Trunk-Branch Ensemble Convolutional Neural Networks for Large Scale, Few-Shot Video-to-Still Face Recognition

P. Chen
Trunk-Branch Ensemble Convolutional Neural Networks for Large Scale, Few-Shot Video-to-Still Face Recognition

by

P. Chen

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday July 29, 2019 at 15:00.

Student number: 4703308
Project duration: November 1, 2018 – July 31, 2019
Thesis committee: Dr. N. Yorke-Smith, TU Delft, supervisor
Dr. J. Urbano, TU Delft
D.T.G. Mocking, MSc. ORTEC B.V.

An electronic version of this thesis is available at http://repository.tudelft.nl/.
# Contents

1 Introduction ................................................. 2  
1.1 Context ................................................. 2  
1.2 Brief literature survey ................................. 3  
1.3 Research goal and research questions ............... 4  
1.4 Outline ................................................. 5  

2 Background .................................................. 6  
2.1 Challenges in practical face recognition .......... 6  
2.2 Face recognition architecture ......................... 7  
2.3 Convolutional neural networks (CNNs) .............. 10  
2.4 Loss functions ........................................... 15  
2.5 Datasets ................................................. 18  

3 Research Methodology ....................................... 21  
3.1 Hypothesis .............................................. 21  
3.2 Implementation details ................................. 22  
3.3 Research approaches ................................... 23  

4 Experiments and Results .................................. 26  
4.1 Difference between training on MS1M subset (20k) .... 26  
4.2 Benchmarks and different loss functions ............ 27  
4.3 TBE-CNN Performances ............................... 27  
4.4 Performances on V2S identification dataset ......... 30  
4.5 Performances on S2S identification dataset ......... 32  
4.6 Insights and answers to research questions ......... 34  

5 Conclusions and Recommendations ...................... 36  
5.1 Further research ....................................... 37  

References ...................................................... 38
Abstract

Due to the huge improvement of hardware's computational power, complex neural networks can be calculated and more and more interesting research topics are discussed and implemented into our real world. One of the most famous topics in the research is Face Recognition (FR). By training deep neural networks with lots of images containing people's faces, machines can identify people well in images that the network has not seen before. This application can be used into areas such as security checking, customer emotional responses in a store, and so on.

For the real-world face recognition, factors such as occlusion and pose-variant (cross face) would affect the identification/verification performance. In addition, large number of classes also increase the complexity, which makes verification/identification even harder. In order to deal with these issues, how to extract discriminative embeddings is a challenging task for the researchers. This research aims at video-to-still (V2S) face identification, which means given few images per person (p.p.) as our database (also called gallery), we tend to identify if a person in a video is someone in our database or not. We use end-to-end Trunk-Branch Ensemble Convolutional Neural Networks (TBE-CNN) combined with state-of-the-art InceptionNet to create informative image patches and boost the features for occlusion and pose-variant issues. The images for learning and identifying comprise occlusions, different face directions and also resolution issues. Moreover, we create a large scale, few shot video-to-still (still-to-still) face recognition dataset with different settings to evaluate the the models and find the preferred settings for real-world face recognition application. How to ensure the accuracy under these practical noises and settings is the goal of this research.
Introduction

This thesis is part of an internship at ORTEC in Zoetermeer, The Netherlands. ORTEC is a company that provides advanced analyses services, planning and scheduling products for its customers. The company has been operating for over 35 years and has many offices in Europe, America, and Asia. ORTEC’s customers include large multinationals such as Shell and Coca-Cola. ORTEC aims at providing efficient algorithmic solutions in the form of software packages for its customers. ORTEC is always searching for ways to improve and has strong connections to the academic world. ORTEC supervises research projects to contribute their knowledge and learn the current state-of-the-art techniques to remain at the top of the market.

ORTEC created a face recognition system for one of its clients to identify the client’s customers in-store. The system extracts features of people based on one image per person, and identifies them while they pass a camera. However, people who wear glasses or do not directly face the camera can not be recognized well by the system. Therefore, how to deal with these problems became the goal of this research. In addition, if the system is introduced to other clients, it is necessary to provide information such as a quantifiable description of how good the model is, and the preferred settings for implementing the system toward the clients preferences. In many cases, the client’s customer base is large and only a few images per person are available. Therefore, we aim to solve the problems by improving the model, and provide clients with performance metrics and insights into the model for real-world face recognition tasks.

1.1. Context

Unconstrained face recognition is one of the most challenging problems of computer vision. With numerous use cases like criminal identification, attendance systems, face-unlock systems, and many others, face recognition could become a part of implementations in our lives. The simplicity of using these recognition tools is one of the major reasons for its widespread adoption in industrial and administrative use. But at the same time, in situations where computational power is limited, how to maintain its complexity and accuracy is difficult.

Due to the revolutionary work of Krizhevsky [35] winning the ImageNet Large Scale Challenge 2012 [55] by Convolutional Neural Networks (CNNs), many CNN-based approaches for computer vision problems have been proposed and have tremendous improvement. Performances in tasks such as image recognition [24, 67], object detection [40, 54], and biomedical image analysis [10] improved a lot by these CNN-based models. Many deep learning based frameworks such as DeepFace [70], FaceNet [56], and DeepID [63] also have achieved great accuracy in face recognition tasks on major datasets like LFW [37] and YTF [78].

The rise in the performance in image recognition was observed when CNN architectures of increasing depth were created. Examples of these networks are GoogLeNet [66, 68] and ResNet [24]. Whereas, it is found that after certain depth, the performance tends to saturate towards mean accuracy, i.e., after a certain level, more depth has almost no effect over performance [24]. At the same time, large scale application of face recognition would be prohibitive due to the need of high computational resources for deep architectures. Thus, in recent years, researchers are also working on the other aspects of the CNN model like loss functions, nonlinearities,
optimizers, etc. One of the major works done in this field includes the development of suitable loss functions, specifically designed for face recognition. Early works towards loss functions include Center Loss [76] and Triplet Loss [56] which focused on reducing the distance between the current sample and positive sample and increase the distance for the negative ones. Recent loss functions like Soft-Margin Softmax Loss [39], Cosine Loss [74], L2-Softmax Loss [53], Large-Margin Softmax Loss [43], and A-Softmax Loss [44] have shown promising performances which is close to (better than) human recognition.

Moreover, research of implementations in the real-world environment have been proposed since the given inputs for face verification might not be ideal. Factors such as occlusions, resolution, age difference, and pose-variation would exist in the real case and further result in misidentification. Some research tended to overcome these issues by generating pose variation and illumination variation faces as inputs for training through 3D image synthesizing [45, 46], 2D deep neural networks [83, 84], and data augmentation [18, 19, 42, 63, 65, 72], some research tended to deal with these issues by assembled networks [19, 32, 42, 62–64, 75]. These approaches with great performances on public datasets allow researchers to learn and extend their knowledge for implementations in the real-world settings.

Depending on the scenarios, a face recognition system can be categorized into still-to-still (S2S), still-to-video (S2V), video-to-still (V2S), and video-to-video (V2V) based FR. Recently, some techniques such as [23, 29] have successfully generated representative embeddings for S2V and V2S tasks. These techniques compensated the lack of facial region of interests using multiple face representations, synthetic face generation and augmentation of the training set. Instead of processing the training sets, [11, 20, 27] learned effective face embeddings directly through their deep architecture and hierarchical nonlinear mapping. Extracting features from different parts of the faces have been widely used in recognition tasks [28, 31, 41, 81, 82], Ding. [19] and Parchami. [50] proposed an end-to-end Trunk-Branch Ensemble CNN model along with an improved triplet loss function to learn blur-insensitive face embeddings for video FR. The model tends to overcome occlusions and pose-variant issues by considering different face components as branches and integrating them into the network. The trunk network learns a holistic embedding of the image and the branches combined with shared low- and middle-level layers can better exploit their complementary information. The models performed well in V2S datasets such as COX [29] and PaSC [5], however, since these dataset only contain few number of identity as gallery (1,000 in COX and the 293 in PaSC), performance on large-scale, few-image V2S identification which fits our task still remains unknown.

Therefore, in this research, a V2S identification dataset is created for our task, which contains 55k identities with few images per person (p.p.) as database (gallery) and in total 1,000 identities’ video frames as tests (probes). Besides the dataset, an improved TBE-CNN aiming at finding the suitable branches for V2S FR is proposed in this research. Both the trunk network and its inception blocks are changed to Inception-v3 [68] and refined based on the ideas in InsightFace [16]. Different current state-of-the-art loss functions will be used in training process in order to increase the discriminative capabilities for both verification and identification.

1.2. Brief literature survey

We review the literature in two parts: 1) video-based face recognition (VFR), and 2) deep learning methods for face recognition since these parts are highly related to our research goal.

In video-based face recognition models, face sequences and/or face images are used together in the training stage. However, the main challenge of VFR is that either the video-based dataset is unlabeled or they are noisy/small dataset. Compared with the large-scale image dataset such as MegaFace [34] and Celeb-1M [22], there are not enough classes (identities) for training the model to learn the discriminative features. Therefore, instead of using the video dataset for training directly, research such as [19], and [50] generated artificially simulated video dataset by motion blur, shearing, mirroring, rotating, and other techniques on image sets. These augmentation methods are often used in still-to-video or video-to-still face recognition scenario. Furthermore, in order to find discriminative face features for face recognition, [19] proposed the idea of Trunk-branch ensemble CNN (TBE-CNN) combining with refined triplet loss (MDR-TL) to deal with occlusions, image blur, and pose-variant issues. An extension called HaarNet [50] was proposed to efficiently learn the robust face representations since the face landmarks detection in [19] might fail because of the occlusions. Both of them have great performances on public video-based face dataset (YouTube Faces [78],
COX [29]), which seems to fit our research goal. However, in practical implementations, we would have large amount of people/identities/classes saved in our database. So far, most of the video-based dataset now for video-to-still or still-to-video face recognition task contains less than 3,000 people in the database and only tends to compare the performance of identifying people who had seen/learned in the database. Questions such as will the performance remain the same if we have an increasing amount of people (e.g. 50k or more) existed in the database? and what is the misidentification rate? remains unknown for these models and tests.

As for deep learning methods, there have been many research proposed to achieve the goal of large-scale face recognition. As mentioned in Chapter 1.1, researchers found out that, after a certain threshold, models with more depth has little effect on the performances and are not suitable for large-scale face recognition's applications because of the high computational resources. More works on loss functions [16, 39, 43, 44, 53, 56, 74, 76], data augmentation [19, 26, 50], non-linearity processing [30] are proposed to find better discriminative face embeddings. Currently, DeepInsight proposed InsightFace [16], a deep learning architecture with refined ResNet [24] models and a new marginal loss function in angular space (ArcFace), for large-scale face recognition and verification. First, the author proved that data cleaning can easily result in better performances on public face verification sets (LFW [37] and CFP-FP [58]) and proposed their cleaned MS1MV2 dataset, originally from [22]. Second, the performance can also be improved by making some changes in deep convolutional neural networks (such as using BatchNorm-dropout-FC-BatchNorm as output, and reducing the image size and kernel size of the first convolutional layer). Last but not least, the proposed ArcFace also has the best performances on many large-scale face dataset (MegaFace [34], Trillion pairs [15], etc.) competing with other state-of-the-art loss functions such as Center Loss [76], CosFace [74], SphereFace [44], and Triplet-loss [56]. However, most of the tests in the paper are still-to-still and video-to-video face verification/recognition, the performance on video-to-still face recognition remains unclear.

Based on these aforementioned questions on video-to-still face recognition, in this research we create a mock dataset for video-to-still face recognition with few images per person, large classes and propose a refined TBE-CNN structure to compete with these state-of-the-art models. First, our test set contains people who exist and do not exist in the database, which help us to understand both the identification rate (true acceptance rate) and misidentification rate (false acceptance rate) of each model. We would like to understand how good our model performs compared with those video-based face and image-based face recognition models in practical settings. Second, the dataset contains different settings: 1) 1 image, 2-5 images, and 6-10 images p.p. in the database, and 2) different length per video clip (20 frames, 50 frames, and 100 frames, usually around 30 frames/second), which helps us answer the preferable settings (e.g. #image(s) p.p., #frames per video, etc.) for our task.

