UNIVERSITY OF SCIENCE AND TECHONOLOGY OF HANOI

BACHELOR THESIS

Deep face recognition system

---

by NGUYEN Lan Huong

Information and Communication Technology major
Academic year 2018 - 2021

Supervisor:

NGUYEN Tuan Anh - Aimesoft JSC

Declaration

I, NGUYEN Lan Huong, declare that the work in this thesis is entirely my own under the guidance of my supervisor NGUYEN Tuan Anh and that it has never been previously published. All methods, comments, and statistics referenced from other authors and organizations' work have been acknowledged and indicated clearly. If any fraud is found, I will take full responsibility and happily accept penalty from the thesis defense committee and my university.

Hanoi, July 2021

NGUYEN Lan Huong

Acknowledgement

First and foremost, I would like to express my sincere gratitude toward my supervisor NGUYEN Tuan Anh for having offered me an internship position and guided me throughout the thesis writing process. I would also like to thank my colleagues at Aimesoft JSC, especially PHAM Huu Bien and PHAM Quang Nhat Minh, for providing their insight into the problems I aimed to address.

My appreciation also extends to Dr. DOAN Nhat Quang, my internal supervisor at the university for giving me helpful advices during the thesis writing and defending process.

I would also like to show my dearest cherish to my best friend, VU Dinh Anh. I could not have finished this thesis without his warming emotional support and significant help in solving the technical issues I encountered.

Last but not least, I thank all family members, friends and the Information and Communication faculty at USTH who have supported me during the past academic years.

Abstract

A face recognition system is one that can recognize identity of a human face from an image or video frame. Deep learning methods i.e. convolution neural networks have dramatically improved the accuracy of this technology over the past years, replacing traditional methods such as Eigenface [1] or Gabor feature [2]. In this thesis project, we aim to develop a simple face recognition system with a deep learning backbone. First, we build and train a deep learning model called ResNet with margin-based softmax cross-entropy loss. Next, we implement the system by combining the trained model with various components and algorithms. The result system has about 80% accuracy when dealing with 2000 to 5000 identities in the database with only a 2% rate of mispredicting as another registered person. It also shows 90% accuracy when presented with a stranger. This system is qualified to be used as an identification application for users or employees at a small to medium scale company.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context and motivation

Face recognition is a technology that can extract and analyze a human face from an image or video frame and compare it against a face database. One of the most famous application of this technology is helping law enforcement agencies identify suspects from surveillance videos. It can also be employed by businesses as an identity authentication method for employees or users. Although its performance is much less reliable than iris or fingerprint recognition, face recognition is still universally favourable for its contactless approach.

Since the 1970s, numerous face recognition methods have been proposed as is shown in figure 1.1, most of which utilize just a few layers for data representation. A milestone in facial feature learning is the Eigenface approach [1], where dimensionality reduction is performed on the probability distribution of face images in a training set. This allows for any face to be represented by a combination of a few Eigenfaces (Figure 1.2) and classified based on the combination's coefficients. The 2000s witnessed a rise in local feature or texture analysis methods such as Gabor filter [2] and local binary patterns [3]. These methods are further improved by changing handcrafted features to learning-based ones [4], [5], [6].

Nonetheless, traditional face matching methods had made slow progress and proved to be prone to failure when dealing with variance in facial expressions, poses, lighting, etc. With the appearance of convolution neural networks, namely AlexNet [8], VGG [9] and ResNet [10], face verification tasks have witnessed an unprecedented accuracy level of above 99 percent, according to a survey in 2020 [11]. The superiority of this approach over traditional ones is that it is capable of representing faces with a self-learning and multi-level hierarchy of abstract representations instead of just a few layers.

Despite those great advancements, face recognition systems still have to overcome numerous challenges, the biggest of which is the enormous number of unseen faces. This means a deep learning model must be well generalized in order to distinguish faces

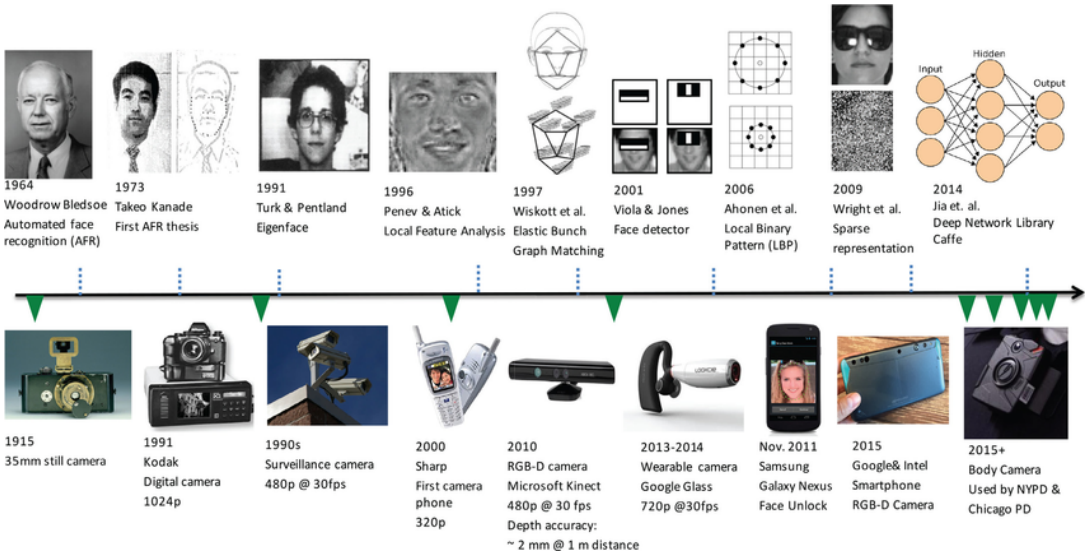

Figure 1.1: Major milestones in the history of automated face recognition [7]

Figure 1.2: Linear combination of Eigenfaces [1]

out of the training data set. Moreover, it must be capable of dealing with diverse changes in age, make up, glasses and so on. Finally, each model requires to be optimized differently according to its goals, for example, a system can be specialized in recognizing Asian faces or faces with masks.

1.2 Expected outcome

The outcome of this thesis project is a simple face recognition system. The system provides an interface for a new user to register their face by providing several reference images. Later on, when the system is presented with an unseen image, it can identify it as belonging a registered person or a stranger.

Since the main use case for the system is to recognize employees or customers of a

company, we expect it to achieve at least 80% accuracy recognizing 2000 different registered identities. It should also show a low rate of mispredicting a registered as another registered because it is less harmful to mispredict them as a stranger.

1.3 Objective

In this thesis, a survey about existing deep learning models and related methods for face recognition tasks is first conducted. We then implement the system with a neural network backbone and experiment with various methods. Their effects on the performance of the system are evaluated at the end.

1.4 Thesis organization

Chapter 2 provides an overview about our approach. Chapter 3 and 4 describes in detail the two development phases, deep learning model development and system implementation relatively. Chapter 5 presents the result and analyzes the system's performance. Chapter 6 concludes the thesis and discusses future work.

Chapter 2

General approach

2.1 Face recognition overview

There are myriad existing face recognition systems but they generally tend to operate in 4 main steps. Our system also follows this frame.

1. **Face detection:** Human faces are located in an image or a video frame. The face is then cropped and normalized according to the detected facial landmarks.
2. **Feature extraction:** Distinguishing factors are extracted from detected faces. These factors can include the shape of the eyes, the contour of the cheekbones, the size of the nose etc.
3. **Feature encoding:** Extracted features are analyzed and then encoded into numerical values, which act like a unique fingerprint for the human face. This step reduces large dimensions of facial features to a compact embedding vector.
4. **Face matching:** An unknown embedding is compared against a database of different face prints. If a match is found, the identity of the unknown is determined, otherwise it is concluded a stranger.

2.2 Development process

A face recognition system can be composed of numerous parts but the most important one is the feature extracting and encoding engine. The performance of the system largely depends on this component and thus, the technology behind it requires extensive research and development time.

After the main face analysis technology has been finished, other components are built around it in order to achieve a functioning system. These components include the pre-processor that does face detection and normalization, the database that stores face prints of all identities and an identifier that compares and matches face prints etc.

As a result, the timeline of thesis project is divided into 2 phases.

1. **Deep learning model development phase:** We build and train a neural network to extract and encode human facial features from images.
2. **System implementation phase:** The best model from the previous phase and other components are assembled into a complete system. These components employ specialized algorithms that contribute to the face recognition process.

Chapter 3

Phase 1: Deep learning model development

This chapter tries to solve a multi-class classification problem where each class is a different human identity using a neural network. Through learning to correctly classify all faces in a large and diverse data set, the deep learning model becomes increasingly sensitive to characteristics of the human face and eventually can differentiate faces it has never seen before. The steps for solving this problem can be summarized as:

1. **Material preparation:** Collect and pre-process a sizable and labeled data sets of face images for training and validation.
2. **Model construction:** Survey and choose a suitable neural network architecture for extracting and classifying information in images.
3. **Model optimization:** Iterate through the training data set and optimize the model's parameters using a specialized loss function. Validate the model after each training period.
4. **Model checkpointing:** Continue the training process until validation results converge. Save the model with that set of parameters.

3.1 Theoretical background

3.1.1 Neural network definition

According to François Chollet in the book *Deep learning with Python* [12], a neural network is a way to model complex, non-linearly related data as consecutive layers of increasingly meaningful representations. This data model can be used for purposes such as classification, prediction, etc.

Figure 3.1 illustrates an example neural network built to extract features from an image and classify it as one of the 10 digits from 0 to 9. The network transforms an image of

the number 4 into layers, each represents a different characteristic of it. The latter the layer, the more indicative and insightful it is about the final result.

