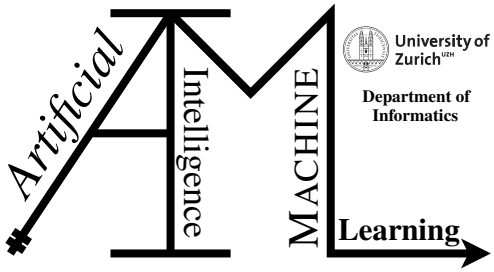
Facial Video Recognition via 3D Convolutional Networks

Master Thesis

Xinyi Zhang
19-772-235

Submitted on
February 1, 2021

Thesis Supervisor
Prof. Dr. Manuel Günther



Master Thesis

Author: Xinyi Zhang, xinyi.zhang@uzh.ch

Project period: August 9, 2021 - February 1, 2022

Artificial Intelligence and Machine Learning Group
Department of Informatics, University of Zurich

Acknowledgements

I thank Prof. Dr. Manuel Günther's supervision on the entire project. He gave me lots of guidance and feedback quickly and effectively during the process of my thesis.

Thank Dr. Tiago de Freitas Pereira and the resources given by the UZH to be able to complete my code part of the thesis.

I thank my parents for the support of my master's study in Zürich during the COVID-19 pandemic. Without them, I can not finish my master's degree successfully.

Abstract

Face recognition has been popular in the video recently. As the development of deep learning, various CNNs models are implemented into face recognition such as ResNet, MobileNet, MobileFaceNet. During this experiment, we verified that the light CNN model – Stacked2D, and 3D MobileFaceNet can extract features from several frames at the same time on the video dataset (YoutubeFaces). First, the baseline model – the original 2D MobileFaceNet combined ArcFace loss function model is trained from the face recognition task. Then, this model is implemented as the feature extractor in *bob* framework, which can construct a face recognition pipeline easily. Using the same process, the Stacked2D and 3D MobileFaceNet models with Arcface are trained using YTF dataset. In the end, we run the video recognition pipeline in *bob* framework and compare the results using different models. In this experiment, we verify that it is feasible to use 2D, Stacked2D, and 3D MobileFaceNet models in video face recognition, and the model with larger frames input can perform better because it can capture more spatial and temporal information from video data.

Zusammenfassung

Die Gesichtserkennung war in letzter Zeit im Video beliebt. Mit der Entwicklung von Deep Learning werden verschiedene CNNs Modelle in die Gesichtserkennung implementiert, wie zum Beispiel ResNet, MobileNet und MobileFaceNet. Während dieses Experiments haben wir verifiziert, dass das leichte CNN Modell wie Stacked2D und 3D MobileFaceNet die Merkmale aus mehreren Frames gleichzeitig auf dem Videodatensatz (YoutubeFaces) extrahieren kann. Zuerst wird das Basislinienmodell, die Kombination vom originalen 2D MobileFaceNet und ArcFace Verlustfunktionsmodell, aus der Gesichtserkennungsaufgabe trainiert. Dann wird dieses Modell als Merkmalsextrahierer im Bob Framework implementiert, das auf einfache Weise eine Gesichtserkennungspipeline erstellen kann. Mit dem gleichen Verfahren werden die Stacked2D und 3D MobileFaceNet Modelle mit Arcface unter Verwendung des YTF-Datensatzes trainiert. Am Ende führen wir die Videoerkennungspipeline im *bob* Framework aus und vergleichen die Ergebnisse mit verschiedenen Modellen. In diesem Experiment verifizieren wir, dass es möglich ist, 2D, Stacked2D und 3D MobileFaceNet Modelle in der Video-Gesichtserkennung zu verwenden. Dieses Modell mit der Eingabe größerer Frames kann eine bessere Leistung erbringen, da es mehr räumliche und zeitliche Informationen aus Videodaten erfassen kann.

Contents

Chapter 1

Introduction

Recently, face recognition is one of the most popular areas in computer vision, which has been applied in a variety of domains. The purpose of face recognition is to extract features of a person from the face image and use this to identify the person's identity. There are many techniques and algorithms applied in the face recognition area, especially as deep learning develops.

Based on the availability of large-scale training data and the rapid development of deep learning methods, face recognition from 2D images has achieved very good recognition results. Many (surveillance) videos, however, do not feature high-resolution photos, and face sizes are modest. Similarly, recognition performances on video datasets like YouTubeFaces (Martinez-Diaz et al., 2018) or IJB-C (Maze et al., 2018) do not match still picture performance. Videos naturally provide more information than a single image. And many 3D networks have been effectively applied to Computer Vision applications such as human action recognition (Ji et al., 2013), object identification (Singh et al., 2019), picture segmentation (Mashiko, 2020), and face recognition (Mishra and Singh, 2021).

1.1 Motivation

As Mishra and Singh (2021) described, for face recognition in video, they represented the video as a set of frames. They used a set of frames instead of the original video data and selected some frames from a video as input. To use multiply frames of the video as input, Mishra and Singh (2021) extended the 2D DenseNets model into 3D, which can process 16 frames images at the same time.

Mishra and Singh (2021) think that compared with traditional 2D CNNs models, the 3D models can capture the spatial and temporal-domain information at the same time. In their experiment, the 3D DenseNets designed for video recognition can achieve 97% accuracy on their dataset (CVBL). It proves that it is possible to apply the 3D neural network to video face recognition.

1.2 Description of Work

In this thesis, the main task is to use the YouTubeFaces dataset to train different CNNs structures of MobileFaceNet with ArcFace: 2D model, Stacked2D model, and 3D model in the face recognition task. Then we compare the models for the loss and accuracy during training. Finally, we propose to learn spatio-temporal features using deep Stacked2D and 3D Convolution Neural Networks, implement these models as feature extractors, run the face recognition pipeline based on the *bob* environment and compare their results.

1.3 Thesis Outline

The thesis is structured as follows:

In Chapter 2, we will give an overview of related work regarding the procedure and techniques in face recognition including 2D and 3D.

In Chapter 3, there is some description of methods and approaches which are used during this experiment, including the YouTubeFaces dataset and its protocol, theory of MobileFaceNet and ArcFace, the pipeline in bob framework, and some optimization methods.

In Chapter 4, we would like to show the results of models for classification tasks and face recognition pipeline in the bob framework. Besides, we compare these results and do an analysis.

In Chapter 5, we will discuss the results of this experiment, analysis the results, and provide some insights into areas where my research can be further improved in the future.

In Chapter 6, we would like to give a total conclusion for this master's experiment.

Chapter 2

Related work

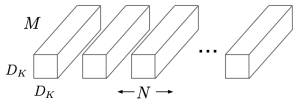
2.1 Deep Architectures for Face Recognition

2.1.1 Face Recognition for Image

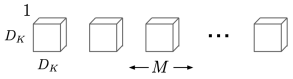
As we talked before, there are many neural networks have been effectively applied to Computer Vision applications, for example, face recognition (Liu et al., 2021), object identification (Cai et al., 2016), and image classification (Russakovsky et al., 2015). The popularity of Convolution Neural Networks has sparked curiosity and optimism to solve face recognition problems which significantly improve the state of the art in face recognition applications.

There are many efficient 2D CNNs architectures that have been proposed recently for the common face recognition tasks. Krizhevsky et al. (2017) introduced the AlexNet in order to improve the performance of the ImageNet challenge. In the 2012 ImageNet LSVRC2012 competition, AlexNet achieved significant accuracy, with the top-5 accuracy of 84.7% compared to 73.8% for the second-best, and it was one of the first deep learning convolution networks which used convolution layers and receptive fields to investigate spatial correlation in an image frame. In the AlexNet architecture, there are 5 Convolution layers with three different sizes of kennels (11x11, 5x5, 3x3) and 3 Fully Connected levels. In addition, the activation function is Rectified Linear Unit (ReLU). However, AlexNet has an enormous number of parameters, requiring a large memory and GPU resources for training. In 2014, the VGG (Simonyan and Zisserman, 2015) model was created as a result of a requirement to reduce the parameters in the Convolution layers and decrease training time. There are two variants of VGGNet: VGG16 and VGG19, with the different sizes of layers in the models. All of the variable sizes convolution kernels used in Alexnet(11x11, 5x5, 3x3) are duplicated by using multiple 3x3 kernels. Typically, researchers stack more layers in Deep Neural Networks to solve a challenging problem with the purpose of improving accuracy and performance. However, it is found that there is a maximum depth threshold for a Convolution neural network model. In both training and testing steps, the 56-layer neural network does not outperform the 20-layer network. This implies that the performance of the model worsens, as the additional layers are added (He et al., 2016). ResNet is a deep residual learning framework introduced by He et al. (2016). It overcomes the challenge of the vanishing gradient problem even with extremely deep neural networks. ResNet comes in a variety of flavors, such as ResNet-50 and ResNet-101.

