Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Shahria Afrin

Partial-face Recognition using Patch-based Deep Learning

An experiment of face recognition on masked faces

Master's thesis in Information Security - Technology
Supervisor: Guoqiang Li
June 2021

NTNU
Kunnskap for ei betre verd

Shahria Afrin

Partial-face Recognition using Patch-based Deep Learning

An experiment of face recognition on masked faces

Master's thesis in Information Security - Technology
Supervisor: Guoqiang Li
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Abstract

The need for partial-face recognition to recognize partial-faces from CCTV footages or other pictures existed since the rise of technology. When attempting to do a crime, perpetrators cover their face to conceal their identity from the law. They either tie a handkerchief or bandana while committing crimes like robbery, kidnapping etc. The need to recognize these face covered criminals has also created a need for partial-face recognition. But the need has increased in today's reality where wearing a face mask has become a necessity to stay healthy. Having a partial-face recognition system with high efficiency will improve the security systems like passport control, access control and biometric face authentication without the need for people to remove their face masks. This thesis proposes a method using a patch-based approach on deep learning for face recognition of partial-faces covered by face masks. The face areas of masked covered faces are first patched, and the patches are then used to train a pre-trained deep Convolutional Neural Networks (CNN), FaceNet. The models created after training the CNN with these patches are then used for face recognition. Two fusion techniques (Score-level and feature-level fusion) for the patches were used out of which Score-level fusion gave the best results. The methods used in this thesis consists of using a 3D face masking tool to create masked datasets, face detection and alignment, a patch-based transfer learning and fine tuning of a pre-trained deep learning model (FaceNet), feature extraction using the trained models, and score-level and feature-level fusion of the different patches. The final step of using the score-level fusion gave promising results on both test datasets. It gave an accuracy of 86.75% and EER of 13.25 % for FEI dataset, and accuracy of 91.395% and EER of 8.605% for TUFTS dataset. This shows that partially covered faces by face masks can possibly be recognized by using patch-based deep learning with high accuracy. We believe the performance can be improved by using more diverse face image dataset with high number of images, and pose and illumination variations for training the models. The method can also be improved by replacing the triplet loss function with hierarchical triplet loss function or quadruplet loss function, and by increasing the number of patches for the patch-based deep learning. In the future, this thesis can be extended for face recognition of other partial-face images.

Keywords: Convolutional Neural Network, Transfer Learning, Finetuning, Patch-based, Deep Learning, Partial-face Recognition, Support Vector Machine.

Preface

The basis of this thesis came from my interest in biometrics. The idea of this thesis first originated from my curious thought that if there were such better biometric systems why was there no system to identify criminals from images if their faces were only partially covered. Upon discussion of the possibility of such a system with my supervisor Guoqiang Li, and professor Christoph Busch, they suggested that this can perhaps be done using patch-based deep learning. Following this, the research questions for this thesis were formulated.

I would like to thank my supervisor for being available whenever I needed guidance. I would also like to thank him for his valuable input and support in writing this thesis. Thanks, are also due to my aunt, Shaharima Parvin, who has taught me how to write a good research paper, with proper citations and no plagiarism. Finally, I would like to thank my partner and my parents whose support and motivation has brought me this far.

Contents

List of figures

List of tables

Abbreviations

CNN	Convolutional Neural Network
BFM	Base Face Model
EER	Equal Error Rate
FNMR	False non match rate
FMR	False match rate
Leaky ReLU	Leaky Rectified Linear Unit
MTCNN	Multi-Task Cascaded Convolutional Networks
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit
ROC	Receiver Operating Characteristic
SVM	Support Vector Machine

1. Introduction

Face recognition techniques have been around for years. In today's world frontal face recognition is widely used with high efficiency for security reasons and identification purposes such as for Passports, ID cards, etc. But this usually requires a controlled environment with proper lighting, good image quality, posture of the face and fully uncovered face image. Face images with occlusion or partially covered faces have low recognition rates with these face recognition systems [1].

Recently, due to the COVID-19 pandemic situation people are wearing face masks to reduce the transmission of the virus. This is making almost everyone wear a mask. The current face recognition techniques cannot perform face recognition on these masked faces. Moreover, criminals like thieves and robbers cover their faces with masks or clothes when committing a crime. And the current situation has given them a golden opportunity to blend in by wearing COVID-19 masks and some criminals are taking it [2], [3]. Since the common state-of-the-art algorithms for face recognitions cannot be used on partial faces with high accuracy, it is difficult to identify these criminals even if we have a picture of them. Many criminals committing these crimes also have previous records. This means that the law enforcement database has their images which can be used as a reference image to match them to if we have a method for recognizing these partial face images.

Face recognition can still be done with high accuracy when the nose is uncovered but has low accuracy where only the eye and forehead area is uncovered [4]–[6]. Since, fully masked faces have their nose covered we need a method which works for partial-faces with only the top half uncovered.

1.1 Problem Description

Even though face recognition has attained a great progress in the last few years due to the use of deep learning and other accomplished methods such as Principal Component analysis (PCA) method, it still gives a low recognition rate when it comes to partial faces which are covered with masks [4], [7]. As a result, it is easy for criminals to get away with crimes by concealing their identity by a mere mask or scarf. The current pandemic situation has made it mandatory to wear masks to control the infection rates and stop the spread of the virus. In situations like this the community access control and face authentication fail [8] and people need to remove

their masks putting everyone at risk. It is therefore, very important to have a face recognition system which can recognize partial-faces with high accuracy.

A good partial-face recognition method will help identify the criminals from partial-face images helping the law enforcement agencies detect and catch them. This will reduce masked crime rates due to the criminals' fear of getting caught and also help the victim obtain justice. This method could also benefit security checks in places like airports and passport control as this may not require people to take off their masks for facial recognition.

1.2 Research Questions

The two main research questions for this thesis are:

- To what extent can partial-face or face covered with mask be recognized?
- Can patch-based deep learning be used for partial-face recognition?

1.3 Proposed methodology

For this thesis we require more than one dataset as one of them will be used for transfer learning. The other datasets will be used for testing our proposed methodology. Since, people started wearing masks recently, the number of masked datasets is limited. We therefore, need a method to create these masked datasets.

After we have the datasets, we need to detect and extract faces from these datasets so that we can only focus on the face. This will help with better feature extraction. Alignment of the face also increases the face recognition rates [9].

Partial face images with only eye and nose area have low recognition rate. Using patched face images in the training set increases the recognition rate [10]. Patched-based deep learning helps the models extract more detailed features from the image than if the whole image is used. Patching images therefore, improves the accuracy in some cases [11]. So, to increase the accuracy we use a patch-based approach for this thesis.

Deep learning Face recognition has made tremendous progress in the last few years. Convolutional Neural Network is one of the most successful deep learning methods. The CNN method is effective in image classification [12], object detection [13], and voice print recognition [14]. The CNN can learn abstract expression like a human brain by using a deep

architecture [15], [16]. It is also very successful in face recognition, as described in [17], "The powerful aspect of Deep CNN is the fact that, given an ample training set, these large-scale pattern recognition machines can be optimized end-to-end in order to develop features that amplify the identity signal, while being robust to other PIE (pose-illumination-expression) variations" The PIE problem is one of the problems other methods suffer from and is unable to give good face recognition rates when there is large PIE variations like in [18].

CNN can be used in three ways, which is, training a network from scratch, fine-tuning an existing model, or using off the shelf CNN features. Fine tuning an existing model is called transfer learning [19]. Training a network from scratch requires huge amount of data and takes a lot of time. It is therefore, more efficient to use pre-trained CNN models such as VGGFace, VGG16, VGG19, FaceNet etc. [20].

Therefore, we choose to use transfer learning and fine tuning on a CNN based pre-trained model for our feature extraction and face recognition.

Fusion techniques can be used to improve the accuracy of multiple biometric systems together. There are four different levels of fusions: sensor level fusion, feature level fusion, score level fusion and decision level fusion [21]. Score-level fusion is the fusion between the output scores of multiple biometric systems and can be used to give a better accuracy using the fused scores [22]. Feature level fusion is the fusion between the features from different biometric systems before the scores are calculated. This is also used to improve accuracy for biometric identifications [23]. For this thesis, we use score-level and feature-level fusion between the patches.

So, to summarize, the overview of the proposed methodology is as follows:

- Due to lack of masked face datasets for training and evaluation, a mask tool is applied to create masked datasets
- For each masked face image, face detection and face alignment are done to only get the face area
- Each extracted and aligned face is then patched into 3 patches; top, top-left and top-right patch

- Transfer learning and fine tuning on a pre-trained model (FaceNet) is then done using the 3 patches from one of the datasets and three different models are generated
- Using these models, feature vector for each image from the other two datasets are obtained
- Score-level fusion and feature-level fusion is then done using these feature vectors to improve the accuracy and reduce the equal error rate.

A flowchart of the proposed methodology can be seen in Figure 1.

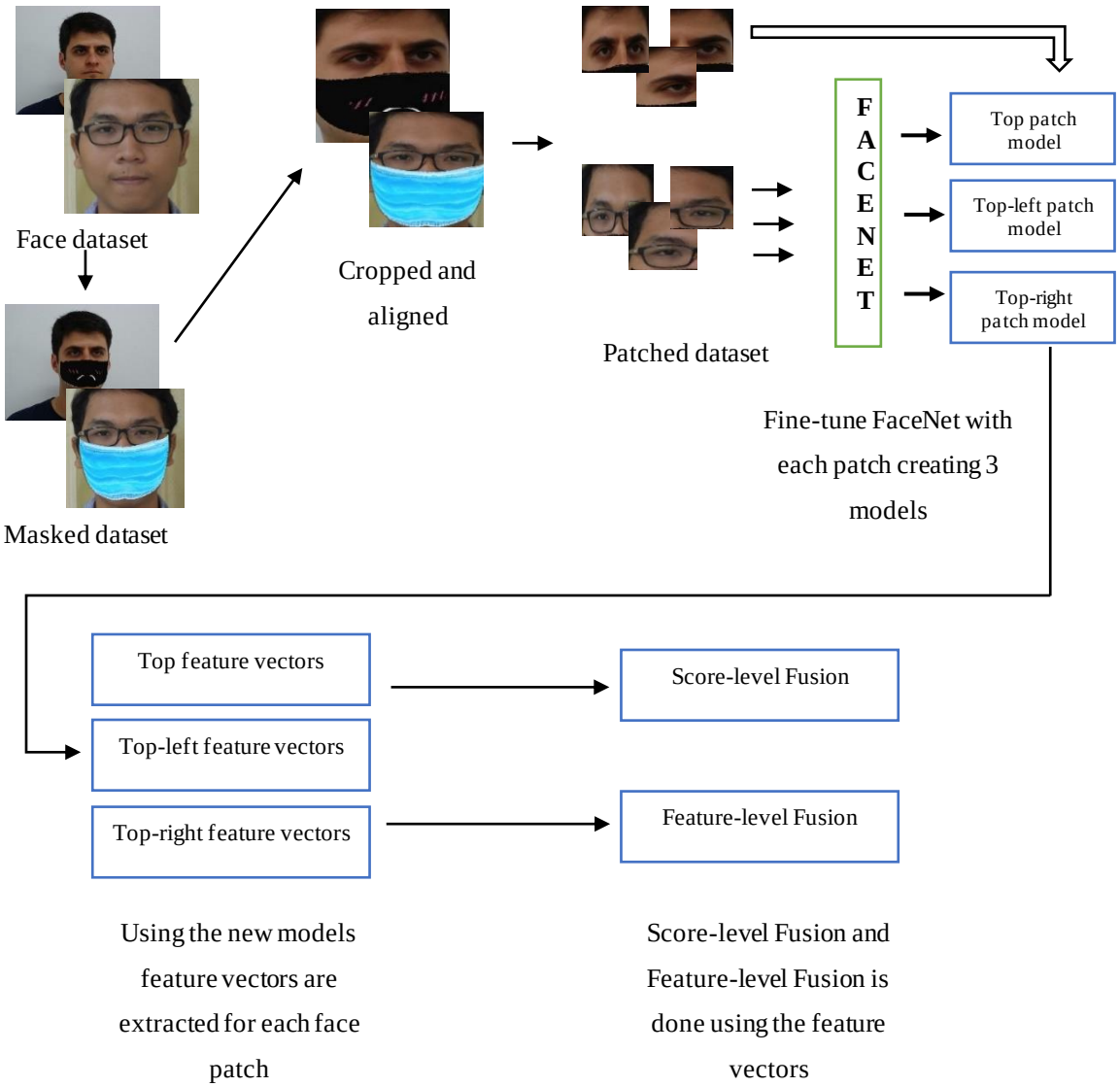


Figure 1: Flowchart of the proposed methodology

2. Motivation and Related Work

This thesis is motivated by the need of a good face recognition technique for partial faces. The current COVID-19 situation has increased the need of such system further for face biometric authentication and security. The motivations for each area of the proposed methodology are described in this section.

2.1 Simulated masked face dataset

Lack of masked face dataset has increased a need for methods to apply face masks digitally.

InsightFace is an open source 2D and 3D face analysis library that can be used in python [24]. They use a lot of face datasets for their projects. To improve and increase the data even more they use data augmentation. One of the data augmentations they use is to add masks to faces. This is done by using a library for 3D manipulation of the face, Face3D [25]. This can be utilized for creating simulated masked face image datasets [26].

The masked face dataset explained in the paper "Masked Face Recognition Dataset and Application" also used a simulated masked face images to increase the volume and diversity of their dataset. They used Dlib library to mask their dataset using the face landmarks around the lower region of the face [27].

Figure 2 shows samples of the two tools to apply face masks on faces.

2.2 Face detection

MTCNN (Multi-task Cascaded Convolutional Networks), RetinaFace and OpenPose are state of the art algorithms to detect faces. Out of these, MTCNN and RetinaFace uses a pre-trained model to detect eyes, nose, and mouth and produce a bounding box, while OpenPose uses ears, eyes, nose, and neck without using a pre-trained model [28].

MTCNN is the most widely used face detection method. MTCNN method is used for face detection efficiently in multiple projects as it uses a lightweight CNN architecture for real-time performance combining the face detection and alignment into one [29]. RetinaFace gives a more accurate placement of the five facial features and better accuracy in face detection than MTCNN [9].

Examples of using InsightFace tool for masking faces [26]

Examples of using Dlib tool for masking faces [27]

Figure 2 : Examples of Simulated masked face images

MTCNN is done as a three stage CNN. Firstly, a shallow CNN is used to produce candidate windows. The number of windows is then reduced using a more complex CNN and discarding the windows with no faces. This reduces the number by a lot. Finally, a more powerful CNN is used to get the final window with output which gives the five facial landmarks positions. By reducing the number of window each time and increasing the complexity, this performs well and faster compared to other algorithms in real-time [30]. The whole image can then be cropped by increasing the output window by some pixel to incorporate the entire face as done in [31].

RetinaFace is a pixel-wise robust single-level face detector. It gives a bounding box and five face landmarks, which can be used for face detection and alignment like MTCNN. RetinaFace performs better than MTCNN in locating the face landmarks but requires more time than MTCNN[9], [32].

2.3 Patch-based method

The paper, Patch-based face recognition from video proposes patch-based method of face detection from a video. In this method they extract patches of the face from the video frames and stitch them for reconstructing the face and then perform face recognition on it. The method used in this paper, however, has a chance of having severe noise in the reconstructed image due

to self-occlusion and region rectification errors as the patches used in the reconstruction can come from different views. The method can also fail in case of large changes in pose, illumination and difference in expressions of the face [18].

In the patch-based face Recognition proposed by [33], face images are divided into patches and these patches are converted into column vectors. These are then converted into "image matrix" using which, the correlation between patches is calculated [33]. [33] uses this image matrixes in a two-dimensional principal component analysis (PCA) framework calculating the correlation of the patches. The projection matrix for feature extraction is then obtained. This method has a better accuracy than one-dimensional PCA, two-dimensional PCA, and two-directional two-dimensional PCA.

In [34], Gabor features were used along with patch-based face recognition and this gave promising results dealing with the unreliability environment in small sample Face recognition and an improves computational efficiency and computational speed. This overcomes the PCA method which, even though has the most outstanding performance, is limited by the sample size and requires a lot of time.

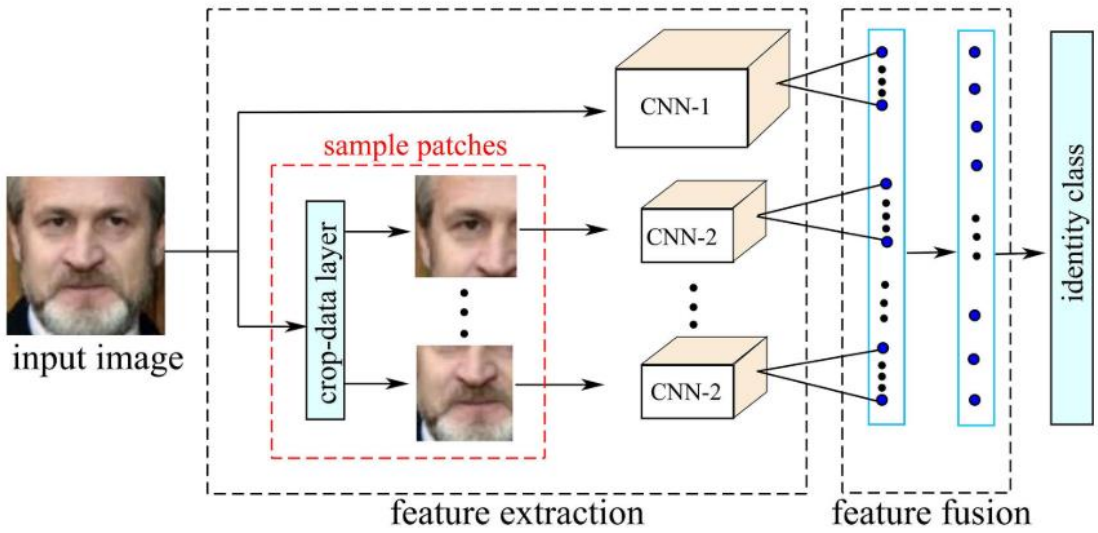


Figure 3: An example of a patch-based deep learning face recognition [6]

[7] uses Local binary patterns (LBP) as a local descriptor followed by Kernel-PCA method after the image is divided into non-overlapping patches. Multiple sub-Support vector machine classifiers are created randomly by a patch sampling technique. This gives good results for faces

in which the lower part is missing, like a face covered with mask. [35] also uses a similar approach of using SVM but with alignment-based face recognition.

Patch-based method has proved to give good face recognition rate and has been used with many methods. Although it fails to give good face recognition rate when there is a large change in pose, illumination and expression using large poses and illumination in the training set will improve the accuracy. Patch-based system will also be affective in partial face recognitions.

2.4 Deep Learning - Transfer learning and Fine Tuning

The paper "Robust Face Recognition Using the Deep C2D-CNN Model Based on Decision-Level Fusion" proposes a color 2-dimentional principal component analysis named C2D-CNN. This method combines the features extracted by CNN from the original pixels and then make decision-level fusion. The CNN model in this paper consists of convolution layer, normalization layer, layered activation function layer, probabilistic max-pooling layer, and fully connected layer for feature extraction. This model provides a robust face recognition and performed better than DeepFace, WSTFusion, COST-S1 and COTS-S1+S2 [15]. The training time of the CNN layers noticed in [15], however, shows that it takes hours to train per layer.

As mentioned in the paper, Deep face recognition using imperfect facial data which uses convolutional neural network (CNN) along with pre-trained VGG-Face model for recognizing imperfect faces from CCTV cameras, partial faces containing only eyes, nose and cheeks individually have low recognition rates. This can be improved slightly using individual parts of the face as references. The face recognition is then done using Support Vector Machine and Cosine similarity [20]. Convolutional neural network 'deep' structures were successfully used by many to generate features needed for face recognition [36].

Good CNN models require large training data and huge amount of computational power to train the models. Training these models with large data also requires a lot of time as seen from above. Therefore, using pre-trained face models can help save time giving similar accuracy. The VGG-Face model used in [10], [20] was trained on an enormous dataset containing 2.6M face images of more than 2.6K individuals. This would take lots of resources and massive amount of time to train from scratch without proper computational power. Therefore, it was used as a pre-trained model [20].

