Constructions of High-Performance Face Recognition Pipeline and Embedded Deep Learning Framework

by

Him Wai Ng

B.ASc., (Hons.), Simon Fraser University, 2016

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Applied Science

in the
School of Engineering Science
Faculty of Applied Science

SIMON FRASER UNIVERSITY
Summer 2018

Approval

Name:	**Him Wai Ng**
Degree:	**Master of Applied Science**
Title:	**Constructions of High-Performance Face Recognition Pipeline and Embedded Deep Learning Framework**
Examining Committee:	**Chair: Carlo Menon** Professor
	Jie Liang Senior Supervisor Professor
	Faisal Beg Supervisor Professor
	Jiangchuan Liu Internal Examiner Professor
Date Defended/Approved:	June 28, 2018

Abstract

Face recognition has been very popular in many research and commercial studies. Due to the uniqueness of human faces, a robust face recognition system can be an alternative to biometrics such as the fingerprint or eye iris recognition in security systems. Recent development in deep learning contributed to many of the success in solving difficult computer vision tasks, including face recognition. In this thesis, a thorough study is presented to walk through the construction of a robust face recognition pipeline and to evaluate the components in each stage of the pipeline. The pipeline consists of four components, face detection module, face alignment module, metric space face feature extraction module, and feature identification module. Different implementations of each module are presented and compared. The performance of each implementation of the system is evaluated on multiple datasets. The combination of a coarse-to-fine convolutional neural network (CNN) based face detection, geometric-based face alignment and discriminative features learning with additive angular margin method are found to achieve the highest accuracies in all datasets.

One drawback of this face recognition pipeline is that it consumes a lot of computational resources, making it hard to be deployed on embedded hardware. It would be beneficial to develop a method that allows advanced deep learning algorithms to be run on resource-limited hardware, such that many of the existing devices can become intelligent with low cost. In this thesis, a novel lapped CNN (LCNN) architecture that is suitable for resource-limited embedded systems is developed. The LCNN uses a divide-and-conquer approach to apply convolution to a high-resolution image on embedded hardware. The LCNN first applies convolution to sub-patches of the image, then merges the resulting outputs to form the actual convolution. The resulting output is identical to that of applying a larger-scale convolution to the entire high-resolution image, except that the convolution operations on the sub-patches can be processed sequentially or parallelly by resource-limited hardware.

Keywords: Face Recognition, Deep Learning, Convolutional Neural Network, CNN Hardware Implementation, Receptive Field, Discriminative Features Learning, Surveillance

Acknowledgements

I would like to thank Professor Jie Liang for his guidance, supervision, and patience throughout the development of my thesis. I would like to thank Xing Wang for his time, effort, and constructive advice on my work.

I would also like to thank my parents and my girlfriend for their support and love throughout the completion of the thesis and my graduate career.

Finally, I would like to thank everyone from AltumView Systems Inc. for their understanding and support throughout the completion of the thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1.

Introduction

Human faces are unique and easily distinguishable. Researchers suggested that this is a result of evolution, such that human can identify each other easily [1]. By exploiting this evolutional feature, a series of improvement can be made to existing human-machine interaction, surveillance technology, and security systems. Traditional identification system such as fingerprint or iris recognition system requires a person to actively work together with the machine. Not only does this pose a high cognitive load to the person, it is also inefficient. In contrast, a face recognition system can verify the identification of the person without his or her notice, therefore, making it a potentially superior alternative.

The development of human face recognition system with computer began in the 1960s, in which Woodrow Bledsoe devised a technique called “man-machine facial recognition” [2]. Limited by the computing resources and imaging technology available at that time, Bledsoe had to use a graphics tablet (RAND TABLET), the tablet was used by an operator to extract the coordinates of facial features such as the center of pupils, the inside corner of eyes, the outside corner of eyes, location of nose tip, etc. A list of 20 distances is generated from these coordinates. Given a photograph of an unknown face, the system would use a method based on these distances to retrieve the image in the database most closely associated with the provided photograph. However, this system’s accuracy was inhibited by many factors such as different face angles, ages, lighting condition and so on. In 1987, a new way of comparing faces called Eigenface was introduced by Sirovich and Kirby [3]. Distinctions among different faces are computed using eigenvectors, these eigenvectors are computed from a covariance matrix using a technique called principal component analysis (PCA), this matrix was generated from measuring the distance between key features of a human face. EigenFace typically worked well on frontal faces and worked better than Bledsoe’s method on different face poses. However, such improvements were not enough to be used in practical scenarios [4].

Since then, computer technology has been improving vastly, new face recognition algorithms were developed to extract features from the faces that are beyond simple geometrical measurements. In 2002, Liu proposed to use a novel Gabor-Fisher classifier for face recognition [5]. In their work, They used Gabor wavelet to extract multi-oriental information from a human face [6]. 62 Gabor features were extracted from a face image, when tested on 600 FERET frontal face images corresponding to 200 subjects [7], their method achieved 100% accuracy even when the photos were acquired under variable illumination and facial expressions. However, the accuracy could still be inhibited by different face angles, ages, and poses. In addition, this method was only tested on a small-scale dataset, which was not representative to any kind of real-world scenario. During the same period, other feature extraction methods were also introduced. Chang proposed to use the histogram of oriented gradient (HOG) features of a human face to perform face recognition [8]; Rahim introduced the use of local binary pattern (LBP) to extract local face features [9]. However, all of these methods only worked well when the photo of a human face is in a frontal position, and preprocessing was needed to remove the effect of different illumination. This was far from being applied in daily life. It was not until the success of deep learning in other computer vision tasks [10] that researchers started to adopt deep learning techniques into face recognition to allow practical application in daily scenarios.

Deep learning is a sub-branch of a more general field called machine learning, in which researcher studies algorithms that can perform pattern recognition. Deep learning researchers focus on developing algorithms based on multilayer artificial neural networks (ANN). The first general, working learning algorithm for a deep, feedforward, multilayer artificial neural network was introduced by Ivakhnenko in 1965 [11]. In 1989, LeCun et al. took the standard backpropagation algorithm to train a deep neural network architecture called Neocognitron [12] to perform handwritten ZIP codes recognition on mail [13]. Although the algorithm was working, it took 3 days to train. This work later led to the development of a convolutional neural network (CNN) architecture Le-Net5 [14], which inspired many other CNN-based algorithms in the field. A lot of hype was created around this field in the 1990s, however, due to the limited amount of computational resources, training data and inherent issues such as vanishing gradient of these architectures. This family of algorithms was not able to train to outperform other algorithms such as support vector machine (SVM) in computer vision tasks. In 2006, a

series of papers published by Hinton, showed that a many-layered feedforward neural network could be effectively pre-trained one layer at a time, treating each layer in turn as an unsupervised restricted Boltzmann machine (RBM), then fine-tuning it using supervised backpropagation [15] [16]. These papers attracted the attention of researchers back to this field. With the advancement on Graphics Processing Unit (GPU), deep neural networks could be trained much faster than before [17], and the abundance of digital data on the internet also helped the training of these networks. In the year 2012, Krizhevsky won the ImageNet Large Scale Visual Recognition Challenge by a large margin by training an eight-layer CNN on GPU [18], this is the first work to show that deep learning algorithms such as CNN have significant advantages over traditional algorithms on computer vision task. Since then, more and more researchers have been applying CNNs on different computer vision tasks, including face recognition, and obtained state-of-the-art results.

However, CNNs are hungry in computational resources. For example, a simple network such as AlexNet [18] with only eight layers, already has 60 million parameters for computation. This is a very large number for embedded devices. Therefore, many deep learning algorithms can only be run on expensive computer hardware such as the Graphics Processing Unit (GPU). A goal of this thesis is to present a novel CNN architecture that allows the computation of expensive large CNNs on embedded hardware.

This thesis first introduces the construction of a face recognition pipeline based on deep learning algorithms, studies the different components of such pipeline, and investigates the effect of different implementations of each component. Then a novel CNN architecture called the lapped CNN (LCNN) is presented [19][20][21][22][23][24]. Experiments on the LCNN are also presented to study its performance on embedded hardware.

Chapter 1 presents the goals of this thesis, history of face recognition, and a brief introduction to deep learning.

Chapter 2 introduces the basics of machine learning, deep learning, and convolutional neural networks. This chapter also includes a brief overview of a practical face recognition pipeline, and a discussion on the contribution of each component in the

pipeline to the overall recognition accuracy. Four components are introduced, face detection module, face alignment module, metric space feature extraction module, and feature-based face identification module.

The detailed implementation of each component of the face recognition pipeline is presented in Chapter 3. One or more implementations are presented and compared for each component. For the face detection module, two algorithms are presented, they are both multi-task cascaded convolutional networks approaches [25][26]. For face alignment module, two alignment methods are presented, simple centering face method and geometrical alignment based on facial key points. For the metric space face feature extraction module, four architectures are presented, the FaceNet architecture [27], the SphereFace architecture [28], and the ArcFace architecture [29]. Finally, the feature comparison metric is presented in the feature identification module.

The experiments of using different implementations of the modules are presented in Chapter 4, along with an introduction to the test datasets used in this study. The accuracy of each configuration of the pipeline is tested on two datasets, one private dataset representing a daily surveillance scenario, and one private dataset representing the scenario of comparing input photos to ID photos. The best configuration is selected.

Chapter 5 introduces the LCNN architecture that can help to deploy advanced CNN algorithms into resource-limited embedded hardware [19][20][21][22][23][30]. The LCNN can decompose the computation of a large CNN into multiple smaller CNNs, the outputs of these smaller CNNs can be merged to become identical to the outputs of the large CNN. Thus, enabling the complicated CNN algorithms to be run on embedded hardware. The performance of the LCNN is also studied in the chapter.

The best configuration of the face recognition pipeline is studied in Chapter 6, along with a discussion of the LCNN performance.

Chapter 7 concludes this thesis with a summary of the study and a discussion of potential future work.

Chapter 2.

Background

2.1. Machine Learning

Machine learning is a subarea of artificial intelligence, which itself is a subfield of computer science. A classical definition of machine learning was given by Tom Mitchell in 1997: *"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."* [31] In other words, machine learning is a collection of algorithms that can learn to complete a task or to make accurate predictions, without being explicitly programmed to. The learning process is always done by passing in data or training samples to the computer, allowing it to automatically build a mathematical model of the task or the data based on the samples. This model can then be used to generate new predictions or to perform actions and make decisions. There are three major types of machine learning algorithms: supervised, unsupervised and, reinforcement learning.

2.1.1. Supervised Learning

Correct responses (targets) are passed into the computer in accompaniment with the training samples. Based on this input-target training set, the algorithm generalizes to produce correct responses to all possible inputs.

2.1.2. Unsupervised Learning

No targets are provided. Only the training samples are passed into the computer. The algorithm then tries to group similar samples together and form categories automatically. New input can then be classified into existing categories.

2.1.3. Reinforcement Learning

Instead of passing in the correct responses, the algorithm only gets told when the prediction is wrong. The goal of it is to explore all the different possibilities to maximize its score or to get to a correct answer.

In this study, supervised learning is the only relevant type of machine learning algorithms. In particular, deep learning is the only type of supervised learning that is relevant to this thesis.

2.2. Deep Learning

Deep learning refers to the use of many layers of artificial neural network (ANN) in order to perform recognition and classification of highly non-linear patterns in data [10]. Artificial neural networks are a family of computational models that were inspired by biological neural networks and are widely used for pattern recognition and classification [32]. Figure 1 shows a basic computation unit which is known as a neuron in an ANN.

