf{x}_t)p(\mathbf{x}_t)}{p(\mathbf{z}_t|\mathbf{z}_{1:t-1})p(\mathbf{z}_{1:t-1})} \\
&= \frac{p(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_t)p(\mathbf{x}_t|\mathbf{z}_{1:t-1})\cancel{p(\mathbf{z}_{1:t-1})}\cancel{p(\mathbf{x}_t)}}{p(\mathbf{z}_t|\mathbf{z}_{1:t-1})\cancel{p(\mathbf{z}_{1:t-1})}\cancel{p(\mathbf{x}_t)}} \\
&= \frac{p(\mathbf{z}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{z}_{1:t-1})}{p(\mathbf{z}_t|\mathbf{z}_{1:t-1})}.
\end{aligned} \tag{2.17}$$

Equation 2.17 can be viewed as an equivalent of Bayes' theorem for this time varying process. The likelihood is given by $p(\mathbf{z}_t|\mathbf{x}_t)$, while the prior is $p(\mathbf{x}_t|\mathbf{z}_{1:t-1})$. The denominator is sometimes referred to as the evidence, and is only required for normalisation, as $p(\mathbf{z}_t|\mathbf{z}_{1:t-1}) = \int p(\mathbf{z}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{z}_{1:t-1})\mathrm{d}\mathbf{x}_t$. The prior can be calculated by

$$p(\mathbf{x}_t|\mathbf{z}_{1:t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1})\mathrm{d}\mathbf{x}_{t-1}, \tag{2.18}$$

which can be proven using the Chapman-Kolmogorov equation [72]. The distribution $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ is given by the process model, and is often referred to as the transition density.

Equations 2.18 and 2.17 can be used iteratively to obtain estimates of the posterior

distribution, without storing the entire history of states or measurements. Equation 2.18 is referred to as the *prediction stage*, as it projects what the state is likely to be before an observation is made. This is used to calculate the posterior in Equation 2.18, which is referred to as the *update stage*. The update stage incorporates the observation through the likelihood distribution. The posterior is then used to estimate the current state $\hat{\mathbf{x}}_{t|t}$, usually by taking the expectation. It will then be used to calculate the prior for the next stage, and the process continues to repeat itself in this fashion.

Due to the need to evaluate integrals, the recursive Bayesian filtering problem does not always have a closed form solution. In situations where the underlying distributions are discrete, grid-based methods [71] can be used to give optimal situations. For the continuous case, an optimal solution exists when the transition, likelihood and initial state densities are Gaussian, and the process and measurement models are linear, leading to the following alternative process and observation models

$$\mathbf{x}_t = \mathbf{F}\mathbf{x}_{t-1} + \mathbf{v}_t, \tag{2.19}$$

$$\mathbf{z}_t = \mathbf{H}\mathbf{x}_t + \mathbf{n}_t, \tag{2.20}$$

where $\mathbf{F}$ and $\mathbf{H}$ are matrices and $\mathbf{v}_t$ and $\mathbf{n}_t$ are normally distributed random variables with zero mean. When these conditions are met, it can be shown that the posterior and prior densities will also be Gaussian, as the posterior density will be the normalised product of two Gausssians, and the prior will will be the convolution of two Gaussians [23]. The optimal solution for these conditions is the Kalman filter [73]. As a Gaussian distribution is completely described by its mean and covariance matrix, the Kalman filter estimates these parameters, and the Kalman estimate of the state is simply the mean of the posterior distribution. The prediction equations for the mean and covariance are

$$\hat{\mathbf{x}}_{t|t-1} = \mathbf{F}\hat{\mathbf{x}}_{t-1|t-1}, \tag{2.21}$$

$$\mathbf{P}_{t|t-1} = \mathbf{F}\mathbf{P}_{t-1|t-1}\mathbf{F}^{\mathsf{T}} + \mathbf{Q}, \tag{2.22}$$

where $\hat{\mathbf{x}}_{t|t-1}$ is the prior state estimate (the mean of the prior distribution), $\hat{\mathbf{x}}_{t-1|t-1}$ is the previous Kalman estimate (the mean of the posterior at the previous time step), $\mathbf{P}_{t|t-1}$ is the prior covariance matrix estimate, $\mathbf{P}_{t-1|t-1}$ is the previous covariance estimate and

$\mathbf{Q}$ is the process noise covariance matrix (the covariance of $\mathbf{v}_t$). The update equations for the mean and covariance are

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t(\mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_{t|t-1}), \tag{2.23}$$

$$\mathbf{P}_{t|t} = \mathbf{P}_{t|t-1} - \mathbf{K}_t\mathbf{H}\mathbf{P}_{t|t-1}, \tag{2.24}$$

where $\hat{\mathbf{x}}_{t|t}$ is the Kalman estimate (the mean of the posterior for this time step), $\mathbf{z}_t$ is the measurement, $\mathbf{P}_{t|t}$ is the covariance of the posterior at this time step, and $\mathbf{K}_t$ is the Kalman gain, which is given by

$$\mathbf{K}_t = \mathbf{P}_{t|t-1}\mathbf{H}^{\mathsf{T}}(\mathbf{H}\mathbf{P}_{t|t-1}\mathbf{H}^{\mathsf{T}} + \mathbf{R})^{-1}, \tag{2.25}$$

where $\mathbf{R}$ is the observation noise covariance matrix. Thus, for initialisation the Kalman filter requires an initial estimate of the state $\hat{\mathbf{x}}_0$, and estimates of $\mathbf{Q}$ and $\mathbf{R}$. The estimates of $\mathbf{Q}$ and $\mathbf{R}$ can have a large role in determining the behaviour of the Kalman filter.

There are several well known extensions to the Kalman filter designed to deal with non-linear tracking. Unlike the Kalman filter, these techniques are not optimal. The first and simplest of these is the Extended Kalman Filter (EKF) [74]. The EKF is identical to the Kalman filter, except that it is designed for situations where the state transitions and measurement model are non-linear, and are represented by Equations 2.13 and 2.14 rather than Equations 2.19 and 2.20. The solution is to take the local linearisation of the functions $\mathbf{f}$ and $\mathbf{h}$, which gives the Jacobian matrices

$$\mathbf{F} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\Big|_{\hat{\mathbf{x}}_{t-1|t-1}}, \tag{2.26}$$

$$\mathbf{H} = \frac{\partial \mathbf{h}}{\partial \mathbf{x}}\Big|_{\hat{\mathbf{x}}_{t|t-1}}. \tag{2.27}$$

As the EKF is often sub-optimal, its exact behaviour is unpredictable, and usually depends on the nature of the non-linear functions $\mathbf{f}$ and $\mathbf{h}$. In situations where these functions can be well approximated as linear functions on a time scale comparable to the one in which the observations are made, the EKF can give good performance. However, for highly non-linear models, the EKF usually performs poorly. Also, there are issues due to the fact that a non-linear function of a Gaussian variable is not itself normally

distributed, and therefore the EKF simply fits a Gaussian to this undetermined distribution. This leads to poor performance when the underlying distribution is multi-modal.

Another sub-optimal adaptation of the Kalman filter is the Unscented Kalman Filter (UKF), also known as the sigma point filter [75]. The UKF is in a sense similar to particle filtering (which will be explained later) in that it seeks to approximate the posterior distribution as a weighted set of samples, known as sigma points. However, the UKF chooses the sigma points deterministically, while the particle filter uses random sampling. Also, the UKF uses far fewer sigma points ($2k+1$, where $k$ is the dimensionality of $\mathbf{x}$) than the typical number of particles used by a particle filter. The UKF has been found to outperform the EKF in many scenarios [76]. It can estimate the posterior mean and covariance correctly up to the second order Taylor series coefficient for all non-linearities. The filter works by calculating a set of sigma points $\mathbf{X}^i_{t-1}$ as follows

$$\mathbf{X}^0_{t-1} = \bar{\mathbf{x}}_{t-1}, \tag{2.28}$$

$$\mathbf{X}^i_{t-1} = \bar{\mathbf{x}}_{t-1} + (\sqrt{(L+\lambda)\mathbf{P}_{t-1}})_i \qquad i = 1, ..., L, \tag{2.29}$$

$$\mathbf{X}^i_{t-1} = \bar{\mathbf{x}}_{t-1} - (\sqrt{(L+\lambda)\mathbf{P}_{t-1}})_{i-L} \qquad i = L+1, ..., 2L, \tag{2.30}$$

where $\bar{\mathbf{x}}_{t-1}$ is the mean, $\lambda$ is a scaling parameter and $(\sqrt{(L+\lambda)\mathbf{P}_{t-1}})_i$ is the $i$th row or column of the matrix square root of $(L+\lambda)\mathbf{P}_{t-1}$. The intuition behind this methodology is that the columns of the covariance matrix of a Gaussian can be seen as vectors representing the principal axes of variance for the distribution, and so the points are taken at the peak of the Gaussian and along a contour of constant probability. The points collectively characterise the distribution quite well if it is Gaussian or close to being Gaussian, but badly if it is multi-modal. The sigma points are propagated through the non-linear process model (Equation 2.13) to obtain the points $\mathbf{X}^i_t$, which are then used to calculate the mean and covariance by

$$\bar{\mathbf{x}}_t = \sum_{i=0}^{2L} W_i \mathbf{X}^i_t, \tag{2.31}$$

$$\bar{\mathbf{P}}_t = \sum_{i=0}^{2L} W_i (\mathbf{X}^i_t - \bar{\mathbf{x}}_t)(\mathbf{X}^i_t - \bar{\mathbf{x}}_t)^\mathsf{T} + \mathbf{Q}, \tag{2.32}$$

where the weights are $W_0 = \lambda/(d+\lambda)$ and $W_i = 1/2(d+\lambda)$.

As has been mentioned, extensions to the Kalman filter to deal with non-linear systems are sub-optimal. A better approach is particle filtering. The particle filter seeks to approximate the posterior distribution by a set of weighted samples

$$\hat{p}(\mathbf{x}_t|\mathbf{z}_{1:t}) = \frac{1}{N_s}\sum_{i=1}^{N_s} \tilde{w}_t^i \delta(\mathbf{x}_t - \mathbf{x}_t^i), \tag{2.33}$$

where $N_s$ is the number of particles, $\tilde{w}_t^i$ is the weight associated with sample $\mathbf{x}_t^i$ and $i = 1, ..., N_s$ is the sample index number. The weights are normalised so that $\sum_i \tilde{w}_t^i = 1$. This approximation can then be used to decide where the target is, for example, by calculating an approximation of the expectation

$$\mathbb{E}[\mathbf{x}_t|\mathbf{z}_{1:t}] \approx \frac{1}{N_s}\sum_{i=1}^{N_s} \tilde{w}_t^i \mathbf{x}_t^i. \tag{2.34}$$

The issues that need to be addressed are how the particles are drawn, and how the weights are calculated. To approximate the posterior using standard Monte Carlo techniques, a very large number of samples could be drawn from the posterior, and the samples could be weighted uniformly. However, this is completely impractical, as the posterior could be very complex, and computers can only practically generate random numbers from a few standard distributions. The solution is to use importance sampling [23], which allows for the approximation of a distribution $p(\cdot)$ by drawing samples from a different distribution $q(\cdot)$, referred to as the proposal, or importance distribution. These samples can be used by weighting them to account for the difference between $p(\cdot)$ and $q(\cdot)$. The unnormalised weights are derived below

$$\begin{aligned}
\mathbb{E}[\mathbf{x}_t|\mathbf{z}_{1:t}] &= \int \mathbf{x}_t \frac{p(\mathbf{x}_t|\mathbf{z}_{1:t})}{q(\mathbf{x}_t|\mathbf{z}_{1:t})} q(\mathbf{x}_t|\mathbf{z}_{1:t}) \mathrm{d}\mathbf{x}_t \\
&= \int \mathbf{x}_t \frac{p(\mathbf{z}_{1:t}|\mathbf{x}_t)p(\mathbf{x}_t)}{p(\mathbf{z}_{1:t})q(\mathbf{x}_t|\mathbf{z}_{1:t})} q(\mathbf{x}_t|\mathbf{z}_{1:t}) \mathrm{d}\mathbf{x}_t \\
&= \int \mathrm{x}_t \frac{w_t(\mathbf{x}_t)}{p(\mathbf{z}_{1:t})} q(\mathbf{x}_t|\mathbf{z}_{1:t}) \mathrm{d}\mathbf{x}_t,
\end{aligned} \tag{2.35}$$

where the weights are given by $w_t(\mathbf{x}_t) = \frac{p(\mathbf{z}_{1:t}|\mathbf{x}_t)p(\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{z}_{1:t})}$. This leads to the approximation

$$\begin{aligned}
\mathbb{E}[\mathbf{x}_t|\mathbf{z}_{1:t}] &= \frac{1}{p(\mathbf{z}_{1:t})}\int \mathbf{x}_t w_t(\mathbf{x}_t) q(\mathbf{x}_t|\mathbf{z}_{1:t})\mathrm{d}\mathbf{x}_t \\
&= \frac{\int \mathbf{x}_t w_t(\mathbf{x}_t) q(\mathbf{x}_t|\mathbf{z}_{1:t})\mathrm{d}\mathbf{x}_t}{\int p(\mathbf{z}_{1:t}|\mathbf{x}_t)p(\mathbf{x}_t)\frac{q(\mathbf{x}_t|\mathbf{z}_{1:t})}{q(\mathbf{x}_t|\mathbf{z}_{1:t})}\mathrm{d}\mathbf{x}_t} \\
&= \frac{\int \mathbf{x}_t w_t(\mathbf{x}_t) q(\mathbf{x}_t|\mathbf{z}_{1:t})\mathrm{d}\mathbf{x}_t}{\int w_t(\mathbf{x}_t) q(\mathbf{x}_t|\mathbf{z}_{1:t})\mathrm{d}\mathbf{x}_{\boldsymbol{t}}} \\
&\approx \frac{\sum_{i=1}^{N_s} \mathbf{x}_t^i w_t^i}{\sum_{i=1}^{N_s} w_t^i} \\
&\approx \sum_{i=1}^{N_s} \tilde{w}_t^i \mathbf{x}_t^i, \qquad (2.36)
\end{aligned}$$

where the normalised weights are $\tilde{w}_t^i = \frac{w_t^i}{\sum_{j=1}^{N_s} w_t^j}$.

It can be shown [71] that if the proposal density is chosen to satisfy the factorisation $q(\mathbf{x}_{0:t}|\mathbf{z}_{1:t}) = q(\mathbf{x}_t|\mathbf{x}_{0:t-1}, \mathbf{z}_{1:t})q(\mathbf{x}_{0:t-1}|\mathbf{z}_{1:t-1})$, then the weights can be updated recursively using the formula

$$w_t = w_{t-1}\frac{p(\mathbf{z}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{x}_{t-1})}{q(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t})}. \qquad (2.37)$$

Thus, the choice of proposal density $q(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t})$ is one of the major decisions when creating a particle filter. One of the popular choices is to use the transition density $p(\mathbf{x}_t|\mathbf{x}_{t-1})$, which then simplifies Equation 2.37 to $w_t = w_{t-1}p(\mathbf{z}_t|\mathbf{x}_t)$. Another is to use the likelihood distribution as the proposal density. The unscented particle filter [77] uses a UKF to create the proposal density for a particle filter.

The form of particle filter that has been derived up to this point is fairly generic, and suffers from a few serious practical problems. The main problem is that of *degeneracy*, where the weights of the particles fall to low values over time, signifying that the samples are in regions of low probability. It has been proved that over time the variance of the weights can only increase [78]. It has also been proven that the optimal proposal distribution to minimise this variance is $p(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t})$, though in many situations, it is impractical to use this distribution. One way of tackling degeneracy is to resample the distribution. There are simple algorithms available to sample from densities represented by particles [79], and these result in more particles being generated in regions of high probability. After this resampling takes places, the particles are given uniform weighting. The Sampling Importance Resampling (SIR) particle filter uses resampling after at every

time step to mitigate the problem of degeneracy. Particle filtering was first introduced in the computer vision community as the Condensation (Conditional Density Propagation) algorithm [79] which is exactly equivalent to the SIR particle filter.

The techniques mentioned up to this point, give a brief overview of the field of recursive Bayesian filters, but many more techniques exist. The particle filtering framework is very flexible, and lends itself easily to modifications. Other popular particle filters not mentioned so far include the auxiliary particle filter [80], the regularised particle filter [81], and the Rao-Blackwellised particle filter [82]. Several filters have been designed to address the problems that arise when tracking multiple targets. These include the Joint Probability Data Association Filter (JPDAF) [83], although this suffers from the drawback of being unable to deal with targets leaving or entering the field of view. Another alternative is Multiple Hypothesis Tracking (MHT) [84]. A popular technique for computer vision is the mixture particle filter [85], which has been combined with the AdaBoost machine learning algorithm [36] to create the Boosted Particle Filter (BPF), which has been demonstrated by tracking hockey players during a game [86]. The Probability Hypothesis Density (PHD) filter [87] has become very popular outside the field of computer vision, and exists in forms which parallel the various versions of the Kalman filter [88] and the particle filter [89]. It is able to generate probability distributions for the cardinality of the targets by using finite set statistics [90].

There are techniques that combine aspects of kernel methods and recursive Bayesian filtering, such as the approach in [91], which uses Gaussian mixture distributions that are filtered by a modified particle filter, and decides which mode represents the most likely hypothesis by running the mean shift algorithm over a range of kernel bandwidths.

## 2.3 Conclusions

This chapter has summarised some of the literature regarding detection and tracking in the field of computer vision. The limitations of current techniques along with the state of the art are now considered, and these issues are used to inform the research undertaken in this thesis.

Many existing methods for detecting people in images suffer from the limitation of being unable to meet the earlier stated latency requirement of being able to evaluate a

$640 \times 480$ pixel image in less than a second on a desktop computer. One of the faster methods is the integral channel features detector [6], which also shows promising results in terms of accuracy. For this reason, this detector will be used as a starting point in Chapter 3.

For tracking, recursive Bayesian filters are limited by the fact they require a method for detection. The mean-shift tracking algorithm is limited by the fact that existing methods for tracking through scale are prone to failure, or require more operations by an order of magnitude. Thus, one direction for research would be to formulate methods for mean-shift tracking through scale, which will be explored in Chapter 5.

# Chapter 3

# Detecting People in Images

This chapter addresses the problem of detecting instances of people in images. As has been seen in Section 2.1.1, there are a variety of methods to accomplish this task. When addressing such a specific problem, it is often useful to use a well established method as a starting point. The approach used here is based on the integral channel features detector [47]. As was mentioned in Section 2.3, one of the benefits of this method is its speed and accuracy over other techniques. This detector learns a sliding window classifier using the AdaBoost algorithm applied to gradient orientation and colour features. The detector is described in the following three sections, starting with an explanation of the features used in Section 3.1. In Section 3.2 the AdaBoost algorithm used to learn the classifier is introduced. Section 3.3 explains how the classifier is used to detect instances of people in images of arbitrary size. Results for the detector using various different parameter settings are presented in Section 3.4. Finally, Section 3.5 explains how the speed of the classifier can be increased, and the chapter ends with a discussion and summary.

## 3.1 Features

Let $\mathbf{I} \in \mathbb{R}^{m_1 \times m_2 \times 3}$ be an RGB image, with elements $\mathbf{I}(j,k,c) \in \mathbb{R}$ where $j$ and $k$ indicate the row and column of a pixel, and $c$ indicates the channel, which are ordered as red, green and blue. Individual channels of an image are denoted by $\mathbf{I}^c \in \mathbb{R}^{m_1 \times m_2}$. Note that a *window*, which is a rectangular region within an image will also be denoted by $\mathbf{I}$.

As has been mentioned, it is often necessary to extract features from images in order to perform tasks such as classification. A feature map is denoted by $\mathbf{\Phi}(\mathbf{I}) \in$

$\mathbb{R}^{m_1 \times m_2 \times m_3}$, where $m_3$ is finite and positive. As before, the entries of a feature map are $\mathbf{\Phi}(\mathbf{I}, j, k, c) \in \mathbb{R}$, although it should be noted that the number of channels is now different. An individual feature channel is denoted by $\mathbf{\Phi}^c(\mathbf{I}) \in \mathbb{R}^{m_1 \times m_2}$. For brevity of notation, occasionally the $\mathbf{I}$ will be dropped, to give $\mathbf{\Phi}$ or $\mathbf{\Phi}(j, k, c)$.

The default feature map that will be used is the same as that in [47], which has ten channels. The first six channels of $\mathbf{\Phi}$ are gradient orientation channels, the seventh is a gradient magnitude channel, and the last three channels are CIELUV colour space channels. Images are filtered with a binomial filter prior to feature extraction. How these channels are generated is explained next.

### 3.1.1 Image Filtering

Before feature extraction takes place, it is common to apply filtering techniques to reduce image noise or to reduce the variance of input images. The binomial filter is used here. A one dimensional binomial filter $\mathbf{h}_l$ is simply a normalised vector of binomial coefficients of length $l$. For example, possible filters are $\mathbf{h}_3 = \frac{1}{4}\left[\begin{smallmatrix}1 & 2 & 1\end{smallmatrix}\right]^\mathsf{T}$ and $\mathbf{h}_5 = \frac{1}{16}\left[\begin{smallmatrix}1 & 4 & 6 & 4 & 1\end{smallmatrix}\right]^\mathsf{T}$ (note that $(\cdot)^\mathsf{T}$ denotes a transpose). Only binomial filters with an odd length are considered. An image is smoothed by convolving each channel with $\mathbf{h}_l$ and then $\mathbf{h}_l{}^\mathsf{T}$. This is more efficient than convolving by the two dimensional filter $\mathbf{h}_l\mathbf{h}_l{}^\mathsf{T}$, which would give the same result.

### 3.1.2 Gradient Channels

The feature map that will be used will contain a gradient magnitude channel and several gradient orientation channels. Gradient orientation channels contain magnitude information for a specific orientation range. Such features are very popular for object detection [4].

Image gradients are generated by convolving an image with a difference operator. In the case of a colour image, a finite difference operator can be convolved with each channel to find gradients in the $x$ and $y$ directions

$$\mathbf{G}_x^c = \left[\begin{smallmatrix}-1 & 0 & 1\end{smallmatrix}\right] * \mathbf{I}^c, \tag{3.1}$$

$$\mathbf{G}_y^c = \begin{bmatrix}-1\\0\\1\end{bmatrix} * \mathbf{I}^c, \tag{3.2}$$

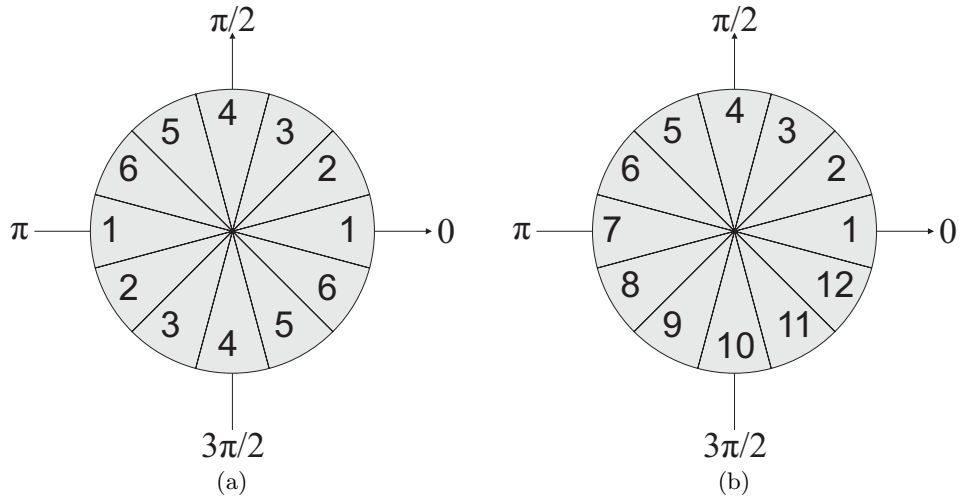



Figure 3.1: The binning process can be (a) contrast insensitive (shown for $m_b = 6$), where gradients that go from light to dark or from dark to light regions are treated in the same way if they have the same orientation, or (b) contrast sensitive (shown for $m_b = 12$) where there are separate bins for gradients with the same orientation, but different contrast directions.

where $\mathbf{G}_x^c, \mathbf{G}_y^c \in \mathbb{R}^{m_1 \times m_2}$. Values at the border of $\mathbf{I}^c$ are replicated to ensure that the convolution can be computed at the boundaries of the image. The magnitude of an entry for a channel can be computed as

$$\mathbf{G}_{mag}^c(j,k) = \sqrt{\mathbf{G}_x^c(j,k)^2 + \mathbf{G}_y^c(j,k)^2}. \tag{3.3}$$

To generate gradient orientation channels, for each pixel a gradient magnitude and gradient orientation must be defined. A popular method to do this for colour images is to use the magnitude and orientation from the colour channel with the largest gradient magnitude value. Let $c_{max}(j,k) = \arg\max_c \{\mathbf{G}_{mag}^c(j,k)\}$. For brevity, let $c_{max}(j,k)$ be abbreviated to $c_{max}$. The gradient magnitude and orientation can be expressed as

$$\mathbf{G}_{mag}(\mathbf{I},j,k) = \max_c \{\mathbf{G}_{mag}^c(j,k)\}, \tag{3.4}$$

$$\mathbf{G}_\theta(\mathbf{I},j,k) = \arctan\left(\frac{\mathbf{G}_y^{c_{max}}(j,k)}{\mathbf{G}_x^{c_{max}}(j,k)}\right), \tag{3.5}$$

where $\mathbf{G}_\theta(\mathbf{I},j,k) \in [0, 2\pi)$. Gradient orientation channels are constructed by assigning gradient magnitudes to bins based on the value of $\mathbf{G}_\theta(\mathbf{I},j,k)$. Let the number of bins be $m_b$. The range $[0, 2\pi)$ is divided in to equal sized bins. Let $b_\theta(\cdot)$ be a function that