A way to use the old weights from a pre-trained model for creating a new trained model with the same domain is called transfer learning. During transfer learning we separate the feature extraction layers from the classification layer and add a new classification layer. Training this by freezing the feature extraction layers then tailors the model to our needs. Another transfer learning technique is fine tuning. This is done after the previous transfer learning technique. In this method we unfreeze all or parts of the feature extraction layers and train our new dataset on it using very low learning rate to avoid destroying the pre-trained knowledge and overfitting while improving the accuracy [37]–[39].

2.5 Partial face recognition

A dynamic feature matching approach which is a novel partial face recognition approach is proposed in the paper Dynamic Feature Matching for Partial Face Recognition. This is a combination of fully convolutional networks and sparse representation classification to address partial face recognition problem regardless of various face sizes. A frame or still image can be extracted from the video and different face recognition methods can be used on it to improve the recognition results. The results in this method were more effective in terms of accuracy and computation efficiency [4].

[18] also uses face detection on videos using patch-based face recognition. In the paper, different partial face images from the video are stitched together to create a whole face image used for face recognition [18].

Another, recent paper proposed a deep learning-based method and quantization-based technique to deal with the recognition of the masked faces. They used a transfer learning approach with a pre-trained CNN and a deep Bag of features layer for face recognition on masked faces and achieve high accuracy [8].

[20] analyses face recognitions on different parts of the faces and partial face patches individually. The paper states that face recognition on images with only eye area has low recognition accuracy and can be improved by added the partial face patches while training the pre-trained model VGGFace [20].

2.6 Support vector machine and Squared Euclidean distance

Face recognition can be done in multiple ways. The most common ways are to use a classifier and a distance metric. The most effective method for this is using Support Vector Machine as

classifier and Cosine Similarity as distance metric. This two can separate different data more accurately [20]. Squared Euclidean distance is a widely used distance metric which can be used to find the distance between two points or in case of face recognition between two feature vectors. Euclidean distance has been used for face recognition by [40] along with many other image classification methods.

Support Vector Machine – SVM is a supervised machine learning algorithm which separates data using hyperplanes into different classes. SVM can be used for both binary classification and multi classification problems. In this project multi classification is required. This can be done based on a One-vs-One (OVO) approach [41]. SVM is of two types linear and non-linear kernel. In case of face recognition using linear SVM works better [20]. Figure 4 shows an illustration of a multi classification using OVO approach. Each class separates itself from other classes one by one. This finally creates multiple classes. This means, in the case below, red separates itself from blue once then later the green and so on [42].

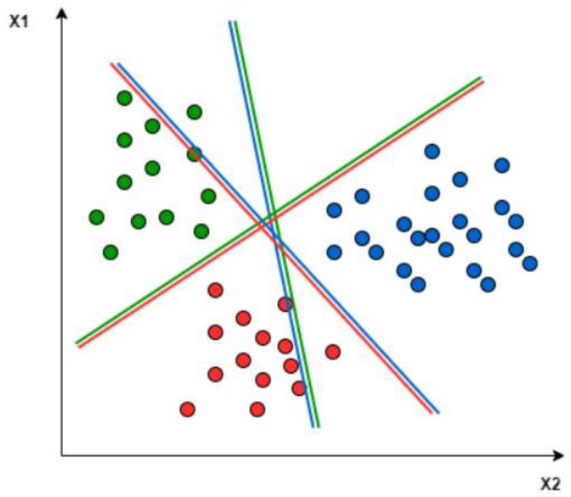


Figure 4: Illustration of multi classification using OVO approach [42]

Squared Euclidean Distance – As the name suggests Squared Euclidean distance is the square of the Euclidean distance. Since, Euclidean distance has a square root, the square in the Squared Euclidean distance cancels it. Square Euclidean distance is therefore, the sum of the square of the distance between two vectors.

$$Distance = \sum (x_1 - x_2)^2$$

Squared Euclidean distance is used when the distances are very small and using square root reduces it further [43]. Squared Euclidean distance has also been used in classification, clustering, image processing, and other areas to save the computational expenses as it avoids the computation of square root [44]. In case of feature vectors, the distances are very small and using square root give an even smaller distance.

3. Methodology

3.1 Datasets used

3.1.1 KomNET: Face Image Dataset from Various Media for Face Recognition

This dataset is used for transfer learning and fine tuning the pre-trained model Facenet. KomNet face image dataset is created for performing face recognition by using three different media sources; mobile phone camera, digital camera, and social media. For this thesis, the images from digital camera were used. The dataset consists of 50 individuals each of them having a total of 24 images with different poses with frontal face images. The images were obtained without conditions like as lighting, background, haircut, mustache and beard, head cover, glasses, and differences of expression. The images were collected in Computer Laboratory, Department of Electrical Engineering, Politeknik Negeri Bali, Bali, Indonesia. Samples of the images from this dataset can be seen below[45].

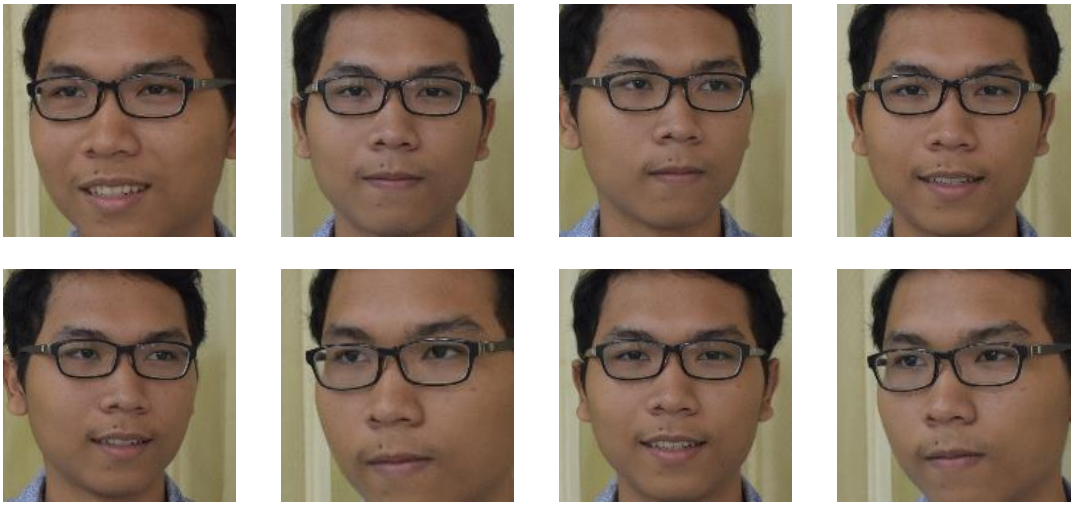


Figure 5: Examples of images of a single individual from the KomNET dataset [45]

3.1.2 FEI Face Database

This dataset is used for the face recognition in this thesis and testing the fine-tuned model. The FEI dataset is a Brazilian dataset which was created using faces of students and staff at the Artificial Intelligence Laboratory of FEI. The individuals used were between 19 to 40 years old with different appearance, hairstyle and adorns. It has 14 images for each individual in the

dataset. For this thesis, the 14th image which had almost no lighting was discarded as for some individual no face was detected with MTCNN or RetinaFace. The images are of homogeneous background consisting of frontal face images with profile rotation of up to 180 degrees. It has two front faces images one with no expression and one smiling face image. The images consisting of poses with 180-degree rotation to the left and right were discarded as well. This was because during the pre-processing of the dataset i.e., adding face masks to them using a tool by InsightFace (explained later in the paper), the face masks on these images were not placed correctly and a part of their nose was not covered with a face mask. Examples of images of different poses can be seen in the figure below [46]–[50].

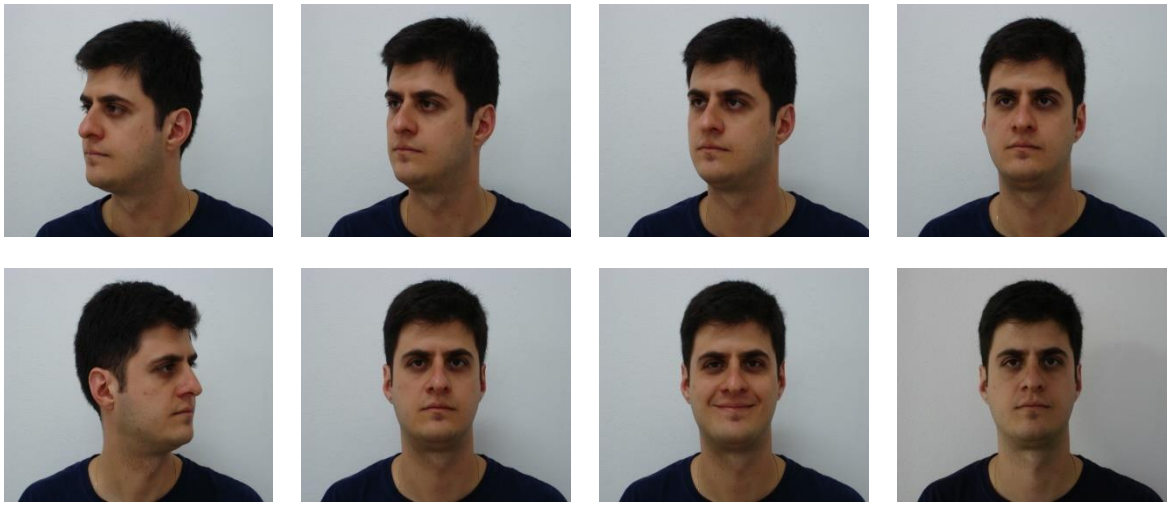


Figure 6: Examples of images of a single individual from the FEI database [46]–[50]

3.1.3 Tufts-Face-Database

This data set is used for face recognition like the FEI face database. Unlike the previously mentioned datasets this dataset consists of diverse individuals. The individuals are from more than over 15 countries and have an age range of 4 to 70 years. The images collected were of students, staff, faculty, and their family members at Tufts University. Each image was taken in front of a neutral deep blue background and constant lighting conditions. The dataset consists of frontal face images with one smiling image, one mouth opened image, one eyes closed image and the rest neutral expression images. The poses of the images have a little rotation to the left or right. For this thesis, two individuals were discarded as they contained only 4 images in total while the rest had 8 images per person. This was done to keep consistent number of images of

all the individuals in train and test sets. Example of images in this dataset can be seen from the figure below [51].

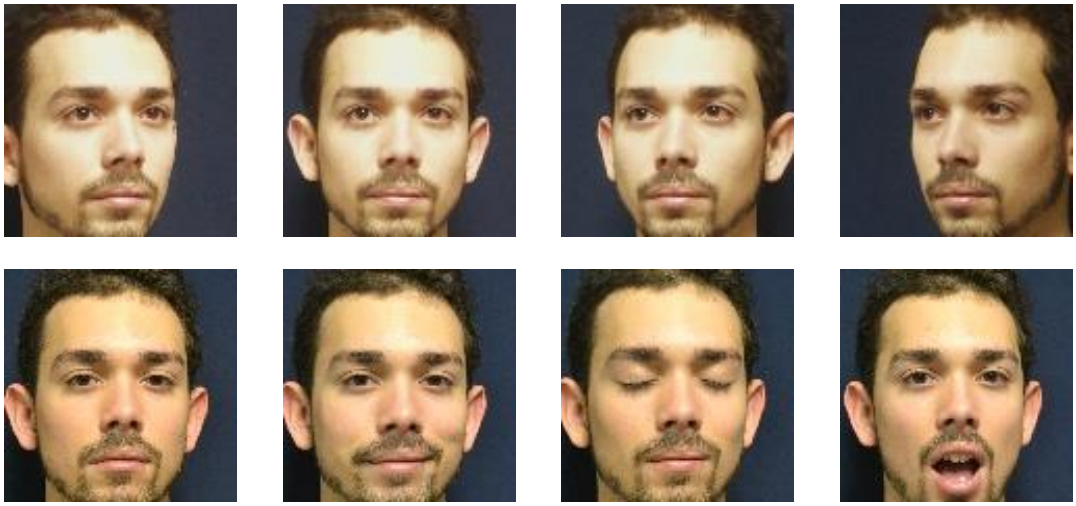


Figure 7: Examples of images of a single individual from the Tufts-Face-Database [51]

3.2 Data Pre-processing

3.2.1 Masking

Due to covid, everyone started wearing masks since 2020. There are not many datasets for face recognition which contains all masked faces as this happened only recently. Most datasets with masked faces have 3 to 5 images per person. When this is divided into train and validation set the amount of data is not enough for transfer learning and does not give a good model. The model in this case has low accuracy in predicting unknown or unseen images and hence is low in accuracy for feature extraction and face recognition. To get a model which has better accuracy in feature extraction and face recognition for unknown or unseen image we need a dataset which has many images per person [37], [52].

This thesis requires more than one face image dataset with at least one of them having many images per person as one dataset is used for transfer learning and fine tuning a pre-trained model, as explained above, the more the amount of image per person the better the model. For this reason, a masked face dataset was created using normal face recognition datasets with frontal face images and a masking tool which is used to apply masks on these images.

InsightFace is a deep learning toolkit for face analysis based on MXNet deep learning framework [24]. It has a tool created for adding masks to faces automatically which is used for data augmentation while training their face recognition models [26]. This tool is created by using face3d which is a python toolkit for processing 3D face. Face3d uses 3D68 model and Base face model (BFM) which is a pre-trained model. The BFM is created using raw BFM morphable model [53] and information from face profiling, 3DDFA (Face Alignment Across Large Poses: A 3D Solution) [29], HFPE (High-Fidelity Pose and Expression Normalization for Face Recognition in the Wild) [54] and UV coordinates from 3D Morphable Models as Spatial Transformer Networks [25], [55]. Some examples of face images before and after face masks are applied using this tool can be seen in Figure 8.

Figure 8 : Face images before and after using the face mask renderer tool [26]

3.2.2 Face detection

Before we can train our model, we need to extract faces from the images for better results of face recognition. The most widely used method for this is the Multi-task Cascaded Convolutional Networks (MTCNN) [29], [56]. MTCNN is a cascaded system which gives

quality results and is much faster than other methods [29], [56]. However, MTCNN was not able to extract all the faces from our dataset due to the occlusion (face mask). It could not detect faces from FEI dataset when the lighting used was very dim for the occluded faces. Another issue with using MTCNN was that the five facial landmarks MTCNN detected was a little less accurate which caused the face alignment performed in the next section to be less accurate. RetinaFace, which is a robust single-stage face detector, on the other hand was able to extract all the face images and also gave more accurate face landmarks than MTCNN. Based on our pre-analysis of comparing MTCNN and RetinaFace, we decide to apply RetinaFace for face detection on our masked datasets.

RetinaFace is a single-stage pixel-wise face detection framework using a state-of-the-art dense face localization method. This is done by exploiting multi-task loses obtained from extra-supervised and self-supervised signals. RetinaFace simultaneously predicts face score, face box, five facial landmarks, and 3D position and correspondence of each face pixel [9]. In this thesis, we used the face box output to extract faces and the five facial landmarks for alignment. An example of RetinaFace architecture can be seen in Figure 9.

The implementation of RetinaFace can be seen in the figure below. RetinaFace is implemented using a five level feature pyramids, denoted by Pn in Figure 10. The first 4 feature levels are computed from the corresponding ResNet residual stages, denoted by Cn, as seen from the figure. The last pyramid feature level P6 is computed through a 3×3 convolution with stride=2 on C5. Before the multi-task loss is calculated from each anchor independent context modules are applied on the five level feature pyramid. This increases the receptive field and enhances the rigid context modelling power. Then the multi-task loss is calculated for positive anchors which are s (1) a face score, (2) a face box, (3) five facial landmarks, and (4) dense 3D face vertices projected on the image plane [9].

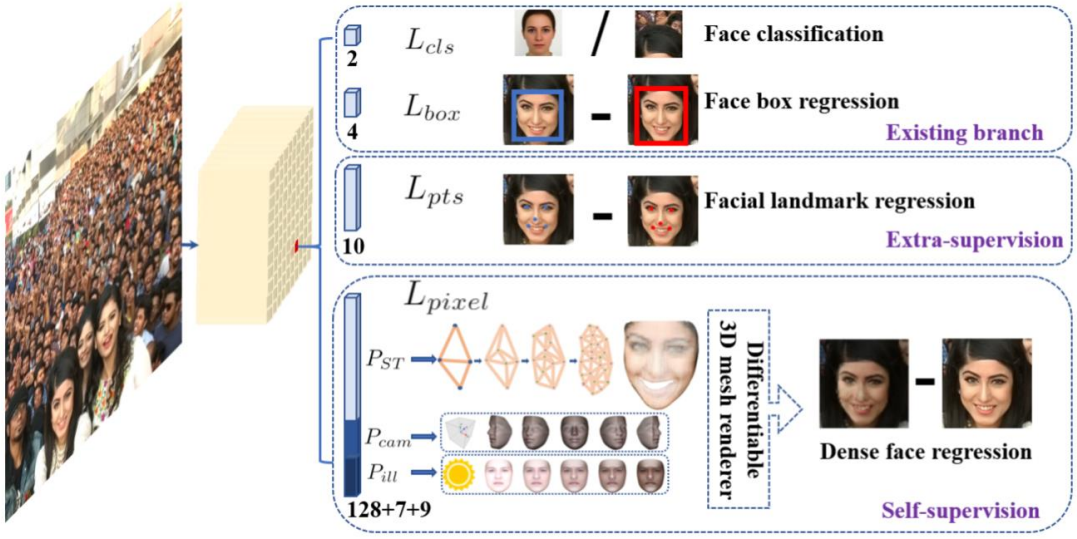


Figure 9 : RetinaFace single-stage pixel-wise face localization [9].

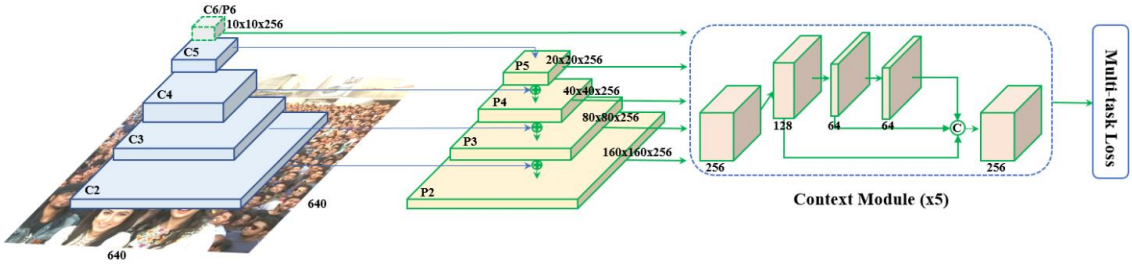


Figure 10 : RetinaFace architecture with five level feature pyramid and independent context module followed by multi-task loss for each positive anchor [9].

Face recognition rate from the paper ArcFace which uses MTCNN from face detection and alignment increased on all databases when RetinaFace was used instead of MTCNN. Compared to MTCNN RetinaFace also decreased the normalized mean error rate and failure rate in locating the five facial landmarks [9].

Since face detection and alignment affects the performance of face recognition and using RetinaFace locates the five facial features with better accuracy [9], RetinaFace was used for this Thesis. It was implemented using InsightFace which is a deep learning toolkit for face analysis based on MXNet deep learning framework [24], [57].

3.2.3 Face alignment

Face alignment is another important step which improves face recognition accuracy [9], [58]. Specially for this thesis it is important to align faces for the next step, face patching. Since, for

face images covered by masks we can only see the top half of the face. The patches we require are of the top half. Alignment makes it easier for patching the faces.

From RetinaFace while we get the face bounding box and the five face landmarks. The eye coordinates from the five face landmarks are selected for face alignment. After the alignment the face is cropped from the image using the coordinated of the face box. The concept in which the face is aligned is explained in the following paragraph.

First the two eye coordinates are located. Then a line is drawn connecting the two eyes. To align the face images, we have to rotate the image to make this line parallel to the horizon [58], [59].

To get the angle needed to rotate the image we draw another dot using the left eye and right eye coordinates and join the dots with two more lines, creating a triangle. Using this triangle, we can find the angle of rotation θ. From Figure 11 we can see that tan θ can be calculated by Δy/Δx. Therefore, to get the angle θ we need to calculate tan-1 (Δy/Δx). After the angle is calculated, the image is rotated in that angle. Then the image is cropped using the face box coordinated from RetinaFace and resized to 160 * 160 as the FaceNet pre-trained model requires this image size. Figure 12 shows an example of all the steps used for face alignment and face extraction.