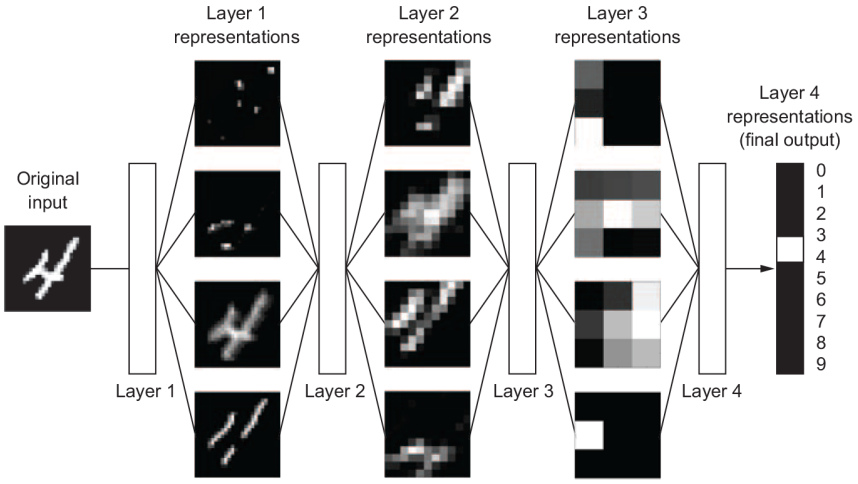

Figure 3.1: Deep representations learned by a digit-classification model [12]

3.1.2 Deep learning layers

Depending on its purpose, a layer in a neural network performs a particular set of mathematical manipulations on the input array and return another one, which can be fed into the next layer and so on. The most basic layers are the input and output layer, anything between them is called a hidden layer (Figure 3.2). Although there are numerous types of hidden layers, only two has been used in this thesis project, namely fully connected and convolutional.

Fully connected layer

A dense or fully connected layer is typically used for embedding or classification purpose. Each of its units is a weighted sum of the previous layer's units. It transforms an n-dimensional input vector X_n into an m-dimensional output vector H_m using a set of weights $W_{m\times n}$ and biases B_n, which can be written as the following matrix operations:

$$\begin{bmatrix} h_1 \\ h_2 \\ \dots \\ h_m \end{bmatrix} \leftarrow \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & \dots & w_{2,n} \\ \dots & & & \\ w_{m,1} & w_{m,2} & \dots & w_{m,n} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_m \end{bmatrix} \tag{3.1}$$

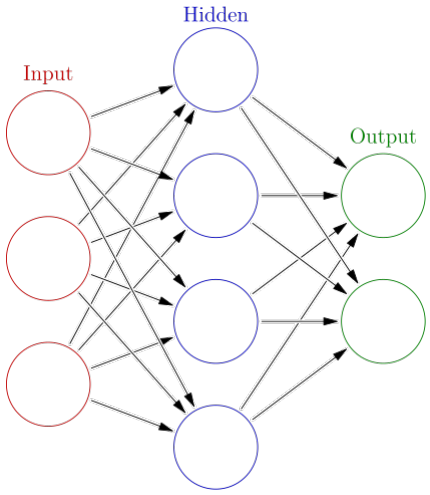

Figure 3.2: Layers of a neural network. [13]

or in short:

$$H \leftarrow W \cdot X + B \tag{3.2}$$

Convolutional layer

Convolutional layers are usually stacked onto each other to extract features and patterns from an image input. As is shown in figure 3.3, the first layers usually captures low-level or abstract features such as edges and lines. The deeper the layer, the more specific are the filters to the training data set.

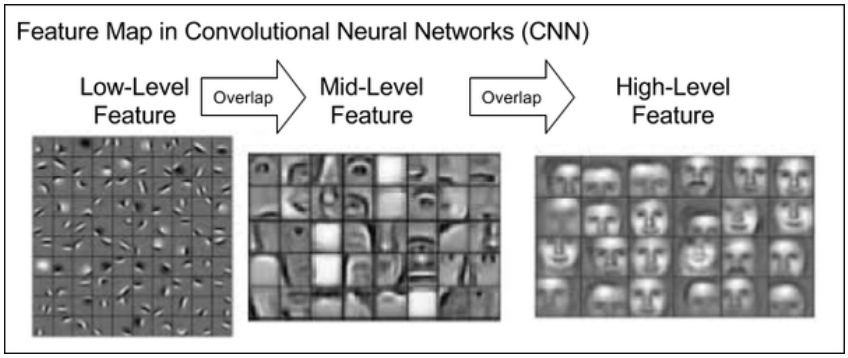

Figure 3.3: Features extracted by convolutional layers [14]

This type of layer takes a tensor of shape (n, c, w, h) as input, where n is the number of images in a batch, c is the number of channels of each image, w and h are the width and height. If the input has more than one color channel, the convolutional operation is performed on each channel separately. The layer passes an image through multiple

filters and and returns exactly that many new filtered channels. A filter is a rectangle window H of size $m \times n$ that is slid across the original image, each pixel of which is multiplied with the pixel of the image part lying underneath. The convolutional operation can be written as:

$$u_{ij} \leftarrow \sum_{p=0}^{m-1} \sum_{q=0}^{n-1} x_{si+p,sj+q} h_{pq} \tag{3.3}$$

where u_{ij} is the new pixel at the position i, j; m and n are the size of the rectangle filter, s is the stride with which the filter is slid across the image, h_{pq} is the pixel of the filter at the position p, q and $x_{si+p,sj+q}$ is the original image's pixel underneath it.

3.1.3 Model optimization concepts

Loss function

Every layer in a neural network has its own set of parameters, for example weights and biases in a fully connected layer or filters in a convolutional layer. However, it is impossible to calculate the perfect values for those parameters, the only way to derive them is through trial and error until a set of values that yield the best possible results is found.

A cost or loss function acts as guidance for the optimization process of a neural network; its parameters are updated gradually so that the value of that function is minimized. The loss is carefully crafted to address specific optimization goals. To highlight the importance of its design, the book *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks* [15] stated that:

"The cost function reduces all the various good and bad aspects of a possibly complex system down to a single number, a scalar value, which allows candidate solutions to be ranked and compared."

Gradient descent and back-propagation

Weights of neural network are optimized with gradient descent. The algorithm finds the gradient vector i.e. the 0 derivative of the loss function and gradually updates the model's parameters along that slope, guiding its value towards the minimum. The process is iterated over batches of data until the the loss or validation results converge. The gradient is back-propagated to update layers' parameters back to front.

Despite its effectiveness, this optimization algorithm still faces difficult problems such as vanishing gradients. This usually happens in big networks with many staked layers, where the gradient is progressively small toward the front, hence prevents the update of weight values. Another problem is degradation or accuracy saturation, where adding more layers to a suitably deep model leads to higher training error.

3.2 Materials

3.2.1 Data sets

Training data set

The model is trained using the public MS-Celeb-1M data set [16], which originally contains one million images of 100,000 celebrities. The identities come from fairly diverse range of nationalities and ages, despite a presenting imbalance between males and females. The InsightFace [17] project provides a washed up version of the data set, eliminating mislabeled images. After the clean up, there remains about 85,742 identities, each has from 2 to 600 images.

Validation data set

The model's parameters are selected using the validation results on three different data sets during the training process, all three are popular benchmarks for face verification and identification. The validation task is verifying if two images are the same person, which requires data sets to be split into same and not-same pairs.

The first data set is Labeled Faces in the Wild (LFW) [18], containing 13,233 images of 5,749 individuals, where 1,680 subjects are depicted in two or more images and the rest appear only in one image. The second one is Celebrities in Frontal-Profile in the Wild (CFP-FO) [19], containing 10 frontal and 4 profile images of 500 individuals. The last one is AgeDB-30 [20], containing 16,488 images of people of various ages from 1 to 101; the average number of images per subject is 29.

3.2.2 Pre-processing

This step of pre-processing eliminates irrelevant information such as background, outfit, hairstyle and the variance of camera angle to provide standardized inputs for the deep learning model. Firstly, face detection is performed by RetinaFace [21], a robust single-stage and state-of-the-art face detector. This model receives an image and returns a bounding box around any detected face, along with five facial landmarks annotating the eyes, the nose tip and mouth corners (second image, figure 3.4). Following the paper [22], those landmarks and the bounding box are used to crop and align the image (third image, figure 3.4) with similarity transformation. Next, the image is resized to 256×256 pixels to fit with the deep learning model's input shape. Finally, each pixel is normalized by subtracting 127.5 and then dividing by 128.

Data augmentation is also performed in order to add more diversity. The list of augmentation operations include brightness, contrast, saturation and color augmentation, mirroring and compression.

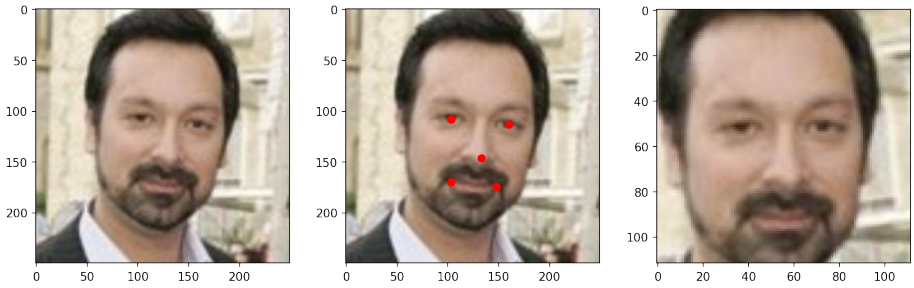

Figure 3.4: Original image - Detected facial landmarks - Cropped and aligned face.

3.3 Deep learning model architecture

Similar to myriad image classification models, our neural network is comprised of three main parts: the feature extractor, the embedding layer and the classifier. This section will explain in detail each component.