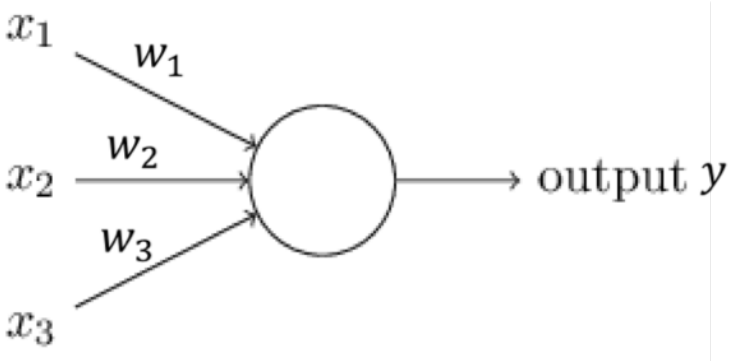


Figure 1 Structure of a neuron in ANN

The output of the neuron is related to the sum of the products of the inputs and their corresponding connection weights to the neuron:

$$y = f(x_1 w_1 + x_2 w_2 + x_3 w_3)\,, \tag{1}$$

where f(x) is a linear or non-linear transformation of the sum. In this thesis, three types of transformation were used for neurons in different layers:

$$f1(x) = \max(x, 0)\,, \tag{2}$$

$$f2(x) = \begin{cases} x, x > 0 \\ alpha * x, x < 0 \end{cases}, \tag{3}$$

$$f3(x) = \frac{1}{1+e^{-x}}, \tag{4}$$

where alpha in Equation (3) is an experimental parameter determined by model performance on test data. In the deep learning literatures, f1 is known as the rectifier linear unit (ReLU), f2 is known as leaky ReLU, and f3 is known as sigmoid.

A multiplayer ANN shown in Figure 2 is formed by connecting the neurons layer by layer, where each layer contains an arbitrary number of neurons.

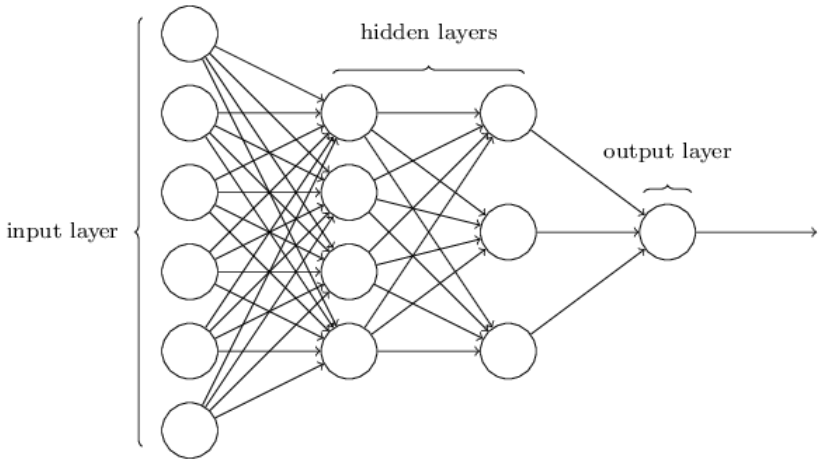


Figure 2 A multilayer neural network

This network can then be used to generate prediction at its output layer. The prediction is compared to the ideal predicted value from the training set and the error is back-propagated to the entire network. The values of the weights in each connection is then adjusted to minimize the prediction error [33]. This error minimization process is known as back-propagation. In this thesis, an algorithm known as stochastic gradient descent [34] was used for such purpose. The idea is to calculate the gradient of the cost resulting from the prediction error with respect to each weight in the network, then adjust the weights in the direction where the gradient is most negative. The algorithm can be summarized with the following equation:

$$\boldsymbol{W_{t+1}} = \boldsymbol{W_t} - \alpha \sum \nabla Q\big(\boldsymbol{E}(\boldsymbol{W}, \boldsymbol{X})\big)\,, \tag{5}$$

where Q(**E**) is the cost function of the prediction error, **E(W, X)** is the prediction error and is thus a function of the weight matrix **W**, the weights of the network at time t+1 is equal to the weights at time t minus some correction values calculated from the cost function.

2.3. Convolutional Neural Network

By arranging the connection between the neurons, one can create different ANN models with vastly different architectures. One type of ANN models that is particularly successful in computer vision tasks is a Convolutional neural network (CNN). Figure 3 shows an example of this network.

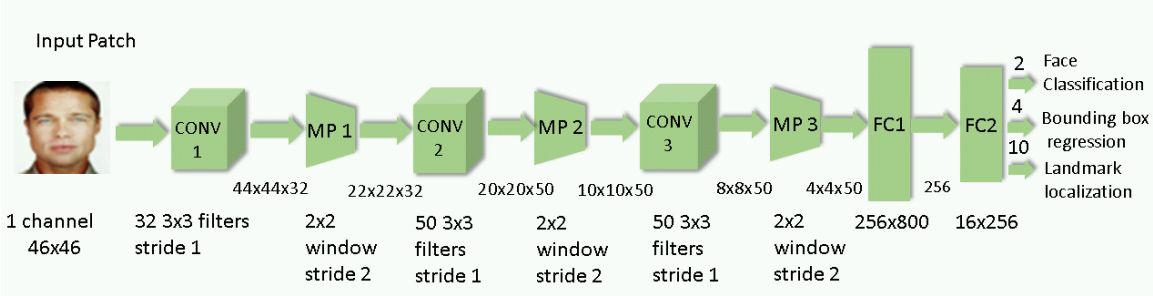


Figure 3 Architecture of a classical CNN, here for face detection, Each place is a feature map, i.e. a set of units whose weights are constrained to be identical.

Each convolution layer in a CNN contains a number (N) of two-dimensional m x k filters, and the weights of each filter are all shared for all location of the input vector. These filters are then applied to the input image through a process known as 2-D Convolution [35] and generate a number (N) of feature maps, the size of each feature map is calculated by:

$$h = \frac{(H-k+2p)}{s} + 1, \tag{6}$$

$$w = \frac{(W-m+2p)}{s} + 1, \tag{7}$$

where h is the height of the feature map, H is the height of input image, k is the height of the filter, w is the width of the feature map, W is the width of input image, m is the width of the filter, p is the convolutional padding, and s is the convolutional stride. The padding and stride size can be different along the width and height axes.

Convolutional layer helps in reducing the total number of weights that would be required when using a fully connected ANN. The outputs, which are usually referred to as feature maps, resulting from the convolution layer are then pooled to smaller size in the pooling or subsampling layer by either the max pooling operation or average pooling operation. This allows the network behaviour to become invariant to any translational transformation of the input. The last few layers of the CNN are usually fully connected ANN layers to perform classification or recognition tasks. The weights of the filters are updated by the back-propagation rules and adapted to the prediction task.

CNNs have been widely studied in the recent years due to its extraordinary performance in computer vision tasks such as image classification and object localization in images. There have been extensive studies showing that a CNN model is able to abstract the raw image input into high-level feature vectors within the network [12]. Therefore, enabling the possibility of transferring such learned feature vectors into other models for tasks related to image inputs and this technique is called transfer learning [4]. In this thesis, the task is to generate sparse features for face images for identification or verification.

2.4. Face Recognition Pipeline

Generically, a face recognition pipeline consists of four modules: the face detection module, the face alignment module, the metric space face feature extraction module, and the feature-based face identification module. The face detection module is responsible for identifying the location of all faces in an image, the face alignment module standardizes all of the faces appearing in different rotational angles and lighting condition to a normalized pixel distribution, the metric space face feature extraction module is then responsible for transferring these aligned faces in the color domain into an abstract vector space, in which each vector corresponds to one face, and ideally linearly separable from each other, such that distances between faces can be calculated. Finally, the face recognition module uses these features to identify the person corresponding to this face.

2.4.1. Face Detection Module

In order to perform face recognition reliably, one important condition is that all faces that appear in the scene must be detected and captured. This is the objective of the face detection module. Before the rise of deep learning, most of the face detection modules used either the Viola-Jones method [36] or HOG feature-based detection method [37]. Although Viola-Jones method can run on very cheap hardware with fast speed, it has a high missing rate. The HOG detection method is relatively more accurate than Viola-Jones method and can be trained to detect faces in different poses. Although the HOG detection method can still be run in real-time on a CPU, it has a slower running time, and the detection rate is also lower than modern CNN approaches. Therefore, many of the modern face detection modules that can use GPU have shifted from traditional detection algorithms to CNN-based algorithms. In this thesis, two CNN-based face detection methods are used for this module, one modified from the *Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks* [25]; and one *called A Convolutional Neural Network Cascade for Face Detection* [26]. The modification done in the first algorithm was mainly to speed up the processing time, such that not only can it be run on GPU machines, it can also be run on decent embedded hardware.

2.4.2. Face Alignment Module

Face alignment is an important step in a face recognition pipeline. Faces detected in the scene can be appeared in different poses and light conditions. It is important to have a method to bring all of these faces to a constant position, such that different faces can be compared to each other in a standard manner. One example is a rotated face, when comparing a rotated face to a reference face image, it is necessary to rotate one of the faces to match the rotational angle of another face in order to extract comparable face features in the later stage. In this thesis, two methods are used to implement this module, one that simply crops the face to become the center of an image; and one that performs similarity transformation [38] of the face to a reference position based on face landmarks.

2.4.3. Metric Space Face Feature Extraction Module

After the face images are aligned, they are mapped to a metric space, producing feature vectors, such that distances between them can be calculated and the similarities between them can be quantified. This process can be achieved by applying a series of carefully designed non-linear transformations to the image. In this thesis, such transformation is done by CNNs and the transformation is learned from data. In order to produce accurate comparison between faces, these feature vectors must have the property that faces of the same identity are very close to each other and faces of different identities are very far away from each other.

2.4.4. Feature Identification Module

This is the last stage of the face recognition pipeline, in which the extracted metric space face features are used to compare with the ones that were stored in the system in advance. The similarities of these features indicate how close they are in the metric space and can be used as a confidence value to determine if two or multiple face images belong to the same person. A summary of the face recognition pipeline is presented in Figure 4.

