January 2022

A Face Recognition Method Using Deep Learning To Identify Mask And Unmask Objects

Saroj Mishra

A FACE RECOGNITION METHOD USING DEEP LEARNING TO IDENTIFY MASK AND UNMASK OBJECTS

by

Saroj Mishra

Master of Science in Computer Science, University of North Dakota, 2022

A Thesis

Submitted to the Graduate Faculty

of the

University of North Dakota

in partial fulfillment of the requirements

for the degree of

Master of Science

Grand Forks, North Dakota

August
2022

Name: Saroj Mishra

Degree: Master of Science

This document, submitted in partial fulfillment of the requirements for the degree from the University of North Dakota, has been read by the Faculty Advisory Committee under whom the work has been done and is hereby approved.

DocuSigned by:
Dr. Hassan Reza

Dr. Hassan Reza

DocuSigned by:
Wen-Chen Hu

Dr. Wen-Chen Hu

DocuSigned by:
Eunjin Kim

Dr. Eunjin Kim

Name of Member 4 - delete if not needed

Name of Member 5 - delete if not needed

Name of Member 6 - delete if not needed

This document is being submitted by the appointed advisory committee as having met all the requirements of the School of Graduate Studies at the University of North Dakota and is hereby approved.

DocuSigned by:
Chris Nelson

Chris Nelson
Dean of the School of Graduate Studies

7/26/2022

Date

PERMISSION

Title	A Face Recognition Method Using Deep Learning to Identify Mask and Unmask Objects
Department	School of Electrical Engineering and Computer Science
Degree	Master of Science

Saroj Mishra
07/26/2022

ACKNOWLEDGMENTS

First, I wish to express my sincere appreciation to my advisor, Dr. Hassan Reza, for his continual guidance in my education and research. I have had the privilege of taking multiple classes from him, and it can clearly be seen that he genuinely cares for his students. I have thoroughly enjoyed each time I have had a meeting with Dr. Reza, he is always personable, kind, and willing to help. Also, I would like to thank Dr. Wen-Chen Hu and Dr. Eunjin Kim for all of their help and for agreeing to serve on my committee. I have had the privilege of taking courses from both Dr. Hu and Dr. Kim during my time here, and I thoroughly enjoyed them all. I am so thankful for their guidance and support during my time in the master's program at the University of North Dakota. Similarly, I would also like to thank all the faculty and staff at the School of Electrical Engineering and Computer Science for the help they provided during my graduate career. My mom and dad deserve an extraordinary amount of thanks for all of their continual support and prayers. I would especially like to thank my wonderful wife Rakshya Bista for all of her support and encouragement throughout my graduate study. Finally, I would like to thank my friends and other family members for their love and support while on this academic journey.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

At the present, the use of face masks is growing day by day and it is mandated in most places across the world. People are encouraged to cover their faces when in public areas to avoid the spread of infection which can minimize the transmission of Covid-19 by 65 percent (according to the public health officials). So, it is important to detect people not wearing face masks. Additionally, face recognition has been applied to a wide area for security verification purposes since its performance, accuracy, and reliability [15] are better than any other traditional techniques like fingerprints, passwords, PINs, and so on. In recent years, facial recognition is becoming a challenging task because of various occlusions or masks like the existence of sunglasses, scarves, hats, and the use of make-up or disguise ingredients. So, the face recognition accuracy rate is affected by these types of masks. Moreover, the use of face masks has made conventional facial recognition technology ineffective in many scenarios, such as face authentication, security check, tracking school, and unlocking phones and laptops. As a result, we proposed a solution, Masked Facial Recognition (MFR) which can identify masked and unmasked people so individuals wearing a face mask do not need to take it out to authenticate themselves. We used the Deep Learning model, Inception ResNet V1 to train our model. The CASIA dataset [17] is applied for training images and the LFW (Labeled Faces in the Wild) dataset [18] with artificial marked faces are used for model evaluation purposes. The training and testing masked datasets are created using a Computer Vision-based approach (Dlib). We received an accuracy of around 96 percent for our three different trained models. As a result, the purposed work could be utilized effortlessly for both masked and unmasked face recognition and detection systems that are designed for safety and security verification purposes without any challenges.

Keywords: Face Recognition, Masked Facial Recognition, Verification, Security, Accuracy, CASSIA Dataset, LFW Dataset, Deep Learning, Dlib, Computer Vision

CHAPTER 1

INTRODUCTION

This chapter first introduces the background of the Facial Recognition System (FRS) and gives a description of the general problem. Section 1.2 shows the scope of the work. The scope of the work consists of a general methodology description. Section 1.3 presents the motivation of the research towards the use of Masked Facial Recognition (MFR) using deep learning. Lastly, section 1.4 shows the approaches to solve the problem which is followed by expected results in section 1.5.

1.1 Problem Definition

The use of face masks is growing rapidly with Covid-19. People are required to wear a face mask all the time when they are outside or at large indoor gatherings to minimize the spread of infection. So, it is important to detect those people with face masks for health safety reasons. Face recognition is the process of automatically identifying an individual from captured images or videos [4] and face detection is the process of identifying the face from the captured image or the specified image from the database. It is a significant key area of research today as its applications are becoming more important in various fields like ATM machines, criminal identification, access restriction, video conferencing, issuing drivers' licenses & passports, and monitoring the public areas.

Moreover, the use of face masks has made conventional facial recognition technology ineffective in many scenarios, such as face authentication, security check, tracking school, and office attendance, and unlocking phones and laptops. Furthermore, the different algorithms that succeed on unmasked faces have been unable to generalize such successes on masked faces. One of the advantages associated with detecting an unmasked face is that the deep learning models would use the whole facial features/landmarks to identify someone. However, with a masked face, the nose and mouth are occluded. So, the problem of identifying individuals with just the eyes and sometimes, the forehead is more challenging [1]. Therefore, it is expected that the purposed solution could be utilized to recognize both masked and unmasked faces.

Since the work of Schroff et al. [15] in 2015, the idea of the FaceNet model, a unified embedding for face recognition has been widely used for facial recognition. The researcher presented a FaceNet model, which learns directly from face images to calculate the Euclidean distances, and these distances are directly compared to measure the face matching. Additionally, once these FaceNet embeddings are created, they could easily be implemented for tasks such as face identification, validation, and clustering by calculating the Euclidean distances. So, we applied a similar embeddings technique in our Masked Facial Recognition (MFR) research. Although many works have been presented [26, 1, 13, 16, 25] in the past for masked face recognition, our proposed solution provides better accuracy and can recognize both masked and unmasked faces easily.

1.2 Scope of work

The scope of this work falls under the MFR (Masked Facial Recognition) system. The approach solution could be used for both masked and unmasked facial recognition systems with high accuracy. Especially in the current situation where everyone needs to wear a face mask to minimize the spread of Covid-19, this work helps to identify the public without a face mask and encourages them to wear masks which increases the safety of the public. Additionally, the purposed solution includes a masked facial recognition system so it can recognize the masked and unmasked people. As a result, people in enclosed spaces who need to verify their identity on mobile phones, laptops, or other devices, do not need to take off their face masks as the purposed solution can recognize masked faces easily. Moreover, we verified the robustness of the purposed solution for masked and unmasked facial recognition under various conditions like gender, skin tone, age, types of masks, etc. Therefore, this work could be used for different purposes including security and safety of the people.

1.3 Motivation

In the present day, due to Covid-19, face masks have been mandated in most places across the world. People are encouraged to cover their faces when in public areas to avoid the spread of infection which can reduce the transmission of Covid-19 by 65% (according to the public health officials). Face recognition is widely used to secure any system because it is better than any other traditional techniques like PIN, password, fingerprint, and so on and is most reliable to identify a person efficiently. Additionally, facial recognition has been extensively applied for security verification purposes since its performance, accuracy, and reliability [15] are better than any other traditional techniques like fingerprints, passwords, tokens, and so on. Nowadays, it has become an arduous task due to various occlusions or masks such as scarves, sunglasses, hats, makeup, and other different types of disguise elements and they are causing a significant impact on the accuracy of facial recognition systems (FRS). As a result, we proposed a solution an MFR (Masked Facial Recognition) to solve this issue.

1.4 Approach

The main goal of our research paper is to perform real-time Masked Facial Recognition (MFR). The work is divided into three main parts to achieve the goal. The first part is data collection and preparation. We take the CASIA dataset [17] for training face images. The obtained images were not ready to use for training, so we performed various cleaning, alignment, and removal operation to make them ready for model training purposes. The CASIA dataset does not include masked faces, so we used the augmented method to generate masked faces for our dataset. Furthermore, the second part involved training the face recognition model that would be used for MFR. The training is done using the deep learning (Inception ResNet V1) model. We applied different hyperparameter functions for training and the model is evaluated using the LFW (Labeled Faces in the Wild) dataset [18] with artificial masked face images. Similarly, the accuracy and loss functions were calculated in every epoch to validate the model. We evaluated the three different trained models with five, ten, and fifteen training images per class. Lastly, the real-time MFR is carried out using our trained model. All these three steps are explained in more detail in the methodology section.

1.5 Expected outcomes

The purpose of our research is to collect data and complete the data preparation, train the model for MFR and its evaluation, and perform real-time masked and unmasked facial recognition. It's expected to receive the input image frame through a laptop camera. Moreover, it is anticipated that the purposed solution would be useful for masked and unmasked facial recognition under various conditions like gender, skin tone, age, types of masks, etc. As a result, this work could be utilized for different purposes including security and safety of the people.

1.6 Thesis Structure

The organization of the paper is as follows. Chapter II presents the theoretical background of the research. It introduces the Face Recognition System (FRS), Deep Learning (DP), Convolutions Neural Network (CNN), and their block diagrams with some details. We start chapter III by introducing the works that have been done in facial recognition, and masked facial recognition. Chapter IV outlines the different approaches we have applied to our work. There are data collection and preparation, model training, model evaluation, and real-time Masked Facial Recognition (MFR). Similarly, Chapter V describes the results of our work, experiment setup, performance evaluation, limitation and advantages of the approach solution, performance Metrix, and comparisons of our solution against different methods. Lastly, chapter VI concludes with the conclusion along with some suggestions for future research.

CHAPTER 2

BACKGROUND

This chapter introduces facial recognition technology, workflow, and methods. Also, introduce deep learning (DP), convolutions neural network (CNN), and their concepts. Additionally, the importance of DP to build the recognition system and the development processes of face recognition and CNN are discussed briefly.

2.1 Facial Recognition System

Facial recognition is the process of automatically identifying an individual from captured images or videos [4]. It is a significant key area of research today. Its applications are becoming more important in various fields like ATM machines, criminal identification, access restriction, video conferencing, issuing drivers' licenses and passports, and monitoring public areas. The imaging conditions, feature occlusion, inter-person similarity, and variance of faces are making the task of face recognition more challenging. Face recognition algorithms deal with a vast number of images, which leads to millions of operations, so it needs to have a specialized model for real-time implementation [3]. Moreover, various algorithms have been proposed for face recognition in the last few decades, with varying degrees of success. These algorithms analyze images and extract information such as shape, size, and location of facial features. So, the algorithms with the highest accuracy typically require intensive computation [4].

2.2 Workflow of a Facial Recognition System

The facial recognition process usually has five interrelated steps shown in figure 2.2 [30]. The first step is capturing the input image. The image is captured through a camera source, and it passes to the Face detection model. In this step, the face of the person is detected from the whole captured image. Additionally, the third step is Features extraction, which extracts the specific and unique features from the detected face to match them with the corresponding images in the database. So, in this step, the face embeddings are generated. Similarly, face matching is done by calculating the Euclidian distance between the input image's embedding with the embeddings of database images. If the distance is less than the threshold value, then the identification of the person is accomplished. As a result, face recognition is completed by computing the distances between one input image with N number of database images.

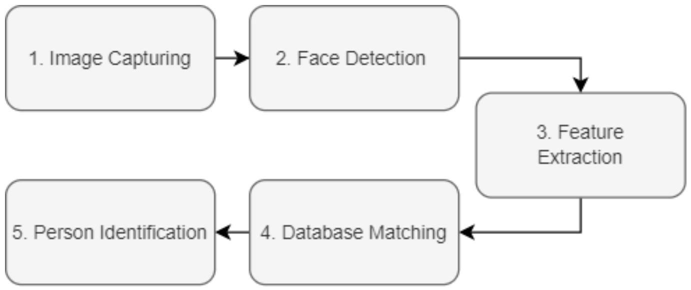


Figure 2.2 Workflow of facial recognition process

2.3 Face Recognition Methods

Many face recognition algorithms have been developed in the past. Some of the commonly used methods are Knowledge-based methods, Template matching, and appearance-based methods. All these methods are briefly explained below.

2.3.1 Knowledge-Based Methods

This method is also known as the rule-based method since the efforts are made to capture the knowledge of faces and then subsequently, they are translated into a set of rules. For example, there are facial features that are symmetrically located and areas on the face that differ in intensity. Deriving the appropriate set of rules is the major problem with knowledge-based methods, which should be neither too general nor too detailed. Additionally, a potential solution for overcoming these problems is to create a hierarchical knowledge-based approach. This method is not able to work efficiently for complex images where the face invariant features are not visible, this method fails to work efficiently. As a result, a robust method should be employed to overcome this difficulty.

2.3.2 Templating Matching

Templating Matching is the process of identifying faces by considering only specific regions represented in templates. The pixels of the input image are compared against a template image using a metric measure such as Euclidean distance. In the first step, four features such as eyes, nose, mouth, and whole face are selected from a template and applied to all the available faces. The area of the input image is compared to the pictures in the database for each template. As a result, the face matching scores are calculated, and the identification decision is made based on it. Moreover, high accuracy rates of around 90% have been reported using this approach. The main advantage of this method is that the implementation is simple, however, it is inappropriate for variations in pose and illumination.

2.3.3 Appear-based methods

This approach is similar to template matching in which templates are taken from the set of examples in the images. Mainly, this method depends on techniques from statistics and machine learning to find the specific characteristics of face images. There are different algorithms for appear-based methods which are explained briefly below.

Eigenface-based methods PCA algorithms: The most famous and widely accepted approach for facial recognition is known as the Karhunen–Loeve method. It is the most thoroughly studied method for face recognition, with its main usability being a reduction in the dimensionality of the image. This method was first applied for face recognition and then subsequently used for facial reconstruction. The main advantage of this approach is that it minimizes the data by the 1000th time. Furthermore, this process is very quick, as it is utilized only for training the sample. While this approach demands a full-frontal image of the subject’s face, and in real-time situations, this rarely happens.

Linear discriminant analysis: The Linear discriminant analysis (LDA) also known as Fisher’s discriminant analysis, is the dimensionality reduction technique. Among appearance-based approaches, LDA is utilized for feature selection. Among various appearance-based methods, LDA is applied for feature selection. This method overcomes the limitations of the PCA method. Among all the appearance-based approaches, LDA is the most Similar to PCA, it is based on Euclidean distance. Moreover, it is a supervised method, and unlike PCA, it uses label information for enhancing separability between different classes. Moreover, it also aims to reduce the variation within the class. In many LDA-dependent face-recognition techniques, initially, PCA is used for dimensionality reduction, and then LDA is used to maximize the discrimination power of feature selection

Neural networks: The solution to the issues faced in linear methods was delivered by several nonlinear approaches such as neural networks. Normally, a net is considered with a neuron in every pixel. The feature extraction step is more efficient than linear methods because of the non-linearity of the network. Moreover, using this approach, an accuracy rate of 96.2% was noted. The training time is higher compared to the classification time. Neural networks have been combined with various models, such as Gabor filters and Hidden Markov Models (HMM). A semi-supervised method was used for identifying the human face. The researchers used unsupervised methods for feature extraction and supervised techniques for finding those features which can lower classification error. Additionally, they utilized feed-forward neural networks for classification. The probabilistic decision-based neural network was modeled for three distinct aspects which are face detection, eye localization, and face recognition. The main advantage of the neural network is its capability to capture the tough class of face patterns. While the number of classes increases, it becomes difficult to implement neural networks. Also, this process is not suitable for a single model.

Deep learning and artificial intelligence: In this technology era, the techniques of machine learning, deep learning, and artificial intelligence have subsequently influenced and impacted the broad area of daily services and logical functions. Nowadays facial recognition has been verified its indispensable impact in different applications. In a previous study, an algorithm to choose the experience of customers at a restaurant with no staff was developed. Based on expressions, the food and environment were rated. That system uses the pre-trained CNN (convolutional neural network) model. Similarly, using CNN models and applying deep-learning techniques, emotion recognition was achieved from audio-visual emotional big data.

Support vector machine: The support vector machines (SVM) are the linear classifiers that extend the margin between the decision hyperplane and the training set. This is accomplished by finding a hyperplane that divides and maximizes the distance from any of the classes to the hyperplane, where a set of points belong to two distinct classes. Therefore, an optimal hyperplane should work by minimizing the classification error of the unseen test patterns. Additionally, first, the feature extraction is achieved using PCA (Principal Component Analysis), and then discrimination between the features is accomplished using SVMs. The main advantage of SVMs as compared to other classical methods is that they can achieve better performance.

2.4 Why Use the Face for Recognition

There are many ways to authenticate a person, while biometric-based techniques have appeared as the most promising option for recognizing individuals in recent years. Other approaches like tokens, keys, passwords, and smart cards require the involvement of individuals in any way and have chances to be misplaced, stolen, forgotten, or forged. However, the biological traits of an individual cannot be misplaced or forgotten. Biometric-based recognition systems include the characteristics of the individual such as the face, palm, retina, voice, finger geometry, and so on. Among all the methods, face recognition offers many advantages, so it is preferred the most. Some of the advantages are mentioned here: however, all these techniques require some voluntary action by the user, and face recognition can be done passively without any explicit effort since face images can be received from a distance by a camera. This is useful for identification, security, and surveillance purposes. Similarly, some methods depend on the hands and fingers which can be affected useless if the epidermis tissue is harmed in some way. Retina identification requires costly equipment, and they are much too sensitive to body motion. Also, Voice recognition is susceptible to background noises in public areas and hearing fluctuations on a phone line or tape recording. On the other hand, facial images can be obtained easily with an affordable camera. Good facial recognition algorithms and appropriate image preprocessing can compensate for noise and slight changes in orientation, scale, and lighting.

2.5 Problems with Face Recognition Algorithms

Since the development of face recognition, it has often had to overcome various challenges. There are several factors that make facial recognition an arduous task. When the face image changes in illumination, expression, and pose, that makes identification problems much harder. Additionally, Age changes the facial texture and shape while occluded images left partial facial features for processing, thus making the overall problem of face recognition much harder. The face recognition performance deteriorates significantly when variations are found in illumination, facial pose, and expression. Furthermore, other factors such as image resolution, orientation, blurring, and time delay also contribute to facing recognition obstacles.

2.6 Deep Learning

Deep learning is the pile of Convolutional Neural Network (CNN) layers and CNN is one of the most effective neural networks that has shown its superiority in a wide range of applications, including image classification, recognition, retrieval, and object detection. In addition, Neural Network (NN) is a sub-field and a key area of machine learning which are biological brain-inspired function approximators and have been successfully applied to various issues such as classification, regression, control, learning (online and offline), and robotics [2]. Furthermore, the neural network is an enormously powerful and robust classification technique that can be used for predicting not only the known data but also the unknown data. It works well for both linear and non-linear various datasets [19].