First, a convolutional backbone called ResNet is used as the feature extractor that learns characteristics of the human face. Ensuing it is an embedding layer that encodes those extracted features to a compact vector and a final layer that classifies those vectors.

3.3.1 Feature extraction backbone

Related architectures

Two of the most popular feature extracting architectures for image classification problems are VGG [9] and Inception[23].

VGG [9], introduced in 2014, simply stacks 3×3 convolution layers on top of each other in combination with max pooling to increase depth and reduce volume size (Figures 3.5). It was once the state-of-the-art architecture and is still widely adopted. However, its considerable training time and size has proven to be disadvantageous.

Inception [23] net uses a module that computes 1×1, 3×3 and 5×5 convolutions and then feed into the next layer a combination of their outputs (Figure 3.6). The superiority of this architecture is the ability to extract multi-level features while maintaining a relatively small size. Nonetheless, its complexity may cause difficulties during implementation.

ResNet architecture

In this project, we choose ResNet [10] as the main feature extractor for the following reasons. Firstly, it already has multiple existing implementations due to its simplicity and popularity. Secondly, ResNet is capable of achieving state-of-the-art performance while still requiring a relatively small number of parameters. According to an analysis

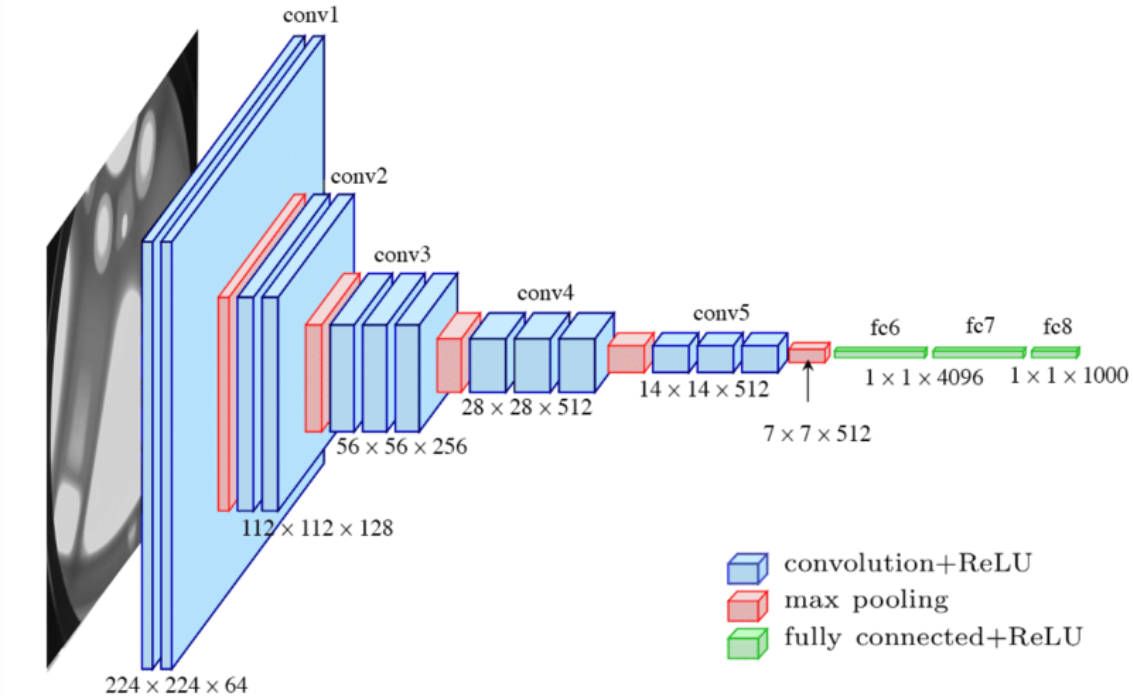

Figure 3.5: The standard VGG-16 architecture [9]

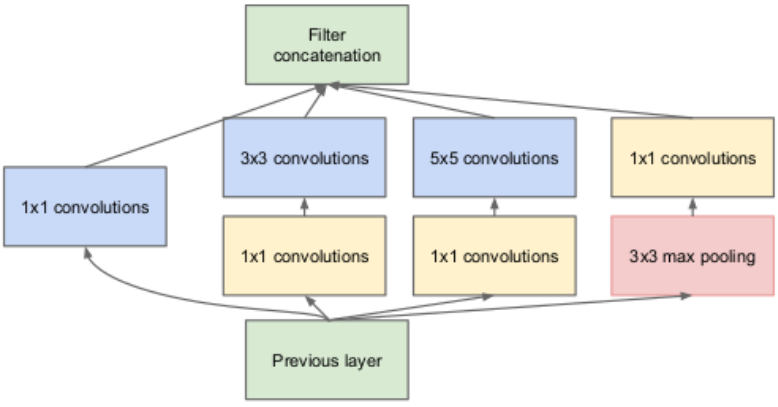

Figure 3.6: An inception module [23]

in 2017 [24], an average-sized ResNet-101 has the top-3 highest accuracy among a total of 14 surveyed architectures (Figure 3.7).

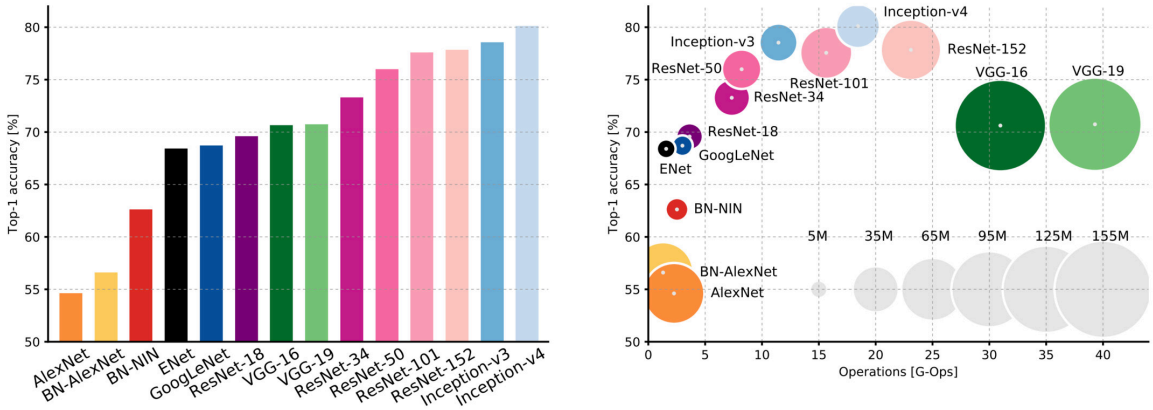

Figure 3.7: Left: Top 1 accuracy vs. network. Right: Top 1 accuracy vs number of operations and parameters size [24]

In 2015, the paper *Deep Residual Learning for Image Recognition* [10] introduced skip connections, or shortcuts to jump over some layers in a convolutional neural network. Figure 3.8 simplifies the structure of a residual block described by the paper. The input x is first passed through some stacked non-linear layers and then the output $F(x)$ is added to the original input element-wise to return the final mapping $H(x)$.

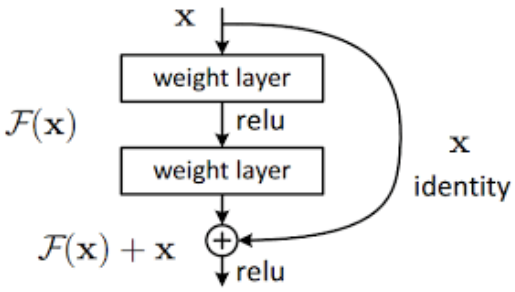

Figure 3.8: Mapping function of a residual block [10]

In a plain convolutional network such as VGG [9], stacked layers directly learn the desired mapping $H(x)$ from input x. A residual block, on the other hand, learns the difference between x and $H(x)$ instead. The paper hypothesizes this difference to be

less difficult to learn, stating that: "To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers."

The most important feature of the proposed skip connection is that it allows the gradient to flow backwards without encountering any weight layer and thus, not become too small. As a result, gradients can reach the initial layers and help them learn the correct weights.

Not only being an efficient work around for the vanishing gradient problem, residual blocks can also overcome the degradation of accuracy saturation problem. Figure 3.9 shows that as the depth of plain networks increases, so does its training error. ResNet, on the other hand, actually gains accuracy with depth. Consequently, ResNet architecture is capable of learning deeper features than plainly stacked networks, expanding up to above 100 convolutional layers.

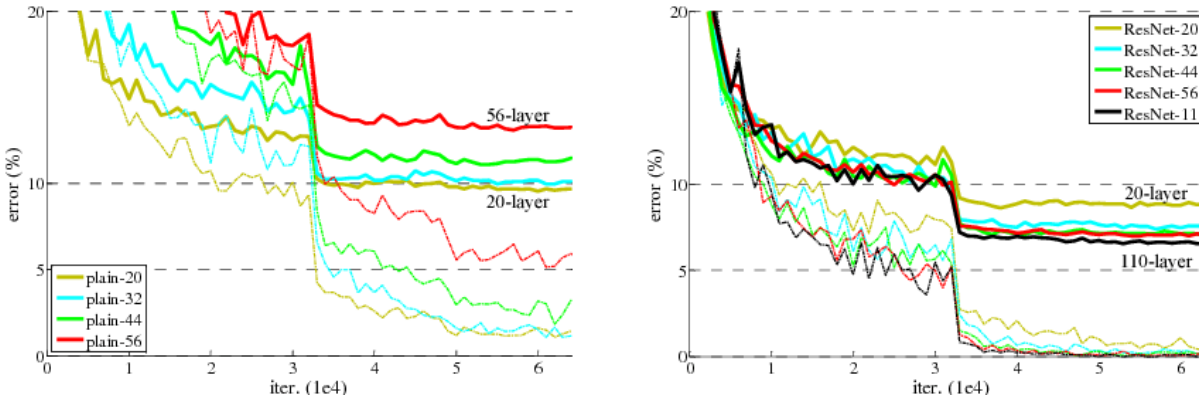

Figure 3.9: Training error on CIFAR-10 of plain and residual networks [10]

Figure 3.10 demonstrates ResNet architectures in different sizes. All start with a 64-filter convolutional layer with a 7×7 kernel and stride 2, followed by max pooling to reduce the size by half. The rest of the network consists of stacked residual blocks of decreasing size and increasing depth.