1.3. Research goal and research questions

The goal of this research is to identify clients (large number of classes) who uploaded their photos (one image or few images) in our database given images/videos as test set. For people who are not in our database, we should identify them as ‘Not in the database’. Both database and test set contain images that fit the real-world situation (pose-variant, occlusion, different illumination, and so on). In other words, this research aims at creating/find suitable models for V2S/S2S face identification with large classes and few shots per person.

Since current video-to-still identification tests [5, 19, 50] only consider few identities (gallery) in the database, which could not answer the scalability in the real implementation, we create the video-to-still dataset for our test by combining parts of different dataset (MS1MV2 [16], and COX [29]). The probes contain people who exist and do not exist in the database, which allows us to see the precision and recall of the identification. More details of the dataset is described in section 2.5.

In addition to the scale difference of the database, probes (human faces) taken in the real world (e.g. surveillance videos) usually suffer from issues such as pose variations, occlusions and image blur. According to previous research [19, 50], the idea of TBE-CNN combined with different loss functions such as ArcFace/Triplet Loss/CosFace tends to overcome the aforementioned issues. However, which part(s) of face to be used will result in better performance in our task still remains unclear. Therefore, among their ideas, we propose an extended deep learning architecture which combines Trunk-branch ensemble networks with Inception-v3
This research will explore and apply deep learning model to answer the question:

**How to improve/maintain the accuracy of current state-of-the-art model for large scale, few images video-to-still (still-to-still) face recognition?**

For the deep learning architecture, we will find the proper branches (parts of face) for dealing the occlusions, pose-variant issues, and find the best loss functions from previous research (ArcFace [16], CosFace [74], Triplet Losses [19, 50, 56]) for our task. As for the test of our created dataset, we would answer questions regarding scalability, required number of images per person, and preferred number of frames per probe. In other words, the sub-questions of this research are:

1. What is the best branch(es) and loss function for our task?
2. What is the scalability (stability) of our model?
3. How many images per person are preferred as database, and how many frames (seconds) are preferred as probe?

### 1.4. Outline

Since there have been many approaches on face recognition tasks, we therefore decided to first reproduce the current state-of-the-art models and create test sets regarding our video-to-still task. Next, we implement these models to our test sets in order to obtain the benchmarks for our tasks and check what makes the identities be misclassified. After that we implement refined Trunk-branch ensemble- CNN with different branches and some other modifications to improve the performance. Finally we will compare performances in different factors that are mentioned in the sub-research questions (e.g. scalability, how many frames and images do we need, and so on).

Chapter 2 first gives some background of face recognition process/architecture and challenges in the real-world implementations, after which it provides more details about what functions are used to extract better discriminative features, and details of current popular datasets for training and testing. Chapter 3 states our research methodology, which is inspired by Ding’s [19] and Parchami’s [50] research. We further specify our approaches including how our test sets are created, specification of our models, and the implementation details of the frameworks and hardware used in this research. The experimental results and the comparisons to the benchmarks are shown and discussed in Chapter 4. Finally, we end with a conclusion and some recommendations in Chapter 5.
In this chapter, we will start with challenges of face recognition in the real world (section 2.1). In section 2.2, current face recognition pipelines including detector, extractor, and matching methods are introduced. Since this research focuses on how to improve the performance of the face embeddings extractor for our task by using different deep convolutional neural network (DCNN) model, we will discuss everything we need to know about Convolutional neural network in Section 2.3. Next, we discuss some loss functions that have been widely used in face recognition and object detection for finding more discriminative features. We end this chapter with some popular verification/identification datasets for model evaluation.

2.1. Challenges in practical face recognition

The most discriminative features on face are the combination of eyes, nose, shape of face and forehead. Most of the time, we suggest that faces for testing and learning are clear frontal face. However, in the real-world face recognition, images in both database and probes sometimes are not perfect, which will further affect the recognition's performance. Here we list those important factors:

• **Occlusion**
  In real-world cases, people would wear sunglasses, glasses, lenses, mask, and so on. These materials will cover parts of the informative faces or make people look differently than before, resulting in further distance between the extracted deep features. Fig. 2.1 (last column) shows an example of occlusion (sunglasses).

• **Pose-variant**
  As shown in Fig. 2.1 (the third column), identities in the images may hard to be classified or may easily be misclassified as the same person when they are taken in different angles. Similar to the occlusion issue, performance would drop since variant pose also lets us have fewer informative parts of face for recognition. Normally, preprocessing steps such as cropping and alignment are implemented to adjust the face to our preferable position before feeding into the model.

• **Lower Resolution**
  When images are in lower resolution, it is hard for the machine to find the correct match since most of the informative parts for FR such as eyes and nose are blurred. Although using better device to obtain high-resolution images would increase the learning quality, the cost of the device is also higher than using surveillance camera. It is unrealistic to ask for those expensive device everywhere. Therefore, how to maintain the performance when only low-quality images are available is also a challenge in real-world recognition.

• **One-shot learning**
  Number of image per identity will affect the identification performance as well. The model will be overfitting when we do not have enough images for training, therefore, most of the time the model is trained on multiple images combined with boosting methods such as horizontal flipping.
However, in the real-world cases, it is hard to collect certain people’s images due to their willingness and privacy issues. This will indeed limit the ability to learn discriminative features.

2.2. Face recognition architecture

There are three main parts to achieve general face recognition, which are face detection, face representation extractor, and identification/verification method. Each part has their impact on face recognition’s performance, and has been researched and evolved for so many years. Basic structures and different approaches of each part are described in the following subsections.

2.2.1. Face detection

First, it is important to detect if there is someone in an image and to find the location of his/her face. Most of the recent datasets (e.g. [22, 34, 37, 78]) use rectangular bounding boxes as output images. Not only the noisy background will be removed but the face can be aligned through the detector, which can significantly improve the performance. This can be considered as data preprocessing steps.

Algorithms of the face detector such as Viola&Jones Haar-cascade algorithm [71], SURF [4], HOG [14] were used before the burst of neural networks based on deep architectures. Recently, these methods were gradually replaced by deep architectures [38, 80], which achieved much better performances in face detection.

In addition to the designs applied to 2D images, research on 3D and 2D+3D face detection are also been discussed. It is mentioned that 3D data is relatively insensitive to pose and illumination variations [36], therefore many researchers have tried either to use 3D information or to transfer 2D detection techniques into 3D data [57]. However, the complexity of implementing these techniques is high, which is not suitable for our real detection now. Therefore, this research decided to focus on 2D face detection techniques.

Recently, a well-known face detection technique is MTCNN [16, 74, 80], which uses a cascaded architecture with three stages deep convolutional networks to predict face and landmark location in a coarse-to-fine manner. Then the faces is cropped, aligned, normalized, and resized based on the requirement of the CNN architectures. Example images that are cropped by trained MTCNN [16] are shown in Fig. 2.1.

![Example images cropped by MTCNN](image)

Figure 2.1: Examples of output images cropped and aligned through MTCNN model [16, 80]. Images on the first row are the original images in LFW [37], and images on the second row are the outputs.

2.2.2. Face representation extractor

Given an image of a person as input, face representation extractor tends to encrypt informative face features into a high-dimensional vector. Researchers currently use deep convolutional neural networks to extract deep discriminative features for identifying persons. Recently, the popular deep architectures are VGGFace2 [9], OpenFace [2], FaceNet [56], DeepID [63], InceptionNets [66, 68, 69], ResNets [16, 24, 26] and so on. Here we give some brief introductions about these models.

**VGG (2014).** The runner-up of 2014 (scoring 7.33%) was the first to recognize the importance of a CNN’s
depth for its performance as well as the advantage of using receptive fields of at most $F = 3$ in convolutional layers. This can be explained by the possibility to replicate any convolutional layer with $F = 5$ (and odd) by two consecutive convolutional layers with $F = 3$. This even increases the expressiveness of the model, because of the extra non-linearity between these layers. Similarly, we can replicate a convolutional layer with $F = 7$ by 3 consecutive convolutional layers with $F = 3$, and so on. Although they did not win the challenge, the VGGNet of Simonyan and Zisserman (2014) is worth mentioning because of its appealingly homogeneous architecture. It stacks five blocks of multiple convolutional layers, followed by three fully-connected layers and a softmax output. A ReLU activation is applied after every layer and, in addition, each of the five blocks is followed by a max-pooling layer. A downside of this CNN is that it has around 140 million parameters, making it computationally expensive to train and evaluate the model. Nevertheless, due to its simple but effective architecture, VGGNet has been an inspiration for many later architectures.

GoogleLeNet (2014) [66] and Inception-v3 (2016) [68]. The architecture of GoogleLeNet is of similar depth as that of VGGNet, but containing the more advanced Inception modules, introduced by Szegedy et al. (2014). Simply stated, these modules compute multiple convolutional (and max-pooling) layers at once and concatenate the results, so that the model can determine itself which convolutions are most informative. Together with the use of less fully-connected layers (which are very expensive), this results in a relatively efficient CNN of only 5 million parameters. GoogleNet also provides the basis of popular models developed by Szegedy et al. (2015, 2016), where the latter also uses the ideas of the 2015 ILSVRC winner.

Based on GoogleNet, some refined versions are proposed: with batch-normalization (inception-v2 [30]), and additional factorization ideas (inception-v3 [68]). The concept of batch-normalization is introduced later. The aim of the factorization is to reduce the number of parameters without decreasing the network efficiency. The author reduces the kernel size of convolutional layer to reduce the number of parameters needed. First, the $5 \times 5$ convolutional layer in inception module A (Fig. 4 in [68]) is changed to two $3 \times 3$ convolutional layers, the dimension of the output remains the same but the number of parameters is reduced from $5 \times 5 = 25$ to $2 \times 3 \times 3 = 18$, which is reduced by 28%. Next, convolutional layer with $7 \times 7$ kernel in module B is replaced by passing through $1 \times 7$ convolutional layer then $7 \times 1$ convolutional layer, which reduces the number of parameters by 71%. With changes on convolutional layers and auxiliary classifier, the refined inception-v3 model became the 1st Runner Up for image classification in ILSVRC 2015.

Residual Net (ResNet [24]) and its refined version [16]. Ideally, the identification performance should increase when deeper model is applied for training. However, researchers noticed that the training accuracy decreases while the model goes deeper, which is not overfitting. They suggested the reason behind this phenomenon is because of gradient vanishing/exploding. K. He proposed ResNet, which added two mappings called identity mapping (shortcut connection) and residual mapping (block) to deal with these issues. The identity mapping passes the previous features to the next layer. When the model goes deeper but has already saturated, the performance will at least remain the same because of the identity mapping. According to figure 2.2, we can compute the output $H(x) = F(x) + x$, where $F(x) = W_2 + \sigma(W_1 x)$. The author proves that letting the model learning $F(x) = H(x) - x$ is easier than learning $H(x) = F(x) + x$. If the identity mapping is optimal, it is easier to learn $F(x) = 0$ than to learn $H(x) = x$, which is the reason why $F(x)$ is called the ‘residual block’. Under several tests, the final ResNet not only won the 1st place ILSVRC and COCO 2015 competition in ImageNet Detection, ImageNet localization and many others, but made training a network with 100 and 1000 layers possible. In [16], the author proposed a refined version of ResNet by changing the first input image size from $224 \times 224$ to $112 \times 112$, and change the kernel size (from $7 \times 7$ to $3 \times 3$) and stride (from 2 to 1) of first convolutional layer to keep the feature size unchanged. Moreover, the author suggested that using BN (BatchNorm)- Dropout- FC (fully-connected layer)- BN after the last convolutional layer results in the best method of obtaining the discriminative face embeddings. The proposed (trained)ResNets can be found in the link of [16].

In addition to deep architectures, we also implement discriminative loss functions to further decrease the distance between intra-class (samples in a class) and enlarge the distance between inter-class (samples from different class) in geometric space or in angular space. Some widely-used loss functions will be introduced in Section 2.4.
2.2.3. Face identification and verification

A trained face identification model has the goal of identifying whether an input image/video is someone it has seen before during training, and if so who the person is. Unlike face identification, face verification is to let the machine classify if a pair of images is the same person or not. Usually, the identification output is the probabilities of an image corresponding to each of the trained classes, and the verification output could be either a value indicating the similarity of a pair of images or a binary result denoting is/is not the same person. Examples of face identification and verification are shown as in Fig. 2.3.
2.3. Convolutional neural networks (CNNs)

In order to fully understand CNNs, the structure and the components of CNNs are introduced in this section. When talking about CNNs, we usually consider it as classification issue. That is, given a data instance, a structure called classifier, will compute a score for all $C$ possible classes. The class with the highest score is then predicted as the corresponding class for the data instance. The number of $C$ is set based on how many classes we used for training and prediction. In our cases, $C$ is set to 20,000 for our training process and around 85,000 in [16]. In a special case that $C = 2$, we call this binary classification. An algorithm that returns a classifier based on a labelled training data is called a learner. We can consider CNNs as a learner which processes the data, learns how to identify classes from the labelled training data and predicts them given testing data. In the following, we will introduce some concepts of CNNs and methods for preventing overfitting and efficient learning.