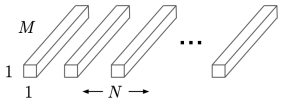
The popular CNNs, such as AlexNet, VGG, and ResNet, have a state of the art performance on face recognition, but these models need a large number of parameters and memory resources. So there is a lighter and faster CNN design – MobileNet (Howard et al., 2017). Depthwise separable convolutions were introduced in MobileNetV1 (Howard et al., 2017) as an efficient substitute for standard convolution layers. Depthwise separable convolutions factor classic convolution by



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 2.1: The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.(Howard et al., 2017)

separating spatial filtering in terms of feature generating mechanism. A 3x3 depthwise convolution for spatial filtering within each channel and a 1x1 pointwise convolution for exchanging information across channels make up depthwise separable convolutions, which is shown in Figure 2.1. The majority of MobileNetV1's computations and parameters are based on point-wise convolutions, which are 31x and 70x faster in principle than depthwise convolutions. After that, MobileNetV2 (Sandler et al., 2018) proposed a more efficient MBConv block and introduced the linear bottleneck and inverted residual structure. A 1x1 expansion convolution is followed by depthwise convolutions and a 1x1 projection layer in MBConv's structure. If and only if the input and output have the same number of channels, they are connected using a residual connection. This structure keeps a compact representation at the input and output while internally expanding to a higher-dimensional feature space, increasing the depthwise convolution's share of calculations and the expressiveness of spatial filtering (Chen et al., 2018).

However, if these CNNs models are applied into mobile devices, the models should have a high accuracy and respond quickly. For these reasons, Chen et al. (2018) proposed a lightweight CNNs model named MobileFaceNet, which implements residual bottlenecks part from MobileNetV2 (Sandler et al., 2018). MobileFaceNet obtains accuracy up to 99.28 % on labeled faces in the wild (LFW) dataset, and a 93.05% accuracy on face recognition in the AgeDB dataset.

2.1.2 Face Recognition for Video

In the case of face recognition in video dataset, there are some main ideas: applying Recurrent Neural Network directly, or using 3D convolution neural network to extract features from video dataset. For using RNN model, Yue-Hei Ng et al. (2015) applied LSTM neural network to capture the temporal information from videos for face recognition. There are some works showing that 3D CNNs can have a good performance for video dataset (Tran et al., 2015). Baccouche et al. (2010) used 3D-convolution neural network to 9-frame videos. In Tran et al. (2015) 's experiment,

they used 3D CNNs to capture both spatial and temporal dimension features at the same time. In 2021, Mishra and Singh (2021) extended 2D residual network into 3D CNNs model. With this experiment, it is clear that according to change the CNNs from 2D to 3D structure, it can be used to face recognition of video. However, in Mishra and Singh (2021) 's experiment, they just implemented the 3D residual network on their tiny dataset (CVBL) and they didn't compare the results with other facial video recognition techniques.

2.1.3 Loss Function in Face Recognition

In face recognition research, the pretrained CNNs such as ResNets, VGG, MobileNets can be used as the powerful feature extractors. At the same time, the loss function is used as the final point of the CNNs to perform the backpropagation and the training. A wise choice in the loss function would result in high-performance models.

In DL face recognition, traditional softmax loss (Parkhi et al., 2015) is widely applied. However, for intraclass samples and diversity for inter-class samples, there are higher similarities that the traditional softmax loss function can not handle. It can cause a different performance for deep face recognition under large intraclass appearance variations.

Based on the softmax loss function, Liu et al. (2017) added margin-based method, which introduced namely Angular-Softmax (A-Softmax)loss. It is used as a way for CNNs to learn angularly discriminative features from input data. In the same year, Deng et al. (2017) proposed the Marginal Loss function, which attempts to maximize inter-class distances while minimizing intra-class variations, both desirable aspects of a loss function. The Margin Loss function focuses on the marginal samples to accomplish this.

Wang et al. (2018) worked on an additive margin for softmax loss and gave a generic function for large margin property to the target logit of softmax loss with feature and weights normalized, motivated by the enhanced performance of SphereFace utilizing Angular-Softmax Loss. Then Deng et al. (2019) proposed ArcFace, which is an additive margin of its own. And with the propose of learning more angle features, they added the margin to the angle instead of the cosine.

2.2 Open source framework - *bob*

Although face recognition has been very popular all over the world, the majority of facial recognition studies are conducted for commercial purposes, making their findings non-reproducible. However, reproducibility is an important feature of scientific research and is required for evaluation. In order to change this situation, Bob (Anjos et al., 2012) was created by the Idiap Research Institute's biometric security and privacy section to meet this demand. It is a free and open-source framework for signal processing and machine learning.[1] The source codes are available on GitLab easily.[2] Bob covers a wide range of biometric research topics and is simple to use due to the Python environment's integration with the C++ library (Günther et al., 2012). Bob, in particular, is an ideal environment for comparing facial recognition algorithms for this study due to the consistency of the results. There are 4 parts for the whole face recognition pipeline: Database, Preprocessor, Extractor, and Algorithm. Researchers can define these parameters directly and combine them into a pipeline easily, which we will discuss in more detail in Chapter 3.

[1] https://www.idiap.ch/software/bob/
[2] https://gitlab.idiap.ch/bob

2.3 Dataset for Face Recognition

With the popularity of the face recognition area, there are a variety of datasets available for various applications. The Labeled Faces in the Wild (LFW) dataset (Huang et al., 2008) is the most widely used dataset. It is a collection of face photos developed to investigate the topic of unconstrained face verification. Face verification, also known as pair matching, is a public benchmark for labeled faces in the wild. Furthermore, the CelebFaces Attributes Collection (CelebA) (Yang et al., 2015) is a large-scale face attributes dataset with over 200,000 celebrity photos and 40 attribute annotations. The images in this set contain a wide range of poses and backgrounds. Microsoft Celeb (MS-Celeb-1M) (Guo et al., 2016) includes 10 million face pictures collected from the Internet.

Above all are image datasets for face recognition. There are two popular video datasets: IJB-C and YouTubeFaces. The IJB-C dataset is a face recognition dataset based on video (Maze et al., 2018). It contains roughly 138,000 face photos, 11,000 face videos, and 10,000 non-face images, and is an extension of the IJB-A dataset. The YouTubeFaces (YTF) dataset (Wolf et al., 2011) is designed for unconstrained face recognition in videos, which we will discuss in detail in Chapter 3.2.

Chapter 3

Approach

In this section, we will describe our approach including which kind of dataset and neural network we have used. Besides, in order to improve the model's performance, we apply some popular optimal methods. We also use bob.bio.face and bob.bio.video packages in open source library *bob* so that we can construct a face recognition pipeline to test our models.

3.1 Model

There are one baseline model and two architectures of neural network used to process video frames, all of them consisting of MobileFaceNet and ArcFace(Figure3.1).

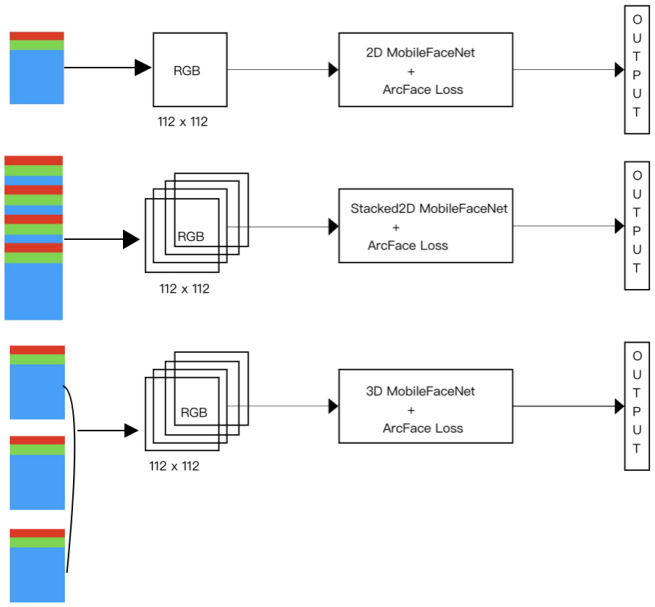


Figure 3.1: Architectures for models.