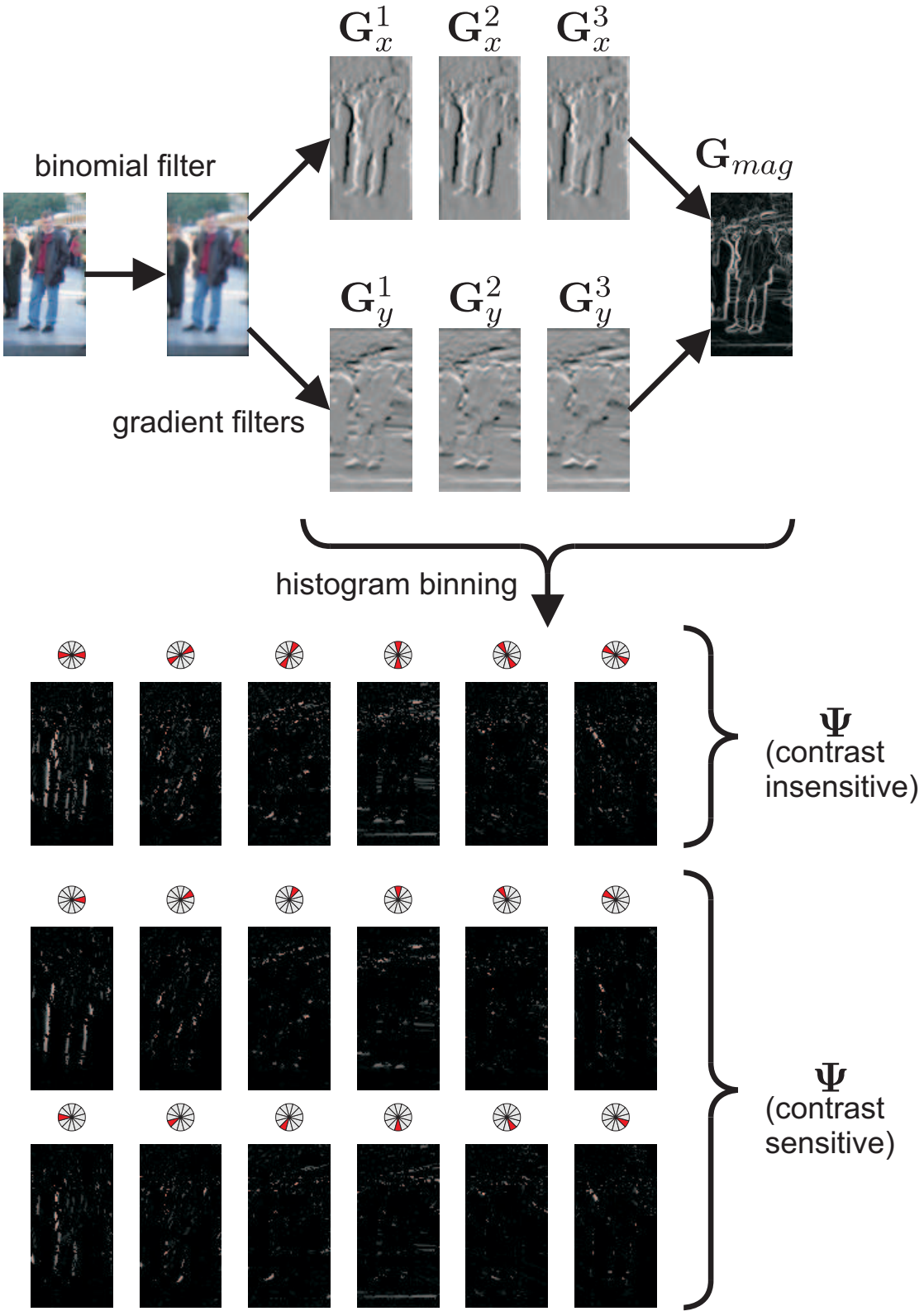



Figure 3.2: Gradient features are generated from the process shown above. First, images are smoothed with a binomial filter. Gradient filters are then applied to each colour channel in the $x$ and $y$ directions. These results are used to calculate the magnitude and orientation of the gradients, which are used to construct gradient orientation channels. Gradient orientation channels can be contrast sensitive, or contrast insensitive.

maps gradient orientations to bin indices so that $b_\theta : [0, 2\pi) \rightarrow \{1, \ldots, m_b\}$. It should be noted that the binning process can be contrast insensitive or contrast sensitive. Image gradients along an axis can represent transitions from high intensity regions to low intensity regions (i.e. $\mathbf{I}(j,k,c) > \mathbf{I}(j+2,k,c)$) or vice versa. Contrast sensitive binning makes a distinction between gradients with the same visual orientation, but differing directions of contrast, while contrast insensitive binning does not. Figure 3.1 illustrates the difference between these two types of binning, as well as the layout of the bins. Gradient orientation channels are then defined as

$$\boldsymbol{\Psi}(\mathbf{I}, j, k, c) = \begin{cases} \mathbf{G}_{mag}(\mathbf{I}, j, k) & \text{if} \quad c = b(\mathbf{G}_\theta(\mathbf{I}, j, k)) \\ 0 & \text{otherwise} \end{cases} . \tag{3.6}$$

The process of creating gradient orientation channels is illustrated in Figure 3.2. In the default setup for generating features, only contrast insensitive features will be used. However, the performance of contrast sensitive features will be examined later in Section 3.4.5.

### 3.1.3 CIELUV Channels

The second type of feature used is based on the CIELUV colour space. A variety of colour spaces exist, but the CIELUV space is used here as this has been shown to improve results in pedestrian detection [47]. This colour space is designed to achieve perceptual uniformity, so that Euclidean distances in this space correspond to perceptual differences in the colour [92]. It consists of three channels denoted $L^*$, $u^*$ and $v^*$. The $L^*$ channel measures luminance (relative brightness), while the $u^*$ and $v^*$ channels measure chromacity (colour).

### 3.1.4 Channel Features

The features that will be used by the classifiers developed in this thesis are simple rectangular sums over feature map channels, as in [47]. These are often referred to as *channel features*. Let $\boldsymbol{\rho} = [\, x_{\boldsymbol{\rho}} \; y_{\boldsymbol{\rho}} \; w_{\boldsymbol{\rho}} \; h_{\boldsymbol{\rho}} \; c_{\boldsymbol{\rho}} \,]^\mathsf{T}$ be a vector containing the parameters that

describe a feature. A feature $g \in \mathbb{R}$ is computed on an image $\mathbf{I}$ by

$$g(\mathbf{I}, \boldsymbol{\rho}) = \sum_{\substack{(x_{\boldsymbol{\rho}} \leq j < x_{\boldsymbol{\rho}} + w_{\boldsymbol{\rho}}) \wedge \\ (y_{\boldsymbol{\rho}} \leq k < y_{\boldsymbol{\rho}} + h_{\boldsymbol{\rho}})}} \boldsymbol{\Phi}^{c_{\boldsymbol{\rho}}}(\mathbf{I}, j, k). \tag{3.7}$$

These features summarise the intensity of a channel in a particular area within an image. For example, in the the far left of Figure 3.3, the translucent red rectangle represents the area covered by a feature for a vertical edge channel. This feature could indicate the presence or absence of a leg. Combining information from multiple features can give discriminative information that could indicate the presence or absence of a particular object (how the features are combined is addressed later in Section 3.2). Channel features can overlap, and the number of features that can be generated for even a small image is very large. For a feature map $\boldsymbol{\Phi}$ that is $m_1 \times m_2 \times m_3$, an expression for the number of possible features $N_f$ can be derived by first considering the number of rectangular features of a fixed width $w$ and height $h$ for a single channel

$$N_{w,h} = (m_1 - w + 1)(m_2 - h + 1). \tag{3.8}$$

The total number of features $N_f$ will simply be $N_{w,h}$ summed over every value of $w$ and $h$ and multiplied by the number of channels $m_3$

$$\begin{aligned}
N_f &= m_3 \sum_{w=1}^{m_1} \sum_{h=1}^{m_2} N_{w,h} \\
&= m_3 \sum_{w=1}^{m_1} \sum_{h=1}^{m_2} (m_1 - w + 1)(m_2 - h + 1) \\
&= m_3 \sum_{w=1}^{m_1} (m_1 - w + 1) \sum_{h=1}^{m_2} (m_2 - h + 1) \\
&= (m_1(m_1 + 1) - \sum_{w=1}^{m_1} w)(m_2(m_2 + 1) - \sum_{h=1}^{m_2} h)m_3 \\
&= (m_1(m_1 + 1) - \frac{1}{2} m_1(m_1 + 1))(m_2(m_2 + 1) - \frac{1}{2} m_2(m_2 + 1))m_3 \\
&= \frac{1}{4} m_1(m_1 + 1)m_2(m_2 + 1)m_3.
\end{aligned} \tag{3.9}$$

For the purposes of learning a detector in Section 3.2, the feature maps for the training set will have dimensions of $m_1 = 128$, $m_2 = 64$ and $m_3 = 10$ as the default values. This

yields a total of 171,724,800 features. This set of features is far too large to be fully explored, and so a common practice is to subsample the spatial dimensions $m_1$ and $m_2$ by some factor $M$ to reduce the number of features by a factor of roughly $M^4$, as can be seen from Equation 3.9. The factor $M$ is usually referred to in literature as the "shrink factor" [47] [50]. For a shrink factor of 4, the number of features is 718,080. Even with this reduction in the number of features, full exploration of the feature space is not possible with the hardware resources used throughout this thesis (a 32-bit computer with 4 gigabytes of Random Access Memory (RAM)) if we wish to hold all the training data in RAM at once. A shrink factor of 4 will be used by default throughout this work.

It should be noted that channel features can be computed quickly by computing the integral histogram of $\mathbf{\Phi}$ [93]. The integral histogram is a generalisation of the integral image [36], and can be used to calculate any rectangular channel sum by looking up only four values. An integral histogram $\mathbf{\Lambda} \in \mathbb{R}^{m_1 \times m_2 \times m_3}$ is formed by summing all values above and to the left of each entry of a feature map $\mathbf{\Phi}$

$$\mathbf{\Lambda}^c(\mathbf{I}, j, k) = \mathbf{\Lambda}(\mathbf{I}, j, k, c) = \sum_{j' \leq j, k' \leq k} \mathbf{\Phi}^c(\mathbf{I}, j', k'). \tag{3.10}$$

Equation 3.7 can then be rewritten as

$$g(\mathbf{I}, \boldsymbol{\rho}) = \mathbf{\Lambda}^{c_{\boldsymbol{\rho}}}(\mathbf{I}, x_{\boldsymbol{\rho}}, y_{\boldsymbol{\rho}}) - \mathbf{\Lambda}^{c_{\boldsymbol{\rho}}}(\boldsymbol{I}, x_{\boldsymbol{\rho}} + w_{\boldsymbol{\rho}}, y_{\boldsymbol{\rho}}) - \mathbf{\Lambda}^{c_{\boldsymbol{\rho}}}(\boldsymbol{I}, x_{\boldsymbol{\rho}}, y_{\boldsymbol{\rho}} + h_{\boldsymbol{\rho}}) + \mathbf{\Lambda}^{c_{\boldsymbol{\rho}}}(\boldsymbol{I}, x_{\boldsymbol{\rho}} + w_{\boldsymbol{\rho}}, y_{\boldsymbol{\rho}} + h_{\boldsymbol{\rho}}). \tag{3.11}$$

Figure 3.3 presents a visualisation of Equation 3.11. To generate an integral histogram from a feature map, all the entries of the feature map must be positive. This condition is met by the gradient channels described in Section 3.1.2, but not by the CIELUV channels in Section 3.1.3. The ranges for the chromacity channels $u^*$ and $v^*$ are $[-134, 220]$ and $[-140, 122]$ respectively. To satisfy the positivity condition, an offset is added to all chromacity values to shift the ranges to $[0, 354]$ and $[0, 262]$.

### 3.1.5 Summary

To summarise, this section has addressed the type of features that will be used to learn a detector. After smoothing the input image with a binomial filter, a feature map with

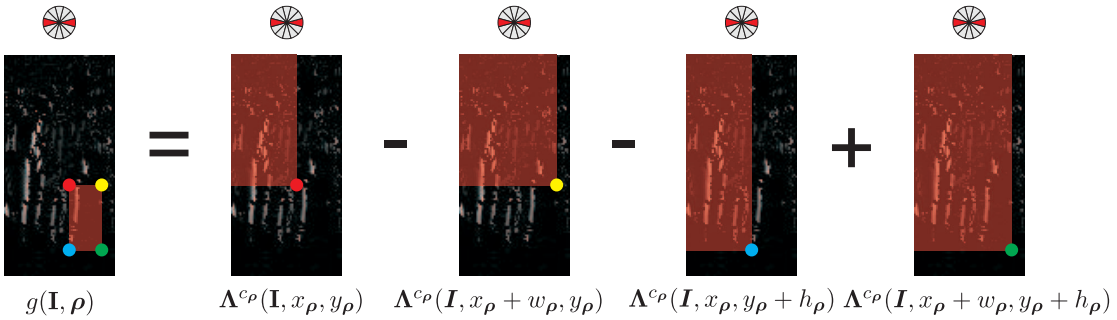



Figure 3.3: A channel feature is the sum over a rectangular area within a channel, shown on the far left. The four corners of the channel feature are shown as coloured circles. Integral histogram entries are equal to the sum of all entries from the top left hand corner down to the position of the histogram entry. Thus, four integral histogram entries can be combined as illustrated on the right to give the value of a channel feature.

gradient orientation channels, a gradient magnitude channel and CIELUV channels is generated. This feature map is used to generate an integral histogram so that channel features can be computed quickly with only a few operations. The next section will address how a detector is learned using this feature set.

## 3.2 Learning a Detector

The previous section has outlined the methods that will be used to extract features from images. This section will look at how a classifier, also referred to as a detector in this context, can be created to indicate whether a person is present in an image based on the values of multiple features. Classifiers are created by training them on a labelled set of training images. Here, the AdaBoost algorithm is used in conjunction with binary decision trees. Methods for training will be described along with practical considerations for implementing the algorithm.

### 3.2.1 AdaBoost

The algorithm that will be used to learn a detector is Adaptive Boosting, commonly referred to as AdaBoost. Boosting algorithms are a class of techniques that were originally motivated by attempting to address whether it was possible to create a highly accurate classifier (known as a *strong classifier*) from several inaccurate classifiers (known as *weak classifiers*). It was found that this was possible and further refinements of these concepts led to the creation of AdaBoost [40]. Since its creation, researchers have sought to ex-

plain the behaviour of AdaBoost and offer alternative derivations of the algorithm. One alternative is to view the algorithm as the sequential minimization of an exponential cost function [23] [37]. It can also be shown that AdaBoost can be seen as a form of functional gradient descent [94].

A general version of AdaBoost will now be described. Let a training set be $\mathcal{S} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ where $\mathbf{x}$ belongs to some instance space $\mathcal{X}$and $y \in \{-1, +1\}$ is a label. The aim is to construct a function $f : \mathcal{X} \to \{-1, +1\}$ that can assign the correct label to instances from $\mathcal{X}$ that were not observed during training. It is assumed that there is a weak learning algorithm (known as a weak learner) available $\mathsf{L}(\mathcal{S}, D)$ that accepts a training set $\mathcal{S}$ and a distribution over the training examples $D(i)$ where $D(i) > 0$ and $\sum_{i=1}^{n} D(i) = 1$. The weak learner $\mathsf{L}$ returns a base classifier $h(\mathbf{x})$, also known as a weak classifier, where $h : \mathcal{X} \to \{-1, +1\}$. The base classifier that is returned minimises the error $\epsilon$ defined as

$$\epsilon = \sum_{i:h(\mathbf{x}_i) \neq y_i} D(i). \tag{3.12}$$

It can be seen that the error $\epsilon$ is simply the sum of the weights for all misclassified training examples. To create a strong classifier, $\mathsf{L}(\mathcal{S}, D)$ is called $R$ times (each call is referred to as a round), and each time a different distribution $D$ is used. The distribution $D$ is adjusted between rounds to increase the values for misclassified examples, and decrease the values for correctly classified examples. This leads to each weak classifier compensating for the mistakes of the previous classifiers. The final classifier $f$ is the sign of a combination of the base classifiers

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{r=1}^{R} \alpha_r h_r(\mathbf{x})\right) \tag{3.13}$$

where $\alpha_r$ is defined in Algorithm 1 and $\operatorname{sgn}(\cdot)$ is the signum function.

The full description of AdaBoost is shown in Algorithm 1. It should be noted that there are multiple equivalent formulations of the algorithm, and that the formulation is slightly different when the label values are taken to be $\{-1, +1\}$ rather than $\{0, 1\}$. The description given in Algorithm 1 is general, and so specific details on how AdaBoost is applied to train an object detector are given next:

**Algorithm 1** AdaBoost

**Require:**

- A training set $\mathcal{S} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$.
- A learning algorithm $\mathsf{L}(\mathcal{S}, D)$ that accepts a training set $\mathcal{S}$ and a distribution $D$ and returns a base classifier $h$ where $h : \mathcal{X} \to \{-1, +1\}$.

Initialise $D_0(i) \leftarrow \frac{1}{n}$ for $i = 1, \ldots, n$
Initialise $H_0(\mathbf{x}) \leftarrow 0$
**for** $r \leftarrow 0$ to $R$ **do**
  Let $h_{r+1} \leftarrow \mathsf{L}(\mathcal{S}, D_r)$
  Let $\epsilon_{r+1} \leftarrow \sum_{i:h_{r+1}(\mathbf{x}_i) \neq y_i} D_r(i)$
  **if** $\epsilon_{r+1} \geq \frac{1}{2}$ **then**
    **return** $H_r$
  **end if**
  Let $\alpha_{r+1} = \frac{1}{2} \log\left(\frac{1-\epsilon_{r+1}}{\epsilon_{r+1}}\right)$
  Let $H_{r+1} = H_r + \alpha_{r+1} h_{r+1}$
  Let $D_{r+1}(i) \leftarrow \begin{cases} D_r(i)/2(1-\epsilon_{r+1}) & \text{if} \quad h_{r+1}(\mathbf{x}_i) = y_i \\ D_r(i)/2\epsilon_{r+1} & \text{if} \quad h_{r+1}(\mathbf{x}_i) \neq y_i \end{cases}$
**end for**
**return** $H_{R+1}$

- Training sets for object detection are often highly unbalanced, with many more negative examples than positive ones. To account for this, the initial distribution for boosting $D_0$ is calculated as

$$D_0(i) = \frac{1}{2n_+} \qquad \text{if} \quad y_i = +1, \tag{3.14}$$

$$D_0(i) = \frac{1}{2n_-} \qquad \text{if} \quad y_i = -1, \tag{3.15}$$

  where $n_+$ and $n_-$ are the number of positive and negative training examples respectively. This approach was first adopted in [36].

- The training set for an object detector is a set of features computed from the set of training images. Let an image training set be defined as $\mathcal{S}_\mathbf{I} = \{(\mathbf{I}_1, y_1), \ldots, (\mathbf{I}_n, y_n)\}$. To proceed with training, a vector of features must be extracted for each training example. As the total number of unique features is very large (see Section 3.1.4), usually a random subset is used. Let $\{\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_U\}$ be a set of vectors defining the parameters of $U$ channel features, where by default $U = 30000$ as in [6]. Let a

vector of features $\mathbf{g} \in \mathbb{R}^U$ be defined as

$$\mathbf{g}(\mathbf{I}_i) = \begin{bmatrix} g(\mathbf{I}_i, \boldsymbol{\rho}_1) & \cdots & g(\mathbf{I}_i, \boldsymbol{\rho}_U) \end{bmatrix}^\mathsf{T}, \tag{3.16}$$

where $g(\mathbf{I}, \boldsymbol{\rho})$ is defined in Equation 3.11. The feature training set is then defined as $\mathcal{S}_\mathbf{g} = \{(\mathbf{g}(\mathbf{I}_i), y_1), \ldots, (\mathbf{g}(\mathbf{I}_n), y_n)\}$, and this is the set used to train a detector.

- In general, a weak classifier $h(\mathbf{x})$ is a function of the data vector $\mathbf{x}$. When training an object detector, the data vector is $\mathbf{g}$ as defined in Equation 3.16. Because of the high dimensionality of $\mathbf{g}$, it is not feasible to have a base classifier that processes the entire feature vector. Feature selection must be performed so that a base classifier acts upon only a very small number of the entries.

- The base classifiers that will be used are decision trees. These are explained in the next section.

### 3.2.2 Decision Trees

AdaBoost works by training multiple base classifiers, where the choice of base classifier is left open. The base classifier that will be used in this thesis is the binary decision tree, as these classifiers can be trained and evaluated very quickly compared to alternatives like SVMs and linear discriminants. Decision trees are simple and flexible classifiers, which can be applied to a variety of learning problems. They are composed of split nodes and leaf nodes, with split nodes applying functions to the input to determine which branch to follow next, and leaf nodes representing the decisions returned by the tree. Figure 3.4 shows two balanced binary decision trees with split nodes as blue circles, and leaf nodes as green squares. The depth of a tree refers to the depth of the split nodes, and so the trees in Figure 3.4 have depths of one and two. A decision tree of depth one is also known as a *stump*. This may be seen as the simplest kind of decision tree, and can be used to build more complex trees. A stump $h^{stump}(\mathbf{I}, \boldsymbol{\rho}, \theta, q)$, sometimes abbreviated to $h^{stump}(\mathbf{I})$, can be defined as

$$h^{stump}(\mathbf{I}, \boldsymbol{\rho}, \theta, q) = \begin{cases} +1 & \text{if} \quad qg(\mathbf{I}, \boldsymbol{\rho}) < q\theta \\ -1 & \text{otherwise} \end{cases}, \tag{3.17}$$

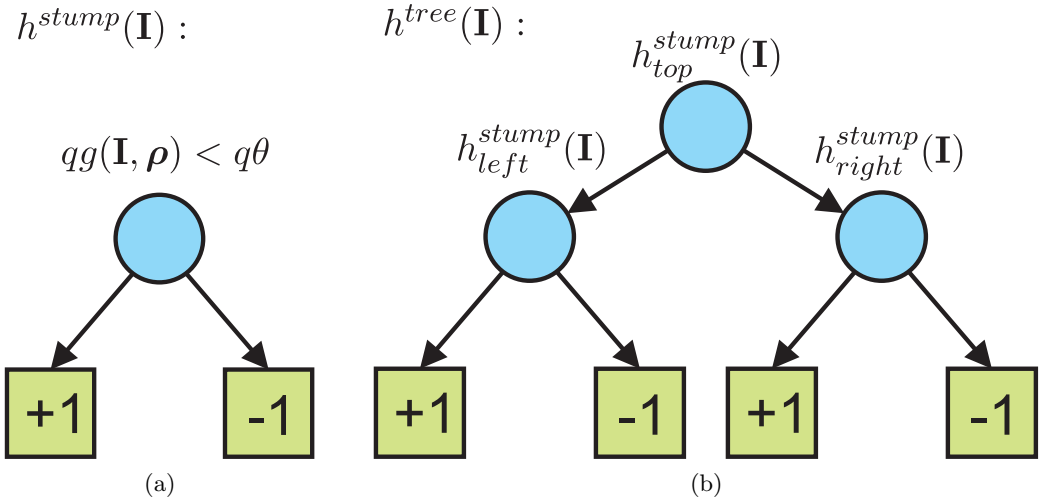



Figure 3.4: (a) A stump classifier takes an input image $I$, evaluates a feature $g$, compares this against a threshold $\theta$, and then returns a label based on the polarity $q$. Returning a label can be seen as selecting a branch. (b) A decision tree is composed of multiple stump classifiers, and is therefore capable of more complex classifications.

where $q \in \{-1, +1\}$ is the polarity of the stump classifier, and $\theta \in \mathbb{R}$ is a threshold. A stump classifier compares a feature value $g$ against a threshold $\theta$. The polarity $q$ controls the type of comparison. The result of the comparison is used to return a label. A diagram of a stump classifier is shown in Figure 3.4a. As a result of their simplicity, stump classifiers perform poorly by themselves. Tree classifiers are slightly more complex, and have a stump classifier for each split node. A depth two decision tree is shown in Figure 3.4b.

The problem of training a stump classifier will now be examined in some detail. It should be mentioned that since training a decision tree involves training several stump classifiers, the discussion here is also relevant to the training of decision trees. To train a stump classifier, the classifier with the lowest error on the training set must be found, where the error is defined in Equation 3.12. A stump classifier is determined by the parameters $\boldsymbol{\rho} = (x_{\boldsymbol{\rho}}, y_{\boldsymbol{\rho}}, w_{\boldsymbol{\rho}}, h_{\boldsymbol{\rho}}, c_{\boldsymbol{\rho}})$, $\theta$ and $q$. Thus, the lowest error achieved by a stump classifier on a training set is

$$\epsilon_{min} = \min_{\boldsymbol{\rho},\theta,q} \left\{ \sum_{i:h^{stump}(\mathbf{I}_i,\boldsymbol{\rho},\theta,q)\neq y_i} D(i) \right\}. \tag{3.18}$$

Finding the best stump classifier involves optimising over three parameters. The pa-

rameters are discrete and the error function is not differentiable. The error function is not convex in all of the parameters either. There are $U$ features, and for each feature, a set of $V$ candidate thresholds $\Theta_u = \{\theta_1^u, \ldots, \theta_V^u\}$ is generated by calculating $V$ evenly spaced values in the interval from the minimum feature value $\min_i \{g(\mathbf{I}_i, \boldsymbol{\rho}_u)\}$ to the maximum feature value $\max_i \{g(\mathbf{I}_i, \boldsymbol{\rho}_u)\}$. Let the error for a particular feature, threshold and polarity choice be denoted

$$\epsilon_{v+}^u = \sum_{i:h^{stump}(\mathbf{I}_i, \boldsymbol{\rho}_u, \theta_v^u, +1) \neq y_i} D(i), \tag{3.19}$$

$$\epsilon_{v-}^u = \sum_{i:h^{stump}(\mathbf{I}_i, \boldsymbol{\rho}_u, \theta_v^u, -1) \neq y_i} D(i), \tag{3.20}$$

where the subscript $+$ or $-$ is used to indicate the value of the polarity $q$. A naive strategy to find the best stump classifier would be to calculate every value of $\epsilon_{v+}^u$ and $\epsilon_{v-}^u$ and choose the set of parameters that produce the minimum value. The number of parameter combinations is $U \times V \times 2$, and the cost of exploring all these combinations is considered next.