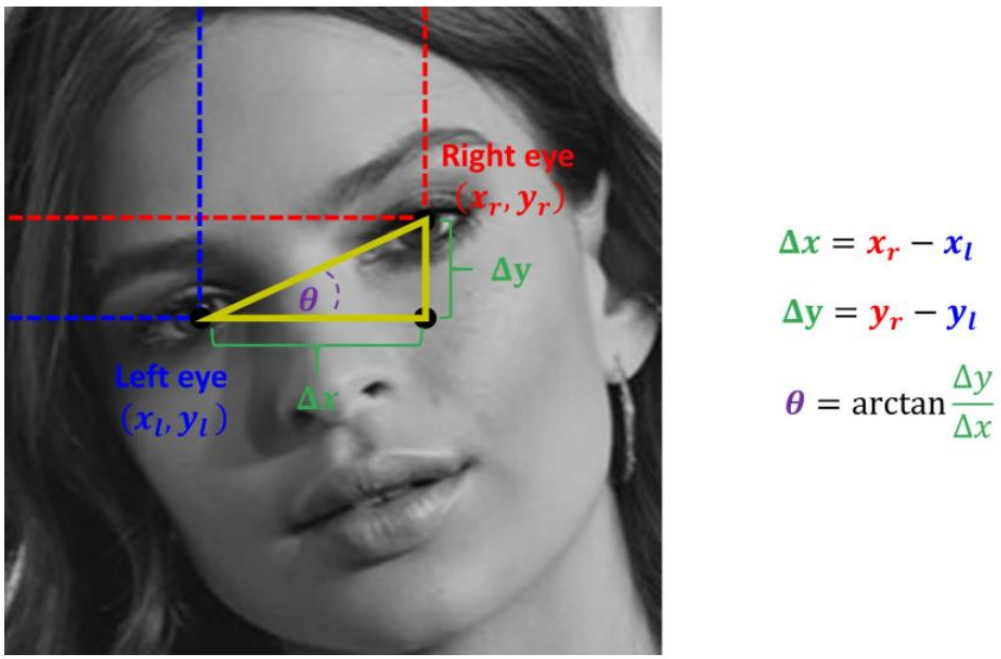


Figure 11: Example of finding the angle from the triangle [59]

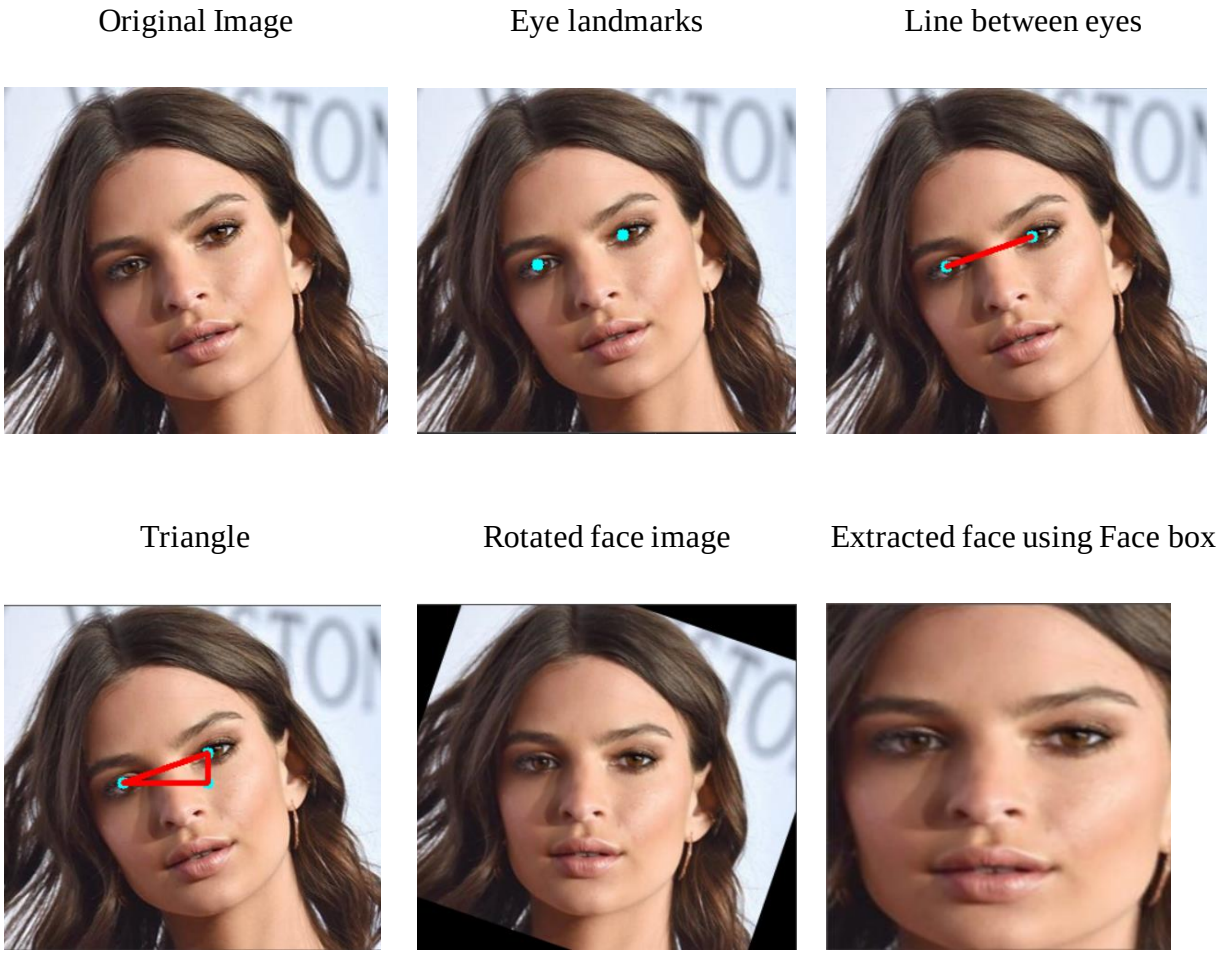


Figure 12 : Example of the face alignment and face extraction process of a face image

For each dataset that is used in this thesis all faces are cropped and aligned before patching. This gives almost similar and consistent patches for all the face image datasets. Since the patches in the dataset used for transfer learning and fine-tuning the models is similar to the patches used for testing the models, the accuracy of face recognition is higher than if there was no alignment done.

3.2.4 Patching

The next step after face alignment is to patch the face images into multiple patches so that from each patch, we can find out the important features of that patch and finally obtain features from the entire image.

Since the face images in the datasets of this thesis are already extracted and aligned from the previous steps, we can easily patch the face images. This is done by first cropping the face in half + 10 pixels and considering the only top half for the first patch. As the face image is of n*n size the top part after cropping contains the face not covered by mask. 10 pixels are added extra during cropping as due to some individuals' poses the area below the eye is cropped out. The top patch is then divided into two more patches, one is the top left patch containing the left eye and another the top right patch containing the right eye. An example of patching a masked face image is shown in Figure 13.

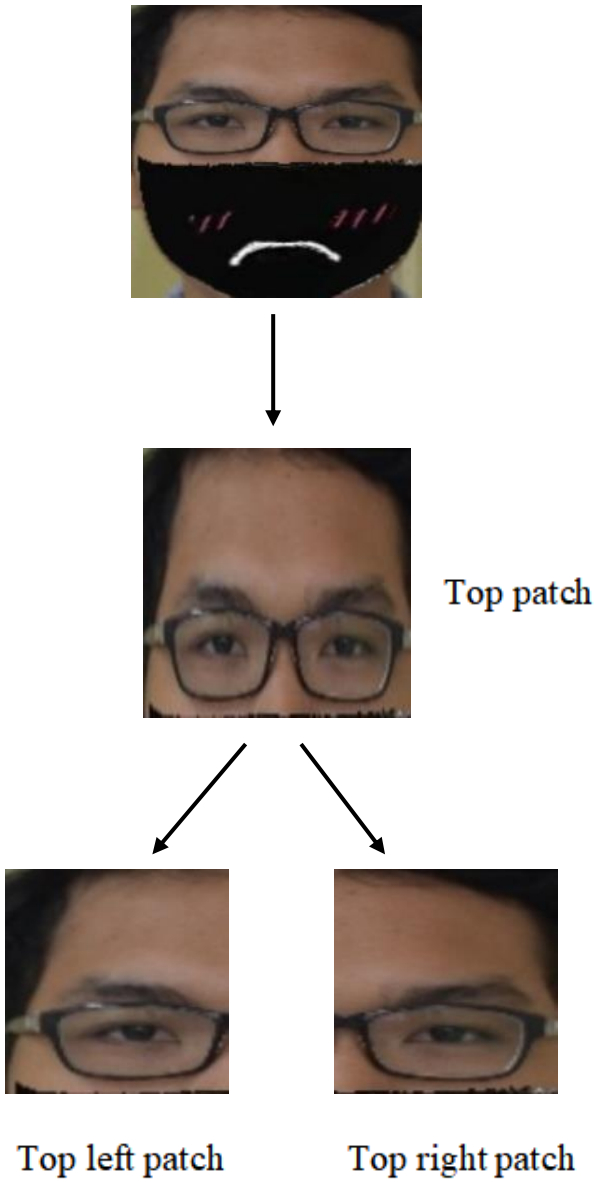


Figure 13 : Example of patching a masked face image

Figure 14 : Example of patching without adding the extra 10 pixels

3.3 Transfer learning and finetuning

Pre-trained models are trained on large amount of data and can be used for a wide variety of image classifications. If we want to perform a specific image classification and train models from scratch for it, it will require large amount of data and computational capability which is not practical for a normal home laptop. Therefore, we can use transfer learning and fine tuning in such cases where we can use the knowledge from a pre-trained data and use our own data to create a model which provides classification according to our needs.

3.3.1 FaceNet: A Unified Embedding for Face Recognition and Clustering

FaceNet is a system used for face recognition by directly learning to map face images to a compact Euclidean space. Distance between these mapping corresponds to a measure of face similarity which can be used for face recognition, verification and clustering. FaceNet takes in color images as inputs and gives embeddings as outputs, which is a vector of size 128. This output can be used as feature vector for face analysis. [60]

FaceNet uses a deep CNN network followed by L2 normalization. This gives, as outputs, embeddings which is then followed by a triplet loss function during training. Triplet loss is a function which minimizes the distance between an anchor and a positive and maximizes the distance between an anchor and negative, A positive here is when both have same identity and negative is when both has different identity. The dimensionality of the output is selected to be 128 as when the dimensionality is increased to get better result it requires more training to achieve the same accuracy and therefore, even more than that to get better accuracy. [60].

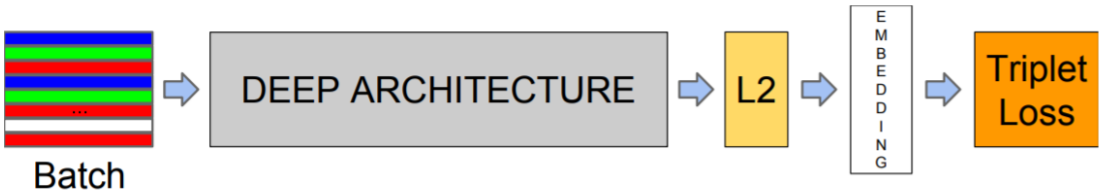


Figure 15: Model Structure for FaceNet [60]

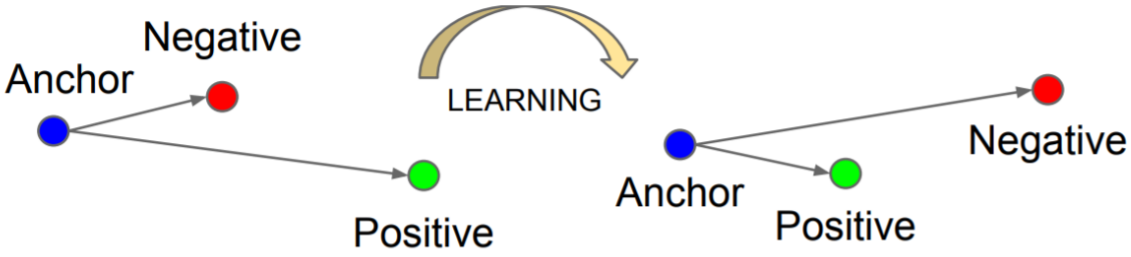


Figure 16: Triplet Loss maximization and minimization for training [60]

Inception based CNNs used for FaceNet reduces the model size giving comparable performance as other CNN models [60]. Inception model have multi-branch architectures which has powerful representation ability in its dense layers [61]. Residual model is good for training very deep architecture. The hybrid of inception and ResNet thus give good efficiency [62], [63]. FaceNet implementation in "Face Recognition using FaceNet (Survey, Performance Test, and Comparison)" is done using Inception ResNet v1 with training dataset as CASIA- WebFace and VGGFace2 Inception both of which gave above 99% accuracy on LFW as test data [64]. Therefore, for this thesis FaceNet pretrained model based on Inception- ResNet hybrid CNN is used. The pre-trained models in "Face Recognition using FaceNet (Survey, Performance Test, and Comparison)" is trained using TensorFlow and can be used with TensorFlow [65] whereas the FaceNet model by Hiroki Tainai is Keras based. Therefore, to use a Keras pre-trained model for this thesis, the pretrained Keras FaceNet model by Hiroki Tainai is used. This pre-trained model also uses an Inception ResNet v1 architecture and is trained on MS-Celeb-1M dataset. It takes a color image input of size 160*160 pixels [66], [67]. Keras pre-trained model is chosen instead of TensorFlow as Keras is easier than TensorFlow to code from scratch [68].

3.3.2 Transfer learning

Transfer learning is used for improving the performance on a related task by using the knowledge from another set of tasks from an interrelated learning problem [69]. The training

data and future data in this interrelated learning problems or algorithms does not need to be of the same feature space or have the same distribution [38].

The most common workflow of transfer learning is as follows [39]:

1. **Taking layers from a pre-trained model**

   For this thesis, we take all the layers from the pre-trained model FaceNet except the last two layers for our transfer learning.

2. **Freeze the layers taken from the pre-trained model to preserve the knowledge it learned from previous large dataset it was trained on**

   In this step, we freeze all the layers taken from the pre-trained model. As the base model (FaceNet) contains Batch normalization layers we have to set the training of the layers as training = False to keep it in inference mode during the next step. If this is not done the non-trainable weights of the Batch-normalization layer will destroy the knowledge of the model when the layers are unfrozen for fine tuning (next step) by updating the weights [39].

3. **Add new unfrozen layers on top to learn from the old features of the pre-trained model and use it to predict new features on new dataset**

   For this thesis for the classification, 3 fully connected layers (Dense layers) are added with Leaky ReLu activation where alpha was set to be 0.03. Leaky ReLu is same as ReLu activation with an ability to go to the negative values depending on the alpha we set. A leaky ReLu most of the time gives better results than ReLu [70]. For this thesis Leaky ReLu gave better accuracy then ReLu activation, so the models were trained using Leaky ReLu. The next layer added on top of the dense layers is a batch normalization layer to accelerate training with less number of epochs. It also provides regularization and reduces generalization error [71]. Since we used Batch normalization, we do not use any Dropout here as using a dropout along with the batch normalization reduced the accuracy of the models. This is because the combination of these causes overfitting [71]. Lastly, another dense layer with SoftMax activation is added on top; as the dataset, the models are trained on has a total of 50 classes.

4. **Final step is to train the newly added layers using new dataset**

For fitting the model and training it Adam optimizer is used. Adam optimizer performs better in practice as it achieves good results with less training cost and iteration over entire dataset as seen in Figure 17 [72]. The learning rate 0.001 of Adam was kept as default learning rate in Keras and other deep learning libraries. This is suggested by the original paper as a good default setting for machine learning problems [72].

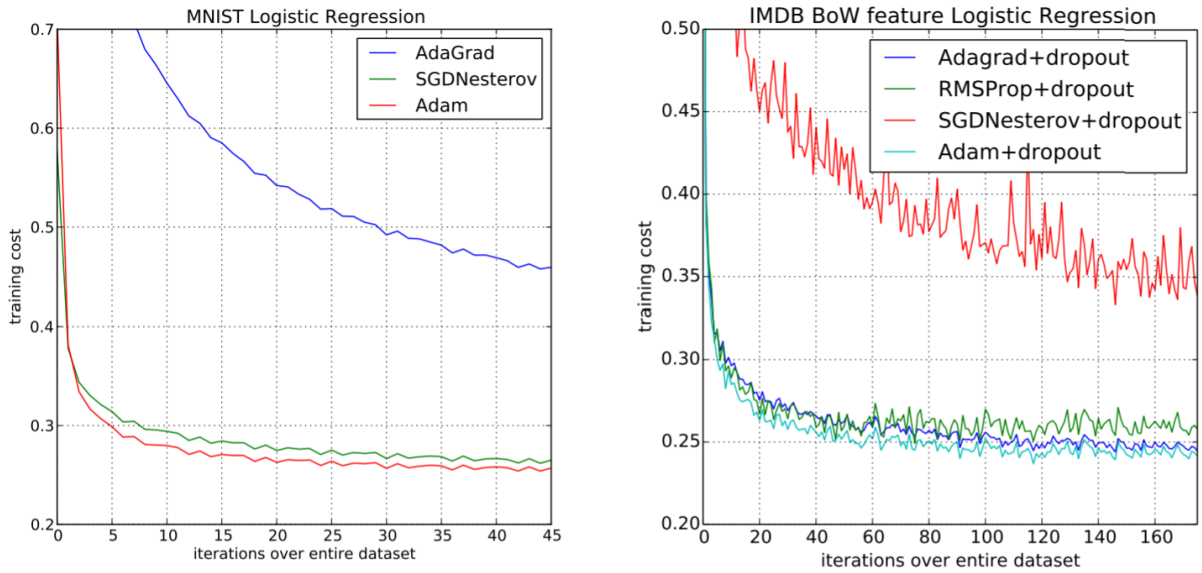
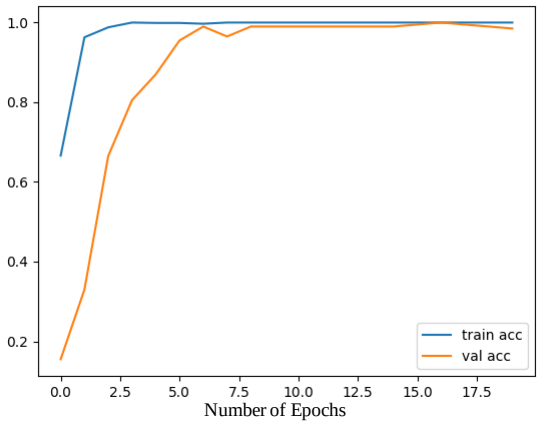


Figure 17: Comparison of Adam to Other Optimization Algorithms [72]

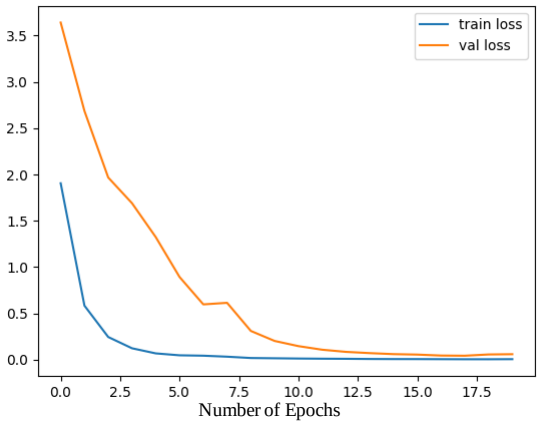
Three models were trained using transfer learning, one for the top patch, two for the top split in two (top-left and top-right patch). The KomNET dataset was used here for training and validation. The dataset was split into a 70:30 ratio for the train and validation sets. Before training, each input is normalized to -1 to +1 range from (0, 255) as inception-resnet pre-trained models require the input to be normalized in this range. This was done using the formula:

$$New\ input = \frac{(old\ input - mean)}{sqrt(variance)}$$

To avoid overfitting, number of epochs was set to 20 with early stopping. The Early stopping was done based on the validation loss. If the validation loss is reduced the model is saved and over written every time, we get a lower validation loss. If the validation loss keeps decreasing for 3 consecutive epochs, we stop the training.