A residual block first uses 1×1 convolution, replacing pooling as the downsampling layer; then a 3×3 and a 1×1 convolution. The skip connection goes from the first to the last layer of each block. In between layers, one can add batch normalization of drop out to improve the training quality and avoid overfitting.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3.10: Various ResNet sizes [10]

3.3.2 Embedding and classification layer

Embedding layer

Extracted facial features are next compressed into a compact embedding vector and normalized by dividing each component by the Euclidean norm (Equation 3.4) so that its magnitude becomes 1. This means all embedding vectors lie on a hypersphere with the radius of 1.

$$x_i \leftarrow \frac{x_i}{\sqrt{\sum_{j=1}^{D} x_j^2}} \tag{3.4}$$

By encoding multi-dimensional facial features into vectors, abstract relationships between them are interpreted as spatial distance for better visualization and comparison. More specifically, vectors that represent similar faces are close together while different faces are far away. Figure 3.11 illustrates one among many complex relationships encoded in a latent vector space. The two women at the top left have close embeddings but both of theirs are far from those of the two men on the bottom right. The man on the bottom left's embedding is equidistant to the two previous groups because his face is sort of a hybrid of both.

The embedding vector space's dimension D must be large enough so that there is sufficient distance between different classes but not so large that the layer requires too many parameters and may overfit. Figure 3.12 shows that the the 512-dimensional space is the suitable size since the mean of the its nearest angles decreases slowly when the class number increases exponentially. This vector space is theoretically capable of containing up to 100 million face profiles.

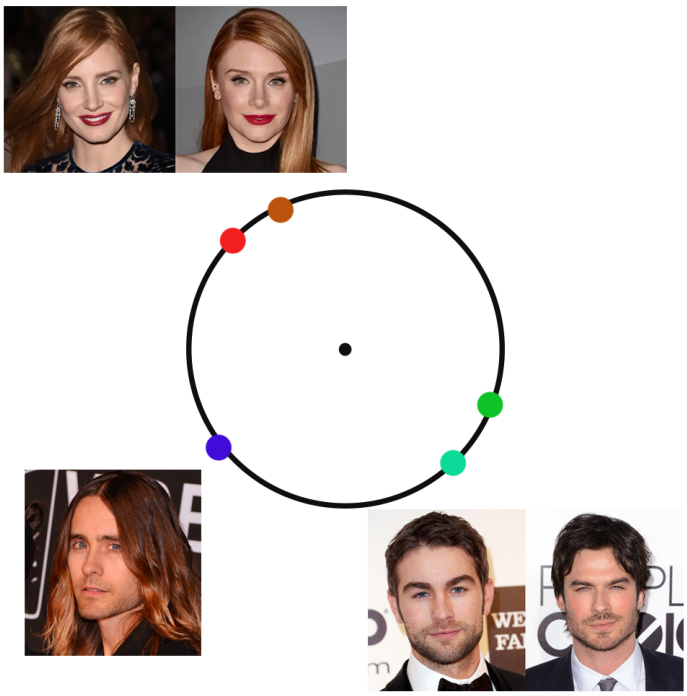

Figure 3.11: Simple illustration of a latent vector space

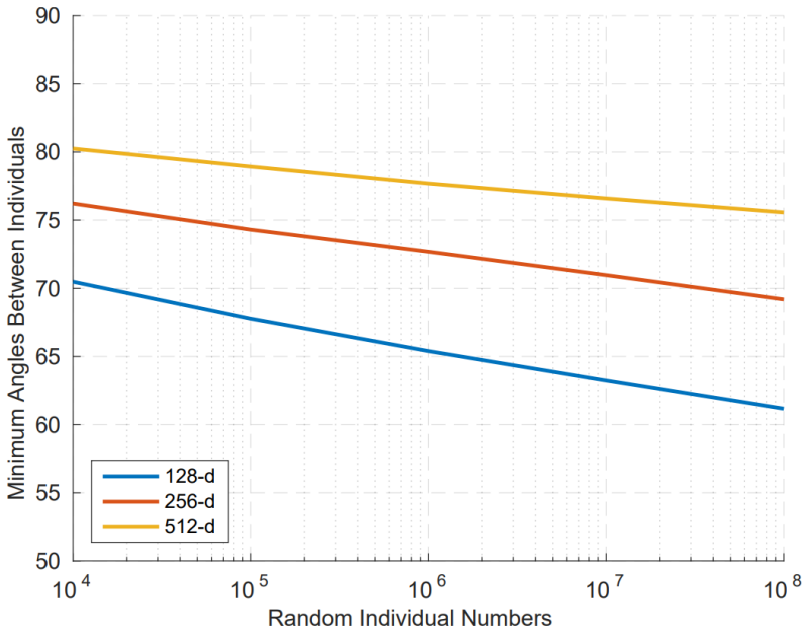

Figure 3.12: Comparison between dimensional space and number of classes [25]

A 512-unit fully connected layer is therefore employed to encode features into a 512-D embedding vector. We first flatten them and then use the sequence of layers: batch normalization - drop out - fully connected - batch normalization.

Classification layer

Finally, the embedding layer is follow by a fully connected layer with n units corresponding to n identities in the data set. After transforming the input using equation 3.1, the layer is passed through a softmax activation function, normalizing it to a probability distribution over n classes. In other words, the value of each unit now represents the predicted probability of belonging each class (Equation 3.5) and the model is trained to maximize the probability of belonging to the correct class.

$$P_t = \frac{e^{w_t x + b_t}}{\sum_{j=1}^{n} e^{w_j x + b_j}} \tag{3.5}$$

where P_t is the probability of the sample belonging to class t, x is the input to the layer, w_t and b_t is the weight and bias of the t^{th} unit corresponding to class t.

3.4 Model optimization

There are numerous existing losses designed for face image classification tasks, including softmax cross-entropy, triplet, costellation etc. However, we decided to employ a state-of-the-art loss function call margin based softmax cross-entropy loss, this section will explain its superiority over others.

3.4.1 Related losses

Softmax cross-entropy loss

Softmax cross-entropy is the most widely used loss function in classification problem. After activating the last classification layer with the softmax function (Equation 3.5), the loss is computed as:

$$L_{softmax} = -\frac{1}{M} \sum_{i=1}^{M} log(P_{y_i}) \tag{3.6}$$

where M is the total number of samples used to train, y_i is the i_{th} sample's correct class, P_{y_i} is the softmax-activated value of the unit representing that class. By minimizing this loss function, the predicted probability of the correct class is maximized.

Combining that with equation 3.5, we have:

$$L_{softmax} = -\frac{1}{M}\sum_{i=1}^{M} log\frac{e^{w_{y_i}x_i+b_{y_i}}}{\sum_{j=1}^{n} e^{w_j x_i+b_j}} \tag{3.7}$$

It did not take long for researchers to realize that softmax cross-entropy alone cannot sufficiently discriminate between embedding features. More specifically, it is incapable of decreasing the intra-distance between features belonging to the same person and increase the inter-difference between different people. As more novel loss functions are explored, the generalization ability of face recognition models are finally enhanced.

Intra and inter loss

A naive solution to the defect of softmax cross-entropy loss is simply adding one or both of the two following functions to it. Intra loss (Equation 3.8) tries to minimize the angle between the embedding vector and its corresponding facial center w_{y_i}. On the contrary, inter loss (Equation 3.9) tries to maximize the angle between different facial centers w. Nevertheless, using one of the above losses cancels the effect of the other and therefore, they do not make much improvement when combined.

$$L_{intra} = \frac{1}{\pi N}\sum_{i=1}^{N} \theta_{y_i} \tag{3.8}$$

$$L_{inter} = -\frac{1}{\pi N(n-1)}\sum_{i=1}^{N}\sum_{j=1,j\neq y_i}^{n} arccos(W_{y_i}^T W_j) \tag{3.9}$$

Euclidean-based losses

A popular alternative family of losses called Euclidean-based directly manipulates the intra and inter distance between samples, as illustrated in figure 3.13. Contrastive loss takes a random pair, pushes them together if they are the same face and pulls them apart if not. Triplet loss takes a step further by composing a triplet consisting of an anchor image, a positive of the same class and a negative of a different class. Thus, the intra and inter distances from the anchor are optimized simultaneously. Multi-class loss and constellation loss do this more efficiently by using numerous negative anchors at a time.

Despite the above advantages, these losses suffer from significant data expansion as there is a virtually infinite number of groups that can be generated the training data. Since the constitution of a pair or a triplet is random, these losses can only take into account a representative fraction of all possible sets. Additionally, researchers have found that they have difficulties converging.