Let the computational cost of a stump training algorithm be the approximate number of addition and subtraction operations needed to evaluate the errors. The error is computed by evaluating the right hand side of Equation 3.19 or Equation 3.20, which involves summing over all misclassified examples. Let the number of misclassified examples in the sum be be denoted by $n_{v+}^u$ for $\epsilon_{v+}^u$, and $n_{v+}^u$ for $\epsilon_{v-}^u$. If two stump classifiers differ only by their polarity, then all the examples correctly classified by one will be incorrectly classified by the other, as long as no example has a value equal to the threshold of the classifier. In practice this is rarely the case, and even when the situation does arise, it is unlikely to have a significant impact on the final result. As a consequence of this, it can be shown that $n_{v+}^u + n_{v-}^u = n$, even though the precise values of $n_{v+}^u$ and $n_{v-}^u$ are not known. Therefore, the cost of evaluating the error for every set of parameters is $U \times V \times n$. It should be noted that this is just the cost of training a single stump classifier, and typically thousands of these must be trained for a boosting classifier.

A simple way to improve upon the naive approach is to make use of the observation in the previous paragraph that reversing the polarity of a classifier changes the output label for an input example. Since the weights satisfy the condition $\sum_{i=1}^n D(i)$, this

implies that a classifier with an error of $\epsilon$ will have an error of $1 - \epsilon$ when the polarity is reversed, subject to the conditions mentioned in the previous paragraph. As a result, the error only needs to be evaluated for one polarity, and then the error for the opposite polarity can be calculated through one subtraction operation. This lowers the cost to $U \times V \times (n^u_{v+} + 1)$. It is reasonable to assume that $(n^u_{v+}) \approx \frac{n}{2}$, as $n^u_{v+} + n^u_{v-} = n$, and there is no reason to believe that $n^u_{v+}$ would be significantly more or less than $n^u_{v-}$ on average. Therefore, the cost is approximately $U \times V \times \frac{n}{2}$, which is roughly half that of the naive approach, but still very high. The process for training a stump this way is shown in Algorithm 2. It should be noted that the algorithm requires the candidate thresholds $\Theta_u$ as an input. This is for reasons of efficiency, as the candidate thresholds can be generated once and for all from the feature training set $\mathcal{S}_\mathbf{g}$, and as the stump learning algorithm will be called multiple times by AdaBoost, it makes sense to perform this calculation outside of this loop.

**Algorithm 2** StumpLearnSlow

**Require:**

- A feature training set $\mathcal{S}_\mathbf{g} = \{(\mathbf{g}(\mathbf{I}_i), y_1), \ldots, (\mathbf{g}(\mathbf{I}_n), y_n)\}$ where $\mathbf{g} \in \mathbb{R}^U$.
- A distribution $D$ over the training examples.
- A set of candidate thresholds $\Theta_u = \{\theta^u_1, \ldots, \theta^u_V\}$ for each feature $u = 1, \ldots, U$.

Initialise $\epsilon_{min} \leftarrow 1$, $\boldsymbol{\rho}_{best} \leftarrow ?$, $\theta_{best} \leftarrow ?$ and $q_{best} \leftarrow ?$
**for** $u \leftarrow 1$ to $U$ **do**
  **for** $v \leftarrow 1$ to $V$ **do**
    Let $\epsilon^u_{v+} \leftarrow 0$
    **for** $i \leftarrow 1$ to $n$ **do**
      **if** $h^{stump}(\mathbf{I}_i, \boldsymbol{\rho}_u, \theta^u_v, +1) \neq y_i$ **then**
        Let $\epsilon^u_{v+} \leftarrow \epsilon^u_{v+} + D(i)$
      **end if**
    **end for**
    Let $\epsilon^u_{v-} \leftarrow 1 - \epsilon^u_{v+}$
    **if** $\epsilon^u_{v+} < \epsilon_{min}$ **then**
      Let $\epsilon_{min} \leftarrow \epsilon^u_{v+}$, $\boldsymbol{\rho}_{best} \leftarrow \boldsymbol{\rho}_u$, $\theta_{best} \leftarrow \theta^u_v$ and $q_{best} \leftarrow +1$
    **end if**
    **if** $\epsilon^u_{v-} < \epsilon_{min}$ **then**
      Let $\epsilon_{min} \leftarrow \epsilon^u_{v-}$, $\boldsymbol{\rho}_{best} \leftarrow \boldsymbol{\rho}_u$, $\theta_{best} \leftarrow \theta^u_v$ and $q_{best} \leftarrow -1$
    **end if**
  **end for**
**end for**
**return** $\epsilon_{min}, \boldsymbol{\rho}_{best}, \theta_{best}, q_{best}$

It is possible to vastly improve on the speed of Algorithm 2. The calculation of the

error can be broken down into components which can be computed separately and reused to calculate $\epsilon^u_{v+}$ for different values of $v$. Various algorithms exploit the structure of a problem to accelerate computation, such as the fast Fourier transform [95], which speeds up the computation of the discrete Fourier transform using a divide and conquer strategy. For training a stump classifier, the aspect that allows the training to be accelerated is the fact that the candidate thresholds $\Theta_u$ form an *ordered set*, where $\theta^u_v < \theta^u_{v+1}$. Before this can be demonstrated, the notion of quantising features must be introduced. Feature values $g(\mathbf{I}_i, \boldsymbol{\rho}_u) \in \mathbb{R}$ can be mapped to discrete values $\gamma^u_i \in \{1, \ldots, V+1\}$ by assigning them to bins defined by the thresholds $\Theta_u$

$$\gamma^u_i = \begin{cases} 1 & \text{if} \quad g(\mathbf{I}_i, \rho_u) < \theta^u_1 \\ 2 & \text{if} \quad \theta^u_1 \leq g(\mathbf{I}_i, \rho_u) < \theta^u_2 \\ \vdots & \\ V & \text{if} \quad \theta^u_{V-1} \leq g(\mathbf{I}_i, \rho_u) < \theta^u_V \\ V+1 & \text{if} \quad g(\mathbf{I}_i, \rho_u) \geq \theta^u_V \end{cases} . \tag{3.21}$$

It will later be shown that the discretised feature values $\gamma^u_i$ can be used for training rather than the feature values $g(\mathbf{I}_i, \boldsymbol{\rho}_u)$. A discrete feature training set $\mathcal{S}_{\boldsymbol{\Gamma}} = \{(\boldsymbol{\Gamma}(\mathbf{I}_i), y_1), \ldots, (\boldsymbol{\Gamma}(\mathbf{I}_n), y_n)\}$ is defined where $\boldsymbol{\Gamma}(\mathbf{I}_i) \in \mathbb{R}^U$ is a discrete feature vector

$$\boldsymbol{\Gamma}(\mathbf{I}_i) = \begin{bmatrix} \gamma^1_i & \cdots & \gamma^U_i \end{bmatrix}^\mathsf{T} . \tag{3.22}$$

It will now be shown that the error can be broken down into simpler components. It is observed that the error $\epsilon^u_{v+}$ defined in Equation 3.19 may be expressed as

$$\begin{aligned} \epsilon^u_{v+} &= \sum_{\substack{i: \\ (y_i=+1) \wedge \\ (h^{stump}(\mathbf{I}_i, \boldsymbol{\rho}_u, \theta^u_v, +1) = -1)}} D(i) + \sum_{\substack{i: \\ (y_i=-1) \wedge \\ (h^{stump}(\mathbf{I}_i, \boldsymbol{\rho}_u, \theta^u_v, +1) = +1)}} D(i) \\ &= \sum_{\substack{i: \\ (y_i=+1) \wedge \\ (g(\mathbf{I}_i, \boldsymbol{\rho}_u) \geq \theta^u_v)}} D(i) + \sum_{\substack{i: \\ (y_i=-1) \wedge \\ (g(\mathbf{I}_i, \boldsymbol{\rho}_u) < \theta^u_v)}} D(i). \end{aligned} \tag{3.23}$$

Stated in words, Equation 3.23 indicates that the error is the sum of the error over positive examples and the error over negative examples. The second line is arrived at by

substituting Equation 3.17 for $h^{stump}(\mathbf{I}_i, \boldsymbol{\rho}_u, \theta_v^u, +1)$. To simplify the notation, let $e_v^{u+}$ and $e_v^{u-}$ be defined as

$$e_v^{u+} = \sum_{\substack{i: \\ (y_i=+1)\wedge \\ (g(\mathbf{I}_i,\boldsymbol{\rho}_u)\geq\theta_v^u)}} D(i), \tag{3.24}$$

$$e_v^{u-} = \sum_{\substack{i: \\ (y_i=-1)\wedge \\ (g(\mathbf{I}_i,\boldsymbol{\rho}_u)<\theta_v^u)}} D(i), \tag{3.25}$$

so that $\epsilon_{v+}^u = e_v^{u+} + e_v^{u-}$. Note that the superscript + or − indicates the values of $y_i$. Because the set of candidate thresholds $\Theta_u = \{\theta_1^u, \ldots, \theta_V^u\}$ are in increasing order, it can be observed that $e_v^{u+}$ and $e_v^{u-}$ can be written recursively as

$$e_v^{u+} = e_{v+1}^{u+} + \sum_{\substack{i: \\ (y_i=+1)\wedge \\ (\theta_v^u\leq g(\mathbf{I}_i,\boldsymbol{\rho}_u)<\theta_{v+1}^u)}} D(i) \qquad \text{for} \quad v < V, \tag{3.26}$$

$$e_V^{u+} = \sum_{\substack{i: \\ (y_i=+1)\wedge \\ (g(\mathbf{I}_i,\boldsymbol{\rho}_u)\geq\theta_V^u)}} D(i), \tag{3.27}$$

$$e_v^{u-} = e_{v-1}^{u-} + \sum_{\substack{i: \\ (y_i=-1)\wedge \\ (\theta_{v-1}^u\leq g(\mathbf{I}_i,\boldsymbol{\rho}_u)<\theta_v^u)}} D(i) \qquad \text{for} \quad v > 1, \tag{3.28}$$

$$e_1^{u-} = \sum_{\substack{i: \\ (y_i=-1)\wedge \\ (g(\mathbf{I}_i,\boldsymbol{\rho}_u)<\theta_1^u)}} D(i). \tag{3.29}$$

By substituting Equation 3.21 into the previous set of equations, they can be expressed as

$$e_v^{u+} = e_{v+1}^{u+} + \sum_{i:(y_i=+1)\wedge(\gamma_i^u=v+1)} D(i) \qquad \text{for} \quad v < V, \tag{3.30}$$

$$e_V^{u+} = \sum_{i:(y_i=+1)\wedge(\gamma_i^u=V+1)} D(i), \tag{3.31}$$

$$e_v^{u-} = e_{v-1}^{u-} + \sum_{i:(y_i=-1)\wedge(\gamma_i^u=v)} D(i) \qquad \text{for} \quad v > 1, \tag{3.32}$$

$$e_1^{u-} = \sum_{i:(y_i=-1)\wedge(\gamma_i^u=1)} D(i). \tag{3.33}$$

Thus, $e_v^{u+}$ and $e_v^{u-}$ can be calculated for $v = 1, \ldots, V$ if $\sum_{i:(y_i=-1)\wedge(\gamma_i^u=\gamma)} D(i)$ is known

for $\gamma = 1, \ldots, V+1$. Let $d_\gamma^{u+} = \sum_{i:(y_i=+1)\wedge(\gamma_i^u=\gamma)} D(i)$ and $d_\gamma^{u-} = \sum_{i:(y_i=-1)\wedge(\gamma_i^u=\gamma)} D(i)$. The values of $d_\gamma^{u+}$ and $d_\gamma^{u-}$ are simply the sums of weights which have the same label $y_i$ and also the same discrete feature value $\gamma_i^u$. These can easily be computed from the discrete feature training set $\mathcal{S}_\mathbf{\Gamma}$. The full process for training a stump classifier is shown in Algorithm 3.

Now the complexity of Algorithm 2 and Algorithm 3 will be compared. As has already been seen, the approximate number of additions and subtractions for Algorithm 2 is roughly $U \times V \times \frac{n}{2}$. In Algorithm 3, an outer loop iterates over each feature $U$ times. Within this loop, there are four loops where addition and subtraction operations take place. The first of these iterates $n$ times over each training example and performs one addition per iteration. The second and third loops iterate $V-1$ times each, performing one addition per iteration. The fourth loop iterates $V$ times and performs one addition and one subtraction per iteration. Thus, the total number of additions and subtractions for Algorithm 3 is $U \times (n + 4V - 2)$. Usually $n >> V$, with typical values of $n = 12000$ and $V = 256$, and so Algorithm 3 is a considerable improvement over Algorithm 2. It is important to stress that in practice, the speed of an algorithm relies not only on the number of operations, but the type of operation, and other factors relating to hardware, such as whether memory is accessed sequentially or randomly, and the number of branch prediction errors. As a result, the ultimate measure of speed can only come through actual experiments, which confirm that Algorithm 3 is faster. It will also be seen in Section 3.2.3 that Algorithm 3 can provide vast improvements in memory consumption.

Figure 3.5 shows the time taken for training a boosting classifier using Algorithms 2 and 3, with the training time plotted against the total rounds of boosting. It should be noted that the $y$-axis of the graph is logarithmic. For the training runs in Figure 3.5, training parameters were fixed so that $U = 30000$, $n = 7288$, $V = 255$ and stump classifiers were used as weak classifiers. The total rounds of boosting were varied, and it can be seen that Algorithm 3 is faster than Algorithm 2 by a least one order of magnitude.

A decision tree is created by training a stump classifier for each split node in the tree. Thus, a decision tree learning algorithm would repeatedly call Algorithm 2 or Algorithm 3. In this thesis, a greedy training algorithm is used, where the best performing

**Algorithm 3** StumpLearnFast

**Require:**

- A discrete feature training set $\mathcal{S}_{\mathbf{\Gamma}} = \{(\mathbf{\Gamma}(\mathbf{I}_i), y_1), \ldots, (\mathbf{\Gamma}(\mathbf{I}_n), y_n)\}$ where $\mathbf{\Gamma} \in \mathbb{R}^U$
- A distribution $D$ over the training examples.
- A set of candidate thresholds $\Theta_u = \{\theta_1^u, \ldots, \theta_V^u\}$ for each feature $u = 1, \ldots, U$.

Initialise $\epsilon_{min} \leftarrow 1$, $\boldsymbol{\rho}_{best} \leftarrow ?$, $\theta_{best} \leftarrow ?$ and $q_{best} \leftarrow ?$
**for** $u \leftarrow 1$ to $U$ **do**
  **for** $\gamma \leftarrow 1$ to $V + 1$ **do**
    Initialise $d_\gamma^{u+} \leftarrow 0$
    Initialise $d_\gamma^{u-} \leftarrow 0$
  **end for**
  **for** $i \leftarrow 1$ to $n$ **do**
    Let $\gamma \leftarrow \gamma_i^u$
    **if** $y_i = +1$ **then**
      Let $d_\gamma^{u+} \leftarrow d_\gamma^{u+} + D(i)$
    **else**
      Let $d_\gamma^{u-} \leftarrow d_\gamma^{u-} + D(i)$
    **end if**
  **end for**
  Let $e_V^{u+} \leftarrow d_{V+1}^{u+}$
  **for** $v \leftarrow V - 1$ to 1 **do**
    Let $e_v^{u+} \leftarrow e_{v+1}^{u+} + d_{v+1}^{u+}$
  **end for**
  Let $e_1^{u-} \leftarrow d_1^{u-}$
  **for** $v \leftarrow 2$ to $V$ **do**
    Let $e_v^{u-} \leftarrow e_{v-1}^{u-} + d_v^{u-}$
  **end for**
  **for** $v \leftarrow 1$ to $V$ **do**
    Let $\epsilon_{v+}^u \leftarrow e_v^{u+} + e_v^{u-}$
    Let $\epsilon_{v-}^u \leftarrow 1 - \epsilon_{v+}^u$
    **if** $\epsilon_{v+}^u < \epsilon_{min}$ **then**
      Let $\epsilon_{min} \leftarrow \epsilon_{v+}^u$, $\boldsymbol{\rho}_{best} \leftarrow \boldsymbol{\rho}_u$, $\theta_{best} \leftarrow \theta_v^u$ and $q_{best} \leftarrow +1$
    **end if**
    **if** $\epsilon_{v-}^u < \epsilon_{min}$ **then**
      Let $\epsilon_{min} \leftarrow \epsilon_{v-}^u$, $\boldsymbol{\rho}_{best} \leftarrow \boldsymbol{\rho}_u$, $\theta_{best} \leftarrow \theta_v^u$ and $q_{best} \leftarrow -1$
    **end if**
  **end for**
**end for**
**return** $\epsilon_{min}, \boldsymbol{\rho}_{best}, \theta_{best}, q_{best}$

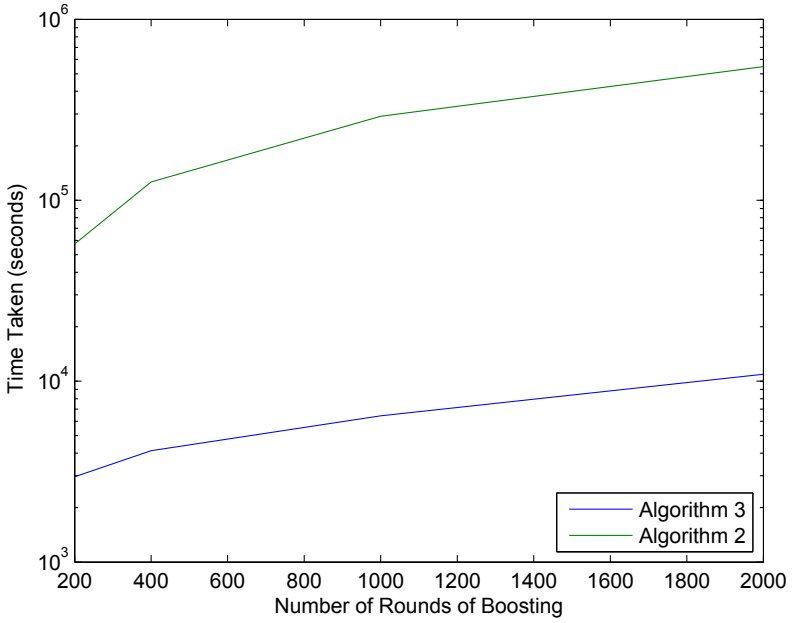



Figure 3.5: The training time for a boosting classifier plotted against the number of rounds of boosting. As can be seen, Algorithm 3 consistently outperforms Algorithm 2. For all training runs in this graph, training parameters were fixed so that $U = 30000$, $n = 7288$, $V = 255$ and stump classifiers were used as weak classifiers.

stump classifier is trained for each split node, and only balanced trees are considered. The training process begins at the top of a tree. After a stump classifier has been trained for a node, the training set for that node is partitioned into two new smaller sets based on how each training example is labelled by the newly trained classifier. These two new sets are used to train the two child nodes, and the training procedure descends through the tree in this fashion until the maximum specified depth is reached. The actual implementation of the algorithm performs the training in a breadth first manner, and so the training time is approximately linear in the tree depth. For example, depth two decision trees take roughly twice as long to train as stump classifiers.

### 3.2.3 Implementation Issues

It has been shown how a person detector can be trained with AdaBoost applied to binary decision trees. The actual implementation of the algorithm for the experiments in this thesis is written in the C++ programming language. When implementing the algorithm, practical considerations must be made in regard to the amount of memory available. Algorithms generally work at a higher speed if the data that they process is

stored in a computer's RAM rather than on the hard drive. However, RAM is limited. Let us consider the memory consumption of a feature training set $\mathcal{S}_{\mathbf{g}}$. The majority of the memory is taken up by the training vectors $\mathbf{g} \in \mathbb{R}^U$, and so the labels $y_i$ and candidate thresholds $\Theta_u$ will not be considered in these calculations. Each entry of the vector $\mathbf{g}$ is a real number, and can be stored as a single precision floating point value. Labels can be stored as boolean values. When programming in C++, the size of a variable type is not fixed by the language standard, and may vary between compilers, but for a large number of compilers a single precision floating point value consumes four bytes. A training set will consist of $U \times n$ of these values. Thus, the memory requirements scale linearly with the number of training examples and the number of features. Typically $U = 30000$ and the $n \approx 13000$. As has been mentioned, in this thesis we are specifically interested in the implementation of training on a desktop computer with a 32 bit architecture and 4 gigabytes of RAM, but it is worth considering how the issues that may be encountered would vary depending on the resources available. In terms of software, an alternative implementation could use MATLAB rather than C++. The memory requirements in this case would be the same, but an overhead would be incurred for running the MATLAB console. Also, MATLAB is highly optimised for operations involving matrices, of which there are none in this algorithm. In terms of hardware, alternative platforms could include 64 bit computers and graphics processing units. Computers with a 64 bit architecture could have more than 4 gigabytes of memory, allowing more training examples and features to be used, and a 32 bit implementation could be recompiled to work on such a platform. Graphics processing units would allow the training of individual weak classifiers to be accelerated, but would require an implementation using an appropriate application programming interface.

With the previously stated typical values of $U = 30000$ and $n \approx 13000$, and the assumption that a single precision floating point value requires 4 bytes of storage, the total memory that is used for training is 1.34 gigabytes. On a 32 bit computer with 4 gigabytes of RAM, only 3 gigabytes of RAM are available, and a proportion of this will be used by the operating system. While the training set under these settings is within limits, it is not possible to double the number of features or training examples without running out of memory.

The memory consumption of the training algorithm can be vastly decreased by using a quantised feature set $\mathcal{S}_{\mathbf{\Gamma}}$, as is required by Algorithm 3. In this setup, the entries of a vector $\gamma \in \mathbb{R}^U$ are integers. If $V$ is bounded at 255, then $\gamma \in \{1, \ldots, 256\}$. This means $\gamma$ can be stored in a single byte, rather than the four bytes required for $g$. Thus, a fourfold reduction in memory consumption can be achieved.

The training set used throughout this thesis is the INRIA person training dataset [51]. The negative training images in this set can be used to generate millions of negative training examples by sampling windows at arbitrary scales and translations, but as has been illustrated, it is only practical to use several thousand. In order to obtain a representative training set from the negative image set, the approach from [47] is used. A classifier is trained with an initial negative set of 5000 randomly sampled windows. The resulting classifier is then applied to the negative training images to gather 5000 additional negative examples, in a process commonly referred to as *bootstrapping*. These additional examples will be false positives, and will therefore be useful for correcting the performance of the classifier. The entire training and bootstrapping process is then repeated again to give up to 5000 more negative training examples, for a maximum total of 15000 negative training examples used to train the final classifier. However, in the last round of bootstrapping, the performance of the classifier has usually improved to the point where the number of false positives is much lower than 5000, and so normally the number of negative training examples generated by this process will be $\approx 10000$.

### 3.2.4 Summary

In this section, the training procedure for training a person detector has been outlined. The AdaBoost algorithm was described, along with decision trees and stump classifiers. An efficient training method was outlined, and practical issues for implementing the algorithm were addressed. The next section will describe how the detector is run on an image at multiple scales to obtain bounding box detections.

## 3.3 Running a Detector

In the previous section, it has been shown how a detector can be trained. The resulting detector can be used to classify images that have the same dimensions as those in the

training set. In this section, it will be shown how the detector can be applied to an image of arbitrary size to detect people at multiple scales. To do this, first an image has to be rescaled multiple times, and an integral histogram must be computed for each scale. Then, the integral histograms will sometimes be padded to allow detection of people who take up a larger extent of the image. At this point, the detector can be evaluated on the integral histograms to obtain a list of bounding boxes. Finally, non-maximum suppression is applied to the resulting bounding boxes with the aim of creating only a single bounding box for each target instance.

### 3.3.1 Image Rescaling

There are a variety of methods for rescaling images. Two widely used approaches are nearest neighbour interpolation and bilinear interpolation. Nearest neighbour interpolation is a very simple approach where every pixel in the new image is assigned the value of the nearest pixel in the original image. Bilinear interpolation assigns a value in the new image by calculating a weighted sum of four pixels in the original image, where the weights are determined by the location of the pixel within the neighbourhood. When downscaling an image, the quality can deteriorate severely, with rapid changes in intensity being introduced, as shown in Figure 3.6b. To mitigate this, the image can be convolved with a kernel prior to downscaling. Figure 3.6c shows an image that was convolved with a box filter prior to downscaling with nearest neighbour interpolation. The dimensions of the box kernel are the inverse of the scale factor. It can be seen that the abrupt changes in intensity present in Figure 3.6b are gone. Throughout this thesis, the default approach for rescaling images will be to convolve the image with a box filter and then apply nearest neighbour interpolation. The effect of box filtering on performance is examined in Section 3.4.2.

### 3.3.2 Padding Integral Histograms

Throughout this thesis, the INRIA Person Dataset is used to train detectors. As was mentioned in Section 2.1.3, the cropped positive training examples provided in the dataset are padded at the borders, so that a $128 \times 64$ pixel image will contain a person who is approximately 96 pixels tall with a border of 16 pixels on each side. The reason

(a) (b) (c)

Figure 3.6: (a) An image. (b) The original image downscaled with nearest neighbour interpolation. (c) The original image convolved with a box filter prior to downscaling with nearest neighbour interpolation. Smoothing with a box filter substantially improves results.

for this given in [4] is that introducing this padding improves detection results. However, as a result of this padding, any person in an image that has a height equal to that of the image can not be detected. To allow for the detection of such instances, padding can be added to the boundaries of an integral histogram.