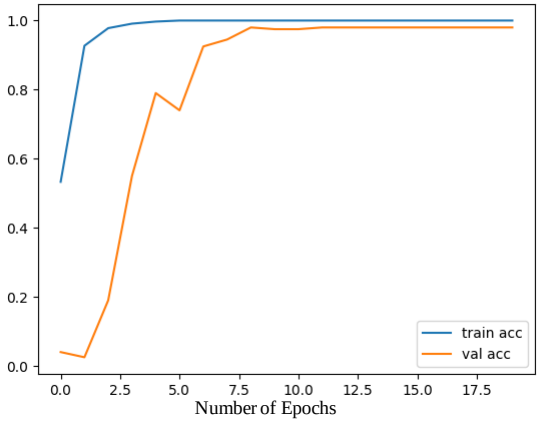


Train accuracy vs. validation accuracy

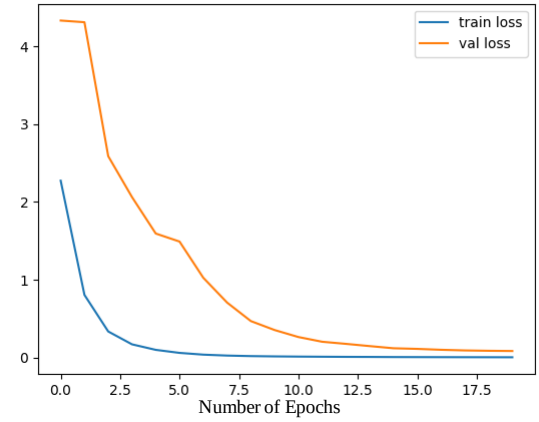


Train Loss vs. validation loss

Figure 18 : Graphs of training the top patch

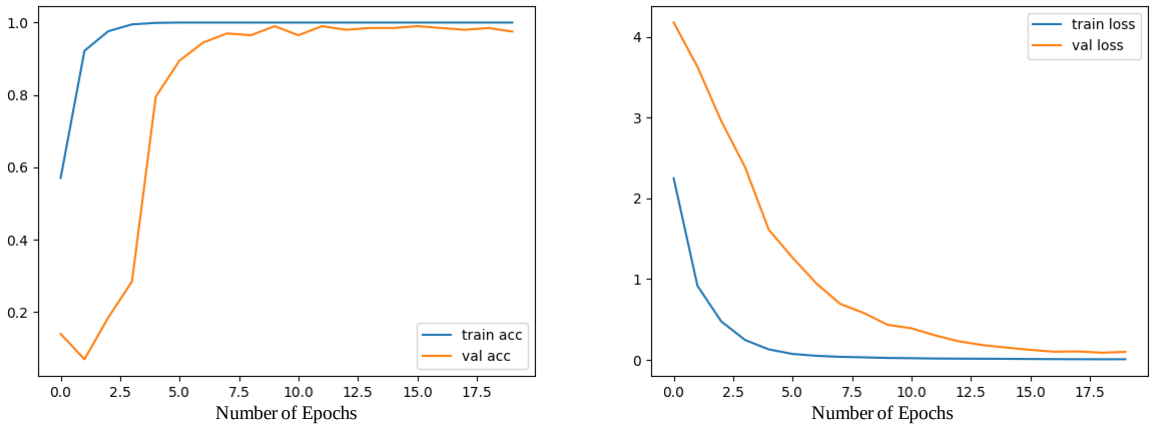


Train accuracy vs. validation accuracy



Train Loss vs. validation loss

Figure 19 : Graphs of training the top-left patch



Train accuracy vs. validation accuracy Train Loss vs. validation loss

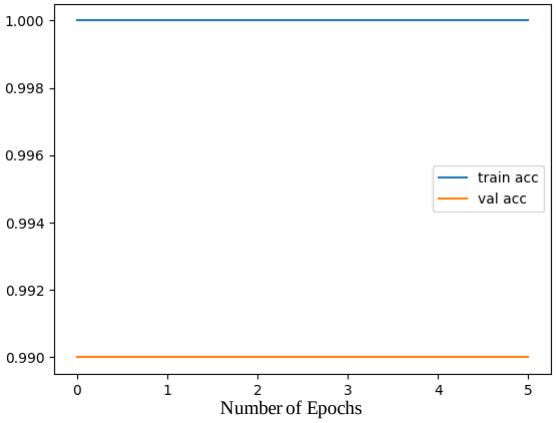
Figure 20 : Graphs of training the top-right patch

3.3.3 Fine tuning

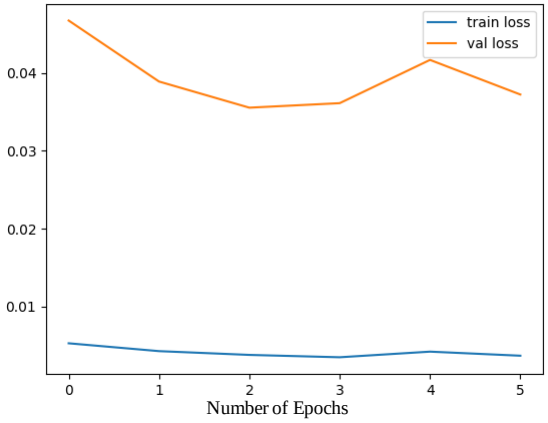
Fine tuning is part of the transfer learning technique in which we unfreeze a few or all of the layers from the pre-trained model after the previously shown steps of transfer learning. Fine-tuning the model gives better accuracy than already pre-trained models from scratch as the new model gets tailored to our requirements [73]. This can give meaningful improvements as it adapts the pre-trained features to the new data incrementally [39].

For this thesis, we unfroze all the layers from the pre-trained model FaceNet after training the newly added layers. Since, previously we set trainable to false, unfreezing the layer keeps the Batch normalization layers in inference mode. Hence, their weights are not updated. A very low learning rate is used here so that we do not overfit the model to our known test and validation set as this will cause the model to extract features poorly on unseen dataset. The learning rate used for fine-tuning our three models from the previous step is 1e-5. Increasing this gives good training and validation accuracy but give low face recognition accuracy on unseen dataset. Similar to the transfer learning step, we use the same early stopping method in this step to avoid overfitting.

We can see from the graphs below; the train accuracy and validation accuracy almost remain the same while fine tuning but the validation loss decreases. This gives better accuracy than if the models were not fine-tuned.

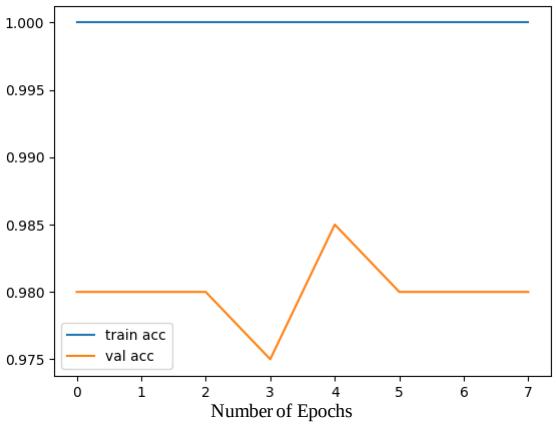


Train accuracy vs. validation accuracy

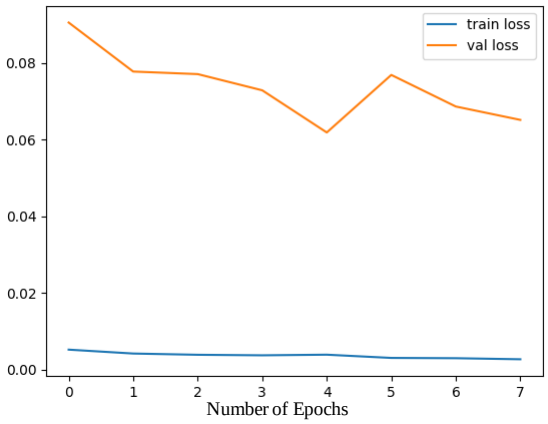


Train Loss vs. validation loss

Figure 21 : Graphs of fine tuning the top patch

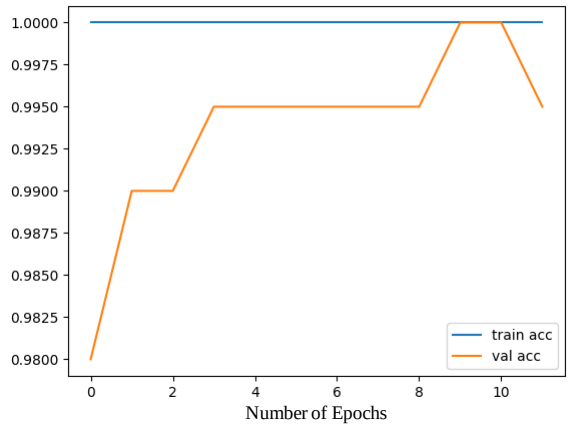


Train accuracy vs. validation accuracy

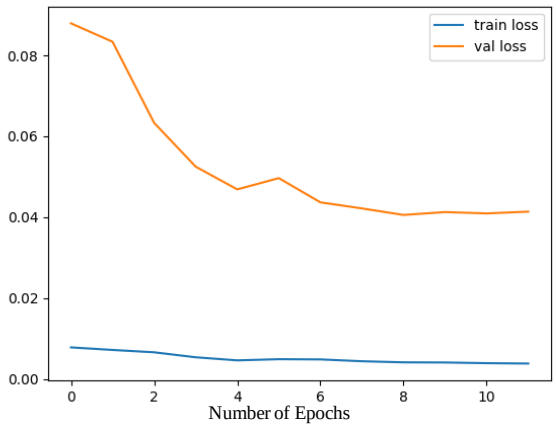


Train Loss vs. validation loss

Figure 22 : Graphs of fine tuning the top-left patch



Train accuracy vs. validation accuracy



Train Loss vs. validation loss

Figure 23 : Graphs of fine tuning the top-right patch

3.3.4 Feature extraction

Embeddings or feature vector of each image from the FEI and TUFTS dataset is obtained in this step using the models created in the previous step. Each model takes an input of face patch image in size 160*160. The output is taken from the second last layer of the models. This gives an embedding of 128 values which is the feature vector. The top, top-right and top-left models are used to extract top, top-right and top-left embeddings respectively. For face recognition Squared Euclidean distance between the reference embedding and probe embedding is calculated. When both reference and probe are of same person the Squared Euclidean distance is small and it is large when they are different.

3.4 Fusion strategy

Score-level fusion and feature-level fusion are done using multiple samples to reduce the equal error rate and improve the accuracy of biometric recognition [74], [75]. For this thesis both score-level and feature-level fusion is done to compare which gives better accuracy in our case.

3.4.1 Score-level fusion

Score-level fusion is done by generating a new match score by combining the match scores outputs from multiple biometric matchers. The match scores from every output are normalized before combining them and a weight is used for each match score output [74]. Score-level

fusion of multiple biometric systems gives a better accuracy than if only one biometric system is used [76].

For this thesis, match scores for each patch were calculated separately i.e., top patch match scores, top-left match score and top-right match scores, and saved as arrays. The match scores in this case are the Squared Euclidean distances between the probe and reference face patches. The three-match score arrays are then normalized using the min-max normalization individually. The min-max normalizing normalizes the data to a range of 0 to 1. It can be achieved by the following formula:

$$x_{normalized} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Where x is the values to be normalized.

The normalized match scores are then combined using the following formula:

$$score_{fused} = w_1 * score_{top} + w_2 * score_{top-left} + w_3 * score_{top-right}$$

Where w_1, w_2 and w_3 are weights assigned to each match score. And $w_1 + w_2 + w_3 = 1$

$Score_{fused}$ is the final similarity score which is then divided into Genuine scores and Imposter scores. Where Genuine score is the distance between the same probe and same reference person and Imposter score is the distance between different probe and reference person. From these scores, similarity score for each pair is calculated. Which is then used for calculating the EER, FNMR and FMR. The process is described in the next chapter.

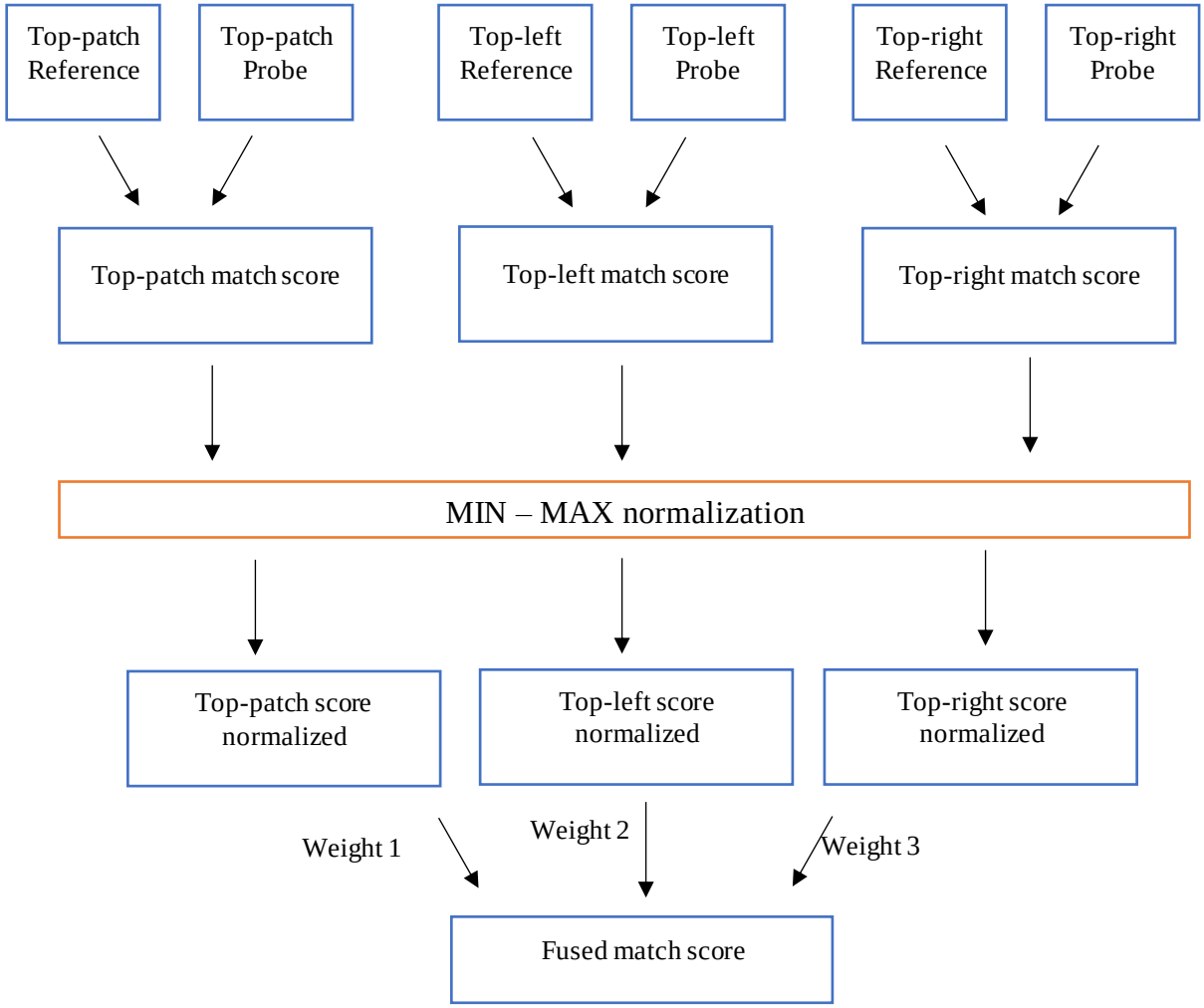


Figure 24: Flowchart of Score-level fusion

3.4.2 Feature-level fusion

Feature-level fusion is relatively difficult to achieve in practice because different biometric systems may have incompatible feature sets and may have unknown correspondence among different feature spaces. Fusion in some cases may be very complicated [75], [77].

In this thesis, we have three feature types with similar feature space. All our features are from face patches. So, feature fusion is easy in this case. For feature fusion each feature vector is normalized first. In our case, min-max normalization is used. Then, the three normalized feature vector is concatenated into one single feature vector [78]. For calculating the match score, the Squared Euclidean distance between fused feature vector of probe and fused feature vector of reference is calculated. This is then divided into Genuine scores and Imposter scores. From

these scores, score similarity score for each pair is calculated to determine the threshold for the scores which gives the least equal error rate. The Scores below the threshold are rejected and above the threshold are accepted by the system.

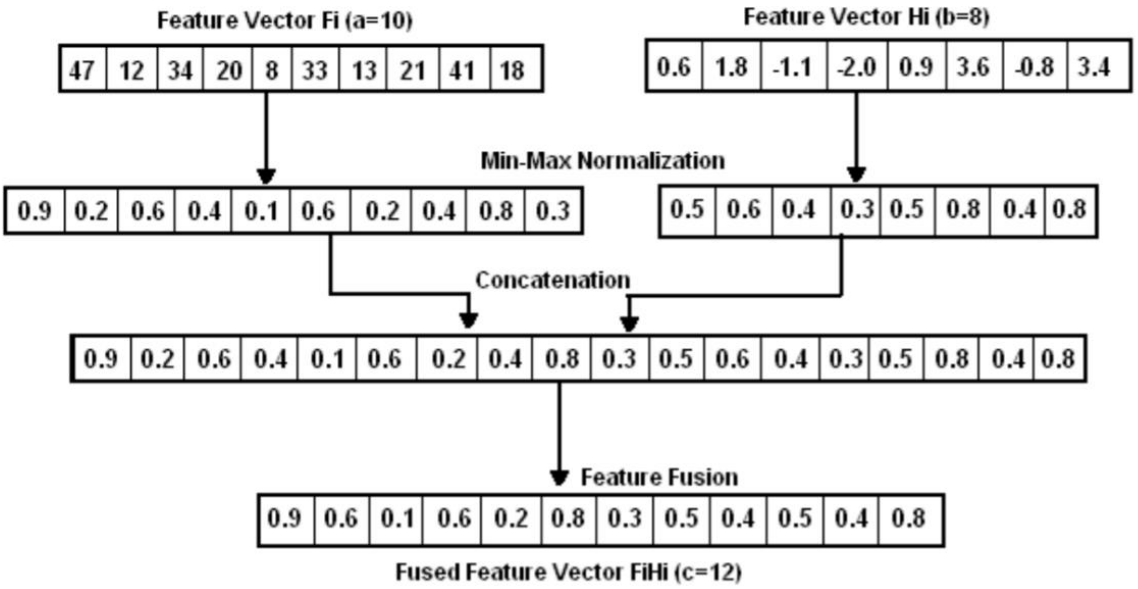


Figure 25: Example of feature level fusion of two feature vectors [78]

4. Experimental evaluation

The results from the experiment detailed above are discussed in this section. For evaluation of the models, two feature comparator methods were used. First method used was the SVM classifier and Squared Euclidean distance. It was used on the feature vectors which were extracted using the three models respectively. Second method used was to set a threshold for the similarity scores of the probe and references. The probes having low similarity scores than the threshold was rejected and the ones having more were accepted by the system. For improvement of the second method score-level and feature level fusion is done.

Table 1: Evaluated subjects

	FEI dataset	**TUFTS dataset**
No. of evaluation subjects	109	109
No. of reference samples	872	545
No. of probe samples	327	327

4.1 SVM and Squared Euclidean Distance evaluation

SVM

The feature vectors of the reference samples are separated into different classes using hyperplanes by SVM. The distance between the feature vector of each probe samples with the classes created from the reference samples were calculated. The probe sample was classified as the class it has the least distance with.

Squared Euclidean Distance

For Squared Euclidean Distance method the Squared Euclidean Distance between the feature vectors of each probe and reference is calculated. The probe is classified as the class of the reference it has the least Squared Euclidean Distance with.

For both SVM and Squared Euclidean Distance the accuracy is calculated by (the number of correctly classified probes / the total number of probe samples) *100

The SVM classifier yielded 95.107% accuracy for the top patch of FEI dataset and 92.966% accuracy for the top patch of TUFTS dataset. And Squared Euclidean distance for face recognition, yields an accuracy of 95.413% for FEI dataset and 93.272% accuracy for TUFTS dataset. This is better compared to the accuracy of top patch in the paper "Deep face recognition using imperfect facial data", which uses FEI dataset for training VGG-Face and LFW for testing. They propose a similar method of adding face patches to their train set. The accuracy their best algorithm yields is 90.2% which uses Cosine similarity [20]. Their method reaches 95% for three-fourth face images but since in our case we only have half of the face (fully masked face) we compare the results to their top half face results from their paper. Another paper that had a similar approach of performing face recognition on the top half of the face is the "Dynamic Feature Matching for Partial Face Recognition" where the maximum accuracy they gained for top half of the face is 46.3%. In paper "Efficient Masked Face Recognition Method during the COVID-19 Pandemic" transfer learning and fine tuning of CNN was used with Deep bag of features (BoF) technique. The result on this gave an accuracy of 88.9% on Simulated Masked Face Recognition Dataset while the accuracy using CNN and SVM was 86.1% [8], [79]. Although this difference may be because their test dataset had more images than the datasets used in this thesis. The SVM classifier test accuracies can be seen in Table 3 and Squared Euclidean distance face recognition accuracy can be seen in Table 4.