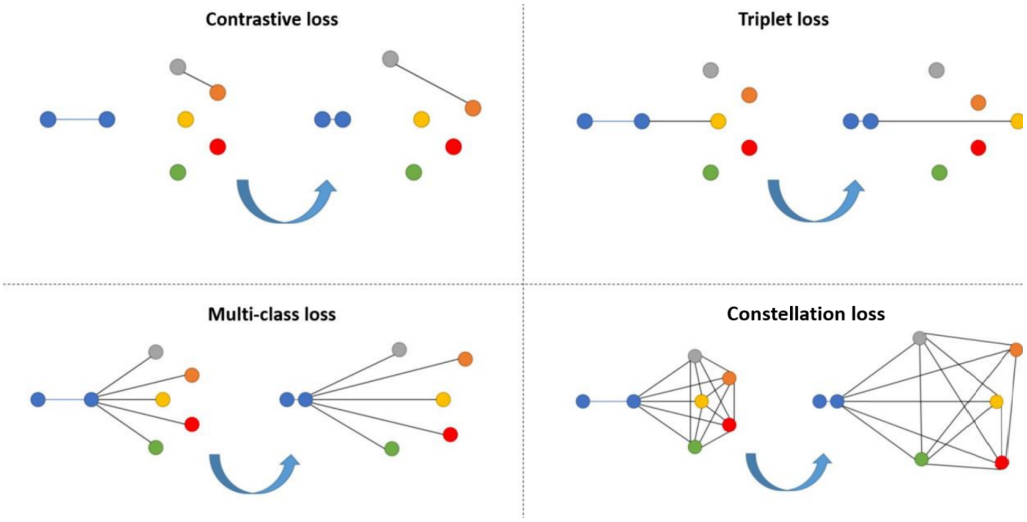

Figure 3.13: Gradient descent step for contrastive loss, triplet loss, multi-class N-pair and the constellation loss [26]

3.4.2 Margin-based softmax cross-entropy losses

The margin based softmax cross-entropy loss used in this project is capable of producing the same intra and inter class optimization effect as the euclidean-based losses. On top of that, it requires much less computational overhead and is able to converge quickly since it is just a slight variation of the original softmax cross-entropy loss.

Softmax cross-entropy loss (Equation 3.7) is first modified by fixing all biases B to 0 and transforming $w_j x_i$ to $\|w_j\|\|x_i\|cos\theta_j$, where θ_j is the angle between the sample x_i and the weight w_j. If the embedding vector x_i and all the weights are L2 normalized (Equation 3.4), their magnitudes becomes 1 and the loss becomes equation 3.10. The problem is now simplified to just minimizing the cosine of the angle θ_{y_I} between the embedding vector and the corresponding weight of its class.

$$L_{modified} = -\frac{1}{M}\sum_{i=1}^{M} log\frac{e^{cos\theta_{y_i}}}{e^{cos\theta_{y_i}} + \sum_{j=1,j\neq y_i}^{n} e^{cos\theta_j}} \tag{3.10}$$

Next, a margin m is introduced into the equation to expand the separating region between identities. SphereFace [22] uses a multiplicative angular margin as $cos(m_1\theta_{y_i})$. ArcFace [25] adds an angular margin to the cosine as $cos(\theta_{y_i} + m_2)$. CosFace [27] applies an additive cosine margin as $cos\theta_{y_i} - m_3$. Figure 3.14 illustrates the effects of the above margins to the boundary between two classes. Despite their differences, all three methods are much better at compressing faces of the same identity and enhancing the separation between different ones than the original softmax cross-entropy loss. This has proven to be incredibly effective in handling unseen faces.

Combining all of the above margins and re-scaling the cosines by a factor s, we have the

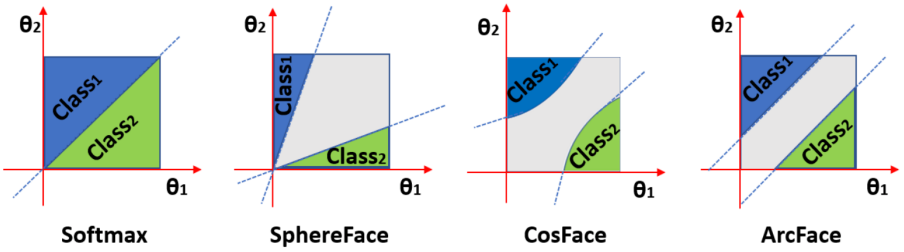

Figure 3.14: Decision margins of different loss functions under binary classification case [25]

final function 3.11, where the parameters s, $m1$, $m2$, $m3$ are chosen based on validation results.

$$L_{combined} = -\frac{1}{M}\sum_{i=1}^{M} log\frac{e^{s(cos(m_1\theta_{y_i}+m_2)-m_3)}}{e^{s(cos(m_1\theta_{y_i}+m_2)-m_3)}+\sum_{j=1,j\neq y_i}^{n} e^{s(cos\theta_j)}} \tag{3.11}$$

A big draw back of this loss is that all of its parameters s, $m1$, $m2$, $m3$ are chosen based on experience and experiments. The scaling parameter s is usually 64. The InsightFace project [17] has trained a model to find optimal settings of $m1$, $m2$, $m3$ according to the validation results on three different data sets, their work is demonstrated in table 3.1.

Table 3.1: Validation result of different settings of m1, m1, m2

Training loss	m1	m2	m3	LFW [18]	CFP-FP [19]	AgeDB-30 [20]
Softmax	1	0	0	99.28	88.50	95.13
SphereFace	1.5	0	0	99.76	94.17	97.30
CosineFace	1	0	0.35	99.80	94.4	97.91
ArcFace	**1**	**0.5**	**0**	**99.83**	**94.04**	**98.08**

In this project, we decide to train our own model using the combined margin loss with the ArcFace settings and combine it with Intra and Inter loss (equation 3.8 and 3.9) and compare it with InsightFace's model.

Chapter 4

Phase 2: Recognition system implementation

This section describes the architecture of our face recognition system and explains in detail the algorithm or technology behind each 4 main components, which are:

1. **The pre-processor:** This part contains several sub-components. Its roles include face detection, image normalization and reference image quality control.
2. **The embedding vector extractor:** It extract facial features from an image and encode them to an embedding vectors. The technology behind this component is the deep learning model from the previous training phase.
3. **The database:** This is where the ID number, the face prints i.e reference embedding vectors of users are stored for later inference.
4. **The identifier:** This is just an algorithm that compares and matches an unknown embedding against the database and predicts its identity.

The system's components co-operate to perform two main tasks - registration and identification. The figures below describe the pipeline for each process.

Registration process of a person (Figure 4.8):. The system first takes a digital image or a video frame that capture the face of the person as inputs. This can either belong to a new face or someone that has already been registered into the system. After pre-processing the image, the system pass it through a deep learning model that extracts facial features of out of the image and converts them to an embedding vector. Finally, all embedding vectors are stored in a database along with their owner's ID.

Identification process of an image (Figure4.2): Just as the registration process, the image is pre-processed and encoded to an embedding vector. A similarity score between the vector and that of each person in the database is then computed and used to classify the image as either belonging to one of the registered or a stranger.

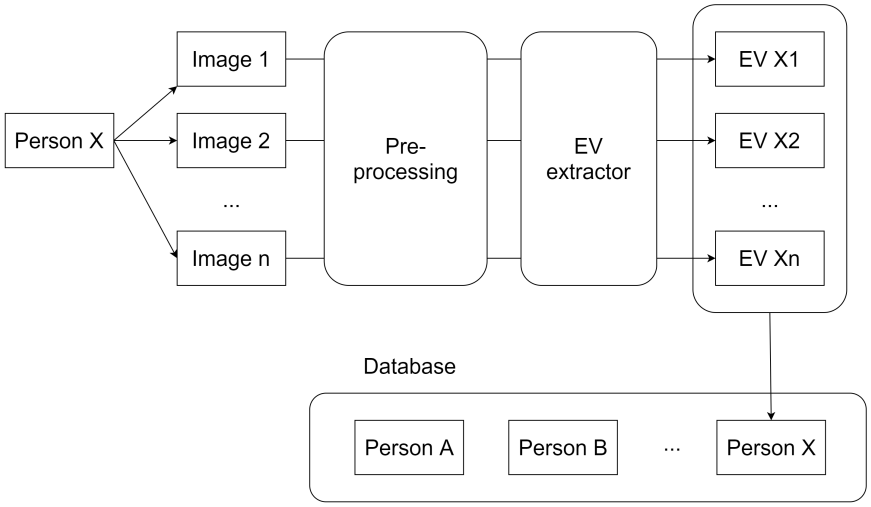

Figure 4.1: Registration pipeline

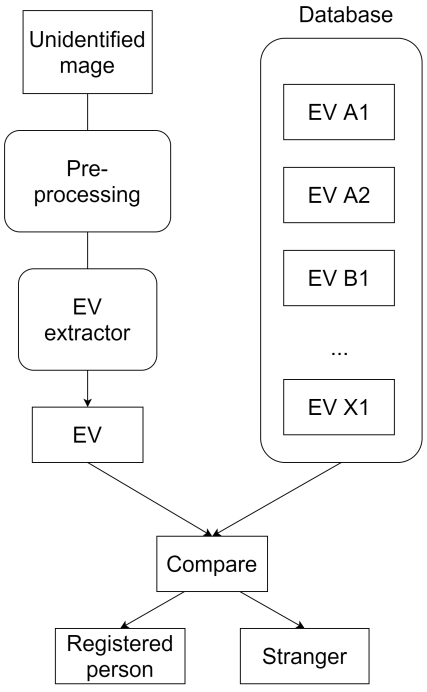

Figure 4.2: Identification pipeline

4.1 Pre-processor

4.1.1 Face detection and normalization

This pre-processing step is no different from that in the model training phase. Firstly, RetinaFace [21] detects faces from the input image and returns five facial landmarks. The image is then aligned, cropped, resized and pixel values are normalized. Only data augmentation is no longer needed.

4.1.2 Face angle constraint

Incentive

In an ideal setting, our system is provided with a detailed reference frame that consists of images portraying the user facing numerous directions. It is, however, virtually impossible to capture that many angles in real life and hence, it is best that user take the clearest side of their face i.e. the frontal view to register. In fact, some face verification applications such as Apple's Face ID do require their users to align their face to a mesh that is facing forward.