The NN has been used in multiple areas such as object detection, speech recognition, face recognition, fingerprint recognition, forecasting, and so on [5]. A standard feed-forward neural network is made up of multiple input layers of neurons, some of them hidden, and an output layer of neurons. A neuron is a basic part of a neural network. It processes signals by accepting them as an input and then outputs a signal using a function. Moreover, the NN receives information on the environment as a normal signal from its input layer and then outputs a signal through the output layer of neurons [2]. The general workflow of Neural Networks is shown in figure 2.2. In our research, we applied the CNN layer to train the MFR (Masked Face Recognition) model. It is challenging to obtain all the facial characteristics from a single layer so multiple CNN layers are utilized to extract various patterns of the face images. As a result, deep learning is significantly important to learn all the details of facial attributes.

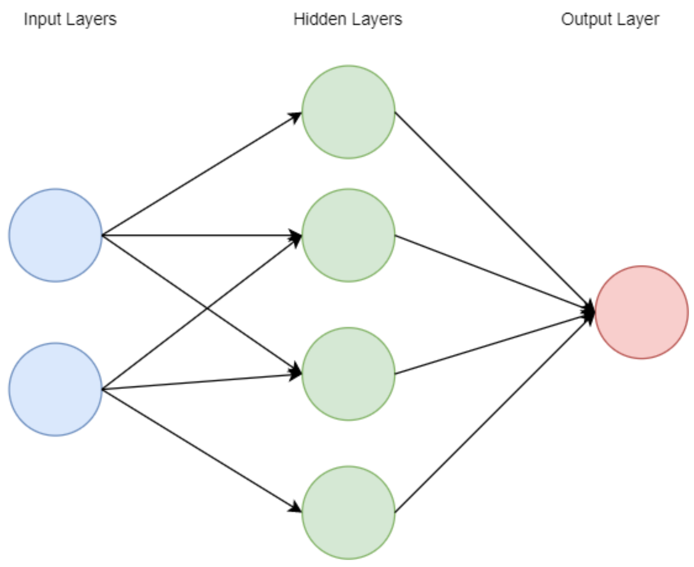


Figure 2.2 The workflow diagram of NN

In figure 2.2, the input layer receives information in the form of a numeric expression and transfers it to the hidden layers, which calculate the weighted sum and weights. The information is displayed as activation values, where each layer has given a number, the higher the number greater the activation. Additionally, this information is then transferred throughout the network. Based on the strength of the connection which are weights, inhibition, and transfer functions, the activation value is transmitted from layer to layer. Individual layers sum up the activation values it collects; then transform the value based on its transfer function [5]. Similarly, the Activation value goes via hidden layers through the network until it makes the output layer. The output layer then reflects the meaning. The neural network could have many inputs, hidden, and output layers. There are several types of neural networks (NN) and some of them are recurrent neural network (RNN), convolutional neural network (CNN), and deep convolutional Network (DCN). The NN is applied in various fields such as computer vision, time series prediction, pattern recognition, robot control, anomaly detection, object detection, and so on.

CHAPTER 3

LITERATURE REVIEW

This chapter discusses the related work and identifies what aspects of previous work will be applied. These studies focus on a few selected research that contributes to the use of facial recognition with and without face masks and how it can be implemented using different approaches. This chapter’s primary goal is to provide a review of the literature and present an overview of the current research that contributes to the use of MFR (Masked Facial Recognition). Moreover, it describes some of the challenges in the development of facial recognition systems, a summary of the related works and the approaches, their limitations and benefits, the research gap, and our contributions to solving the problems with existing approaches.

3.1 Face Recognition Developing Stages

The earliest pioneers of facial acknowledgment were Woody Bledsoe, Helen Chan Wolf, and Charles Bisson. In 1964 and 1965, Bledsoe began working with computers with Wolf and Bisson to identify the human face. Due to the financing of the project coming from an unnamed intelligence agency, much of their work was never made public. However, later it turned out that their initial work applied the manual marking of different facial landmarks on the faces, such as eyes centers, nose, and mouth. These were then statistically turned by a computer to compensate for pose variation. Then the distance between the landmarks was calculated and compared between images to determine identity automatically [7].

These earliest steps into Facial Recognition in a manner consistent with the Bledsoe, Wolf, and Bisson were severely hampered by the technology of the era, but it remains an important first step in proving that Facial Recognition was a practical biometric. Carrying on from Bledsoe's original work, the baton was picked up in the 1970s by Goldstein, Harmon, and Lesk who expanded the work to include 21 specific subjective markers including hair color and lip thickness to automate the recognition. The National Institute of Standards and Technology (NIST) started Face Recognition Vendor Tests (FRVT) in the early 2000s. Building on FERET (Face Recognition Technology), FRVTs (Face Recognition Vendor Tests) were designed to provide independent government evaluations of commercially available facial recognition systems and prototype technologies. These assessments were designed to provide law enforcement agencies and the U.S. government with the information needed to determine the best ways to deploy facial recognition technology.

Back in 2010, Facebook began implementing facial recognition features that helped identify people whose faces may feature in Facebook photos that users update daily. The feature was immediately controversial with the news media, triggering a slew of privacy-related articles. However, Facebook users did not seem to mind. Having no clear adverse effect on the Web site's use or popularity, more than 350 million pictures are uploaded and tagged using face recognition every day. Facial Recognition technology has advanced rapidly from 2010 onwards and September 12, 2017, was another significant breakthrough for the integration of facial recognition into our day-to-day lives. This was the date that Apple launched the iPhone X, which was the first iPhone users could unlock with Face ID – Apple's marketing term for facial recognition. So, in this way, the development of facial recognition took place from past to present.

3.2 Masked Facial Recognition

Masked face recognition refers to techniques in which the system needs to recognize the individual whose face is occluded. Masked facial recognition is one of the most difficult problems because masks cover more than half of the face, and it is difficult to identify someone with just eyes, eyebrows, and forehead areas. Therefore, masked face recognition often requires large datasets, and huge calculations to train the best model which can help to recognize the person accurately.

In 2021 Vu et al. [31] presented Masked face recognition with convolutional neural networks and local binary patterns. The researcher proposed the combination of deep learning and local binary pattern features approach to identify the masked faces by using RetinaFace. A RetinaFace is a face detector, which is a combination of self-supervised and extra-supervised multi-task learning that can deal with various scales of faces. Moreover, they extracted the local binary pattern features from the eye, forehead, and eyebrow areas of masked faces and joined them with features learned from RetinaFace into a unified framework for recognizing masked faces. Additionally, they used the 300-subject dataset collected from their institution named COMASK20. The researcher mentioned that they compared their system with the published Essex dataset also and they received 98% and 87% f1 scores for COMASK20 and Essex datasets, respectively. These showed that their system has shown effectiveness and suitability as compared to other methods like Dlib and InsightFace.

In 2021 Ullah et al. [1] presented a novel DeepMaskNet model for face mask detection and masked facial recognition. Testing people who are not wearing face masks manually in public places is a challenging task. Moreover, using face masks makes traditional face recognition techniques ineffective, typically designed for unveiled faces. Therefore, the researcher introduced a reliable system that can detect people who do not wear face masks and recognize different people while wearing face masks. In this paper, they proposed a novel DeepMasknet framework capable of both face mask detection and masked facial recognition. Moreover, presently there is an absence of a unified and diverse dataset that can be used to evaluate both face mask detection and masked facial recognition. For this purpose, they also developed a largescale and diverse unified mask detection and masked facial recognition (MDMFR) dataset to measure the performance of both the face mask detection and masked facial recognition methods. The proposed work has two main phases. The first phase includes the data collection and dataset preparation, while the second phase presents a novel Deepmasknet model construction for face mask detection and masked facial recognition. They got an accuracy of 100% for face detection and 93.33% for masked facial recognition. Researchers said that their experimental results on multiple datasets including the cross-dataset setting showed the superiority of their DeepMasknet framework over the contemporary models.

In 2020 Mundial et al. [14] presented a paper on facial recognition problems in the covid-19 pandemic. The researchers proposed a methodology that can improve the existing facial recognition technology capabilities with masked faces. They used a supervised learning method to recognize masked faces together with in-depth neural network-based facial features. A dataset of masked faces was collected to train the Support Vector Machine classifier on a state-of-the-art Facial Recognition Feature vector. Their proposed methodology gave recognition accuracy of up to 97% with masked faces. They mentioned that this model performed better than existing devices not trained to handle masked faces.

In 2019 Ejaz et al. [16] presented the implementation of principal component (PCA) analysis on masked and non-masked face recognition. In this paper, a statistical procedure was selected that is applied in the recognition of the non-masked face and applied in the masked facial recognition technique. This method achieved an accuracy of masked face image recognition on average of 72% whereas non-masked face was on average 95%. PCA gave a poor recognition rate for masked face images rather than non-masked faces. It was found that extracting facial features from a masked face is less than a non-masked face because of missing features from masked faces. As a result, the researcher concluded that the PCA Analysis is better for normal face recognition but not for masked face recognition.

In 2020 Anwar et al. [12] presented masked face recognition for secure authentication. With the recent worldwide COVID-19, face masks have become an important part of our lives. People are encouraged to cover their faces when in public areas to avoid the spread of infection which can reduce the transmission of Covid-19. Face recognition system is commonly used for security verification purposes and the use of face masks has made conventional facial recognition technology ineffective in many scenarios, such as face authentication, security check, community visit check-in, tracking school, office attendance, and unlocking phones and laptops. Because of Covid-19, people in closed spaces must wear face masks to verify their identity on their mobile phones or laptops. Many organizations use facial recognition as a means of authentication and have already developed the necessary datasets in-house to be able to deploy such a system. Unfortunately, masked faces make it difficult to be detected and recognized, thereby threatening to make the in-house datasets invalid and making such facial recognition systems inoperable. As a result, the researcher addressed a methodology to use the current facial datasets by augmenting them with tools that enable masked faces to be recognized with low false-positive rates and high overall accuracy, without requiring the user dataset to be recreated by taking new pictures for authentication. They presented an open-source tool, MaskTheFace to mask faces effectively creating a large dataset of masked faces. The dataset generated with this tool is then used towards training an effective facial recognition system with target accuracy for masked faces. They received an increase of around 38% in the true positive rate for the Facenet system. Additionally, the researcher tested the accuracy of the re-trained system on a custom real-world dataset MFR2 and report similar accuracy.

In 2021 Mandal et al. [13] proposed masked face recognition using ResNet-50. Over the last twenty years, there have seen several outbreaks of different coronavirus diseases across the world. These outbreaks often led to respiratory tract diseases and have proved to be fatal sometimes. Currently, we are facing an elusive health problem with the arrival of the COVID-19 disease of the coronavirus family. Airborne transmission is one of the modes of transmission of COVID- 19 and it transfers when humans breathe, speak, sing, cough, or sneeze in droplets released by an infected person. As a result, public health officials have prescribed the use of face masks that can reduce disease transmission by 65% [13]. Facial recognition systems are used for security verification purposes and the use of face masks presents a difficult challenge since these systems were typically trained with human faces without masks but now due to the onset of the Covid-19 pandemic, they are forced to identify faces with masks. Therefore, the researcher studied the same problem by developing a deep learning model capable of accurately identifying face masks. In this paper, the authors trained a ResNet-50-based architecture that performs well at recognizing masked faces. The results of this study could be seamlessly integrated into existing facial recognition programs designed to detect faces for safety verification purposes.

3.3 Unmasked Face Recognition

Facial recognition is the process of automatically identifying an individual from captured images or videos [4]. It is a significant key area of research today. Its applications are becoming more important in various fields like ATM machines, criminal identification, access restriction, video conferencing, issuing drivers’ licenses and passports, and monitoring the public areas. Moreover, the imaging conditions, feature occlusion, inter-person similarity, and variance of faces are making the task of face recognition more challenging. The facial recognition process usually has five interrelated steps: image capturing, face detection, feature extraction, database matching, and person identification.

In 2015 Schroff et al. [15] proposed the FaceNet model, a unified embedding for face recognition and clustering. Despite significant recent progress in face recognition, implementation of face verification and recognition effectively poses serious challenges to current approaches. The researcher presented a FaceNet model, which learns directly from face images to calculate the Euclidean distances, and these distances are directly compared to measure the face matching. Additionally, once these FaceNet embeddings are created, they could easily be implemented for tasks such as face identification, validation, and clustering by calculating the Euclidean distances. Their method applied a deep convolution network trained to optimize the facial embedding, while the previous methods of deep learning used an intermediate layer of a bottleneck. A new triplet mining approach generated the non-matching and matching face patches to train. The main advantage of their approach was greater representation efficiency where they used only 128 bytes/face and obtained state-of-the-art facial recognition performance. In the massively used Labeled Faces in the Wild (LFW) dataset, their system reached the accuracy of 99.63%, a new record high. On the YouTube Faces database, it received 95.12%. Furthermore, their system shortened the error rate by 30% on both datasets as compared to the top published result of other papers [24]. Also, the researcher presented the notion of harmonic embeddings and a harmonic triplet loss, which represented various versions of face embeddings that were compatible with each other and authorized for direct comparison.

In 2015 Simonyan et al. [11] proposed very deep convolutional networks for large-scale image recognition. In this work, they studied the effect of the depth of the convolutional network on its accuracy in a large-scale image recognition environment. Their main contribution was a thorough evaluation of networks of increasing depth using an architecture with exceedingly small (3×3) convolution filters, which showed that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers. These results were the basis of their 2014 ImageNet Challenge submission, where the research team secured first and second place in localization and classification tracks, respectively. They also showed that their representations generalize well to other datasets, where they achieve state-of-the-art results. To promote further research on the use of deep visual representations in computer vision, the researchers made two of the most powerful ConvNet models public.

In 2013 Jindal et al. [28] proposed PCA (Principal Component Analysis) with the Artificial Neural Network (ANN) method which identified features of the face images extracted using the PCA method. PCA is a dimensionality reduction method and keeps most of the variations present in the data set. It captures the variations in the dataset and uses this information to convert the face images. It calculates the functional vectors for various face points and forms a column matrix of these vectors. In this paper, the mathematical function Log-sigmoid was applied for the eigenfaces of the same person, the specific neural network gave the output as one and for the eigenfaces of another person, it gave the output as 0. After that, only the recognized faces were found as output one. Therefore, Neural Network forms an Identity matrix for different face images using the outputs 1s and 0s. The errors in the output layer were sent back to the earlier layers and refresh the weights of these layers which reduced the error. If the minimum distance between the tested eigenface image and the trained input eigenface image is less than the threshold value, then the output of the network is one and the trained eigenface image is chosen from the Identity matrix as an output image and further identified as a resulted face image otherwise the test face image is denied as a non-human or unknown face image. The purposed face recognition system worked with high accuracy and provided better success rates even for noisy face images [5].

In 2015 Yooa et al. [27] introduced a hybrid method of face recognition by using face region information taken from the detected face region. The researcher designed the hybrid approach based on the ASM (Active Shape Model) and PCA (Principal Component Analysis) methods for the image preprocessing part. At this step, they used a Charge Coupled Device camera to obtain facial images using AdaBoost, and then Histogram Equalization was applied to enhance the image quality. In the formation of the state part of the fuzzy rules, the input space was divided with the use of Fuzzy C-Means clustering. In the summary part of the fuzzy rules, the connection weights of the RBF NNs were represented by four types of polynomials such as linear, quadratic, constant, and reduced square. An advised Polynomial-based RBF NNs were implemented for facial recognition and their performance was quantified from the perspective of the performance and recognition rate. This enhancement can be attributed to the fact that the unnecessary parts of the image were removed with the use of the ASM [6].

3.4 Summary of the Related Works

All the previous work that has been done in facial recognition has its own techniques, model, advantages, and disadvantages. Different research was carried out with varying rates of success in the past. In table 3.4, we have provided the summary of existing masked and unmasked systems in terms of their approach, model, accuracy, dataset, advantages, disadvantages, and so on.

Table 3.4 Summary of the related work for masked facial recognition

Research	Model/Method	Dataset/Accuracy	Advantages/Disadvantages
[1]	Presented a novel DeepMaskNet model using CNN	Dataset: MDMFR, Kaggle Accuracy: 93.33%	• This system can detect the face mask and also recognize the masked faces • Verified the model under diverse conditions like age, gender, type of masks, illumination, face angles, etc.
[12]	Proposed a model for masked face recognition using FaceNet	Dataset: VGGFace2, MFR2 Accuracy: 38% higher true positive for FaceNet system	• Provided low-false positive rates and overall high accuracy • Created a large dataset for masked faces
[13]	Masked face recognition using ResNet-50, and applied transfer learning technique	Dataset: RWMFD Accuracy: 89.70% (masked), 47.91% (unmasked)	• Applied different hyperparameters turning to identify the masked faces • Use of imbalanced data significantly reduced the accuracy of masked face recognition
[14]	Used supervised learning approach with in-depth neural network based facial feature for masked face recognition	Dataset: VGGFACE2, LFW Accuracy: up to 97%	• Trained the Support Vector Machine classifier on a state-of-the-art facial recognition feature vector
[15]	Presented the unified embedding concept using FaceNet model for face recognition	Dataset: LFW, YouTube Accuracy: 99.63% (LFW), 95.12% (YouTube)	• The system cuts the error rate by 30% as compared to other systems on both datasets, • This approach could be easily implemented for identification, validation, and clustering
[16]	PCA for masked and unmasked face recognition, used statistical procedure technique, and performed experiment using MATLAB	Dataset: ORL, Own created datasets Accuracy: 72% (masked), 95% (unmarked)	• This method is used to minimize a big dataset to a small dataset with all the information • PCA provided a poor recognition rate for masked faces than unmasked faces • Concluded that this method is only better for normal face recognition
[25]	Masked face recognition using Face-eye-based method	Dataset: SMFRD, RMFRD Accuracy: 95%	• Built the face mask dataset, used both real and artificial masked face dataset to train the model
[31]	Masked face recognition using deep learning and LBP	Dataset: COMASK20, Essex Accuracy: 87% (COMASK20), 98% (Essex)	• This approach outperformed other methods like Dlib and InsightFace,

Table 3.4 shows the summary of different works that have been done in the masked and unmasked facial recognition system. It is noticed that research [15] shows the best accuracy for unmasked facial recognition. It provided an accuracy of 99.63% for the LFW dataset. Whereas other research such as [1, 12, 14, 16, 25, 31] provide better accuracy for masked face recognition.

3.4.1 Limitation of the Existing Work, Research Gaps, and our Contributions

We reviewed the previous related works that have been done in our research fields to get more ideas. In the work of Schroff et al. [15] in 2015, the idea of the FaceNet model, a unified embedding for unmasked face recognition has been widely used for facial recognition. This approach is based on learning a Euclidean embedding per image using a deep convolution network. As a result, we applied a similar embeddings technique in our Masked Facial Recognition (MFR) research. The work [15] generated a large training model size and took a long time to train the model so we focus to solve this issue in our work. The small model can run on mobile devices and is compatible with a large server-side model.

Moreover, the researcher [13] did get better accuracy for masked face recognition using ResNet-50, however, this method worked well for unmasked faces. They used the imbalanced data to train the model which significantly reduced the accuracy of masked face recognition. We trained our model using Inception ResNet V1, which is the combination of the Inception model and Residual Network. So, we applied the balanced images per identity, performed face alignment to select only the facial part, and removed the mislabeled images from each class. As a result, all these operations helped to boost the accuracy significantly of our trained model.