**Feed-forward computation.** First we will introduce the computation flow of CNNs. A CNN is represented as a set of nodes ordered in one or more layers and are connected in a feed-forward manner. An example of a regular neural network is shown in Fig. 2.4. It contains 3 nodes as input, a depth (number of non-input layer) of 3, and 2 hidden layers with width (numbers of nodes in the layer) of 4. Given $m$ number of nodes in the current layer and $n$ nodes of the next layer, if the network is fully-connected, which means that all $m$ nodes contributes certain weights to every $n$ nodes, the number weights (without bias) needed are computed as $m \cdot n$. For example, the number of weight needed from the input layer to the first hidden layer is $3 \cdot 4 = 12$, if we consider a bias $b_j \in \mathbb{R}$ into the computation, then the number parameters needed is $3 \cdot (4+1) = 15$. And the number of parameters from the input layer to the output layer including bias is $3 \cdot (4+1) + 4 \cdot (4+1) + 4 \cdot (1+1) = 43$ parameters, which is often used as a measure for the size of neural networks. Let $V_l$ denotes the set of nodes in layer $l$, then the output $x_j$ including bias $b_j$ is computed as $X_j = b_j + \sum_{i \in V_{l-1}} w_{i,j}x_i$.

![Figure 2.4: A regular 3-layer neural network with three nodes as input layer, and two outputs. Image is taken from [33].](image)

**Activation function.** An activation function is a node that is put at the end or in between the layers. It can be considered as a transfer function that maps the output of previous layers to other values. The transformed output is then sent to the next layer of neurons as input. Activation functions can generally be divided into two types: linear and non-linear. Linear activation function, is to transfer the output to another value by multiplying weights for each neuron. It has the problem that the derivative of the function is a constant which is not related to the input, which makes back-propagation impossible because we cannot know which weights in the input neurons can contribute better prediction. In contrast, the derivative of non-linear activation functions are related to the inputs, which makes back-propagation possible and can learn how to predict complex dataset by stacking multiple layers and non-linear activation functions together. Thus, in CNNs we usually talk about non-linear activation functions. Some commonly used activation functions are:

- **Sigmoid function:**
  $f(x) = 1/(1 + e^{-x})$
  Sigmoid function maps any real value to a value between 0 and 1. Sigmoid function is specially used in models where we would like to predict the probability as output.

- **Hyperbolic tangent activation function (tanh):**
  $f(x) = \tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$
  Unlike sigmoid function, the output of hyperbolic tangent activation function is in the range between -1 and 1 (zero centered). Since the derivative of tanh is steeper than sigmoid function, sometimes it is
2.3. Convolutional neural networks (CNNs)

preferred over sigmoid function. However, both sigmoid and tanh have gradient vanishing problem, meaning that for very high or very low values, there is almost no change to the prediction and results in slow divergence.

- **ReLU (Rectified Linear Unit) activation function:**

  \[ f(x) = \max(0, x) \]

ReLU is the most used activation function in the world right now. The major benefit of ReLU is the reduced likelihood of the gradient vanishing. When the input is larger than 0, the gradient has a constant value instead of extremely small value in tanh and sigmoid functions. The constant gradient of ReLUs results in faster learning. But ReLU has **dying ReLU problem** when the inputs approach to zero or are negative. The gradient of the function becomes zero and the network cannot perform back-propagation.

- **Leaky ReLU activation function:**

  \[ f(x) = \max(0.01x, x) \]

It is an attempt to solve the dying ReLU problem, as it does not have zero-slope parts. Instead of giving a predetermined value 0.01 in leaky ReLU, we can also let the neural network learn the parameter \( \alpha \) for better prediction itself, which is then called **Parametric ReLU (PReLU)**: \( f(x) = \max(\alpha x, x) \). Plots of all mentioned activation functions and their derivatives are displayed in Fig. 2.5.

<table>
<thead>
<tr>
<th>Name</th>
<th>Plot</th>
<th>Equation</th>
<th>Derivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity</td>
<td></td>
<td>( f(x) = x )</td>
<td>( f'(x) = 1 )</td>
</tr>
<tr>
<td>Logistic (a.k.a. Soft step)</td>
<td></td>
<td>( f(x) = \frac{1}{1 + e^{-x}} )</td>
<td>( f'(x) = f(x)(1 - f(x)) )</td>
</tr>
<tr>
<td>Tanh</td>
<td></td>
<td>( f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 )</td>
<td>( f'(x) = 1 - f(x)^2 )</td>
</tr>
<tr>
<td>Rectified Linear Rect. (ReLU)</td>
<td></td>
<td>( f(x) = \begin{cases} 0 &amp; \text{for } x &lt; 0 \ x &amp; \text{for } x \geq 0 \end{cases} )</td>
<td>( f'(x) = \begin{cases} 0 &amp; \text{for } x &lt; 0 \ 1 &amp; \text{for } x \geq 0 \end{cases} )</td>
</tr>
<tr>
<td>Parametric Rectified Linear Rect. (PReLU)</td>
<td></td>
<td>( f(x) = \begin{cases} \alpha x &amp; \text{for } x &lt; 0 \ x &amp; \text{for } x \geq 0 \end{cases} )</td>
<td>( f'(x) = \begin{cases} \alpha &amp; \text{for } x &lt; 0 \ 1 &amp; \text{for } x \geq 0 \end{cases} )</td>
</tr>
</tbody>
</table>

**Input layer.** In CNNs, the input layer denotes the input of the structure. In computer vision tasks such as image recognition and face recognition, we usually use images as the input. Essentially, every image can be represented as a matrix of pixel values with one or more channel (e.g. a certain component of an image, here it denotes red, green, blue) stacked over each other. An image from a standard camera will have three channels and can be imaged as three 2d-matrices stacked together, with each pixel value in the range of 0 to 255. This means that the input layer is a 3-Dimensional input with size of \( \text{height} \times \text{width} \times 3 \). It is quite common to use normalized (mean = 0, and \( \text{std} = 1 \)) Red-Green-Blue (RGB) pixel values as input.

**Convolutional layer.** The primary purpose of convolution is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data, which is called kernel or filter. In general, the filter has the dimension of \( F \times F \times D \), where \( F \) is called the receptive field, and \( D \) equals to the depth of the input layer. By sliding the filter over the image and computing the dot product, we then obtain another matrix, which is called a feature map. A CNN will learn the weights of the filters by itself during the training process for our recognition task. An illustration of the convolutional terms and process is shown in Fig. 2.6.
The way to slide the filters over the input layer is defined by:

1. The filter (kernel) size $F$.
2. The stride $S$, determining the number pixels we move at a time.
3. The size of the zero-padding $P$ around the boarder, which is used to preserve the spatial size of the input volume so the input and output width and height are the same (or reduce slowly).

Given an input layer with size $H_1 \times W_1 \times D_1$, and $K$ amount of filters with size of $F \times F \times D_1$, the slides with stride $S$, padding $P$ will output a feature map with size $H_2 \times W_2 \times D_2$, where:

$$H_2 = \frac{H_1 - F + 2P}{S} + 1, \quad W_2 = \frac{W_1 - F + 2P}{S} + 1, \quad D_2 = K$$

**Pooling layer.** Similar to convolutional layer, pooling layer (also called downsampling) reduces the dimensionality of each feature map but keeps the most important information in each sliding kernel. There are several types of pooling layer: Average, Max etc. Unlike the convolutional layer, a pooling layer computes a fixed function of each region (kernel) without computing the dot product. Therefore, there is no extra parameters needed in pooling layers. An illustration of how max-pooling and average-pooling work with $F = P = 2$ is shown as in Fig. 2.7. Given an input volume with size $H_1 \times W_1 \times D_1$ and the receptive field $F$, the size of the output after pooling will be $H_2 \times W_2 \times D_2$, where:

$$H_2 = \frac{H_1 - F}{S} + 1, \quad W_2 = \frac{W_1 - F}{S} + 1, \quad D_2 = D_1$$

In particular, pooling results in smaller feature dimension, reduces the number of parameters and further reduce the computations of the network, which could control overfitting. Moreover, it helps us get an almost scale-invariant representation of our image [13].

**Stochastic gradient descent and mini-batch gradient descent.** Gradient descent is an optimization algorithm used for finding the weights that minimize the error of the model on the training dataset. It is done by changing the values in the model that move it along a slope (gradient) of errors down to a minimum error value. There are many types of gradient descent depending on the number of training data used to compute the error. **Stochastic gradient descent (SGD),** calculates the error and updates the model for each sample in the training dataset. The frequent updates of stochastic gradient descent could result in faster learning on some problems. However, the model is updated so frequently, which is more computationally expensive and will take significantly long time to train models on large dataset. Therefore, the **mini-batch gradient descent** comes in. Instead of updating the model for each sample, mini-batch gradient descent algorithm splits the training set into subsets (called batches) and updates the model after computing the errors of each batch. Some terminologies are used in mini-batch gradient descent. A cycle through the training dataset is called an *epoch*. Given a dataset with $N$ samples and a batch size $B$, *iterations* is the number of batches needed to complete one epoch, which is $\frac{N}{B}$.
2.3. Convolutional neural networks (CNNs)

Backpropagation. The gradients of the losses which are computed after the feed-forward computation tell the network how to change all the weights and biases to decrease the cost. Backpropagation is the algorithm for computing the complicated gradients. Given an example shown as in Fig. 2.8, the cost function of the computational graph is \( f(x, y, z) = (x+y)z \). The function can be broken down into two functions: \( q = x+y \) and \( f = qz \). Therefore, we know how to compute the gradient of each function: \( \frac{\partial f}{\partial z} = q \), \( \frac{\partial f}{\partial q} = z \), and \( \frac{\partial q}{\partial x} = 1 \), \( \frac{\partial q}{\partial y} = 1 \). Since we are interested in the gradients of \( f \) with respect to the input \( x, y, \) and \( z \). The chain rule tells us that using multiplication is the correct way to chain the gradient expressions together, that is: \( \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} \). The way computing the gradients from output to the input is called the backward pass. Given the inputs \((x, y, z) = (-2, 5, -4)\), we then obtain each gradient as: \( \frac{\partial f}{\partial x} = -2 \cdot 1 = -4 \), \( \frac{\partial f}{\partial y} = -2 \cdot 1 = -4 \), and \( \frac{\partial f}{\partial z} = (x+y) = 3 \).

By propagating the gradients from the last layer back to each neuron, the weight of each neuron is updated and hopefully approaches to the (local) optimum.

Overfitting, underfitting and regularization. When training a model, it is always important to know if the model is overfit or underfit. Generally, overfitting means that a model models too well on the training data but performs badly on new (test) data. It happens when a model learns the noise and detail which exist in the training dataset but cannot be generalized to new dataset. Unlike overfitting, underfitting refers to a fact that neither the training dataset nor the test dataset are modelled well by a model. Sometimes we can check if the model is overfitting or underfitting based on the training and testing curve. Fig. 2.9 shows the trends of training loss vs validation loss in overfitting, underfitting and good fitting situations.

Usually, we could prevent overfitting in a deep learning model by training on a larger dataset, or changing the complexity of the model (changing the number of weights, or changing the number of parameters). We can also use regularization to control the complexity of the learned model. In deep learning, some commonly methods of regularization are dropout and batch normalization.
2. Background

(a) Underfitting

(b) Good fit

(c) Overfitting

Figure 2.9: Train loss vs validation loss for checking underfitting, good fit, or overfitting. Images are taken from [7].

**Dropout.** Introduced by [61], dropout ignores a number of layer outputs while training. We can consider this step as an effect of making the training process noisy, which lets nodes in a layer probabilistically take more or less responsibility for the inputs. It is also mentioned in [6, 61], saying that dropout could make the model more robust:

“This conceptualization suggests that perhaps dropout breaks-up situations where network layers co-adapt to correct mistakes from prior layers, in turn making the model more robust.”