3.1.1 MobileFaceNet

MobileFaceNet (Chen et al., 2018) architecture is partly inspired by the MobileNetV2 (Sandler et al., 2018) architecture. The residual bottlenecks that came from MobileNetV2 are used as the main building blocks. They used PReLU as the non-linearity activation function instead of ReLU, and after a linear global depthwise convolution layer, Chen et al. (2018) defined a linear 1×1 convolution layer as the feature output layer. At the beginning of the network, we also use a linear 1×1 convolution layer following a linear global depthwise convolution layer as the feature output layer. The input of the MobileFaceNet is a single RGB image in Figure 3.1. And the detailed architecture is mentioned in Table 3.1.

The second model – Stacked2D MobileFaceNet model (Table 3.2) is a variant of 2D MobileFaceNet[3] with the input size of 12 x 112 x 112. The idea of the Stacked2D model is to apply the 2D Convolution neural network into a sequence of input images, like multiple frames in Figure 3.1. Here we can enlarge the kennel's size from 3 to 12. And the number 12 means that there are 4 frames as the input at the same time, and each frame has 3 channels: RGB. Besides, we also used 8 frames as the input for the stacked2D model, which needs to change the red number 12 (Table 3.2)to 24.

However, the 2D CNNs model can be implemented to capture features from the spatial dimensions only. Besides, in the Stacked2D MobileFaceNet model, we need to reshape the 3D input (4 x 3 x 112 x 112) to 2D input (12 x 112 x 112), which destroys the structure of the original 3-channel image and combine the multiple frames.

We apply the 3D CNNs to compute features information from both spatial and temporal dimensions at the same time. In the 3D CNNs model, the 3D kernel is used to preprocess multiple frames together. Based on the original MobileFaceNet model described above, we describe a 3D MobileFaceNet model, shown in Table 3.3. We consider four frames of size 112 x 112 as input to the 3D model and make sure the input format is 3 x 4 x 112 x 112. Here we use 3 x 4 instead of 4 x 3 because the 4 x 3 means that the model extract information based on different color channels, while the model with the input of 3 x 4 size means capture the feature information according to the single RGB image in Figure 3.1.

We design the first layer with a 3-dimensional (3D) kernel size of 3 x 3 x 3 instead of 3 x 3 in the 2D model. At the same time, we fine-tune the size of padding and stride so that we can get a similar architecture with 2D MobileFaceNet.

Input	Operator	Stride	Padding	Output Channel
3 x 112 x 112	Conv_block3 x 3	(2,2)	(1,1)	64
64 x 56 x 56	Conv_block3 x 3	(1,1)	(1,1)	64
64 x 56 x 56	Depth_Wise3 x 3	(2,2)	(1,1)	64
64 x 56 x 56	Residual3 x 3	(1,1)	(1,1)	64
64 x 56 x 56	Depth_Wise3 x 3	(2,2)	(1,1)	128
128 x 28 x 28	Residual3 x 3	(1,1)	(1,1)	128
128 x 28 x 28	Depth_Wise3 x 3	(2,2)	(1,1)	128
128 x 14 x 14	Residual3 x 3	(1,1)	(1,1)	128
128 x 14 x 14	Conv_block1 x 1	(1,1)	(0,0)	512
512 x 7 x 7	Linear_block7 x 7	(1,1)	(0,0)	512
512 x 1 x 1	Linear	-	-	512

Table 3.1: Architecture of 2D MobileFaceNet

[3]https://github.com/TreB1eN/InsightFace_Pytorch/blob/master/model.py

Input	Operator	Stride	Padding	Output Channel
12 x 112 x 112	Conv_block3 x 3	(2,2)	(1,1)	64
64 x 56 x 56	Conv_block3 x 3	(1,1)	(1,1)	64
64 x 56 x 56	Depth_Wise3 x 3	(2,2)	(1,1)	64
64 x 56 x 56	Residual3 x 3	(1,1)	(1,1)	64
64 x 56 x 56	Depth_Wise3 x 3	(2,2)	(1,1)	128
128 x 28 x 28	Residual3 x 3	(1,1)	(1,1)	128
128 x 28 x 28	Depth_Wise3 x 3	(2,2)	(1,1)	128
128 x 14 x 14	Residual3 x 3	(1,1)	(1,1)	128
128 x 14 x 14	Conv_block1 x 1	(1,1)	(0,0)	512
512 x 7 x 7	Linear_block7 x 7	(1,1)	(0,0)	512
512 x 1 x 1	Linear	-	-	512

Table 3.2: Architecture of Stacked2D MobileFaceNet

Input	Operator	Stride	Padding	Output Channel
3 x 4 x 112 x 112	Conv_block3 x 3 x 3	(1,2,2)	(1,1,1)	64
64 x 4 x 56 x 56	Conv_block3 x 3 x 3	(1,1,1)	(1,1,1)	64
64 x 4 x 56 x 56	Depth_Wise3 x 3 x 3	(1,2,2)	(1,1,1)	64
64 x 4 x 56 x 56	Residual3 x 3 x 3	(1,1,1)	(1,1,1)	64
64 x 4 x 56 x 56	Depth_Wise3 x 3 x 3	(1,2,2)	(1,1,1)	128
128 x 4 x 28 x 28	Residual3 x 3 x 3	(1,1,1)	(1,1,1)	128
128 x 4 x 28 x 28	Depth_Wise3 x 3 x 3	(1,2,2)	(1,1,1)	128
128 x 4 x 14 x 14	Residual3 x 3 x 3	(1,1,1)	(1,1,1)	128
128 x 4 x 14 x 14	Conv_block1 x 1 x 1	(1,1,1)	(0,0,0)	512
512 x 4 x 7 x 7	Linear_block4 x 7 x 7	(1,1,1)	(0,0,0)	512
512 x 1 x 1 x 1	Linear	-	-	512

Table 3.3: Architecture of 3D MobileFaceNet

3.1.2 ArcFace

We have shown the idea of the different loss functions in face recognition experiments. In this part, we would like to show the mathematical theories for the ArcFace loss function. As we talked about before, ArcFace (Deng et al., 2019) loss function is modified by softmax loss as the objective function. The traditional Softmax is as follows:

$$L_1 = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{n} e^{W_j^T x_i + b_j}} \tag{3.1}$$

where x denotes the feature vector of the *i th* sample, W and b are the weight and bias. If here we set the bias as 0 for simplicity and transform the function as:

$$W_j^T x_i = \|W_j\| \, \|x_i\| \cos\theta_j \tag{3.2}$$

where θ is the angle between the weight W and feature x (Deng et al., 2019). We can normalize the weight to 1 using the L2 norm. The feature is normalized and re-scaled to s. And the normalization steps make the predictions just depend on the angle θ between the feature and the weight.

The learned embedding is distributed on hypersphere with radius s as(Deng et al., 2019):

$$L_2 = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s\cos\theta_{y_i}}}{e^{s\cos\theta_{y_i}} + \sum_{j=1,j\neq y_i}^{n} e^{s\cos\theta_j}} \tag{3.3}$$

An additive angular margin penalty m is added between weight and feature to enhance the intraclass compactness and interclass discrepancy. Since the proposed additive angular margin penalty is equal to the geodesic distance margin penalty in the normalized hypersphere, it is named ArcFace(Deng et al., 2019). The ArcFace loss function is defined as follows:

$$L_3 = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s\left(\cos\left(\theta_{y_i}+m\right)\right)}}{e^{s\left(\cos\left(\theta_{y_i}+m\right)\right)} + \sum_{j=1,j\neq y_i}^{n} e^{s\cos\theta_j}} \tag{3.4}$$

where s is the the radius of the hyper-sphere, m is the additive angular margin penalty between x_i and W_{y_i}. θ is the angle between the weight W and feature x (Deng et al., 2019). And $\cos(\theta + m)$ is the margin and it makes the class-separations more strictly(Deng et al., 2019).