As it is observed from Table 3 and Table 4, in case of FEI datasets the accuracy improved for the Squared Euclidean distance significantly. This is because FEI datasets contain poses with faces rotated almost 180 degrees to left or right. When SVM is used, it is difficult for it to cluster these into classes as face images with left or right profile have different face features than that of forward-facing images. Same as the Cosine Similarity gave better results for face recognition than SVM for partial face recognition in [20]. Our Squared Euclidean distance approach was similar to the cosine similarity as both are distance metric.

Table 2: Comparison of recognition results with CNN and SVM techniques for the top half of the face or masked face.

	Top patch accuracy
He *et al.*, 2019	46.3%
Hariri, 2021, Almabdy and Elrefaei, 2019	86.1%
Elmahmudi and Ugail, 2019	90.2%
Proposed methodology accuracy (Average of both dataset accuracies)	**94.3%**

Table 3 : Accuracies of the SVM classifier face recognition on the different patches of FEI and TUFTS datasets.

	Patches	**Accuracy**
FEI dataset	Top half	95.107
	Top-left	74.924
	Top-right	66.667
TUFTS dataset	Top half	92.966
	Top-left	67.568
	Top-right	76.147

Table 4: Accuracies of the Squared Euclidean distance face recognition on the different patches of FEI and TUFTS datasets

	Patches	Accuracy
FEI dataset	Top half	95.413
	Top-left	86.238
	Top-right	80.428
TUFTS dataset	Top half	93.272
	Top-left	65.165
	Top-right	73.700

4.2 Face recognition algorithm evaluation

For a real-life face recognition system, we cannot only depend on face recognition systems using classifiers like SVM or distances like Cosine Similarity and Squared Euclidean distance. If this is done, imposters will get accepted to the system as whichever reference the probe imposter's feature vector has minimum distance with. We, therefore, need to set a threshold for rejecting maximum imposters and accepting genuine scores. This can be done by calculating the Genuine attempts rejected (False non-match rate) and Imposter attempts accepted (False Match Rate) for each threshold. Only reducing the Imposter attempts accepted will give a secure system but the number of Genuine attempts rejected will also be high. This will decrease the accuracy and efficiency of the system. Therefore, for a good system we choose the point where the curves of Genuine attempts reject and Imposter attempts accepted meet. This is where the genuine attempts rejected is minimum for the minimum imposter attempts accepted. This point gives us the Equal error rate (EER) and setting the threshold to this point gives maximum accuracy [21], [80].

4.2.1 Evaluation details

In order to compare the reference samples and probe samples, we calculate the distance between their feature vectors. From these distances, the similarity score is calculated by 1- normalized distance. Where, the distance is normalized using Min-Max normalization.

The similarity scores are saved as two separate text files, one with the genuine scores and the other with the imposters scores. Genuine scores are scores between the same subjects and imposter scores are scores between different subjects. The number of Genuine scores for the FEI dataset is 2,616 and TUFTS dataset is 1,638 and the number of Imposter scores for the FEI dataset is 282,528 and TUFTS dataset is 180,180.

The evaluation metrics are described below:

- **False non-match rate (FNMR)**

  False non-match is when the biometric matcher categorizes a pair (probe and reference) from the same individuals as coming from different individuals. The rate of false non-match is false non-match rate [81].

  False non-match rate is calculated for each threshold independently. Anything less than threshold is rejected by the system. Therefore, genuine scores below the threshold also gets rejected. Let, False non-match be the number of genuine scores which are less than the threshold. So, the False non-match rate is equal to (False non-match /total no. of genuine scores) *100.

- **False match rate (FMR)**

  False match is when the biometric matcher categorizes a pair (probe and reference) from the different individuals as coming from same individuals. The rate of False match is the False match rate [82].

  False match rate is calculated for each threshold independently. Anything equal to or above the threshold is accepted by the system. Therefore, imposter scores equal to or above the threshold also gets accepted. Let, False match be the number of imposter scores which are greater than or equal to the threshold. So, the False match rate is equal to (False match /total no. of imposter scores) *100.

- **Equal error rate (EER)**

  Equal error rate is the point at which the False non-match rate is minimum for the minimum False match rate. This point is where the graph of FNMR and FMR meet. So, EER is the point at which FNMR = FMR. The threshold at this point is the most optimal threshold.

- **ROC**

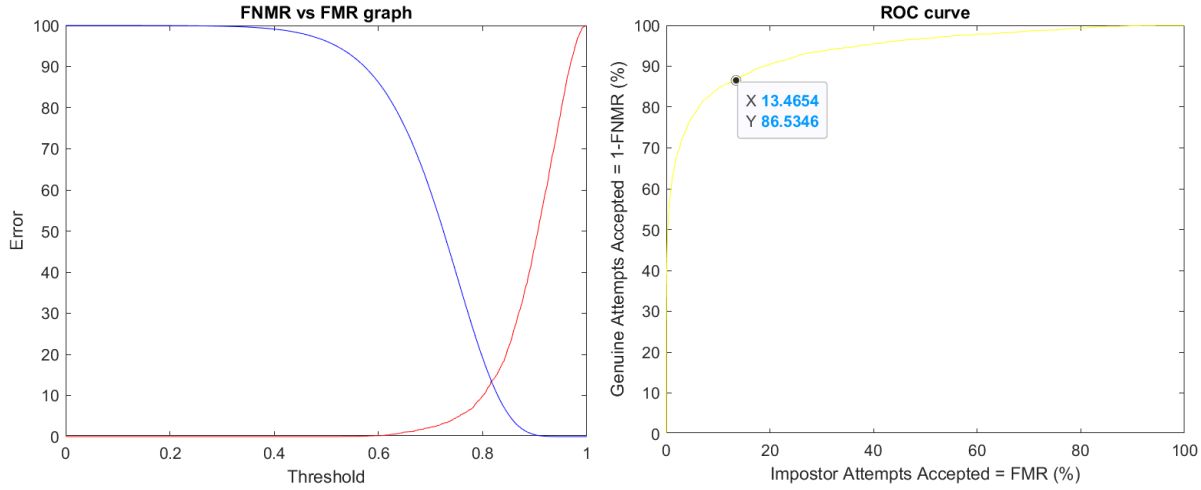
  ROC curve is the plot of FMR which is the Imposter attempts accepted vs 1-FNMR which is the Genuine attempts accepted [83].

The following graphs shows False non match rate vs. False match rate and ROC curves (ROC curves also shows the EER and accuracy) for each of our patches before any fusion is done (for better readability the following diagrams are presented in separate single pages):

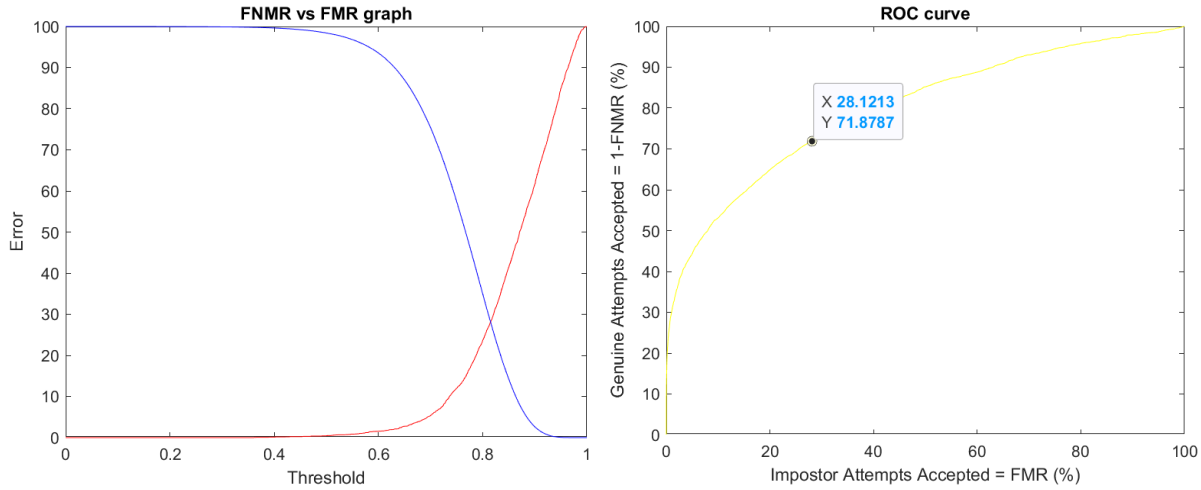
Table 5: False non match rate (FNMR) vs. False match rate (FMR) and ROC curve of the different patches of FEI dataset

FEI dataset

Top patch



Top-left patch



Top-right patch

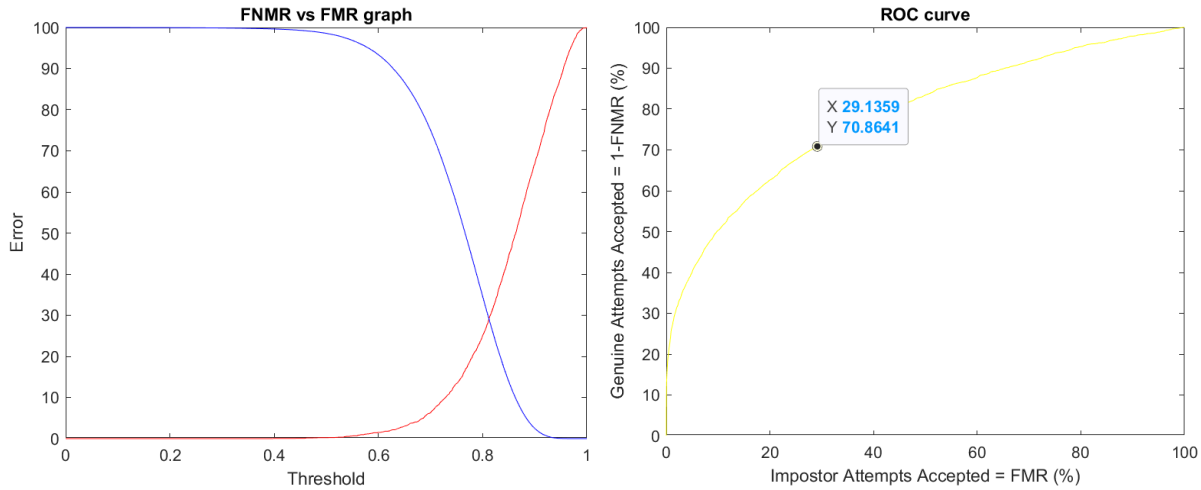
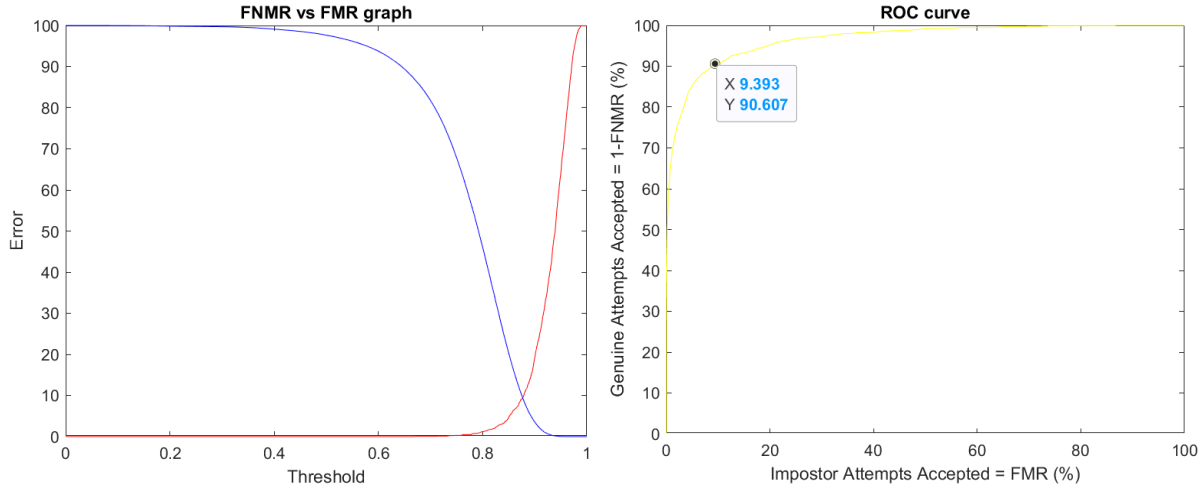


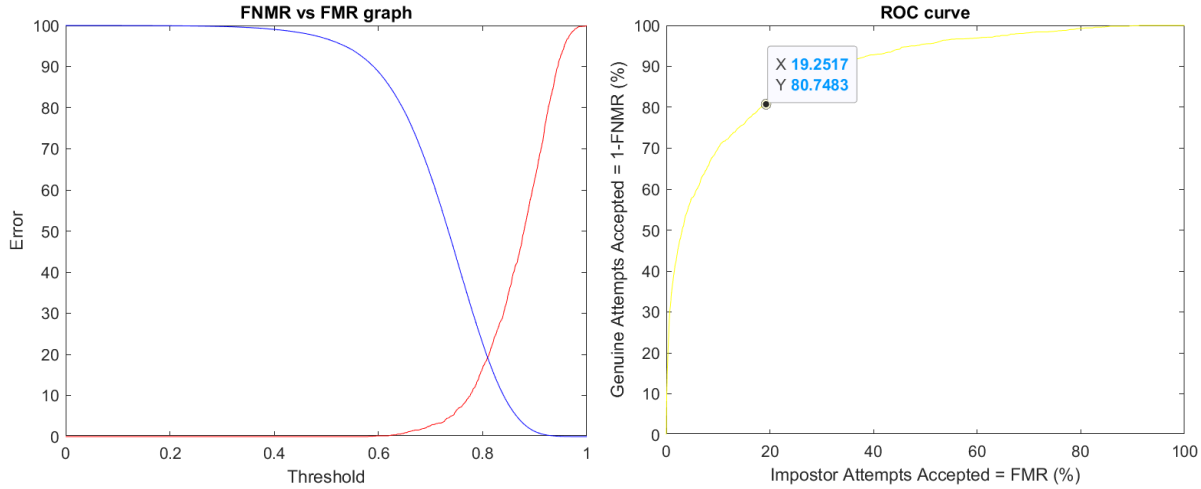
Table 6: False non match rate (FNMR) vs. False match rate (FMR) and ROC curve of the different patches of TUFTS dataset

TUFTS dataset

Top patch



Top-left patch



Top-right patch

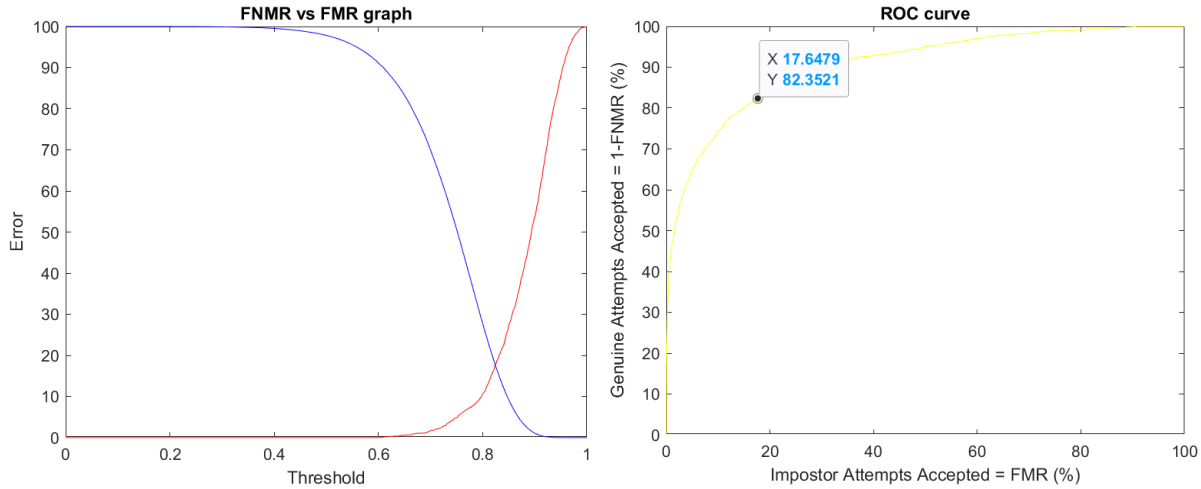


Table 7: EER and Accuracy of the models on FEI and TUFTS dataset

	Patches	EER
FEI dataset	Top half	13.4654
	Top-left	28.1213
	Top-right	29.1359
TUFTS dataset	Top half	9.3930
	Top-left	19.2517
	Top-right	17.6479

Contrary to the accuracy of SVM and Squared Euclidean distance on FEI dataset, this method performed poorly on FEI dataset compared to TUFTS dataset. This is because, as mentioned previously the FEI dataset have large differences in its poses. The feature vector of different poses (left/right profile and front profile) are different. So, the distance between a left/right profile image and front face image is large even though the images are of same person. Since, the method depends on Squared Euclidean distance between probe and reference images it gives high equal error rate for the FEI dataset compared to the TUFTS dataset.