Additionally, there are many systems that apply statistical procedures, Cosine, or Euclidean techniques to perform face matching. Our research focused on the Euclidean distance function to calculate the distance using face embeddings. Among all the Euclidean distance function methods, our system received better accuracy for masked face recognition. The face embeddings are useful for face matching as compared to other methods [13, 16, 25]. Although many works have been done [26, 1, 13, 16, 25] in the past for Masked Facial Recognition, we proposed a system that provides better accuracy for MFR and built a single system that can identify both masked and unmasked faces. Also, we trained a small face recognition model which helped to increase the recognition rate. Inception ResNet V1 [20] with training dataset is applied to train our MFR (Masked Facial Recognition) model. The previous work utilized real face mask images for training, whereas we generated artificial face mask images for our model training and received better recognition accuracy. As compared to the previous approaches, we used balanced images per class for model training. As a result, we received significantly high recognition accuracy even if we utilized a small number (5, 10, and 15) of images per identity.

Similarly, we applied the artificial way to create a masked face dataset for training and testing using the computer vision method. Additionally, we selected the fixed number of images per class from the CASIA dataset, and for each selected image we augmented that image four times with a different look for better diversity of training datasets. This means two masked faces and two normal faces are generated from a single image using image processing. So, all these images are applied for model training. Overall, all these approaches helped to improve the accuracy and performance of our Masked Facial Recognition system. More information about our model, approach, work, contributions, datasets, and results are presented in the methodology section.

CHAPTER 4

METHODOLOGY

This section describes the three main parts: the first is data collection and preparation, the second is training the MFR (Masked Facial Recognition) model and its evaluation, and the last one is real-time masked facial recognition using our trained model. All three steps are explained in detail below.

4.1 Data Collection and Preparation

The overall process for data collection and preparation is shown in figure 4.1. The following sections provide more information on data collection and preparation.

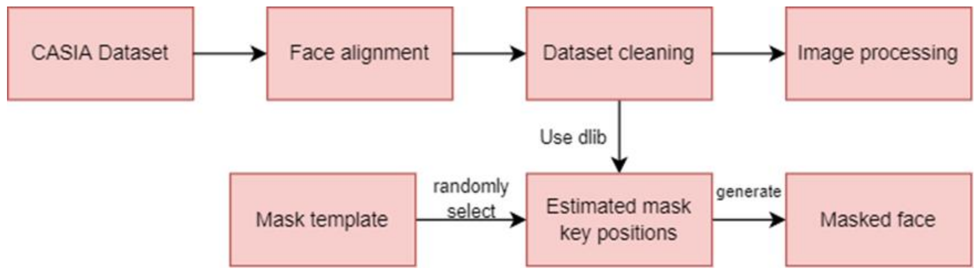


Figure 4.1 Data preparation process

4.1.1 Training Dataset

The first step of this research paper is to collect the dataset so that we can prepare the face images to train the model. For that, we used the CASIA dataset [17] which has 10585 classes, and each class has less than ten to more than one hundred images of the same person. The training images of type PNG, JPG, and VMP are only taken for data preparation. We only required the facial part of each image, however, the CASIA dataset has images with other attributes like hair, neck, and shoulder. Therefore, image alignment is taken as the first step to crop only the facial part of the images. As table 4.1.1 shows, we created three different datasets to train three different models with different numbers of training images.

Table 4.1.1 Summary of our three datasets used for training

CASIA Dataset	**# Class**	**# Images per class**	**Augmented by**	**# Masked images per class**	**# Unmasked images per class**	**Total Images**
Dataset - 1	10,585	5	4	10	10	211,700
Dataset - 2	10,585	10	4	20	20	423,400
Dataset - 3	10,585	15	4	30	30	635,100

4.1.2 Face Alignment

Face alignment is a process of cropping the face part from images and the cropped image represents the facial features. It is performed by using face detection. Face detection is the process of identifying the face from the captured image or the specified image from the database. So here, image alignment is carried out by using the SSD (Single Shot Detector) [10] face detection model. There are various methods for face detection like MTCNN (Multi-Task Cascaded Convolutional Neural Networks), Dlib, and OpenCV but we preferred the SSD method for image alignment since it is faster and easy to implement. At first, it detected the face from each CASIA class then we cropped the detected face part and saved it to the respective class folder. The margin of cropped image is set to 20 in all four directions so that the cropped face would have all the facial characters and fewer background images. Similarly, we resized all the images to [112,112,3] format where the first two numbers (112) represent the height, and width respectively, and the last value 3 denotes the number of channels. Our training images have three channels Red, Green, and Blue (RGB) format. As a result, image alignment allowed us to reduce the size of each image which helped to improve the performance and accuracy of the training model. The sample images after face alignment are shown in figure 4.1.2. Also, the source code for the face alignment is given in section A.2.

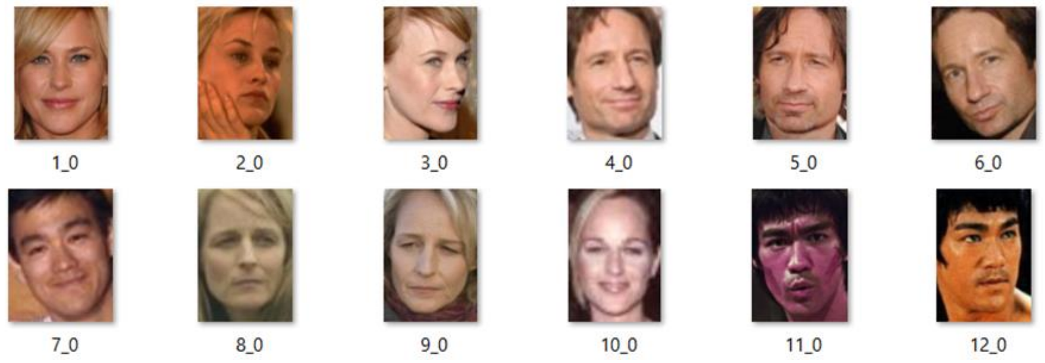


Figure 4.1.2 Sample images after face alignment

4.1.3 Data Cleaning

The CASIA classes might include mislabeled images from other classes. For example, if the class has faces that do not belong to that class, our aim is to remove those images which affect the accuracy of the training model. The process is shown in figure 4.1.3. It is not possible to remove mislabeled images from each folder by us, since it takes a lot of time. So, this is done by using FaceNet pre-trained weights [29]. First, we selected the images one by one in the same class as the target images, and the others are regarded as reference images. So, we get the 128-dimensions of embeddings for the target image, and reference images by using the FaceNet model and then calculated the average Euclidean distance between the target image and reference images. The formula to calculate the Euclidean distance is shown in the equation below. If the average distance surpasses the threshold value (0.8), then we removed that target image from the class. That means we could say that the target image does not belong to this class. Additionally, if two faces are similar, then their average Euclidean distance always should be near zero. We set the threshold value of 0.8 to remove the outlier images.

$$\text{Euclidean distance (d)} = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2 + \cdots + (x_{128} - y_{128})^2} \qquad (4\text{-}1)$$

Where x and y represent the embeddings of the target and reference images respectively. The size of the embedding is 128 dimensions, so it starts with the 1st value and goes up to the 128th value to calculate the Euclidean distance.

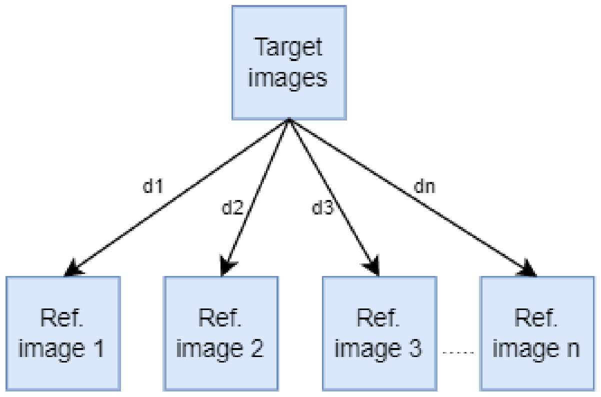


Figure 4.1.3 Process to calculate the distance between target and reference images

4.1.4 Create Masked Faces

The masked face datasets are created using a computer vision-based approach (Dlib). Dlib is useful to locate the mask key position on the face using facial landmarks. It has 68 facial landmarks, and the mouth is represented by points 48 to 68. So, these regions of interest (ROI) of the face are replaced by a random mask temple out of 16 mask images to generate the masked face images. Additionally, we resized the face mask template according to the size of mouth ROI, so it helped to fix the mask perfectly on a face image. Additionally, all the mask templates are saved in PNG files since it has four-channel, and the fourth channel is used to describe the transparency. We applied sixteen mask templates, and a random mask is selected at a time. The mask template images are shown in figure 4.1.4 (b). As a result, this approach helped to convert the CASIA dataset to a masked face dataset to train our Masked Facial Recognition model. It is difficult to collect the same person's images with a face mask and without a face mask, so this approach helps to convert any existing face dataset to a masked face dataset. The process to create a masked face dataset is shown in figure 4.1.4 (a).

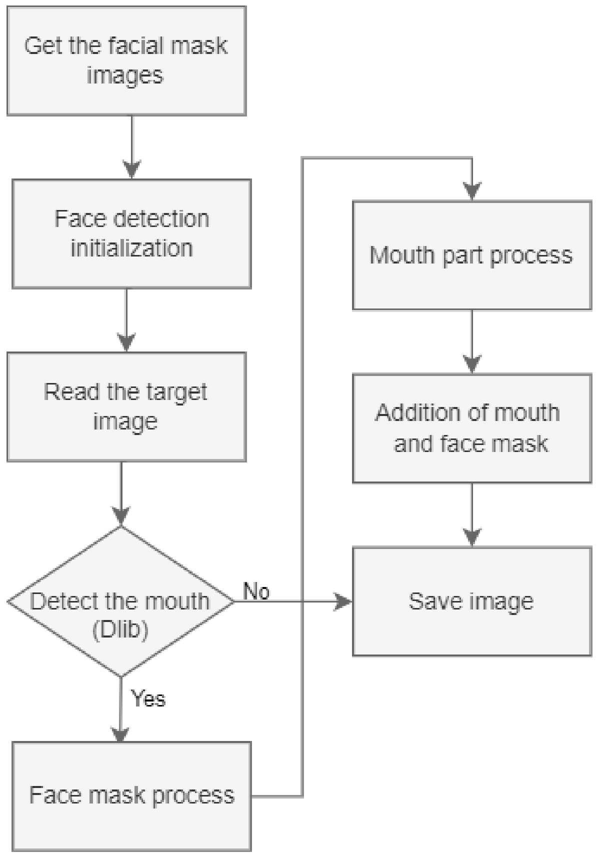


Figure 4.1.4 (a) Process to create masked face dataset

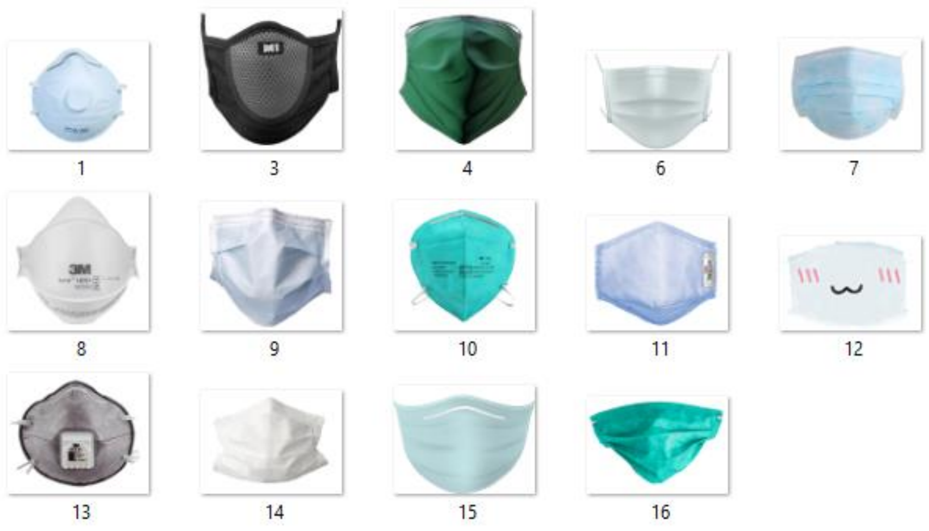


Figure 4.1.4 (b) Mask template images

4.1.5 Create Balance Dataset

The CASIA dataset has many classes, and each class has less than ten to more than 100 images of the same identity. Data imbalance is a big issue since it affects the accuracy of the trained model [13]. So, to solve this problem we selected an equal number of images for each class as shown in table 4.1.1. We randomly took a fixed number of images (5, 10, and 15) from each class at a time and created more images of the same individual with different looks using the image processing method. For each selected image we augmented each image four times. That means two masked faces and two normal faces are generated from a single image. This process utilizes random masks, random crops, random blur, random angles, random flip, and random brightness methods. All these operations were performed by using OpenCV [9] and Dlib [8]. So, this method helps to create a balanced number of images for each class. As a result, we used these balanced images to train our model. Table 4.1.1 shows how we created three different datasets to train three models with different sizes of training images. Also, it shows the number of masked and unmasked images that we used for each training model. Figure 4.4.5 shows the sample augmented images used for training. Similarly, the image processing and images argumentation source code is given in section A.3.

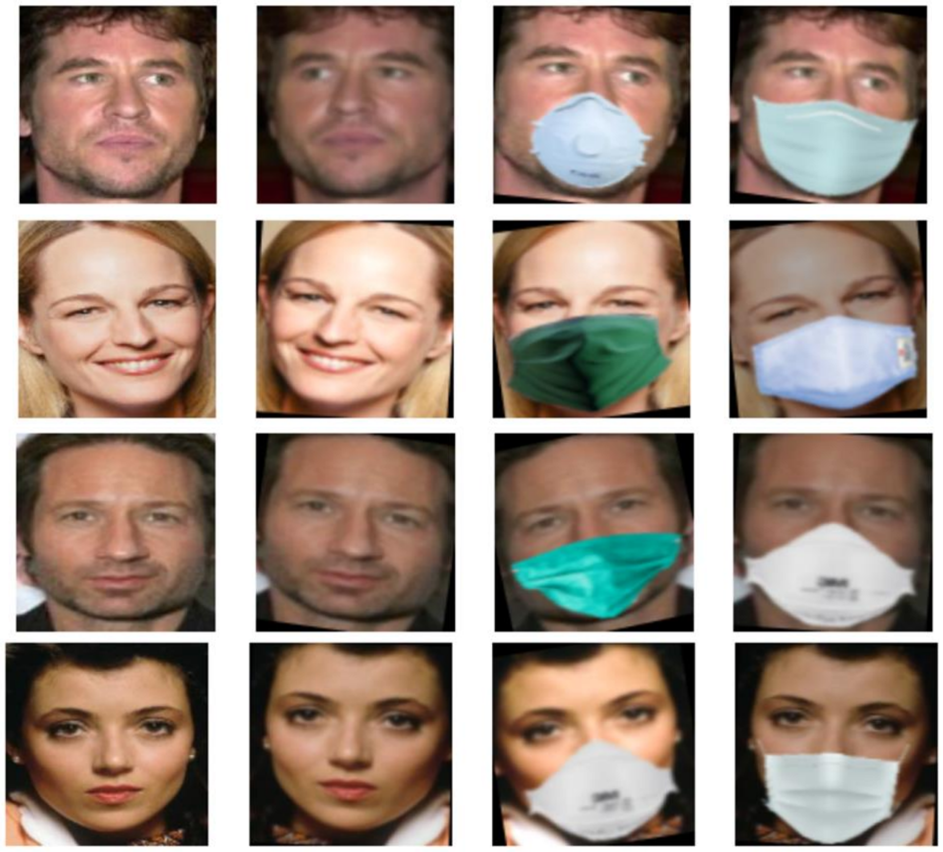


Figure 4.1.5 Sample augmented images from induvial selected images

4.2 Model Training and Evaluation

All the information on training details, training parameters, model training for masked face recognition, model evaluation, model improvement, and architecture of Inception Resnet V1 are discussed in the following section.

4.2.1 Training and Testing Details

Our model training environment includes TensorFlow 2.1, Python 3.7, OpenCV 4.5, Matplotlib 3.5, Dlib 19.23, and NumPy 1.21 of versions. The training images of size 112*112 height and width with 3 channels are used. The smaller image size can make the inference much faster to perform real-time masked face recognition, so a small image size is adopted for training. In addition, a previously prepared CASIA dataset was used for training the model. The three training datasets information is presented in table 4.1.1. We selected a fixed number of images per class, and for each face image, we created four different images. Two were masked images with distinctive looks, and two were different looks without a face mask as shown in figure 4.1.5. Overall, two masked and two unmasked images are applied per image for model training. The real-world LFW [18] dataset was tested to evaluate the model for unmasked faces using the same and different face pairs. On the other hand, we created masked faces using Microsoft celebrity face images and applied those images to evaluate the model for masked faces. The images that the trained model never learned are applied to test its performance. The testing datasets information is presented in table 4.2.1.

Table 4.2.1 Summary of masked and unmarked datasets used for model evaluation

Dataset	Type	# Class	# Images	# Images per class	Testing pair
LFW	Real-world unmasked images	5,749	64,973	11.3	Same face and different face pairs - 6,000
Microsoft Celebrity face database	Artificial masked images	85,744	85,744	1	2,000

4.2.2 Model Training

In the first step of figure 4.2.2, the training dataset includes both masked and unmasked images. For unmasked faces, whole facial features are used for model training, whereas eyes, eyebrows, and forehead areas are utilized for masked faces to extract the features. Masked faces occluded the nose, mouth, and cheeks areas so uncovered areas would only be used for feature extraction. Moreover, Inception ResNet V1 [20] with a training dataset is applied to train the MFR (Masked

Facial Recognition) model. Inception ResNet V1 has many CNN (Convolutional Neural Network) layers to perform massive calculations and store all the image features. More information about our AI model is given in the following septate section. Moreover, training loops include the different epoch operations, where we set the epoch size and perform the training loop repeatedly to optimize the model. While training the model, in each epoch the accuracy and loss function are calculated using testing images and the Cross-Entropy function, respectively. The value of the epoch is set as a fixed number, and, in each epoch, we optimized the training model weights repeatedly and calculated the accuracy and loss function of the training model. It is noted that the accuracy was increased with the increase of epoch size in training. Furthermore, accuracy is calculated by using a total number of correct predictions divided by a total number of predictions. Also, the average loss of the training model is calculated using Cross-Entropy and which is the difference between output probabilities and answers. While training the model, it was always expected to get higher accuracy and lower loss value. The process involved in the modeling training is shown in figure 4.2.2. The formula to calculate the accuracy and loss function is shown in the equations below.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{4-2}$$

$$\text{The loss (Cross-Entropy)} = -\sum_{c=1}^{M} y_{i,c} \log(p_{i,c}) \tag{4-3}$$

Where y and p represent the label and prediction, respectively.