In this experiment, we use the ArcFace loss as the last layer of models, and the code is from *github–InsightFace*.[4]

3.2 Video dataset, Protocol

3.2.1 YouTubeFaces dataset

We have discussed that there are several databases available for the evaluation of unconstrained face recognition both for images and videos. In this experiment, we use the YouTubeFaces database to train our models.

YouTubeFaces (YTF) database is a large video database designed for unconstrained face recognition in videos (Wolf et al., 2011). It consists of 3,425 videos of 1,595 subjects with significant variations in expression, illumination, pose, resolution, and background. In this database, each subject has an average of 2.15 videos and an average length of a video clip is 181.3 frames.

3.2.2 Protocol for YouTubeFaces dataset

The construction of proper assessment protocols defined on acceptable databases is one of the most important components in measuring the development of a research project. The protocols for YouTubeFaces have been defined by Prof. Dr. Manuel Günther in 2015, so that we can use it directly from *bob.bio.video.database* package. In this experiment, the subjects in YouTubeFaces are divided into 10 folds, we use the 9 folds (2-10) for the training model. Then the models are implemented into the face recognition pipeline and the pairs from the "fold0" are used for testing models.

[4]https://github.com/TreB1eN/InsightFace_Pytorch/blob/master/model.py

Figure 3.2: Example frames from the spectrum of videos available in the YouTube Faces data set.(Wolf et al., 2011)

3.3 Methods for avoiding overfitting

As we said before, the YouTubeFaces database (Wolf et al., 2011) just consists of 1,595 subjects and 621,126 images in general, which means that YTF is a small dataset for training a deep convolution neural network. To avoid overfitting problems, in this experiment, we also implement some methods to avoid overfitting and improve the model's performance.

Overfitting occurs when the model has a high variance. With the development of deep learning technologies, many functional solutions have been applied to deep learning networks, such as dropout, batch normalization, pretraining and data argumentation, and so on (Shorten and Khoshgoftaar, 2019).

3.3.1 Dropout

Srivastava et al. (2014) introduced dropout as the new regularization technology. Dropout is a regularization approach that approximates concurrent training of a large number of neural nets with various designs. During training, certain layer outputs are disregarded or "dropped out" at random (Figure 3.3). This causes the layer to appear and behave as if it were a layer with a different number of nodes and connections than the previous layer. During training, each update to a layer is done using a new "view" of the configured layer, while no nodes are dropped during testing (Shorten and Khoshgoftaar, 2019).

3.3.2 Batch Normalization

Batch normalization (Ioffe and Szegedy, 2015) is another regularization method that can normalize the set of activations in the layer. Each activation is normalized by subtracting the batch mean and dividing by the batch standard deviation (Shorten and Khoshgoftaar, 2019). Along with standardization, it is a standard approach for preprocessing pixel values. The procedure is following by Table 3.4:

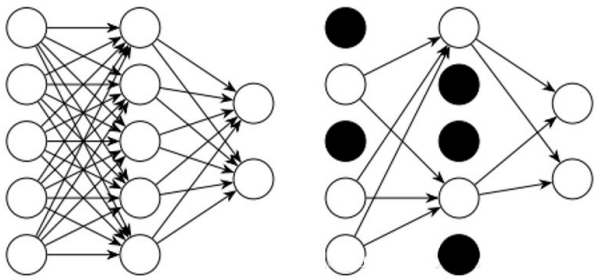


Figure 3.3: Dropout

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots x_m\}$
Output: $\{y_i = \mathbf{BN}_{\gamma,\beta}(x_i)\}$
$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad //mini-batch \quad mean$
$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \quad //mini-batch \quad variance$
$\widehat{x_i} \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad //normalize$
$y_i \leftarrow \gamma\widehat{x_i} + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad //scale \quad and \quad shift$

Table 3.4: Batch Normalization

3.3.3 Pretraining

Transfer learning and pretraining are theoretically quite similar (Shorten and Khoshgoftaar, 2019). The network architecture is designed in pretraining and then trained on a large dataset such as ImageNet (Russakovsky et al., 2015). This differs from Transfer Learning in that the network design, such as VGG-16 (Simonyan and Zisserman, 2015) or ResNet (He et al., 2016), as well as the weights, must be transmitted in Transfer Learning. Pretraining can allow for the establishment of weights with large datasets while maintaining network architectural flexibility, for example, we can delete the specific layers, add more layers in the model, load pretrained weights for specific layers, and so on.

In this experiment, we initialized all the models with the pretrained weight of the MobileFaceNet model which was trained by *insightface*.[5]

3.3.4 Data argumentation

Data argumentation (Shorten and Khoshgoftaar, 2019) is a technique for avoiding overfitting at its origin. It can be used to enhance the amount of a training dataset by transforming existing data while keeping the label. Transform techniques for image collections include horizontal flipping, color space augmentations, and random cropping.

In this experiment, we call the function from *Pytorch* and apply the geometric transformation directly, like horizontal flip, affine, and RandomCrop transformation. *horizontal flip* transformation can horizontally flip the given image randomly with a given probability, and *Random affine* transformation changes the image keeping center invariant. *RandomCrop* transformation can crop

[5]https://github.com/TreB1eN/InsightFace_Pytorch

the image randomly using the fixed size. In Figure 3.4, some instances of Data Augmentations are shown.

Figure 3.4: Image applied in Data Augmentation.

3.4 bob environment

3.4.1 Face recognition pipeline in bob.bio.face

After training in a face recognition task, the model can be applied to the face recognition pipeline in *bob* framework as the feature extractor. The pipeline of face recognition is shown in Figure4.5:

Figure 3.5: Pipeline for face recognition

The vanilla_biometrics pipeline is designed for face recognition in bob.bio.face, which consists of four parts:

Dataset We choose YTF dataset for face recognition pipeline. It has been defined and the interface in Bob contains face recognition protocols for the YTF dataset.

```
database = YoutubeDatabase(protocol="fold0", frame_selector=frame_selector)
```

Preprocessing After we define the dataset, the input frames with annotation will be send to preprocessing part. There are various preprocessors in bob.bio.face package. In this experiment, we

fouces on *FaceCrop* in bob.bio.face.preprocessor.FaceCrop.

```
preprocessor_transformer = FaceCrop(cropped_image_size=(112,112),
    cropped_positions={"topleft": (0, 0), "bottomright": (112, 112)},
    color_channel="rgb")
transform_extra_arguments = (None if (cropped_positions is None
    or fixed_positions is not None) else (("annotations", "annotations"),))
```

Extractor In previous master's project work (Linghu and Zhang, 2021), we have implemented the special extractor class for pretrained model of PyTorch. Here we use a class named *mymodel()*, which has a similar architecture with *Base Transformer* using PyTorch models. In order to implement the model which is trained using multiple frames, we need to define the *_transform(X)* function specifically so that this extractor can process the multiple preprocessed frames.

```
class myModel(TransformerMixin, BaseEstimator):
    def __init__(self, **kwargs):
        # Do some preprocessing before extracting features
    def transform(self, X):
        def _transform(X):
            # Define the input size for different models
    def place_model_on_device(self, device=None):
    def _load_model(self):
        # Load the pretrained model into pipeline
    def __getstate__(self):
    def _more_tags(self):
```

Algorithm After receiving the features of the frames, the algorithm should be implemented to compute the score of the experiment, i.e. the similarities of the registered faces. In this experiment, we use *cosine_distance* as the algorithm. It can be called directly from *bob.bio.base.pipelines.vanilla_biometrics.Distance()* which enrolls the model, i.e. representation of identities, by storing their feature vectors and scores the similarities by computing the distance of the model to the probe by the cosine distance function.

```
algorithm = Distance(distance_function = scipy.spatial.distance.cosine,
        is_distance_function = True)
```

Evaluation The system of bob uses a threshold value to decide whether a similarity score could indicate that the two samples are from the same person. We use the Receiver Operation Characteristic (ROC) plot to evaluate the results. It is a True Match Rate (TMR) by False Match Rate (FMR) curve. So for each FMR, we pursue a higher corresponding TMR.