Using dropout, for each node, weights are updated by the backpropagation algorithm with probability $p$ (with probability $1-p$ a node’s weights remain unchanged). Example of the connection in a network before and after dropout algorithm and the difference of node during training and testing is shown in Fig. 2.10. Dropout is also proved to be effective in some cases in the experiment in [60].

Figure 2.10: Illustrations of dropout from [61].
Batch Normalization (BN) Batch Normalization was first introduced in [30]. The authors showed a great improvement by adding batch normalization into the layers. Today, batch normalization is used in almost all CNN architectures. The goal of implementing batch normalization is to solve the problem of internal covariate shift. Covariate shift means the distribution of the features is different in different subset of train/test dataset. Take Fig. 2.11 as example, we tend to let a model learn to classify if an image contains a rose. As we can see in the figure, some of the roses are in different colors and some of them are fully-blown roses. If the selected subset are really different, this results in different distribution in the feature space, and the model will need more time to learn how to classify the rose. The internal covariate shift refers to covariate shift happening in a neural network. While going from one layer to another layer, the input distribution changes every time when the weights are updated, the higher layers then are forced to adapt to the drift, which requires small learning rate and slows down learning. BN first normalizes the inputs of each layer to zero means and unit variances. Then the normalized inputs are scaled up by learnable $\gamma$ and shifted by $\beta$ as the non-linearity’s input to increase learning capacity. When applying batch normalization, small changes in parameter to one layer do not get propagated to other layers. This makes it possible to use higher learning rates for the optimizers and makes gradient propagation in the network more stable.

![Figure 2.11](image_url)
Figure 2.11: Illustration of covariate shift (taken from [47]), where the top two rows and the bottom two rows are different subset of data, and the last column shows their distribution in feature space. The Blue line denotes the decision boundary between two classes.

2.4. Loss functions

One of the important factors in face recognition is the face extractor, which creates face embeddings for verification and identification tasks. Deep Convolutional Neural Networks (DCNNs) are the powerful architectures for these missions. However, one of the main challenges in feature learning is the design of loss functions in order to obtain discriminative features. In this section some of the popular and powerful loss functions are discussed. We also propose a refined version of triplet center loss for better efficiency and better performances on large-scale face recognition.

2.4.1. Center loss

Center loss [76] was proposed to compensate for cross-entropy loss by learning a center for each class and pushing the deep features to their centers. The total loss is computed as the total squared Euclidean distances between the deep features and their own centers, which is formulated as:

$$L_c = \frac{1}{2} \sum_{i=1}^{n} D\left(f(x^i), c_{y_i}\right)$$

(2.1)
where $c_y \in \mathbb{R}^d$ denotes the center of class $c_y$, with $d$ the dimension of features. Function $D(\cdot)$ denotes the squared Euclidean distance. The gradients of the features are the direction toward their centers, and the centers are updated based on every mini-batch. Squared Euclidean distance is denoted as:

$$D(f_i, c_y) = \frac{1}{2}\|f_i - c_y\|^2_2.$$  \hfill (2.2)

### 2.4.2. Triplet loss

Triplet loss [56] is calculated on triplet samples $(x_i^a, x_i^+, x_i^-)$, where $x_i^a$ is the anchor of a triplet, $x_i^+$ the sample in the same class of anchor (positive), $x_i^-$ the sample in different class of anchor (negative). The idea of triplet loss is to encourage the distance between the anchor to positive image ($D(f_i, f(x_i^+))$), and the distance of the anchor to negative image ($D(f_i, f(x_i^-))$) to be larger than at least a margin $m$. The loss is formulated as:

$$L_t = \sum_{i=1}^n \max(D(f_i, f(x_i^a)), D(f_i, f(x_i^+)) - D(f_i, f(x_i^-)) + m, 0).$$ \hfill (2.3)

### 2.4.3. Triplet-center loss

Triplet-center [25] loss was proposed to deal with the drawbacks of center loss and triplet loss. In center loss, although the features are push to their given centers, distances between the centers are not in consideration. On the other hand, in triplet loss, the number of triplets grows dramatically when the size of dataset increases, which results in longer training time. In addition, its performance heavily depends on the selection of the anchor, negative, and positive examples. Therefore, considering the belonged center and the closest other center as positive and negative sample in triplet-center loss, we are able to efficiently learn the discriminative features. Given a batch of training data with $M$ samples, triplet-center loss is defined as:

$$L_{tc} = \sum_{i=1}^M \max(D(f_i, c_{y_i}) - \min_{j \neq y_i} D(f_i, c_j)) + m, 0)$$ \hfill (2.4)

where $D(\cdot)$ denotes the squared Euclidean distance (Eq.2.2)

### 2.4.4. ArcFace, SphereFace, and CosFace

Instead of considering the distances of features in a high-dimensional space, [44, 73] consider the geometric distributions of the features. When both the deep features and the weights of the last fully connected layers (without bias) are normalized, the linear transformation of them can also be considered as their angular distance, which is formulated as:

$$W_{y_i}^T x_i = \cos(\theta_{y_i})$$ \hfill (2.5)

where $x_i$ denotes the deep features of the $i_{th}$ sample, and $W_{y_i}$ the corresponding weight of its class.

SphereFace [44] assumes that weights of the last fully connected layer can be considered as class centers in an angular space, then the angles between the deep features and the weights can be penalized to learn the features well. Similar to the idea of triplet loss, adding a margin is one of the methods to improve the separability. ArcFace [16], SphereFace [44], and CosFace [74] are proposed based on three different penalties. The combined formula can be written as:

$$L = -\frac{1}{N} \sum_{i=1}^N \log \frac{e^{\theta_{y_i}}}{\sum_{j=1}^N e^{\cos(m_i \theta_{y_i} + m_j - m_3)}} + \sum_{j=1, j \neq y_i}^N e^{\cos \theta_j}$$ \hfill (2.6)

where $m_1$, $m_2$, and $m_3$ are penalties from SphereFace, ArcFace and CosFace individually.
2.4.5. Evaluations

To compare the performance of a model, different evaluation metrics are implemented based on its tasks. For face verification, we set a threshold $T$ for deciding if people in a pair of image is the same person or not. If the distance (here we implement euclidean distance) $D$ between two images is larger than threshold (e.g. $D \geq T$), the model considers them as different people. If $D < T$ then the model suggests they are the same person. Owing to this decision, researchers use false acceptance rate (FAR), false rejection rate (FRR), and true acceptance rate (TAR) as evaluation metrics.

**False Acceptance Rate, FAR**

Given $M$ pairs of images of different people and a trained threshold $T$, the model misclassifies $N$ pairs of images as the same person (e.g. there are $N$ pairs of images with distances $D < T$), the false acceptance rate is defined as:

$$FAR = \frac{N \text{ pairs with their } D < T}{M \text{ pairs of different people}}$$

**True Acceptance Rate, TAR**

Given $H$ pairs of images of the same person and a trained threshold $T$, the model classifies $K$ pairs of images as the same person correctly (e.g. there are $K$ pairs of images with distances $D < T$), the true acceptance rate is defined as:

$$TAR = \frac{K \text{ pairs with their } D < T}{H \text{ pairs of same person}}$$

**False Rejection Rate, FRR**

Given $H$ pairs of images of the same person and a trained threshold $T$, the model misclassifies $L$ pairs of images as different people (e.g. there are $L$ pairs of images with distances $D > T$), the false rejection rate is defined as:

$$FRR = \frac{L \text{ pairs with their } D > T}{H \text{ pairs of same person}} = 1 - TAR$$

In order to compare the performance of the models, what we usually do is to fix one metric and compare the other one. For example, in [34], they compare TAR at $FAR=10^{-6}$ ($TAR@FAR=10^{-6}$) and in [77] they compare $TAR@FAR=1e^{-4}$. In addition to these metrics, researchers also plot the receiver operator characteristic (ROC) curve, which is a curve of FRR versus FAR, to understand which model performs better. In this research, these evaluation metrics will be used in our mock V2S identification.

**Averaged accuracy from 10-fold cross validation**

In some datasets such as LFW [37], CFP-FP [58], and YTF [78], we follow the policy of "Unrestricted, Labeled Outside Data" and use the provided 6,000 pairs (7,000 in CFP-FP) of images containing both matched and unmatched pairs for evaluation. The test sets are split into 10 folds, by looping each fold as the test set and the other nine as the training set for finding the threshold, we will obtain an averaged accuracy for face verification.
2.5. Datasets

For the dataset, first we train the model on part of refined MS1M dataset [16], which contains 20,000 identities and 1.3M images in total. Next, we explore the performance of our feature extractor based on LFW (Unrestricted, Labeled Outside Data) [37], CFP-FP [58] to check the improvement from different settings. As for video face verification, we utilize YTF dataset [78] for testing. Below is the description of the dataset and its usage.

2.5.1. MS-Celeb-1M dataset (MS1M, MS1MV2)

The original MS-Celeb-1M dataset [22] contains around 1M images for 100k celebrities, with distractor images (images of other celebrities or ordinary people on the web) in some of the identities. According to [16], the model performs better when trained without distractor images and therefore proposed semi-automatic refined version of MS-Celeb-1M dataset (MS1MV2), which removes most of the outliers (distractor images). Images in MS1MV2 are aligned, normalized, and cropped into $112 \times 112$ through MTCNN [80] architecture. In this research, we train the model on MS1MV2 as well to have fair comparison with other existing models.

2.5.2. Labeled Faces in the Wild (LFW)

Labeled faces in the wild dataset [37] is one of the most popular verification dataset to test the performance of a model. The dataset contains 13,233 face images collected from the web and is selected into 6000 pairs with almost half of them are the same person and half of them are not. Since the faces are taken in the wild world, issues of pose-variant, occlusion, and low-resolution exist in the dataset. An example of a pair is shown as in Fig. 2.12. In our test, we explore our performance with the refined version which is cropped and aligned by MTCNN [16], which is shown as in Fig. 2.13. We are able to test if the model has the ability to verify people well in pose-variant situation.

Figure 2.12: Example image pair of LFW dataset (Hootie Johnson 1 and 2), some of the images in the dataset have occlusion, pose-variant, and low-resolution issues which might affect the verification performance.

Figure 2.13: Example image pair of refined LFW dataset (Hootie Johnson 1 and 2), the faces are cropped, aligned, and scaled through MTCNN network to remove the noisy background.
2.5.3. Celebrities in Frontal-Profile in the Wild (CFP-FP)

Besides the widely used LFW dataset, CFP-FP dataset \cite{58} contains pairs of images with frontal face and large pose-variant face (profile face). Similar to LFW, the dataset is divided into 10 splits. For each split there are 350 pairs of images indicate the same person and 350 pairs indicate different person. In total there are 7000 pairs of faces for verification experiment. In our experiment, all images are cropped and aligned by MTCNN (obtained from \cite{16}), which is the same process as LFW. Differences between original pair and refined pair of faces are shown in Fig. 2.14 and Fig. 2.15.

Figure 2.14: Example image pair of CFP_FP dataset (Amanda Seyfried), most of the frontal face (left image) and profile face (right image) also have occlusions and resolution issues.

Figure 2.15: Example image pair of refined CFP_FP dataset (Amanda Seyfried), the faces are cropped, aligned, and scaled through MTCNN network

2.5.4. YouTube Faces dataset (YTF)

YouTube faces dataset \cite{78} contains 3,425 videos of 1,595 different people. Each video has at least 48 frames, some of them contains clear faces and some of them do not. Compares to LFW dataset, YTF database is for video face verification. Instead of taking only one feature vector, we compute the averaged feature vector extracted from each frame, and verify if two videos contain the same person. An example database is shown in Fig. 2.16.

Figure 2.16: Original example sequence of video frames in YTF database (Conan_OBrien/5). The faces has been aligned already, we also cropped the faces out and scaled by MTCNN architecture. The final feature representation is the mean of the feature vectors extracted from each frame.
### 2.5.5. COX dataset

COX database [29] is one of the database containing still and video faces designed for studying three typical scenarios of video-based face recognition: Video-to-Still (V2S), Still-to-Video (S2V) and Video-to-Video (V2V) face recognition. There are 1,000 identities in total, each identity has one frontal face as gallery, and three surveillance-like videos captured by three camcorder probes, which are shown in Fig. 2.17.