If the prediction is equal to the answer, then the face matching is correct, and we increase the correct prediction by one. As a result, we used this value to calculate the accuracy of the trained model in each epoch. Even though we get the prediction probability from our trained model, we do not utilize that for face matching. Instead of predictions, we used embeddings for face matching.

Our trained model generated the embeddings for any input images so that would be applied to calculate the Euclidean distance. Embedding is a facial feature that is transformed into a sequence of numbers, and these are used to describe facial characteristics. We can generate embeddings of lengths 64, 128, 256, or 512, but we used 128-d embeddings size to represent the facial feature. The smaller size of the embedding might not include all the facial features, whereas the larger size will take more computation time. So, 128-d embeddings are used to represent the facial feature in our research. It is expected to run the training loop until it reaches the set epoch value, and we would expect to get higher accuracy and less loss function. After the training epoch is done, the fixed model is saved in a local folder, and we utilize that fixed model to perform real-time Masked Facial Recognition. The training model will be saved in the PB (Protocol Buffer) file, and it includes prediction and embeddings.

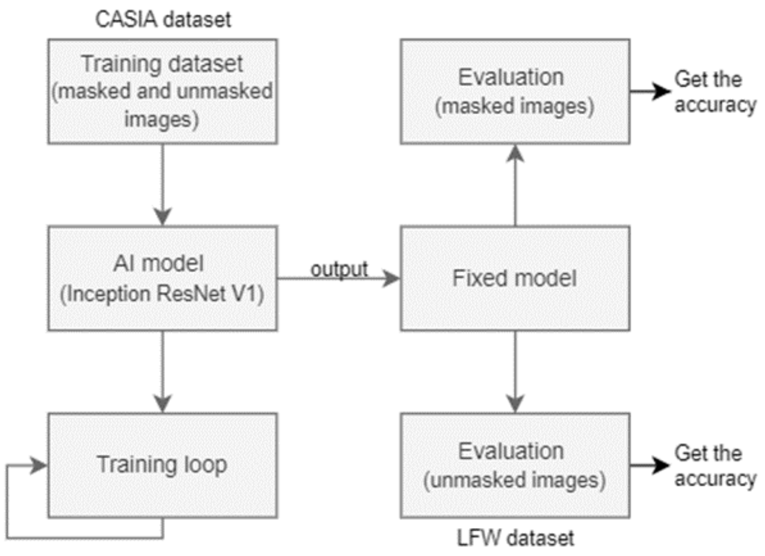

CASIA dataset
Training dataset (masked and unmasked images)
AI model (Inception ResNet V1)
Training loop
output
Fixed model
Evaluation (masked images)
Get the accuracy
Evaluation (unmasked images)
Get the accuracy
LFW dataset

Figure 4.2.2 Model training process for MFR

4.2.3 Model Improvement

In each epoch, we optimized our training model by providing the same images with a distinctive look. So, it is done by applying random masks, random crops, random blur, random angles, random flip, and random brightness using image processing and computer vision approach. Also, our training dataset has more than 10,000 unique classes with different ages, races, and genders so we passed all those images which helped to learn our model from diverse face images. Additionally, we performed image alignment for both training and testing images which allowed us to reduce the size of each image and only included the facial part. That helped to improve the performance and accuracy of the training model. Also, the removal of the mislabeled image from each class helped to maximize the accuracy of the system since our model will not learn from the wrong images. From the research [13], we comprehended that the imbalanced images affect the accuracy of the training model, so we selected the balance number of images from each class for our three different trained models. Furthermore, we applied the smaller batch size (96, or 192 images per iteration) which improved the training time of our model. As a result, our three different models only took on average 30 hours to get the maximum accuracy. Similarly, we utilized small training images with height and width 112*112 respectively and reduced the filter size by half so that helped to reduce the size of our training model. Small training models are important since it takes less time for inference and increases the performance of recognition. Also, the small model can run on mobile devices and is compatible with a larger server-side model.

4.2.4 Architecture of our AI Model

4.2.4.1 Why this architecture

The normal classification model only outputs the probabilities of trained class numbers, and it is impossible to train different faces all over the world. The very deep convolutional networks

(Inception model) have been applied for facial recognition systems in the past and it has shown better performance and low computational cost [21]. The combination of Inception architecture [22] and residual network [23] (Inception ResNet V1) provides better recognition performance since training with residual connections accelerates the training of Inception networks. Therefore, Inception ResNet V1 architecture is proposed to use, and this model provides the embeddings which are applied to perform face matching. The block diagram of the Inception model and Residual network are shown in figures 4.2.4.1 (a), and 4.2.4.1 (b) respectively.

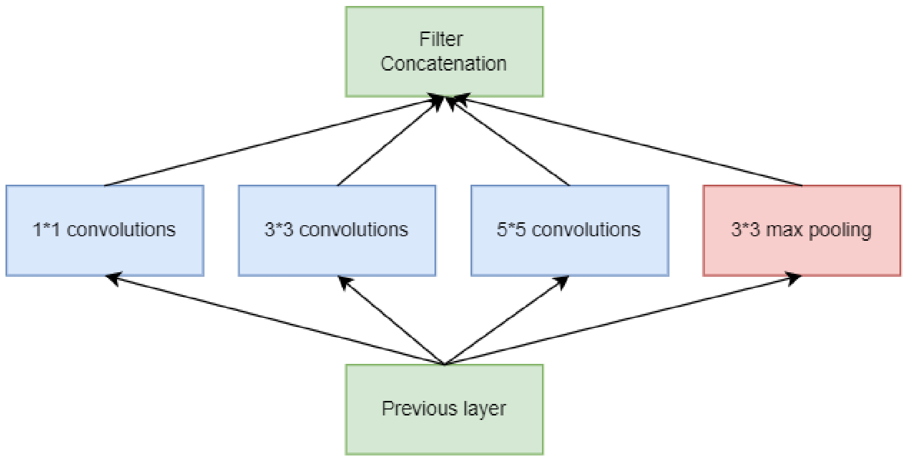


Figure 4.2.4.1 (a) Inception module

From figure 4.2.4.1 (a), the Inception module is the combination of 1*1, 3*3, and 5*5 convolutions layers and 3*3 max pooling. Here, 1*1 convolutions are applied to modify the channel numbers, and the final goal is to decrease the model weights. The smaller model means fewer calculations and makes the inference much faster.

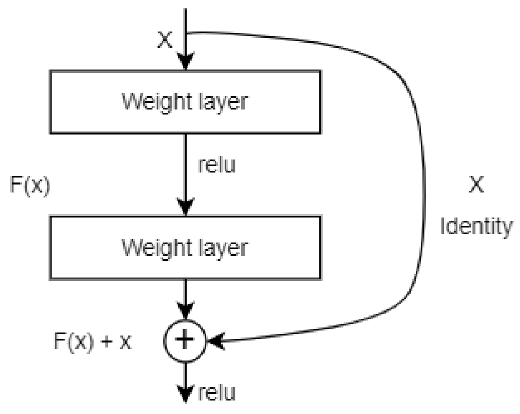


Figure 4.2.4.1 (b) Residual learning: a building block

From figure 4.2.4.1 (a), the Residual learning includes the weights of size 3*3, and filter size 64. The activation function, relu is applied for calculation. More information on the architecture is provided in the following section.

4.2.4.2 Architecture of Inception ResNet V1

The prepared input images are fed to the AI (Artificial Intelligence) model for model training. Inception ResNet V1 has weight layers, CNN (Convolutional Neural Network) layers, average pooling, stem, reduction, fully connection (FC), and softmax. Different parameters are applied to Inception ResNet architecture such as filter size of 32, 64, 80, 192, or 256, 3*3 kernel size, the activation function (ReLU), batch size of 32, 96, or 192, the learning rate is 0.0005, optimization method (Adam), strides, fix epoch number, model size, and balance images from every 10585 classes. Batch size is helpful to improve the training performance. We can’t feed all the training images at one time, so we pass the fixed batch number of images for each iteration. The total number of training images divided by batch size is the total number of iterations per epoch. The training parameters for our three training models are presented in table 4.2.4.2.

Table 4.2.4.2 Parameters for three different training models

Parameters	**Values**		
	Model 1	**Model 2**	**Model 3**
Feature number	128-d	128-d	128-d
Learning rate	0.005	0.005	0.005
Maximum epoch	38	50	61
Batch size	192	96	96
Loss	Cross Entropy	Cross Entropy	Cross Entropy
Optimizer	Adam	Adam	Adam
Model shape	[112,112,3]	[112,112,3]	[112,112,3]
Masked images	317,550	211,700	105,850
Unmasked images	317,550	211,700	105,850

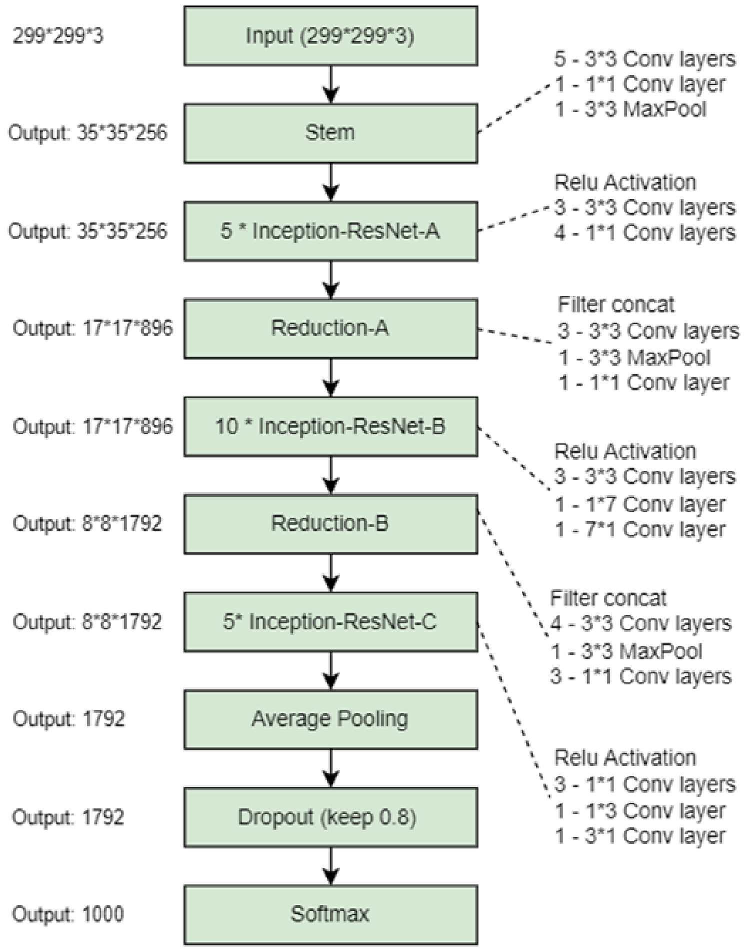


Figure 4.2.4.2 Architecture of Inception ResNet V1 model

Inception ResNet [20] is a combination of the Inception and ResNet models where it has ten different steps to train the model. As figure 4.2.1 shows in the first step, the input images are passed through the stem. The stem has five 3*3 convolutions of different filter sizes 32, 64, 80, 192, and 256, respectively. It has one 3*3 MaxPool of stride 2. Also, it includes 1*1 Convolution. The 1*1 is applied to modify the channel numbers, and the final goal is to decrease the model weights. The smaller model means fewer calculations and makes the inference much faster. The weights are calculated by multiplying kernel size, filter size, and the number of channels. Similarly, the second step is 5 times Inception ResNet-A. It contains three 3*3 convolutions of filter size 32, and four 1*1 convolutions of filter size 32, and 256. Moreover, the third step is Reduction-A which includes the convolutions of 1*1, 3*3, and MaxPool of size 3*3. Additionally, it is followed by 10 times of Inception ResNet-B, Reduction-B, 5 times Inception ResNet-C, Average Pooling, Dropout, and the last one is Softmax. Average Pooling computes the average value of each feature map and returns that value. The embeddings of size 128 dimensions are received from Dropout once the normalization is done on it. Dropouts passed through the Softmax to get the prediction value. The value of prediction will be 10585 since our CASIA dataset has that number of classes. We get the prediction value from Softmax whose output ranges from 0 to 1. Softmax can make the bigger number become a larger ratio and make the smaller numbers even smaller. As a result, our trained model includes embeddings and predictions. The source code of the Inception ResNet V1 model is given in section A.1.

4.2.4.3 How to generate embeddings

The face embeddings are the numeric values of facial features which are used to recognize the person by calculating the Euclidean distance between the target and reference images. Moreover, we can generate embeddings of lengths 64, 128, 256, or 512, but we used 128-d embeddings size to represent the facial feature. The smaller size of the embedding might not include all the facial features, whereas the larger size will take more computation time. So, 128-d embeddings are used to represent the facial feature in our research. In our model, the embeddings of size 128-d are received from Dropout once the normalization is done on it. The process to get the embeddings is shown in figure 4.2.4.3.

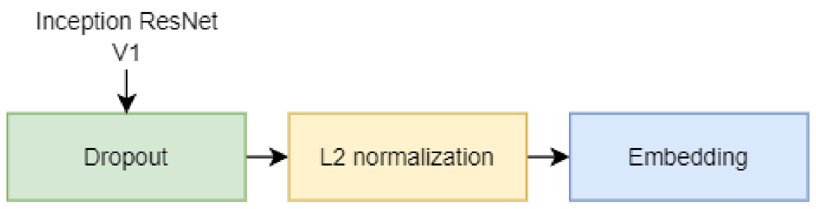


Figure 4.2.4.3 Process to get the embeddings

The source code to generate embeddings is given below:

prelogits, _ = inception_resnet_v1(tf_input, tf_keep_prob, phase_train=tf_phase_train, bottleneck_layer_size=embed_length, weight_decay=0.0, reuse=None)

prelogits = tf.identity(prelogits,name='prelogits')

embeddings = tf.nn.l2_normalize(prelogits, 1, 1e-10, name='embeddings')

4.2.4.4 How to use embeddings to perform face matching

First, calculate the embeddings of input images and face database. Then compute the Euclidean distances between the input embeddings with all the face database embeddings. As shown in figure 4.2.4.4, the input image is the target image, and the face database is the reference image. Similarly, we calculated the Euclidian distance between target images with each reference image as shown in figure 4.2.4.4 using face embeddings. Among all the distances, we find out the smallest distance, and if the smallest distance is even smaller than the threshold value (0.8), then that would be an answer. Therefore, the smallest Euclidean distance is compared with the threshold value to make the final face matching decision. The formula to calculate the Euclidean distance is shown in the equation below.

$$\text{Euclidean distance (d)} = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2 + \cdots + (x_{128} - y_{128})^2} \qquad (4\text{-}4)$$

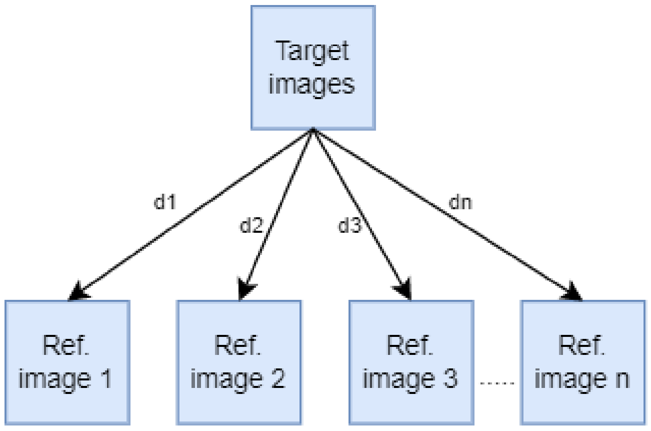


Figure 4.2.4.4 Calculate Euclidean distance between target and reference images

4.2.5 Model Evaluation

We used LFW face images and artificial masked faces that the model never learned to evaluate its real ability. LFW images are mainly used for unmasked faces, whereas artificial masked images are utilized for the evaluation of masked images. The Microsoft celebrity face database is utilized to generate masked faces using the computer vision approach. The summary of the testing dataset is provided in table 4.2.1. As a result, we prepared both types of masked and unmasked faces to evaluate our trained model.

4.2.5.1 Unmasked face evaluation flowchart

As shown in figure 4.2.5.1, we selected 3,000 same face pairs and 3,000 different face pairs from the LFW dataset. Overall, 6000 face pairs are used to evaluate the model. Our trained model named "Fixed model" is applied to calculate the embedding of each pair. Then the Euclidean distance is calculated using those embeddings pairs. Additionally, for the same face pair, the distance is compared to the threshold value (0.8), and if only the distance is less than the threshold value then the counter of correct prediction is increased by one. On the other hand, for different face pairs, the calculated distance is compared to the threshold value (0.8), and if only the distance is greater than equal to the threshold value then the counter of correct prediction is increased by one. Similarly, the accuracy is computed by diving the total number of correct predictions by a total number of face pairs (6,000). In this way, we evaluated our trained model using unmasked face images. The overall flowchart of unmasked face evaluation is shown in figure 4.2.5.1.

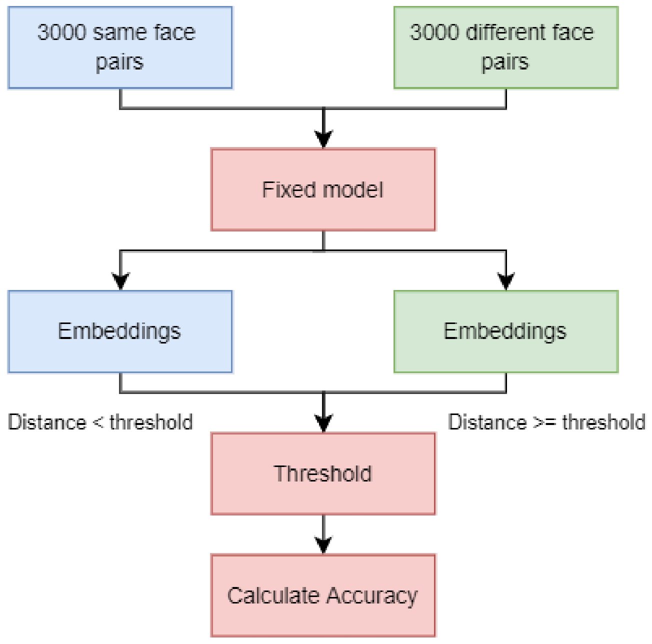


Figure 4.2.5.1 Unmasked face evaluation flowchart

4.2.5.2 Masked face evaluation flowchart

As shown in figure 4.2.5.2 (b), we selected artificial masked images (tar_images) and a face database (ref_images) to evaluate our model. We created artificial masked faces using a computer vision approach as mentioned in section 4.1.4. For both artificial masked faces and real-world face databases, we used the Microsoft celebrity face database to create our dataset. More information about the testing data is provided in table 4.2.1. Moreover, our trained model named "Fixed model" is utilized to generate the embedding for both tar_images and ref_images. Once we get the target embeddings (tar_embeddings) and reference embeddings (ref_embeddings) for both target and reference images. The Euclidean distances are calculated for each tar_embeddings with all the ref_embeddings as shown in figure 4.2.5.2 (a). Among all the distances, we find out the smallest distance, and if the smallest distance is even smaller than the threshold value (0.8), and the name of the target face is equal to the name of the reference face, then the counter of correct prediction is increased by one. Similarly, all the target images are compared with the reference images by calculating the Euclidean distance. Additionally, the accuracy is computed by diving the total number of correct predictions by total number of predictions (2,000). We received the accuracy of our trained model at around 97% for masked face evaluation. In this way, we evaluated our trained model using masked faces. The overall flowchart of masked face evaluation is shown in figure 4.2.5.2 (b).