3.4.2 bob.bio.video

The package bob.bio.video is the part of bob.bio packages. It is a tool to run video face recognition experiments. According to this package, researchers make usage of the *video-wrapper* entry-point

so that they can apply vanilla_biometrics_pipeline from bob.bio.face into video FR. Here is the definition of the *video-wrapper*:

```
# Fetaching the pipeline from the chain-loading
pipeline = locals().get("pipeline")
pipeline.transformer = video_wrap_skpipeline(pipeline.transformer)
```

Chapter 4

Experiments

During the experiment, we preprocess the YouTubeFaces dataset first, so that the dataset can be used as input to train the face classification models, which are 2D, stacked2D, and 3D MobileFaceNet.

After that, these models will be implemented into the face recognition pipeline as a feature extractor in the bob pipeline.

4.1 Data Preprocessing

We use *frame_images_DB.tar.gz* as the dataset, which consists of the file *label.txt* showing the position's value for the bounding box. The function *Face_crop()* from *bob.bio.face* can use these values as annotation and return the required size. Because some data argumentation methods are applied to input frames, here the crop_size of *Face_crop()* is 160 x 160 instead of 112 x 112 (Figure 4.1 - 4.2).

Figure 4.1: Example image of YTF

Figure 4.2: After face-crop using bob

After the cropping part, we split all video data based on the protocol: there are 10 folds of objects totally. We choose the objects in folds 2-10 to train face classification models. And the frames in each video of the object are divided with the ratio of 8:2 for the training set and validation set. In order to reduce the memory requirement and loading data quickly, all the frames are converted into the *.hdf5* file and combined the frames from the same video cut of the object into one *.hdf5* file. Based on the introduction documentation of *Pytorch* [6], the dataset classes *MyTrainingdataset()*

[6]https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

and *MyValiddataset()* are defined separately for this experiment, we also define that the *transform* of data argumentation for training part.

```
class MyTrainingdataset(Dataset):
    def __init__(self, data_dir,transform=None):
        # Iterate all the frames in training YTF dataset, store video label and start index
    def __len__(self):
        # Compute the size of dataset
    def __getitem__(self, index):
        # Load data using its index.
        # Apply data augmentation and preprocess data.
```

4.2 Experiment 1 : 2D MobileFaceNet with ArcFace as baseline

4.2.1 Set up

To compare the performance of the models, there is a baseline model established in this experiment. From *insightface*[5], we can download the pretrained weights of the MobileFaceNet, which is trained using MS-Celeb-1M dataset(Guo et al., 2016).

4.2.2 Results

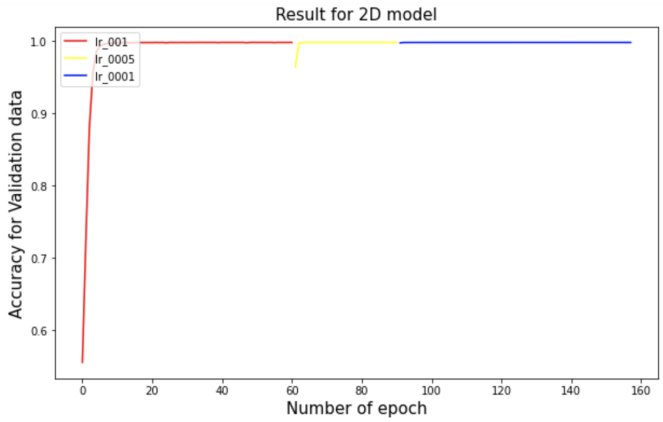


Figure 4.3: Accuracy for validation data during training 2D MobileFaceNet model.

Figure 4.3 shows that the accuracy of the validation dataset changes during training. The baseline model is trained for 158 epochs totally and each epoch takes 8 minutes. It is clear that

[5]https://github.com/TreB1eN/InsightFace_Pytorch

with the increasing number of iterations, the accuracy for the validation dataset improves as well, and the highest accuracy for the baseline model has gone to 99.78% (Table 4.1).

4.3 Experiment 2 : Stacked2D MobileFaceNet with ArcFace

4.3.1 Set up

Based on the original MobileFaceNet model, by modifying the input channel of the model's first layer, we modify the input size from 3 to 12, so that the model can pass 4 images as input at the same time. Also, four images are combined (each image has three *RGB* channels) using function *numpy.array.reshape()* and are converted into the input data. Besides, we still initialize the model with the pretrained weight from *insightface*. In order to fit the shape of weights for the Stacked2D model using the original model, we copy the weights of the first layer 4 times.

If the accuracy of validation within 20 epochs is not improved, the learning rate is reduced at the same time to continue training. In addition, in this experiment, we also train with 8 frames models with the same initial weight.

4.3.2 Results

The Stacked2D model with 4 frames is trained for 334 iterations in general, while the model with 8 frames is trained within 85 iterations. The different number of iterations is because we use the fixed learning rate (0.001) instead of the adaptive learning rate in the 8-frames model.

From Figures 4.4 and 4.5, it is observed that compared with the original model, these two Stacked2D models have a slower convergence speed and longer training time. When the learning rate decreases, there is a sharp drop in the accuracy of the validation dataset. Also, in the training process, both the 4-frames and 8-frames models can achieve 99.78% accuracy in the validation set.

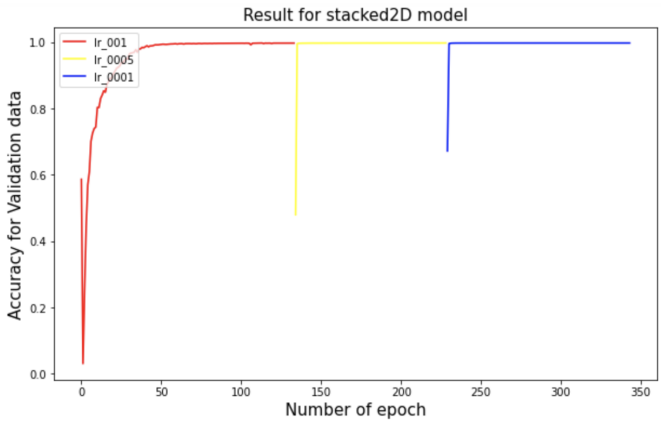


Figure 4.4: Accuracy for validation data during training Stacked2D model (4 frames).

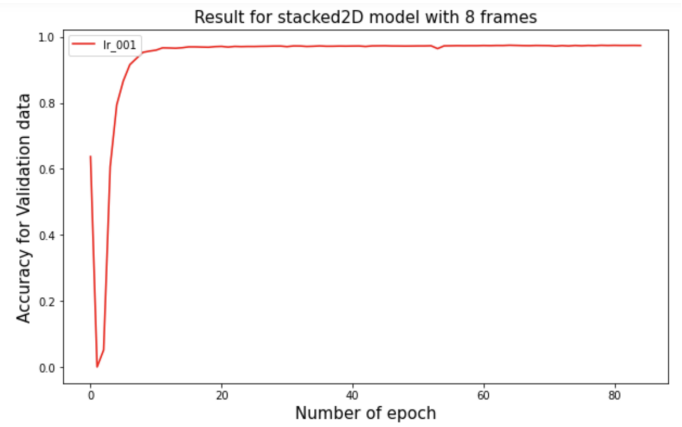


Figure 4.5: Accuracy for validation data during training Stacked2D model (8 frames).

4.4 Experiment 3 : 3D MobileFaceNet with ArcFace

4.4.1 Set up

According to the original MobileFaceNet model, we have the same initial parameters for learning rate (lr=0.01), optimization methods (SGD), and early stopping (20 epochs). We define the 3D MobileFaceNet model as constructed which shows in Chapter 3.4. We use 4 frames as input and make sure the input size is 3 x 4 x 112 x 112. The function *transpose()* and *reshape()* are introduced to adjust the input shape.

For initializing all the weights, we use the function *torch.unsqueeze()* first to extend a 1-element dimension, then we apply the function *torch.cat()* so that we can copy the 2D model's weight many times and load it into 3D model.

4.4.2 Results

Figure 4.6 shows the accuracy of the validation dataset increases during training. There are 125 epochs total for training the 3D model. After 4 iterations, the accuracy achieves 99.16%, then it fluctuates insignificantly within a small range, and in the end, the accuracy can reach 99.78%.

Meanwhile, the 3D model with an input of 8 frames achieves an accuracy 99.76% , which is lower than the 4-frames model.