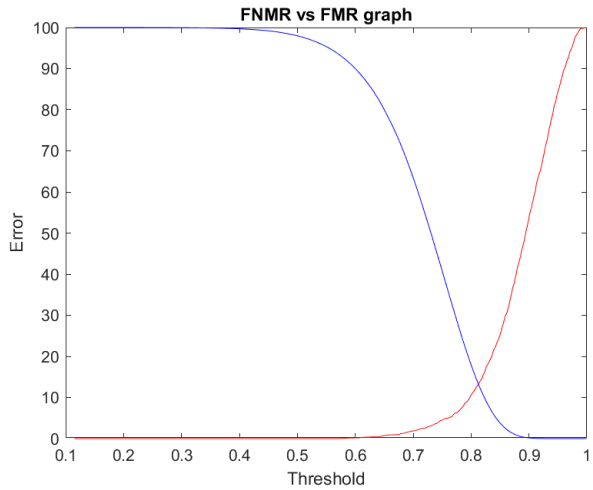
4.2.2 Score-level and Feature-level fusion evaluation

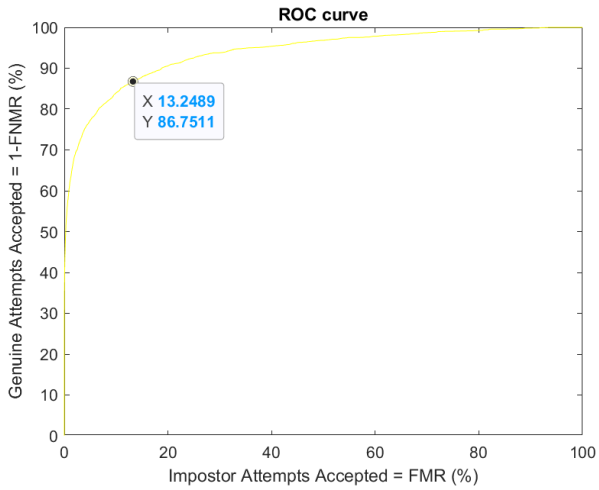
To improve the accuracy and reduce the EER we used score-level and feature-level fusion. The following graphs show False non match rate (FNMR) vs. False match rate (FMR) and ROC curves (ROC curves also shows the EER and accuracy) for score-level fusion and feature-level fusion of the three patch feature vectors:

Table 8: False non match rate (FNMR) vs. False match rate (FMR) and ROC curve of Score-level fusion on FEI and TUFTS dataset

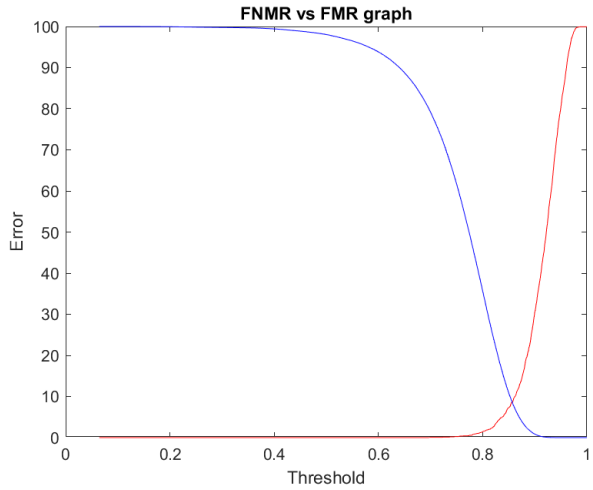
Score-level fusion

FEI dataset





TUFTS dataset



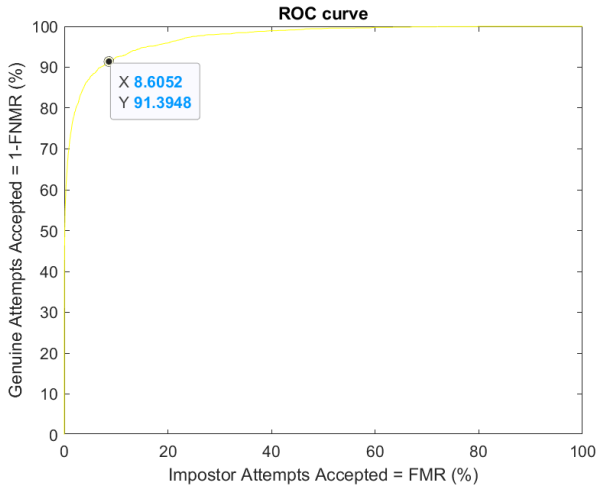
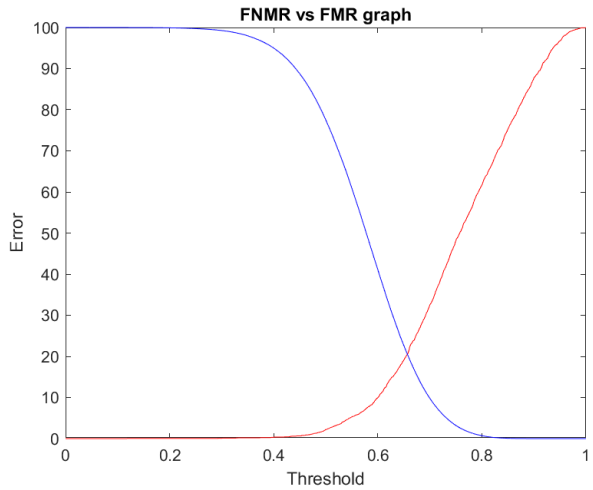
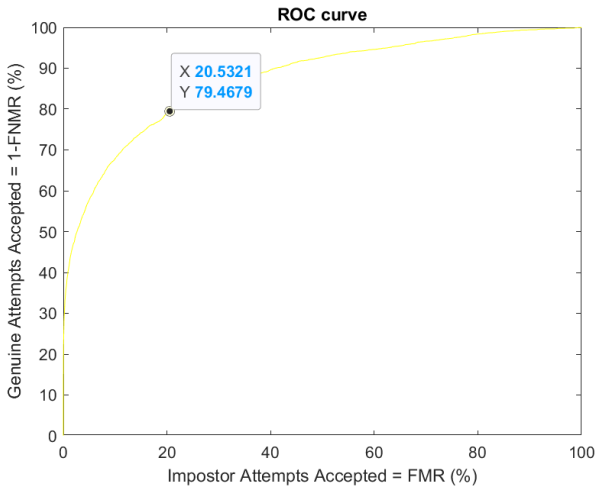


Table 9: False non match rate (FNMR) vs. False match rate (FMR) and ROC curve of Feature-level fusion on FEI and TUFTS dataset

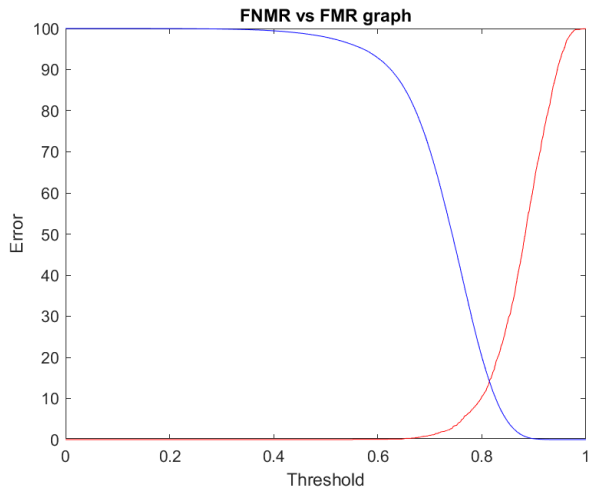
Feature-level fusion

FEI dataset





TUFTS dataset



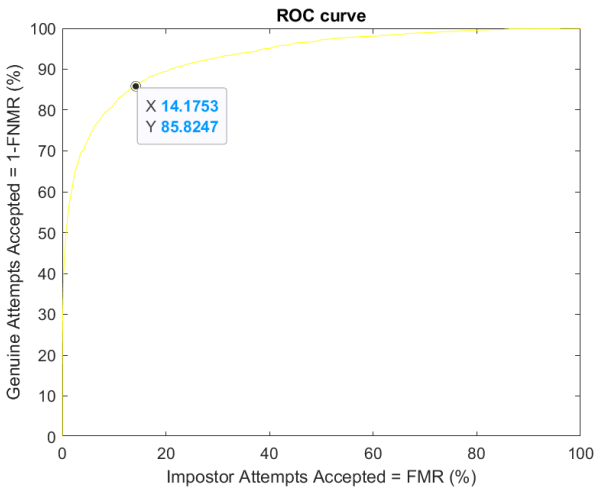


Table 10: EER and accuracy of score-level and feature-level fusion on FEI and TUFTS dataset

Fusion	Dataset	EER
Score-level	FEI dataset	**13.2489**
	TUFTS dataset	**8.6052**
Feature-level	FEI dataset	20.5321
	TUFTS dataset	14.1753

For score-level fusion the weights for the three fused scores were determined by trial-and-error method. It was seen that increasing the weight of the top patch and decreasing the top-left and top-right patch improved the accuracy from weights (w1=0.5, w2=0.25, w3=0.25) till (w1=0.9, w2=0.05, w3=0.05). After this the accuracy of FEI dataset started reducing. Weights of the top-left and top-right patches were chosen to be the same as accuracy of these patches were almost the similar to each other for both FEI and TUFTS dataset as it can be seen from the results before.

Table 10 shows that the Score-level fusion gave a better accuracy than feature-level vector in our case. Score-level fusion accuracy is also better than the accuracy we had before. Feature-level fusion did not work in our case because the top-left and top-right accuracy was not good compared to the top patch. For feature-level fusion we are fusing all the three features without any weights which gives then all equal weights. Therefore, the accuracy drops due to the low accuracies if top-left and top-right patch. On the other hand, for the score-fusion we set less weights on the lower accuracy top-left and top-right patches. This therefore, gives us better results.

5. Conclusion

The experimental evaluation of this thesis, we believe, confirms that patch-based transfer learning and fine tuning a CNN based pre-trained model can be used for partial face recognition with high accuracy. As seen from the experiments performed in this thesis, masked face recognition can be successfully performed using deep learning. Face recognition for only parts of the faces like top-left or top-right can also be done with an accuracy of up to 80% on some databases. This answers our research questions: To what extent can partial-face or face covered with mask be recognized? And can patch-based deep learning be used for partial-face recognition.

This thesis incorporated the methods of using a 3D face masking tool to create masked datasets, face detection and alignment on the masked datasets, a patch-based transfer learning and fine tuning of a pre-trained deep learning model (FaceNet), feature extraction using the trained models on different test datasets, and score-level and feature-level fusion of the different patches.

KomNET dataset does not have too many pose variations i.e., there are no 180-degree rotation side profiles, and more than half of the images in FEI are of left and right profiles. We separated the training dataset and testing dataset by using two completely different datasets, KomNET dataset for training and FEI dataset for testing, we think that is the main reason for the low performances [18]. The left and right profiles gave different features than the front profiles. Therefore, there were low similarity scores (large distances) between some pairs from the same persons. The performance on datasets with large pose variations can be improved by using datasets with large pose variations for training the models.

The performance on TUFTS dataset was better compared to the FEI dataset and gave a final accuracy 91.3948 % which still has a large EER of 8.6052 %. Apart from the reason mentioned above about having different train and test datasets, another reason for this might be because the image quality of TUFTS is lower than the other datasets used. When it is scaled to 160*160 pixels some of the images are blurry. To have FaceNet perform better on low image qualities we have to add low quality images while training [60]. This will help us improve the accuracy further.

The lighting used in both FEI and KomNET datasets are similar, whereas the lighting used for TUFTS dataset is different. Therefore, another way to improve the performance on datasets like TUFTS can be to use light changes as data augmentation or use different lighting images while training the models. As a difference in illumination causes the face recognition accuracy to drop [18].

Even though our models were trained with only 50 classes consisting of 1200 images in total (both train and validation together) our accuracy reached to 86.7511% for FEI and 91.3948 % TUFTS. Thus, increasing the train data for the transfer learning and fine tuning will increase the accuracy even further [37].

From the experiments in this thesis, we can conclude that partial face recognition can be achieved with high accuracy using patch-based deep learning.

6. Future work

The performance of the methodology in this thesis used can be improved by increasing the number of training images by:

- Using data augmentation while training the models specially for different lighting conditions
- Increasing the number of classes and images per person while training the models
- Using a large variety of poses for the training dataset
- Using both good quality and bad quality images for the training

A way of possibly improving the method used in this thesis is to use a different loss function than the triplet loss function in the model. As triplet loss function is highly sensitive to noise using a hierarchical triplet loss function can possibly improve the accuracy [64], [84]. Using quadruplet loss function can perhaps improve the accuracy as well as triplet loss function can cause a relatively large intra-class variation and reducing this intra-class variation and enlarging the inter-class variation can improve the model accuracy [85]–[87].

Increasing the number of patches and making them overlapping like in the paper "Patch strategy for deep face recognition" can also improve the accuracy as the models will be able to learn more underlying features [6].

An extension of this experiment can be to use the proposed methods on other partial face images. This can be used for recognizing people from low resolution cameras or different angle cameras like CCTV footages.

7. References

[1] N. Damer, J. H. Grebe, C. Chen, F. Boutros, F. Kirchbuchner, and A. Kuijper, "The Effect of Wearing a Mask on Face Recognition Performance: an Exploratory Study," *BIOSIG 2020 - Proc. 19th Int. Conf. Biometrics Spec. Interes. Gr.*, Jul. 2020, [Online]. Available: http://arxiv.org/abs/2007.13521.

[2] "Criminals use coronavirus masks to conceal themselves," *Al Arabiya*, 2020.

[3] D. Babwin and S. Dazio, "Coronavirus Masks a Boon for Crooks Who Hide Their Faces," *NBC Miami*, 2020.

[4] L. He, H. Li, Q. Zhang, and Z. Sun, "Dynamic Feature Matching for Partial Face Recognition," *IEEE Trans. Image Process.*, vol. 28, no. 2, pp. 791–802, Feb. 2019, doi: 10.1109/TIP.2018.2870946.

[5] R. Weng, J. Lu, J. Hu, G. Yang, and Y.-P. Tan, "Robust Feature Set Matching for Partial Face Recognition," in *2013 IEEE International Conference on Computer Vision*, Dec. 2013, pp. 601–608, doi: 10.1109/ICCV.2013.80.

[6] Y. Zhang, K. Shang, J. Wang, N. Li, and M. M. Y. Zhang, "Patch strategy for deep face recognition," *IET Image Process.*, vol. 12, no. 5, pp. 819–825, May 2018, doi: 10.1049/iet-ipr.2017.1085.

[7] I. Cheheb, N. Al-Maadeed, S. Al-Madeed, A. Bouridane, and R. Jiang, "Random sampling for patch-based face recognition," in *2017 5th International Workshop on Biometrics and Forensics (IWBF)*, Apr. 2017, pp. 1–5, doi: 10.1109/IWBF.2017.7935104.

[8] W. Hariri, "Efficient Masked Face Recognition Method during the COVID-19 Pandemic," May 2021, doi: 10.21203/rs.3.rs-39289/v1.

[9] J. Deng, J. Guo, Y. Zhou, J. Yu, I. Kotsia, and S. Zafeiriou, "RetinaFace: Single-stage Dense Face Localisation in the Wild," *arXiv*, May 2019, [Online]. Available: http://arxiv.org/abs/1905.00641.

[10] A. Elmahmudi and H. Ugail, "Experiments on Deep Face Recognition Using Partial Faces," in *2018 International Conference on Cyberworlds (CW)*, Oct. 2018, pp. 357–362, doi: 10.1109/CW.2018.00071.

[11] D. Misra, C. Crispim-Junior, and L. Tougne, "Patch-Based CNN Evaluation for Bark

Classification," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12540 LNCS, Edinburgh, United Kingdom, 2020, pp. 197–212.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

[13] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017, doi: 10.1109/TPAMI.2016.2577031.

[14] C. Sun, Y. Yang, C. Wen, K. Xie, and F. Wen, "Voiceprint Identification for Limited Dataset Using the Deep Migration Hybrid Model Based on Transfer Learning," *Sensors*, vol. 18, no. 7, p. 2399, Jul. 2018, doi: 10.3390/s18072399.

[15] J. Li, T. Qiu, C. Wen, K. Xie, and F.-Q. Wen, "Robust Face Recognition Using the Deep C2D-CNN Model Based on Decision-Level Fusion," *Sensors*, vol. 18, no. 7, p. 2080, Jun. 2018, doi: 10.3390/s18072080.

[16] Y.-X. Yang, C. Wen, K. Xie, F.-Q. Wen, G.-Q. Sheng, and X.-G. Tang, "Face Recognition Using the SR-CNN Model," *Sensors*, vol. 18, no. 12, p. 4237, Dec. 2018, doi: 10.3390/s18124237.

[17] I. Masi, Y. Wu, T. Hassner, and P. Natarajan, "Deep Face Recognition: A Survey," in *2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, Oct. 2018, pp. 471–478, doi: 10.1109/SIBGRAPI.2018.00067.

[18] Changbo Hu, J. Harguess, and J. K. Aggarwal, "Patch-based face recognition from video," in *2009 16th IEEE International Conference on Image Processing (ICIP)*, Nov. 2009, no. December 2009, pp. 3321–3324, doi: 10.1109/ICIP.2009.5413935.

[19] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 1717–1724, doi: 10.1109/CVPR.2014.222.

[20] A. Elmahmudi and H. Ugail, "Deep face recognition using imperfect facial data," *Futur. Gener. Comput. Syst.*, vol. 99, pp. 213–225, Oct. 2019, doi: 10.1016/j.future.2019.04.025.

[21] M. J. Sudhamani, M. K. Venkatesha, and K. R. Radhika, "Revisiting feature level and score level fusion techniques in multimodal biometrics system," in *2012 International Conference on Multimedia Computing and Systems*, May 2012, pp. 881–885, doi: 10.1109/ICMCS.2012.6320155.

[22] F. Wang and J. Han, "Multimodal biometric authentication based on score level fusion using support vector machine," *Opto-Electronics Rev.*, vol. 17, no. 1, Jan. 2009, doi: 10.2478/s11772-008-0054-8.

[23] Yongsheng Gao and M. Maggs, "Feature-Level Fusion in Personal Identification," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005, vol. 1, pp. 468–473, doi: 10.1109/CVPR.2005.159.

[24] J. Guo and J. Deng, "InsightFace: a Deep Learning Toolkit for Face Analysis." http://insightface.ai/index.html (accessed Jan. 15, 2021).

[25] Y. Feng, "Face3d," *Github*. https://github.com/YadiraF/face3d (accessed Jan. 16, 2021).

[26] InsightFace, "Face Mask Renderer tool," *Github*. https://github.com/deepinsight/insightface/tree/master/recognition/tools (accessed Jan. 15, 2021).

[27] Z. Wang *et al.*, "Masked Face Recognition Dataset and Application," Mar. 2020, [Online]. Available: http://arxiv.org/abs/2003.09093.

[28] J. J. Lee, U. J. Gim, J. H. Kim, K. H. Yoo, Y. H. Park, and A. Nasridinov, "Identifying customer interest from surveillance camera based on deep learning," in *Proceedings - 2020 IEEE International Conference on Big Data and Smart Computing, BigComp 2020*, Feb. 2020, pp. 19–20, doi: 10.1109/BigComp48618.2020.0-105.

[29] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks," *IEEE Signal Process. Lett.*, vol. 23, no. 10, pp. 1499–1503, Oct. 2016, doi: 10.1109/LSP.2016.2603342.

[30] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks," *IEEE Signal Process. Lett.*, vol. 23, no. 10, pp. 1499–1503, Oct. 2016, doi: 10.1109/LSP.2016.2603342.

[31] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, "VGGFace2: A Dataset for Recognising Faces across Pose and Age," in *2018 13th IEEE International Conference on*

Automatic Face & Gesture Recognition (FG 2018), May 2018, pp. 67–74, doi: 10.1109/FG.2018.00020.

[32] X. Guo and J. Nie, "Face Recognition System for Complex Surveillance Scenarios," *J. Phys. Conf. Ser.*, vol. 1544, no. 1, May 2020, doi: 10.1088/1742-6596/1544/1/012146.

[33] T.-X. Jiang, T.-Z. Huang, X.-L. Zhao, and T.-H. Ma, "Patch-Based Principal Component Analysis for Face Recognition," *Comput. Intell. Neurosci.*, vol. 2017, pp. 1–9, 2017, doi: 10.1155/2017/5317850.

[34] Z. Xu, Y. Liu, M. Ye, L. Huang, H. Yu, and X. Chen, "Patch Based Collaborative Representation with Gabor Feature and Measurement Matrix for Face Recognition," *Math. Probl. Eng.*, vol. 2018, pp. 1–13, 2018, doi: 10.1155/2018/3025264.

[35] A. Vijayan, S. Kareem, and J. J. Kizhakkethottam, "Face Recognition Across Gender Transformation Using SVM Classifier," *Procedia Technol.*, vol. 24, pp. 1366–1373, 2016, doi: 10.1016/j.protcy.2016.05.150.

[36] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, "A convolutional neural network cascade for face detection," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, vol. 07-12-June, pp. 5325–5334, doi: 10.1109/CVPR.2015.7299170.

[37] S. T. Krishna and H. K. Kalluri, "Deep learning and transfer learning approaches for image classification," *Int. J. Recent Technol. Eng.*, vol. 7, no. 5S4, pp. 427–432, 2019.

[38] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010, doi: 10.1109/TKDE.2009.191.

[39] F. Chollet, "Transfer learning & fine-tuning," *Keras*, 2020. https://keras.io/guides/transfer_learning/.

[40] R. Ahdid, S. Safi, and B. Manaut, *Euclidean and geodesic distance between a facial feature points in two-dimensional face recognition system*, vol. 14, no. 4A Special Issue. 2017.

[41] A. C. Lorena, A. C. P. L. F. de Carvalho, and J. M. P. Gama, "A review on the combination of binary classifiers in multiclass problems," *Artif. Intell. Rev.*, vol. 30, no. 1–4, pp. 19–37, Dec. 2008, doi: 10.1007/s10462-009-9114-9.

[42] A. D. Back, "Multiclass Classification Using Support Vector Machines," *Baeldung*, 2020. https://www.baeldung.com/cs/svm-multiclass-classification (accessed Dec. 09, 2020).

[43] N. H. Spencer, *Essentials of Multivariate Data Analysis*. Chapman and Hall/CRC, 2013.

[44] M. K. Hossain and S. Abufardeh, "A new method of calculating squared euclidean distance (SED) using PTreE technology and its performance analysis," in *Proceedings of 34th International Conference on Computers and Their Applications, CATA 2019*, 2019, pp. 45–54, doi: 10.29007/trrg.

[45] I. N. G. A. Astawa, I. K. G. D. Putra, M. Sudarma, and R. S. Hartati, "KomNET: Face Image Dataset from Various Media for Face Recognition," *Data Br.*, vol. 31, p. 105677, Aug. 2020, doi: 10.1016/j.dib.2020.105677.

[46] C. E. Thomaz and G. A. Giraldi, "A new ranking method for principal components analysis and its application to face image analysis," *Image Vis. Comput.*, vol. 28, no. 6, pp. 902–913, Jun. 2010, doi: 10.1016/j.imavis.2009.11.005.

[47] E. Z. Tenorio and C. E. Thomaz, "Analise Multilinear Discriminante De Formas Frontais De Imagens 2D De Face," in *X Sbai - Simpósio Brasileiro de Automação Inteligente*, 2011, vol. X, pp. 1043–1048.