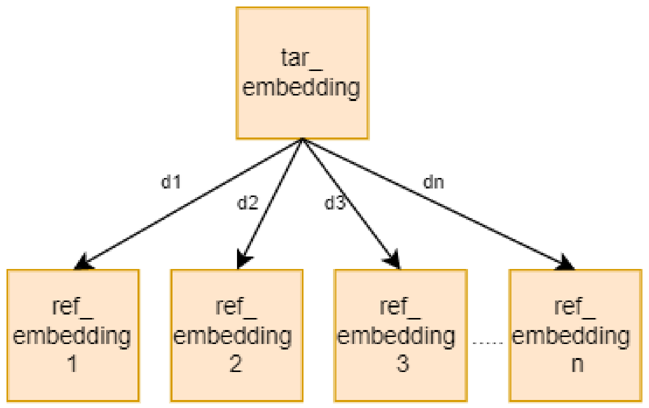


Figure 4.2.5.2 (a) Calculate distance between one tar_embedding to many ref_embedding

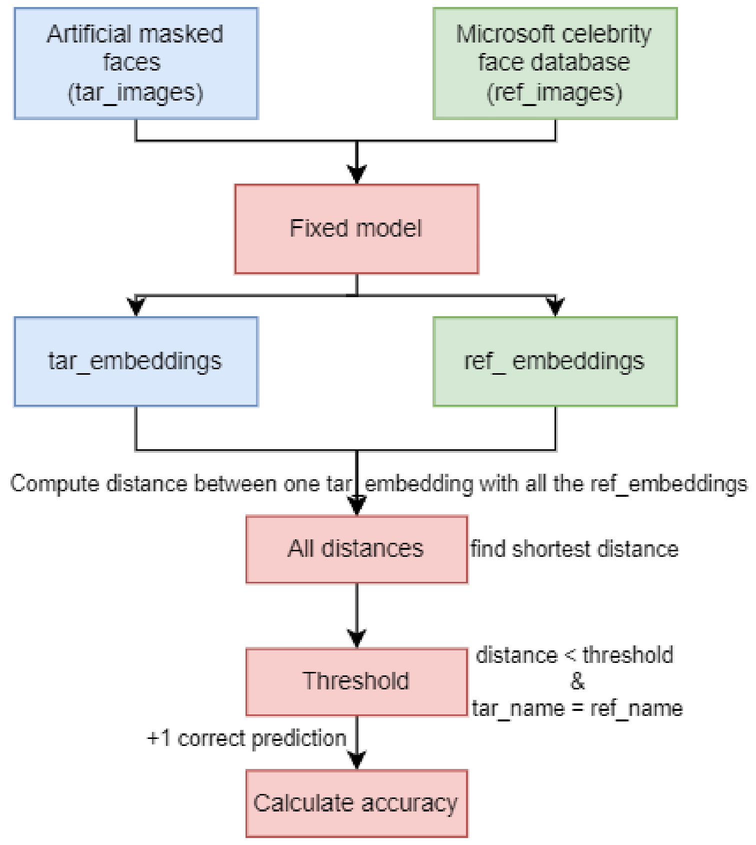

Artificial masked faces (tar_images)
Microsoft celebrity face database (ref_images)
Fixed model
tar_embeddings
ref_ embeddings
Compute distance between one tar_embedding with all the ref_embeddings
All distances
find shortest distance
Threshold
distance < threshold & tar_name = ref_name
+1 correct prediction
Calculate accuracy

Figure 4.2.5.2 (b) Unmasked face evaluation flowchart

4.2.5.3 Accuracy comparison with FaceNet model using the same dataset

We passed the same masked faces for both models: one is our trained model (Fixed model), and another is FaceNet [15] pre-trained model. It was studied that our model worked really well for masked face images, however, the FaceNet model did not work well for masked faces. Table 4.2.5.3 shows the accuracy comparisons between the three models.

FaceNet model is a prominent model for normal face recognition (99% accuracy), however, this trained model did not work well for masked face recognition. As we can see from table 4.2.5.3, our trained model and FaceNet model used the same training dataset, testing dataset and same architecture, but our model achieved around 97% accuracy for Masked Face Recognition (MFR). So, we can say that our model worked better for masked face recognition.

Table 4.2.5.3 Accuracy comparisons between the three different models

Model name	Accuracy (Threshold – 0.8)	Testing dataset	Training dataset	Architecture
Fixed model (our)	96.9%	Artificial masked face	CASIA	Inception ResNet V1
20180408-102900 [15]	45.11%	Artificial masked face	CASIA	Inception ResNet V1
20180402-114759 [15]	60.49%	Artificial masked face	VGGFace2	Inception ResNet V1

4.3 Real-Time Masked Facial Recognition using our Trained Model

First, as shown in figure 4.3, our system loads all the face database images and computes the embeddings of each face image using a Fixed model. The Fixed model is our trained model for MFR. We performed the face alignment for the face database also so that it removed the unnecessary part from the face images. This makes the system ready to perform the facial matching once the system reads the input images. We utilized the Microsoft celebrity face images that our model never learned for the face database. Our face image was also included in the face database for the experiment. The Microsoft celebrity images consist of a total of 85744 pictures but in our experiment, we only used 2000 images for testing purposes. All those images were real-world unmasked faces.

Secondly, the input image is read from the real-time video streaming using a laptop camera and we used the high-resolution input images to achieve better performance. The SSD (Single Shot Detector) model detected the face and face mask of each input frame. Then the facial part is cropped from the image frame and resized to the [112, 112, 3] format. Additionally, our trained model, the Fixed model is applied to find out the embeddings of an input image. Embeddings represent the facial feature in the numeric format of size 128- dimensions. And these embeddings are used to calculate the Euclidean distance.

Finally, the Euclidean distances are calculated by using the embedding of the input face with preloaded embeddings of all the face databases. This is a one-to-many calculation as shown in figure 4.2.5.2 (b). Preloading the embedding to the system makes facial recognition much faster since the input image does not need to wait to calculate the Euclidean distance. Moreover, the distance should be around zero if two images belong to the same person. Among all the distances, we find out the smallest distance, and if the smallest distance is even smaller than the threshold value (0.8), then that would be an answer. Therefore, the smallest Euclidean distance is compared with the threshold value to make the final facial recognition decision. If the distance is less than the threshold value (face matched), then the name of the person will be printed on the input image frame, otherwise, an unknown message is printed. Similarly, if the person is wearing a face mask, this system shows the "Mask" message, else "No Mask" message is displayed. The overall process of real-time Masked Facial Recognition (MFR) is shown in figure 4.3. Hence in this way, we built a single system that can recognize both masked and unmasked faces with high accuracy. This has indicated the effectiveness and suitability of the proposed method. The result of real-time masked facial recognition is shown in figure 5.7. Also, the source code for real-time masked face recognition is given in section A.5.

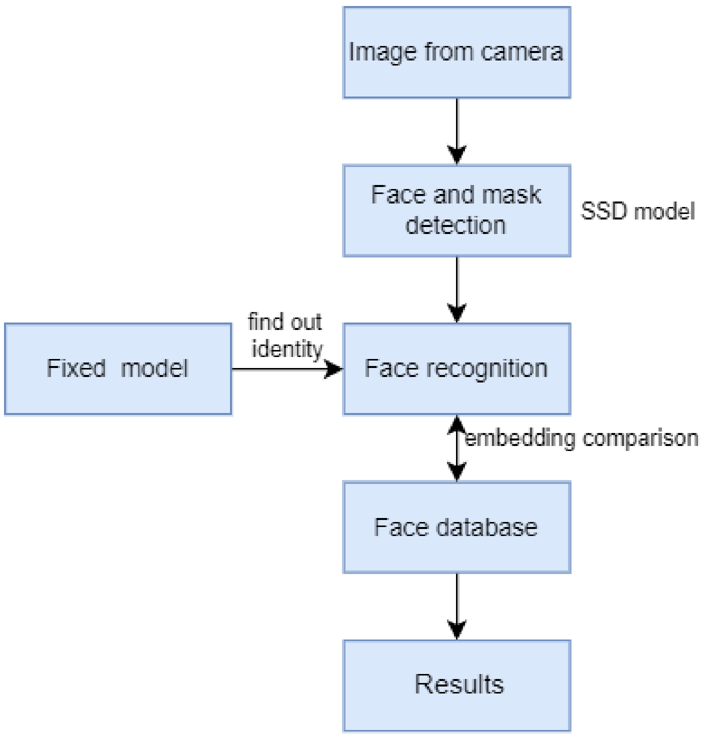


Figure 4.3 Real-time masked facial recognition process

CHAPTER 5

RESULTS AND DISCUSSION

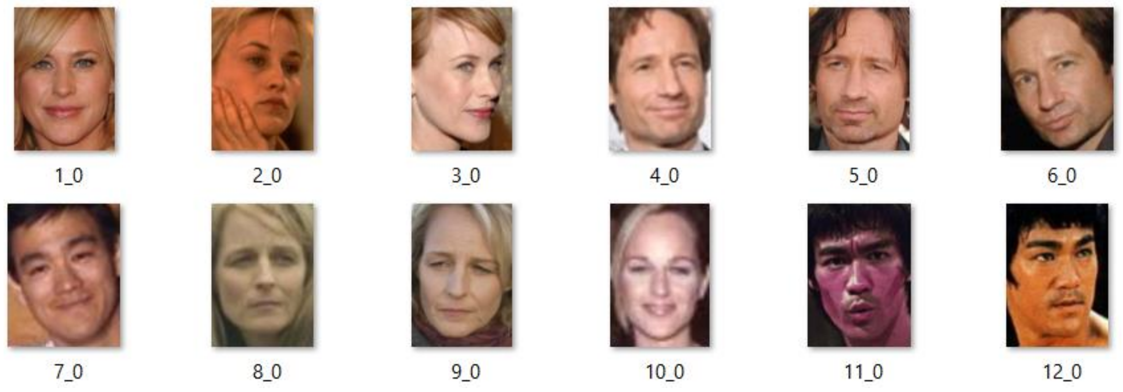
This chapter describes the output of the experiments carried out on MFR (Masked Facial Recognition). It presents an in-depth explanation of various experiments meant to assess the effectiveness of our solution. Also, it highlights additional information about our experimental setup, training, and testing dataset, comparisons of our approach against other methods, limitations and advantages of our approach, performance metric, training parameters, and performance evaluation for different setups of environments.

5.1 Experiment Setup

Our experiments were carried out on a computer with an Intel Core i7 vPro processor, 16 GB of RAM, 500 GB SSD (Solid-State Drive) Hard disk, Windows 10 OS (operating system), and Nvidia GPU (Graphics processing units) card. Additionally, we used Python as a programming language, and tools such as OpenCV, TensorFlow, CUDA, NumPy, and Matplotlib for image processing and model training, whereas Jupyter and PyCharm were utilized as an IDE (Integrated Development Environment). Furthermore, TensorFlow is significantly important to perform the massive math calculations for building the model, and GPU helps to do the operation much faster. GPU has thousands of cores which can finish many calculations faster than CPU which normally has 8 cores. TensorFlow offers many models and relative functions and helps to communicate with GPU to do the task. Whereas OpenCV has rich image processing functions like reading, saving, resizing, displaying, cropping, transforming, and changing the color format of the images so it was applied for data preparation, image processing, and model training for Masked Facial Recognition (MFR).

5.2 Dataset

We had to have both masked and unmasked face images to train our model. CASIA [17] datasets are applied for unmasked images after image preprocessing, and we generated artificial masked images from the CASIA dataset using the mask augmentation method (Dlib). LFW (Labeled Faces in the Wild) [18] datasets with artificial masked faces were used for testing. We created artificial test masked images from the Microsoft face database. The process to create masked faces is given in section 4.1.4. All the datasets are open-source images that are easily accessible online. The training and testing dataset information is given in tables 4.1.1, and 4.2.1 respectively. The training and testing faces are used in size [112, 112, 3] format. Moreover, we implemented the image processing method to make the same face images with different looks that utilized random crops, random noise, random angle, random flip, and random brightness methods. Also, we performed face alignment, and data cleaning to make the dataset ready for the training. The sample training and testing faces are shown in figures 5.2 (a), and 5.2 (b) respectively.



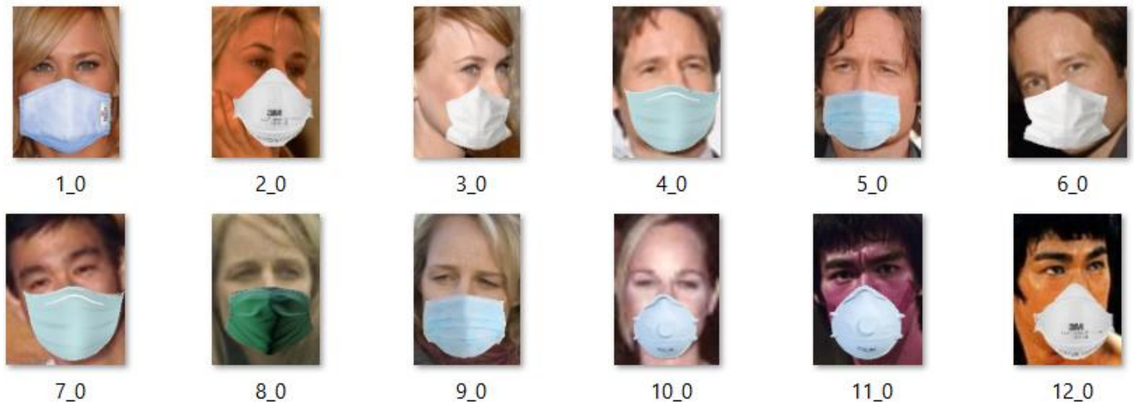


Figure 5.2 (a) Sample masked and unmasked training images

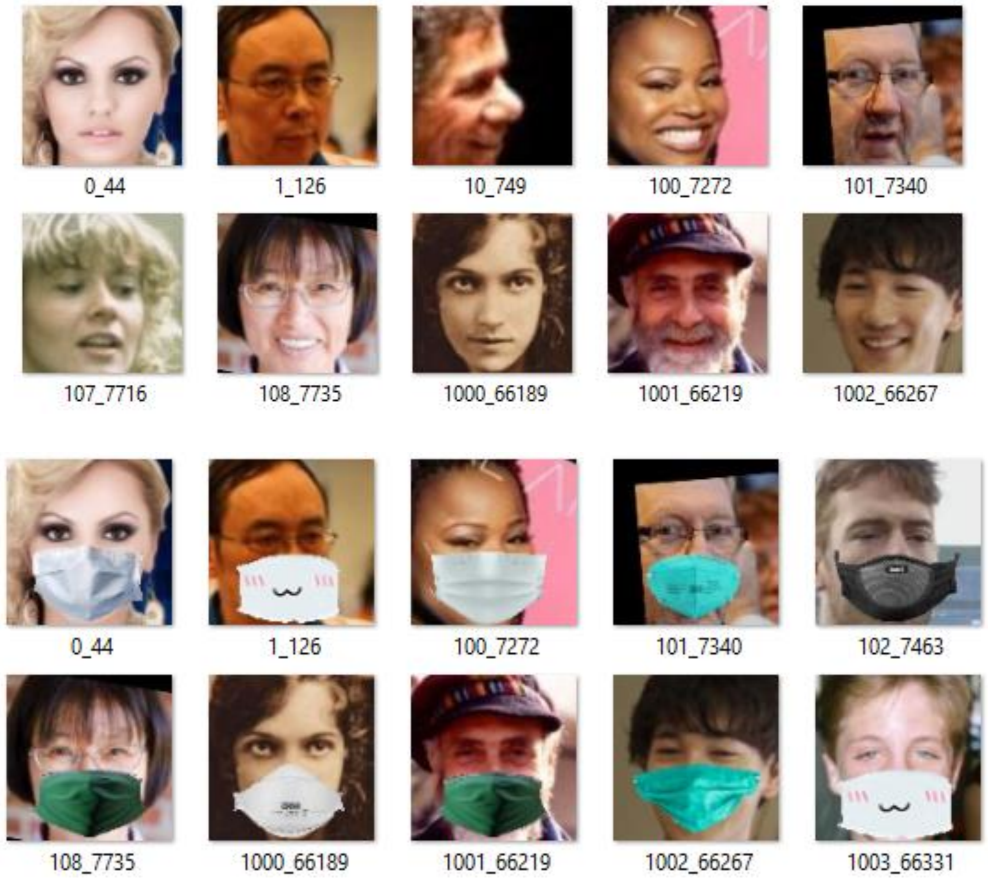
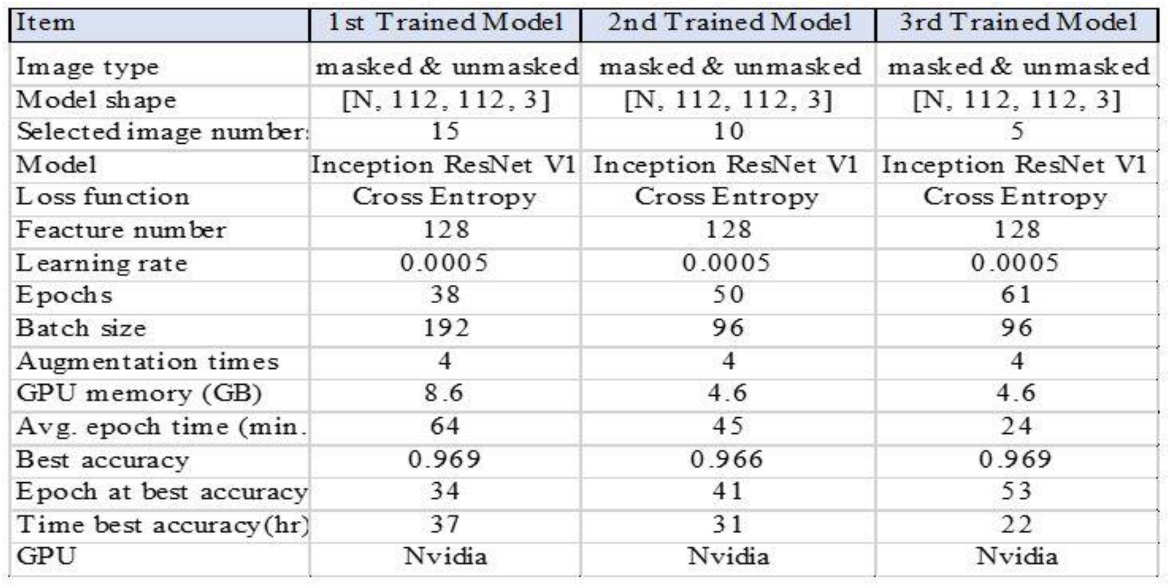


Figure 5.2 (b) Sample masked and unmasked testing images

5.3 Performance Evaluation of our Trained models

We trained the three different models with five, ten, and fifteen training images from each class and augmented each image four times to create more images with different looks. The training parameters that we used and the performance evaluation of the three different trained models are presented in table 5.3. More information on training and testing datasets is given in tables 4.1.1, and 4.2.1 respectively.