An image taken from a direct view point usually portrays facial features the clearest and side views are harder to differentiate, even with the human eyes. Figure 4.3 illustrates how two people can have extremely similar side profiles (top row) despite completely distinctive frontal views (bottom row). Additionally, a side profile is sometimes eclipsed by hair or shoulders, which hinders the recognition process even more.

Figure 4.3: Similar side views (top row) and different direct views (bottom row) of two people

In this project, a simple method is proposed to verify if a face is deviating no more than 30 degrees from the center. The method utilizes the facial landmarks detected by RetinaFace and thus requires little extra processing power.

There are indeed myriad existing head pose estimation methods but many of them involve complex algorithms or even neural networks to accurately calculate the face's direction. That level of exactness is excessive for this angle constraining task.

Data set study

In order to deduce the method, we first analyze the Head Pose Image Database [28]. The data set contains a total of 2790 photos of 15 people panning and tilting their heads by various angles, each pan-tilt pair produces a unique pose configuration. The "tilt" values in the data set are -90, -60, -30, -15, 0, +15, +30, +60, +90 degrees and the "pan" values are -90, -75, -60, -45, -30, -15, 0, +15, +30, +45, +60, +75, +90 degrees (Figure 4.4).

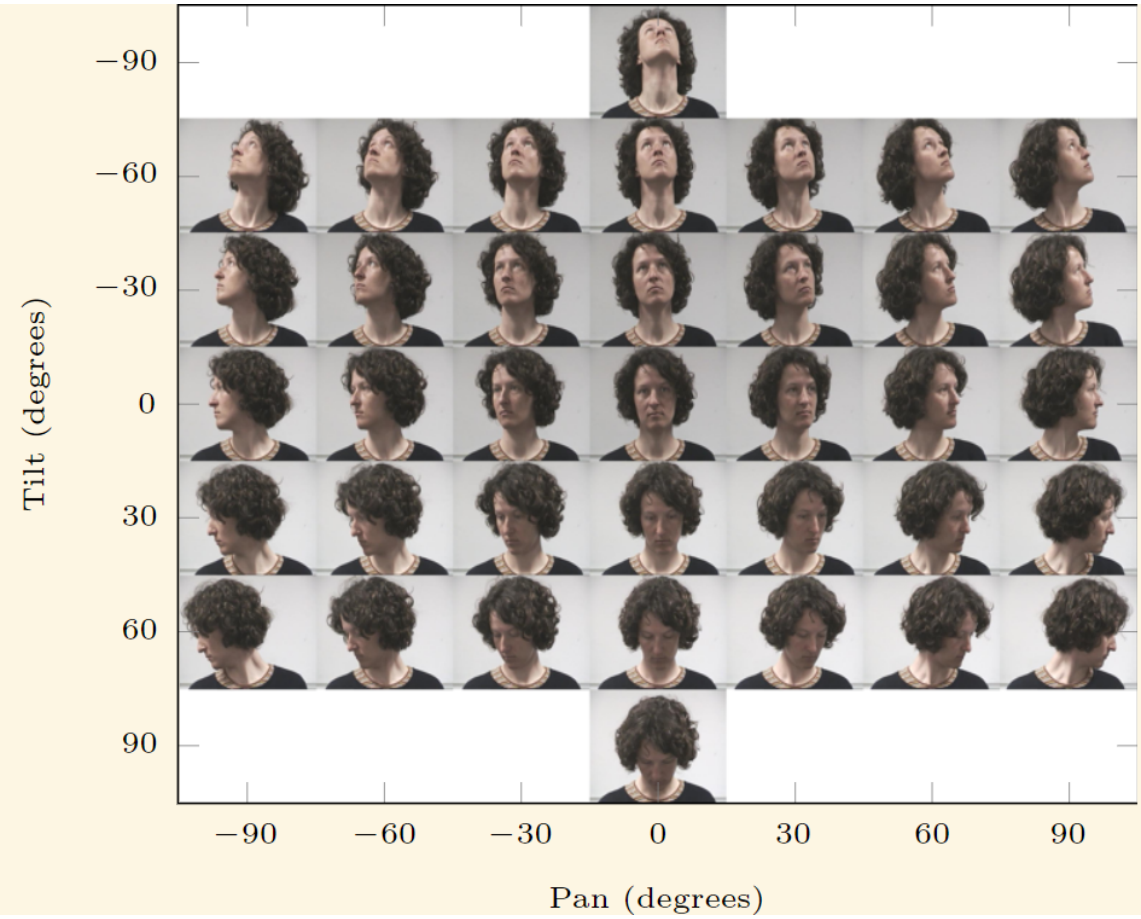

Figure 4.4: Some annotated samples from the Head Pose Image Database [28]

The landmarks of each photo in the data set are detected by RetinaFace [21]. It is clear that the eyes and the corners of the mouth always create a quadrangle and when the face is facing straight, the tip of the nose lies approximately around the center of it (Figure 4.5). The more the face tilts and pans, the further away from the center the nose tip, to the point where it lies completely outside of the quadrangle (Figure 4.6).

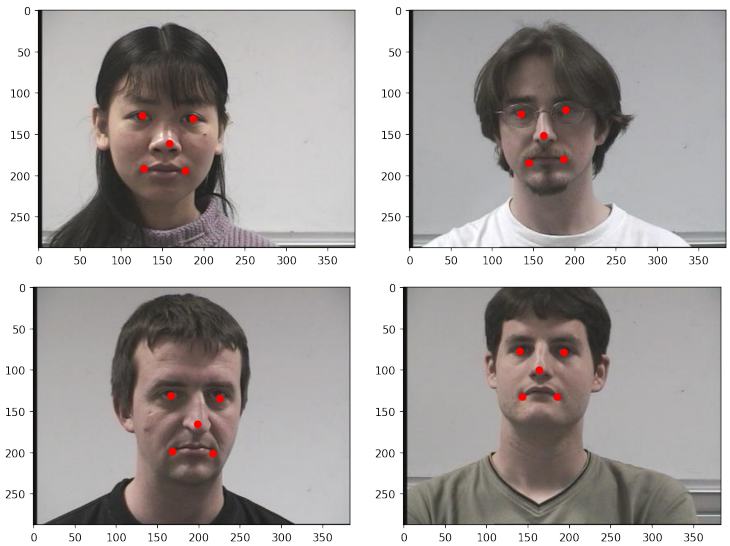

Figure 4.5: Facial landmarks of different people's frontal views

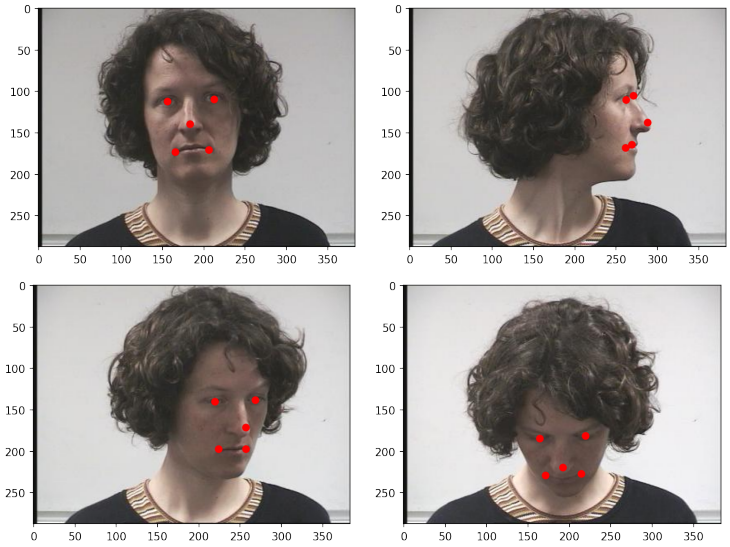

Figure 4.6: Facial landmarks of frontal, 3/4 and side view of one face

Our survey shows that among faces tilting and panning no more than 30 degrees in the data set, 98% have the nose tip lying inside the quadrangle. On top of that, 75% of the faces that deviate over 30 degrees does not satisfy this condition.

After eliminating the unsatisfying 75%, we measure the the distances from the nose tip to 4 sides of the quadrangle: d_{top}, d_{bottom}, d_{left}, d_{right}. Next, we find the optimal vertical threshold for the ratio between d_{top} and d_{bottom} and an optimal horizontal threshold for the ratio between d_{left} and d_{right} (Figure 4.7).

Figure 4.7: d_{top} and d_{bottom} - d_{left} and d_{right}

Result

Table 4.1 presents the computed vertical and horizontal thresholds that optimize accuracy, precision, recall and F1. It is evident that maximizing precision means a tight constraint that restricts faces mainly around the center view. On the other hand, the settings for three other metrics allow more variation. In fact, they are quite similar and do not produce much different results. Subsequently, we decide to use the thresholds for the best F1 as it balances between precision and recall.

Table 4.1: Optimal thresholds for various metrics

Metric	Vertical threshold	Horizontal threshold
Best accuracy	2	22
Best precision	2	3
Best recall	6	22
Best F1	3	22

In conclusion, the two conditions to verify if a face is panning and tilting no more than 30 degrees from the center view are expressed as below. This method achieves 80% classification accuracy on the Head Pose Image Database.

- The nose tips lies inside the quadrangle created by the eyes and mouth corners.
- The ratio between d_{top} and d_{bottom} is no more than 3 and the ratio between d_{left} and d_{right} is no more than 22.

This has been proven by the results in the next chapter to be an effective quality control method for registering images. If face angle constraint is also applied during the prediction process, any image that does not satisfy the conditions is automatically classified as belonging to a stranger.

4.2 Embedding vector extractor and database

Embedding vector extractor

The optimized neural network from the previous training phase is plugged in the system as an embedding vector extractor. After training, this deep learning model has been generalized for all human faces, except for its classification layer.

The last layer has n units corresponding to n classes in the training data set and thus, its weights are specific to the trained identities. In order for the model to deal with unseen faces, the classifier is removed, leaving only the convolutional backbone and the embedding layer.