Model	Highest accuracy	Number of epoch
Baseline - 2D model	99.78%	158
Stacked 2D model (4 frames)	99.78%	344
Stacked 2D model (8 frames)	99.78%	85
3D model (4 frames)	99.80%	126
3D model (8 frames)	97.76%	63

Table 4.1: Accuracy when training for different models

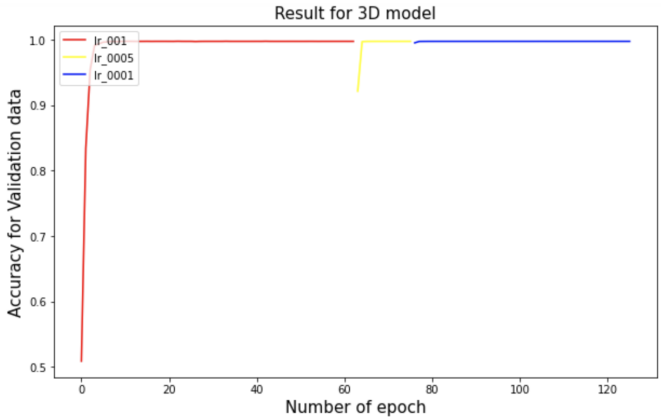


Figure 4.6: Accuracy for validation data during training 3D model (4 frames).

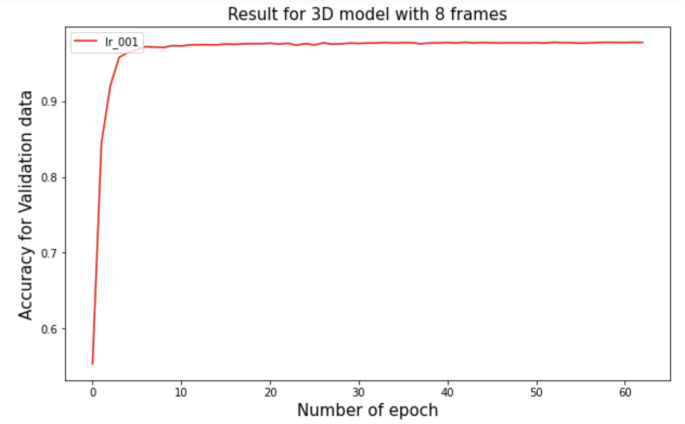


Figure 4.7: Accuracy for validation data during training 3D model (8 frames).

4.5 Experiment 4: Run pipeline in bob.bio.video

4.5.1 Set up

We construct three extractors for the face recognition pipeline in the bob as we described in Chapter 3.4.1. And in my previous work (Linghu and Zhang, 2021), the API of PyTorch extractors has been defined, so we just need to change the method *transform(self, X)* for different dimensions of three models.

```
def transform(self, X):
    # X is the set of preprocessed frames
    # Transform X into format tensor
    # Copy last frames to keep the sames output with labels
    # For loop to read the required size of data:
        x = X[i:i+4] # x:torch.Size([4, 3, 112, 112])
        # Adjust the shape for different models
        x = x[None, ...] # extend the dimension for batch size
        # Pass the data into pretrained model and return features
        # Add features into one set
    # Return the feature set
```

4.5.2 Results

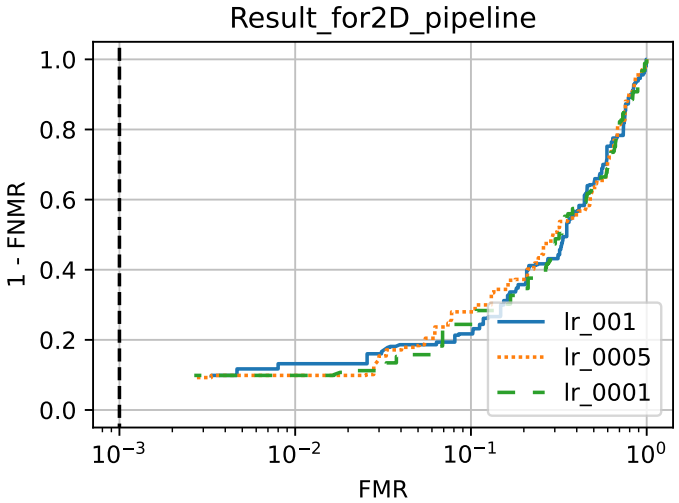


Figure 4.8: Results of 2D model with different learning rates in pipeline.

Figures 4.8– 4.10 show that models trained using different learning rates do not have a significant difference in performance in the bob pipeline.

For all the models trained when the learning rate equals to 0.001, 2D MobileFaceNet has the

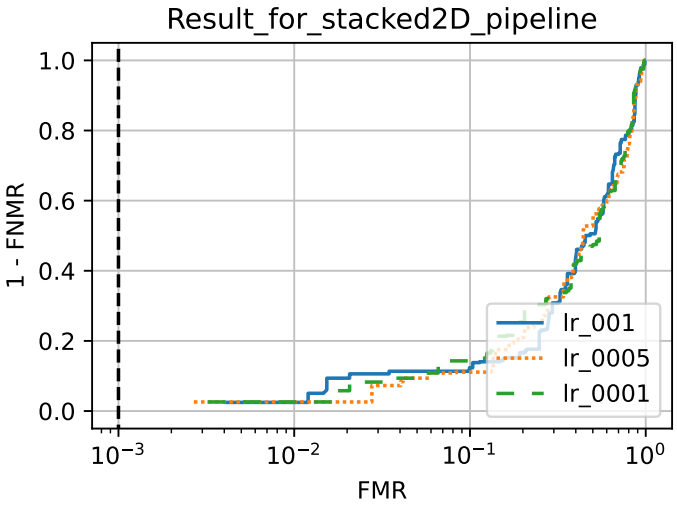


Figure 4.9: Results of stacked2D model with different learning rates in pipeline.

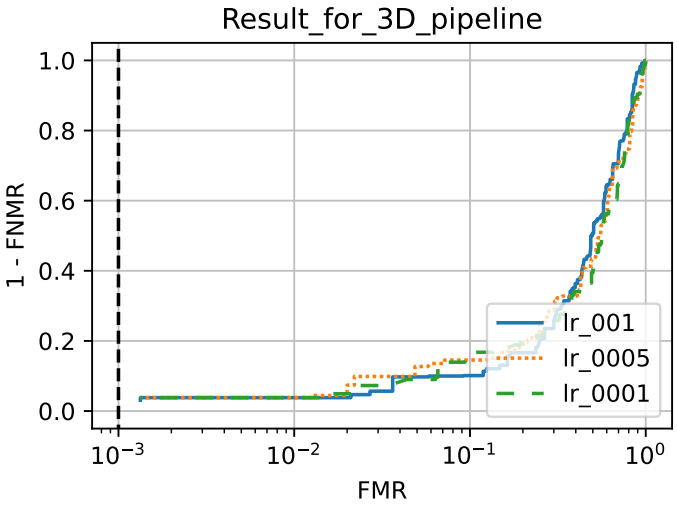


Figure 4.10: Results of 3D model with different learning rates in pipeline.

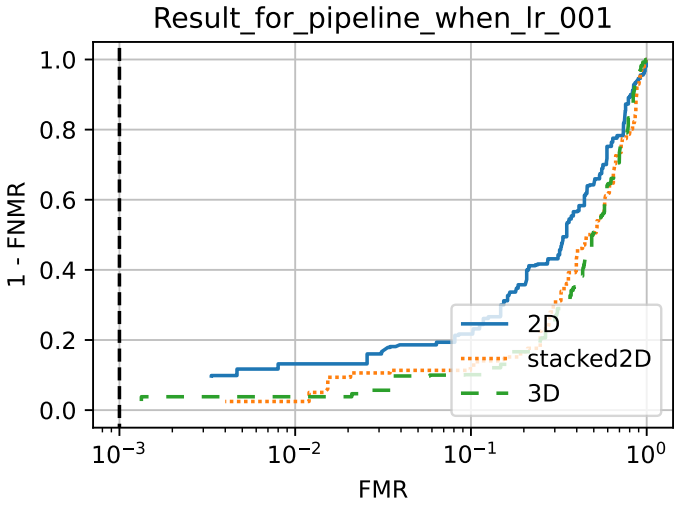


Figure 4.11: Results for different models when learning rate=0.001 in pipeline.

best performance 20% *TMR* at 10^2 *FMR*, while Stacked2D and 3D have 17% *TMR* nearly (Figure 4.11).