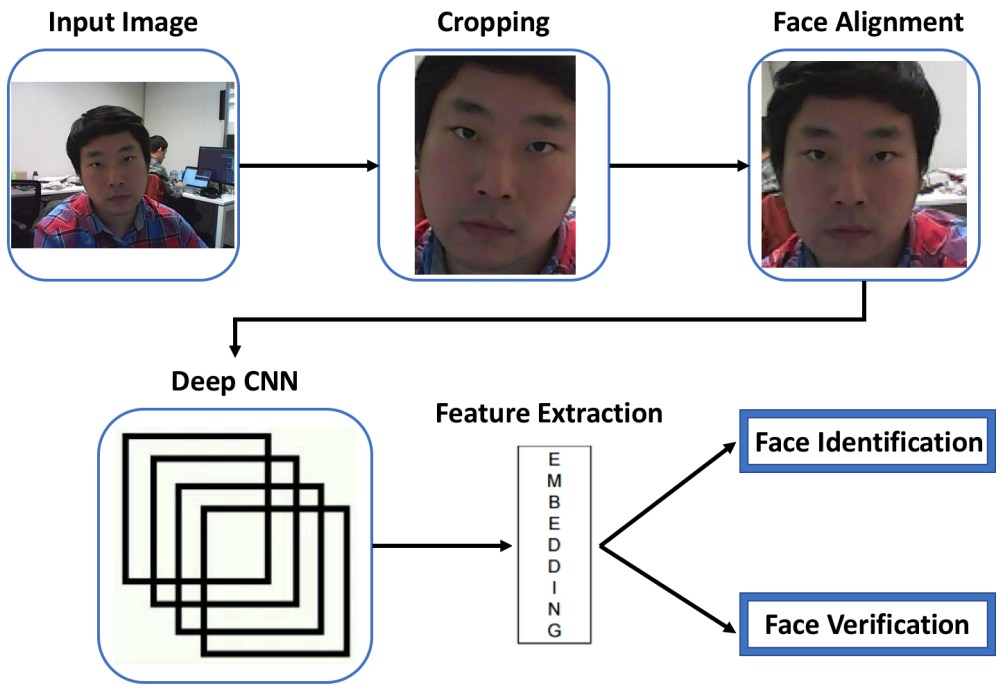


Figure 4 Schematic Diagram of a Face Recognition Pipeline

Chapter 3.

Implementation of the Modules in Face Recognition Pipeline

For all of the modules, the implementation was done with TensorFlow [39], including both the training and testing. However, the final pipeline was ported into a production environment for real-time performance using the TensorFlow C++ API. All of the software implementation and testing were done on a computer with the following hardware specification:

Table 1 Computer specification used for this thesis

Processor	i7 Skylake 6700k 4.3Ghz
Memory	32GB DDR4
GPU	GTX 1080 8GB
Storage	1TB SSD

3.1. Face Detection Module

Face detection is the first and most critical module in the face recognition pipeline. The detection accuracy of this module is an important factor. If the true positive detection rate of this module is low, then a lot of faces will be missed and not recognized. If the false positive rate is high, then a lot of images will be considered as faces and this will affect the recognition accuracy. Computational speed of this module is another important factor, when there are a considerable number of faces appearing in the scene, the system must be able to detect and send these faces to the later stages of the pipeline in real-time, otherwise, surveillance systems that rely on face recognition technology won't be effective. In this thesis, two implementations of the face detection

module are presented. Both of these methods are tested in experiments in Chapter 4. The one that gives the best detection rate will be used for the pipeline.

3.1.1. Modified Coarse-to-fine multi-stage CNN

The first method was modified from the Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks (MTCNN), it was proposed by Zhang in 2016 and has been one of the most popular state-of-the-art face detection methods in the deep learning-based field [25]. This method uses three stages of CNNs to perform a coarse-to-fine face detection. Figure 5 shows a high-level overview of this coarse-to-fine detection process. In this thesis, the modified method still follows closely to the original algorithm, except for some parameters of the network architectures.

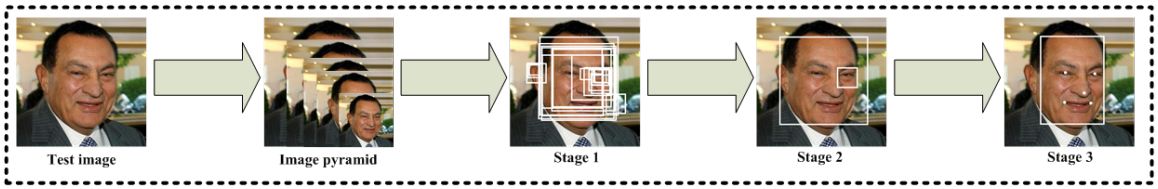


Figure 5 Overview of the MTCNN detection stages. The Input image went through three stages to refine the location of a face [25].

The original MTCNN method takes in a colour image as input, then generates an image pyramid by resizing the image down to different sizes repeatedly with a constant scaling ratio. This process generates a set of images that have different sizes, and the smallest size of the image in this pyramid is 12 by 12 pixels. Each of these images in the image pyramid is first fed into the first stage CNN, which is called a P-Net and its architecture is shown in Figure 6. The P-Net is a fully convolutional network (FCN) and consists of four convolutional stages, but only the first convolutional layer is followed by a max pooling layer. The second last stage is a convolutional layer with 32 output channels, and each output channel is connected to three different convolutional layers in parallel. As a result, the final layers of the network produce three outputs, the probability of the input image is a face, the location of these faces, and the location of the facial landmarks of these faces. It is clear that for each input image, the P-Net can propose an arbitrary number of possible faces, each corresponds to a region of 12 by 12 pixels on the input image.

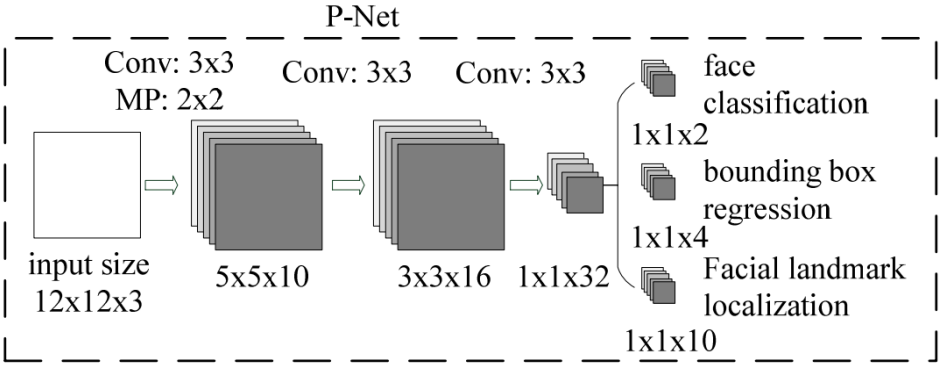


Figure 6 Overview of the MTCNN P-Net architecture. P-Net consists of 2 fully convolutional layers and produces it outputs in the last parallel fully connected layer.

After processing all of the images in the image pyramid, the P-Net will have proposed a number of faces, but with a very high false positive rate. Thresholding on the probabilities of these faces can reduce the number of false positive faces, but further refinement is needed. These proposed faces are then scaled to 24 by 24 pixels before passing to the R-Net, which further refines the probabilities and location of these faces. After the R-Net refinement, a large number of false positive faces would have been discarded. The architecture of R-Net is shown in Figure 7.

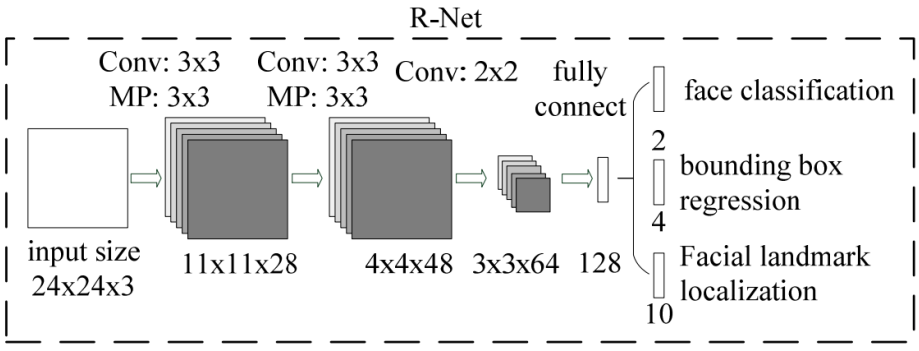


Figure 7 Overview of the MTCNN R-Net architecture. R-Net consists of 3 fully convolutional layers and 3 fully connected layers. It produces it outputs in the last parallel fully connected layers.

The R-Net uses a fully connected layer as its output, besides that, it is very similar to the P-Net. A final refinement stage is then performed using the O-Net. The architecture of O-Net is shown in Figure 8. The remaining proposed faces from R-Net are scaled to 48 by 48 pixels, then the O-Net used a deeper CNN architecture to produce the final detection results. The results contain three types of information, the probability of the detected face is really a human face, the location of the face, and the

refined face landmark locations. The location of the face consists of four coordinates of a box surrounding the face, the horizontal pixel value of top left corner, the vertical pixel value of top left corner, the horizontal pixel value of bottom right corner, and the vertical pixel value of bottom right corner.

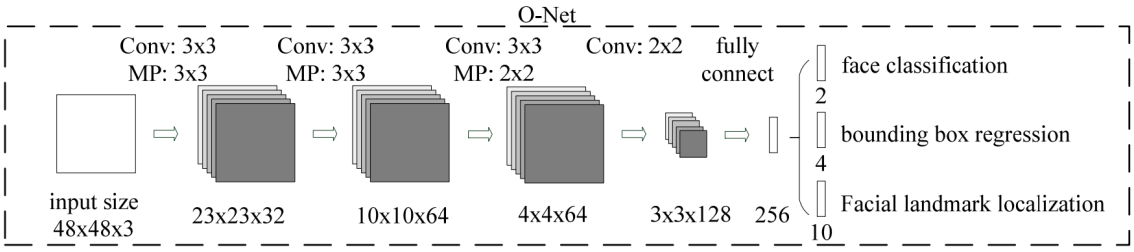


Figure 8 Overview of the MTCNN O-Net architecture. O-Net consists of 4 fully convolutional layers and 3 fully connected layers. It produces it outputs in the last parallel fully connected layer.

After the final O-Net refinement stage, all face regions in the image can be detected. However, it is possible that some face regions are highly overlapping and in fact referring to the same face, therefore, an algorithm called non-maximal suppression (NMS) [40] is applied to the overlapping regions to finalize the location of this face. Figure 9 shows an example of before and after applying NMS.

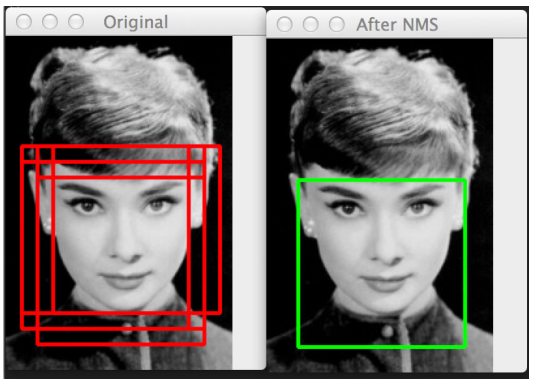


Figure 9 Example of applying NMS to remove redundant bounding boxed [41]*. Left: Original overlapping bounding boxes. Right: One bounding box after NMS.*

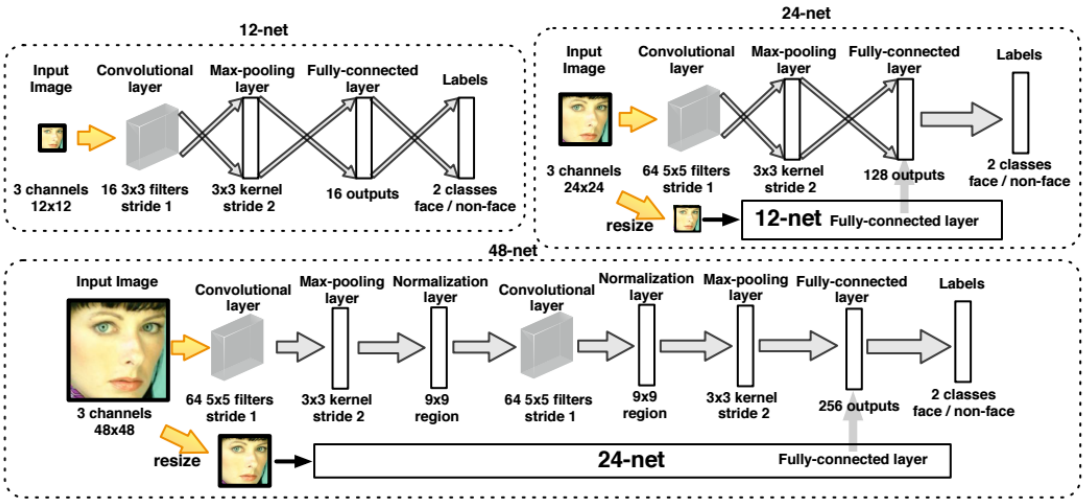
Training Details

In this thesis, the MTCNN face detector architecture was modified to contain less convolutional layers and accepts a larger input size of 16 by 16 pixels in the P-Net. This modified MTCNN is called Hi3519-MTCNN, because it was targeted to run on the Hi3519 embedded hardware board. The Hi3519-MTCNN was first trained from scratch on two datasets, the WIDER FACE dataset [42] and CelebA dataset [43]. WIDER FACE

dataset is a face detection benchmark dataset. There are 32,203 images and 393,703 face labels. The images have a high variation in scale, pose, and occlusion. WIDER FACE dataset is organized based on 61 event classes. For each event class, 40% of the data were used for training, 10% for evaluation and 50% for testing. The WIDER FACE dataset was used to train the face bounding box regression task. For the face landmark task, CelebA is used. CelebA is a large-scale face attributes dataset with more than 200K celebrity images, each with 40 attribute annotations.