Database

This project only employs a minimal data scheme and any database optimization is out of scope. Each person has an unique ID number and multiple reference embeddings. If adaptive threshold (section 4.3.2) is applied, each embedding has one corresponding threshold.

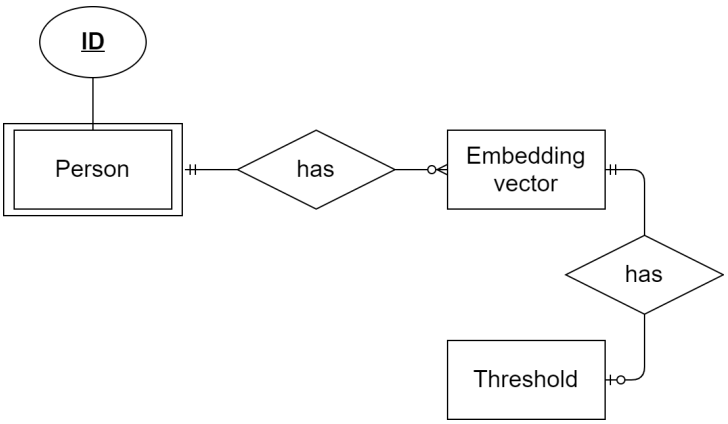

Figure 4.8: Entity relationship diagram of the database

4.3 Identifier

4.3.1 Predicting algorithm

Extracted embedding vectors are L2 normalized (Equation 3.4) and hence, can either be compared using the Euclidean distance or the cosine distance. In this project, we use the Euclidean distance.

One possible prediction method is computing the average embedding of each person's registering images as the facial center and the testing image is classified the same class as its closest facial center. Despite its robustness and small memory consumption, this method has relatively low accuracy, especially in few-shot situations where each profile only registers a minimal number of images.

In this project, a more efficient method is employed: classifying a sample based on its nearest neighbors. The pseudo-code for this voting strategy is written in algorithm 1. First, the algorithm compares the sample to all embeddings in the database; if the distance is smaller than a threshold, the embedding's class is considered one of the candidates. If none surpasses this threshold, the sample is predicted as belonging to a stranger. As for the remaining case, the algorithm selects the class with the highest appearing frequency in the candidate list. If two or more classes have the same frequency, the one with the closest embedding to the sample is chosen.

4.3.2 Adaptive threshold

An embedding's ID is only considered a candidate if its distance to the sample is smaller than a threshold. However, the optimal threshold is usually case-specific as different data sets or even different folds in the same set require a different value. And since real-life testing data is usually unseen and constantly changing, this value is hard to obtain. Therefore, the paper *Data-specific Adaptive Threshold for Face Recognition and Authentication* [29] proposed an adaptive threshold that tunes itself according to the growth of the registered database.

Original algorithm

The basic idea behind this scheme is to find a just enough margin to separate a sample from its surrounding neighbors. In other words, the threshold for each embedding is the distance between itself and the current closest sample not belonging to the same class. If only one class exists in the database, the threshold for each of its sample is set to infinity. As more identities appear in the database and the embedding hypershere gets more crowded, thresholds compress themselves to adapt to a more tightly-packed neighborhood. The pseudo-code for this method is written in algorithm 2.

Algorithm 1 Class prediction process of embedding vector v

```
for each sample in database do
  sample.d ← distance(sample.embedding, v)
  if sample.d < threshold then
    candidates.append(sample)
  end if
end for
if length(candidates) == 0 then
  prediction ← 'stranger'
else
  candidates ← sorted(candidates, key=lambda x: x.d)
  candidateClasses ← list(sample.class for each sample in candidates)
  f_max ← 0
  for each class in unique(candidateClasses) do
    f ← frequency of class in candidateClasses
    if f > f_max then
      prediction ← class
      f_max ← f
    end if
  end for
end if
return prediction
```

Algorithm 2 Registration process of a new sample x with an adaptive threshold

```
x.threshold ← ∞
for each sample in database do
  if sample.class ≠ x.class then
    d ← distance(sample.embedding, x.embedding)
    if sample.threshold > d then
      sample.threshold ← d
    end if
    if x.threshold > d then
      x.threshold ← d
    end if
  end if
end for
database.add(x)
```

Modified algorithm

We first investigated the adaptive threshold's adaptation according to the size of the database, the average threshold is measured after the registration of each 32-image batch. This survey is conducted on a test data set with 8000 different identities (see section 5 for data set description). Figure 4.9 shows that the adaptive threshold decreases as the number of identities in hypersphere increases and it will continue to if we raise the number of identities even more.

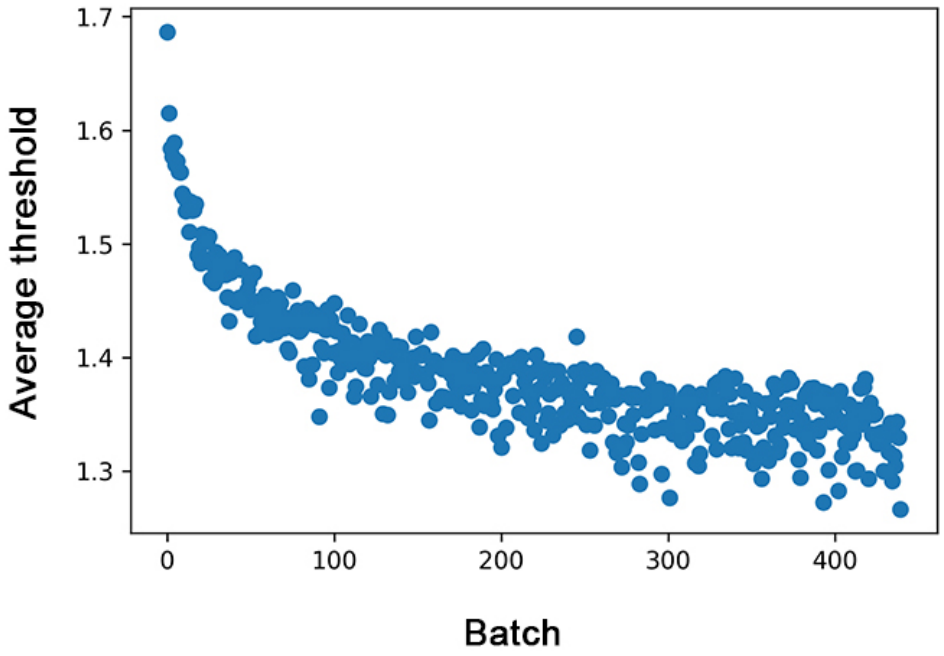

Figure 4.9: Average threshold value versus each registration batch

Looking at the threshold value of the first batches, we noticed that when the density of the embedding hypersphere is small, the minimal distance between samples of two classes are too large. This means each sample's threshold is only capable of separating it with other registered identities, not the ones that has yet to appear in the database i.e strangers. As a result, the adaptive thresholding scheme is only better than constant in large scale applications, not with a small database.

Thus, we propose a slight modification to the original algorithm that prevents large threshold values in a small database: any threshold that is larger than a constant t is fixed to t. The modification only involves changing initial threshold value from infinity to t. This value is specific to each data set and needs to be tuned accordingly.

Chapter 5

Result and discussion

Testing in conducted on the CASIA-Webface data set, containing 10,575 unique people with 494,414 images in total. We cleaned up duplicate samples and identities in the data set and also eliminated profiles with fewer than 8 images. The data set is then divided into two separate sets with the ratio 80:20.

- **The registered set:** People in this set are registered into the database using only 2 images, then the system would try to identify 6 other photos of them.
- **The stranger set:** People in this set are not registered to see if the system can correctly identify them as strangers.

The system is evaluated using the four basic metrics for any classification problems: accuracy, precision, recall and F1 (Equation 5.1 to 5.4). These metrics are computed using values from the confusion matrix in table 5.1. Each class's metrics are calculated individually and the final result is a weighted average of all.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{5.4}$$

Metrics are also calculated separately on the registered and strangers. The most important metrics for the registered test set are *accuracy and F1* because the cost of both false positive and false negative is high. As for the stranger set, *recall* is mainly used since mispredicting a stranger as a registered person is more harmful.

On top of that, the registered test set requires two more measurements, which are the percentage mistaken for *another registered person* and for a *stranger* among all false

Table 5.1: Confusion matrix

		True label	
		Positive	Negative
Prediction	Positive	True positive (TP)	False positive (FP)
	Negative	False negative (FN)	True negative (TN)

predictions. It is generally better to minimize the previous to 0 at the cost of the latter in practical applications.

$$Mistaken\ with\ stranger = \frac{FP_{stranger}}{FP_{stranger} + \sum\limits_{\substack{class \in C \\ class \neq label}} FP_{class}} \tag{5.5}$$

$$Mistaken\ with\ another\ registered = 1 - Mistaken\ with\ stranger \tag{5.6}$$

5.1 Deep learning model training results

We employ the ResNet-100 architecture and train it using the combined margin loss with settings $m1 = 1$, $m2 = 0.5$, $m3 = 0$ and combine it with Intra and Inter loss (equation 3.8 and 3.9). We also compare its performance with that of pre-trained ResNet-100 by the InsightFace project using the same combined margin settings. Beside verification results on the three data sets LFW[18], CFP-FP[19] and AgeDB-30[20], we also use identification accuracy on the CASIA-Webface data set. The comparison is presented in table 5.2.

Table 5.2: Results of our training vs InsightFace's

Training loss	LFW	CFP-FP	AgeDB-30	CASIA-Webface
Ours	99.70	96.1	97.90	0.72
InsightFace pre-trained	**99.83**	**94.04**	**98.08**	**0.82**

It is clear that none of our training attempts can reproduce the results of the pre-trained models by the InsightFace project, which is maybe due to the fundamental difference in our training settings. The InsightFace projects trained their models on 4 GPUs in paralell with a large batch size. Out available resources only allows training on 1 GPU and 32 samples per batch.