Table 5.3 Performance evaluation of three different models



Item	1st Trained Model	2nd Trained Model	3rd Trained Model
Image type	masked & unmasked	masked & unmasked	masked & unmasked
Model shape	[N, 112, 112, 3]	[N, 112, 112, 3]	[N, 112, 112, 3]
Selected image number:	15	10	5
Model	Inception ResNet V1	Inception ResNet V1	Inception ResNet V1
Loss function	Cross Entropy	Cross Entropy	Cross Entropy
Feacture number	128	128	128
Learning rate	0.0005	0.0005	0.0005
Epochs	38	50	61
Batch size	192	96	96
Augmentation times	4	4	4
GPU memory (GB)	8.6	4.6	4.6
Avg. epoch time (min.	64	45	24
Best accuracy	0.969	0.966	0.969
Epoch at best accuracy	34	41	53
Time best accuracy(hr)	37	31	22
GPU	Nvidia	Nvidia	Nvidia

Table 5.3 shows that the first trained model selected only 15*4 images from each class and this model received the best accuracy of 96.9% in the 34th epoch. Similarly, the second trained model received only 10*4 images for each class, and the third trained model selected only 5*4 images per class and obtained the best accuracy of 96.6% and 96.9% respectively. We trained the model with the imbalanced and balanced dataset, and it is studied that the significant improvement in the accuracy is with the balanced dataset, so we applied an equal number of training images for each class of our training model.

Moreover, the Inception ResNet V1 deep learning model is applied for all three training models. Each input face is augmented four times to generate more training images and Nvidia GPU (Graphics Processing Units) made the training process much faster since all the trained models received the best accuracy in 30 hours on average. It is observed that even if we used a smaller number of training images, we have achieved significantly better testing accuracy for the LFW dataset. It is because we focused the training image more on face alignment, removed mislabeled images, and took the balanced images per class which helped to improve the accuracy of our trained model. Additionally, if we select a smaller number of training images, it would also be easy to train the AI (Artificial Intelligence) model with normal GPU cards such as GTX 1660.

5.4 Evaluation Metrix

To analyze the performance of the trained models, we used the following metrics:

- **Accuracy:** Accuracy is calculated by using a total number of correct predictions divided by a total number of predictions.
- **The loss:** The average loss of the training model is calculated using Cross-Entropy and which is the difference between output probabilities and answers.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\textit{Total number of predictions}} \tag{5-1}$$

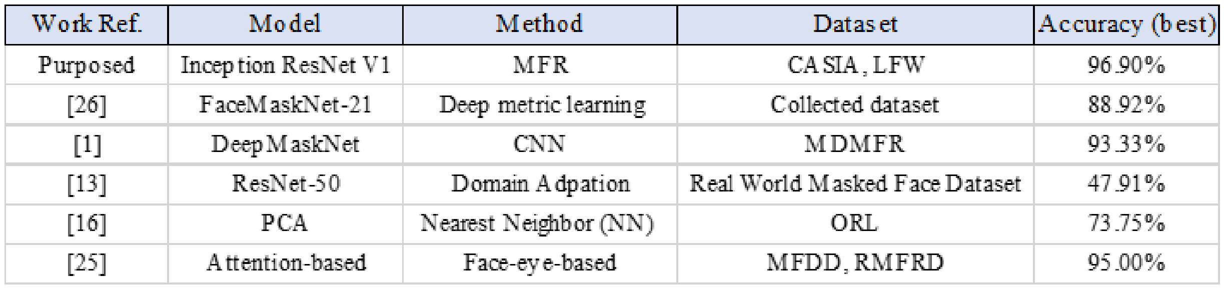
$$\text{The loss (Cross-Entropy)} = -\sum_{c=1}^{M} y_{i,c} \log(p_{i,c}) \tag{5-2}$$

Where y and p represent the label and prediction, respectively.

5.5 Comparison of Current Approach Against Other Methods

Table 5.5 shows the comparison of our work with other different methods. Based on these results, we studied that our MFR (Masked Facial Recognition) method significantly outperforms the five other models. Additionally, we achieved an accuracy of 1.9 percent higher than the second-best performing model (Attention-based) and around 49 percent more than the worst performing model (ResNet-50) for masked and unmasked facial recognition. Our purposed model (Inception ResNet V1) is the combination of Inception architecture [22] and residual network [23] and it provides better recognition performance since training with residual connections accelerates the training of Inception networks. This has indicated the effectiveness and suitability of the proposed method.

Table 5.5 Comparison of current approach against other methods



Work Ref.	Model	Method	Dataset	Accuracy (best)
Purposed	Inception ResNet V1	MFR	CASIA, LFW	96.90%
[26]	FaceMaskNet-21	Deep metric learning	Collected dataset	88.92%
[1]	DeepMaskNet	CNN	MDMFR	93.33%
[13]	ResNet-50	Domain Adpation	Real World Masked Face Dataset	47.91%
[16]	PCA	Nearest Neighbor (NN)	ORL	73.75%
[25]	Attention-based	Face-eye-based	MFDD, RMFRD	95.00%

5.6 Limitations and Advantages

The limitations of our MFR system are the input masked face should have at least eyes and forehead parts visible for recognition, it might not work for more tilted masked images toward left or right and setting the model training environment might be challenging since it requires GPU setup, at least i5 processor, 16 GB of RAM, and 500 GB SSD (Single Shot Detector) Hard disk for better performance. Also, our approach required a lot of work for data collection and preparation, and it could be difficult to get masked faces for more than 10,000 identities to train our model.

The advantages of our MFR system are it supports multiple mask types for face recognition, and it works for diverse types of faces, ages, and genders. Also, our model supports both single and multi-face images for recognition and can convert any face datasets to masked face datasets. From our model evaluation, it works better for both masked and unmasked faces so the probability of giving the right recognition decision is around 97%. Our single system can identify faces with and without face masks. We improved the model training time and decreased the size of the trained model as compared to other methods [12, 13, 15]. So, all these advantages indicate the effectiveness and suitability of the proposed method for masked facial recognition.

5.7 Result Analysis

We have created a system that can recognize both masked and unmasked face images. Our trained model generates the embeddings for any given input image, and that embedding is applied for face matching. Embedding represents the facial feature in the numeric format, and it has the size of 128-dimensions. In our research, we implemented a new augmented way to create a masked face dataset and performed the image alignment and data cleaning using Dlib, OpenCV, and SSD (Single Shot Detector) model. Similarly, we applied the balanced face images from each class and received significantly better results than the model trained with imbalanced images. Balanced images include an equal number of masked and unmasked face images. LFW and face mask datasets were tested for evaluation, and they showed superiority over any contemporary models [1, 13, 16, 25, 26]. Our evaluation model included the LFW dataset and artificial masked images which were not used to learn the training model. LFW images after preprocessing are applied for unmasked face evaluation, while artificial masked images are used for masked face evaluation. We generated artificial masked images from celebrity images from the Microsoft face database. It has more than 8500 unique faces. The training and testing dataset information is given in tables 4.1.1, and 4.2.1 respectively.

Furthermore, we verified the robustness of our purposed model for masked and unmasked facial recognition under various conditions like gender, skin tone, age, types of masks, etc. As a result, we achieved the MFR (Masked Facial Recognition) of around 97% accuracy for our three different trained models. The training and testing faces are used in size [112, 112, 3] format. The effectiveness of our trained model is evaluated for both masked and unmasked faces. The eyes, eyebrows, and forehead areas are utilized for masked faces to extract the facial feature, whereas the whole face is used for unmasked faces. The accuracy and loss function are calculated to test the model while training it. Also, we compared the accuracy by using the same masked faces for our model and pre-trained FaceNet model in section 4.2.5.3. Our model worked really well for masked faces as compared to the FaceNet model. Additionally, we built a single system that can identify both masked and unmasked faces, and also it can recognize more than one face at a time.

On top of that, we trained the small training model of a size of around 96 MB for our three different trained models which provided a better recognition rate. We applied the small training image size, performed the face alignment, and used the smaller filter size which helped to minimize the size of our training model. In model training, 1*1 convolutions are applied to modify the channel numbers, and the final goal was to decrease the model weights. The smaller model means fewer calculations and makes the inference much faster. Similarly, we speed up our training model by dividing the training images into different batch sizes and also GPU helped to do the operation much faster. As a result, we train three different models in 30 hours on average time. The results of our real-time masked face recognition are shown in figure 5.7.

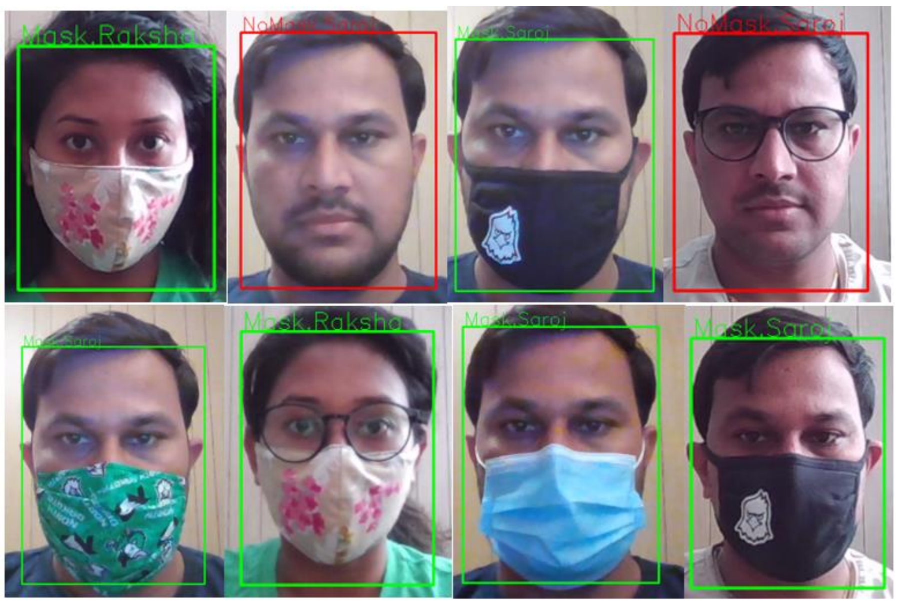


Figure 5.7 Real-time masked facial recognition results

CHAPTER 6

CONCLUSION

6.1 Conclusions

In conclusion, this research paper has presented a solution to identify the masked and unmasked faces accurately. The proposed approach provided around 97% accuracy for MFR (Masked Facial Recognition). Furthermore, the masked face dataset was created using a computer vision technique. CASIA datasets were used to train the model after performing image preparation and the LFW (Labeled Faces in the Wild) dataset and artificial masked faces were tested to evaluate the performance of our model. Also, the performance of three different models has been studied for MFR. Additionally, we verified the robustness of our purposed model for masked and unmasked facial recognition under various conditions like gender, skin tone, age, types of masks, etc. As a result, the purposed solution could be seamlessly integrated for both masked and unmasked face recognition and detection systems that are designed for safety and security verification purposes without any challenges.

6.2 Future work

In the future, we intend to use the real-time mask face dataset since some of our generated masked images do not perfectly fit the rotated faces, so using the real-world masked images could increase the recognition accuracy of the system. Also, it is expected to increase the number of balanced images for each class to train the model for better quality and diversity (we applied a maximum of 60 faces per class in our experiment). Additionally, we would try to build a small facial recognition model which could improve the overall recognition rate of the system.

APPENDICES A

A.1 Inception ResNet V1 Architecture Python Code

```
#Inception-ResNet-V1 Architecture
def block35(net, scale=1.0, activation_fn=tf.nn.relu, scope=None, reuse=None):
    """Builds the 35x35 resnet block."""
    with tf.variable_scope(scope, 'Block35', [net], reuse=reuse):
        with tf.variable_scope('Branch_0'):
            tower_conv = slim.conv2d(net, 32, 1, scope='Conv2d_1x1')
        with tf.variable_scope('Branch_1'):
            tower_conv1_0 = slim.conv2d(net, 32, 1, scope='Conv2d_0a_1x1')
            tower_conv1_1 = slim.conv2d(tower_conv1_0, 32, 3, scope='Conv2d_0b_3x3')
        with tf.variable_scope('Branch_2'):
            tower_conv2_0 = slim.conv2d(net, 32, 1, scope='Conv2d_0a_1x1')
            tower_conv2_1 = slim.conv2d(tower_conv2_0, 32, 3, scope='Conv2d_0b_3x3')
            tower_conv2_2 = slim.conv2d(tower_conv2_1, 32, 3, scope='Conv2d_0c_3x3')
        mixed = tf.concat([tower_conv, tower_conv1_1, tower_conv2_2], 3)
        up = slim.conv2d(mixed, net.get_shape()[3], 1, normalizer_fn=None,
                         activation_fn=None, scope='Conv2d_1x1')
        net += scale * up
        if activation_fn:
            net = activation_fn(net)
    return net
# Inception-Resnet-B
def block17(net, scale=1.0, activation_fn=tf.nn.relu, scope=None, reuse=None):
    """Builds the 17x17 resnet block."""
    with tf.variable_scope(scope, 'Block17', [net], reuse=reuse):
        with tf.variable_scope('Branch_0'):
            tower_conv = slim.conv2d(net, 128, 1, scope='Conv2d_1x1')
        with tf.variable_scope('Branch_1'):
            tower_conv1_0 = slim.conv2d(net, 128, 1, scope='Conv2d_0a_1x1')
            tower_conv1_1 = slim.conv2d(tower_conv1_0, 128, [1, 7],
                                        scope='Conv2d_0b_1x7')
            tower_conv1_2 = slim.conv2d(tower_conv1_1, 128, [7, 1],
                                        scope='Conv2d_0c_7x1')
        mixed = tf.concat([tower_conv, tower_conv1_2], 3)
        up = slim.conv2d(mixed, net.get_shape()[3], 1, normalizer_fn=None,
                         activation_fn=None, scope='Conv2d_1x1')
        net += scale * up
        if activation_fn:
            net = activation_fn(net)
    return net
# Inception-Resnet-C
def block8(net, scale=1.0, activation_fn=tf.nn.relu, scope=None, reuse=None):
    """Builds the 8x8 resnet block."""
    with tf.variable_scope(scope, 'Block8', [net], reuse=reuse):
        with tf.variable_scope('Branch_0'):
            tower_conv = slim.conv2d(net, 192, 1, scope='Conv2d_1x1')
        with tf.variable_scope('Branch_1'):
            tower_conv1_0 = slim.conv2d(net, 192, 1, scope='Conv2d_0a_1x1')
            tower_conv1_1 = slim.conv2d(tower_conv1_0, 192, [1, 3],
                                        scope='Conv2d_0b_1x3')
            tower_conv1_2 = slim.conv2d(tower_conv1_1, 192, [3, 1],
                                        scope='Conv2d_0c_3x1')
        mixed = tf.concat([tower_conv, tower_conv1_2], 3)
        up = slim.conv2d(mixed, net.get_shape()[3], 1, normalizer_fn=None,
                         activation_fn=None, scope='Conv2d_1x1')
        net += scale * up
        if activation_fn:
            net = activation_fn(net)
    return net
def reduction_a(net, k, l, m, n):
    with tf.variable_scope('Branch_0'):
        tower_conv = slim.conv2d(net, n, 3, stride=2, padding='VALID',
                                 scope='Conv2d_1a_3x3')
    with tf.variable_scope('Branch_1'):
        tower_conv1_0 = slim.conv2d(net, k, 1, scope='Conv2d_0a_1x1')
        tower_conv1_1 = slim.conv2d(tower_conv1_0, l, 3,
                                    scope='Conv2d_0b_3x3')
        tower_conv1_2 = slim.conv2d(tower_conv1_1, m, 3,
                                    stride=2, padding='VALID',
                                    scope='Conv2d_1a_3x3')
    with tf.variable_scope('Branch_2'):
        tower_pool = slim.max_pool2d(net, 3, stride=2, padding='VALID',
                                     scope='MaxPool_1a_3x3')
    net = tf.concat([tower_conv, tower_conv1_2, tower_pool], 3)

    return net

def reduction_b(net):

    with tf.variable_scope('Branch_0'):

        tower_conv = slim.conv2d(net, 256, 1, scope='Conv2d_0a_1x1')

        tower_conv_1 = slim.conv2d(tower_conv, 384, 3, stride=2,

                                   padding='VALID', scope='Conv2d_1a_3x3')

    with tf.variable_scope('Branch_1'):

        tower_conv1 = slim.conv2d(net, 256, 1, scope='Conv2d_0a_1x1')

        tower_conv1_1 = slim.conv2d(tower_conv1, 256, 3, stride=2,

                                    padding='VALID', scope='Conv2d_1a_3x3')

    with tf.variable_scope('Branch_2'):

        tower_conv2 = slim.conv2d(net, 256, 1, scope='Conv2d_0a_1x1')

        tower_conv2_1 = slim.conv2d(tower_conv2, 256, 3,

                                    scope='Conv2d_0b_3x3')

        tower_conv2_2 = slim.conv2d(tower_conv2_1, 256, 3, stride=2,

                                    padding='VALID', scope='Conv2d_1a_3x3')

    with tf.variable_scope('Branch_3'):

        tower_pool = slim.max_pool2d(net, 3, stride=2, padding='VALID',

                                     scope='MaxPool_1a_3x3')

    net = tf.concat([tower_conv_1, tower_conv1_1,

                        tower_conv2_2, tower_pool], 3)

    return net

def inference(images, keep_probability, phase_train=True,

              bottleneck_layer_size=128, weight_decay=0.0, reuse=None):
```

```
    batch_norm_params = {

        # Decay for the moving averages.

        'decay': 0.995,

        # epsilon to prevent 0s in variance.

        'epsilon': 0.001,

        # force in-place updates of mean and variance estimates

        'updates_collections': None,

        # Moving averages ends up in the trainable variables collection

        'variables_collections': [ tf.GraphKeys.TRAINABLE_VARIABLES ],

    }

    with slim.arg_scope([slim.conv2d, slim.fully_connected],

                        weights_initializer=slim.initializers.xavier_initializer(),

                        weights_regularizer=slim.l2_regularizer(weight_decay),

                        normalizer_fn=slim.batch_norm,

                        normalizer_params=batch_norm_params):

        return inception_resnet_v1(images, is_training=phase_train,

                    dropout_keep_prob=keep_probability,  bottleneck_layer_size=bottleneck_layer_size,
reuse=reuse)


def inception_resnet_v1(inputs, is_training=True,

                        dropout_keep_prob=0.8,

                        bottleneck_layer_size=128,

                        reuse=None,

                        scope='InceptionResnetV1'):



    end_points = {}
```

```
with tf.variable_scope(scope, 'InceptionResnetV1', [inputs], reuse=reuse):
  with slim.arg_scope([slim.batch_norm, slim.dropout],
                      is_training=is_training):
    with slim.arg_scope([slim.conv2d, slim.max_pool2d, slim.avg_pool2d],
                        stride=1, padding='SAME'):

      # 149 x 149 x 32
      net = slim.conv2d(inputs, 32, 3, stride=2, padding='VALID',
                        scope='Conv2d_1a_3x3')
      end_points['Conv2d_1a_3x3'] = net
      # 147 x 147 x 32
      net = slim.conv2d(net, 32, 3, padding='VALID',
                        scope='Conv2d_2a_3x3')
      end_points['Conv2d_2a_3x3'] = net
      # 147 x 147 x 64
      net = slim.conv2d(net, 64, 3, scope='Conv2d_2b_3x3')
      end_points['Conv2d_2b_3x3'] = net
      # 73 x 73 x 64
      net = slim.max_pool2d(net, 3, stride=2, padding='VALID',
                            scope='MaxPool_3a_3x3')
      end_points['MaxPool_3a_3x3'] = net
      # 73 x 73 x 80
      net = slim.conv2d(net, 80, 1, padding='VALID',
                        scope='Conv2d_3b_1x1')
      end_points['Conv2d_3b_1x1'] = net
```

```
# 71 x 71 x 192
net = slim.conv2d(net, 192, 3, padding='VALID',
                  scope='Conv2d_4a_3x3')
end_points['Conv2d_4a_3x3'] = net
# 35 x 35 x 256
net = slim.conv2d(net, 256, 3, stride=2, padding='VALID',
                  scope='Conv2d_4b_3x3')
end_points['Conv2d_4b_3x3'] = net
 # 5 x Inception-resnet-A
net = slim.repeat(net, 5, block35, scale=0.17)
end_points['Mixed_5a'] = net
# Reduction-A
with tf.variable_scope('Mixed_6a'):
    net = reduction_a(net, 192, 192, 256, 384)
end_points['Mixed_6a'] = net
# 10 x Inception-Resnet-B
net = slim.repeat(net, 10, block17, scale=0.10)
end_points['Mixed_6b'] = net
# Reduction-B
with tf.variable_scope('Mixed_7a'):
    net = reduction_b(net)
end_points['Mixed_7a'] = net
# 5 x Inception-Resnet-C
net = slim.repeat(net, 5, block8, scale=0.20)
end_points['Mixed_8a'] = net
```

```
            net = block8(net, activation_fn=None)
            end_points['Mixed_8b'] = net
            with tf.variable_scope('Logits'):
                end_points['PrePool'] = net
                #pylint: disable=no-member
                net = slim.avg_pool2d(net, net.get_shape()[1:3], padding='VALID',
                                      scope='AvgPool_1a_8x8')
                net = slim.flatten(net)
                print("flatten shape:",net.shape)
                net = slim.dropout(net, dropout_keep_prob, is_training=is_training,
                                   scope='Dropout')
                end_points['PreLogitsFlatten'] = net
            net = slim.fully_connected(net, bottleneck_layer_size, activation_fn=None,
                    scope='Bottleneck', reuse=False)
    return net, end_points
```