3.1.2. Cascaded CNN for Face Detection

The second method is similar to the first method. The algorithm also performs face detection in three stages in a coarse-to-fine manner. However, for each stage, it splits up the face location regression and face classification tasks into two different neural networks. The face classification part of this algorithm is shown in Figure 10 and the face location regression part is shown in Figure 11. It can be seen that the network architectures and stages are similar to MTCNN. The differences are the output stages, in



which the outputs are separated into two networks.

Figure 10 Face classification part of the Cascaded CNN algorithm [26]*. Similar to Hi3519-MTCNN, the algorithm used a coarse-to-fine approach to refine the detection of a face.*

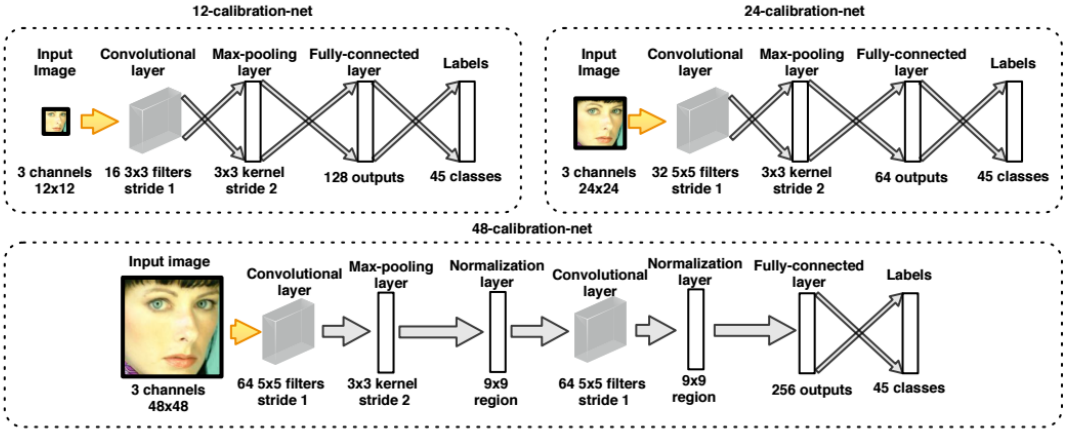


Figure 11 Face location regression part of the Cascaded CNN algorithm [26]*. Similar to Hi3519-MTCNN, the algorithm used a coarse-to-fine approach to refine the location of a face.*

Training Details

Similar to the Hi3519-MTCNN, the networks were trained from scratch on the WIDER FACE dataset [42] and CelebA dataset [43].

3.2. Face Alignment Module

Face alignment acts as a preprocessing step in the face recognition pipeline, in which it helps to standardize the input image. This helps to minimize the impact on the recognition accuracy that can be caused by different light conditions, colour distortions, and pose variations. Typically, face alignment uses the face landmarks of a face to align it to a certain position in the input image. For the face detection methods that are used in this thesis, an external software library called Dlib [44] is used to obtained face landmarks for the detected faces. Five landmarks are used for alignment, the location of the two eye centers, the location of the nose tip, and the location of the two mouth corners. Once the face landmarks are obtained, two methods are used to perform face alignment, cropping the face out and place padding to make it the center of an input image; and using similarity transform [38] to align the face with reference to a set of standard face landmarks. In this thesis, the values of the reference set of face landmarks were obtained from the authors of MTCNN. The reason to use two alignment methods is that in the later stage of the pipeline, different face feature extraction methods use different alignment methods.

3.2.1. Cropping Face to Center

The first method uses the facial landmarks to crop the face image out of the input scene, and then adjust the location of the face such that it is at the center of the cropped image. Five landmarks are used to perform cropping and centering, the location of two eye centers; the location of the nose tip; and the location of the two mouth corners. Figure 12 shows an example of these landmarks and the process of cropping out the face out and place it in the center of a smaller image.

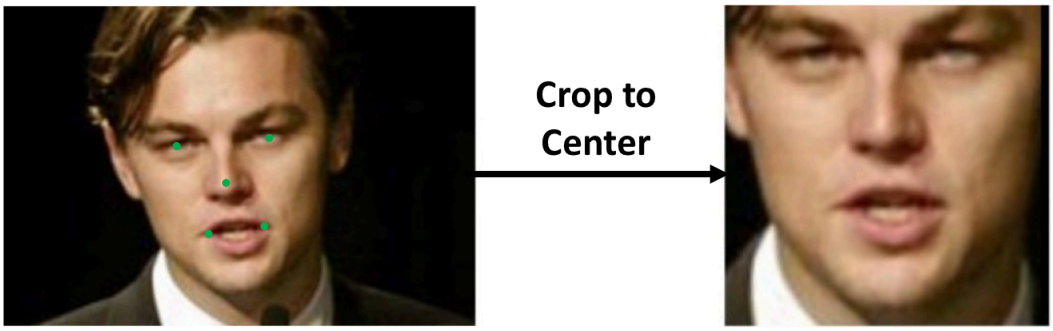


Figure 12 Example of the five-point face landmarks (left) and the resulting center cropped image (right).

Additional scene content can be added as background to the face image by adjusting the cropping padding size. In this thesis, padding size of 20 pixels is used.

3.2.2. Similarity Transform to Reference Facial Landmarks

The second method also uses the face landmarks to crop out the face image. However, different from the previous method, this method first applies a similarity transform to the scene image, such that the locations of the detected facial landmarks in the scene image get mapped to a reference set of face landmark locations through image transformation. Then the face is cropped from this transformed image. Figure 13 shows an example of this transformation. Another different point from the previous method is that the similarity transform can cause certain distortion to the faces. But it helps to transform any kind of face poses to a standard frontal face pose.

Figure 13 Example of a similarity transform. The right image contains a face that was rotated, after similarity transform, the resulting face in right image is rotated back to the frontal face orientation.

3.3. Metric Space Face Feature Extraction Module

In order to compare the similarity or difference between two faces, it is important to find a representation of these faces such that the similarity can be quantified. This is the goal of the face feature extraction module. The aligned faces are mapped to a metric space such that face vectors belonging to the same identity are very close to each other, while the face vectors belonging to different identities are far away from each other. There have been extended researches on finding an effective mapping for this purpose. In this thesis, this is implemented with CNNs. Three architectures are evaluated, the FaceNet approach [27], the SphereFace approach [28], and the ArcFace approach [29]. The FaceNet approach uses a metric learning method to generate unified face embedding in the Euclidean space for direct distance comparison. While the other two approaches modify the softmax loss function to generate features that have small intra-class (same identity) distance and large inter-class (different identity) distance in the angular space.

3.3.1. Unified Face Embedding in Euclidean Space with FaceNet

The FaceNet approach uses a metric learning method called Triplet Loss to produce face embedding in Euclidean space [27]. During the end-to-end training of the deep CNN, an extra loss term is placed at the end. Figure 14 shows the FaceNet structure.

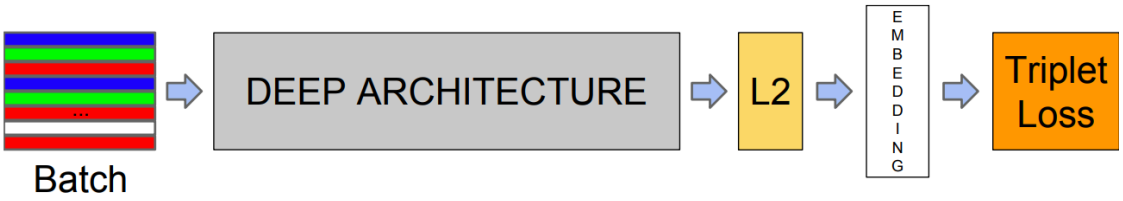


Figure 14 Structure of the FaceNet architecture [27]*, the face features are contained in the embedding layer for triple loss training.*

The goal of Triplet Loss is to train the network to produce an embedding of a face in the end stage, such that the embedding is a d-dimensional Euclidean space vector and lives on the d-dimensional hypersphere with a radius equal to 1. In this thesis, the dimension of the embedding is 128. Another constraint for the embedding is that an image x_i^a (an anchor) of a person is closer to all other images x_i^p (positive samples) of the same person than to any images x_i^n (negative samples) of different people. This combination of anchor, positive images and negative images in a set is called a triplet and is visualized in Figure 15.

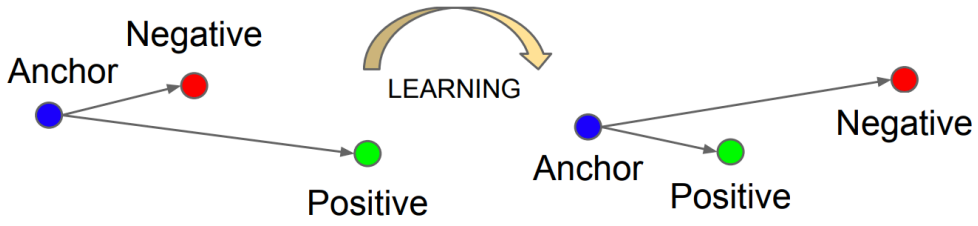


Figure 15 Visualization of the triplets [27]*. The distance between the anchor and negative become larger than the distance between the anchor and the positive after training with Triplet Loss.*

The constraints can be achieved by carefully designing the triple loss function and selecting the triplet pairs during training. Equation of the Triplet Loss is shown in following:

$$L = \sum_i^N [\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha]_+ , \quad (7)$$

$$\forall \left(f(x_i^a), f(x_i^p), f(x_i^n)\right) \in \mathcal{T} \quad (8)$$

where $f(x)$ is the output from the last CNN layer and is a non-linear transformation of the input image x, α is the margin that is enforced between positive and negative face pairs. $\mathcal{T}$ is the set of all possible triplets in the training data and has cardinality N. The equation basically forces the network to behave in a way that the L2 distance between

the anchor and the positive images are at least a margin α larger than the L2 distance between the anchor and the negative images.