At the end, we decided to use the InsightFace's pre-trained model for our system.

5.2 Recognition system's performance evaluation

5.2.1 Overall evaluation

We measure the performance of the system with varying number of registered identities. It is natural for the metrics to deteriorate when the database increases in size.

Table 5.3: Metrics on the registered set with increasing number of people in the database

Database size	2000 identities	4000 identities	8000 identities
Accuracy	0.818	0.805	0.798
Precision	0.738	0.728	0.725
Recall	0.818	0.805	0.798
F1	0.770	0.758	0.752

Since we want to minimize the number of registered people mispredicted as another one in the database, it is good that 90% of all false predictions are mispredictions as a stranger (Table 5.4). It would be the best if this number is increased to 100% so that no one is mistaken with another person in the database.

Table 5.4: False predictions analysis on the registered set with increasing number of people in the database. Keep in mind that the percent is among only false predictions, not all.

Database size	2000 identities	4000 identities	8000 identities
Mistaken with another registered	0.053	0.086	0.128
Mistaken with stranger	0.947	0.914	0.872

Table 5.5: Metrics on the stranger set with increasing number of people in the database

Database size	2000 identities	4000 identities	8000 identities
Accuracy	0.980	0.972	0.945
Precision	0.980	0.972	0.945
Recall	1.000	1.000	1.000
F1	0.990	0.986	0.972

5.2.2 Ablation studies on applied methods

In this section, we conduct separate evaluation for the two important applied techniques in the system: face angle constraint and adaptive threshold.

Face angle constraint evaluation

We compare the system's performance when registered images satisfy and not satisfy the proposed face angle conditions. We only use the original adaptive threshold as the identification algorithm in this case since the proposed modified version needs a predefined initial threshold specific to the database.

It is evident that our proposed method have successfully selected high-quality registration images. The filtered images prove to be poor reference frames as using them result in a significant drop in the metrics on the registered set.

Table 5.6: Metrics on the registered set with when registration images do and do not satisfy face angle conditions

Face angle conditions	not satisfied	**satisfied**
Accuracy	0.148	**0.747**
Precision	0.446	**0.808**
Recall	0.148	**0.747**
F1	0.196	**0.754**

Adaptive threshold evaluation

Our proposed modified adaptive threshold algorithm requires a hand-chosen initial threshold and thus, we experiment with various values of this on a small set of 500 people to see its effect on the system's performance.

For this particular data set, we find 1.20 is the best initial threshold because it results in a high rate of mispredicting as a stranger among all false predictions, which minimizes the rate of mispredicting as another registered person, without sacrificing too much accuracy. Nonetheless, one can choose other values that fit with other data and other optimization goals.

Table 5.7: Effect of various initial threshold values on the registered set

Initial threshold	1.30	1.25	**1.20**	1.15	1.10
Accuracy	0.850	0.847	**0.838**	0.824	0.809
Ratio mistaken with stranger	0.670	0.819	**0.909**	0.941	0.954

Chapter 6

Conclusion and future work

To summarize, in this thesis, we have surveyed the state-of-the-art deep learning and classification methods and employ some of them to build a simple face recognition system. The system has about 80% accuracy when dealing with 2000 to 5000 identities in the database and only a 2% rate of mispredicting as another registered person. It also show 90% accuracy when presented with a stranger. This system is qualified to be used as an identification application for users or employees at a small to medium scale company.

As for future work, we would like to explore other deep learning backbones such as the Inception network [23] and train them with a large data set for better validation and testing results.

We would also like to experiment with other novel loss functions, such as the one proposed in the paper *AdaCos: Adaptively Scaling Cosine Logits for Effectively Learning Deep Face Representations* [30]. This is an improved version of the combined margin softmax cross-entropy loss used in this thesis where parameters can automatically adapt instead of being chosen by hand.

We can also improve the system's security by adding a layer of protection against attacks. There are four main types of attacks against face recognition system as is shown in figure 6.2. The real face can be hidden and the system is presented with a mask or print of another face, or a playback video on a digital screen. A technology called face spoofing can separate live faces from these attacks.

In the context of the COVID-19 pandemic, people are required to wear masks in public, which significantly hinders the performance of current face recognition systems. Novel models can be developed to detect faces wearing a mask and even predict their identities.

In conclusion, this project has granted us the opportunity to improve researching and coding skills and expand our knowledge in the fields of deep learning and computer vision.

Figure 6.1: Face recognition attacks [31]

Figure 6.2: People wearing masks in public [32]

Bibliography

[1] M. Turk and A. Pentland, "Face recognition using eigenfaces," in *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1991, pp. 586–591. DOI: 10.1109/CVPR.1991.139758.

[2] L.-L. Huang, A. Shimizu, and H. Kobatake, "Classification-based face detection using gabor filter features," USA: IEEE Computer Society, 2004, ISBN: 0769521223.

[3] S. Brahnam, L. C. Jain, L. Nanni, and A. Lumini, *Local Binary Patterns: New Variants and Applications*. Springer Publishing Company, Incorporated, 2013, ISBN: 3642392881.

[4] Z. Cao, Q. Yin, X. Tang, and J. Sun, "Face recognition with learning-based descriptor," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 2707–2714. DOI: 10.1109/CVPR.2010.5539992.

[5] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "Pcanet: A simple deep learning baseline for image classification?" *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5017–5032, Dec. 2015, ISSN: 1941-0042. DOI: 10.1109/tip.2015.2475625. [Online]. Available: http://dx.doi.org/10.1109/TIP.2015.2475625.

[6] Z. Lei and S. Z. Li, "Learning discriminant face descriptor for face recognition," in *Computer Vision – ACCV 2012*, K. M. Lee, Y. Matsushita, J. M. Rehg, and Z. Hu, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 748–759.

[7] "50 years of biometric research: Accomplishments, challenges, and opportunities," *Pattern Recognition Letters*, vol. 79, pp. 80–105, 2016, ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2015.12.013.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, ISSN: 0001-0782. DOI: 10.1145/3065386. [Online]. Available: https://doi.org/10.1145/3065386.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. arXiv: 1409.1556 [cs.CV].

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. arXiv: 1512.03385 [cs.CV].

[11] M. Wang and W. Deng, "Deep face recognition: A survey," *Neurocomputing*, vol. 429, pp. 215–244, Mar. 2021, ISSN: 0925-2312. DOI:

10.1016/j.neucom.2020.10.081. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2020.10.081.

[12] F. Chollet, *Deep Learning with Python*, 1st. USA: Manning Publications Co., 2017, ISBN: 1617294438.

[13] *Artificial neural network*, https://commons.wikimedia.org/wiki/File:Colored_neural_network.svg.

[14] *How do stacked cnn layers work?* https://datascience.stackexchange.com/questions/77830/how-do-stacked-cnn-layers-work.

[15] R. D. Reed and R. J. Marks, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. Cambridge, MA, USA: MIT Press, 1998, ISBN: 0262181908.

[16] Y. Guo, L. Zhang, Y. Hu, X. He, and J. Gao, "Ms-celeb-1m: A dataset and benchmark for large-scale face recognition," 2016. arXiv: 1607.08221 [cs.CV].

[17] *Insight face project*, https://github.com/deepinsight/insightface.

[18] G. B. Huang, M. A. Mattar, T. L. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database forstudying face recognition in unconstrained environments," Tech. Rep., 2008.

[19] S. Sengupta, J.-C. Chen, C. Castillo, V. M. Patel, R. Chellappa, and D. W. Jacobs, "Frontal to profile face verification in the wild," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2016, pp. 1–9. DOI: 10.1109/WACV.2016.7477558.

[20] S. Moschoglou, A. Papaioannou, C. Sagonas, J. Deng, I. Kotsia, and S. Zafeiriou, "Agedb: The first manually collected, in-the-wild age database," pp. 1997–2005, 2017. DOI: 10.1109/CVPRW.2017.250.

[21] J. Deng, J. Guo, Y. Zhou, J. Yu, I. Kotsia, and S. Zafeiriou, "Retinaface: Single-stage dense face localisation in the wild," 2019. arXiv: 1905.00641 [cs.CV].

[22] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, "Sphereface: Deep hypersphere embedding for face recognition," 2018. arXiv: 1704.08063 [cs.CV].

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014. arXiv: 1409.4842 [cs.CV].

[24] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," 2017. arXiv: 1605.07678 [cs.CV].

[25] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "Arcface: Additive angular margin loss for deep face recognition," 2019. arXiv: 1801.07698 [cs.CV].

[26] A. Medela and A. Picon, "Constellation loss: Improving the efficiency of deep metric learning loss functions for optimal embedding," 2019. arXiv: 1905.10675 [cs.LG].

[27] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, and W. Liu, "Cosface: Large margin cosine loss for deep face recognition," 2018. arXiv: 1801.09414 [cs.CV].

[28] N. Gourier, D. Hall, and J. Crowley, "Estimating face orientation from robust detection of salient facial structures," 2004.

[29] H.-R. Chou, J.-H. Lee, Y.-M. Chan, and C.-S. Chen, "Data-specific adaptive threshold for face recognition and authentication," 2018. arXiv: 1810.11160 [cs.CV].

[30] X. Zhang, R. Zhao, Y. Qiao, X. Wang, and H. Li, "Adacos: Adaptively scaling cosine logits for effectively learning deep face representations," 2019. arXiv: 1905.00292 [cs.CV].

[31] R. Cai and C. Chen, "Learning deep forest with multi-scale local binary pattern features for face anti-spoofing," Oct. 2019.

[32] P. Nuki, "The hundred years' war over face masks - and why we'll all be wearing them soon," 2020.