A.2 Image Alignment Python Code

```
#Image Alignment Source Code
import numpy as np
import os, time, cv2
import matplotlib.pyplot as plt
def model_restore_from_pb(pb_path, node_dict,GPU_ratio=None):
    tf_dict = dict()
    with tf.Graph().as_default():
        config = tf.ConfigProto(log_device_placement=True,  #print out GPU or CPU is adopted
                        allow_soft_placement=True,  #allow tf to use alternative devices
                        )
        if GPU_ratio is None:
            config.gpu_options.allow_growth = True  # The program can access as much resource as
possible
        else:
            config.gpu_options.per_process_gpu_memory_fraction = GPU_ratio  # limit the GPU
resource
        sess = tf.Session(config=config)
        with gfile.FastGFile(pb_path, 'rb') as f:
            graph_def = tf.GraphDef()
            graph_def.ParseFromString(f.read())
            sess.graph.as_default()
            tf.import_graph_def(graph_def, name='')  # import the calculation graph
        sess.run(tf.global_variables_initializer())
        for key, value in node_dict.items():
            try:
                node = sess.graph.get_tensor_by_name(value)
                tf_dict[key] = node
            except:
                print("node:{} does not exist in the graph")
        return sess, tf_dict
class FaceMaskDetection():
    def __init__(self,pb_path,margin=44,GPU_ratio=0.1):
        # ----var
        node_dict = {'input': 'data_1:0',
                    'detection_bboxes': 'loc_branch_concat_1/concat:0',
                    'detection_scores': 'cls_branch_concat_1/concat:0'}
        conf_thresh = 0.8
        iou_thresh = 0.7
        # ====anchors config
        feature_map_sizes = [[33, 33], [17, 17], [9, 9], [5, 5], [3, 3]]
        anchor_sizes = [[0.04, 0.056], [0.08, 0.11], [0.16, 0.22], [0.32, 0.45], [0.64, 0.72]]
        anchor_ratios = [[1, 0.62, 0.42]] * 5
        id2class = {0: 'Mask', 1: 'NoMask'}
        # ----model init
        # ====generate anchors
        anchors = self.generate_anchors(feature_map_sizes, anchor_sizes, anchor_ratios)
        # for inference , the batch size is 1, the model output shape is [1, N, 4],
        # so we expand dim for anchors to [1, anchor_num, 4]
        anchors_exp = np.expand_dims(anchors, axis=0)
        # ====model restore from pb file
        sess, tf_dict = model_restore_from_pb(pb_path, node_dict,GPU_ratio = GPU_ratio)
        tf_input = tf_dict['input']
        model_shape = tf_input.shape  # [N,H,W,C]
        print("model_shape = ", model_shape)
        img_size = (tf_input.shape[2].value,tf_input.shape[1].value)
        detection_bboxes = tf_dict['detection_bboxes']
        detection_scores = tf_dict['detection_scores']
        # ----local var to global
        self.model_shape = model_shape
```

```
        self.img_size = img_size
        self.sess = sess
        self.tf_input = tf_input
        self.detection_bboxes = detection_bboxes
        self.detection_scores = detection_scores
        self.anchors_exp = anchors_exp
        self.conf_thresh = conf_thresh
        self.iou_thresh = iou_thresh
        self.id2class = id2class
        self.margin = margin
    def generate_anchors(self,feature_map_sizes, anchor_sizes, anchor_ratios, offset=0.5):
        '''
        generate anchors.
        :param feature_map_sizes: list of list, for example: [[40,40], [20,20]]
        :param anchor_sizes: list of list, for example: [[0.05, 0.075], [0.1, 0.15]]
        :param anchor_ratios: list of list, for example: [[1, 0.5], [1, 0.5]]
        :param offset: default to 0.5
        :return:
        '''
        anchor_bboxes = []
        for idx, feature_size in enumerate(feature_map_sizes):
            cx = (np.linspace(0, feature_size[0] - 1, feature_size[0]) + 0.5) / feature_size[0]
            cy = (np.linspace(0, feature_size[1] - 1, feature_size[1]) + 0.5) / feature_size[1]
            cx_grid, cy_grid = np.meshgrid(cx, cy)
            cx_grid_expend = np.expand_dims(cx_grid, axis=-1)
        cy_grid_expend = np.expand_dims(cy_grid, axis=-1)
        center = np.concatenate((cx_grid_expend, cy_grid_expend), axis=-1)
        num_anchors = len(anchor_sizes[idx]) + len(anchor_ratios[idx]) - 1
        center_tiled = np.tile(center, (1, 1, 2 * num_anchors))
        anchor_width_heights = []
        # different scales with the first aspect ratio
        for scale in anchor_sizes[idx]:
            ratio = anchor_ratios[idx][0]  # select the first ratio
            width = scale * np.sqrt(ratio)
            height = scale / np.sqrt(ratio)
            anchor_width_heights.extend([-width / 2.0, -height / 2.0, width / 2.0, height / 2.0])
        # the first scale, with different aspect ratios (except the first one)
        for ratio in anchor_ratios[idx][1:]:
            s1 = anchor_sizes[idx][0]  # select the first scale
            width = s1 * np.sqrt(ratio)
            height = s1 / np.sqrt(ratio)
            anchor_width_heights.extend([-width / 2.0, -height / 2.0, width / 2.0, height / 2.0])
        bbox_coords = center_tiled + np.array(anchor_width_heights)
        bbox_coords_reshape = bbox_coords.reshape((-1, 4))
        anchor_bboxes.append(bbox_coords_reshape)
    anchor_bboxes = np.concatenate(anchor_bboxes, axis=0)
    return anchor_bboxes
def decode_bbox(self,anchors, raw_outputs, variances=[0.1, 0.1, 0.2, 0.2]):
    '''
    Decode the actual bbox according to the anchors.
```

```
        the anchor value order is:[xmin,ymin, xmax, ymax]
        :param anchors: numpy array with shape [batch, num_anchors, 4]
        :param raw_outputs: numpy array with the same shape with anchors
        :param variances: list of float, default=[0.1, 0.1, 0.2, 0.2]
        :return:
        '''
        anchor_centers_x = (anchors[:, :, 0:1] + anchors[:, :, 2:3]) / 2
        anchor_centers_y = (anchors[:, :, 1:2] + anchors[:, :, 3:]) / 2
        anchors_w = anchors[:, :, 2:3] - anchors[:, :, 0:1]
        anchors_h = anchors[:, :, 3:] - anchors[:, :, 1:2]
        raw_outputs_rescale = raw_outputs * np.array(variances)
        predict_center_x = raw_outputs_rescale[:, :, 0:1] * anchors_w + anchor_centers_x
        predict_center_y = raw_outputs_rescale[:, :, 1:2] * anchors_h + anchor_centers_y
        predict_w = np.exp(raw_outputs_rescale[:, :, 2:3]) * anchors_w
        predict_h = np.exp(raw_outputs_rescale[:, :, 3:]) * anchors_h
        predict_xmin = predict_center_x - predict_w / 2
        predict_ymin = predict_center_y - predict_h / 2
        predict_xmax = predict_center_x + predict_w / 2
        predict_ymax = predict_center_y + predict_h / 2
        predict_bbox = np.concatenate([predict_xmin, predict_ymin, predict_xmax, predict_ymax],
axis=-1)
        return predict_bbox
    def single_class_non_max_suppression(self,bboxes, confidences, conf_thresh=0.2,
iou_thresh=0.5, keep_top_k=-1):
        '''
        do nms on single class.
        Hint: for the specific class, given the bbox and its confidence,

        1) sort the bbox according to the confidence from top to down, we call this a set

        2) select the bbox with the highest confidence, remove it from set, and do IOU calculate
with the rest bbox

        3) remove the bbox whose IOU is higher than the iou_thresh from the set,

        4) loop step 2 and 3, util the set is empty.

        :param bboxes: numpy array of 2D, [num_bboxes, 4]

        :param confidences: numpy array of 1D. [num_bboxes]

        :param conf_thresh:

        :param iou_thresh:

        :param keep_top_k:

        :return:

        '''

        if len(bboxes) == 0: return []

        conf_keep_idx = np.where(confidences > conf_thresh)[0]

        bboxes = bboxes[conf_keep_idx]

        confidences = confidences[conf_keep_idx]

        pick = []

        xmin = bboxes[:, 0]

        ymin = bboxes[:, 1]

        xmax = bboxes[:, 2]

        ymax = bboxes[:, 3]

        area = (xmax - xmin + 1e-3) * (ymax - ymin + 1e-3)

        idxs = np.argsort(confidences)

        while len(idxs) > 0:

            last = len(idxs) - 1
```

```
            i = idxs[last]

            pick.append(i)

            # keep top k

            if keep_top_k != -1:

                if len(pick) >= keep_top_k:

                    break

            overlap_xmin = np.maximum(xmin[i], xmin[idxs[:last]])

            overlap_ymin = np.maximum(ymin[i], ymin[idxs[:last]])

            overlap_xmax = np.minimum(xmax[i], xmax[idxs[:last]])

            overlap_ymax = np.minimum(ymax[i], ymax[idxs[:last]])

            overlap_w = np.maximum(0, overlap_xmax - overlap_xmin)

            overlap_h = np.maximum(0, overlap_ymax - overlap_ymin)

            overlap_area = overlap_w * overlap_h

            overlap_ratio = overlap_area / (area[idxs[:last]] + area[i] - overlap_area)

            need_to_be_deleted_idx = np.concatenate(([last], np.where(overlap_ratio >
iou_thresh)[0]))

            idxs = np.delete(idxs, need_to_be_deleted_idx)


        # if the number of final bboxes is less than keep_top_k, we need to pad it.

        # TODO

        return conf_keep_idx[pick]

    def inference(self,img_4d,ori_height,ori_width):

        # ----var

        re_boxes = list()

        re_confidence = list()

        re_classes = list()
        re_mask_id = list()

        y_bboxes_output, y_cls_output = self.sess.run([self.detection_bboxes,
self.detection_scores],

                                                      feed_dict={self.tf_input: img_4d})

        # remove the batch dimension, for batch is always 1 for inference.

        y_bboxes = self.decode_bbox(self.anchors_exp, y_bboxes_output)[0]

        y_cls = y_cls_output[0]

        # To speed up, do single class NMS, not multiple classes NMS.

        bbox_max_scores = np.max(y_cls, axis=1)

        bbox_max_score_classes = np.argmax(y_cls, axis=1)

        # keep_idx is the alive bounding box after nms.

        keep_idxs = self.single_class_non_max_suppression(y_bboxes,
bbox_max_scores,  conf_thresh=self.conf_thresh,

                                                          iou_thresh=self.iou_thresh )

        # ====draw bounding box

        for idx in keep_idxs:

            conf = float(bbox_max_scores[idx])

            #print("conf = ",conf)

            class_id = bbox_max_score_classes[idx]

            bbox = y_bboxes[idx]

            #print(bbox)

            xmin = np.maximum(0, int(bbox[0] * ori_width - self.margin / 2))

            ymin = np.maximum(0, int(bbox[1] * ori_height - self.margin / 2))

            xmax = np.minimum(int(bbox[2] * ori_width + self.margin / 2), ori_width)

            ymax = np.minimum(int(bbox[3] * ori_height + self.margin / 2), ori_height)

            re_boxes.append([xmin, ymin, xmax - xmin, ymax - ymin])
            re_confidence.append(conf)

            re_classes.append('face')

            re_mask_id.append(class_id)

        return re_boxes, re_confidence, re_classes, re_mask_id

def img_alignment(root_dir,output_dir,margin=44,GPU_ratio =
0.1,img_show=False,dataset_range=None):

    # ----record the start time

    d_t = time.time()

    # ----var

    face_mask_model_path = r'face_mask_detection.pb'

    img_format = {'png','bmp','jpg'}

    width_threshold = 100 + margin // 2

    height_threshold = 100 + margin // 2

    quantity = 0

    # ----collect all folders

    dirs = [obj.path for obj in os.scandir(root_dir) if obj.is_dir()]

    if len(dirs) == 0:

        print("No sub folders in ",root_dir)

    else:

        dirs.sort()

        print("Total class number: ", len(dirs))

        if dataset_range is not None:

            dirs = dirs[dataset_range[0]:dataset_range[1]]

            print("Working classes: {} to {}".format(dataset_range[0], dataset_range[1]))

        else:

            print("Working classes:All")
```

```
#----init of face detection model
fmd = FaceMaskDetection(face_mask_model_path,margin,GPU_ratio)
# ----handle images of each dir
for dir_path in dirs:
    paths = [file.path for file in os.scandir(dir_path) if file.name.split(".")[-1] in img_format]
    if len(paths) == 0:
        print("No images in ",dir_path)
    else:
        #----create the save dir
        save_dir = os.path.join(output_dir,dir_path.split("\\")[-1])
        if not os.path.exists(save_dir):
            os.makedirs(save_dir)
        #----
        quantity += len(paths)
        for idx,path in enumerate(paths):
            img = cv2.imread(path)
            if img is None:
                print("Read failed:",path)
            else:
                ori_height,ori_width = img.shape[:2]
                img_ori = img.copy()
                img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
                img = cv2.resize(img,fmd.img_size)
                img = img.astype(np.float32)
                img /= 255
```

```
                        img_4d = np.expand_dims(img,axis=0)
                        bboxes, re_confidence, re_classes, re_mask_id =
fmd.inference(img_4d,ori_height,ori_width)


                        for num,bbox in enumerate(bboxes):
                            if bbox[2] > width_threshold and bbox[3] > height_threshold:
                                img_crop = img_ori[bbox[1]:bbox[1] + bbox[3],bbox[0]:bbox[0] + bbox[2],
:]
                                save_path = os.path.join(save_dir,str(idx) + '_' + str(num) + ".png")
                                # print("save_path:",save_path)
                                cv2.imwrite(save_path,img_crop)
                                #----display images
                                if img_show is True:
                                    plt.subplot(1,2,1)
                                    plt.imshow(img_ori[:,:,::-1])
                                    plt.subplot(1,2,2)
                                    plt.imshow(img_crop[:,:,::-1])
                                    plt.show()
    # ----statistics(to know the average process time of each image)
    if quantity != 0:
        d_t = time.time() - d_t
        print("ave process time of each image:", d_t / quantity)
```