[48] V. do Amaral, C. Fígaro-Garcia, G. J. F. Gattas, and C. E. Thomaz, "Normalização espacial de imagens frontais de face em ambientes controlados e não-controlados," *FaSCi-Tech*, vol. 1, no. 1, 2009, [Online]. Available: http://fatecsaocaetano.edu.br/fascitech/index.php/fascitech/article/view/9/8.

[49] V. Amaral and C. E. Thomaz, "Normalizacao Espacial de Imagens Frontais de Face - Technical Report," São Paulo, Brazil, 2008.

[50] L. L. de O. Junior and C. E. Thomaz, "Captura e Alinhamento de Imagens: Um Banco de Faces Brasileiro - Undergraduate Technical Report," São Paulo, Brazil, 2006.

[51] K. Panetta *et al.*, "A Comprehensive Database for Benchmarking Imaging Systems," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 3, pp. 509–520, Mar. 2020, doi: 10.1109/TPAMI.2018.2884458.

[52] J. Cho, K. Lee, E. Shin, G. Choy, and S. Do, "How much data is needed to train a medical image deep learning system to achieve necessary high accuracy?," Nov. 2015, [Online]. Available: http://arxiv.org/abs/1511.06348.

[53] P. Paysan, R. Knothe, B. Amberg, S. Romdhani, and T. Vetter, "A 3D Face Model for Pose and Illumination Invariant Face Recognition," in *2009 Sixth IEEE International Conference on*

Advanced Video and Signal Based Surveillance, Sep. 2009, pp. 296–301, doi: 10.1109/AVSS.2009.58.

[54] Xiangyu Zhu, Z. Lei, Junjie Yan, D. Yi, and S. Z. Li, "High-fidelity Pose and Expression Normalization for face recognition in the wild," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, vol. 07-12-June, pp. 787–796, doi: 10.1109/CVPR.2015.7298679.

[55] A. Bas, P. Huber, W. A. P. Smith, M. Awais, and J. Kittler, "3D Morphable Models as Spatial Transformer Networks," in *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, Oct. 2017, vol. 2018-Janua, pp. 895–903, doi: 10.1109/ICCVW.2017.110.

[56] E. Kaziakhmedov, K. Kireev, G. Melnikov, M. Pautov, and A. Petiushko, "Real-world Attack on MTCNN Face Detection System," in *2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*, Oct. 2019, pp. 0422–0427, doi: 10.1109/SIBIRCON48586.2019.8958122.

[57] J. Deng, J. Guo, Y. Zhou, J. Yu, I. Kotsia, and S. Zafeiriou, "RetinaFace Face Detector," *Github*. https://github.com/deepinsight/insightface/tree/master/detection/RetinaFace (accessed Feb. 01, 2021).

[58] S. I. Sergil, "Face Alignment for Face Recognition in Python within OpenCV," 2020. https://sefiks.com/2020/02/23/face-alignment-for-face-recognition-in-python-within-opencv/.

[59] D. Hacker, "How to align faces with OpenCV in Python," 2020. http://datahacker.rs/010-how-to-align-faces-with-opencv-in-python/.

[60] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, vol. 07-12-June, pp. 815–823, doi: 10.1109/CVPR.2015.7298682.

[61] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, vol. 2016-Decem, pp. 2818–2826, doi: 10.1109/CVPR.2016.308.

[62] R. M. Kamble *et al.*, "Automated Diabetic Macular Edema (DME) Analysis using Fine Tuning with Inception-Resnet-v2 on OCT Images," in *2018 IEEE-EMBS Conference on Biomedical*

Engineering and Sciences (IECBES), Dec. 2018, pp. 442–446, doi: 10.1109/IECBES.2018.8626616.

[63] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," *31st AAAI Conf. Artif. Intell. AAAI 2017*, pp. 4278–4284, Feb. 2016, [Online]. Available: http://arxiv.org/abs/1602.07261.

[64] I. William, D. R. Ignatius Moses Setiadi, E. H. Rachmawanto, H. A. Santoso, and C. A. Sari, "Face Recognition using FaceNet (Survey, Performance Test, and Comparison)," in *2019 Fourth International Conference on Informatics and Computing (ICIC)*, Oct. 2019, pp. 1–6, doi: 10.1109/ICIC47613.2019.8985786.

[65] D. Sandberg, "Face recognition using Tensorflow," *Github*, 2018. https://github.com/davidsandberg/facenet (accessed Feb. 27, 2021).

[66] J. Brownlee, *Deep Learning for Computer - Vision Image Classification , Object Detection , and Face Recognition in Python UNLOCK Computer Vision With Deep Learning*. 2019.

[67] H. Taniai, "Keras-facenet," *Github*. https://github.com/nyoki-mtl/keras-facenet (accessed Mar. 01, 2021).

[68] S. Gadicherla, "Tensorflow Vs Keras? — Comparison by building a model for image classification.," *Hackermoon*, 2020. https://hackernoon.com/tensorflow-vs-keras-comparison-by-building-a-model-for-image-classification-f007f336c519.

[69] M. T. Rosenstein, Z. Marx, L. P. Kaelbling, and T. G. Dietterich, "To Transfer or Not To Transfer," *NIPS 2005 Work. Transf. Learn.*, 2005.

[70] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical Evaluation of Rectified Activations in Convolutional Network," May 2015, [Online]. Available: http://arxiv.org/abs/1505.00853.

[71] J. Brownlee, *Better Deep Learning. Train Faster, Reduce Overfitting, and Make Better Predictions*, vol. 1, no. 2. 2018.

[72] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Dec. 2014, [Online]. Available: http://arxiv.org/abs/1412.6980.

[73] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using Deep Learning for Image-Based Plant Disease Detection," *Front. Plant Sci.*, vol. 7, Sep. 2016, doi: 10.3389/fpls.2016.01419.

[74] A. Ross and K. Nandakumar, "Fusion, Score-Level," in *Encyclopedia of Biometrics*, Boston,

MA: Springer US, 2009, pp. 611–616.

[75] A. Rattani, D. R. Kisku, M. Bicego, and M. Tistarelli, "Feature Level Fusion of Face and Fingerprint Biometrics," in *2007 First IEEE International Conference on Biometrics: Theory, Applications, and Systems*, Sep. 2007, pp. 1–6, doi: 10.1109/BTAS.2007.4401919.

[76] P. Drozdowski, N. Wiegand, C. Rathgeb, and C. Busch, "Score Fusion Strategies in Single-Iris Dual-Probe Recognition Systems," in *Proceedings of the 2018 2nd International Conference on Biometric Engineering and Applications - ICBEA '18*, 2018, pp. 13–17, doi: 10.1145/3230820.3230823.

[77] A. A. Ross and R. Govindarajan, "Feature level fusion of hand and face biometrics," in *Biometric Technology for Human Identification II*, Mar. 2005, vol. 5779, no. March, p. 196, doi: 10.1117/12.606093.

[78] S. K. Bhardwaj, "An Algorithm for Feature Level Fusion in Multimodal Biometric System," *Int. J. Adv. Res. Comput. Eng. Technol.*, vol. 3, no. 10, pp. 3499–3503, 2014.

[79] S. Almabdy and L. Elrefaei, "Deep Convolutional Neural Network-Based Approaches for Face Recognition," *Appl. Sci.*, vol. 9, no. 20, p. 4397, Oct. 2019, doi: 10.3390/app9204397.

[80] M. E. Schuckers, *Computational Methods in Biometric Authentication*. London: Springer London, 2010.

[81] M. E. Schuckers, "False Non-Match Rate," in *Computational Methods in Biometric Authentication*, 2010, pp. 47–96.

[82] M. E. Schuckers, "False Match Rate," in *Computational Methods in Biometric Authentication*, 2010, pp. 97–153.

[83] M. E. Schuckers, "Receiver Operating Characteristic Curve and Equal Error Rate," in *Computational Methods in Biometric Authentication*, 2010, pp. 155–204.

[84] W. Ge, W. Huang, D. Dong, and M. R. Scott, "Deep Metric Learning with Hierarchical Triplet Loss," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, pp. 272–288.

[85] W. Chen, X. Chen, J. Zhang, and K. Huang, "Beyond Triplet Loss: A Deep Quadruplet Network for Person Re-identification," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, vol. 2017-Janua, pp. 1320–1329, doi: 10.1109/CVPR.2017.145.

[86] D. Cheng, Y. Gong, S. Zhou, J. Wang, and N. Zheng, "Person Re-identification by Multi-Channel Parts-Based CNN with Improved Triplet Loss Function," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, vol. 2016-Decem, pp. 1335–1344, doi: 10.1109/CVPR.2016.149.

[87] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, "A Discriminative Feature Learning Approach for Deep Face Recognition," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016, pp. 499–515.

Appendix

Appendix A – Codes

Before running the codes use pip to install all the imported packages

A1- Code example of creating masked face dataset

- Download and install the libraries and models required following the instructions from https://github.com/deepinsight/insightface/tree/master/recognition/tools

- Edit the code mask_renderer.py depending on your dataset, an example of the code used for this thesis is given below.

```
# this program is used to create masked face datasets


import os, sys, datetime
import numpy as np
import os.path as osp
import face3d
from face3d import mesh
from face3d.morphable_model import MorphabelModel
import cv2


class MaskRenderer:
    def __init__(self, model_dir, render_only=False):
        self.bfm = MorphabelModel(osp.join(model_dir, 'BFM.mat'))
        self.index_ind = self.bfm.kpt_ind
        uv_coords =
face3d.morphable_model.load.load_uv_coords(osp.join(model_dir,
'BFM_UV.mat'))
        self.uv_size = (224, 224)
        self.mask_stxr = 0.1
        self.mask_styr = 0.33
        self.mask_etxr = 0.9
        self.mask_etyr = 0.7
        self.tex_h, self.tex_w, self.tex_c = self.uv_size[1],
self.uv_size[0], 3
        texcoord = np.zeros_like(uv_coords)
        texcoord[:, 0] = uv_coords[:, 0] * (self.tex_h - 1)
        texcoord[:, 1] = uv_coords[:, 1] * (self.tex_w - 1)
        texcoord[:, 1] = self.tex_w - texcoord[:, 1] - 1
        self.texcoord = np.hstack((texcoord, np.zeros((texcoord.shape[0],
1))))
        self.X_ind = self.bfm.kpt_ind
        if not render_only:
            from image_3d68 import Handler
            self.if3d68_handler = Handler(osp.join(model_dir, 'if1k3d68'),
```

```
0, 192, ctx_id=0)

    def transform(self, shape3D, R):
        s = 1.0
        shape3D[:2, :] = shape3D[:2, :]
        shape3D = s * np.dot(R, shape3D)
        return shape3D

    def preprocess(self, vertices, w, h):
        R1 = mesh.transform.angle2matrix([0, 180, 180])
        t = np.array([-w // 2, -h // 2, 0])
        vertices = vertices.T
        vertices += t
        vertices = self.transform(vertices.T, R1).T
        return vertices

    def project_to_2d(self, vertices, s, angles, t):
        transformed_vertices = self.bfm.transform(vertices, s, angles, t)
        projected_vertices = transformed_vertices.copy()  # using stantard
camera & orth projection
        return projected_vertices[self.bfm.kpt_ind, :2]

    def params_to_vertices(self, params, H, W):
        fitted_sp, fitted_ep, fitted_s, fitted_angles, fitted_t = params
        fitted_vertices = self.bfm.generate_vertices(fitted_sp, fitted_ep)
        transformed_vertices = self.bfm.transform(fitted_vertices,
fitted_s, fitted_angles,
                                                  fitted_t)
        transformed_vertices = self.preprocess(transformed_vertices.T, W,
H)
        image_vertices = mesh.transform.to_image(transformed_vertices, H,
W)
        return image_vertices

    def build_params(self, face_image):

        landmark = self.if3d68_handler.get(face_image)[:, :2]
        # print(landmark.shape, landmark.dtype)
        if landmark is None:
            return None  # face not found
        fitted_sp, fitted_ep, fitted_s, fitted_angles, fitted_t =
self.bfm.fit(landmark, self.X_ind, max_iter=3)
        return [fitted_sp, fitted_ep, fitted_s, fitted_angles, fitted_t]

    def generate_mask_uv(self, mask, positions):
        uv_size = (self.uv_size[1], self.uv_size[0], 3)
        h, w, c = uv_size
        uv = np.zeros(shape=(self.uv_size[1], self.uv_size[0], 3),
dtype=np.uint8)
        stxr, styr = positions[0], positions[1]
        etxr, etyr = positions[2], positions[3]
        stx, sty = int(w * stxr), int(h * styr)
        etx, ety = int(w * etxr), int(h * etyr)
        height = ety - sty
        width = etx - stx
        mask = cv2.resize(mask, (width, height))
        uv[sty:ety, stx:etx] = mask
        return uv

    def render_mask(self, face_image, mask_image, params, auto_blend=True,
positions=[0.1, 0.33, 0.9, 0.7]):
```

```
        uv_mask_image = self.generate_mask_uv(mask_image, positions)
        h, w, c = face_image.shape
        image_vertices = self.params_to_vertices(params, h, w)
        output = (1 - mesh.render.render_texture(image_vertices,
self.bfm.full_triangles, uv_mask_image, self.texcoord,
                                                 self.bfm.full_triangles,
h, w)) * 255
        output = output.astype(np.uint8)
        if auto_blend:
            mask_bd = (output == 255).astype(np.uint8)
            final = face_image * mask_bd + (1 - mask_bd) * output
            return final
        return output


if __name__ == "__main__":
    tool = MaskRenderer('assets_mask')
    # folder of the unmasked dataset
    folder = "F:/thesis/datasets/cropped_aligned_TUFTS_dataset"
    # folder to save the masked images
    folder_save = "F:/thesis/datasets/TUFTS"
    # the mask image you want to apply
    mask_image = cv2.imread("masks/black-mask.png")
    for i in os.listdir(folder):
        fold = os.path.join(folder, i)
        for filename in os.listdir(fold):
            image = cv2.imread(os.path.join(fold, filename))
            fold_s = os.path.join(folder_save, i)
            if image is not None:
                params = tool.build_params(image)
                mask_out = tool.render_mask(image, mask_image, params)
                # make new folders to sort each individual in diff folders
                if not os.path.exists(fold_s):
                    os.makedirs(fold_s)
                cv2.imwrite(os.path.join(fold_s, filename), mask_out)
```

A2 – After masking the faces face detection, extraction and alignment is done. The following code is used for that. Code consists of both RetinaFace and MTCNN incase one fails to detect faces as they are not 100% accurate

MTCNN can be install by using pip and RetinaFace model can be found at

https://github.com/deepinsight/insightface/tree/master/detection/RetinaFace

```
# This program uses MTCNN and RetinaFace to extract faces from the masked
face databases
# If RetinaFace fails it uses MTCNN

import insightface
from matplotlib import pyplot
from mtcnn.mtcnn import MTCNN
import os
import numpy as np
import cv2


# face detection and extraction
def extract_face(filename, folder_save, model, detector):
    # load image from file
    pixels = pyplot.imread(filename)
    # create the detector, using default weights
    img = cv2.imread(filename)
    model.prepare(ctx_id=-1, nms=0.4)
    bbox, landmark = model.detect(pixels, threshold=0.5, scale=1.0)

    if not bbox.any():
        # detect faces in the image
        results = detector.detect_faces(pixels)
        if not results:
            print(filename)
            return None
        else:  # MTCNN is RetinaFace does not fine face
            # extract the bounding box from the first face
            x1, y1, width, height = results[0]['box']
            x2, y2 = x1 + width, y1 + height
            # sometimes these are negative so make them zero
            if y1 < 0:
                y1 = 0
            if y2 < 0:
                y2 = 0
            if x1 < 0:
                x1 = 0
            if x2 < 0:
                x2 = 0
            left_eye_center = results[0]['keypoints']['left_eye']
            right_eye_center = results[0]['keypoints']['right_eye']
            # align face before cropping
            rotated = align_face(img, left_eye_center, right_eye_center)
            rotated = rotated[y1:y2, x1:x2]
            rotated = cv2.resize(rotated, (224, 224))
            # cv2.imshow("image", rotated)
            # cv2.waitKey(0)
            cv2.imwrite(folder_save, rotated)
```

```
            return rotated
    else:  # RetinaFace
        x1, y1, x2, y2 = bbox[0][0:4]
        print(x1, y1, x2, y2)
        left_eye_x, left_eye_y = landmark[0][0]
        right_eye_x, right_eye_y = landmark[0][1]
        left_eye_center = (left_eye_x, left_eye_y)
        right_eye_center = (right_eye_x, right_eye_y)
        # align face before cropping
        rotated = align_face(img, left_eye_center, right_eye_center)
        # sometimes these are negative so make them zero
        if y1 < 0:
            y1 = 0
        if y2 < 0:
            y2 = 0
        if x1 < 0:
            x1 = 0
        if x2 < 0:
            x2 = 0
        rotated = rotated[int(y1):int(y2), int(x1):int(x2)]
        rotated = cv2.resize(rotated, (224, 224))
        # cv2.imshow("image", rotated)
        # cv2.waitKey(0)
        cv2.imwrite(folder_save, rotated)
        return rotated


# Face alignment
def align_face(img, left_eye_center, right_eye_center):
    delta_x = right_eye_center[0] - left_eye_center[0]
    delta_y = right_eye_center[1] - left_eye_center[1]
    angle = np.arctan(delta_y / delta_x)
    angle = (angle * 180) / np.pi

    h, w = img.shape[:2]
    center = (w // 2, h // 2)
    M = cv2.getRotationMatrix2D(center, (angle), 1.0)
    rotated = cv2.warpAffine(img, M, (w, h))
    # cv2.imshow("image", rotated)
    # cv2.waitKey(0)
    return rotated


# change the folder values for different datasets
folder = "F:/thesis/datasets/TUFTS"
folder_save = "F:/thesis/datasets/TUFTS"
model = insightface.model_zoo.get_model('retinaface_r50_v1')
detector = MTCNN()
for i in os.listdir(folder):
    fold = os.path.join(folder, i)
    folder_s = os.path.join(folder_save, i)
    for filename in os.listdir(fold):
        image = os.path.join(fold, filename)
        save = os.path.join(folder_s, filename)
        if image is not None:
            pixels = extract_face(image, save, model, detector)
```

A3 – An example of the code for patching the aligned faces, change the folder link to use with any datasets

```
# this program patches extracted faces and saves them in
# folders for easy training of deep learning

import cv2
from PIL import Image
import os
import numpy as np


# function to patch the face images
def patch_faces(image, filename, i):
    image = cv2.imread(image)
    # folders of the patched dataset
    folder_save_top = "F:/thesis/face_recog/TUFTS all/top"
    folder_save_topr = "F:/thesis/face_recog/TUFTS all/top_right"
    folder_save_topl = "F:/thesis/face_recog/TUFTS all/top_left"

    height, width, channels = image.shape
    crop_bot = image[int(height / 2):height, 0:width]
    crop_top = image[0:int(height / 2) + 10, 0:width]

    crop_top = Image.fromarray(crop_top)
    crop_top = crop_top.resize((160, 160))
    save_patches(crop_top, filename, i, folder_save_top)

    top_left = image[0:int(height / 2) + 10, 0:int(width / 2)]
    top_left = Image.fromarray(top_left)
    top_left = top_left.resize((160, 160))
    save_patches(top_left, filename, i, folder_save_topl)

    top_right = image[0:int(height / 2) + 10, int(width / 2):width]
    top_right = Image.fromarray(top_right)
    top_right = top_right.resize((160, 160))
    save_patches(top_right, filename, i, folder_save_topr)

# function to save the patches in different folders
def save_patches(image, filename, i, folder_save):
    folder_save = os.path.join(folder_save, i)
    save = os.path.join(folder_save, filename)
    if not os.path.exists(folder_save):
        os.makedirs(folder_save)
    image = np.array(image)
    cv2.imwrite(save, image)


# folder of the unpatched face dataset
folder = "F:/thesis/datasets/TUFTS"
for i in os.listdir(folder):
    fold = os.path.join(folder, i)
    for filename in os.listdir(fold):
        image = os.path.join(fold, filename)
        patch_faces(image, filename, i)
```

A4 – The following is the code to read the datasets from folders and saving it in a ".npz" format to load and use later while feature extraction. Do this for all the datasets used for testing. Train folder is the reference folder and validation folder is the probe folder.

```
from os import listdir
from os.path import isdir
from PIL import Image
from numpy import savez_compressed
from numpy import asarray


def resize_extracted_faces(filename, required_size=(160, 160)):
    image = Image.open(filename)
    image = image.convert('RGB')
    face = asarray(image)
    image = Image.fromarray(face)
    image = image.resize(required_size)
    face_array = asarray(image)
    return face_array


def load_faces(directory):
    faces = list()
    for filename in listdir(directory):
        path = directory + filename
        face = resize_extracted_faces(path)
        faces.append(face)
    return faces


def load_dataset(directory):
    X, y = list(), list()
    for subdir in listdir(directory):
        path = directory + subdir + '/'
        if not isdir(path):
            continue
        faces = load_faces(path)
        labels = [subdir for _ in range(len(faces))]
        print('loaded %d examples for class: %s' % (len(faces), subdir))
        X.extend(faces)
        y.extend(labels)
    return asarray(X), asarray(y)

# change the folder links for different datasets
trainX, trainy = load_dataset("F:/thesis/face_recog/FEI
final/masked_full/train/")
print(trainX.shape, trainy.shape)
valX, valy = load_dataset("F:/thesis/face_recog/FEI
final/masked_full/validation/")
print(valX.shape, valy.shape)
savez_compressed("FEI.npz", trainX, trainy, valX, valy)
```

A5- The code below is to use transfer learning and fine tuning on pre-trained model FaceNet.