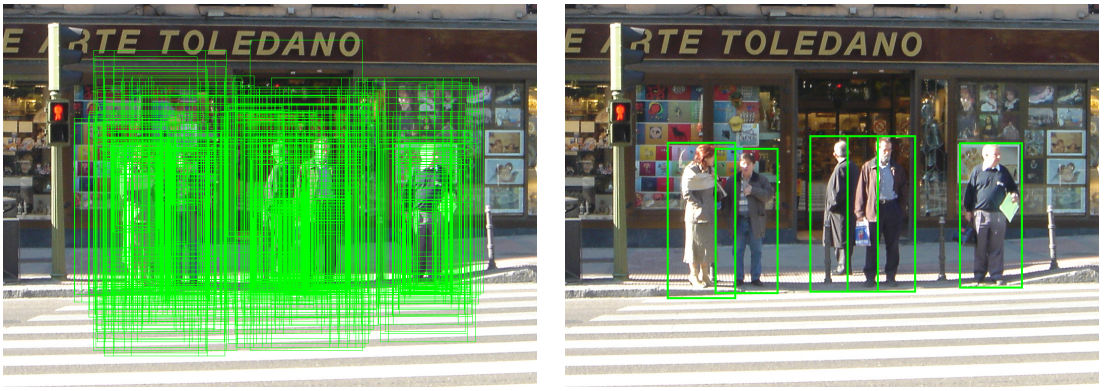
### 3.3.3 Resizing Bounding Boxes

As mentioned in the previous section, the positive training examples from the INRIA Person Dataset have a border around each person. To make sure the bounding boxes output by the detector reflect the extent of a person, they must be resized. We use the scale factors mentioned in the addendum to [47], which are 0.67 for the width and 0.78 for the height of each bounding box. This resizing is carried out prior to non-maximum suppression, described in the next section.

### 3.3.4 Non-maximum suppression

After running a sliding window classifier on an image, there will typically be multiple overlapping bounding boxes around a pedestrian, as shown in Figure 3.7a. The process of reducing these is known as Non-Maximum Suppression (NMS). NMS is necessary, as a reliable evaluation protocol will penalise a detector that produces multiple bounding boxes for a single target instance. Figure 3.7b shows results after NMS has been applied.

A number of NMS techniques exist. A method based on the mean shift algorithm is outlined in [96]. However, much simpler methods are often capable of achieving good results, and so throughout this thesis a method outlined in the addendum of [47] will




(a) (b)

Figure 3.7: The output bounding boxes from a detector shown in green (a) before non-maximum suppression and (b) after non-maximum suppression. It can be seen that non-maximum suppression greatly reduces the number of bounding boxes.

be used. The first step of this method is to sort the bounding boxes into descending order by the real valued score that they obtain from the boosting classifier. The score is simply defined as the value on right hand side in Equation 3.13 prior to evaluating the sgn function. Starting from the bottom of the list and moving upwards, each box below the current bounding box is removed from the list if its score is lower than the current box *and* if the value of an overlap criteria between the two boxes is greater than a defined threshold. The overlap criteria between two bounding boxes is defined as the area of overlap between the two boxes, divided by the area of the smallest box. The default threshold value is 0.65, the same as that used in the addendum of [47].

### 3.3.5 Summary

This section has summarised how a detector is run on an image of arbitrary size to detect instances of people of arbitrary size and translation. The input image is rescaled multiple times with nearest neighbour interpolation preceded by box filtering. An integral histogram is generated from each rescaled image, and then the detector is evaluated in a dense fashion over each integral histogram. The bounding boxes produced by the detectors are then processed using NMS. Results for a range of experiments are given in the next section.

## 3.4 Results

The previous three sections have outlined how a person detector can be trained on a labelled dataset, and then run on novel images. In this section, the performance of the developed detector is measured under a variety of different settings against other detectors. The Caltech Pedestrian Detection Benchmark [5] is used to evaluate the performance, and details of this benchmark were discussed in Section 2.1.2. The results graphs plot the miss rate against the number of false positives per image. The Caltech Pedestrian Detection Benchmark can be used to generate results for five different pedestrian datasets. In this section, the INRIA person dataset [51] and the ETH pedestrian dataset [54] are used. It should be noted that the ETH pedestrian dataset contains pedestrians that are smaller than the smallest pedestrian that can be detected by the detectors presented here, and so for all experiments with this data set, the test images are rescaled to twice their original size.

As has been mentioned, the approach for creating a detector outlined here is based on that of the integral channel features detector described in [47]. However, this detector has been completely re-implemented for this thesis, and there are some differences from the implementation from [47]:

- The gradient features used throughout this thesis are extracted from Red-Green-Blue (RGB) images.
- By default, the gradient orientation channels used by the feature maps $\mathbf{\Phi}$ in this thesis are not normalised. The effect of normalising these channels is explored in Section 3.4.4, but the form of normalisation used is different from that used in [47].
- The integral histograms generated by the code written for this thesis do not use interpolation between gradient orientation bins. Removing interpolation reduces the amount of floating point arithmetic that must be executed for each pixel, and allows the binning of edges without computing the inverse tangent function.
- When running a detector at multiple scales, nearest neighbour interpolation is used to rescale images rather than bilinear interpolation. Experiments showed

| Parameter | Default Value |
|---|---|
| Number of Contrast Insensitive Channels | 6 |
| Number of Contrast Sensitive Channels | 0 |
| LUV Channels | Yes |
| Shrink Factor | 4 |
| $U$ (Number of Candidate Features) | 30000 |
| $R$ (Number of Training Rounds) | 2000 |
| $V$ (Number of Thresholds) | 255 |
| Depth of Decision Trees | 2 |
| Box Filter anti-aliasing (for negative examples) | Yes |
| Binomial Filter Length | 3 pixels |

Table 3.1: The default settings used for training a detector.

| Parameter | Default Value |
|---|---|
| Number of Scales per Octave | 8 |
| Stride | 4 pixels |
| Box Filter anti-aliasing | Yes |
| Binomial Filter Length | 3 pixels |

Table 3.2: The default settings used for running a detector on images.

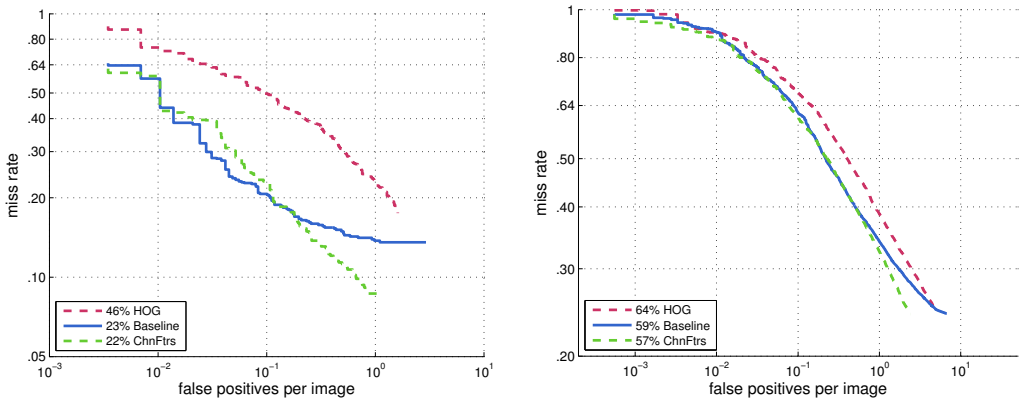
that there was relatively little difference in visual quality for the two methods, but nearest neighbour interpolation is slightly faster.

In addition to the differences described above, the experiments in this section explore other novelties such as the effect of different image filtering settings (Section 3.4.2), altering the orientation bin layout (Section 3.4.3), different normalisation schemes for gradient orientation features (Section 3.4.4) and contrast sensitive features (Section 3.4.5).

### 3.4.1 Default Settings

The first set of results to be presented are generated under a set of default parameters. The effect of varying these parameters will be examined in later sections. The parameters can be divided into those that must be determined at training time, and those that must be determined at test time. The default values are summarised in Tables 3.1 and 3.2.

The results under these settings are shown in Figure 3.8 under the moniker "Baseline", and have been compared against the original integral channel features detector [47], and the HOG detector [4]. It can be seen that the detector has similar performance to the detector from [47], with an average miss rate of 23% instead of 22% on the INRIA dataset, and 59% instead of 57% on the ETH dataset. The results are more consistent




(a) (b)

Figure 3.8: Result for the default settings given in Tables 3.1 and 3.2 for the (a) INRIA person dataset and (b) the ETH pedestrian dataset.

on the ETH dataset, due to the larger number of images. On the INRIA dataset, the baseline detector performs better than the detector from [47] in the region of the graph spanning a false positive rate of $10^{-2}$ to $10^{-1}$, and is worse in the region from $10^{-1}$ to $10^{0}$. It should be noted that the $y$-axis of the graph is logarithmic, and so the performance gap in this region of the graph appears to be larger than it is. The discrepancies between the baseline detector and the detector from [47] can be attributed to the differences that are outlined in Section 3.4. It can be seen that both the baseline detector and the original integral channel features detector outperform the HOG detector by a very large margin.

### 3.4.2 Image Filtering

In this section, the role of image filtering on detector performance is examined. In Section 3.1.1, it was mentioned that images are filtered by a binomial filter prior to the generation of the feature map $\mathbf{\Phi}$. The radius of the binomial filter can be adjusted to modify the amount of smoothing. Also, in Section 3.3.1, it is noted that box filtering is applied prior to scaling images.

Image filtering takes place when running a classifier on an image, and also when cropping negative examples during bootstrapping. To examine the effects of filtering at training time and test time, four classifiers were trained with different filtering parameters, and each was tested on the INRIA dataset with four different sets of filtering

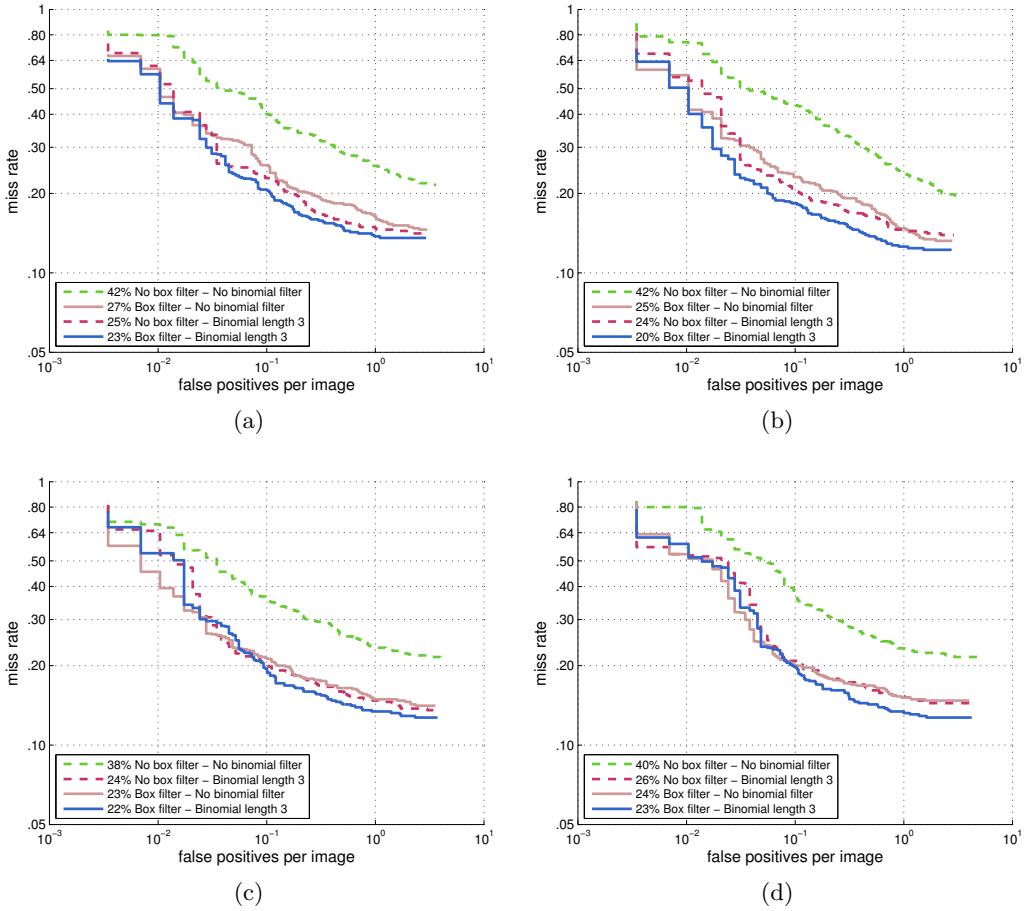
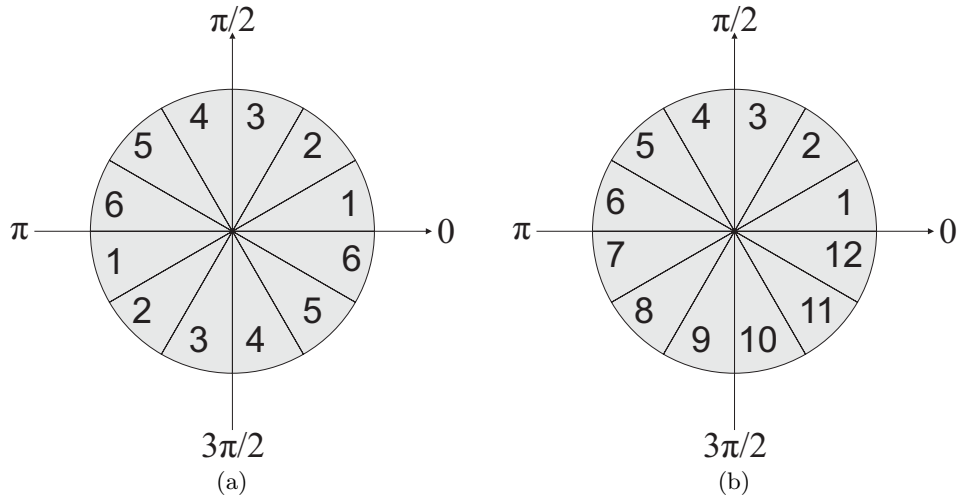



Figure 3.9: The results for four different classifiers trained with different filtering settings, and tested each with four different sets of parameters. The four different training parameter settings are (a) box filtering and binomial filtering of length 3, (b) no box filtering and binomial filtering length 3, (c) box filtering and no binomial filtering and (d) no box filtering and no binomial filtering.

parameters. The four different parameter sets were box filtering and binomial filtering, no box filtering and binomial filtering, box filtering and no binomial filtering, and no box filtering and no binomial filtering. All binomial filtering was done with a binomial filter of length 3, and details are given in Section 3.1.1. The results are shown in Figure 3.9, and a few clear patterns emerge. Results are consistently better when running the classifier with both box filtering and binomial filtering, and consistently much worse when a classifier is run with no form of filtering. Somewhat surprisingly, filtering parameters seem to have much less of an impact at training time, although using no filtering gives worse results in the region of the graph spanning a false positive per image rate of $10^{-2}$ to $10^{-1}$.

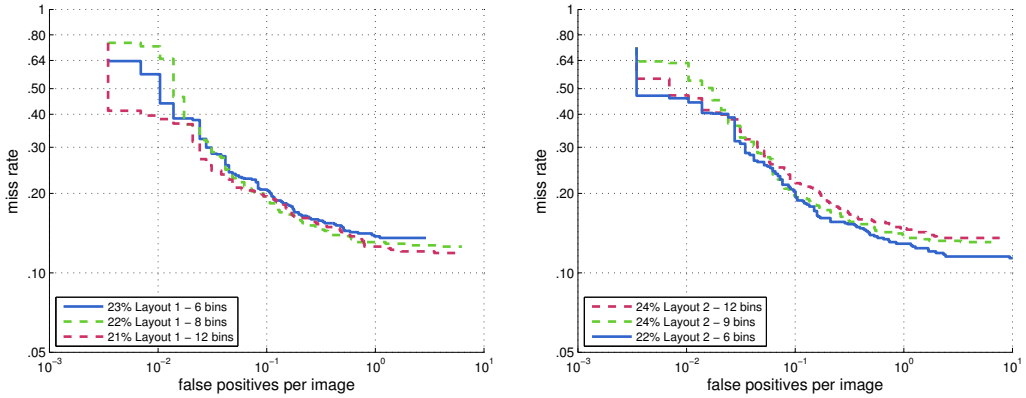


(a) (b)

Figure 3.10: An alternative layout for (a) contrast insensitive orientation bins and (b) contrast sensitive bins.

### 3.4.3 Orientation Bin Layout

The default layout for the gradient orientation bins is shown in Figure 3.1. As can be seen, the bins are arranged in such a way that horizontal and vertical edges fall directly within the extent of a bin. This is slightly different from the default layout that is favoured by HOG features, shown in Figure 3.10, where horizontal edges fall on the boundary between bins. The layouts in Figures 3.1 and 3.10 will be referred to as layouts 1 and 2 respectively. In this section the effect of the number of bins and the bin layout is examined. Figure 3.11a shows the results for layout 1. It should be noted that with this layout, it is not possible to have an odd number of bins. It can be seen that the results are very similar for different numbers of bins, with small increases in performance as the number of bins increases. Figure 3.11b shows the results for layout 2, where once again there is little difference between using different numbers of bins. It can be seen however, that with layout 2, slightly better results are achieved with fewer bins. Overall, layout 1 slightly outperforms layout 2. It is interesting to note that the results do not decline dramatically when the number of bins increase, despite the fact that the number of possible features increases, and therefore a much smaller fraction of the available features is explored.




(a) (b)

Figure 3.11: Results on the INRIA dataset for (a) bin layout 1 and (b) bin layout 2.

### 3.4.4 Normalising Gradient Channels

The default gradient orientation channels used by the developed detector do not use any form of normalisation. Past research has shown that normalising gradient orientation features can have a pronounced effect on detection performance, as normalisation offers a degree of invariance to changes in illumination. HOG features are normalised with respect to four neighbouring regions within an image. In [47], gradient orientation values are normalised with respect to the largest orientation value in a local neighbourhood.

In this section, normalisation is applied to gradient orientation features in a local fashion. This is done by dividing the value of a feature $g$ by the value of the gradient magnitude for the area covered by the feature. This form of normalisation has been used in [36], [97] and [98]. As well as testing the effect of dividing gradient orientation features by the gradient magnitude, the effect of dividing by the gradient magnitude squared is also examined. The results of normalisation are compared against the default detector with no normalisation in Figure 3.12. As can be seen, normalisation by the gradient magnitude results in the same performance as no normalisation, and both have an average miss rate of 23%. Normalising by the gradient magnitude squared slightly degrades the performance. This may be due to the fact that the gradient magnitude squared covers a large range, and dividing by this number maps the feature values to a relatively small range.

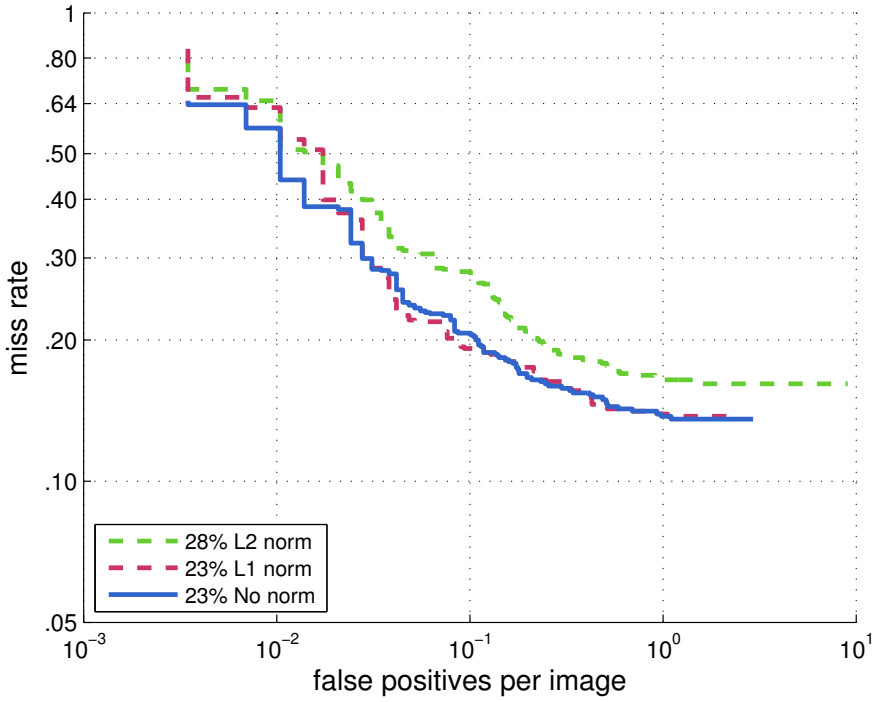



Figure 3.12: The results on the INRIA dataset under different normalisation schemes.

### 3.4.5 Contrast Sensitive Features

As was explained in Section 3.1.2, gradient orientation features can be contrast sensitive or contrast insensitive. Up to this point, only contrast insensitive features have been used in the detectors presented in this thesis. In this section, the effect of using contrast sensitive features is examined. These features are outlined in Section 3.1.2.

Different combinations of features are experimented with in this section. As adding contrast sensitive features vastly increases the total number of possible features, the total number of features considered during training is now doubled from 30000 to 60000. This doubles both the training time and the memory consumption of the training algorithm.

Four classifiers are trained with different groups of features. Those four groups are contrast sensitive features with CIELUV features, contrast insensitive features with CIELUV features, contrast sensitive with insensitive features, and finally, contrast sensitive and insensitive features with CIELUV features.

The results are shown in Figure 3.13. It can be seen that the default feature group of contrast insensitive features with CIELUV features performs best. However, the results also show that it is possible to achieve reasonable performance without CIELUV features by using both contrast sensitive and contrast insensitive features, with an average miss

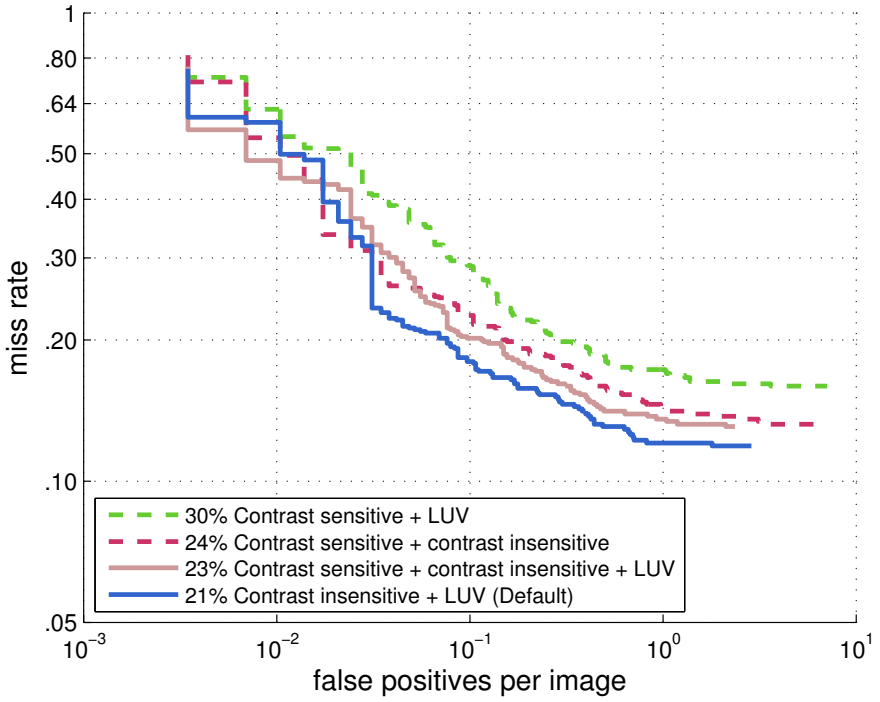



Figure 3.13: The results for different feature sets are shown. Each detector is trained with 60000 features rather than the default of 30000.

rate of 24%. Using all types of features does not give the best performance, which can be explained by the fact that the number of possible features in this situation is very large, and so the 60000 features used for training are a small proportion of the total number of features.

## 3.5 Improving the Speed of Boosting Classifiers

So far, the accuracy of various different classifiers has been measured. Another important factor for a classifier is speed. This section examines the speed of some of the classifiers developed in this section, and examines methods that can be used to improve the speed.

A variety of methods exist for improving the speed of boosting classifiers. Most of these involve constructing a *cascade*, whereby the sequence of evaluations of weak classifiers for a window can be terminated early if some condition is met. This was first introduced in [36], where weak classifiers were grouped into stages determined by a target false positive and detection rate. An alternative method for accelerating detection is presented in [99], where a boosting classifier is trained normally, and then a cumulative rejection threshold is specified for each weak classifier. This concept was expanded upon

in [100]. Crosstalk cascades [49] involve using multiple cascades that can exchange information, and are trained using a validation set. Some methods attempt to achieve gains in speed by taking into account the asymmetry between the number of positive and negative examples that will be encountered by the classifier [101].

### 3.5.1 Constant Rejection Thresholds

Let the cumulative score for a window be $H_{\hat{r}}(\mathbf{I}) = \sum_{r=1}^{\hat{r}} h_r^{tree}(\mathbf{I})$ where $\hat{r} \in \{1, \ldots, R\}$. An extremely simple method to accelerate detection for a boosting classifier without sacrificing accuracy is to terminate the sequence of evaluations when the cumulative score $H_{\hat{r}}(\mathbf{I})$ for an example falls below a fixed threshold $\theta_{reject}$. This idea was first presented in [49] as *constant rejection thresholds*, where an increase in speed of up to a factor of a hundred was obtained for $\theta_{reject} = -1$. It is effective because the majority of negative windows will have negative scores even after only a very small number of weak classifier have been evaluated.

The classifier trained in Section 3.4.1 was run with constant rejection thresholds of different values, and the results are shown in Figure 3.14. The results are somewhat surprising. As expected, less negative values of the threshold degrade the performance slightly. However, as the threshold becomes more negative, the results can actually improve, although not by a significant amount. This may be a sign that the trained classifier is overfitting to the training data. The next section will present results for the speed of various classifiers.