![Samples of still image (middle-left) and video clips captured by three camcorders (cam1, cam2, and cam3 from top to bottom row.](image)

As for evaluation, the author randomly selected 300 subjects for training and the rest of the 700 subjects for testing. This procedure was repeated ten times to have a convincible result. Since three camcorders were set in three different locations, the intensity of illumination, occlusion and pose-variant will also be different. Therefore, the result contains three rank-1 accuracies (V3-S, V2-S, and V1-S) for comparison.

A summary of the datasets above and other public datasets are shown in Table 2.1.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>#Identity</th>
<th>#Image/Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASIA [79]</td>
<td>10K</td>
<td>0.5M</td>
</tr>
<tr>
<td>VGGFace2 [9]</td>
<td>9.1K</td>
<td>3.3M</td>
</tr>
<tr>
<td>MS1MV2 [16]</td>
<td>85K</td>
<td>5.8M</td>
</tr>
<tr>
<td>MS1M-DeepGlint [16]</td>
<td>87K</td>
<td>3.9M</td>
</tr>
<tr>
<td>LFW [37]</td>
<td>5,749</td>
<td>13,233</td>
</tr>
<tr>
<td>CFP-FP [58]</td>
<td>500</td>
<td>7,000</td>
</tr>
<tr>
<td>CPLFW</td>
<td>5,749</td>
<td>11,652</td>
</tr>
<tr>
<td>AgeDB-30</td>
<td>568</td>
<td>16,488</td>
</tr>
<tr>
<td>YTF [78]</td>
<td>1,595</td>
<td>3,425</td>
</tr>
<tr>
<td>MegaFace (S2S) [34]</td>
<td>530(P)</td>
<td>1M(G)</td>
</tr>
<tr>
<td>COX (V2S, S2V) [29]</td>
<td>1,000</td>
<td>3,000</td>
</tr>
<tr>
<td>PaSC (V2V) [5]</td>
<td>265</td>
<td>2802</td>
</tr>
<tr>
<td>Trillion-Pairs (S2S) [15]</td>
<td>5,749(P)</td>
<td>1.58M(G)</td>
</tr>
<tr>
<td>iQIYI-VID</td>
<td>4,934</td>
<td>172,835</td>
</tr>
</tbody>
</table>

Table 2.1: Face datasets for training and testing. "(P)" and "(G)" refer to the probe and gallery set, respectively. "S" and "V" refer to still (images) and videos respectively.
Research Methodology

After having clear understanding about the structure and elements in CNNs, public datasets, training/testing processes, loss functions, and the problems we are aiming to solve, it is time to know the methodology and hypotheses used in this research. In this chapter, we first elaborate our hypotheses of our TBE-CNN. After which we describe which datasets are used for training the DCNNs and validation, which programming languages are used and how to train the models. In Section 3.3, the process to create our 55k V2S (S2S) dataset, and detail information of the branches are described. Eventually, we put the pseudo-code of our TBE-CNN model in order to allow the readers to reproduce our works.

3.1. Hypothesis

Since occlusion and different poses let an image have fewer informative features, the distance between two face embeddings extracted from the image increases. If the distance gets larger than the verification threshold, it will result in misclassification. An example shown as Fig. 3.1, it is also pretty hard for human to verify if a pair of image is the same person because of the occlusion.

![Figure 3.1: Differences of Euclidean distances between with and without occlusions in LFW dataset. Given three images of Claudia Pechstein 1,3, and 4 (Top), and Paula Radcliffe 2,3,and 4 (bottom), we can see that with occlusions (sun glasses), the distances between the extracted features get larger.](image)

According to the hypothesis in [3, 19], image patches are less sensitive to pose-variant and occlusion and have already been used in recognition tasks [28, 31, 41, 81, 82]. However, which parts of branches are recommended for video FR remains unknown. Therefore, there are two hypotheses in this research: 1) instead of selecting Haar-like image patches [50], we suggest that using patches like holistic face, upper face, and bottom part of face can result in better verification/identification stability for our V2S identification in the practical world implementations and can deal with occlusions and pose-variant issues as well. We select
the image patches by ourselves since the performances of human’s eyes have similar accuracy to machines’ decision. Face patches containing only cheeks or forehead are not taken into consideration because we cannot even use these parts with little information for identification. 2) We suggest that using more frames for identification would have better result.

We decided to select three image patches as the branches for our V2S identification, which are: 1. forehead to chin (holistic face without hair and background), 2. forehead to nose (upper face including eyes), and 3. nose to chin (bottom face). To test whether an image patch(s) is useful or not, we first train the trunk network with original cropped images, then compare the differences of adding each patch as branches into the trunk network. For the first patch (holistic face), although it has already been used in [19], since they used lots of image patch to get their final result, whether this part really somehow overcomes cross-pose identification/verification remains unknown. We suggest that features from holistic face without hair and noses will contribute more details and remove the noisy background. As for the other patches, we suggest that by learning how to identify people with lack of certain parts, the model could overcome occlusion issue. An example of an input image and its patches to the network is shown as in Fig. 3.2.

![Input Images](image)

**Figure 3.2:** Different patches being cropped and passed through the branches layers.

### 3.2. Implementation details

#### 3.2.1. Datasets

Table 2.1 has shown the current training and testing datasets. Because of our limited GPU usage, instead of using the whole MS1MV2 [16] dataset, we employ its subset (20k identities, around 1.3M images) for training and use the rest of them to create our V2S (S2S) dataset. In addition, we also explore efficient face verification datasets (e.g. LFW [37], and CFP-FP [58], and YTF [78]) to check the improvement of different settings.

In order to compare different models’ performances, we trained ResNet50 and ResNet100 from ArcFace on the subset MS1MV2 dataset to know which model works better. Besides these widely used datasets, we also report the performances of the models on V2S identification datasets [5, 29], which are close to our goal. In order to efficiently compare if our model performs better than the previous benchmarks, we also train our model on CASIA dataset [79] for comparison. Finally, we report performances and curves of the state-of-the-art models on the mock large-scale V2S identification sets. The details of creating mock V2S sets are described in the next section.

#### 3.2.2. Experimental settings

For data preprocessing, datasets are aligned and cropped by a trained MTCNN [16, 80]. We follow recent papers’ works [16, 44, 74] to generate the normalized face crop (112×112 for ResNet, 149×149 for our model, and 96×96 for [19, 50]) by utilizing five facial points [80]. For the embedding network, we employ Inception-v3 [68] and combined with trunk-branch-ensemble branches. 512-D embedding feature is obtained after the last fully-connected layer.
As for the hyper-parameters, we follow [16, 74] to set the feature scale $s$ to 64 and compare the accuracies from different angular margins $m$ of ArcFace (0.3, 0.5, 0.6, 0.7). Both ResNet50 and ResNet100 from ArcFace are implemented by MXNet [12], TBE-CNN models [19, 50], Inception-v3, and our extended networks are implemented by PyTorch [52]. We set the batch size to 32 and train models on one NVIDIA GeForce GTX 1080 Ti (11GB) GPU. During training, we first train the trunk network (Inception-v3) using Softmax function and cross entropy loss, with the learning rate of 0.01, and gradually reduce the learning based on the pytorch ReduceLROnPlateau scheduler with factor 0.1. Next the branches are added and trained again for 5 epochs. Finally, the TBE-CNN model is fine-tuned with other loss functions for more epochs with the learning rate of 0.001 combining with ReduceLROnPlateau scheduler. We obtained these parameter settings by comparing the performances under different settings, and some of them are shown in the next chapter. In this research, the model is trained on ArcFace, Triplet-loss, and Triplet center loss individually. We compare the verification accuracies on LFW, CFP-FP, and YTF in order to find the best loss function(s) for our task.

For the existed V2S identification test, we follow the evaluation metrics in [5, 29] in order to compare with previous research. As for our large-scale V2S (S2S) identification test, we not only provide ROC curves, precision/recall tables for a clear comparison but also provide multiple plots to have a thorough understanding about the scalability of a model and its preferred number of image (frame) per person (video clip) to answer the sub-research questions described in section 1.3.

### 3.3. Research approaches

#### 3.3.1. Large-scale V2S/S2S identification dataset

As described in previous chapters, current video-to-still identification sets [5, 29] did not consider misclassification rate, and the size of database is not that large enough. Therefore, in our experiments, a large scale, few-shot V2S (S2S) identification dataset is created based on MS1MV2 [16] and COX [29] dataset\(^1\). First, we split MS1MV2 [16] dataset containing around 85k identities into two subsets (20k for training the networks, and the rest for testing). Next, in order to answer our sub-research questions well, we extract the dataset again by removing those identities who have less than 20 images. Till this step, there are 57,763 identities left. Finally, we randomly pick 2,000 identities out becoming our probes (1k known with 1, 2-5, and 6-10 images, 1k unknown with 1, 2-5, and 6-10 images) and the rest of them (55,763 identities) as database. The extraction processes are described as shown in Fig. 3.3.

As for the video probes, we decide to use COX database [29], which contains 1,000 identities. There is one-shot gallery and three different video clips for each identity. We randomly split them into 500 unknown people and 500 known people. Combining with the created MS1M gallery, we are able to answer the scalability of our trained model. The extraction process is shown as in Fig. 3.4.

---

\(^1\)The dataset lists can be retrieved based on the DOI number: 10.4121/uuid:abade25d-19fc-43c9-8b86-2bd9a10f0b2c from 4TU.Centre for Research Data

![Figure 3.3: Extraction processes to create our database (gallery) and probes.](image1)

![Figure 3.4: Extraction processes to create our still gallery and video probes.](image2)
3.3.2. Deep convolutional neural network architecture

**Trunk Network** First, according to the models of [19, 50], both trunk networks are GoogleNet [66], which gives us two important ideas:

1. **Information from middle layers would contribute features for better performance.**
2. **Using $1 \times 1$ convolutional layer helps us reduce the number of layer.**

Based on these ideas, the model has evolved to the fourth version (Inception v3 [68], and Inception-ResNet [69]). Based on Fig. 3.5, we are able to see that Inception-v3 performs better than GoogleNet on ImageNet. Therefore, in this research, we tend to extend these papers by combining the idea of trunk-branch ensemble network with proper branches (face patches) into Inception-v3 network. Moreover, according to some changes in [16] comparing to original ResNet [24], we also implement similar changes in Inception-v3 for better performances in our task.

![Figure 3.5: Ball chart reporting the Top-1 accuracy vs. computational complexity. Top-1 accuracy using only the center crop versus floating-point operations (FLOPs) required for a single forward pass are reported. The size of each ball corresponds to the model complexity. Image is taken from [8].](image)

**Branches (Image patch)**

In order to have a better performance in our task, the selection of the image patches plays an important rule. According to [19], it is unclear which image patches should be used for pose-variant and occlusions. In [50], haar-like image patches are implemented and combined as final embedding feature. However, it is easy to see that the model would obtain redundant features given an profile face similar to Fig. 2.15 since the corner might contain useless information or noises such as backgrounds. Therefore, we compare performances of different image patches containing parts of facial five points in order to find suitable patches for our task, which are mentioned above in Fig. 3.2.

Moreover, we notice that the image patches following the processes in [19] contributes too little information as features. Therefore, in our research, we enlarge the size of the image patches from $17 \times 17$ to $25 \times 25$, and compare the performances of these to clarify our suggestion. Faces that cropped as image patches with two different sizes our shown as in Fig. 3.6, we could see that there are more information involved and would deal with occlusions and cross-face issues better than smaller crops.

Last but not least, we also follow some changes of [16] on ResNet [24] to improve the verification/ identification accuracy. We resize the input image from $299 \times 299$ to $149 \times 149$, and reduce the stride size from 2 to 1, and padding from 0 to 1 in order to remain the same size of output. In this section, the coordinates and size of informative image patches are discussed and the corresponding performances will be shown in the next section.
3.3. Research approaches

Figure 3.6: Modification of sizes of image patches. Image on the top-left side is the input image with size of $299 \times 299$. Following the process flow of [19, 50], in Inception-v3 the size of the image patches would be $17 \times 17$, and the corresponding parts of face are shown as in the first row. The second row are the corresponding parts of face when the size of image patches are enlarged to $25 \times 25$.

As for the matching strategy, we use euclidean distance of two face embeddings for verification and identification. For each public dataset [37, 58, 78], we follow their policies and compare the results with the known benchmarks.