The model and model weights can be downloaded from https://github.com/nyoki-mtl/keras-facenet

Change the folder links for the dataset and saving the graphs. The model is saved in the same folder as the python file.

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.layers.normalization import BatchNormalization
from keras.models import load_model
from keras.models import Model
from keras.layers import Dense
from keras.layers import LeakyReLU
import numpy as np
from tensorflow import keras
import tensorflow as tf
from glob import glob
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        # Currently, memory growth needs to be the same across GPUs
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        logical_gpus = tf.config.experimental.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical
GPUs")
    except RuntimeError as e:
        # Memory growth must be set before GPUs have been initialized
        print(e)

train_path = 'F:/thesis/face_recog/more_val/top_left/train'
valid_path = 'F:/thesis/face_recog/more_val/top_left/validation'
folders = glob('F:/thesis/face_recog/more_val/top_left/train/*')
model = load_model('facenet_keras.h5')
# model.summary()
print('Loaded Model')

model.load_weights('facenet_keras_weights.h5')

for layer in model.layers:
    layer.trainable = False

inputs = keras.Input(shape=(160, 160, 3))
x = inputs
norm_layer = keras.layers.experimental.preprocessing.Normalization()
mean = np.array([127.5] * 3)
var = mean ** 2
# Scale inputs to [-1, +1]
x = norm_layer(x)
norm_layer.set_weights([mean, var])

# removing last layers and adding new layers
model = Model(inputs=model.inputs, outputs=model.layers[-3].output)
```

```
x = model(x, training=False)
x = Dense(1024)(x)
x= LeakyReLU(alpha=0.03)(x)
x = Dense(512)(x)
x= LeakyReLU(alpha=0.03)(x)
x = Dense(128, activation='relu')(x)
x= BatchNormalization()(x)
outputs = Dense(len(folders), activation='softmax')(x)
model = keras.Model(inputs, outputs)
model.summary()
model.compile(optimizer='Adam', loss='categorical_crossentropy',
metrics=['accuracy'])


datagen = ImageDataGenerator()
training_set = datagen.flow_from_directory(train_path,
                                           target_size=(224, 224),
                                           batch_size=32,
                                           class_mode='categorical')

test_set = datagen.flow_from_directory(valid_path,
                                       target_size=(224, 224),
                                       batch_size=32,
                                       class_mode='categorical')

checkpoint = ModelCheckpoint("models/final_top_left_2.h5",
                             monitor="val_loss",
                             mode="min",
                             save_best_only= True,
                             verbose=1)

earlystop = EarlyStopping(monitor='val_loss',
                          min_delta=0,
                          patience=3,
                          verbose=1,
                          restore_best_weights= True)
callbacks = [earlystop, checkpoint]

r=model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=20,
  callbacks= callbacks,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)

# loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
fig1 = plt.gcf()
plt.show()
fig1.savefig('F:/thesis/face_recog/more_val/top_left/LossVal_loss')

# accuracies
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
fig1 = plt.gcf()
plt.show()
```

```
fig1.savefig('F:/thesis/face_recog/more_val/top_left/AccVal_acc')


# fine tuning with early stopping
model.trainable = True
model.compile(keras.optimizers.Adam(1e-5), loss='categorical_crossentropy',
metrics=['accuracy'])

r=model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=20,
  callbacks= callbacks,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)

# loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
fig1 = plt.gcf()
plt.show()
fig1.savefig('F:/thesis/face_recog/more_val/top_left/LossVal_loss_fine_tune
')

# accuracies
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
fig1 = plt.gcf()
plt.show()
fig1.savefig('F:/thesis/face_recog/more_val/top_left/AccVal_acc_fine_tune')
```

A6 – This code is of feature extraction from the model created by the previous code

```
from numpy import load
from numpy import expand_dims
from numpy import asarray
from numpy import savez_compressed
from keras.models import load_model
import tensorflow as tf
from keras.models import Model

gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        # Currently, memory growth needs to be the same across GPUs
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        logical_gpus = tf.config.experimental.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical
GPUs")
    except RuntimeError as e:
        # Memory growth must be set before GPUs have been initialized
        print(e)

def get_embedding(model, face_pixels):
    samples = expand_dims(face_pixels, axis=0)
    yhat = model.predict(samples)
    return yhat[0]


# load the face dataset
data = load('npz and svm/TUFTS_top_random.npz')
trainX, trainy, valX, valy = data['arr_0'], data['arr_1'], data['arr_2'],
data['arr_3']
print('Loaded: ', trainX.shape, trainy.shape, valX.shape, valy.shape)
model = load_model('models/final_top_2.h5')

model = Model(inputs=model.inputs, outputs=model.layers[-2].output)

model.summary()
print('Loaded Model')

# convert each face in the train set to an embedding
newTrainX = list()
for face_pixels in trainX:
    embedding = get_embedding(model, face_pixels)
    newTrainX.append(embedding)
newTrainX = asarray(newTrainX)
print(newTrainX.shape)

# convert each face in the test set to an embedding
newValX = list()
for face_pixels in valX:
    embedding = get_embedding(model, face_pixels)
    newValX.append(embedding)
newValX = asarray(newValX)
print(newValX.shape)

# save arrays to one file in compressed format
savez_compressed('final/TUFTS_top_random_embeddings.npz', newTrainX,
trainy, newValX, valy)
```

A7 – The following Code is of using SVM after the feature extraction for accuracy calculation

```
from numpy import load
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import Normalizer
from sklearn.svm import SVC

# load dataset
data = load('final/TUFTS_top_right_embeddings2.npz')
trainX, trainy, valX, valy = data['arr_0'], data['arr_1'], data['arr_2'],
data['arr_3']
print('Dataset: train=%d, validation=%d' % (trainX.shape[0],
valX.shape[0]))
# normalize input vectors
in_encoder = Normalizer(norm='l2')
trainX = in_encoder.transform(trainX)
valX = in_encoder.transform(valX)
# label encode targets
out_encoder = LabelEncoder()
out_encoder.fit(trainy)
trainy = out_encoder.transform(trainy)
valy = out_encoder.transform(valy)
# fit model
model = SVC(kernel='linear', probability=True)
model.fit(trainX, trainy)

yhat_train = model.predict(trainX)
yhat_val = model.predict(valX)
# score
score_train = accuracy_score(trainy, yhat_train)
score_val = accuracy_score(valy, yhat_val)
# summarize
print('Accuracy: train=%.3f, val=%.3f' % (score_train*100, score_val*100))
```

A8 – The following code is using Squared Euclidean Distance for accuracy calculation

```
from numpy import load
import numpy as np

data = load('final/TUFTS_top_right_embeddings2.npz')
trainX, trainy,valX, valy = data['arr_0'], data['arr_1'], data['arr_2'],
data['arr_3']


Correct=0
for x in range(valX.shape[0]):
    distances = list()
    for emb in trainX:
        distance = np.sum(np.square(emb-valX[x]))
        # if you want Euclidean Distance
        # distance = np.sqrt(np.sum(np.square(emb - valX[x])))
        distances.append(distance)

    min_dist = min(distances)

    max_dist = max(distances)
```

```
    distances = (distances-min_dist)/(max_dist-min_dist)
    distances = distances.tolist()
    min_dist = min(distances)

    min_index = distances.index(min_dist)
    if trainy[min_index] == valy[x]:
        Correct=Correct+1

total=valX.shape
Accuracy = (Correct/total[0])*100
print("Accuracy : " , Accuracy)
```

A9 – Saving the distances into two text files one for genuine scores and one for imposter score for FNMR vs FMR and ROC curves to calculate EER and accuracy before fusion

```
from numpy import load
import numpy as np
import os
from sklearn.preprocessing import normalize

# load face embeddings
data = load('final/TUFTS_top_embeddings.npz')
trainX, trainy, valX, valy = data['arr_0'], data['arr_1'], data['arr_2'],
data['arr_3']

same_sub = list()
diff_sub = list()

for x in range(valX.shape[0]):
    for y in range(trainX.shape[0]):
        distance = np.sum(np.square(trainX[y] - valX[x]))
        # if you want Euclidean distance
        # distance = np.sqrt(np.sum(np.square(trainX[y] - valX[x])))
        if trainy[y] == valy[x]:
            same_sub.append(distance)
        else:
            diff_sub.append(distance)


same_sub = np.array(same_sub)
diff_sub = np.array(diff_sub)

conc=same_sub.shape[0]

# normalizing the distances from 0 to 1
all_distances=np.concatenate((same_sub,diff_sub))
all_distances=all_distances.reshape(-1,1)
all_distances = normalize(all_distances, norm='l2', axis=0)
all_distances=all_distances.flatten()
same_sub=all_distances[0:conc]
diff_sub=all_distances[conc:]

np.savetxt('score_files_before_fusion/same_subject_scores_top_TUFTS.txt',
same_sub, fmt='%.5f', newline=os.linesep)
np.savetxt('score_files_before_fusion/different_subject_scores_top_TUFTS.tx
t', diff_sub, fmt='%.5f', newline=os.linesep)
```

A10 – This code is for feature level fusion

After fusion, saving the distances into two text files one for genuine scores and one for imposter score for FNMR vs FMR and ROC curves to calculate accuracy and EER after fusion

```
from numpy import load
import numpy as np
import os
from sklearn.preprocessing import MinMaxScaler

# load face embeddings
data_top = load('final/FEI_top_embeddings2.npz')
data_top_right = load('final/FEI_top_right_embeddings2.npz')
data_top_left = load('final/FEI_top_left_embeddings2.npz')

train_topX, train_topy, val_topX, val_topy = data_top['arr_0'],
data_top['arr_1'], data_top['arr_2'], data_top['arr_3']
train_top_rightX, train_top_righty, val_top_rightX, val_top_righty =
data_top_right['arr_0'], data_top_right['arr_1'], \

data_top_right['arr_2'], data_top_right['arr_3']
train_top_leftX, train_top_lefty, val_top_leftX, val_top_lefty =
data_top_left['arr_0'], data_top_left['arr_1'], \

data_top_left['arr_2'], data_top_left['arr_3']

transformer = MinMaxScaler()
rows, cols = train_topX.shape
cols = cols * 3  # 3 feature vector of same size concatenated
fused_features_train = np.zeros((rows,cols))

# fusing train set
for x in range(train_topX.shape[0]):
    # normalize before fusion
    top = train_topX[x].reshape(-1, 1)  # top feature
    top = transformer.fit_transform(top)  # min-max
    top = top.flatten()

    top_right = train_top_rightX[x].reshape(-1, 1)  # top_right
    top_right = transformer.fit_transform(top_right)  # min-max
    top_right = top_right.flatten()

    top_left = train_top_leftX[x].reshape(-1, 1)  # top_left
    top_left = transformer.fit_transform(top_left)  # min-max
    top_left = top_left.flatten()

    fused = np.concatenate((top, top_left, top_right))
    fused_features_train[x] = fused

rows, cols = val_topX.shape
cols = cols * 3  # 3 feature vector of same size concatenated
fused_features_val = np.zeros((rows,cols))
# fusing val set
for x in range(val_topX.shape[0]):
    # normalize before fusion
    top = val_topX[x].reshape(-1, 1)  # top feature
    top = transformer.fit_transform(top)  # min-max
    top = top.flatten()
```

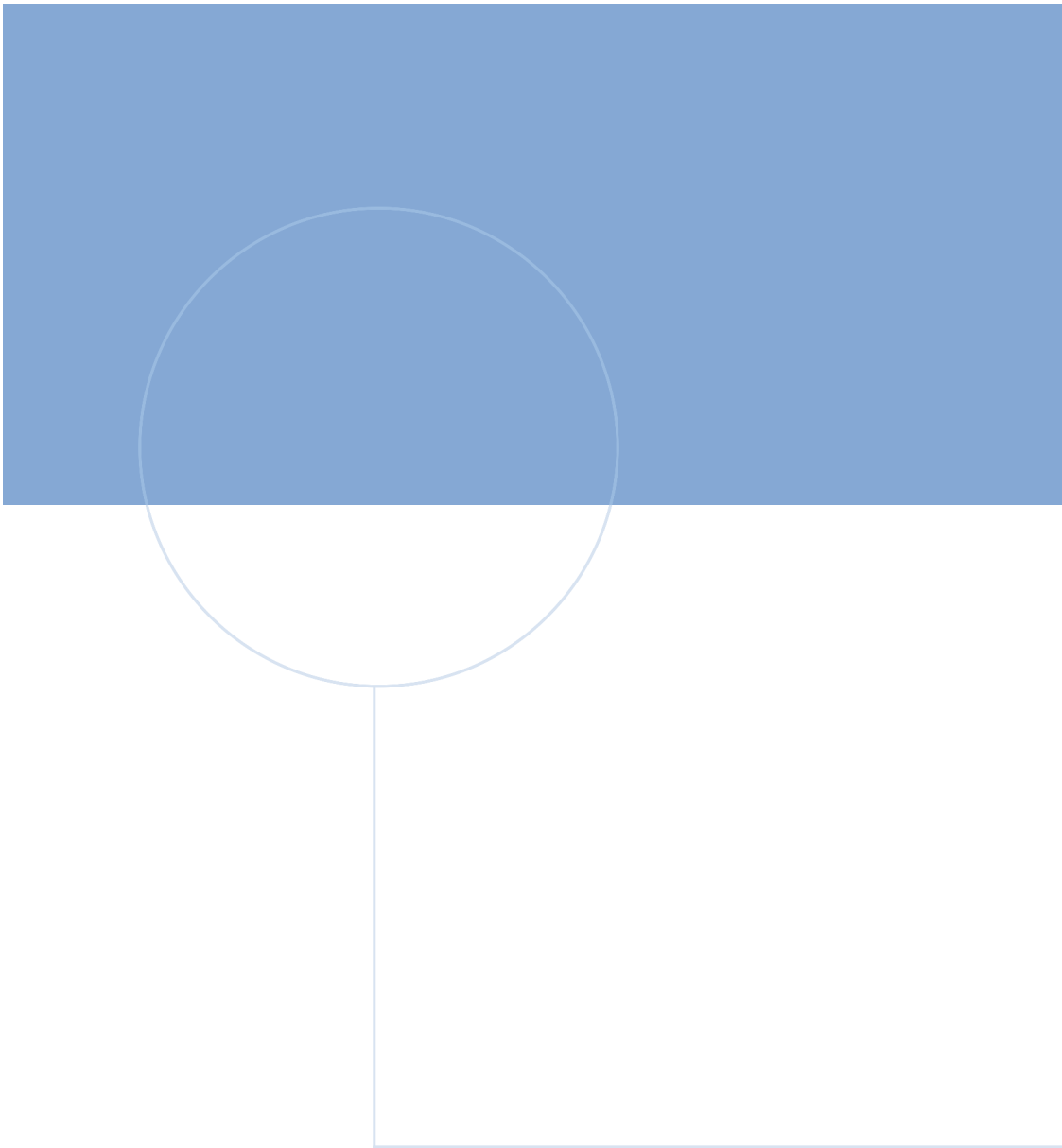
```
    top_right = val_top_rightX[x].reshape(-1, 1)  # top_right
    top_right = transformer.fit_transform(top_right)  # min-max
    top_right = top_right.flatten()

    top_left = val_top_leftX[x].reshape(-1, 1)  # top_left
    top_left = transformer.fit_transform(top_left)  # min-max
    top_left = top_left.flatten()

    fused = np.concatenate((top, top_left, top_right))
    fused_features_val[x] = fused

same_sub = list()
diff_sub = list()

for x in range(fused_features_val.shape[0]):
    for y in range(fused_features_train.shape[0]):
        distance = np.sum(np.square(fused_features_train[y] -
fused_features_val[x]))
        if train_topy[y] == val_topy[x]:
            same_sub.append(distance)
        else:
            diff_sub.append(distance)

same_sub = np.array(same_sub)
diff_sub = np.array(diff_sub)

np.savetxt('score_files_after_fusion/feature_fused_same_FEI.txt', same_sub,
fmt='%.5f',
           newline=os.linesep)
np.savetxt('score_files_after_fusion/feature_fused_different_FEI.txt',
diff_sub, fmt='%.5f',
           newline=os.linesep)
```

A11- Score level fusion code

After fusion, saving the distances into two text files one for genuine scores and one for imposter score for FNMR vs FMR and ROC curves to calculate accuracy and EER after fusion

```
from numpy import load
import numpy as np
import os
from sklearn.preprocessing import MinMaxScaler

# load face embeddings
data_top = load('final/FEI_top_embeddings2.npz')
data_top_right = load('final/FEI_top_right_embeddings2.npz')
data_top_left = load('final/FEI_top_left_embeddings2.npz')

train_topX, train_topy, val_topX, val_topy = data_top['arr_0'],
data_top['arr_1'], data_top['arr_2'], data_top['arr_3']
train_top_rightX, train_top_righty, val_top_rightX, val_top_righty =
data_top_right['arr_0'], data_top_right['arr_1'], \

data_top_right['arr_2'], data_top_right['arr_3']
train_top_leftX, train_top_lefty, val_top_leftX, val_top_lefty =
data_top_left['arr_0'], data_top_left['arr_1'], \

data_top_left['arr_2'], data_top_left['arr_3']
print('Loaded: ', train_topX.shape,
train_top_leftX.shape,train_top_rightX.shape)

transformer = MinMaxScaler()


def calculate_dist(trainX, valX, trainy, valy):
    same_sub = list()
    diff_sub = list()

    for x in range(valX.shape[0]):
        for y in range(trainX.shape[0]):
            distance = np.sum(np.square(trainX[y] - valX[x]))
            if trainy[y] == valy[x]:
                same_sub.append(distance)
            else:
                diff_sub.append(distance)

    same_sub = np.array(same_sub)
    diff_sub = np.array(diff_sub)
    conc = same_sub.shape[0]
    all_distances = np.concatenate((same_sub, diff_sub))
    all_distances = all_distances.reshape(-1, 1)
    transformer = MinMaxScaler()
    all_distances = transformer.fit_transform(all_distances)
    all_distances = all_distances.flatten()
    same_sub = all_distances[0:conc]
    diff_sub = all_distances[conc:]
    return same_sub, diff_sub

# calculate scores of all the patches
same_sub_top, diff_sub_top = calculate_dist(train_topX, val_topX,
train_topy, val_topy)
same_sub_top_left, diff_sub_top_left = calculate_dist(train_top_leftX,
```

```
val_top_leftX, train_top_lefty, val_top_lefty)
same_sub_top_right, diff_sub_top_right = calculate_dist(train_top_rightX,
val_top_rightX, train_top_righty,
                                                        val_top_righty)
# score level fusion
w1, w2, w3 = 0.8, 0.1, 0.1
same_sub= (w1*same_sub_top)+(w2*same_sub_top_right)+(w3*same_sub_top_left)
diff_sub= (w1*diff_sub_top)+(w2*diff_sub_top_right)+(w3*diff_sub_top_left)

np.savetxt('score_files_after_fusion/score_fused_same_FEI.txt', same_sub,
fmt='%.5f',
           newline=os.linesep)
np.savetxt('score_files_after_fusion/score_fused_different_FEI.txt',
diff_sub, fmt='%.5f',
           newline=os.linesep)
```



NTNU

Kunnskap for ei betre verd