### 3.5.2 Results

In this section, results are given for the speed of various different classifiers, with and without constant rejection thresholds. To measure the time taken to process an image, each classifier was run on the 169 images of the INRIA test set that have a resolution of $640 \times 480$ pixels, and the total time taken was divided by the number of images. It is important to obtain results on a set of images, as the time taken when using constant rejection thresholds is dependent upon the image content. All experiments are carried out on a 32-bit computer with an Intel Pentium Dual-Core 2.6GHz processor and 4GB of RAM. Only a single core of the computer's CPU was used, but the Streaming

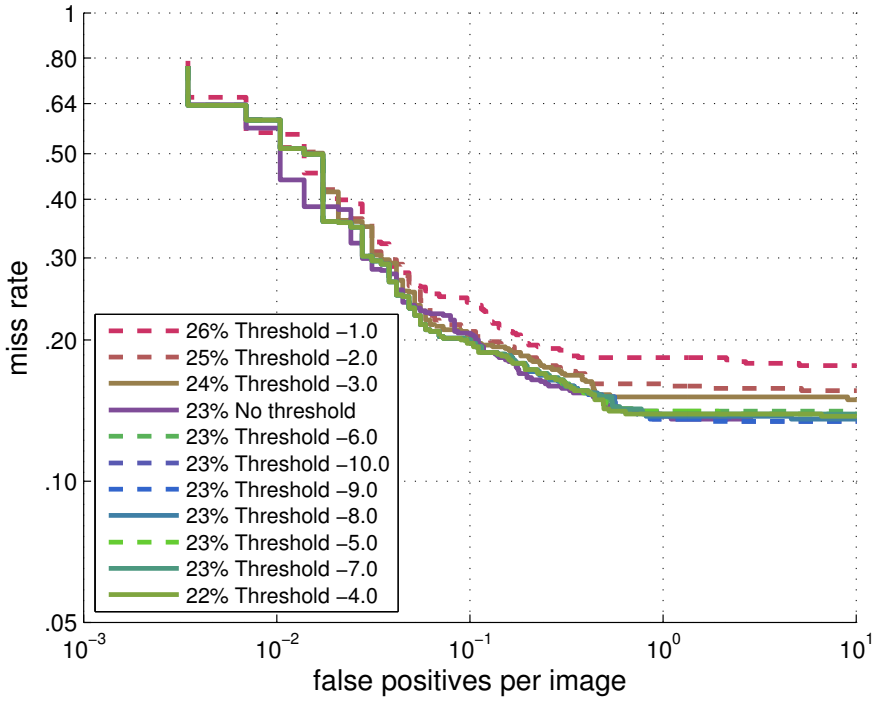



Figure 3.14: The results for the default classifier from Section 3.4.1 run with constant rejection thresholds of different values on the INRIA dataset.

SIMD Extensions (SSE) instruction set [102] was used to improve performance. In these experiments, the input images are not padded.

The speeds quoted cover the process of rescaling the image multiple times (22 times for an input image that is 640 × 480 pixels), smoothing each image with a binomial filter of length 3, generating an integral histogram for each rescaled image, running the classifier on each integral histogram, resizing the bounding boxes obtained from the classifiers, and applying non-maximum suppression to the bounding boxes.

The results are shown in Table 3.3, where the times are given in seconds per image for an image that has a resolution of $640 \times 480$ pixels. It can be seen that with $\theta_{reject} = -5$, there is typically a speed up of more than a factor of ten compared to running the full classifier. Using $\theta_{reject} = -10$ also results in a dramatic speed up. The fastest classifier is the default classifier from Section 3.4.1 with $\theta_{reject} = -5$. The slowest classifier is the classifier that uses magnitude normalisation for gradient orientation features from Section 3.4.4. This is not surprising, as performing this normalisation approximately doubles the amount of computation for each gradient orientation feature. However, this same classifier is the third fastest when used with $\theta_{reject} = -5$.

| | No threshold | $\theta_{reject} = -10$ | $\theta_{reject} = -5$ |
|---|---|---|---|
| Default settings (Section 3.4.1) | 4.0562 | 0.3783 | 0.3235 |
| Layout 1 - 8 bins (Section 3.4.3) | 4.1146 | 0.5540 | 0.3924 |
| Layout 1 - 12 bins (Section 3.4.3) | 4.2426 | 0.5760 | 0.4133 |
| L1-norm (Section 3.4.4) | 6.3750 | 0.4289 | 0.3502 |
| Contrast sensitive + contrast insensitive (Section 3.4.5) | 4.3432 | 0.5728 | 0.3983 |
| Contrast sensitive + contrast insensitive + LUV (Section 3.4.5) | 4.2833 | 0.3991 | 0.3475 |

Table 3.3: A table showing the time taken in seconds per $640 \times 480$ pixel image for various classifiers from this chapter with different constant rejection thresholds.

As expected, increasing the number of gradient orientation bins results in longer times for detection, as it takes longer to generate the integral histograms. This can be seen from the results for the classifiers from Section 3.4.3 in Table 3.3. It can be seen that this difference is relatively minor when no threshold is used. This shows that when a constant rejection threshold is used, the time taken to generate the pyramid of integral histograms becomes a significant bottleneck.

Some of the results for the timing run counter to intuition. The classifier which uses only contrast insensitive and contrast sensitive features from Section 3.4.5 is actually slower than the classifier from Section 3.4.5 that uses those features in addition to CIELUV features. It seems contradictory that using more features would improve the speed. However, the differences in speed are relatively small, and may be due to faster rejection when thresholds are used. These findings emphasise the fact that the speed is dependent not only on how software is written, but on several complex interacting factors related to how the software is compiled, and how memory is accessed on a hardware platform.

In order to gain a deeper insight into how various stages of the detection system impact the overall speed, the code was profiled. The results are shown in Table 3.4, where the stages represent rescaling the input image multiple times, filtering the rescaled images with binomial filters, generating integral histograms for each filtered image, running the classifier on each integral histogram to get bounding boxes, and finally performing non-maximum suppression on these bounding boxes. It should be noted that changing the value of the rejection threshold $\theta_{reject}$ only alters the length of time for the final two stages, and so Table 3.4 shows the timing for the first three stages independent of

| | Rescale all Images | Filter all Images | Generate Integral Histograms | Run Classifier, $\theta_{reject} = -5$ | NMS, $\theta_{reject} = -5$ | Run Classifier, $\theta_{reject} = -10$ | NMS, $\theta_{reject} = -10$ | Run Classifier, no threshold | NMS, no threshold |
|---|---|---|---|---|---|---|---|---|---|
| Default settings | 0.043272 | 0.137201 | 0.096894 | 0.042438 | 0.003695 | 0.099637 | 0.001296 | 3.744455 | 0.034378 |
| Layout 1 - 8 bins | 0.043178 | 0.137023 | 0.102527 | 0.115898 | 0.003675 | 0.263781 | 0.007491 | 3.815834 | 0.016038 |
| Layout 1 - 12 bins | 0.043178 | 0.137023 | 0.112787 | 0.116687 | 0.003625 | 0.281621 | 0.001391 | 3.933229 | 0.016383 |
| L1-norm | 0.043272 | 0.136929 | 0.099574 | 0.06971 | 0.000715 | 0.148663 | 0.000462 | 6.079187 | 0.016038 |
| Contrast sensitive + contrast insensitive | 0.043272 | 0.136651 | 0.098184 | 0.118065 | 0.002128 | 0.291147 | 0.003546 | 4.048165 | 0.016928 |
| Contrast sensitive + contrast insensitive + LUV | 0.043178 | 0.136929 | 0.112792 | 0.049095 | 0.005506 | 0.104980 | 0.001721 | 3.974018 | 0.016383 |

Table 3.4: A table showing the time in seconds for each individual stage of the overall detection algorithm. The first three columns show the time taken for the first three stages of the detector, which are unaffected by the value of $\theta_{reject}$. The last six columns of the table show the timing results for the last two stages of the detector for three different values of $\theta_{reject}$.

the value of $\theta_{reject}$. One of the things that can be observed from Table 3.4 is that the filtering of the images at multiple scales takes a significant amount of the overall time. Therefore, an optimised implementation of the binomial filter could improve the speed results. This could be achieved by packing multiple pixel values, which are one byte per colour channel, into a four byte word to process several values at once. It can also be seen that using rejection thresholds significantly reduces the the time taken to run the classifier.

## 3.6 Discussion and Summary

In this chapter, a detector based on the integral channel features detector was developed. The features and the AdaBoost learning algorithm were described. A practical method for implementing AdaBoost to reduce the computation time and memory consumption was outlined. Issues regarding how to run the classifier at multiple scales and perform non-maximum suppression were addressed. A series of experiments were used to illustrate the effect of novelties, such as altering the bin layout for gradient orientation features, altering the number of bins, implementing various kinds of normalisation for gradient features, and finally, using contrast insensitive features. An important observation was that the CIELUV features can be replaced with contrast insensitive features, with only a small drop in performance, if a classifier that only uses gradient features is

desired.

The results for the default classifier are slightly different from those given in [47]. The performance for the classifier developed in this chapter is better than the original for a false positive per image rate spanning from $10^{-2}$ to $10^{-1}$, but worse from $10^{-1}$ to $10^{0}$. This pattern of performance seems to be repeated for all other classifiers developed throughout the chapter. As was mentioned, this can be attributed to the differences in the implementation for the classifier. The results may indicate overfitting, as the results are worse at lower thresholds, and better at higher thresholds, indicating that positive instances are missed at lower thresholds, but the instances that are detected have much higher scores. The statistical significance of the results could be improved by training each detector multiple times on a different pool of random features, and combining the results together.

The speed of a variety of detectors was also tested, and it was found that constant rejection thresholds could be used to accelerate detection speeds. It should be noted that the results for the classifier speeds in this chapter were carried out on a fairly dated 32-bit machine, without using multiple cores, graphics processing units [50], or fast approximations of the feature map over multiple scales [48]. The top speed reached was just over 3 frames per second at a resolution of $640 \times 480$ pixels.

This chapter is concluded by illustrating some of the results from the detector developed in Section 3.4.5 that uses only contrast sensitive and contrast insensitive features. Figure 3.15 shows 6 of the 288 images from the INRIA test set used for evaluating the performance with the bounding boxes produced by the detector shown in green.

Figure 3.15: Some example results from the detector in Section 3.4.5 that uses gradient features only. (a) A large crowd of people, where most instances of people have been detected. There is a large false positive in the upper left portion of the image. (b) An example with one false negative, for the person that is second from the left. (c) A person on a bicycle is successfully detected, along with a person in the background. There is one false positive in the background. (d) A person riding a bicycle and facing away from the camera is successfully detected. (e) Several fully visible people are detected correctly. (f) Two people standing close together are detected successfully.

# Chapter 4

# Using Multiple Detectors

The previous chapter developed a basic person detector. In this chapter, the idea of using multiple detectors is presented. Multiple detectors can be useful, as each detector can specialise in a particular mode of appearance. Thus, the use of multiple classifiers can yield an increase in accuracy, at the cost of an increase in latency. However, it will be shown in this chapter that the latency does not necessarily have to grow linearly with the number of classifiers. First, it is shown in Section 4.1 that the classifiers developed in Chapter 3 can be run in conjunction with a reflected version of the same classifier in order to improve results. Next, it is demonstrated in Section 4.2.2 that the output from the weak classifiers within a boosting classifier can be used to cluster images of people into different groups based on appearance. In Section 4.2.3, separate classifiers are trained for each of these clusters, and the resulting ensemble of classifiers is applied to images in the same manner as the detector developed in the previous chapter. Results for accuracy and speed are presented, and it is shown that using multiple classifiers is feasible when constant rejection thresholds are used. The chapter is concluded by a discussion and summary.

## 4.1 Using Reflected Classifiers

One noteworthy fact that has not been mentioned so far is that the classifiers developed in Chapter 3 are asymmetric. This is due to the fact that the features used for training are randomly generated, and therefore unlikely to be symmetric, and also the negative training examples are not reflected along the vertical axis, although the positive examples

are.

The asymmetry of boosting classifiers raises an interesting prospect. It should be possible to create a new detector by reflecting an existing one. Reflecting a classifier in the vertical axis can be achieved by reflecting the features $g$ that make up each weak classifier $h$. As was described in Section 3.1.4, each feature is described by a vector of parameters $\boldsymbol{\rho} = [\, x_{\boldsymbol{\rho}}\ y_{\boldsymbol{\rho}}\ w_{\boldsymbol{\rho}}\ h_{\boldsymbol{\rho}}\ c_{\boldsymbol{\rho}}\,]^{\mathsf{T}}$, giving the $x$ and $y$ coordinate of the top left corner, the width and height of the feature, and the channel. Reflecting gradient magnitude features and CIELUV features is straightforward, as only $x_\rho$ needs to be modified. For gradient orientation features, the channel $c_\rho$ must be modified so that the orientation is reflected in the vertical axis as well.

In this section, experiments are carried out to see the effect of using a classifier in conjunction with its vertically reflected counterpart. Let a classifier be denoted by $H(\mathbf{I})$, and its vertically reflected version by $H^{reflect}(\mathbf{I})$. The two classifiers are combined into a joint classifier by taking the maximum score for both classifiers

$$H^{joint}(\mathbf{I}) = \max\left\{H(\mathbf{I}), H^{reflect}(\mathbf{I})\right\}. \tag{4.1}$$

Using two classifiers together in this way could improve the performance, as the extra classifier might detect true positives that were missed by the original classifier. However, it might also degrade the performance, as there will be false positives from two classifiers rather than one. Thus, the results that are observed depend upon which of these two effects dominates. It should be noted that if the false positives from both classifiers are highly correlated, then the second effect will be significantly mitigated, and an increase in performance will be observed. The next section shows the results from this approach using several classifiers.

### 4.1.1 Results

Several of the classifiers developed in Chapter 3 were tested by running them with their reflected counterparts on the INRIA dataset. To accelerate the evaluation, a constant rejection threshold of $\theta_{reject} = -10$ was used.

The results are shown in Figure 4.1. As can be seen, for the six classifiers that were tested, performance improvements were observed in five cases. The only classifier

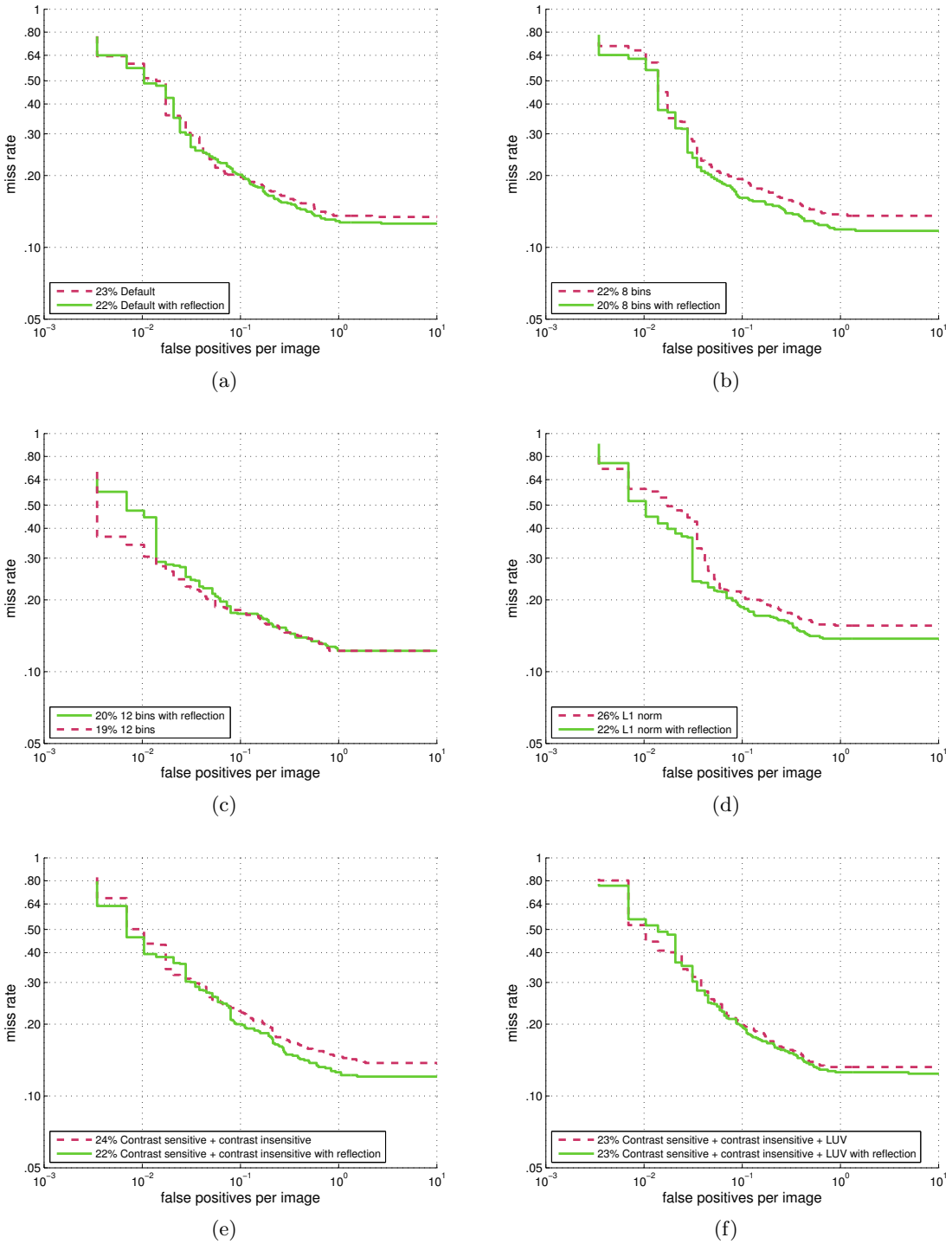



Figure 4.1: The results on the INRIA dataset for several classifiers compared against their "joint" versions described by Equation 4.1 with $\theta_{reject} = -10$. The classifiers are (a) the default classifier from Section 3.4.1, (b) the classifier from Section 3.4.3 with 8 orientation bins, (c) the classifier from Section 3.4.3 with 12 orientation bins, (d) the classifier from Section 3.4.4 with normalised gradient orientation features, (e) the classifier from Section 3.4.5 which uses only contrast sensitive and contrast insensitive features and (f) the classifier from Section 3.4.5 which uses all feature types.

| | No threshold | $\theta_{reject} = -10$ | $\theta_{reject} = -5$ |
|---|---|---|---|
| Default settings (Section 3.4.1) | 8.1793 | 0.4994 | 0.3733 |
| Layout 1 - 8 bins (Section 3.4.3) | 8.3473 | 0.8504 | 0.5210 |
| Layout 1 - 12 bins (Section 3.4.3) | 8.6253 | 0.9103 | 0.5543 |
| L1-norm (Section 3.4.4) | 12.8579 | 0.6049 | 0.4314 |
| Contrast sensitive + contrast insensitive (Section 3.4.5) | 8.8158 | 0.9041 | 0.5333 |
| Contrast sensitive + contrast insensitive + LUV (Section 3.4.5) | 8.6666 | 0.5173 | 0.3984 |

Table 4.1: A table showing the time taken in seconds per $640 \times 480$ pixel image for different classifiers with their reflected counterparts, with different constant rejection thresholds.

where the performance did not improve was the classifier from Section 3.4.3 that used 12 orientation bins, for which the results are shown in Figure 4.1c. It can be seen that the average miss rate degrades by a small amount from 19% to 20%

One of the largest improvements is for the classifier that uses only gradient features from Section 3.4.5 shown in Figure 4.1e, where the average miss rate drops from 24% to 22%. This could indicate that this classifier learns a bias towards a particular pose from the training examples.

### 4.1.2 Classifier Speed

The speed of the joint classifiers was measured. As in Section 3.5.2, all results were computed on images from the INRIA test set that were $640 \times 480$ pixels, and once again a 32-bit computer with an Intel Pentium Dual-Core 2.6GHz processor and 4GB of RAM was used. The results are shown in Table 4.1. As expected, when using the full classifiers, the time taken to evaluate the joint classifier is roughly double the time of an equivalent single classifier, as can be seen by comparing the first column of Tables 3.3 and 4.1. However, when constant rejection thresholds are used, the time taken to evaluate a joint classifier is less than double the amount of time a single classifier would take. This shows that when constant rejection thresholds are used, the time taken to rescale the input image and create integral histograms becomes a significant bottleneck, and so running another classifier immediately after does not affect the speed as much. All of the joint classifiers considered in Table 4.1 take less than a second to evaluate on an image when run with constant rejection thresholds.

### 4.1.3 Summary

It has been shown that a small improvement in performance can be gained by running a classifier with its vertically reflected counterpart. It has also been shown that using these two classifiers together with constant rejection thresholds can lead to a vast increase in speed, making the technique quite practical. Despite the fact that two classifiers are used, the time taken to evaluate can be less than double the time taken for a single classifier.

## 4.2 Using Multiple Classifiers

The previous section examined the effect of using a classifier with its reflected counterpart. The idea of using multiple classifiers is expanded upon in this section by partitioning a set of positive training images into several clusters and training a classifier for each cluster. This allows each classifier to specialise in a particular type of appearance. To partition the set of positive images, $K$-means clustering is used, which is explained in Section 4.2.1. To apply this clustering method to images, vectors that summarise the appearance of an image are extracted using a boosting classifier, and the clusters are visualised in Section 4.2.2. The performance of the multiple classifiers is examined in Section 4.2.5.

### 4.2.1 *K*-means clustering

Clustering is the problem of assigning a set of vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ to $K$ disjoint groups. It is a well studied and common problem, and a variety of solutions exist such as mean-shift [60], Gaussian mixture models [23] and hierarchical clustering [103].

An extremely popular method for clustering data is the $K$-means algorithm [23], which is listed as Algorithm 4. The $K$-means algorithm takes a set of vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ where $\mathbf{x} \in \mathbb{R}^m$ and a specified number of clusters $K$, and returns the cluster centroids $\boldsymbol{\mu} \in \mathbb{R}^m$ and the cluster membership labels for each vector $y_i \in \{1, \ldots, K\}$. The clustering is carried out by alternating between two steps, referred to as the *assignment* step and the *update* step. During the assignment step, vectors are assigned to the cluster with

**Algorithm 4** $K$-means clustering

**Require:**

- A set of vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$.
- The number of clusters $K$.

**for** $k \leftarrow 1$ to $K$ **do**
  Initialise centroid $\boldsymbol{\mu}_k \leftarrow \mathbf{x}_j$ where $j \in \{1, \ldots, n\}$ is random
**end for**
**for** $i \leftarrow 1$ to $n$ **do**
  $y_i \leftarrow \arg\min_k \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$
**end for**
**while** $\{y_1, \ldots, y_n\}$ change from values during the previous iteration **do**
  **for** $k \leftarrow 1$ to $K$ **do**
    $\boldsymbol{\mu}_k \leftarrow \frac{1}{N_k} \sum_{i:y_i=k} \mathbf{x}_i$
  **end for**
  **for** $i \leftarrow 1$ to $n$ **do**
    $y_i \leftarrow \arg\min_k \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$
  **end for**
**end while**
**return** $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K\}$, $\{y_1, \ldots, y_n\}$

the nearest centroid by calculating the label $y_i$ as

$$y_i = \arg\min_k \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2. \tag{4.2}$$

During the update step, the centroid values are updated by taking the average of all the vectors assigned to a cluster, to give new values for $\boldsymbol{\mu}_k$ as

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i:y_i=k} \mathbf{x}_i, \tag{4.3}$$

where $N_k$ is the number of vectors in cluster $k$. Before the algorithm can begin, the values of $\boldsymbol{\mu}$ must be initialised, and this is usually done by randomly selecting a vector from one of the input vectors $\mathbf{x}$ for each centroid. It can be shown that alternating between the assignment and update steps reduces the within cluster sum of squares error (WCSSE) given by

$$\text{WCSSE} = \sum_{k=1}^{K} \sum_{i:y_i=k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2. \tag{4.4}$$

Convergence of the algorithm is guaranteed, and can be detected when an iteration

of the two steps fails to produce a change in the assignment $y_i$ of any of the vectors. However, it is important to note that the algorithm is only guaranteed to converge to a local optimum as Equation 4.4 is not convex in $y_i$ and $\boldsymbol{\mu}_k$, and what this optimum is depends on how the algorithm was initialised. This is a common problem encountered in non-convex optimisation problems, such as expectation maximisation [23] and latent SVMs [44]. A common way to mitigate this problem is to run the $K$-means algorithm multiple times with different initialisations, and to take the solution with the lowest WCSSE value given by Equation 4.4.

It should be noted that the $K$-means algorithm can be modified to use distance measures other than Euclidean distance, such as the $\ell 1$-norm or Hamming distance. It is also important to note that the $K$-means algorithm assumes that the clusters to be found are spherical.

### 4.2.2 Applying Clustering to Image Windows

The $K$-means algorithm could be applied directly to the positive image windows used for training classifiers in Chapter 3 to obtain clusters based on different modes of appearance. However, the dimensionality of each image is equal to the number of pixels, which is 8192. To reduce the complexity of the clustering problem, the dimensionality of the input images could be reduced somehow.

In this section, the idea of clustering windows by using the output from a boosting classifier is explored. As has been shown, boosting classifiers consist of an ensemble of weak classifiers $h_r : \mathbf{I} \rightarrow \{-1, +1\}$, and each weak classifier will make a decision regarding the input, which by itself will often be inaccurate. A final decision on whether the target class is present or not is made by taking the consensus of the weak classifiers, as shown in Equation 3.13. Each weak classifier has an associated coefficient $\alpha_r$, and a real valued score for an input is computed by multiplying each alpha value by the output of the corresponding $h_r$, and summing over $r$. The sign of this value then determines the final decision $f$. The values of $\alpha_r$ are determined at training time as described in Algorithm 1, with larger values of $\alpha_r$ for weak classifiers that are more accurate. It is interesting to note that for a window to be judged as representing a person, only some of the weak classifiers need to return that decision. Thus, for two different images featuring

two different people, the set of weak classifiers that return the label +1 may be different, and may reflect the differences in appearance between the two target instances. For an image window $\mathbf{I}$, the output from a boosting classifier can be used to construct a vector $\boldsymbol{\alpha}(\mathbf{I})$ defined as

$$\boldsymbol{\alpha}(\mathbf{I}) = \begin{bmatrix} \alpha_1 h_1(\mathbf{I}) & \cdots & \alpha_R h_R(\mathbf{I}) \end{bmatrix}^{\mathsf{T}}, \tag{4.5}$$

which summarises the response of the weak classifiers. This vector gives a compact summary of multiple feature values. This concept is not dissimilar to the popular "bag of words" paradigm [104] in computer vision, where histograms of feature frequencies are used as a higher level feature. Here, the vector of responses from the individual weak classifiers is used.