In selecting the triplets to train the network, it is important to select triplets that violate the triplet constraint in (5), such that the network can converge fast due to the large error from the prediction. As a result, the selected triplet pairs should satisfy the following conditions:

$$argmax_{x_i^p} \left\| f(x_i^a) - f\left(x_i^p\right) \right\|_2^2, \tag{9}$$

$$argmin_{x_i^n} \| f(x_i^a) - f(x_i^n) \|_2^2 \tag{10}$$

In this thesis, the triplets are sampled from the training data in a mini-batch style. For each mini-batch, there are around 40 images per identity, and for each identity, randomly sampled negative images from the entire training set are added to it.

The network architecture used for the deep CNN part of the FaceNet approach is called Inception-Resnet-V1 [45].

Training Details

In this thesis, a pretrained FaceNet Inception-Resnet-v1 network was obtained from an open source project [46]. The network was then fine-tuned on a dataset called AsiaFace. This dataset consists of 65000 Asian identities collected from interet and there are 50 images for each identity on average, forming a total of 3 million images dataset. Examples of images from this dataset are shown in Figure 17. However, since this dataset was collected mainly for training the FaceNet architecture, all of the data are stored in triplets, making it hard to be used for training other network architectures.

Figure 16 Example images from the AsiaFace dataset.

3.3.2. SphereFace – Multiplicative Angular Margin

Similar to FaceNet, SphereFace also targets to generate face embedding in a metric space [28], but different from FaceNet, SphereFace does not generate unified embedding for a face in the Euclidean space. Instead, it introduces an angular margin between the faces in the metric space. Through modifying the softmax loss during the face classification task training, SphereFace can train the network to generate face features that have smaller maximal intra-class distance than the minimal inter-class distance in the chosen metric space.

In a typical multi-class classification training, softmax loss is a common loss function [47] and it has the form:

$$L = \frac{-1}{N}\sum_i \log\left(\frac{e^{(W_{y_i}^T x_i + b_{y_i})}}{\sum_{j=1}^{n} e^{(W_j^T x_i + b_j)}}\right), \tag{11}$$

where $\boldsymbol{x}_i$ is the output from the last CNN layer to the fully connected layer for the i-th training sample, y_i is the network prediction of the i-th training sample, $\boldsymbol{W}_{y_i}$ is the weights of the fully connected layer that corresponds to the prediction y_i, b is the bias term, n is the number of prediction classes, and N is the number training samples.

The softmax loss can also be rewritten in the dot product cosine form:

$$L = \frac{-1}{N}\sum_i \log\left(\frac{e^{\left(\left\|W_{y_i}\right\|\|x_i\|\cos(\theta_{y_i,i})+ b_{y_i}\right)}}{\sum_{j=1}^{n} e^{\left(\left\|W_j\right\|\|x_i\|\cos(\theta_{j,i})+ b_j\right)}}\right), \quad (12)$$

where $\theta_{j,i}$ is the angle between vector $\boldsymbol{W}_j$ and $\boldsymbol{x}_i$.

In SphereFace, the algorithm first normalizes $\|\boldsymbol{W}_j\| = 1$ for all j, and zeros the bias. As a result, this modification forced the learned features $\boldsymbol{x}_i$ to have angular boundaries. However, this is not enough to create a large angular margin between the features. In order to increase the angular margin between the learned features, a multiplicative term m is introduced into the angle $\theta_{j,i}$ such that the loss function becomes:

$$L_{ang} = \frac{-1}{N}\sum_i \log\left(\frac{e^{(\|x_i\|\cos(m\theta_{y_i,i}))}}{e^{(\|x_i\|\cos(m\theta_{y_i,i}))}+\sum_{j\neq y_i} e^{(\|x_i\|\cos(\theta_{j,i}))}}\right), \quad (13)$$

where $\theta_{y_i,i}$ now has the range $\left[0,\frac{\pi}{m}\right]$ and the learned features have an angular margin of $\frac{m-1}{m+1}\theta_j^i$ where θ_j^i is the angle between $\boldsymbol{W}_i$ and $\boldsymbol{W}_j$. During training, since gradient descent requires the loss function to be continuous and differentiable, therefore, the definition range of $\cos(\theta_{y_i,i}))$ is expanded by using a monotonically decreasing angle function $\psi(\theta_{y_i,i}))$. The function equals to $\cos(\theta_{y_i,i}))$ in $\left[0,\frac{\pi}{m}\right]$ and thus the loss function becomes:

$$L_{ang} = \frac{-1}{N}\sum_i \log\left(\frac{e^{(\|x_i\|\psi(m\theta_{y_i,i}))}}{e^{(\|x_i\|\psi(m\theta_{y_i,i}))}+\sum_{j\neq y_i} e^{(\|x_i\|\cos(\theta_{j,i}))}}\right), \quad (14)$$

where $\psi\left(\theta_{y_i,i}\right) = (-1)^k\cos(\theta_{y_i,i})) - 2k$ for $\theta_{y_i,i} \in \left[\frac{k\pi}{m},\frac{(k+1)\pi}{m}\right]$ and $k \in [0, m-1]. m \geq 1$ controls the size of angular margin and $m = 1$ only introduces angular boundaries between features.

The network architecture used to generate the convolutional features to the fully connected layers of SphereFace architecture is shown in Figure 18.

Layer	20-layer CNN
Conv1.x	$[3\times3, 64]\times1$, S2 $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 1$
Conv2.x	$[3\times3, 128]\times1$, S2 $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$
Conv3.x	$[3\times3, 256]\times1$, S2 $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 4$
Conv4.x	$[3\times3, 512]\times1$, S2 $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 1$
FC1	512

Figure 17 Network architecture of the SphereFace algorithm.

Training Details

In this thesis, the network was firstly trained on the VGGFace2 dataset [48]. VGGFace2 has 3.31 millions of face images for a total of 9131 identities. The faces in this dataset were also horizontally flipped during training for data augmentation. Then the network was fine-tuned on a dataset called AsiaFace-small. The AsiaFace-small is a smaller dataset compared to AsiaFace, it contains 400,000 images for 13,229 Asian identities. Although the dataset is smaller, it is cleaner and in a more general storage format that can be used for training many other network types.

3.3.3. ArcFace – Additive Angular Margin

Similar to SphereFace, ArcFace also modifies the softmax loss function to create an angular margin between face features [29]. The different is that instead of introducing a multiplicative term into the angle between the weight vector and feature vector, ArcFace introduces an additive term and performs feature normalization, i.e. $\|\boldsymbol{x}_i\| = s$, where s is some constant. The loss function therefore becomes:

$$L_{ang} = \frac{-1}{N}\sum_i \log\left(\frac{e^{(s\cos(\theta_{y_i,i}+m))}}{e^{(s\cos(\theta_{y_i,i}+m))}+\sum_{j\neq y_i}^{n} e^{(s\cos(\theta_{j,i}))}}\right), \quad (15)$$

where s = 30 in this thesis.

Another difference is that authors of ArcFace have studied other CNN architectures carefully and came up with modifications to existing architectures that are more robust to large face pose changes and age variations. The architecture that found to perform the best is referred as SE-LResNet50E-IR in the original paper and this is the architecture used in this thesis.

Training Details

In this thesis, the network was firstly trained on the VGGFace2 dataset [48]. After the training on VGGFace2, the network was fine-tuned on a dataset called AsiaFace-small.

3.4. Feature Identification Module

The feature identification serves two purposes in the face recognition pipeline, matching faces in the scene to identities in a database and verifying the faces in the scene against some images.

3.4.1. Matching faces to known identities in a database

After the face features for all faces are extracted, the face identification module uses a similarity metric to compute the similarities of these faces against all of the face features of known identities in a database. In this thesis, a cosine similarity metric is used, such that 0 means the probability of the same person is 0% and 1 means the probability of the same person is 100%. The face features that need to be identified are stored in a matrix M_i and the face features in the database are stored in another matrix M_r, given that all of the face features are normalized, the similarities can be obtain by $M_i^T \cdot M_r$. The resulting similarities are then scaled and sorted. The identity that has the highest similarity is the most probable matching, given that the similarity exceeds a certain threshold. In this thesis, the threshold is set to 0.7, i.e. probability being the same person exceeds 70%.

3.4.2. Verifying faces against some reference images:

When a person appears in a scene, the module can compare the extracted face features against one or more photos of this person to verify they are the same identity. The procedure is similar to the identification task, but it is much simpler because no dataset search is needed. Similarities between the person in the scene and his or her photos can be used to confirm the person's claimed identity.

Chapter 4.

Experiments and Performances of Different Pipeline Configurations

In order to select the best configuration for the face recognition pipeline, the performance of each module is evaluated. For each module, the performance of each implementation for the module's task is compared to each other. The one that performs the best is selected to use in the pipeline.

4.1. Face Detection Module

There are two important factors to determine the performance of this module, detection accuracy and detection speed. It was explained earlier that detection accuracy determines the final recognition rate and detection speed determines the real-time applicability of the pipeline. Both the Hi3519-MTCNN and Cascaded CNN face detectors are tested on the FDDB dataset [49]. FDDB is a data set of face regions designed for studying the problem of unconstrained face detection. This dataset contains locations of 5171 faces in a set of 2845 images.

Detection accuracies of the two implementations are shown in Figure 19. Five algorithms are compared, Cascaded CNN [26]; two Hi3519-MTCNN variations; Viola-Jones [36] face detection; and the structural face detection method by Yan [50]. Cascaded CNN and the two Hi3519 algorithms were implemented in this thesis, while the other results were reported by the authors and used as references in here. The only difference between the two Hi3519-MTCNN detectors is the input size in P-Net. Although the default input size is 16 by 16 pixels, theoretically, a smaller input size can increase the detection accuracy, therefore, a smaller input size of 12 by 12 pixels was also implemented.