Algorithm 1: Pseudo-code of TBE-CNN on Pytorch (one branch) without last fully-connected layer

**Input:** input image $x$, $1_{st}$ branch from $[r_{11} : r_{12}, c_{11} : c_{12}]$

```python
def forward(self, x):
    x = self.Conv2d_1a_3x3(x)
    x = self.Conv2d_2a_3x3(x)
    x = self.Conv2d_2b_3x3(x)
    x = max_pool2d(x, kernel_size=3, stride=2)
    x = self.Conv2d_3b_1x1(x)
    x = self.Conv2d_4a_3x3(x)
    x = max_pool2d(x, kernel_size=3, stride=2)
    x_1 = x[:, :, r_{11} : r_{12}, c_{11} : c_{12}]
    x = self.Mixed_5bto5d(x)
    x_1 = torch.cat((x_1, x[:, :, r_{11} : r_{12}, c_{11} : c_{12}]), 1)
    x_1 = self.branch_InceptionC(6b-7a)(x_1)
    x_1 = avg_pool2d(x_1, kernel_size=5, stride=1)
    x = self.Mixed_6ato7c(x)
    x = torch.cat((x_1, x), 1)
    x = BatchNorm(x)
    x = Dropout(x)
    x = x.view(x.size(0), -1)
    x = fc(x)
    embedding = BatchNorm(x)
    embedding_normed = L2Normalization(embedding)
    return embedding_normed
```

**Output:** Face embedding $embedding_{normed}$
Experiments and Results

In this chapter, we perform the steps starting from selecting the dataset, testing the loss functions, comparing the performances with different learning rate, which image batches to be used, benchmarks test, and final face verification, S2S/V2S identification results. For face verification (LFW, YTF, CFP-FP) tests, we assess our models using accuracy metric since the number of positive pairs and negative pairs are the same. For the face identification test (S2S and V2S), we compare the true acceptance rate under fixed false acceptance rate (P%FAR<2%). In Section 4.1, we first justify if it is feasible for us to use subset of MS1M [16, 22] for training without dropping too much performances. Section 4.2 explains which loss functions should we use for our FR tasks. After deciding the loss functions and the training dataset, we train our TBE-CNN with different branches and compare the performances in Section 4.3 to find the best model for our task. Furthermore, we also train our TBE-CNN model on CASIA dataset to compare with the benchmarks, the result in shown in Section 4.3.2. Finally, in Section 4.4 we implement the benchmarks and our trained TBE-CNN for our V2S identification test with different environmental settings (e.g. different images per person, different size of database, and different frames per person, and so on) to give a thorough understanding of our model and answer our sub-research questions.

4.1. Difference between training on MS1M subset (20k)

Since we decided to use 20,000 out of 85,000 identities for training and the rest of them for our V2S identification set. We should first justify the performance’s difference between training on these two datasets. As provided in [16], we have had the trained ResNet100 model with verification performances on LFW [37], CFP-FP [58], and YouTubeFaces [78]. Therefore, we only need to train the model on 20,000 identities. Table 4.1 shows the results of refined ResNet50 and refined R100 from [16] training on the subset of MS1M. According to the result, we can notice that only small percentage drops while training on 20,000 identities, therefore, we justify that the settings that we use is feasible (20,000 for training and the rest of them for V2S identification).

<table>
<thead>
<tr>
<th>Model</th>
<th>margin</th>
<th>MS1MV2 (20k ids)</th>
<th>MS1MV2 (85k ids)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LFW (%)</td>
<td>CFP-FP (%)</td>
</tr>
<tr>
<td>InsightFace_R100 (0, E, 3, prelu)</td>
<td>0.5</td>
<td>99.76</td>
<td>97.25</td>
</tr>
<tr>
<td>InsightFace_R50 (0, E, 3, prelu)</td>
<td>0.5</td>
<td>99.56</td>
<td>94.84</td>
</tr>
</tbody>
</table>

Table 4.1: Verification performances of refined ResNet50 and ResNet100 training on the whole MS1M dataset (85k ids) and the subset (20k ids).
4.2. Benchmarks and different loss functions

After deciding the training set and test set, we first train our model with Softmax + cross-entropy loss and then fine-tuned with some other loss function(s). According to Table 4.2 extracted from [16], we notice that training with ArcFace with proper margin \( m \) contributes the best performances. Training on ArcFace + Triplet not only has little worse results but also takes much more time for training because of the methods of selecting hard samples as triplet for training. In addition, compared with SphereFace [44] and CosFace [74], despite all methods enforce the compactness of the intra-class and the diversity of inter-class, ArcFace has constant linear angular decision margin (Fig. 5 in [16]) which is different from them, and has better verification performance. Based on this result, we decided to implement ArcFace with different margin for our task. In the later section we will compare the results with different margin selection.

<table>
<thead>
<tr>
<th>Loss Functions</th>
<th>LFW</th>
<th>CFP-FP</th>
<th>AgeDB-30</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArcFace [16] (0.4)</td>
<td>99.53</td>
<td>95.41</td>
<td>94.98</td>
</tr>
<tr>
<td>ArcFace (0.45)</td>
<td>99.46</td>
<td>95.47</td>
<td>94.93</td>
</tr>
<tr>
<td>ArcFace (0.5)</td>
<td><strong>99.53</strong></td>
<td>95.56</td>
<td><strong>95.15</strong></td>
</tr>
<tr>
<td>ArcFace (0.55)</td>
<td>99.41</td>
<td>95.32</td>
<td>95.95</td>
</tr>
<tr>
<td>SphereFace [44]</td>
<td>99.42</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CosFace [74]</td>
<td>99.33</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Triplet (0.35)</td>
<td>98.98</td>
<td>91.90</td>
<td>89.98</td>
</tr>
<tr>
<td>ArcFace + Triplet</td>
<td>99.50</td>
<td>95.51</td>
<td>94.40</td>
</tr>
<tr>
<td>Softmax</td>
<td>99.08</td>
<td>94.39</td>
<td>92.33</td>
</tr>
<tr>
<td>Norm-Softmax</td>
<td>98.56</td>
<td>89.79</td>
<td>88.72</td>
</tr>
</tbody>
</table>

Table 4.2: Verification results (%) of different loss functions trained on CASIA Face with ResNet50. 
"-" denotes tests which were not mentioned in the paper. The table is extracted from [16].

4.3. TBE-CNN Performances

Initial model and adjusting the batch size

We first train the model with initial learning rate of \( \alpha = 0.01 \), with batch size \( B = 32 \) and we apply dropout with probability \( p = 0.5 \). The model is trained for around \( k = 30 \) epochs. Before training, we separate the dataset into training set and validation set. The validation set contains one image per person for the model to identify. A plot of the loss function per mini-batch for the initial model is given in Fig. 4.1. According to most of the research and previous papers [16, 19, 50], using larger batch size will give better performances. However, since the limited memory usage of our GPU, we can only set the batch size to 32 for our TBE-CNN training, and 64 for training the ResNet.

Adjusting learning rate (lr) and dropout probability \( p \)

Next we aim to find proper learning rate and dropout probability for training. By setting larger learning rate as the first stage, the model can escape from plateau region and the time for training will be reduced. However, if the learning rate is too large, it is hard for the model to converge. We compare two learning rates, 0.05 and 0.01 to find out which learning rate should be used in our further tests. The validation losses are shown as in Fig. 4.2, and we can notice that the validation loss with lr= 0.05 is much higher than the one with lr= 0.01. Therefore, we get the conclusion that using lr= 0.01 in our research would be better. While training, the learning rate will decrease based on trained epochs and the change of loss to achieve better results.

As for the selection of dropout probability \( p \), we started with 0.5, and decided to increase/decrease based on the training and validation accuracies. We notice that \( p \) is different depending on the model. Since we have many versions to compare, \( p \) is set differently based on the best results we got. Generally, \( p \) is set to 0.3- 0.5 to obtain best performances in our refined TBE-CNN structures.
4. Experiments and Results

Figure 4.1: Trends of training loss with batch size $B = 32$, and learning rate $\alpha = 0.01$.

Figure 4.2: The trends of loss with different starting learning rate.

4.3.1. Performances on LFW and CFP-FP

After deciding some hyper-parameters for training, we begin training our TBE-CNN with different proposed branches. We compare the verification results of our model on LFW and CFP-FP with different state-of-the-art models (ResNet50 [16], ResNet100 [16], Inception-v3 [68]) previous TBE-CNN research [19]. The models are trained on the subset (20k ids) of MS1MV2 dataset. The results are shown in Table 4.3.

<table>
<thead>
<tr>
<th>Model</th>
<th>margin</th>
<th>LFW (%)</th>
<th>CFP-FP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>InsightFace_R100 (0, E, 3, prelu)</td>
<td>0.5</td>
<td>99.76</td>
<td>97.25</td>
</tr>
<tr>
<td>InsightFace_R50 (0, E, 3, prelu)</td>
<td>0.5</td>
<td>99.56</td>
<td>94.84</td>
</tr>
<tr>
<td>Inceptionv3 (I)</td>
<td>0.5</td>
<td>99.40</td>
<td>94.75</td>
</tr>
<tr>
<td>TBECNN_V2 (I+U+C)</td>
<td>0.5</td>
<td>99.43</td>
<td>95.42</td>
</tr>
<tr>
<td>TBECNN_V2 (I+U+C)</td>
<td>0.6</td>
<td>99.49</td>
<td>95.12</td>
</tr>
<tr>
<td>TBECNN_V5 (I+H)</td>
<td>0.3</td>
<td>99.59</td>
<td>96.00</td>
</tr>
<tr>
<td>TBECNN_V5 (I+H)</td>
<td>0.5</td>
<td>99.70</td>
<td>96.25</td>
</tr>
<tr>
<td>TBECNN_V5 (I+H)</td>
<td>0.6</td>
<td>99.63</td>
<td>96.54</td>
</tr>
<tr>
<td>TBECNN_V7 (I+H+U)</td>
<td>0.5</td>
<td>99.66</td>
<td>96.58</td>
</tr>
<tr>
<td>TBECNN_V7 (I+H+U)</td>
<td>0.6</td>
<td>99.69</td>
<td>96.71</td>
</tr>
<tr>
<td>TBECNN_V5RS (I+H)</td>
<td>0.5</td>
<td>99.70</td>
<td>97.04</td>
</tr>
<tr>
<td>TBECNN_V5RS (I+H)</td>
<td>0.6</td>
<td>99.63</td>
<td>96.32</td>
</tr>
<tr>
<td>TBECNN_V7RS (I+H+U)</td>
<td>0.5</td>
<td>99.53</td>
<td>96.01</td>
</tr>
<tr>
<td>TBECNN_V7RS (I+H+U)</td>
<td>0.6</td>
<td>99.55</td>
<td>95.68</td>
</tr>
</tbody>
</table>

Table 4.3: Verification performances of TBE-CNN with different branches trained with ArcFace [16] as loss function and MS1MV2 (20k ids) as training dataset.

According to the results, we can notice that Inception-v3 performs better than ResNet50 and is a bit worse than ResNet100, and by adding branches, the verification accuracy gets closer and closer to ResNet100. Moreover, we can easily see that the network V2 with holistic face + upper part of the head (U) + lower part of the head (C) performs worse than the network V7 containing holistic face + upper part only. However, the network V5 with input image and holistic face only as the branch has really similar result comparing with V7, and V5 has one branch less than V7, the parameters are fewer and can be trained faster.

Since the performances of our current TBE-CNN V5 and V7 model have similar performances to ResNet100, we tend to further tune our model by changing the last layer becoming our face embeddings and by changing
the size of the image patch (from 17×17 to 25×25). We first test the performances of using 1. Maxpooling layer, 2. Averaged pooling layer, 3. fully-connected layer, then consider the size of the image patch. Table 4.4 indicates the difference between changing the layer. It seems like using fully connected layer to map the features into face embeddings will result in better verification performance. Next, we compare the results of two different patch size, which is shown in Table 4.5.

<table>
<thead>
<tr>
<th>Model</th>
<th>margin</th>
<th>LFW (%)</th>
<th>CFP-FP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBECNN_V5RS_AVGPooling</td>
<td>0.5</td>
<td>99.64</td>
<td>95.80</td>
</tr>
<tr>
<td>TBECNN_V5RS_MaxPooling</td>
<td>0.5</td>
<td>99.68</td>
<td>96.82</td>
</tr>
<tr>
<td>TBECNN_V5RS_FC</td>
<td>0.5</td>
<td>99.70</td>
<td>97.04</td>
</tr>
</tbody>
</table>

Table 4.4: Verification performances of TBE-CNN with different last layers as our features. The models are trained on subset (20k ids) of MS1MV2 (85k ids).