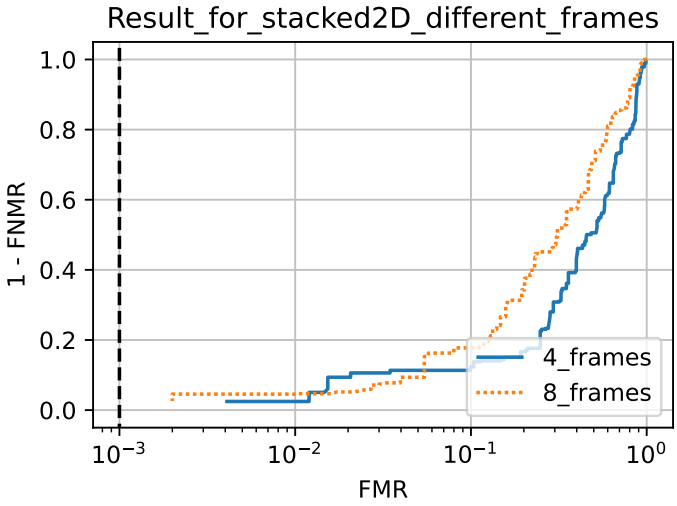


Figure 4.12: Results for stacked2D models with different frame sizes in pipeline.

In addition, by improving the size of input frame from 4 to 8, the Stacked2D model can have a obvious increase from 17.5% to 19.5% *TMR* with $FMR = 10^{-1}$(Figure 4.3).

From Figure 4.13, the 3D model whose input is 8 frames has higher accuracy than the same model with input image is 4 when $FMR = 10^{-1}$.

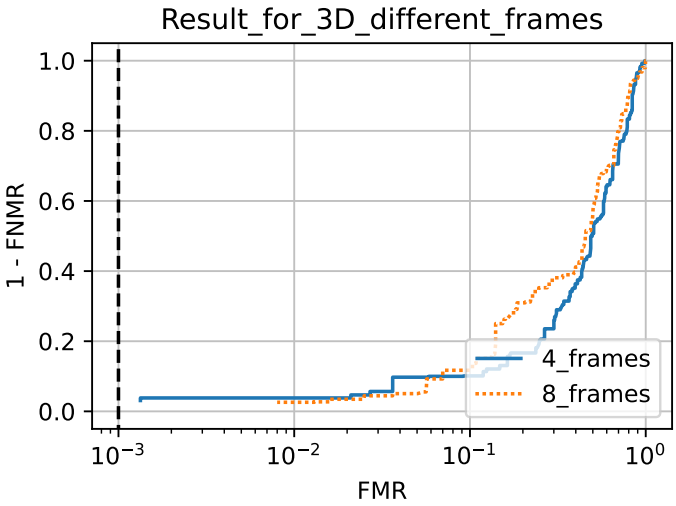


Figure 4.13: Results for 3D models with different frame sizes in pipeline.

Model	Parameter Size	Total Size (MB)	Time per epoch (Minute)
Baseline - 2D model	1,200,512	103.18	8
Stacked 2D model (4 frames)	1,205,696	103.63	16
Stacked 2D model (8 frames)	1,212,608	104.23	28
3D model (4 frames)	1,342,592	399.45	33
3D model (8 frames)	1,442,944	794.13	67

Table 4.2: Information of training for different models

4.5.3 Comparison for models

Table 4.2 shows the required sources and sizes of different MobileFaceNet models. It is observed that as the size of input increases, the requirements of memory and time to train the model are more significant. Besides, the more complex the model, the more parameters of the model are needed for the device. From Table 4.2, the training time for the 3D model using 4 as input size is 8 times that of the original model.

Besides, although in the classification task, the 3D model performs better than Stacked2D and baseline model, it has the lowest accuracy *TMR* = 17%.

From Table 4.3, the baseline model (the original 2D MobileFaceNet) has the highest accuracy *TMR* = 21%. Both for Stacked2D and 3D models, the increasing number of frames has a positive effect on *TMR* (from 17.5% to 19.5% and 17.0% to 17.5%).

Model	TMR
Baseline - 2D model	21%
Stacked 2D model (4 frames)	17.5%
Stacked 2D model (8 frames)	19.5%
3D model (4 frames)	17%
3D model (8 frames)	17.5%

Table 4.3: Result of TMR when $FMR = 10^{-1}$

Chapter 5

Discussion and Future work

We can find that there is a sharp decrease after we implement the pertained weight in the Stacked2D model in Figures 4.4 and 4.5. In that case, the pertained weights may have no effects on the training models, so here we change the iteration part: at the beginning of each epoch, we test the model on validation data firstly, save the model when it has a better performance than before, then train the model.

```
for i in iterations:
    # computer the accuracy of validation data:
        # save the pretrained weights
    # train model using pretrained weights in training data
```

The result can be seen in Figure 5.1: after adjusting the order of training and validation parts, we can get a similar training procedure for the Stacked2D model. And because the time and resources are limited, we don't try the same code in the 8-frame Stacked2D model. We hope that in the future, we can try to use the different methods to initialize weights of models and compare initializing models randomly with initializing models using pretrained weights.

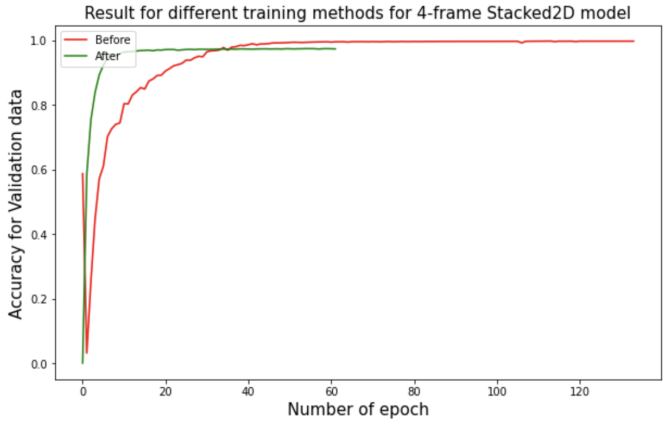


Figure 5.1: Compare the results using diffenert iteration methods for Stack2D model.

From the results of Chapter 4.4.2, although the 8-frame 3D MobileFaceNet model performs

better than the 4-frame 3D MobileFaceNet model, these two models have different sizes of training data: 434,522 in the 4-frame model and 422,098 in the 8-frame model, which causes a tiny difference in accuracy. To a certain extent, the accuracy of the two models is identical.

In the face recognition pipeline, we can get the conclusion that the input frame size is bigger, the model has a higher *TMR*. It is obvious that the models can capture more temporal information from videos if the input consists of more frames. Because of the limitations of resources, we just run the 4-frame models and 8-frames models. Later, we hope we can run the multiple frames size models in the video dataset and verify whether frames size has an important influence on face recognition for video.

The result in the bob pipeline is not good as we expected, we want to confirm whether the architecture of the model and training procedure is correct. Here we use a trick method: training model using all the data in YFT dataset (10 folds) on Stacked2D and 3D models. Same as the previous training process, we train the model first and then apply the pretrained model into the face recognition pipeline in the bob. The results are shown in Figures 5.2 and 5.3. Compared with the models trained using 9-folds data, the *TMR* in both Stacked2D model and 3D model can achieve 100%, it is obvious that all the procedure in our experiment is correct and there are other reasons causes the poor performance in bob pipeline.

We guess the reason is that the YTF dataset we use to train the model is too small. In the future, we plan to use other large biometric videos datasets like IJB-C for video recognition experiments. Besides, the MobileFaceNet is still too complex for the YTF dataset. The models are overfitting during training. We also plan to extend other CNNs models like VGG and ResNet into video datasets. Meanwhile, there are lots of complex 3D neural networks applied in Human Action Recognition tasks (Stergiou and Poppe, 2019), like the FAST 3D model and Pseudo Convolutions model. We would like to implement these spatial-temporal 3D models into the video recognition area later. In addition, the method we split the training and validation data is not the optimal method, because we use a fixed index. Randomly splitting data for training and validation would be a better choice. In the future, we will try to use more various Stacked2D and 3D models for face recognition in video.