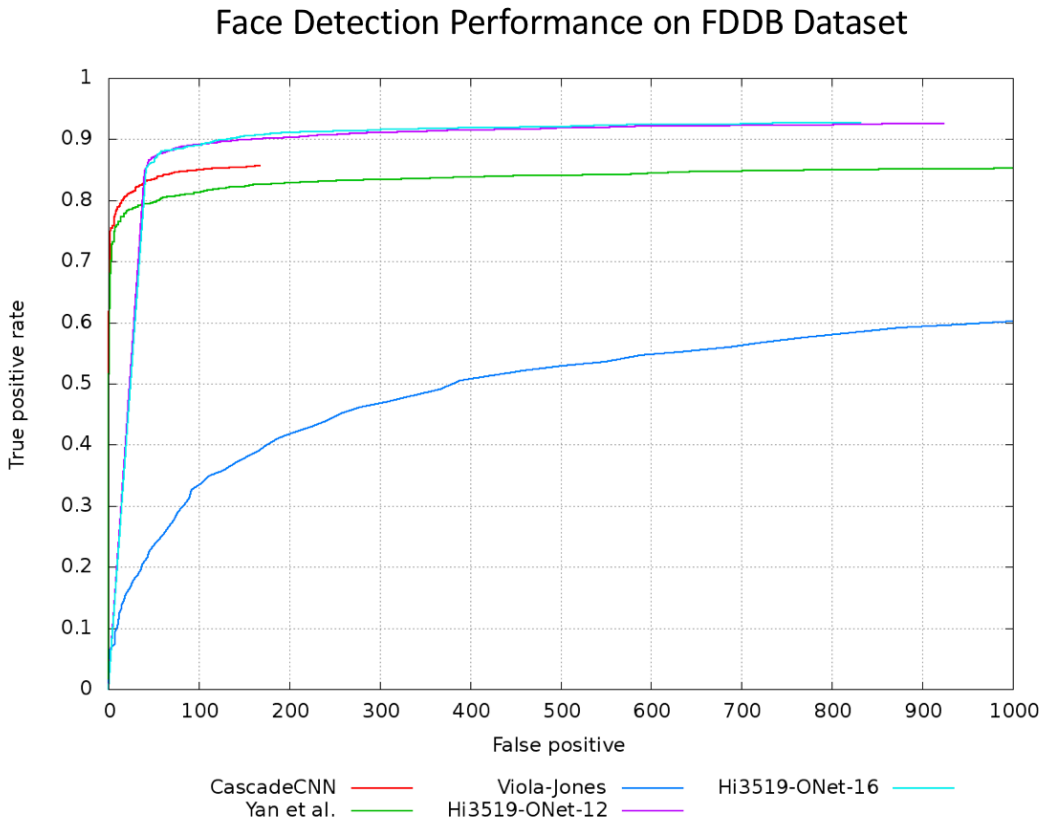


Figure 18 Performance of different face detection algorithm on the FDDB dataset. The CNN based methods have clear advantages over traditional methods. The best performance is achieved by the two Hi3519-MTCNN methods, and it was found that the input image size for P-Net did not affect the accuracy much.

It can be seen that the best performance was achieved by CNN-based methods. However, the two Hi3519-MTCNN detectors achieved much higher accuracies than the cascaded CNN detector. With a very low accepted false positive number of 35 faces, the two Hi3519-MTCNN detectors have 5% higher true positive rate than the Cascaded CNN detector. It is also found that there is no significant performance difference between the two Hi3519-MTCNN detectors, at an acceptable false positive number of 35 faces, both detectors achieve 85% true positive rate. When the number of accepted false positive number of face increase to 100, the two detectors can achieve 89% true positive rate, which is a state-of-the-art result.

Besides detection accuracy, the processing times of the CNN-based detectors are also reported in Table 2. All detectors were run on the computer specified in Table 1 on one image. Minimum detectable face size was set to 16 by 16 pixel, and the input image size is 640 by 480 pixels. A total of 9 clear faces appeared in the image and this

image is shown in Figure 20. Original MTCNN detector's speed was also reported as a reference.

Table 2 Average processing time of different face detection algorithms.

Algorithm	Image Resolution (pixels)	Number of Clear Face(s) in Image	Average CPU Processing Time (ms)	Average GPU Processing Time (ms)
Hi3519-MTCNN (P-Net input size 12 pixels)	640 x 480	9	150.1	5.2
Hi3519-MTCNN (P-Net input size 16 pixels)	640 x 480	9	**148.4**	**4.6**
Cascaded CNN	640 x 480	9	200	8.9
Original MTCNN	640 x 480	9	172.7	6.8

Figure 19 Image use to test the face detectors' processing speed

As can be seen from Table 2, the Hi3519-MTCNN detectors achieve the fastest processing time, which is expected since the architecture was designed to run fast. Similar to the detection accuracy results, there is no significant difference between the two Hi3519-MTCNN detectors. In conclusion, the Hi3519-MTCNN detector with 16 by 16 pixels P-Net input size is the best for the face detection module.

4.2. Metric Space Face Feature Extraction Module

For the feature extraction module, besides computational speed, the most important requirement is that the feature distances to the intra class (same identities) features must be smaller than the distances to all inter class (different identities) features. If the requirement is satisfied, then the recognition accuracy rate using these feature distances can be very high. In this thesis, two datasets representing different scenarios are used to test the performances of the different implementations.

4.2.1. Dataset1

Dataset1 consists of two sets of identity images. The first set contains 700 face images of different identities captured from a surveillance video stream. The second set contains 50,000 ID photos, with 700 of the ID photos correspond to the 700 captured faces in the first set.

The testing task is to match each of the 700 captured face images to its corresponding ID photos in the 50,000-images dataset. The testing implementation must first calculate the similarities between the captured face images to the 50,000 ID photos. For each captured face image, the similarities between it and all of the 50,000 ID photos are used to find the matched ID photos. The testing results after matching all 700 captured face images are reported in Table 3, where top-1 accuracy corresponds to the accuracy that the ID photo with the highest similarity is the true one corresponding to captured face image; top-5 accuracy corresponds to the accuracy that the true ID photo corresponding to the captured face image is within the top five similarities; and top-10 accuracy corresponds to the accuracy that the true ID photo corresponding to the captured face image is within the top ten similarities.

Table 3 Testing results of different implementation on Dataset1

Implementation	Top-1 Accuracy	Top-5 Accuracy	Top-10 Accuracy
FaceNet	56.3%	72.4%	78.6%
SphereFace	57%	71.5%	76.3%
ArcFace	**83.1%**	**91.2%**	**93.7%**

From the results, it can be seen that ArcFace achieved the best accuracy, while the SphereFace performed the worst. Also, it is noted that the accuracies achieved by ArcFace were much higher than the other two methods by a large margin, which is surprising since both the ArcFace and SphereFace target to increase the angular margin between features.

4.2.2. Dataset2

Dataset2 consists of two sets of identity images. The first set contains 100 ID photos of different identities. The second set contains 50,000 ID photos plus 100 daily camera photos, and those 100 camera photos correspond to the 100 ID photos in the first set.

The testing task is to match each of the 100 ID photos to its corresponding photos in the 50,100-images dataset. The testing implementation must first calculate the similarities between the ID photos to the 50,100 photos. For ID photos, the similarities between it to all of the 50,100 ID photos are used to find the matched photo. The testing results after matching all 100 ID photos are reported in Table 4. In this dataset, it can be seen the ArcFace achieves the best performance again, with SphereFace and FaceNet being comparable to each other.

Table 4 Testing results of different implementation on Dataset2

Implementation	Top-1 Accuracy	Top-5 Accuracy	Top-10 Accuracy
FaceNet	64.2%	78%	81.7%
SphereFace	66.7%	78.5%	82.3%
ArcFace	**93.9%**	**99.1%**	**100%**

The processing speeds of the implementation to extract face features are also shown in Table 5. The test was run on a single image on both the CPU and GPU of the computer in Table 1. The test image has a resolution of 600 by 800 pixels.

Table 5 Average processing time of different face feature extraction methods on one face image

Algorithm	Image Resolution (pixels)	Number of Clear Face(s) in Image	Average CPU Processing Time (ms)	Average GPU Processing Time (ms)
FaceNet	600 x 800	1	800.5	21.2
SphereFace	600 x 800	1	**768.7**	**15.8**
ArcFace	600 x 800	1	1002.3	34.5

In contrast to its high recognition accuracy, ArcFace is, in fact, the slowest method among the three. This is due to the more complex network architecture of the method. However, given the large advantage that it has in the recognition accuracy, the slower processing speed is acceptable as it is not significantly slower than the other two approaches. In conclusion, ArcFace is the best method to use for the face feature extraction module. And since ArcFace uses the similarity transformation method for face alignment, it also determines the face alignment module implementation.

4.3. Feature Identification Module

The processing speeds of this module on different database sizes are shown in Table 6. The module was tested on three datasets, Dataset1, half-size Dataset2, and AsiaFace-small. The processing times indicate the CPU times the module take to identify the received face feature(s) from the face feature extraction module.

Table 6 Processing speeds of identifying faces on different database sizes

Dataset	Number of images	Number of face(s) in a scene to identify	Total processing time (ms)
Dataset1-half	25,000	1	8.7
Dataset1-half	25,000	1	49.8
Dataset1	50,000	1	19.4
Dataset1	50,000	10	115.3
AsiaFace-small	400,000	1	133.9
AsiaFace-small	400,000	10	796.2

The results show that the method implemented for this module is sufficient to identify multiple face features from large-scale database. Since the method used here is a linear search method, therefore, the running speed is expected to be linearly increasing with database size.

Chapter 5.

Lapped CNN for Embedded Hardware

The results from the experiments in Chapter 4 show that the best configuration to construct a robust and accurate face recognition system is the combination of Hi3519-MTCNN, similarity transformation-based face alignment, ArcFace, and linear search method for feature identification. This pipeline can achieve high accuracy in practical daily scenes and can run fast on a face database with modest size. Although this pipeline can achieve impressive performance, it is very computationally intensive. Not only do the CNNs consume a lot of processing power, they also consume a lot of memories, which is limited in embedded hardware. In order to allow embedded hardware to execute these CNNs, the lapped CNN (LCNN) architecture was developed [19].

5.1. Architecture

The LCNN architecture follows the divide-and-conquer principle to decompose a large CNN on an input image into two or more tiers or stages. Each tier or stage is a small-scale CNN that operates on a low-resolution input image and have smaller memory footprints. In this thesis, a low cost embedded platform called Hi3519 from HiSilicon is used to illustrate the design and implementation of the LCNN architecture.

For the Hi3519 platform, there is a built-in CNN module that can execute small-scale CNNs through hardware acceleration. This CNN module accepts an input size of up to 1280 pixels. It can have one to eight convolutional layers and three to eight fully-connected layers. Each of the convolutional or fully-connected layer is followed by a ReLU layer, and The ReLU layer following the fully-connected layer is also followed by a dropout layer. The number of kernels in each convolutional layer is at most 50, and only 3 by 3 pixels convolution kernels are allowed. The convolution stride is fixed to be 1. The size of pooling layer is fixed to be 2 by 2 pixels, and the stride is fixed to 2. The maximum input dimension for the fully-connected part is 1024, and the number of neurons in the subsequent hidden layers is at most 256. The output dimension of the fully-connected part is at most 256. It is obvious that for such limited architecture, it is

hard to perform any tasks beyond simple ones such as hand-written digit classification due to the small input image size. However, with the LCNN architecture, the Hi3519's simple CNN module can be used to implement CNNs that operates on large images for complicated tasks such as image classification.

Assuming a CNN architecture is chosen for the Hi3519's hardware CNN module, such that it can perform challenging tasks on high-resolution image. In order to overcome the input size limitation of the hardware CNN module, the high-resolution image is first divided into small subimages, then each of the subimages is fed into the hardware CNN module sequentially for processing. The outputs resulting from these subimages are then merged. This merged result would be exactly identical to that of directly applying the CNN to the high-resolution image and can be used for further processing. In this thesis, a two-tier structure is used to demonstrate this process, they are denoted as CNN1 and CNN2, as shown in Figure 24, and this scheme can be generalized to more than two tiers.

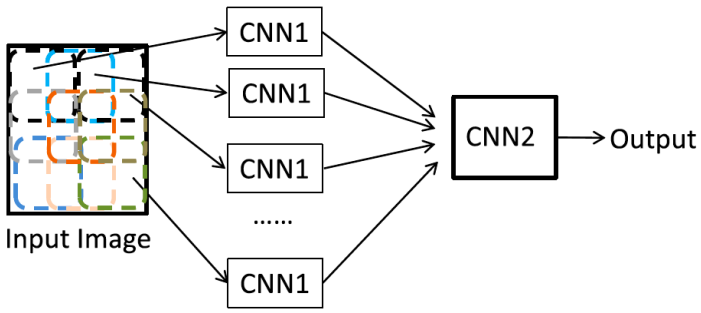


Figure 20 The architecture of LCNN scheme that can reuse simple hardware CNN module.