If the vector $\boldsymbol{\alpha}$ can be used to summarise variations in appearance between different instances of people, then it is reasonable to suggest that instances with a similar appearance would have values of $\boldsymbol{\alpha}$ that were close together, as measured by a distance metric, such as the Euclidean norm. Thus, if values of $\boldsymbol{\alpha}$ were computed for a set of images containing people, these vectors could be clustered to reveal modes of appearance among those people. Using the vectors $\boldsymbol{\alpha}$ presents several advantages over clustering the actual images. The most obvious is that the vector $\boldsymbol{\alpha}$ is considerably more compact than an image. Another advantage is that as $\boldsymbol{\alpha}$ is produced by a boosting classifier, the entries of the vector focus on discriminative information. Finally, by training different boosting classifiers with different sets of features, the nature of the clustering can be altered. By using a boosting classifier trained solely on colour features, $\boldsymbol{\alpha}$ will only reflect information from these features.

In this section, positive training images are clustered by extracting the vectors $\boldsymbol{\alpha}$ and applying $K$-means clustering. Let $\boldsymbol{\alpha}_H(\mathbf{I})$ denote a vector $\boldsymbol{\alpha}$ extracted from an image $\mathbf{I}$ by applying the boosting classifier $H$. By changing the boosting classifier $H$, different values of $\boldsymbol{\alpha}_H(\mathbf{I})$ can be obtained. For example, using a boosting classifier trained only on CIELUV features will create a vector $\boldsymbol{\alpha}$ that only reflects information from the CIELUV colour space. A set of positive images $\{\mathbf{I}_1, \ldots, \mathbf{I}_n\}$ is used with a boosting classifier $H$ to generate a set of vectors $\{\boldsymbol{\alpha}_H(\mathbf{I}_1), \ldots, \boldsymbol{\alpha}_H(\mathbf{I}_n)\}$ which are used as the input to the $K$-means algorithm.

Four different classifiers are used to generate different versions of $\boldsymbol{\alpha}_H(\mathbf{I})$ to generate

different results for clustering. To make the discussion in the following section easier to follow, each of these classifiers is assigned a moniker, and is described next:

- The first classifier is referred to as `default trees`, and is trained using the default settings described in Section 3.4.1.
- The second classifier is referred to as `default stumps`, and is the same as `default trees`, except for the fact that stump classifiers are used rather than depth two decision trees. The motivation behind using stump classifiers is that tree classifiers select features to test using branching, and so two inputs that are given the same label by a decision tree can have different visual characteristics. A stump classifier tests only a single feature, and so any two inputs that are assigned the same label will have responded to a single feature in the same way.
- The third classifier is referred to as `cieluv trees`, and is similar to `default trees`, except that it is trained only with CIELUV features. The motivation for using only CIELUV features is to obtain clusters based only on information from this colour space.
- The fourth classifier is referred to as `cieluv stumps`, and is similar to `cieluv trees`, but uses stump classifiers rather than depth two decision trees.

Experiments are also carried out by applying the $K$-means algorithm with the Hamming distance rather than Euclidean norm. The Hamming distance is a metric that measures the distance between two binary strings by counting the number of bits that differ. To convert $\boldsymbol{\alpha}$ to a binary vector, positive entries become 1 and negative entries become 0. Thus, clustering with the Hamming distance can be used to test if the actual values of $\alpha_r$ have an impact on clustering, and whether it is possible to use an even more compact representation.

As was mentioned in Section 4.2.1, the $K$-means algorithm is sensitive to initialisation, and is usually run multiple times, with the result with the lowest within cluster sum of squares error being taken. In this section, all clustering experiments involve running the $K$-means algorithm 20 times. To visualise what each cluster might represent, the images within a cluster can be averaged together.


Figure 4.2: The average image for different clusters. (a) The average of all the positive training images used. (b) and (c) show the averages for the clusters obtained with the `default trees` classifier using the Euclidean distance, and (d) and (e) show the same results when using the Hamming distance. (f) and (g) show the averages for the clusters obtained with the `cieluv trees` classifier using the Euclidean distance, and (h) and (i) show the same results when using the Hamming distance. (j) and (k) show the cluster averages obtained with the `default stumps` classifier using Euclidean distance and (l) and (m) show the cluster averages when using the Hamming distance. (n) and (o) show the cluster averages obtained with the `cieluv stumps` classifier using the Euclidean distance and (p) and (q) show the same results when using the Hamming distance.

Figure 4.2 shows the results of clustering with $K = 2$ for different classifiers using Euclidean and Hamming distances. The clustering was applied to the 2416 images of the INRIA person training set. Figure 4.2a shows the average of these 2416 images. As can be seen, there is little difference in the clusters obtained using different classifiers or different distance metrics. The two clusters that are obtained correspond to lighter and darker images, with the cluster of darker images containing approximately 60% of all the images on average.

Figure 4.3 shows the results of clustering for $K = 3$. It can be seen that small differences begin to emerge between the clusters for different classifiers. The clusters produced with classifiers that use CIELUV features have average appearances that place more emphasis on colour, with Figures 4.3n, 4.3q, 4.3t, and 4.3w being noticeably more blue in colour than Figures 4.3b, 4.3e, 4.3h, and 4.3k. Also, Figures 4.3m, 4.3p, 4.3s, and 4.3v are slightly more red in colour than Figures 4.3a, 4.3d, 4.3g and 4.3j. The results are very similar regardless of whether the Euclidean distance or the Hamming distance is used for clustering. Once again, the majority of images belong to the cluster

with the darker average appearance.

Figure 4.4 shows the results of clustering for $K = 5$. It can be seen that with this number of clusters, the results for different classifiers are now significantly different, and the distance metric used also affects the results. Some correspondences can be seen across the results, with all sets of clusters having a "dark" cluster, a "light" cluster, and a cluster representing images with people wearing clothes of a blue hue against a light background. However, many clusters are now unique to certain classifier and distance metric combinations. Figure 4.4a shows the cluster averages for the `default trees` classifier using the Euclidean distance. It can be seen that the center cluster seems to represent images of people against green backgrounds, such as natural environments with grass. Similar clusters appear when the Hamming distance is used, as shown in Figure 4.4b, although the central cluster is not as well defined. When the `default stumps` classifier is used, as shown in Figures 4.4c and 4.4d, the clusters are quite similar to those created using the `default trees` classifier, except that the cluster representing people against natural backgrounds is replaced by a less well defined cluster. When classifiers with CIELUV features are used, the average appearances of the clusters place a stronger emphasis on colour. It can be seen from Figures 4.4e, 4.4f, 4.4g and 4.4h that shades of blue and red are more prominent than in Figures 4.4a, 4.4b, 4.4c and 4.4d. It can also be seen that there is a cluster representing images of people against backgrounds with a brown hue. With CIELUV classifiers, the results are similar regardless of whether the Euclidean distance or the Hamming distance is used for clustering.

In this section, it has been shown that the output from a boosting classifier can be used to construct vectors $\boldsymbol{\alpha}_H(\mathbf{I})$ that can be used to cluster images of people into different groups based on the visual characteristics tested for by different features.

### 4.2.3 Training Multiple Classifiers

The previous section has illustrated how the output from a boosting classifier can be used to cluster images of people, where each cluster represents a distinct mode of appearance. In this section, the idea of training individual classifiers for each of these clusters is presented. This allows each individual classifier to specialise in a particular mode of appearance. For example, if the positive training images are divided into $K$ clusters then


Figure 4.3: The average image for different clusters when $K = 3$. (a), (b) and (c) show the cluster averages obtained using the `default trees` classifier using the Euclidean distance, and (d), (e) and (f) show the same results using the Hamming distance. (g), (h) and (i) show the cluster averages obtained using the `default stumps` classifier and the Euclidean distance, and (j), (k) and (l) show the same results using the Hamming distance. (m), (n) and (o) show the cluster averages obtained using the `cieluv trees` classifier and the Euclidean distance, and (p), (q) and (r) show the same results using the Hamming distance. (s), (t) and (u) show the cluster averages obtained using the `cieluv stumps` classifier and the Euclidean distance, and (v), (w) and (x) show the same results using the Hamming distance.

Figure 4.4: The average image for different clusters when $K = 5$. (a) shows the cluster averages obtained using the `default trees` classifier using the Euclidean distance, and (b) shows the same results using the Hamming distance. (c) shows the cluster averages obtained using the `default stumps` classifier using the Euclidean distance, and (d) shows the same results using the Hamming distance. (e) shows the cluster averages obtained using the `cieluv trees` classifier using the Euclidean distance, and (f) shows the same results using the Hamming distance. Finally, (g) shows the cluster averages obtained using the `cieluv stumps` classifier using the Euclidean distance, and (h) shows the same results using the Hamming distance.

$K$ separate classifiers can be trained. The classifiers can then be combined together in a manner similar to that used in Section 4.1. For a set of classifiers $\{H_1(\mathbf{I}), \ldots, H_K(\mathbf{I})\}$, the final score is simply the maximum amongst all the classifiers

$$H^{joint}(\mathbf{I}) = \max\{H_1(\mathbf{I}), \ldots, H_K(\mathbf{I})\}. \tag{4.6}$$

The concept of using multiple boosting classifiers has been introduced in other work. A method developed in the "AnyBoost" [94] framework is given in [105], where multiple classifiers are trained simultaneously. This method has been demonstrated for the task of pedestrian detection [25]. In [50], different classifiers are trained to specialise for particular scales. In the approach presented here, multiple classifiers can be trained independently once clustering has been carried out, and the output from a monolithic boosting classifier is used to guide the clustering of the positive training examples.

### 4.2.4 Calibrating Classifiers

When using multiple classifiers, an important issue is that the real valued scores output by different detectors are not always directly comparable. For example, two classifiers that output real valued scores for images could have completely different ranges for their output values. Thus, when using multiple classifiers in conjunction, it is necessary to adjust their output scores in a process that is often referred to as *calibration*. Calibration did not need to be addressed in Section 4.1, as the reflected counterpart of a classifier will have the same output range. Though sophisticated calibration methods exist [106], in this thesis a simpler method is used. For a set of classifiers $\{H_1(\mathbf{I}), \ldots, H_K(\mathbf{I})\}$, let $\alpha_r^k$ be a weak classifier coefficient for the classifier $k$. Classifiers are calibrated by normalising all values of $\alpha_r^k$ so that $\sum_{r=1}^{R} \alpha_r$ for all values of $k$ is equal to $\max_k\{\sum_{r=1}^{R} \alpha_r^k\}$.

### 4.2.5 Results

In this section, results are presented for using multiple detectors. Six different sets of multiple detectors are presented. Three were trained on clustered images produced by the `default trees` classifier, using values of $K = 2$, $K = 3$, and $K = 5$, and another three were trained on images clustered by the `cieluv trees`, using values of $K = 2$,

$K = 3$, and $K = 5$. All classifiers were trained using the default parameters defined in Section 3.4.1. Tests were carried out using calibration as described in Section 4.2.4, and without calibration. To accelerate the evaluation, as in Section 4.1.1, a constant rejection threshold of $\theta_{reject} = -10$ is used.

The result are shown in Figure 4.5, and as can be seen, are rather mixed. There is no consistent pattern for the effect of calibration, and it seems to only cause small differences in performance. It can be seen that training more classifiers using more clusters seems to degrade the average miss rate, with Figures 4.5e and 4.5f showing that classifiers with $K = 5$ on sets that were clustered using the `default trees` and `cieluv trees` classifiers have average miss rates of 26% and 27% respectively. The best results are obtained by training classifiers on images from clusters created using the `default trees` classifier for $K = 2$, where the resulting classifier has an average miss rate of 19% as shown in Figure 4.5b.

The results obtained in Figure 4.5 seem to indicate that overfitting occurs when more clusters are used. To attempt to alleviate this problem, the classifiers trained via clustering were run again, but this time the default classifier from Section 3.4.1 was added to each set of classifiers, so that now each set of classifiers consists of those trained on clusters, and the default classifier. The results are shown in Figure 4.6, and it can be seen that running the classifiers trained on clusters along with the default classifier generates results that improve upon on the original results when $K = 3$ and $K = 5$. However, the best performance is still obtained with the classifier trained on clusters obtained using the `default trees` classifier with $K = 2$.

Figure 4.7 shows some of the results obtained using the set of classifiers trained on images that were clustered with the `default trees` classifier with $K = 5$. The same images that were used for Figure 3.15 are used again. The majority of detections shown are the result of just one of the detectors, for which the bounding boxes are shown in dark blue, but several of the detections correspond to the other classifiers, shown in red, yellow, green and cyan.

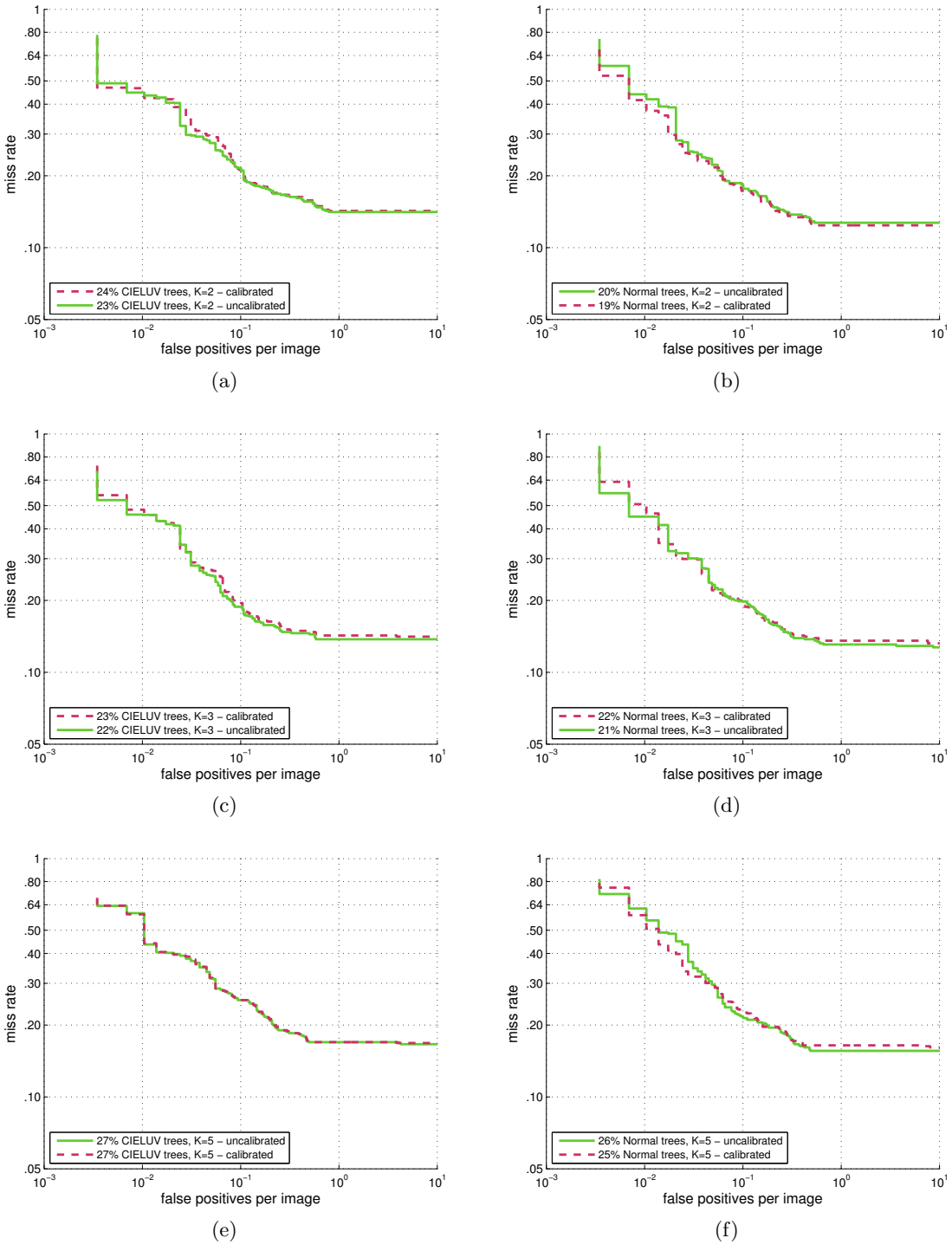



Figure 4.5: The results on the INRIA dataset for several classifiers trained on clustered datasets, with and without calibration, and with $\theta_{reject} = -10$. The classifiers are sets of classifiers trained on clusters created using the `cieluv trees` classifier with (a) $K = 2$, (c) $K = 3$ (e) and $K = 5$, and sets of classifiers trained on clusters created using the `default trees` classifier with (b) $K = 2$, (d) $K = 3$ (f) and $K = 5$.

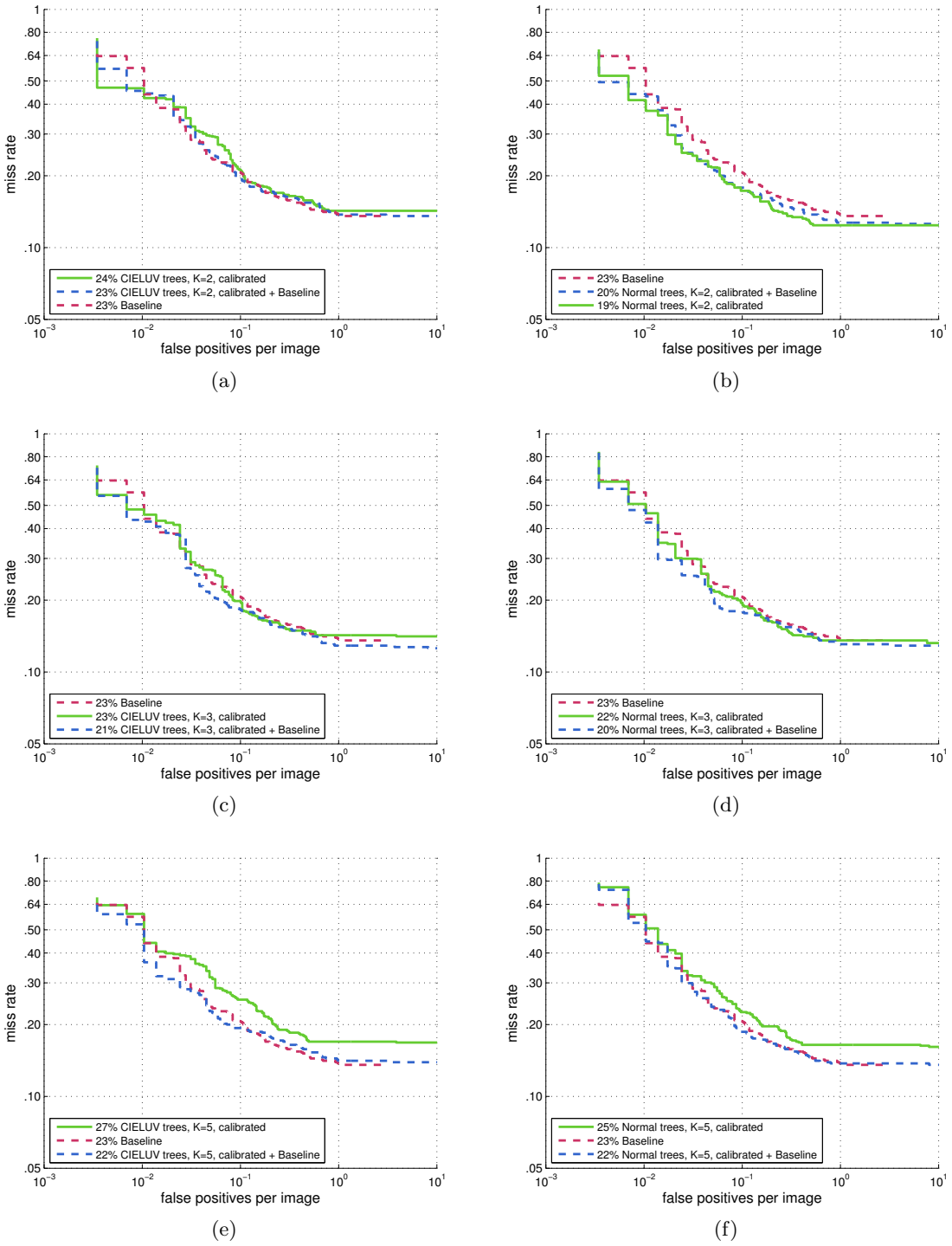



Figure 4.6: The results on the INRIA dataset for the calibrated classifiers from Figure 4.5 combined with the default classifier from Section 3.4.1. Each figure shows the results for the default classifier (under the moniker "Baseline"), one of the classifiers trained on clustered data, and the combination of the two.

Figure 4.7: Examples of the results from the set of detectors trained on positive images clustered using the `default trees` classifier with $K = 5$. The bounding boxes are colour coded to show which classifier they correspond to, with the correspondence shown in (a), (b), (c), (d) and (e).

| Training Set | No threshold | $\theta_{reject} = -10$ | $\theta_{reject} = -5$ |
|---|---|---|---|
| `default trees`, $K = 2$ | 8.1117 | 0.5686 | 0.4043 |
| `cieluv trees`, $K = 2$ | 8.1241 | 0.6128 | 0.4163 |
| `default trees`, $K = 3$ | 12.0094 | 0.6170 | 0.4318 |
| `cieluv trees`, $K = 3$ | 12.0534 | 0.6845 | 0.4487 |
| `default trees`, $K = 5$ | 19.8175 | 0.5493 | 0.4131 |
| `cieluv trees`, $K = 5$ | 20.0119 | 0.8409 | 0.5305 |

Table 4.2: A table showing the time taken in seconds per $640 \times 480$ pixel image for different sets of multiple classifiers, with different constant rejection thresholds.

### 4.2.6 Classifier Speed

The speed of the multiple classifiers was measured. As with Sections 3.5.2 and 4.1.2, all results were computed on images from the INRIA test set that were $640 \times 480$ pixels, and once again, a 32-bit computer with an Intel Pentium Dual-Core 2.6GHz processor and 4GB of RAM. The results are shown in Table 4.2. Once again, when the full classifier is run, as shown in the second column of Table 4.2, the time taken per image scales in a linear fashion. As a single classifier takes roughly 4 seconds, with $K = 2$ the set of classifiers takes roughly 8 seconds, with $K = 3$ the time is around 12 seconds, and with $K = 5$ the time is 20 seconds. However, it can be observed again that when a constant rejection threshold is used such that $\theta_{reject} = -10$ or $\theta_{reject} = -5$, the time taken is always less than one second, and scales much more gracefully as $K$ increases.

## 4.3 Discussion and Summary

It has been shown in this chapter that multiple classifiers can be used to improve the results of person detection. It was first shown how reflected classifiers could be utilised to improve performance, without sacrificing too much speed. The speed of these joint classifiers could be further improved by exploiting the fact that any features that are symmetric with respect to the vertical axis only need to be computed once. Another improvement would be to attempt to alter the training process so the orientation of positive training images with respect to the vertical axis is a latent random variable, which can be inferred during training to create a classifier that is biased towards a particular pose. Such an approach has been applied to SVMs to yield better results [44].

It was shown that the output from the weak classifiers of a boosting classifier reflect

information on the appearance of input images, and can be used to construct vectors for clustering. It was also shown that these vectors can be converted to a compact binary form with little impact on the results of clustering. Future work could explore using more clusters to obtain even more distinct modes of appearance.

Finally, it was shown that multiple classifiers can be trained on sets of positive images obtained via clustering, and that combining multiple classifiers together can improve performance, if these clustered classifiers are combined with a normally trained classifier. With constant rejection thresholds, multiple classifiers can be run in conjunction with only a small increase in running time. Future work could focus on modeling any redundancy between classifiers to improve the speed further

# Chapter 5

# Tracking People in Video

The previous chapters have addressed the problem of detecting people in images rapidly. This chapter focuses on the problem of tracking people in video. The first approach that is presented is based on the mean shift algorithm that was described in detail in Section 2.2.1. It is shown that this algorithm can be adapted to the purpose of tracking based on colour characteristics. A novel method for mean shift tracking through scale is developed and tested. Finally, the problem of tracking multiple people is briefly examined based on a different approach involving the classifiers that were developed in Chapter 3 and Kalman filtering.

## 5.1 Mean Shift Tracking

Mean shift tracking [63] involves tracking a target based on its colour characterisitics. The target's position must be manually initialised in the first frame, and in subsequent frames the best colour match for the target is found.