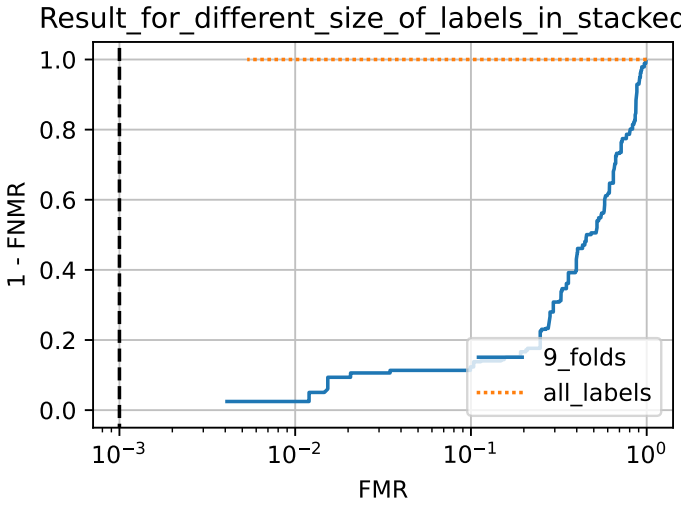


Figure 5.2: Results for different sizes of training data in Stacked2D model.

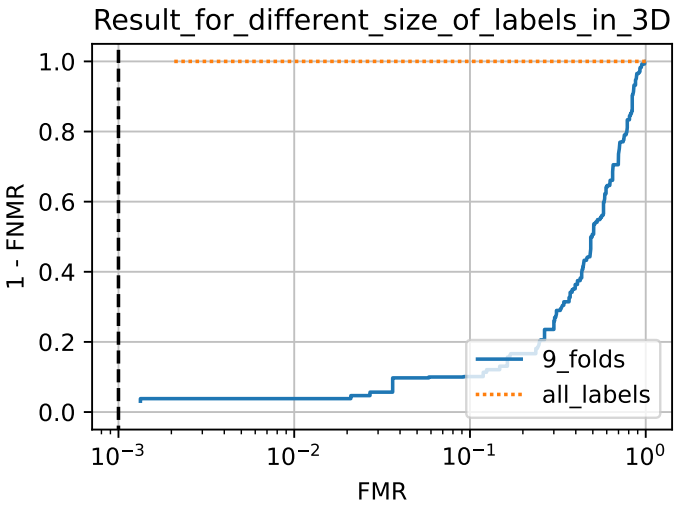


Figure 5.3: Results for different sizes of training data in 3D model.

Chapter 6

Conclusion

In this experiment, we implement three kinds of MobileFaceNet: the original, Stacked2D, and 3D MobileFaceNet. And we add the models with ArcFace loss function. We train these models in video face recognition using the YouTubeFaces dataset. Then we apply these models as feature extractors into the face recognition pipeline in the *bob* framework. Although the results in the face recognition pipeline are not good as we expected, we have proved that it is feasible to run Stacked2D and 3D MobileFaceNet models for face recognition in video, and both of them can preprocess a sequence of images data as input and extract the feature information from multiply frames.

Appendix A

Attachements

List of Figures

List of Tables

Bibliography

Anjos, A., El-Shafey, L., Wallace, R., Günther, M., McCool, C., and Marcel, S. (2012). Bob: a free signal processing and machine learning toolbox for researchers. In *Proceedings of the 20th ACM international conference on Multimedia*, pages 1449–1452.

Baccouche, M., Mamalet, F., Wolf, C., Garcia, C., and Baskurt, A. (2010). Action classification in soccer videos with long short-term memory recurrent neural networks. In *ICANN'10 Proceedings of the 20th international conference on Artificial neural networks: Part II*, pages 154–159.

Cai, Z., Fan, Q., Feris, R. S., and Vasconcelos, N. (2016). A Unified Multi-scale Deep Convolutional Neural Network for Fast Object Detection. In Leibe, B., Matas, J., Sebe, N., and Welling, M., editors, *Computer Vision – ECCV 2016*, pages 354–370, Cham. Springer International Publishing.

Chen, S., Liu, Y., Gao, X., and Han, Z. (2018). Mobilefacenets: Efficient CNNs for accurate real-time face verification on mobile devices. In *Chinese Conference on Biometric Recognition*, pages 428–438.

Deng, J., Guo, J., Xue, N., and Zafeiriou, S. (2019). Arcface: Additive Angular Margin Loss for deep face recognition. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4690–4699.

Deng, J., Zhou, Y., and Zafeiriou, S. (2017). Marginal Loss for Deep Face Recognition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2006–2014.

Guo, Y., Zhang, L., Hu, Y., He, X., and Gao, J. (2016). Ms-celeb-1m: A dataset and benchmark for large-scale face recognition. In *European Conference on Computer Vision*, pages 87–102.

Günther, M., Wallace, R., and Marcel, S. (2012). An open source framework for standardized comparisons of face recognition algorithms. In *ECCV'12 Proceedings of the 12th international conference on Computer Vision - Volume Part III*, pages 547–556.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*.

Huang, G. B., Mattar, M., Berg, T., and Learned-Miller, E. (2008). Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. In *Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition*.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of The 32nd International Conference on Machine Learning*, volume 1, pages 448–456.

Ji, S., Xu, W., Yang, M., and Yu, K. (2013). 3D Convolutional Neural Networks for Human Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of The ACM*, 60(6):84–90.

Linghu, Y. and Zhang, X. (2021). Open-source package for generic deep-network-based face detection and recognition in bob. Master's project, University of Zurich.

Liu, W., Wen, Y., Yu, Z., Li, M., Raj, B., and Song, L. (2017). SphereFace: Deep Hypersphere Embedding for Face Recognition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6738–6746.

Liu, W., Zhou, L., and Chen, J. (2021). Face Recognition Based on Lightweight Convolutional Neural Networks. *Information-an International Interdisciplinary Journal*, 12(5):191.

Martinez-Diaz, Y., Mendez-Vazquez, H., Lopez-Avila, L., Chang, L., Sucar, L. E., and Tistarelli, M. (2018). Toward More Realistic Face Recognition Evaluation Protocols for the YouTube Faces Database. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 413–421.

Mashiko, D. (2020). 3D convolutional neural networks-based segmentation to acquire quantitative criteria of the nucleus during mouse embryogenesis. *npj Systems Biology and Applications*, 6.

Maze, B., Adams, J., Duncan, J. A., Kalka, N., Miller, T., Otto, C., Jain, A. K., Niggel, W. T., Anderson, J., Cheney, J., and Grother, P. (2018). IARPA Janus Benchmark - C: Face dataset and protocol. In *2018 International Conference on Biometrics (ICB)*, pages 158–165.

Mishra, N. and Singh, S. (2021). Face Recognition using 3D CNNs. *https://www.springerprofessional.de/face-recognition-using-3d-cnns/19583950*.

Parkhi, O. M., Vedaldi, A., and Zisserman, A. (2015). Deep face recognition. In *British Machine Vision Conference 2015*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520.

Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1):1–48.

Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR 2015 : International Conference on Learning Representations 2015*.

Singh, R., Mittal, A., and Bhatia, R. (2019). 3D convolutional neural network for object recognition: a review. *Multimedia Tools and Applications*, 78.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Stergiou, A. and Poppe, R. (2019). Spatio-Temporal FAST 3D Convolutions for Human Action Recognition. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 183–190.

Tran, D., Bourdev, L., Fergus, R., Torresani, L., and Paluri, M. (2015). Learning spatiotemporal features with 3D convolutional networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4489–4497.

Wang, F., Cheng, J., Liu, W., and Liu, H. (2018). Additive Margin Softmax for face verification. *IEEE Signal Processing Letters*, 25(7):926–930.

Wolf, L., Hassner, T., and Maoz, I. (2011). Face recognition in unconstrained videos with matched background similarity. In *CVPR 2011*, pages 529–534.

Yang, S., Luo, P., Loy, C.-C., and Tang, X. (2015). From Facial Parts Responses to Face Detection: A Deep Learning Approach. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 3676–3684.

Yue-Hei Ng, J., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., and Toderici, G. (2015). Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.