However, feeding the subimages independently can lead to the problem that features at the boundaries between subimages are not processed properly. This is similar to the blocking artifact in DCT-based image coding and can be resolved by lapped transform [51]. The boundary images overlapping two subimages can be fed into the CNN1, then the boundary-affected outputs from each subimage can be replaced by the correct ones from the boundary images. But this only works for convolutional layers, in order to achieve equivalent for both convolution and maxpooling operations between the large CNN outputs and the small CNNs outputs, two conditions must be met: 1. The boundary effect of convolution should be avoided. 2. The input image size to each

maxpooling operator in both CNNs is even. Using a three stages CNN as an example, it is clearer to show the usage of LCNN approach.

The three stages CNN consists of three convolutional layers with kernel size 3 by 3 pixel, and each follows by a maxpooling layer with 2 by 2 pixels of pooling window. Suppose K is the number of rows or columns of a subimage, after the first convolution, there will be K + 3 – 1 = K + 2 outputs, and only K – 2 of them are not affected by the boundary effect. In order to satisfy condition 2, K – 2 must be an even number and thus:

$$K - 2 = 2x_1 \Rightarrow K = 2 + 2x_1 \ , \tag{16}$$

where x_1 is a positive integer. After the first maxpooling, the size becomes $\frac{(K-2)}{2}$ and becomes $\frac{(K-2)}{2} - 2$ after the second convolution layer, similarly, the even condition requires that:

$$\frac{(K-2)}{2} - 2 = 2x_2 \Rightarrow K = 4x_2 + 6 \, , \tag{17}$$

where x_2 is a positive integer. The size becomes $\frac{(K-2)}{4} - 1$ after the second convolution and becomes $\frac{(K-2)}{4} - 3$ after the second maxpooling, even condition requires that:

$$\frac{(K-2)}{4} - 3 = 2x_3 \Rightarrow K = 8x_3 + 14 \, , \tag{18}$$

where x_3 is a positive integer.

Some solutions of K that satisfy the equations are 22, 30, 38, and 46. For the Hi3519 CNN module, since the input size is limited to 1280 pixels, the size 38 x 30 is picked for further illustration. It is noted that simply putting the 38 x 30 pixels subimages side by side can only form a large image with size 76 x 60 pixels. This input size does not satisfy the even condition and thus extra pixels must be added between neighboring subimages to make the combined image becomes a 78 x 62 pixels image. The last step is to determine the shifting pixels between the neighboring subimages. Since the last stage output of each 38 by 30 subimage is 3 by 2, a shifting distance is needed such that the outputs of two overlapped subimages are shifted by 3 vertically and by 2 horizontally. Since each maxpooling operation reduces the size by half and there are three maxpooling stages, therefore, each output in the final stage corresponds to 8 input

pixels. As a result, the shifting between the neighboring subimages should be 24 by 16 pixels, and the input image should adjust to the size of 86 by 62 pixels, where 86 = 32 + 24 x 2 and 62 = 30 + 16 x 2. This process is illustrated in Figure 25.

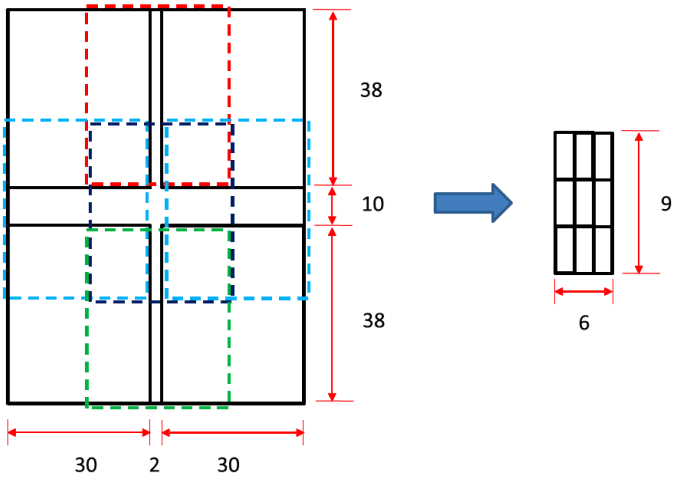


Figure 21 Illustration of full image and subimages configuration to use in LCNN.

5.2. Experiments

The impact of input image size on CNN models were on two tasks, image classification and age estimation. Two CNNs are presented, both accept a small input image size of 36 by 35 pixels and large input image size of 86 by 78 pixels through the LCNN approach. The first CNN N1 has the architecture shown in Table 7 (a) and the second CNN N2 has the architecture shown in Table 7 (b). For the small image size, the CNNs use the Hi3519's CNN module directly and use the LCNN approach to process the large image. For image classification, Tiny ImageNet, a subset of ILSVRC-2012 dataset [52], is used. For age group estimation, the Adiencedb dataset [53] is used.

Table 7 Network architectures of the two CNNs used for testing

(a)	N1 CNN	(b)	N2 CNN
Input	36 x 35	**Input**	86 x 78
Convolutional	Kernel Size: 3 x 3 Stride: 1 Channel: 24	**Convolutional**	Kernel Size: 3 x 3 Stride: 1 Channel: 32
Max-pooling	Pooling Size: 2 x 2 Stride: 2	**Max-pooling**	Pooling Size: 2 x 2 Stride: 2
Convolutional	Kernel Size: 3 x 3 Stride: 1 Channel: 48	**Convolutional**	Kernel Size: 3 x 3 Stride: 1 Channel: 48
Max-pooling	Pooling Size: 2 x 2 Stride: 2	**Max-pooling**	Pooling Size: 2 x 2 Stride: 2
Convolutional	Kernel Size: 3 x 3 Stride: 1 Channel: 48	**Convolutional**	Kernel Size: 3 x 3 Stride: 1 Channel: 48
Fully-connected	Outputs: 256	**Fully-connected**	Outputs: 512
Softmax layer	Outputs: 256	**Softmax layer**	Outputs: 512

For the Tiny ImageNet dataset, 200 classes are randomly selected from the ILSVRC-2012 dataset. For each class, 500 images are sampled as training images and 50 images are sampled as testing images. All images are resized to 90×90 after center crop. As a result, there are 100K training samples and 10K validation samples. The input images to the large CNNs with input size 86×78 are randomly cropped image from the size 90×90 images. To get input size 36×35, 80×80 images are first randomly cropped from the 90×90 images, then resize to 36×35. The testing accuracies are shown in Table

8. The results show that large input size can improve the Top-1 and Top-5 accuracies by about 30% and 15% respectively for the same architecture, and architecture with more parameters can achieve overall higher accuracy.

Table 8 Image classification accuracy of N1 and N2 networks with different input image sizes

Network	N1 (86 x 78)	N1 (36 x 35)	N2 (86 x 78)	N2 (36 x 35)
Top-1 Accuracy	36.25%	28.36%	**38.12%**	29.86%
Top-5 Accuracy	62.75%	54.37%	**64.24%**	55.58%

For the age estimation with the Adiencedb dataset, there are 8 age groups and 5 folders of images. 4 folders were randomly selected for training and 1 folder was left for testing. There are 11K images in the training set and 3K in the validation set. The testing results are shown is Table 9. Although the difference is not as much as the image classification task, but larger input image size can improve the estimation accuracy.

Table 9 Age group estimation accuracy of N1 and N2 networks with different input image sizes

Network	N1 (86 x 78)	N1 (36 x 35)	N2 (86 x 78)	N2 (36 x 35)
Accuracy	**55.27%**	49.34%	53.34%	48.43%

Chapter 6.

Discussion

6.1. Face Recognition Pipeline

The critical component that ensures the pipeline's robustness and accuracy is the face feature extraction module, which is ArcFace in this thesis. Therefore, it is beneficial to gain a better understanding of the reasons that made it outperforms the other two methods. There are three major reasons that the ArcFace method can achieve much better accuracy than the other two methods.

Easier Training

Since the FaceNet model is optimized on the entire dataset and cannot constraint on each individual sample, it requires carefully designed triplets during the training process, this is both time-consuming and performance-sensitive [28]. When there are millions of training data, this process becomes impractical and makes the training of FaceNet a hard problem. In contrast, both SphereFace and ArcFace add constraints to the individual sample and thus do not need to have heavy dataset processing work. However, training of SphereFace involves the process of tuning many hyper-parameters, which can be very time-consuming [28]. ArcFace reduces the number of hyper-parameters through feature normalization and constant introduction in the training process [29], and thus makes the training a much easier process.

Larger Decision Boundary in the Angular Domain

Since face features that are generated from FaceNet live in the Euclidean space, decision boundaries between feature clusters are heavily affected by the training process. The authors of SphereFace closely studied the training process and concluded that the Euclidean margin loss used in FaceNet was incompatible with softmax loss, which was also used in the FaceNet training [28]. This incompatibility may cause the resulting boundaries between face features being not optimal.

As a result, SphereFace and ArcFace use a better way to increase the feature distances by introducing angular margin in the angular domain. However, even though

SphereFace modified the original softmax loss to introduce such margin, the decision boundary still remains as a feature vector, while the modification of ArcFace changes the decision boundary into a marginal sector [29]. The difference between the decision boundaries makes features generated from ArcFace much more separated from each other. Figure 24 shows the decision boundaries of both methods.

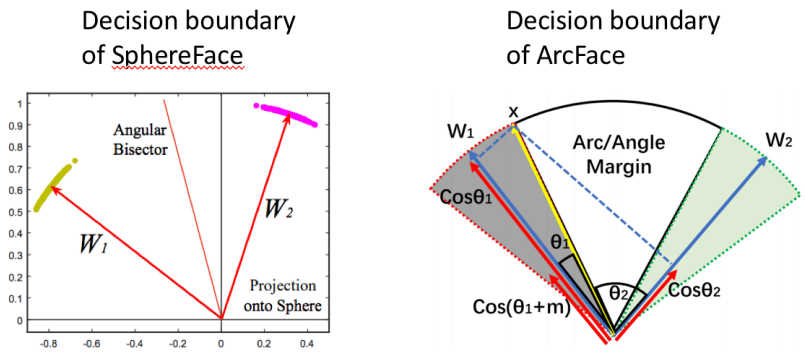


Figure 22 Decision boundaries of SphereFace and ArcFace. W1 and W2 are the weights corresponding to different classes. Left: the decision boundary of SphereFace remains as a feature vector. Righ: the decision boundary of the ArcFace is a marginal sector.

Better Network Architecture Design:

Both of the FaceNet and SphereFace took existing deep learning architectures for training their methods. In contrast, ArcFace authors carefully studied the existing architectures and modified them to better suited for their need. This helps to provide a more ideal architecture for the method to perform optimally.

In conclusion, although both the SphereFace and ArcFace optimized the distances between face features in the angular domain, there are still many differences between the two, and such differences made ArcFace a much better method than the SphereFace. This also explains the result observed in the experiments, in which ArcFace's accuracies are much higher than those of SphereFace.