In this section, the theory behind the mean shift tracking technique from [63] is outlined. The target that is to be tracked is represented by a reference colour histogram in RGB space $\hat{\mathbf{q}} = \begin{bmatrix} \hat{q}_1 & \cdots & \hat{q}_{m_b} \end{bmatrix}^\mathsf{T}$ where $m_b$ is the number of bins for the colour histogram. The aim is to find the area centered on the pixel location $\mathbf{y} \in \mathbb{R}^2$ in each frame $\mathbf{I}_t$ of a video $\{\mathbf{I}_1, \ldots, \mathbf{I}_T\}$ which yields a colour histogram $\hat{\mathbf{p}}(\mathbf{I}_t, \mathbf{y})$ that is most similar to $\hat{\mathbf{q}}$. The similarity between two colour histograms is measured using the Bhattacharyya

coefficient, which is given by

$$\mathrm{BC}(\mathbf{y}) = \mathrm{BC}(\hat{\mathbf{p}}(\mathbf{I}_t, \mathbf{y}), \hat{\mathbf{q}}) = \sum_{i=1}^{m_b} \sqrt{\hat{p}_i(\mathbf{I}, \mathbf{y})\hat{q}_i}. \tag{5.1}$$

Since $\sum_{i=1}^{m_b} \hat{p}_i = 1$ and $\sum_{i=1}^{m_b} \hat{q}_i = 1$, the range of the Bhattacharyya coefficient is $0 \leq \mathrm{BC} \leq 1$. The value of the Bhattacharyya coefficient is high when two histograms are very similar, and low when they are dissimilar. Thus, the tracking problem can be formulated as finding the value of $\mathbf{y}$ in each frame that maximises $\mathrm{BC}(\mathbf{y})$. The histograms $\hat{\mathbf{q}}$ and $\hat{\mathbf{p}}(\mathbf{y})$ are kernel weighted histograms, and the reason for this will be addressed shortly. A kernel weighted histogram modulates the value that a pixel contributes to the histogram based on the position of the pixel relative to a kernel, and so $\hat{p}_i \in \mathbb{R}$ can be expressed as

$$\hat{p}_i(\mathbf{I}, \mathbf{y}) = c_{\mathbf{H}} \sum_{\mathbf{x} \in X} k\left(\|\mathbf{H}(\mathbf{y} - \mathbf{x})\|^2\right) \delta\left(b_{\mathrm{RGB}}(\mathbf{I}(\mathbf{x})) - i\right), \tag{5.2}$$

where $c_{\mathbf{H}}$ is a normalisation constant, $k(\cdot)$ is a finite support kernel function which was explained in detail in Section 2.2.1, and $\mathbf{x} \in \mathbb{R}^2$ are the pixel co-ordinates within the support $X$ of the kernel function $k(\cdot)$. It should be noted that in this chapter, $\mathbf{I}(\mathbf{x}) \in \mathbb{R}^3$ will be used to denote a pixel, in contrast to the notation that was used in Chapter 3. Furthermore, $\delta$ is the Kronecker delta function, $b_{\mathrm{RGB}} : \mathbb{R}^3 \rightarrow \{1, \ldots, m_b\}$ is a function that maps the pixel located at $\mathbf{x}$ to the correct histogram bin index based on its colour, and $\mathbf{H}$ is the bandwidth matrix given by

$$\mathbf{H} = \begin{bmatrix} \frac{1}{h_1} & 0 \\ 0 & \frac{1}{h_2} \end{bmatrix}, \tag{5.3}$$

where $h_1$ and $h_2$ are parameters that determine the scale of the kernel, and also enable the kernel to be elliptical, as opposed to just circular.

It is shown in [63] that the linear Taylor series expansion of the Bhattacharyya coefficient, subject to the condition that we consider histograms that are kernel weighted as was previously mentioned, is comprised of two terms. The first term is constant with respect to $\mathbf{y}$, and the the second term takes the form of a kernel density estimate. As the mean shift algorithm can be used to find the local mode of a kernel density estimate,

it can be used as the basis of a tracking algorithm that maximises the Bhattacharyya coefficient. This is achieved by computing the location $\mathbf{y}$ iteratively using the following equation

$$\mathbf{y}_{j+1} = \frac{\sum_{\mathbf{x}\in X} \mathbf{x} w(\mathbf{I},\mathbf{x}) k'\left(\left\|\mathbf{H}(\mathbf{y}_j - \mathbf{x})\right\|^2\right)}{\sum_{\mathbf{x}\in X} w(\mathbf{I},\mathbf{x}) k'\left(\left\|\mathbf{H}(\mathbf{y}_j - \mathbf{x})\right\|^2\right)}, \tag{5.4}$$

where $\mathbf{y}_{j+1}$ is the new location, $\mathbf{y}_j$ is the previous location, $k'(\cdot)$ is the derivative of $k(\cdot)$, and the weights $w(\mathbf{I},\mathbf{x})$ are given by

$$w(\mathbf{I},\mathbf{x}) = \sum_{i=1}^{m_b} \sqrt{\frac{\hat{q}_i}{\hat{p}_i(\mathbf{I},\hat{\mathbf{y}}_j)}}\delta(b_{\mathrm{RGB}}(\mathbf{I}(\mathbf{x})) - i). \tag{5.5}$$

It should be noted that for any value of $i$ where, $\hat{p}_i = 0$, the result of the division operation in Equation 5.5 is set to 0. Typically, Equation (5.4) will have to be iterated several times to converge to an estimate for the target position in a single frame. Iterations are continued until the mean shift vector $\mathbf{m}_g(\mathbf{y}_j)$ given by Equation 2.8, which is the difference between two consecutive location estimates, has a magnitude below some specified threshold. In this thesis, this threshold is set to one pixel. It is important to note that the histogram $\hat{\mathbf{p}}(\mathbf{I},\mathbf{y}_j)$ will change at every mean shift iteration, as $\mathbf{y}_j$ is changing.

A choice of kernel for $k(\cdot)$ must be made. The most common choice is the Epanechnikov kernel given by Equation 2.10 in Section 2.2.1. By choosing $k(\cdot)$ to be the Epanechikov kernel, the derivative kernel $k'(\cdot)$ becomes the uniform kernel. This considerably simplifies the calculation of (2.12), as all values of $k'(\cdot)$ (within the support of the kernel) are constant. Thus, Equation 5.4 simplifies to

$$\mathbf{y}_{j+1} = \frac{\sum_{\mathbf{x}\in X} \mathbf{x} w(\mathbf{I},\mathbf{x})}{\sum_{\mathbf{x}\in X} w(\mathbf{I},\mathbf{x})} \tag{5.6}$$

### 5.1.1 Mean Shift Tracking Through Scale

As was very briefly mentioned in Section 2.2.1, mean shift tracking through scale is a non-trivial problem. The most extensive and in depth work on this problem is [65], where a method was developed based on Lindeberg's theory of scale selection [66]. This method is briefly explained next.

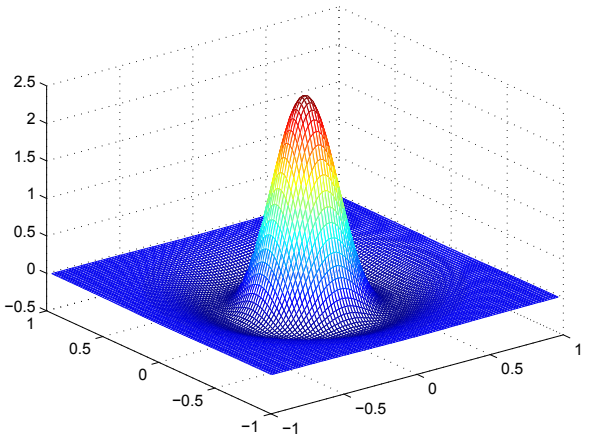



Figure 5.1: The difference of Gaussians kernel.

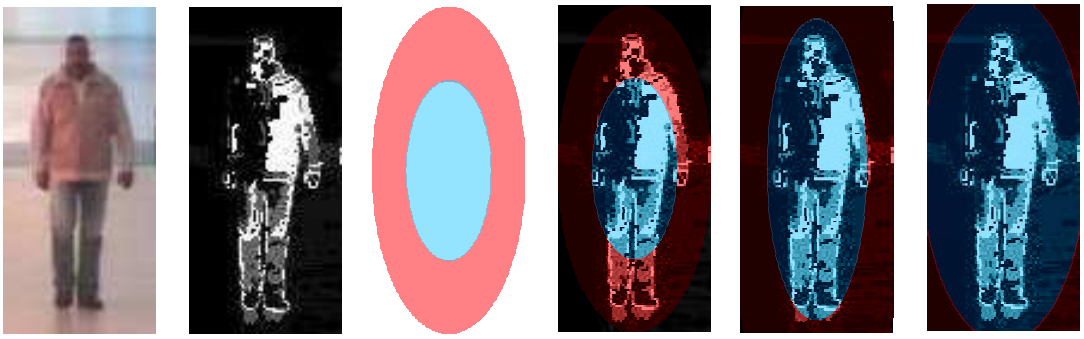
The technique presented in [65] works by interleaving mean shift iterations in the spatial domain (similar to those given by Equation 2.12) with iterations in the scale domain. The current scale is denoted as $\sigma_0$. The difference of Gaussians kernel is used to perform mean shift in the scale domain, and is given by

$$k_D(x,s) = \frac{1}{2\pi\sigma_s^2/1.6}\exp\left(\frac{-x}{2\sigma_s^2/1.6}\right) - \frac{1}{2\pi\sigma_s^2(1.6)}\exp\left(\frac{-x}{2\sigma_s^2(1.6)}\right), \tag{5.7}$$

where $s \in \mathbb{Z}$ is a variable used to control the scale of the kernel through the equation $\sigma_s = \sigma_0\zeta^s$ where $\zeta$ is a user specified parameter (a value of 1.1 is used in [65]), and $\exp(\cdot)$ is the exponential function. The kernel is shown in Figure 5.1. Equation 5.7 is an approximation of the Laplacian of Gaussian function, and is similar in profile to the Mexican hat wavelet [107]. It can be seen that $k_D(x,0)$ will give a kernel at the current scale. The aim is to calculate the new scale $\sigma_0^{new}$. This is given by $\sigma_0^{new} = \sigma_0\zeta^{s_{new}}$, where $s_{new}$ is given by the mean shift equation

$$s_{new} = \frac{\sum_{s=-n}^{n} s \sum_{\mathbf{x}\in X} w(\mathbf{I},\mathbf{x}) k_D\left(\left\|\mathbf{H}(\mathbf{y}_j - \mathbf{x})\right\|^2, s\right)}{\sum_{s=-n}^{n} \sum_{\mathbf{x}\in X} w(\mathbf{I},\mathbf{x}) k_D\left(\left\|\mathbf{H}(\mathbf{y}_j - \mathbf{x})\right\|^2, s\right)}, \tag{5.8}$$

where typically $n = 2$. A more intuitive explanation of this algorithm is given in Figure 5.2. The function $w(\mathbf{I},\mathbf{x})$ simply maps pixels to their weights, implicitly creating a new type of image, as shown in Figure 5.2b. This image was created by replacing each pixel



(a) (b) (c) (d) (e) (f)

Figure 5.2: Using the difference of Gaussians method with mean shift for scale detection: (a) The original image in RGB space. (b) $w(\mathbf{I}, \mathbf{x})$, the back-projection image. (c) A diagrammatic representation of the difference of Gaussians kernel. The red region represents negative values, and the blue represents positive values. (d) A kernel that is too small has been superimposed on $w(\mathbf{I}, \mathbf{x})$. Many values fall in the red region, leading to a smaller value of $\tilde{w}(\mathbf{I}, \mathbf{y}_j, s)$ (e) A kernel that is the correct size. The vast majority of the target is in the blue region, creating a large value for $\tilde{w}(\mathbf{I}, \mathbf{y}_j, s)$ (f) A kernel that is too large. Though the target is in the blue region, the peak is shallower, and so $\tilde{w}(\mathbf{I}, \mathbf{y}_j, s)$ will be smaller.

by the weights given in Equation (5.5), and normalising all values to lie in the range 0 to 255. In this new image, the pixel values will be replaced with the weights given by Equation 5.5, which tends to allocate larger values to pixels that are likely to belong to the target. The target in this image is then masked with difference of Gaussian kernels of different scales. The weighted average of these scales is then taken. The second summation in the numerator of Equation (5.8) evaluates the summation of the kernel multiplied by the weights, and may be written separately as

$$\tilde{w}(\mathbf{I}, \mathbf{y}_j, s) = \sum_{\mathbf{x} \in X} w(\mathbf{I}, \mathbf{x}) k_D \left( \left\| \mathbf{H}(\mathbf{y}_j - \mathbf{x}) \right\|^2, s \right). \tag{5.9}$$

Figures 5.2d, 5.2e and 5.2f show how the differently scaled kernels are used to calculate the appropriate scale of the target. Each kernel consists of a positive central region surrounded by a ring of negative values, as is represented by Figure 5.2c, where the colours red and blue represent negative and positive values respectively. A kernel whose positive region fits tightly to the target as in Figure 5.2e will give the maximum possible value for $\tilde{w}(\mathbf{I}, \mathbf{y}_j, s)$. For a kernel that is too small, as shown in Figure 5.2d, the negative region will overlap with the target, leading to a lower value for $\tilde{w}(\mathbf{I}, \mathbf{y}_j, s)$. For a kernel that is too large, as shown in Figure 5.2f, most of the positive values will overlap with

values of $w(\mathbf{I}, \mathbf{x})$ that are zero. A wider difference of Gaussians kernel has a shallower peak (due to normalisation of the integral), and so the values that do overlap with the non-zero values of $w(\mathbf{I}, \mathbf{x})$ will be smaller. Therefore, $\tilde{w}(\mathbf{I}, \mathbf{y}_j, s)$ is smaller for kernels that are both too big or too small. Thus, $s_{new}$ is calculated as a weighted average of the different scales, weighted by $\tilde{w}(\mathbf{I}, \mathbf{y}_j, s)$. The full mathematical justification for this technique is given in [65].

There are several issues that arise when using the scheme presented in [65]. The method requires the computation of several difference of Gaussian kernels per mean shift iteration per frame. This can rarely be achieved in real-time, even with the efficiency savings from highly optimised algorithms. Also, several parameters must be chosen, such as the number of scales to be considered. Finally, the difference of Gaussian kernels that are used have infinite support, and must therefore be truncated at some point.

A novel method for scale adaptive mean shift tracking is now presented. Rather than using intermediary variables to adjust the scale, it is much simpler to deal directly with the variables $h_1$ and $h_2$ from the bandwidth matrix given in Equation 5.3. A subscript $j$ is now added to these quantities to denote that they are computed via iterations. Assuming that the target to be tracked maintains the same ratio between its width and height, then both $h_{1,j}$ and $h_{2,j}$ can be updated in the same fashion to give the new values $h_{1,j+1}$ and $h_{2,j+1}$

$$h_{1,j+1} = h_{1,j}(1 + \lambda), \tag{5.10}$$

$$h_{2,j+1} = h_{2,j}(1 + \lambda), \tag{5.11}$$

where $\lambda$ is an update factor. Thus, a negative value for $\lambda$ will yield a decrease in target scale, and a positive value will result in an increase in scale. Now a method is needed to generate a value for $\lambda$. First, a new kernel is defined

$$k_R(x) = \begin{cases} \sqrt{x} - 0.5 & x \leq 1 \\ 0 & \text{otherwise} \end{cases}, \tag{5.12}$$

A cross-section through the kernel $k_R(\|\mathbf{H}(\mathbf{y}_j - \mathbf{x})\|^2)$ is shown in Figure 5.3. The values of the kernel range from $-0.5$ to 0.5, with the central region containing negative values,

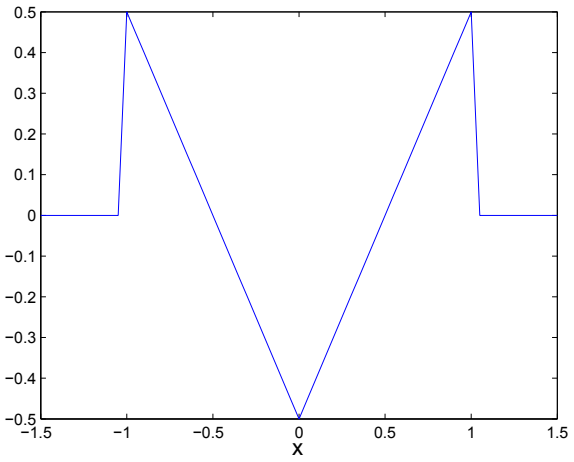



Figure 5.3: A cross-section through the kernel $k_R(\|\mathbf{x}\|^2)$. This kernel is used to perform mean shift style iterations in scale.

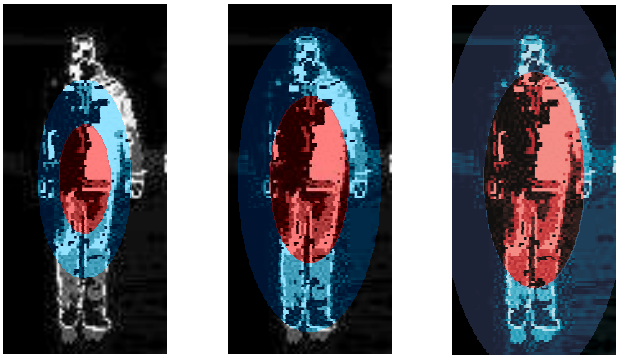
and the outer region consisting of positive values. It is interesting to note that this is in a sense the opposite of the difference of Gaussians kernel, where a central region of positive values is surrounded by a ring of negative values. Also, this kernel has a linear profile, in contrast to the exponential profile for the difference of Gaussians kernel. This scale kernel can be used to calculate the value of $\lambda$ using the following type of mean shift style iteration

$$\lambda = \frac{\sum_{\mathbf{x}\in X} w(\mathbf{I},\mathbf{x}) k_R\left(\|\mathbf{H}(\mathbf{y}_j - \mathbf{x})\|^2\right)}{\sum_{\mathbf{x}\in X} w(\mathbf{I},\mathbf{x})}. \tag{5.13}$$

The iterations given by Equation 5.13 are interleaved with mean shift iterations in the spatial domain given by Equation 5.6.

Note that in Equation 5.13, only a single kernel is used, in contrast to the multiple kernels used in Equation 5.8. As with the difference of Gaussians method presented in [65], an intuition can be developed as to why this method works. Figure 5.4 shows how the value of $\lambda$ will vary with targets of different scale. A target that is too small will produce a negative value for $\lambda$, as it will be concentrated in the central region of the kernel, as shown in Figure 5.4a. A target that is too large will produce a positive value of $\lambda$, as shown in Figure 5.4c. $\lambda$ converges to a low value when the scale is correct, as shown in Figure 5.4b.

The technique developed for scale adaptivity in this thesis has advantages over several



(a) $\lambda > 0$ (b) $\lambda \approx 0$ (c) $\lambda < 0$

Figure 5.4: The proposed method for tracking through scale. As with Figure 5.2, the red regions are negative, and the blue positive, but this time we consider the kernel given by Equation 5.12. (a) Most of the values fall in the positive region, resulting in $\lambda > 0$ (b) The non-zero values of $w(\mathbf{I}, \mathbf{x})$ are roughly equally distributed between the positive and negative regions of the kernel, resulting in $\lambda \approx 0$ (c) The majority of non-zero values of $w(\mathbf{I}, \mathbf{x})$ fall in the negative region of the kernel, leading to $\lambda < 0$.

others. The computational cost is roughly double that of the mean shift algorithm without scale adaptivity, while the method for scale adaptivity developed in [63] is triple the cost. This is due to a reduction in the number of kernels that must be computed per iteration of mean shift. The method presented here represents a considerable reduction in computational complexity from the method presented in [65], which requires that several (typically five) difference of Gaussian kernels are computed per mean shift iteration. In comparison, the method shown here requires the computation of two kernels per mean shift iteration, and the most intensive calculations (the norm of a set of vectors) are common to both kernels, and therefore need only be computed once per iteration.

Finally, the kernel method developed in this paper is less prone to background distractions than the difference of Gaussians method, as the kernel developed here fits the target more tightly. This can be seen by comparing Figures 5.2e and 5.4b, as the correctly scaled difference of Gaussians kernel has a larger support then our scale kernel. Therefore, anything very close to the target that exhibits a similar appearance will affect the difference of Gaussians method, but will have little effect on our technique.

### 5.1.2 Results

The proposed technique for tracking through scale was tested on the CAVIAR dataset [108]. The results on two videos are shown in Figure 5.5. The reference colour histogram

(a)

(b)

Figure 5.5: Examples of tracking through scale. (a) A person is tracked as they walk towards a camera. The kernel grows larger as they get closer. (b) A person in a group walks towards the camera. Note that there are inaccuracies in the first two frames shown due to background distractions. As the target walks into a clearing, the scale estimate improves.

| | Proposed mean-shift method | Boosting classifier |
|---|---|---|
| Percentage of false negatives | 28.17 | 12.83 |

Table 5.1: The percentage of missed detections on a 600 frame sequence from the CAVIAR dataset using the proposed mean-shift method and the boosting classifier from Chapter 3.

$\hat{\mathbf{q}}$ is created by manually initialising the tracker's position in the first frame. It can be seen that the technique is successful, although there are small inaccuracies due to background distraction in the first two frames of the third row.

Table 5.1 shows the percentage of missed detections from the proposed mean-shift tracking algorithm on a 600 frame sequence from the CAVIAR dataset. The results are measured for tracking a single person. The results for the mean-shift algorithm are compared against the boosting classifier from Chapter 3. For clarity, detections are measured here in the same way as Chapters 3 and 4 using an overlap criterion. This criterion is equal to the area of the intersection between a predicted bounding box and the ground truth bounding box divided by the area of the union of the two boxes. If the result is greater than 0.5, the detection is regarded as a true positive. Bounding boxes are obtained from mean-shift kernels by simply taking the smallest rectangle that encapsulates the elliptical mean shift kernel. A missed detection signifies a low overlap between the predicted bounding box and the ground truth bounding box.

Timing results were also obtained on the 600 frame CAVIAR sequence for the proposed mean-shift tracking algorithm and the boosting classifier from Chapter 3. The resolution of these images is $384 \times 288$ pixels. The time per frame for the proposed mean-shift tracking algorithm is 0.03852 seconds, while for the boosting classifier it is 0.13108 seconds. Thus, the proposed mean-shift algorithm is significantly faster than the boosting classifier.

## 5.2 Tracking Multiple Targets

The method presented in the previous section is suitable for tracking single targets, and requires manual initialisation. In many scenarios, it is desirable to be able to track multiple people automatically. Tracking multiple targets is a difficult problem, especially

if the identities of the bounding boxes across time are to be established.

In contrast to using mean shift tracking to track individual targets, the classifiers developed in Chapters 3 and 4 can be used. These classifiers generate bounding boxes for each image, as was described at length in Chapter 3. However, no temporal information is used, and as a result, the correspondences between bounding boxes across time are not known. The process of finding these correspondences is referred to as *data association*, and this is a necessary prerequisite before any of the recursive Bayesian filters described in Section 2.2.2 can be used to filter the measurements. A simple heuristic that can be used to perform data association is to define a distance metric between measurements, and associate measurements between frames that have the shortest distance measure. In Equation 2.14 a measurement for the single target tracking problem was defined. For the multiple target tracking problem, let a set of measurements for a frame at time $t$ be $\{\mathbf{z}_t^1, \ldots, \mathbf{z}_t^{n_t}\}$ where $n_t$ is the number of measurements at time $t$, and a set of measurements at time $t+1$ be $\{\mathbf{z}_t^1, \ldots, \mathbf{z}_t^{n_{t+1}}\}$. The aim is to create a set of *tracks* that consist of disjoint sets of measurements across time. The distance between two measurements in two consecutive frames can be defined as

$$d(\mathbf{z}_t^l, \mathbf{z}_{t+1}^m) = \sqrt{\mathbf{z}_t^{l\mathsf{T}} \mathbf{z}_{t+1}^m}. \tag{5.14}$$

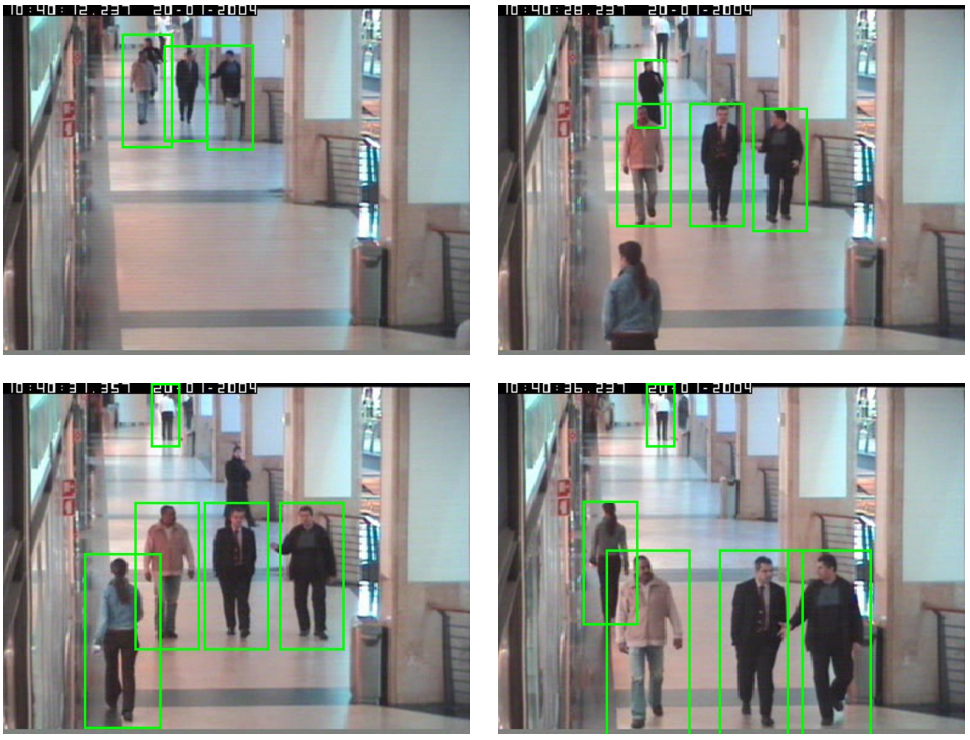
A matrix $\mathbf{D} \in \mathbb{R}^{n_t \times n_{t+1}}$ of the distances between all measurements in two consecutive frames can be constructed. Each row of this matrix gives the distances between a measurement at time $t$ and every other measurement at time $t+1$. Thus, for each row the lowest value of $d$ defines the best match for a measurement at time $t$. To prevent matches when all values in a row are very large, a maximum matching distance $d_{max}$ is defined. The best match for a measurement is made if the value is less than $d_{max}$. When a match is made, the row and column in the matrix that correspond to the match are removed, so that the same measurement cannot be assigned twice, as in [109]. This process is repeated until the matrix is empty. If no match is found for a measurement, then its track is terminated.

Once data association has been performed as described, the measurements that have been matched together into tracks can be filtered using Kalman filters, described in detail in Section 2.2.2. The purpose of this filtering is to smooth any abrupt changes

in a sequence of measurements. A measurement vector for this filtering problem may be defined as $\mathbf{z}_t \in \mathbb{R}^3$, where the three entries of the vector are the target's $x$-position, $y$-position, and scale. The state vector can be defined as $\mathbf{x}_t \in \mathbb{R}^6$, where the entries are the target's $x$ position, $y$ position, scale, $x$ velocity, $y$ velocity, and "scale velocity", which indicates the scale change with respect to time. The state transition model $\mathbf{F}$, defined in Section 2.2.2 by Equation 2.19, can be chosen to assume constant velocity, with Gaussian noise used to account for acceleration. The only remaining parameters to be determined are the process noise covariance matrix $\mathbf{Q}$, and the observation noise covariance matrix $\mathbf{R}$. These parameters have a major impact on how the Kalman filter performs, and this issue is explained next.