<table>
<thead>
<tr>
<th>Model</th>
<th>margin</th>
<th>LFW (%)</th>
<th>CFP-FP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBECNN_V5RS_17×17</td>
<td>0.5</td>
<td>99.61</td>
<td>96.01</td>
</tr>
<tr>
<td>TBECNN_V5RS_25×25</td>
<td>0.5</td>
<td>99.63</td>
<td>96.62</td>
</tr>
</tbody>
</table>

Table 4.5: Verification results (%) of different patch size. The models are trained on subset (20k ids) of MS1MV2 (85k ids).

### 4.3.2. Performances of different methods (models)

Since LFW [37], and YTF [78] datasets are the most widely used benchmark for unconstrained face verification on images and videos, we follow the *unrestricted with labelled outside data* protocol to report the performance comparing with different method and datasets. As reported in Table 4.6, we first notice that training on 20k identities (1.3M images) of MS1MV2 will result in 1.67% worse accuracy on YTF. However, comparing with other open-sourced face recognition models trained on smaller dataset, ResNet100, ResNet50, and our model performs really well. This lets us understand that the accuracy could be increased by training on larger dataset. Although our model is 0.04% lower than ResNet50 on YTF, in LFW and CFP-FP dataset we still outperforms ResNet50 (Table 4.3) and our TBE-CNN Network with tuned input size and branches (V5RS) has higher accuracies than previous TBE-CNN research.

<table>
<thead>
<tr>
<th>Method</th>
<th>#Image</th>
<th>LFW (%)</th>
<th>YTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepID [63]</td>
<td>0.2M</td>
<td>99.47</td>
<td>93.20</td>
</tr>
<tr>
<td>Deep Face [70]</td>
<td>4.4M</td>
<td>97.35</td>
<td>91.4</td>
</tr>
<tr>
<td>VGG Face [51]</td>
<td>2.6M</td>
<td>98.95</td>
<td>97.30</td>
</tr>
<tr>
<td>FaceNet [56]</td>
<td>200M</td>
<td>99.63</td>
<td>95.10</td>
</tr>
<tr>
<td>Baidu [1]</td>
<td>1.3M</td>
<td>99.13</td>
<td>-</td>
</tr>
<tr>
<td>Center Loss [76]</td>
<td>0.7M</td>
<td>99.28</td>
<td>94.9</td>
</tr>
<tr>
<td>SphereFace [44]</td>
<td>0.5M</td>
<td>99.42</td>
<td>95.0</td>
</tr>
<tr>
<td>CosFace [74]</td>
<td>5M</td>
<td>99.73</td>
<td>97.6</td>
</tr>
<tr>
<td>MS1MV2, R100, ArcFace [16]</td>
<td>5.8M</td>
<td>99.83</td>
<td>98.02</td>
</tr>
<tr>
<td>ORTEC Implementation [2, 21]</td>
<td>0.5M</td>
<td>99.38</td>
<td>-</td>
</tr>
<tr>
<td>TBE-CNN [19]</td>
<td>0.49M/ 2.6M / 5.2M</td>
<td>99.08</td>
<td>94.96</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>#Image</th>
<th>LFW (%)</th>
<th>YTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS1MV2 (20k), ResNet100, ArcFace</td>
<td>1.3M</td>
<td>99.76</td>
<td>96.35</td>
</tr>
<tr>
<td>MS1MV2 (20k), ResNet50, ArcFace</td>
<td>1.3M</td>
<td>99.56</td>
<td>96.08</td>
</tr>
<tr>
<td>MS1MV2 (20k), TBE-CNNV5RS, ArcFace</td>
<td>1.3M</td>
<td>99.70</td>
<td>96.04</td>
</tr>
<tr>
<td>MS1MV2 (20k), TBE-CNN*, ArcFace</td>
<td>1.3M</td>
<td>99.51</td>
<td>94.30</td>
</tr>
</tbody>
</table>

Table 4.6: Verification performances (%) of different methods on LFW and YTF. The performances of ResNet100 and ResNet50 on MS1MV2 (20k) are trained by ourselves with provided model through given MXNet code [16]. As for the result of previous TBE-CNN [19] network (the one marked with *), we select the proper branches by ourselves since the lack of information from the paper. "-" denotes tests which were not mentioned in the paper.
4.4. Performances on V2S identification dataset

Next, after the comparisons of face verification, we implement these current best models on COX V2S identification set [29], we follow their train/test protocol for deciding the threshold and the rank-1 face identification accuracy of the video probes. Finally, we implement the models into our mock V2S identification sets with 55,000+ identities in the gallery and different number of frames per video clip as probes, which allows us to get a thorough understanding about the preferred settings and the performances of the models.

4.4.1. COX V2S identification

<table>
<thead>
<tr>
<th>Model</th>
<th>margin</th>
<th>COX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>V1-S</td>
</tr>
<tr>
<td>TBE-CNN_paper</td>
<td>-</td>
<td>93.57</td>
</tr>
<tr>
<td>Haar-like TBE-CNN</td>
<td>-</td>
<td>92.73</td>
</tr>
<tr>
<td>Haar-like TBE-CNN-FT*</td>
<td>-</td>
<td>98.26</td>
</tr>
<tr>
<td>InsightFace, ResNet100</td>
<td>0.5</td>
<td>98.52</td>
</tr>
<tr>
<td>InsightFace, ResNet50</td>
<td>0.5</td>
<td>96.69</td>
</tr>
<tr>
<td>TBE-CNN_V5RS</td>
<td>0.5</td>
<td>98.21</td>
</tr>
<tr>
<td>TBE-CNN_V5</td>
<td>0.5</td>
<td>93.2</td>
</tr>
<tr>
<td>TBE-CNN_V7</td>
<td>0.6</td>
<td>94.9</td>
</tr>
</tbody>
</table>

Table 4.7: Rank-1 identification results on COX Video-to-still identification test, note that the ResNet100, ResNet50, and TBE-CNN models in the bottom are trained on MS1MV2 (20k ids) instead of the whole 85k ids. **" denotes the model fine-tuned with their preprocessing methods, which could also be implemented into all other models.

As reported in Table 4.7, we could notice that all our trained models are better than the other previous V2S face recognition models. It is worth to see that in the third row, the fine-tuned Haar-like TBE-CNN is evaluated as the top-ranked model. According to the paper [50], the objective of this process is to train the network in order to acquire knowledge about similarities based on the augmented images. We consider it as a preprocessing step and could be implemented into all other models for training. Since our models did not add this fine-tuning step, comparing the result in the second row would be fair. As a result, we get a conclusion that our TBE-CNN_V5RS model performs better than refined ResNet50 and is slightly worse than refined ResNet100. We further move to the test of our large class, few-images V2S face recognition task, which is the main focus of our research.

4.4.2. Proposed large class, few images V2S identification set

After deciding our proper branches and hyper-parameters, we compare the performances in our large class, few images V2S identification set. As mentioned in Section 2.4.5, here we fixed the false acceptance rate $FAR \leq 2\%$, and compare the precision of identifying learned people to know which model performs better ($P\%@FAR \leq 2\%$). In the following, we tend to know the trend of different settings (e.g. number of image per person, and number of frames in average). First we will fix the size of the gallery to find the proper number of image in both the gallery and probes, then we use this settings to observe the trend with different size of gallery.

Performances under different number of image per person (p.p.)

Here we consider three different settings in the gallery: 1 image p.p., 2-5 images p.p., and 6-10 images p.p., and also three settings in the probe: averaging 20 frames, 50 frames, and 100 frames per video as face embeddings. The reason why we only consider 1 image p.p is that in the real case of our implementation, it is quite abnormal to ask people to upload too many photos for learning. Performances of ResNet50, ResNet100 and our TBE-CNN with proper branches are shown as in Table 4.8, 4.9, and 4.10 respectively.

As we can see in the tables, using more images in database and in tests result in better precision. This phenomenon is expected since the machine can learn variances from the images and identify the person well.
4.4. Performances on V2S identification dataset

### V2S Identification (P%@FAR ≤ 0.02)

<table>
<thead>
<tr>
<th>Model</th>
<th>Database</th>
<th>20 frames</th>
<th>50 frames</th>
<th>100 frames</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Insight_R50_ArcFace</strong></td>
<td>DB, 1 image p.p.</td>
<td>55.8</td>
<td>71.8</td>
<td>80.2</td>
</tr>
<tr>
<td></td>
<td>DB, 2-5 image p.p.</td>
<td>82.2</td>
<td>90.0</td>
<td>95.0</td>
</tr>
<tr>
<td></td>
<td>DB, 6-10 image p.p.</td>
<td>86.2</td>
<td>92.8</td>
<td>95.6</td>
</tr>
</tbody>
</table>

Table 4.8: V2S identification result of ResNet50 [16] in different settings. The rows are the setting of different image(s) we have per person in the database (gallery), and the columns denote how many frames (or seconds, usually 30 frames per second) we use per person to get an averaged face embedding.

<table>
<thead>
<tr>
<th>Model</th>
<th>Database</th>
<th>20 frames</th>
<th>50 frames</th>
<th>100 frames</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Insight_R100_ArcFace</strong></td>
<td>DB, 1 image p.p.</td>
<td>68.0</td>
<td>78.4</td>
<td>84.8</td>
</tr>
<tr>
<td></td>
<td>DB, 2-5 image p.p.</td>
<td>86.6</td>
<td>91.2</td>
<td>95.2</td>
</tr>
<tr>
<td></td>
<td>DB, 6-10 image p.p.</td>
<td>88.4</td>
<td>94.6</td>
<td>97.0</td>
</tr>
</tbody>
</table>

Table 4.9: V2S identification result of ResNet100 [16] in different settings. The rows are the setting of different image(s) we have per person in the database (gallery), and the columns denote how many frames (or seconds, usually 30 frames per second) we use per person to get an averaged face embedding.

<table>
<thead>
<tr>
<th>Model</th>
<th>Database</th>
<th>20 frames</th>
<th>50 frames</th>
<th>100 frames</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TBECNN_V5RS_ArcFace</strong></td>
<td>DB, 1 image p.p.</td>
<td>59.0</td>
<td>75.0</td>
<td>83.2</td>
</tr>
<tr>
<td></td>
<td>DB, 2-5 image p.p.</td>
<td>80.8</td>
<td>93.2</td>
<td>95.8</td>
</tr>
<tr>
<td></td>
<td>DB, 6-10 image p.p.</td>
<td>88.4</td>
<td>93.0</td>
<td>97.6</td>
</tr>
</tbody>
</table>

Table 4.10: V2S identification result of our resized TBE-CNN5 in different settings. The rows are the setting of different image(s) we have per person in the database (gallery), and the columns denote how many frames (or seconds, usually 30 frames per second) we use per person to get an averaged face embedding.

However, as mentioned before, it is quite unrealistic to request or obtain many images from a person. Hence the performances of one or few images per person are important for implementations where few images are available. When having only one image in the database (the row with DB, 1 image p.p.), our model is 3% higher than ResNet50 and only 1.6% lower than ResNet100. Given these tables, we suggest that in our real implementation, we can request our clients to upload just one photo to us, and we have up to 80% accuracy to identify them well, which is really interesting for us to know. We further show the ROC curves of each model under the best setting (1 image in gallery and averaging 100 frames) to get a thorough understanding of their behaviors, which is shown in Fig. 4.3.

### Performances under different size of saved identities

As reported in the previous experiments, we obtain the insight that the models have more than 80% precision to identify the our learned people within FAR ≤ 2%. Here we further would like to know the stability (scalability) of the model. We begin with 1,000 people in our gallery, and gradually increase the size of the gallery to 55,000. The FAR rate is still fixed at 2% and we compare the rank-1 identification rates. A model is considered as poor scalability if its precision drops dramatically when the size of gallery increases. As reported in Fig. 4.4, we notice that no matter how large the size of the gallery is, our trained models have really stable identification performances. Therefore, we get a conclusion that we are able to implement this into our real-world V2S identification.
4.5. Performances on S2S identification dataset

In addition to V2S identification, we also create a S2S test, which uses 55,000 people as database (same identities in V2S test), but another 1,000 unseen identities and 1,000 seen identities as probes. Similar to V2S identification, the database has three settings: 1 image, 2 to 5 images, and 6 to 10 images p.p., and the probe has two settings: 1 to 5, and 6 to 10 images p.p.. The threshold for identifying seen people is selected under $\text{FAR} \leq 2\%$ and $1\%$. We would get a full perspective about the merits of each model and what the preferred setups for practical S2S identification. The results are shown in Table 4.11, 4.12, 4.13.