6.2. Lapped CNN

From the results in Chapter 5, it is clear that with the LCNN architecture, it is possible to run large size CNN on resource-limited hardware, this opens up many new possibilities. One such possibility is to deploy the face recognition pipeline developed in

this thesis into embedded platforms. Since the MTCNN architecture is relatively simple for all of its networks, this makes it a suitable candidate for such purpose. However, it is trickier to deploy the ArcFace architecture into embedded hardware, due to its complicated network structures such as skip connection, residual blocks, etc. In order to allow the deployment of ArcFace into embedded hardware, the LCNN architecture must be extended to support complicate network structures.

Chapter 7.

Conclusion

The purpose of this thesis is to study the construction of a robust face recognition pipeline with deep learning techniques. A face recognition system can verify the identification of a person without his or her active cooperation, this reduces the cognitive load of the person and allows a more efficient way of identity verification or human-machine interaction. Since deep learning has achieved huge successes in many computer vision tasks. It could also improve the reliability and accuracy of such system.

A face recognition pipeline was constructed in the thesis with various deep learning methods. The pipeline consists of four components, the face detection module, the face alignment module, the metric space face feature extraction module, and the face feature identification module. For the face detection module, two deep learning-based implementations were experimented. Both of them employed a coarse-to-fine approach. The one that combined both the face classification and face localisation tasks into one network was found to achieved higher detection rate and lower computational time. As a result, it was chosen to be used in the pipeline. For the face alignment module, two methods were implemented, one method used provided face landmarks to crop out the face and put it into the center of the new image; the other method use similarity transformation to align the face to a reference set of face landmarks. Each method corresponds to different face feature extraction method in the later stage in the pipeline. For the metric space face feature extraction module, three deep learning methods were experimented, the FaceNet, SphereFace, and ArcFace. It was found that ArcFace achieved much higher accuracies on two private datasets than the other two methods. Although it has the slowest processing time, the advantage that it has in the accuracies is more significant. Therefore, ArcFace was chosen to be used in the pipeline. Since ArcFace relies on similarity transformation method for face alignment, the alignment method was also decided. Finally, the face feature identification module was implemented through a linear search algorithm. The processing speed of this implementation was measured, and it was sufficient to perform real-time feature identification on a modest size dataset in real-time.

The resaons that ArcFace was so much more accurate than the other two methods were also discussed. The high accuracy is a result of three aspects, easier training, larger decision boundaries, and better designed network architecture.

The resulting face recognition pipeline can capture 85% of the faces in a dataset with a very low false positive rate of 35 faces in real-time. If the system was to be used in a surveillance scene such as train station monitoring, it can identify all of the captured faces with an accuracy of 83% in real-time on a 25,000-faces database. The high accuracies in both face detection and identification make it a reliable face recognition system to use in practices.

Based on the experimental results and discussion, it is clear that a robust face recognition pipeline can be constructed with deep learning techniques and achieve state-of-the-art performance. However, many improvements and future work can still be done. There is still room for the face detection and face feature extraction module to improve. Besides that, the face feature identification module is too slow on a very large-scale database that is in the magnitude of millions. A faster search algorithm is needed for practical usage on such database.

Finally, a novel CNN architecture called lapped CNN (LCNN) was developed and studied. The goal of the LCNN is to allow large size CNN to run on resource-limited embedded hardware. LCNN achieve this by splitting up the large CNN into multiple smaller CNNs that can be run sequentially or parallelly on the embedded hardware. Experiments on the Tiny ImageNet and Adiencedb datasets confirm the performance gain by allowing larger CNN to run on embedded hardware. Future work includes extending the LCNN architecture to support more complicated network architectures, such that the face recognition pipeline can be deployed and run on embedded hardware.

References

[1] M. J. Sheehan and M. W. Nachman, “Morphological and population genomic evidence that human faces have evolved to signal individual identity,” *Nat. Commun.*, vol. 5, p. 4800, Sep. 2014.

[2] I. Marqués and M. Graña, “Face processing for security: A short review,” in *Advances in Intelligent and Soft Computing*, 2010, vol. 85, pp. 89–96.

[3] M. Turk, “Eigenfaces and Beyond,” *Acad. Press*, pp. 1–29, 2005.

[4] M. Slavković and D. Jevtić, “Face Recognition Using Eigenface Approach*,” *SERBIAN J. Electr. Eng.*, vol. 9, no. 1, pp. 121–130, 2012.

[5] Chengjun Liu and H. Wechsler, “Gabor feature based classification using the enhanced fisher linear discriminant model for face recognition,” *IEEE Trans. Image Process.*, vol. 11, no. 4, pp. 467–476, Apr. 2002.

[6] H. Cho, R. Roberts, B. Jung, O. Choi, and S. Moon, “An Efficient Hybrid Face Recognition Algorithm Using PCA and GABOR Wavelets,” *Int. J. Adv. Robot. Syst.*, vol. 11, no. 4, p. 59, Apr. 2014.

[7] “FERET database - Wikipedia.” [Online]. Available: https://en.wikipedia.org/wiki/FERET_database. [Accessed: 01-Apr-2018].

[8] C. Shu, X. Ding, and C. Fang, “Histogram of the oriented gradient for face recognition,” *Tsinghua Sci. Technol.*, vol. 16, no. 2, pp. 216–224, 2011.

[9] A. Rahim, N. Hossain, T. Wahid, and S. Azam, “Face Recognition using Local Binary Patterns (LBP),” *Glob. J. Comput. Sience Technol. Graph. Vis.*, vol. 13, no. 4, pp. 469–481, 2013.

[10] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[11] A. Ivakhnenko, *Cybernetic predicting devices,*. New York: CCM Information Corp., 1965.

[12] K. Fukushima, “Biological Cybernetics Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position,” *Biol. Cybern.*, vol. 36, no. 193, 1980.

[13] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation Applied to Handwritten Zip Code Recognition,” *Neural Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied

to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[15] G. E. Hinton, "Learning multiple layers of representation," *Trends in Cognitive Sciences*, vol. 11, no. 10. pp. 428–434, 2007.

[16] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.

[17] "Nvidia CEO bets big on deep learning and VR | VentureBeat." [Online]. Available: https://venturebeat.com/2016/04/05/nvidia-ceo-bets-big-on-deep-learning-and-vr/. [Accessed: 04-Apr-2018].

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Adv. Neural Inf. Process. Syst.*, pp. 1–9, 2012.

[19] X. Wang, H. W. Ng, and J. Liang, "Lapped Convolutional Neural Networks for Embedded Systems," *GlobalSIP*, pp. 1135–1139, 2017.

[20] X. Wang, H. W. Ng, and J. Liang, "Convolutional Neural Network (CNN) System Based on Resolution-Limited Small Scale CNN Modules."

[21] X. Wang, M. Seyfi, M. CHen, H. W. Ng, and J. Liang, "Face Detection Using Small-Scale Convolutional Neural Network (CNN) Modules for Embedded Systems."

[22] X. Wang, M. Seyfi, M. Chen, H. W. Ng, and J. Liang, "Joint Face-Detection and Head-Pose-Angle-Estimation Using Small-Scale Convolutional Neural Network (CNN) Modules for Embedded Systems."

[23] X. Wang, M. Seyfi, M. Chen, H. W. Ng, and J. Liang, "Age and Gender Estimation Using Small-Scale Convolutional Neural Network (CNN) Modules for Embedded Systems."

[24] H. W. Ng, X. Wang, Y. Gao, R. Ma, and Y. Lu, "Enhanced Face-Detection and Face-Tracking for Resource-Limited Embedded Vision Systems."

[25] K. Zhang, Z. Zhang, Z. Li, S. Member, Y. Qiao, and S. Member, "Joint Face Detection and Alignment using Multi - task Cascaded Convolutional Networks," *Spl*, no. 1, pp. 1–5, Apr. 2016.

[26] G. Li, Haoxiang and Lin, Zhe and Shen, Xiaohui and Brandt, Jonathan and Hua, "A Convolutional Neural Network Approach for Face Detection," *Cvpr*, pp. 5325–5334, 2015.

[27] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015, vol. 07–12–June, pp. 815–823.

[28] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, “SphereFace: Deep hypersphere embedding for face recognition,” in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017, vol. 2017–Janua, pp. 6738–6746.

[29] J. Deng, J. Guo, and S. Zafeiriou, “ArcFace: Additive Angular Margin Loss for Deep Face Recognition,” Jan. 2018.

[30] M. Seyfi, X. Wang, M. Chen, K. Wang, W. Wang, H. W. Ng, J. Zheng, and J. Liang, “Method and Apparatus for Real-Time-Face-Tracking and Face-Pose-Selection on Embedded Vision Systems.”

[31] T. M. Mitchell, *Machine Learning*. McGraw-Hill, Inc., 1997.

[32] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., 1995.

[33] F. Rosenblatt, “THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN,” *Psychol. Rev.*, vol. 65, no. 6, pp. 19–8.

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[35] “Spatial convolution.” [Online]. Available: https://graphics.stanford.edu/courses/cs178/applets/convolution.html. [Accessed: 04-May-2018].

[36] P. Viola, “The Viola / Jones Face Detector Classifier is Learned from Labeled Data,” *Training*, 2001.

[37] L. R. dkk Cerna, “Face Detection: Histogram of Oriented Gradients and Bag of Feature Method,” *Int. Conf. Image Process.*, pp. 657–661, 2013.

[38] S. Transformation and W. Mathworld, “Similarity transformation,” pp. 1–6, 2009.

[39] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” 2016.

[40] A. Neubeck and L. Van Gool, “Efficient non-maximum suppression,” in *Proceedings - International Conference on Pattern Recognition*, 2006, vol. 3, pp. 850–855.

[41] “Histogram of Oriented Gradients and Object Detection - PyImageSearch.” [Online]. Available: https://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/. [Accessed: 04-May-2018].

[42] S. Yang, P. Luo, C. C. Loy, and X. Tang, “WIDER FACE: A face detection benchmark,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, vol. 2016–Decem, pp. 5525–5533.

[43] Z. Liu, P. Luo, X. Wang, and X. Tang, “Deep learning face attributes in the wild,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, vol. 2015 Inter, pp. 3730–3738.

[44] “dlib C++ Library.” [Online]. Available: http://dlib.net/.

[45] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning,” 2016.

[46] “Face recognition using Tensorflow.” [Online]. Available: https://github.com/davidsandberg/facenet. [Accessed: 03-May-2018].

[47] “CS231n Convolutional Neural Networks for Visual Recognition.” [Online]. Available: http://cs231n.github.io/linear-classify/#softmax. [Accessed: 04-May-2018].

[48] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, “VGGFace2: A dataset for recognising faces across pose and age,” Oct. 2017.

[49] V. Jain and E. Learned-Miller, “FDDB : A Benchmark for Face Detection in Unconstrained Settings,” -, p. UM-CS-2010-009, 2010.

[50] J. Yan, X. Zhang, Z. Lei, and S. Z. Li, “Face detection by structural models,” *Image Vis. Comput.*, vol. 32, no. 10, pp. 790–799, 2014.

[51] T. D. Tran, Jie Liang, and Chengjie Tu, “Lapped transform via time-domain pre- and post-filtering,” *IEEE Trans. Signal Process.*, vol. 51, no. 6, pp. 1557–1571, Jun. 2003.

[52] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[53] E. Eidinger, R. Enbar, and T. Hassner, “Age and Gender Estimation of Unfiltered Faces,” *IEEE Trans. Inf. Forensics Secur.*, vol. 9, no. 12, pp. 2170–2179, Dec. 2014.