The state estimate given by a Kalman filter is a combination of the measurement $\mathbf{z}_t$ and the predicted state $\hat{\mathbf{x}}_{t|t-1}$, weighted by the process noise and observation noise covariance matrices $\mathbf{Q}$ and $\mathbf{R}$, as shown in Equation 2.23. If the observation noise has much less power than the process noise, then the estimate is weighted towards the measurement. Conversely, if the observation noise is greater, then the estimate is weighted towards the predicted state. Intuitively, the level of noise can be thought of as the level of uncertainty, with the process noise representing uncertainty for the predicted state $\hat{\mathbf{x}}_{t|t-1}$ and the observation noise representing uncertainty for the measurement $\mathbf{z}_t$, and so the Kalman estimate is weighted towards the more certain quantity. For these reasons, experiments in this section use diagonal covariance matrices for $\mathbf{Q}$ and $\mathbf{R}$ with both matrices equal to strike a balance between the smoothing introduced by the process model and the results given by the actual measurements.

Figure 5.6 shows some results for applying Kalman filters to the measurements produced by a boosting classifier. Results were also computed using a particle filter, but were found to be very similar. As can be seen, the detector is capable of detecting most instances of people. The previously described data association method was used to group the measurements into tracks, and Kalman filters were used to filter the measurements for each track. It should be noted that the number of tracks created was much larger than the number of individuals in the video sequence, and so for each individual person there is a series of "broken" tracks. These track fragments are a common occurrence, and are often referred to as *tracklets* [110]. It would be desirable to fuse the tracklets



(a)

(b)

Figure 5.6: Results of running the classifier from Chapter 3 with Kalman filtering on the same images from the CAVIAR dataset that were used in Figure 5.5. (a) The detector is able to detect most of the people present in the images. (b) The detector is again able to most instances, but there is a false positive in the final frame.

together, in order to have tracks that correspond to each person in a video sequence, but this is left for future work. It should be noted that the mean shift algorithm from Section 5.1.1 is considerably faster than the boosting classifier, for which timing results were given in Section 3.5.2.

## 5.3 Discussion and Summary

This chapter has examined the problem of tracking people in videos. The mean shift algorithm was used for single target tracking with initialisation, and a novel, efficient method was presented for tracking through scale. Results for this method were presented on the CAVIAR dataset. As this form of tracking is based solely on colour, it is prone to errors when there are areas of a frame that are of a similar colour to the target. Therefore, future work could focus on using other features for tracking, such as image gradient orientations.

The final section of this chapter briefly examined the problem of tracking multiple people. As was mentioned, an aim for future research would be to create tracks that correspond to the identities of unique instances of people. This could be achieved by fusing the tracklets that were obtained, perhaps by deriving appearance models for each target. A number of other issues would also have to be addressed, such as tracking through occlusion, reducing the number of false positives and being able to interpolate measurements to correct false negatives.

# Chapter 6

# Conclusions and Future Work

This chapter concludes this thesis by summarising the work that has been presented, and exploring directions for future research. The problem of detecting people in images was examined in Chapters 3 and 4. The techniques presented used the AdaBoost algorithm in conjunction with image gradient and CIELUV features. The problem of tracking people in video was addressed in Chapter 5. Mean shift and recursive Bayesian filters were used for tracking. Concluding remarks are made in Section 6.1, and ideas for future work are given in Section 6.2.

## 6.1 Summary and Concluding Remarks

The problems of detecting and tracking people in images and video were examined in this thesis. There are vast variety of methods and techniques for tackling these problems, and this thesis has focused on the use of AdaBoost for detection and mean shift and recursive Bayesian filters for tracking. The focus of this thesis has been on techniques that can evaluate a $640 \times 480$ pixel image in less than a second.

The third chapter of this thesis presented a series of experiments concerning the detection of people in images. The integral channel features detector was used as a starting point, and the different aspects of this detector were discussed in detail. The process of extracting gradient orientation and magnitude features, along with features in the CIELUV colour space was explained. Efficient methods for computing these features were illustrated. The AdaBoost algorithm was examined in detail, and practical issues concerning the implementation of the algorithm were explored. Experiments investigated

the effects of novelties such as altering the layout of gradient orientation bins, using normalisation for gradient orientation features, and using contrast sensitive gradient orientation features. Results were presented to show that the proposed detectors were both fast and accurate. It was also found that with a better implementation of the binomial filter, the speed could be improved.

The fourth chapter examined the use of multiple classifiers, and also showed how boosting classifiers could be used to aid the process of clustering images of people. It was shown that the accuracy of a boosting classifier could be improved by combining it with its vertically reflected version. It was also shown that the output of weak classifiers from a boosting classifier contain information regarding the appearance of input images, and so by applying $K$-means clustering to vectors constructed from these outputs, it is possible to cluster images into sets based on their appearance. Furthermore, these vectors could be converted to a binary form without much loss of information, to give an even more compact summary of an image's appearance. The idea of training multiple classifiers on clustered data sets was also presented. It was shown that multiple classifiers could be combined together to yield performance that was better than that of the classifiers used separately. It was also demonstrated that using multiple classifiers with constant rejection thresholds prevents the running time from becoming too prohibitive.

The fifth chapter of this thesis examined the problem of tracking people in video. The mean shift algorithm was used for single target tracking by finding the best match for a target in terms of RGB colour features. A novel method was presented for mean shift tracking through scale, which involved interleaving iterations in scale space with those in the image plane. It was shown that this method possessed several advantages over competing methods, particularly in regards to the number of kernels used. The problem of tracking multiple people was also very briefly explored using the classifiers developed in previous chapters. These were used in conjunction with Kalman filters that were used to smooth the results for tracks. Measurements were grouped into tracks by a simple data association method based on a distance metric being computed exhaustively for measurements from consecutive frames.

## 6.2 Future Work

There are a number of possible future directions for the research presented in this thesis. Object detection and tracking are both very active areas of research, and are widely applicable to a range of scenarios.

One possible direction of research that would make the techniques presented in this work easier to implement would be aiming to reduce the memory resources and time taken by the AdaBoost training algorithm. This would make it easier to experiment with other feature sets and to tune parameters for training. Techniques such SVMs consume far fewer resources during training. Progress could be made by making the training process approximate rather than exact. In fact, the training process used throughout this thesis is already approximate in the sense that only a fraction of the possible features are considered. Selecting a small representative training set without the use of bootstrapping could greatly reduce the training time. Faster training would allow the multiple classifier technique in Chapter 4 to be extended to more clusters.

The multiple classifier technique from Chapter 4 could be extended to work with a larger number of clusters. With more clusters, it would be expected that the appearance of each individual cluster would become more unique, and this would make each classifier trained on a cluster more specialised. Classifiers could then be used to infer useful information regarding appearance, such as the colour of clothing, or the type of environment. A scheme for sharing weak classifiers between different boosting classifiers would allow savings in computation.

In this thesis, the problem of detecting people in images was explored. An important direction for future research would be to extend the developed detectors to work on other object categories, or even to simply extend the detector to detecting people in more varied scenarios involving occlusion and articulation. This would likely involve training multiple classifiers for each pose of an object class. Training detectors for multiple object categories introduces new possibilities. Contextual cues can be used to aid detection, by incorporating information on the spatial distribution of different object classes in relation to one another. For example, people tend to appear level with vehicles, and both appear below the sky. Also, features could be shared between object categories to reduce the computation at test time. Such an idea has already been presented [111], but has not

been extensively tested on newer, more difficult object detection benchmarks. Grammar models [46] represent an interesting field of research, where detectors are composed of parts, and a set of rules can be used to generate multiple detectors. At present though, the grammar must be manually designed, and results have only been presented for a single object category.

In Section 2.1, it was explained that the majority of object detection methods are comprised of a feature transformation and a machine learning algorithm. This has been the case with all the detectors presented in this thesis. A major issue with this approach is the difficulty in selecting a suitable feature transformation for a particular machine learning algorithm. Too often, this process effectively involves trial and error, leading to the criticism that research in object detection is more "art" than science. A major goal for object detection research that would also have implications in other research fields would be to develop a technique that would dispense with or automate the feature transformation selection process. Progress has recently been made in this direction by using neural networks for object detection with raw pixel intensities as the input values [29], and have achieved state of the art results. However, neural networks are slow to train and current implementations are slower to run than other alternatives, and so techniques developed with other machine learning algorithms will continue to remain relevant.

With tracking, future work could incorporate three dimensional scene information. This would be particularly useful to handle occlusion in a principled manner. Also, it would enable a better approach towards filtering, as motion models and measurements could be mapped to real world co-ordinates. This would require the use of a calibrated camera [112]. Camera calibration is an extensive field of research itself, but an interesting approach would be to guide the calibration by tracking people. As the average height of a person is known, this information can be leveraged to provide a rough initial calibration for a camera, which can be refined thereafter.

Effective tracking could also be used to extract features based on motion. These features, which would contain temporal information unlike the features used throughout this thesis, could capture motion that is characteristic of pedestrians. However, such features would not improve detection performance on static images.

The problem of tracking multiple targets was very briefly examined in this thesis. This is an extremely challenging problem which is the subject of ongoing research. The main difficulty lies in grouping measurements into tracks that correspond to unique targets across time. Approaches that could be explored in future that would be compatible with the boosting classifiers developed in this thesis would include methods that formulate multiple target tracking as a network flow problem [113], and methods that perform data association through Markov chain Monte Carlo methods [114].

The results presented in this thesis have used the INRIA person dataset, the ETH pedestrian dataset and the CAVIAR dataset. When only a single dataset is used, it is possible that any improvements in performance indicates overfitting rather than better generalisation performance. Better conclusions regarding the performance of techniques can be drawn from testing them on multiple datasets. Such tests can be time consuming, but allow for deeper forms of analysis, such as exploring the effect of bias in datasets [115]. Thus, future work could incorporate more varied and difficult datasets, which would also be helpful to determine which future directions for research would result in the largest improvements in performance.

# Bibliography


[1] J. Losty and P. Watkins, "Computer vision for industrial applications," *Software Microsystems*, vol. 2, no. 5, pp. 130–138, 1983.

[2] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3642–3649, 2012.

[3] G. Thomas, "Sports TV applications of computer vision," in *Visual Analysis of Humans* (T. B. Moeslund, A. Hilton, V. Krger, and L. Sigal, eds.), pp. 563–579, Springer London, 2011.

[4] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. 886–893, 2005.

[5] Caltech Pedestrian Detection Benchmark. `http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/`.

[6] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: An evaluation of the state of the art," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, pp. 743–761, Apr. 2012.

[7] R. Layne, T. M. Hospedales, and S. Gong, "Towards person identification and re-identification with attributes," in *Computer Vision ECCV 2012. Workshops and Demonstrations* (A. Fusiello, V. Murino, and R. Cucchiara, eds.), vol. 7583 of *Lecture Notes in Computer Science*, pp. 402–412, Springer Berlin Heidelberg, 2012.

[8] Z. Liu and S. Sarkar, "Improved gait recognition by gait dynamics normalization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 6, pp. 863–876, 2006.

[9] J. Candamo, M. Shreve, D. Goldgof, D. Sapper, and R. Kasturi, "Understanding transit scenes: A survey on human behavior-recognition algorithms," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 1, pp. 206–224, 2010.

[10] W. Knight, "Volvo demos a nifty cyclist detection system." MIT Technology Review, Mar. 2013.

[11] C. Urmson and W. Whittaker, "Self-driving cars and the urban challenge," *IEEE Intelligent Systems*, vol. 23, no. 2, pp. 66–68, 2008.

[12] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, and S. Thrun, "Junior: The Stanford entry in the urban challenge," *Journal of Field Robotics*, vol. 25, pp. 569–597, Sept. 2008.

[13] E. Guizzo, "How Google's self-driving car works," *IEEE Spectrum Online*, vol. 18, Oct. 2011.

[14] J. Lee, "Hacking the Nintendo Wii remote," *IEEE Pervasive Computing*, vol. 7, no. 3, pp. 39–45, 2008.

[15] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, p. 3, 2011.

[16] M. Sun, H. Su, S. Savarese, and L. Fei-Fei, "A multi-view probabilistic model for 3D object classes," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1247–1254, 2009.

[17] M. J. Choi, A. Torralba, and A. Willsky, "A tree-based context model for object recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 2, pp. 240–252, 2012.

[18] D. Gavrila and V. Philomin, "Real-time object detection for "smart" vehicles," in *IEEE International Conference on Computer Vision (ICCV)*, vol. 1, pp. 87–93, 1999.

[19] S. Dupont and J. Luettin, "Audio-visual speech modeling for continuous speech recognition," *IEEE Transactions on Multimedia*, vol. 2, no. 3, pp. 141–151, 2000.

[20] D. Ferrucci, "Build Watson: an overview of DeepQA for the Jeopardy! challenge," in *19th ACM international conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 1–2, 2010.

[21] J. Bennett and S. Lanning, "The Netflix prize," in *Proceedings of KDD cup and workshop*, p. 35, 2007.

[22] M. J. Kearns and U. V. Vazirani, *An introduction to computational learning theory.* Cambridge, MA, USA: MIT Press, 1994.

[23] C. M. Bishop, *Pattern Recognition and Machine Learning.* Information Science and Statistics, Springer, 1st ed., 2007.

[24] K. Tieu and P. Viola, "Boosting image retrieval," in *International Journal of Computer Vision*, pp. 228–235, 2000.

[25] S. Walk, N. Majer, K. Schindler, and B. Schiele, "New features and insights for pedestrian detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1030–1037, 2010.

[26] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov. 2004.

[27] C. M. Bishop, *Neural Networks for Pattern Recognition.* New York, NY, USA: Oxford University Press, 1st ed., 1995.

[28] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, pp. 1527–1554, July 2006.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25 (NIPS)*, pp. 1097–1105, 2012.

[30] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[31] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[32] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun, "Support vector machine learning for interdependent and structured output spaces," in *Proceedings of the 21st International Conference on Machine learning (ICML)*, (New York, NY, USA), p. 104, ACM, 2004.

[33] Y. Yang and D. Ramanan, "Articulated pose estimation with flexible mixtures-of-parts," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1385–1392, 2011.

[34] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, Oct. 2001.

[35] J. Gall, A. Yao, N. Razavi, L. Van Gool, and V. Lempitsky, "Hough forests for object detection, tracking, and action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 11, pp. 2188–2202, 2011.

[36] P. Viola and M. Jones, "Robust real-time object detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2002.

[37] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *The Annals of Statistics*, vol. 38, no. 2, 2000.

[38] C. Huang, H. Al, B. Wu, and S. Lao, "Boosting nested cascade detector for multi-view face detection," in *17th International Conference on Pattern Recognition (ICPR)*, vol. 2, pp. 415–418, 2004.

[39] I. Laptev, "Improvements of object detection using boosted histograms," in *British Machine Vision Conference (BMVC)*, vol. 6, pp. 949–958, 2006.

[40] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[41] C. Papageorgiou and T. Poggio, "Trainable pedestrian detection," in *International Conference on Image Processing (ICIP)*, vol. 4, pp. 35–39, 1999.

[42] Q. Zhu, M.-C. Yeh, K.-T. Cheng, and S. Avidan, "Fast human detection using a cascade of histograms of oriented gradients," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 1491–1498, 2006.

[43] X. Wang, T. Han, and S. Yan, "An HOG-LBP human detector with partial occlusion handling," in *12th IEEE International Conference on Computer Vision (ICCV)*, pp. 32–39, 2009.

[44] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[45] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes (VOC) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, June 2010.

[46] R. B. Girshick, P. Felzenszwalb, and D. Mcallester, "Object detection with grammar models," in *Advances in Neural Information Processing Systems 24 (NIPS)*, pp. 442–450, 2011.

[47] P. Dollár, Z. Tu, P. Perona, and S. Belongie, "Integral channel features," in *British Machine Vision Conference (BMVC)*, vol. 2, p. 5, 2009.

[48] P. Dollár, S. Belongie, and P. Perona, "The fastest pedestrian detector in the west," in *British Machine Vision Conference (BMVC)*, vol. 2, 2010.

[49] P. Dollár, R. Appel, and W. Kienzle, "Crosstalk cascades for frame-rate pedestrian detection," in *Proceedings of the 12th European Conference on Computer Vision (ECCV) - Volume Part II*, pp. 645–659, Springer Berlin Heidelberg, 2012.

[50] R. Benenson, M. Mathias, R. Timofte, and L. Van Gool, "Pedestrian detection at 100 frames per second," in *CVPR*, pp. 2903–2910, June 2012.

[51] INRIA Person Dataset. `http://pascal.inrialpes.fr/data/human/`.

[52] P. Dollar, Z. Tu, H. Tao, and S. Belongie, "Feature mining for image classification," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8, 2007.

[53] S. Maji, A. Berg, and J. Malik, "Classification using intersection kernel support vector machines is efficient," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8, 2008.

[54] A. Ess, B. Leibe, and L. Van Gool, "Depth and appearance for mobile scene analysis," in *11th IEEE International Conference on Computer Vision (ICCV)*, pp. 1–8, 2007.

[55] M. Enzweiler and D. Gavrila, "Monocular pedestrian detection: Survey and experiments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 12, pp. 2179–2195, 2009.

[56] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 304–311, June 2009.

[57] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Computing Surveys*, vol. 38, Dec. 2006.

[58] E. Parzen, "On estimation of a probability density function and mode," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.

[59] M. P. Wand and M. C. Jones, *Kernel Smoothing*. Monographs on statistics and applied probability, Boca Raton (Florida), London, New York: Chapman and Hall/CRC, 1st ed., 1994.

[60] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[61] Y. Cheng, "Mean shift, mode seeking, and clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, pp. 790–799, Aug. 1995.

[62] G. Bradski, "Real time face and object tracking as a component of a perceptual user interface," in *Fourth IEEE Workshop on Applications of Computer Vision (WACV)*, pp. 214–219, Oct. 1998.

[63] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, pp. 564–577, May 2003.

[64] A. Elgammal, R. Duraiswami, and L. Davis, "Probabilistic tracking in joint feature-spatial spaces," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. 781–788, June 2003.

[65] R. Collins, "Mean-shift blob tracking through scale space," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 234–240, June 2003.

[66] T. Lindeberg, "Feature detection with automatic scale selection," *International Journal of Computer Vision*, vol. 30, pp. 79–116, Nov. 1998.

[67] G. Hager, M. Dewan, and C. Stewart, "Multiple kernel tracking with SSD," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. 790–797, June 2004.

[68] M. Fashing and C. Tomasi, "Mean shift is a bound optimization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 471–474, Mar. 2005.

[69] Z. Fan, M. Yang, and Y. Wu, "Multiple collaborative kernel tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, pp. 1268–1273, July 2007.

[70] R. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME - Journal of Basic Engineering*, vol. 82, pp. 35–45, Mar. 1960.

[71] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, pp. 174–188, Feb. 2002.

[72] S. M. Ross, *Introduction to Probability Models.* Orlando, FL, USA: Academic Press, Inc., 9th ed., 2006.

[73] G. Welch and G. Bishop, "An introduction to the Kalman filter," Tech. Rep. TR 95-041, 2004.

[74] A. H. Jazwinski, *Stochastic Processes and Filtering Theory.* San Diego, CA, USA: Academic Press, Inc., 1970.

[75] S. Julier and J. Uhlmann, "A new extension of the Kalman filter to nonlinear systems," in *International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, vol. 3, pp. 2–3, Apr. 1997.

[76] E. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," in *IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium (AS-SPCC)*, pp. 153–158, 2000.

[77] Y. Rui and Y. Chen, "Better proposal distributions: object tracking using unscented particle filter," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 786–793, 2001.

[78] A. Doucet, S. Godsill, and C. Andrieu, "On sequential Monte Carlo sampling methods for Bayesian filtering," *Statistics and Computing*, vol. 10, pp. 197–208, 2000.

[79] M. Isard and A. Blake, "CONDENSATION-conditional density propagation for visual tracking," *International Journal of Computer Vision*, vol. 29, pp. 5–28, 1998.

[80] M. K. Pitt and N. Shephard, "Filtering via simulation: Auxiliary particle filters," *Journal of the American Statistical Association*, vol. 94, no. 446, pp. 590–599, 1999.

[81] C. Musso, N. Oudjane, and F. Le Gland, "Improving regularised particle filters," in *Sequential Monte Carlo Methods in Practice* (A. Doucet, N. de Freitas, and N. Gordon, eds.), Statistics for Engineering and Information Science, pp. 247–271, Springer New York, 2001.

[82] Z. Khan, T. Balch, and F. Dellaert, "A Rao-Blackwellized particle filter for Eigentracking," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 980–986, 2004.

[83] C. Rasmussen and G. Hager, "Probabilistic data association methods for tracking complex visual objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, pp. 560–576, June 2001.

[84] D. Reid, "An algorithm for tracking multiple targets," *IEEE Transactions on Automatic Control*, vol. 24, pp. 843–854, Dec. 1979.

[85] J. Vermaak, A. Doucet, and P. Perez, "Maintaining multimodality through mixture tracking," in *Ninth IEEE International Conference on Computer Vision (ICCV)*, vol. 2, pp. 1110–1116, Oct. 2003.

[86] K. Okuma, A. Taleghani, N. d. Freitas, J. J. Little, and D. G. Lowe, "A boosted particle filter: Multitarget detection and tracking," in *Computer Vision - ECCV 2004* (T. Pajdla and J. Matas, eds.), vol. 3021 of *Lecture Notes in Computer Science*, pp. 28–39, Springer Berlin Heidelberg, 2004.

[87] R. Mahler, "Multitarget Bayes filtering via first-order multitarget moments," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 39, pp. 1152–1178, Oct. 2003.

[88] B.-N. Vo and W.-K. Ma, "The Gaussian mixture probability hypothesis density filter," *IEEE Transactions on Signal Processing*, vol. 54, pp. 4091–4104, Nov. 2006.

[89] N. Whiteley, S. Singh, and S. Godsill, "Auxiliary particle implementation of probability hypothesis density filter," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 46, pp. 1437 –1454, July 2010.

[90] R. P. S. Mahler, *Statistical Multisource-Multitarget Information Fusion*. Norwood, MA, USA: Artech House, Inc., 2007.

[91] B. Han, Y. Zhu, D. Comaniciu, and L. Davis, "Visual tracking by continuous density propagation in sequential Bayesian filtering framework," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, pp. 919–930, May 2009.

[92] G. Wyszecki and W. S. Stiles, *Color Science: Concepts and Methods, Quantitative Data and Formulae*. Wiley Series in Pure and Applied Optics, Wiley-Interscience, 2 ed., Aug. 2000.

[93] F. Porikli, "Integral histogram: a fast way to extract histograms in Cartesian spaces," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. 829–836, 2005.

[94] L. Mason, J. Baxter, P. Bartlett, and M. Frean, "Boosting algorithms as gradient descent," in *In Advances in Neural Information Processing Systems 12 (NIPS)*, pp. 512–518, 2000.

[95] J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[96] N. Dalal, *Finding people in images and videos*. PhD thesis, Institut National Polytechnique de Grenoble, July 2006.

[97] K. Ali, F. Fleuret, D. Hasler, and P. Fua, "A real-time deformable detector," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 2, pp. 225–239, 2012.

[98] R. Benenson, M. Markus, T. Tuytelaars, and L. Van Gool, "Seeking the strongest rigid detector," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3666–3673, June 2013.

[99] L. Bourdev and J. Brandt, "Robust object detection via soft cascade," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 236–243, 2005.

[100] P. Viola, J. C. Platt, and C. Zhang, "Multiple instance boosting for object detection," in *Advances in Neural Information Processing Systems 18 (NIPS)*, pp. 1417–1426, Jan. 2007.

[101] P. Viola and M. Jones, "Fast and robust classification using asymmetric AdaBoost and a detector cascade," in *Advances in Neural Information Processing System 14 (NIPS)*, pp. 1311–1318, 2001.

[102] S. Hinterstoisser, V. Lepetit, S. Ilic, P. Fua, and N. Navab, "Dominant orientation templates for real-time detection of texture-less objects," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2257–2264, June 2010.

[103] S. Salvador and P. Chan, "Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms," in *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 576–584, 2004.

[104] L. Fei-Fei and P. Perona, "A Bayesian hierarchical model for learning natural scene categories," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 524–531, 2005.

[105] B. Babenko, P. Dollár, Z. Tu, and S. Belongie, "Simultaneous learning and alignment: Multi-instance and multi-pose learning," in *Workshop on Faces in "Real-Life" Images: Detection, Alignment, and Recognition*, Oct. 2008.

[106] T. Malisiewicz, A. Gupta, and A. Efros, "Ensemble of exemplar-SVMs for object detection and beyond," in *IEEE International Conference on Computer Vision (ICCV)*, pp. 89–96, 2011.

[107] A. Nabout and B. Tibken, "Object shape recognition using Mexican hat wavelet descriptors," in *IEEE International Conference on Control and Automation (ICCA)*, pp. 1313–1318, June 2007.

[108] CAVIAR Test Case Scenarios, 2004. `http://homepages.inf.ed.ac.uk/rbf/CAVIAR/`.

[109] B. Wu and R. Nevatia, "Detection and tracking of multiple, partially occluded humans by Bayesian combination of edgelet based part detectors," *International Journal of Computer Vision*, vol. 75, no. 2, pp. 247–266, 2007.

[110] W. Ge and R. T. Collins, "Multi-target data association by tracklets with unsupervised parameter estimation," in *British Machine Vision Conference (BMVC)*, vol. 2, p. 5, 2008.

[111] A. Torralba, K. P. Murphy, and W. T. Freeman, "Sharing visual features for multiclass and multiview object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 5, pp. 854–869, 2007.

[112] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd ed., 2004.

[113] L. Zhang, Y. Li, and R. Nevatia, "Global data association for multi-object tracking using network flows," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8, 2008.

[114] B. Benfold and I. Reid, "Stable multi-target tracking in real-time surveillance video," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3457–3464, 2011.

[115] A. Torralba and A. Efros, "Unbiased look at dataset bias," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1521–1528, 2011.