Based on the results, we notice that having more images per person in both probes and database would result in better performances. However, for practical FR implementations, sometimes it is hard to request many images from a person. In the situation of having only few images (or even one image) per person in both probes and database (DB_1, DB_2to5 with probes_1), our TBE-CNN model has better performance than ResNet100 and ResNet50. For example, considering FAR $\leq 1\%$, and 1 to 5 images per person as probes, our model has precision of 81.80% and 92.30% when having 1 image and 2 to 5 images per person in the database. These are better than 79.90%, 89.10% in ResNet100, and 76.00%, 92.10% in ResNet50. Moreover, when FAR$\leq 0.5\%$, ResNets have precision lower than 60% when there is only one image as database and one
image as test (DB_1, Probes_1). Compared with them, our model has precision of 67.5%, which is much more better than ResNets. Therefore, according to these few shots tests, we conclude that our model is a better choice than ResNets in few-shots, large classes S2S identification.

<table>
<thead>
<tr>
<th>Insight_R50_ArcFace</th>
<th>Probes_1</th>
<th>Probes_2to5</th>
<th>Probes_6to10</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_1</td>
<td>74.70%</td>
<td>88.50%</td>
<td>92.30%</td>
</tr>
<tr>
<td>DB_2to5</td>
<td>86.70%</td>
<td>96.90%</td>
<td>99.00%</td>
</tr>
<tr>
<td>DB_6to10</td>
<td>92.30%</td>
<td>99.30%</td>
<td>99.90%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P%@FAR≤0.02</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Insight_R50_ArcFace</th>
<th>Probes_1</th>
<th>Probes_2to5</th>
<th>Probes_6to10</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_1</td>
<td>67.70%</td>
<td>82.30%</td>
<td>87.70%</td>
</tr>
<tr>
<td>DB_2to5</td>
<td>80.10%</td>
<td>92.70%</td>
<td>97.60%</td>
</tr>
<tr>
<td>DB_6to10</td>
<td>88.50%</td>
<td>97.10%</td>
<td>99.90%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P%@FAR≤0.01</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Insight_R50_ArcFace</th>
<th>Probes_1</th>
<th>Probes_2to5</th>
<th>Probes_6to10</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_1</td>
<td>58.40%</td>
<td>73.30%</td>
<td>77.00%</td>
</tr>
<tr>
<td>DB_2to5</td>
<td>74.60%</td>
<td>88.60%</td>
<td>93.80%</td>
</tr>
<tr>
<td>DB_6to10</td>
<td>84.40%</td>
<td>95.10%</td>
<td>98.90%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P%@FAR≤0.005</th>
</tr>
</thead>
</table>

Table 4.11: S2S identification result of ResNet50 [16] in different settings

<table>
<thead>
<tr>
<th>Insight_R100_ArcFace</th>
<th>Probes_1</th>
<th>Probes_2to5</th>
<th>Probes_6to10</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_1</td>
<td>75.80%</td>
<td>90.40%</td>
<td>92.90%</td>
</tr>
<tr>
<td>DB_2to5</td>
<td>86.80%</td>
<td>97.20%</td>
<td>99.10%</td>
</tr>
<tr>
<td>DB_6to10</td>
<td>93.10%</td>
<td>99.20%</td>
<td>99.90%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P%@FAR≤0.02</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Insight_R100_ArcFace</th>
<th>Probes_1</th>
<th>Probes_2to5</th>
<th>Probes_6to10</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_1</td>
<td>69.70%</td>
<td>85.70%</td>
<td>89.50%</td>
</tr>
<tr>
<td>DB_2to5</td>
<td>78.10%</td>
<td>94.60%</td>
<td>98.30%</td>
</tr>
<tr>
<td>DB_6to10</td>
<td>88.20%</td>
<td>97.10%</td>
<td>99.80%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P%@FAR≤0.01</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Insight_R100_ArcFace</th>
<th>Probes_1</th>
<th>Probes_2to5</th>
<th>Probes_6to10</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_1</td>
<td>54.50%</td>
<td>73.20%</td>
<td>75.60%</td>
</tr>
<tr>
<td>DB_2to5</td>
<td>65.30%</td>
<td>83.90%</td>
<td>91.40%</td>
</tr>
<tr>
<td>DB_6to10</td>
<td>75.70%</td>
<td>93.90%</td>
<td>98.90%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P%@FAR≤0.005</th>
</tr>
</thead>
</table>

Table 4.12: S2S identification result of ResNet100 [16] in different settings
### 4.6. Insights and answers to research questions

Based on the experiments above, we provide some insights and answer the research questions:

**Improvement of video-to-still few images, large class's face recognition**

During the experiment, we notice that pose-variance and occlusions still affect the performances of current state-of-the-art models. After a literature survey, we decided to implement TBE-CNN with best branch(es) and better trunk model to reduce these effects. The idea of this architecture is to let the model learn how to identify learned people based on both holistic image and partial faces (image patches). It is an end-to-end model with patches cropped inside the model.

Compared with state-of-the-art models for V2S tests (previous TBE-CNN networks [19, 50]), the accuracy in COX V2S test is improved by using refined ResNet100 [16] and our refined TBE-CNN network with ArcFace as loss function (Table 4.7). Moreover, although in our created V2S tests our refined TBE-CNN model performs a bit worse than refined ResNet100 [16], our model outperforms both refined ResNet100 and refined ResNet50 in S2S test under practical settings (e.g. DB_1, Probes_1). These results further prove our idea that using proper image patches as branches could find more discriminative face features for FR.

**Preferred branch(es) for our tasks**

According to the results in Table 4.3, we notice that adding holistic face as the first branch improves the performance a lot (from 94.75% to 97.04% in CFP-FP). However, adding upper/bottom part of the face as the second not only has little effect on the verification results but also increases the computational time a lot, we conclude that using holistic face in TBE-CNN is preferred for both V2S and S2S FR.

**Performances on real-world FR**

Based on the results of our created V2S and S2S tests, we realize that using more images per person in both database and test (probe) results in better performances.

For large scale V2S FR with one image per person in the database, it is preferred to use more frames (i.e. more seconds) for test since the precision differs a lot (at least 15%). In this case (DB_1, video_avg100), under FAR ≤ 2%, we have over 80% precision to identify our learned people. In our created S2S test, in cases where FAR ≤ 0.5%, both refined ResNet100 and ResNet50 have precision lower than 60% when having one image per person and one image per test (Table 4.11, 4.12). In comparison, our refined TBE-CNN has 67.5%, which greatly outperforms them. Furthermore, our model also performs better under any FAR.

**Table 4.13: S2S identification result of our resized TBE-CNN in different settings**

<table>
<thead>
<tr>
<th>TBE-CNN_V5RS_ArcFace</th>
<th>Probes_1</th>
<th>Probes_2to5</th>
<th>Probes_6to10</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_1</td>
<td>77.70%</td>
<td>91.00%</td>
<td>93.50%</td>
</tr>
<tr>
<td>DB_2to5</td>
<td>87.60%</td>
<td>97.90%</td>
<td>99.10%</td>
</tr>
<tr>
<td>DB_6to10</td>
<td>93.30%</td>
<td>99.50%</td>
<td>99.90%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TBE-CNN_V5RS_ArcFace</th>
<th>Probes_1</th>
<th>Probes_2to5</th>
<th>Probes_6to10</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_1</td>
<td>71.50%</td>
<td>79.50%</td>
<td>89.00%</td>
</tr>
<tr>
<td>DB_2to5</td>
<td>77.20%</td>
<td>94.30%</td>
<td>98.50%</td>
</tr>
<tr>
<td>DB_6to10</td>
<td>84.90%</td>
<td>97.50%</td>
<td>99.70%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TBE-CNN_V5RS_ArcFace</th>
<th>Probes_1</th>
<th>Probes_2to5</th>
<th>Probes_6to10</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_1</td>
<td>67.50%</td>
<td>68.50%</td>
<td>73.90%</td>
</tr>
<tr>
<td>DB_2to5</td>
<td>73.10%</td>
<td>86.90%</td>
<td>94.00%</td>
</tr>
<tr>
<td>DB_6to10</td>
<td>78.90%</td>
<td>93.50%</td>
<td>99.00%</td>
</tr>
</tbody>
</table>
As for the scalability, owing to the low decreasing rates, we have the conclusion that the models can identify people well even if we have 55,000 people saved in the database. Moreover, the performances could also be higher if we train the models on larger datasets since we only trained the models on 20,000 identities.
Conclusions and Recommendations

In this research, we provided a large-scale, few shot video-to-still (still-to-still) face recognition dataset to understand the performances of current state-of-the-art models and preferred settings for implementing face recognition system in the real-world. The dataset contains over 55,000 identities as database, and uses people who have images in the database (i.e. target people) and people who have no images in the database (i.e. non-target people) to answer the questions of the scalability of a model and precision under certain false acceptance rate. In addition, the dataset uses different number of images/frames per person for both database and test (probe) to find out which model outperforms others in what settings. As a result, we notice that having more images per person in the database and using more images/frames for identification contributes better results. In the real case where only one image per person is available in the database, using current state-of-the-art models (refined ResNet100 [16]) would have over 80% precision in V2S identification and stable performances even if there is 55k identities in the database for matching. However, the model only has precision lower than 60% when FAR ≤ 0.5% with only one image per person in database and test (S2S).

Our second contribution is to use a refined TBE-CNN architecture to further improve the performance. By using image patches such as holistic face and upper face as internal inputs to the model, the effect of pose-variant and occlusions is reduced and helps us obtaining better performances in V2S and S2S FR. After different tests, we conclude that using holistic face as branch has the best performances. To compare with current state-of-the-art models (refined ResNets) and models from previous literature [19, 50], our model has precision of 63.20% under FAR ≤ 2% in V2S identification test when there are 55,000 people in the database, which is similar to ResNet100 (84.80%) and better than ResNet50 (80.20%). As for S2S identification, when we have one image per person in the database and one image per person to identify, our refined TBE-CNN has precision of 67.5%, 71.5%, and 77.5% under different FAR, which outperforms ResNet100 (54.5%, 69.70%, 75.80%) and ResNet50 (58.40%, 67.70%, 74.70%).

Compared with ORTEC’s system

The original ORTEC’s system is from [21, 56], which used GoogleNet [66] (can be considered as InceptionNet-v1) as DCNN and trained the model on Facescrub [49]+ CASIA-WebFace [79]. The model has 99.38% on LFW [37] verification, and computational time of the system is around 13-22ms with GPU. However, the detector cannot really deal with images containing pose-variant and/or occlusion. In comparison, current state-of-the-art models use MTCNN [80] as face detector, which can detect faces under aforementioned issues. The refined TBE-CNN model (InceptionNet-v3 [68] as trunk network) and ResNets [16] have better performances on LFW (over 99.70%) but has longer computational time (around 100 ms in MTCNN face detector, around 70 ms in TBE-CNN model and 36 ms in ResNets with GPU). It can be faster if we feed the images to the networks in batches. When passing one image at a time, the original ORTEC system can use more frames per second for analysis. However, when taking the identification rate into consideration, we should use current models (ResNets100, ResNets50, TBE-CNN) for real-world’s large scale FR with few images per person. It will be interesting if the model can eventually be tested in a real-world application.
5.1. Further research

As topic for further research, some other works could be done and be further investigated:

Real-world applications, and tests on some public dataset for comparison

In our created large-scale S2S identification dataset, we have proved that our refined TBE-CNN model outperforms current state-of-the-art models in the case where we have one image per person saved in the database and one image per person for test. It is interesting to know the performance under the real-world application and how to improve further improve the model. Moreover, we have tested the performances of the models based on LFW [37], CFP-FP [58], and COX [29]. It would be also interesting to compare the performances of our model on other public dataset with other state-of-the-art models.

Hyper-parameters tuning

In our research, the models were trained on subset of MS1MV2 [22], which contains only 20k identities. The rests of the images were used to create our identification test set. The performance could be higher if the models are trained on more images and further improve the identification rate. In addition, owing to the limited computational power, our TBE-CNN models were trained with batch size $B = 32$, the performances would be higher if the models are trained with $B = 64$ or 128 by using multiple GPUs.

Last but not least, since we currently use simple Euclidean distance to find the matched identity. It will be compelling if we can develop a better matching strategy for real-world's face recognition.
References


References