A.3 Image Processing and Augmentation Python Code

```
#Image Process and augmentation
```

```
import numpy as np

import os,cv2,dlib,sys

import matplotlib.pyplot as plt

mask_img_dir = r"C:\Users\saroj.mishra\mask_img"

# ----read mask png images

mask_files = [file.path for file in os.scandir(mask_img_dir) if file.name.split(".")[-1] == 'png']

mask_paths = list()

len_mask = len(mask_files)

if len_mask == 0:

    print("Error: no face mask PNG images in  ", mask_img_dir)

# ----face detection init

detector = dlib.get_frontal_face_detector()

predictor =
dlib.shape_predictor(r'C:\Users\saroj.mishra\shape_predictor_68_face_landmarks.dat')

# ----Detect mouth coordinates

def detect_mouth(img,detector,predictor):

    x_min = None

    x_max = None

    y_min = None

    y_max = None

    size = None

    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    faces = detector(img, 0)

    #print("len of faces = ",len(faces))

    if len(faces):

        for coor in (faces):#coordinate format:[(left,top), (right,bottom)]
```

```
            x = list()

            y = list()

            height = coor.bottom() - coor.top()

            width = coor.right() - coor.left()

            # shape = predictor(img_gray, d)

            landmark = predictor(img, coor)

            #----get the mouth part

            for i in range(48, 68):

                x.append(landmark.part(i).x)

                y.append(landmark.part(i).y)

            y_max = np.minimum(max(y) + height // 3, img.shape[0])

            y_min = np.maximum(min(y) - height // 3, 0)

            x_max = np.minimum(max(x) + width // 3, img.shape[1])

            x_min = np.maximum(min(x) - width // 3, 0)

            size = ((x_max-x_min),(y_max-y_min))#(width,height)

        return x_min, x_max, y_min, y_max, size

root_dir = r"C:\Users\saroj.mishra\CASIA-WebFace-aligned"

paths = list()

img_format = {'png', 'jpg'}

for dir_name, sub_dirname, filenames in os.walk(root_dir):

    if len(filenames):

        for filename in filenames:

            if filename[-3:] in img_format:

                paths.append(os.path.join(dir_name,filename))
```

```
def get_4D_data(paths,img_shape,process_dict=None):
    #----var
    re_array = []
    processing_enable = False
    x_range = 10
    y_range = 20
    flip_list = [1, 0]
    kernel_list = [1,3,5,7]
    #----check process_dict
    if isinstance(process_dict,dict):
        if len(process_dict) > 0:
            processing_enable = True#image processing is enabled
    for path in paths:
        img = cv2.imread(path)
        if img is None:
            print("read failed:",path)
        else:
            #----image processing
            if processing_enable is True:
                if 'rdm_crop' in process_dict.keys():
                    if process_dict['rdm_crop'] is True:
#img = cv2.resize(img,(width_rdm_crop,height_rdm_crop))
                        # ----Find a random point
                        x_start = np.random.randint(x_range)
                        y_start = np.random.randint(y_range)
            # ----From the random point, crop the image
            img = img[y_start:, x_start:, :]
        if 'rdm_br' in process_dict.keys():
          if process_dict['rdm_br'] is True:
            mean_br = np.mean(img)
            br_factor = np.random.randint(mean_br * 0.7, mean_br * 1.3)
            img = np.clip(img / mean_br * br_factor, 0, 255)#the multification makes the
numeric type become floating
            img = img.astype(np.uint8)#transform the numeric type to unsigned integer
8(UINT8)
        if 'rdm_mask' in process_dict.keys():
          if process_dict['rdm_mask'] is True:
            x_min, x_max, y_min, y_max, size = detect_mouth(img, detector, predictor)
            if size is not None:
              # ----random selection of face mask
              which = np.random.randint(0, len_mask - 1)
              #print(which)
              item_name = mask_files[which]
              # ----face mask process
              item_img = cv2.imread(item_name, cv2.IMREAD_UNCHANGED)
              #item_img = mask_paths[which]
              print(item_img.shape)
              item_img = cv2.resize(item_img, size)
              item_img_rgb = item_img[:, :, :3]
              #item_img_rgb = item_img_rgb[:,:,::-1]#transform the color format to RGB
```

```
                    item_alpha_ch = item_img[:, :, 3]
                    _, item_mask = cv2.threshold(item_alpha_ch, 220, 255,
cv2.THRESH_BINARY)
                    img_item = cv2.bitwise_and(item_img_rgb, item_img_rgb, mask=item_mask)
                    # ----mouth part process
                    roi = img[y_min:y_min + size[1], x_min:x_min + size[0]]
                    item_mask_inv = cv2.bitwise_not(item_mask)
                    roi = cv2.bitwise_and(roi, roi, mask=item_mask_inv)
                    # ----addition of mouth and face mask
                    dst = cv2.add(roi, img_item)
                    img[y_min: y_min + size[1], x_min:x_min + size[0]] = dst
            if 'rdm_blur' in process_dict.keys():
                if process_dict['rdm_blur'] is True:
                    kernel = tuple(np.random.choice(kernel_list,size=2))
                    print("kernel:",kernel)
                    img = cv2.GaussianBlur(img,kernel,0,0)
            if 'rdm_flip' in process_dict.keys():
                if process_dict['rdm_flip'] is True:
                    flip_type = np.random.choice(flip_list)
                    if flip_type == 1:
                        img = cv2.flip(img, flip_type)
            if 'rdm_noise' in process_dict.keys():
                if process_dict['rdm_noise'] is True:
                    uniform_noise = np.empty((img.shape[0], img.shape[1]), dtype=np.uint8)
                    cv2.randu(uniform_noise, 0, 255)
                    ret, impulse_noise = cv2.threshold(uniform_noise, 240, 255,
cv2.THRESH_BINARY_INV)
                    img = cv2.bitwise_and(img, img, mask=impulse_noise)
            if 'rdm_angle' in process_dict.keys():
                if process_dict['rdm_angle'] is True:
                    angle = np.random.randint(-15, 15)
                    img = cv2.resize(img,(img_shape[1],img_shape[0]))
                    print(img.shape)
                    h, w = img.shape[:2]
                    M = cv2.getRotationMatrix2D((w // 2, h // 2), angle, 1.0)
                    img = cv2.warpAffine(img, M, (h, w))
        #----
        img = cv2.resize(img,(img_shape[1],img_shape[0]))
        img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
        img = img.astype(np.float32)
        img /= 255
        re_array.append(img)
re_array = np.array(re_array)
return re_array
aug_times = 4
path = [np.random.choice(paths)]
img_shape = [112,112,3]
batch_data_shape = [aug_times]
batch_data_shape.extend(img_shape)
batch_data = np.zeros(batch_data_shape,dtype=np.float32)
```

```
p_dict_1 =
{'rdm_mask':False,'rdm_crop':True,'rdm_br':True,'rdm_blur':True,'rdm_flip':True,'rdm_noise':Fal
se,'rdm_angle':True}

p_dict_2 =
{'rdm_mask':True,'rdm_crop':True,'rdm_br':True,'rdm_blur':True,'rdm_flip':True,'rdm_noise':Fal
se,'rdm_angle':True}

p_dict_3 =
{'rdm_mask':True,'rdm_crop':True,'rdm_br':True,'rdm_blur':True,'rdm_flip':True,'rdm_noise':Fal
se,'rdm_angle':True}



for i in range(aug_times):

    if i == 0:

        temp = get_4D_data(path,img_shape,process_dict=None)

    elif i == 1:

        temp = get_4D_data(path,img_shape,process_dict=p_dict_1)

    elif i == 2:

        temp = get_4D_data(path,img_shape,process_dict=p_dict_2)

    elif i == 3:

        temp = get_4D_data(path,img_shape,process_dict=p_dict_3)

    batch_data[i] = temp[0]

plt.figure(figsize=(15,15))

for i in range(aug_times):

    plt.subplot(1,aug_times,i+1)

    plt.imshow(batch_data[i])

    plt.axis('off')

plt.show()
```

A.4 Images Processing and Augmentation Results

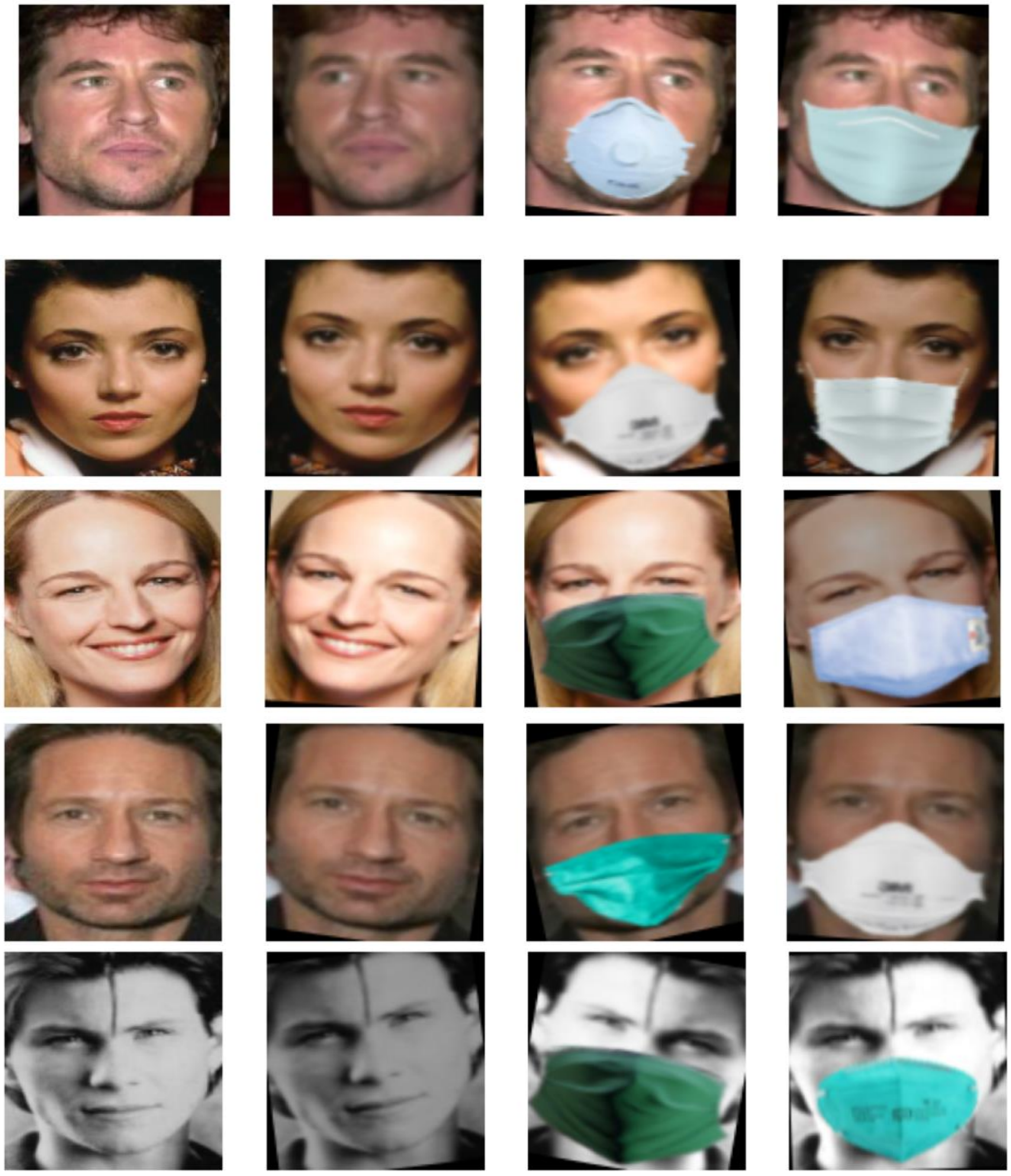


Figure A.4 Image processing and augmentation results

A.5 Real Time Masked Face Recognition Python Code

```
import cv2, os, time, math

import numpy as np

from face_alignment import FaceMaskDetection

from tools import model_restore_from_pb

img_format = {'png','jpg','bmp'}

def video_init(camera_source=0,resolution="480",to_write=False,save_dir=None):

    #----var

    writer = None

    resolution_dict = {"480":[480,640],"720":[720,1280],"1080":[1080,1920]}

    #----camera source connection

    cap = cv2.VideoCapture(camera_source)

    #----resolution decision

    if resolution_dict.get(resolution) is not None:

    # if resolution in resolution_dict.keys():

        width = resolution_dict[resolution][1]

        height = resolution_dict[resolution][0]

        cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)

        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)

    else:

        height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)#default 480

        width = cap.get(cv2.CAP_PROP_FRAME_WIDTH)#default 640

        print("video size is auto set")

    if to_write is True:

        #fourcc = cv2.VideoWriter_fourcc('x', 'v', 'i', 'd')
        #fourcc = cv2.VideoWriter_fourcc('X', 'V', 'I', 'D')
        fourcc = cv2.VideoWriter_fourcc(*'XVID')
        save_path = 'demo.avi'
        if save_dir is not None:
            save_path = os.path.join(save_dir,save_path)
        writer = cv2.VideoWriter(save_path, fourcc, 30, (int(width), int(height)))
    return cap,height,width,writer
def stream(pb_path,
node_dict,ref_dir,camera_source=0,resolution="480",to_write=False,save_dir=None):
    #----var
    frame_count = 0
    FPS = "loading"
    face_mask_model_path = r'face_mask_detection.pb'
    margin = 40
    id2class = {0: 'Mask', 1: 'NoMask'}
    batch_size = 32
    threshold = 0.8
    #----Video streaming initialization
    cap,height,width,writer = video_init(camera_source=camera_source, resolution=resolution,
to_write=to_write, save_dir=save_dir)
    # ----face detection init
    fmd = FaceMaskDetection(face_mask_model_path, margin, GPU_ratio=None)
    # ----face recognition init
    sess, tf_dict = model_restore_from_pb(pb_path, node_dict, GPU_ratio=None)
    tf_input = tf_dict['input']
    tf_embeddings = tf_dict['embeddings']
```

```
#----get the model shape
if tf_input.shape[1].value is None:
    model_shape = (None, 160, 160, 3)
else:
    model_shape = (None, tf_input.shape[1].value, tf_input.shape[2].value, 3)
print("The mode shape of face recognition:",model_shape)
#----set the feed_dict
feed_dict = dict()
if 'keep_prob' in tf_dict.keys():
    tf_keep_prob = tf_dict['keep_prob']
    feed_dict[tf_keep_prob] = 1.0
if 'phase_train' in tf_dict.keys():
    tf_phase_train = tf_dict['phase_train']
    feed_dict[tf_phase_train] = False
#----read images from the database
d_t = time.time()
paths = [file.path for file in os.scandir(ref_dir) if file.name[-3:] in img_format]
len_ref_path = len(paths)
if len_ref_path == 0:
    print("No images in ", ref_dir)
else:
    ites = math.ceil(len_ref_path / batch_size)
    embeddings_ref = np.zeros([len_ref_path, tf_embeddings.shape[-1]], dtype=np.float32)
    for i in range(ites):
        num_start = i * batch_size
        num_end = np.minimum(num_start + batch_size, len_ref_path)
        batch_data_dim =[num_end - num_start]
        batch_data_dim.extend(model_shape[1:])
        batch_data = np.zeros(batch_data_dim,dtype=np.float32)
        for idx,path in enumerate(paths[num_start:num_end]):
            # img = cv2.imread(path)
            img = cv2.imdecode(np.fromfile(path, dtype=np.uint8), 1)
            if img is None:
                print("read failed:",path)
            else:
                #print("model_shape:",model_shape[1:3])
                img = cv2.resize(img,(model_shape[2],model_shape[1]))
                img = img[:,:,::-1]#change the color format
                batch_data[idx] = img
        batch_data /= 255
        feed_dict[tf_input] = batch_data
        embeddings_ref[num_start:num_end] = sess.run(tf_embeddings,feed_dict=feed_dict)
    d_t = time.time() - d_t
    print("ref embedding shape",embeddings_ref.shape)
    print("It takes {} secs to get {} embeddings".format(d_t, len_ref_path))
# ----tf setting for calculating distance
if len_ref_path > 0:
    with tf.Graph().as_default():
        tf_tar = tf.placeholder(dtype=tf.float32, shape=tf_embeddings.shape[-1])
        tf_ref = tf.placeholder(dtype=tf.float32, shape=tf_embeddings.shape)
```

```
            tf_dis = tf.sqrt(tf.reduce_sum(tf.square(tf.subtract(tf_ref, tf_tar)), axis=1))

            # ----GPU setting

            config = tf.ConfigProto(log_device_placement=True,

                                    allow_soft_placement=True,

                                    )

            config.gpu_options.allow_growth = True

            sess_cal = tf.Session(config=config)

            sess_cal.run(tf.global_variables_initializer())

        feed_dict_2 = {tf_ref: embeddings_ref}

    #----Get an image

    while(cap.isOpened()):

        ret, img = cap.read()#img is the original image with BGR format. It's used to be shown by
opencv

        if ret is True:

            #----image processing

            img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

            img_rgb = img_rgb.astype(np.float32)

            img_rgb /= 255

            #----face detection

            img_fd = cv2.resize(img_rgb, fmd.img_size)

            img_fd = np.expand_dims(img_fd, axis=0)

            bboxes, re_confidence, re_classes, re_mask_id = fmd.inference(img_fd, height, width)

            if len(bboxes) > 0:

                for num, bbox in enumerate(bboxes):

                    class_id = re_mask_id[num]

                    if class_id == 0:
```

```
                color = (0, 255, 0)  # (B,G,R) --> Green(with masks)
            else:
                color = (0, 0, 255)  # (B,G,R) --> Red(without masks)
            cv2.rectangle(img, (bbox[0], bbox[1]), (bbox[0] + bbox[2], bbox[1] + bbox[3]),
color, 2)
            # cv2.putText(img, "%s: %.2f" % (id2class[class_id], re_confidence[num]),
(bbox[0] + 2, bbox[1] - 2),
            #             cv2.FONT_HERSHEY_SIMPLEX, 0.8, color)
            # ----face recognition
            name = ""
            if len_ref_path > 0:
                img_fr = img_rgb[bbox[1]:bbox[1] + bbox[3], bbox[0]:bbox[0] + bbox[2], :]  #
crop
                img_fr = cv2.resize(img_fr, (model_shape[2], model_shape[1]))  # resize
                img_fr = np.expand_dims(img_fr, axis=0)  # make 4 dimensions
                feed_dict[tf_input] = img_fr
                embeddings_tar = sess.run(tf_embeddings, feed_dict=feed_dict)
                feed_dict_2[tf_tar] = embeddings_tar[0]
                distance = sess_cal.run(tf_dis, feed_dict=feed_dict_2)
                arg = np.argmin(distance)  # index of the smallest distance
                if distance[arg] < threshold:
                    name = paths[arg].split("\\")[-1].split(".")[0]
            cv2.putText(img, "{},{}".format(id2class[class_id], name), (bbox[0] + 2, bbox[1] -
2),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.8, color)
        #----FPS calculation
        if frame_count == 0:
```

```
        t_start = time.time()
    frame_count += 1
    if frame_count >= 10:
        FPS = "FPS=%1f" % (10 / (time.time() - t_start))
        frame_count = 0
    # cv2.putText(img, text, coor, font, size, color, line thickness, line type)
    cv2.putText(img, FPS, (10, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 3)

    #----image display
    cv2.imshow("Demo by Saroj", img)
    #----image writing
    if writer is not None:
        writer.write(img)
    #----keys handle
    key = cv2.waitKey(1) & 0xFF
    if key == ord('q'):
        break
    elif key == ord('s'):
        if len(bboxes) > 0:
            img_temp = img[bbox[1]:bbox[1] + bbox[3], bbox[0]:bbox[0] + bbox[2], :]
            save_path = "img_crop.jpg"
            save_path = os.path.join(ref_dir,save_path)
            cv2.imwrite(save_path,img_temp)
            print("An image is saved to ",save_path)
else:
```

```
            print("get images failed")
            break
    #----release
    cap.release()
    cv2.destroyAllWindows()
    if writer is not None:
        writer.release()
```

Results of real-time MFR (Masked Facial Recognition) are shown in figure 5.7.

REFERENCES

[1] Ullah, Naeem, et al. "A novel DeepMaskNet model for face mask detection and masked facial recognition." Journal of King Saud University-Computer and Information Sciences (2022).

[2] Mason, Karl, Jim Duggan, and Enda Howley. "A multi-objective neural network trained with differential evolution for dynamic economic emission dispatch." International Journal of Electrical Power & Energy Systems 100 (2018): 201-221.

[3] Kumar, A. Pavan, V. Kamakoti, and Sukhendu Das. "An Architecture for Real Time Face Recognition Using WMPCA." ICVGIP. 2004.

[4] Soyata, Tolga, et al. "Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture." 2012 IEEE symposium on computers and communications (ISCC). IEEE, 2012.

[5] Kasar, Manisha M., Debnath Bhattacharyya, and T. H. Kim. "Face recognition using neural network: a review." International Journal of Security and Its Applications 10.3 (2016): 81-100.

[6] Agagu, T. T., and B. A. Akinnuwesi. "Automated students' attendance taking in tertiary institution using hybridized facial recognition algorithm." Journal of Computer Science and Its Application 19.2 (2012): 1-13.

[7] de Leeuw, Karl Maria Michael, and Jan Bergstra, eds. The history of information security: a comprehensive handbook. Elsevier, 2007.

[8] Dlib: https://github.com/davisking/dlib

[9] OpenCV: https://github.com/opencv/opencv

[10] Liu, Wei, et al. "Ssd: Single shot multibox detector." European conference on computer vision. Springer, Cham, 2016.

[11] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[12] Anwar, Aqeel, and Arijit Raychowdhury. "Masked face recognition for secure authentication." arXiv preprint arXiv:2008.11104 (2020).

[13] Mandal, Bishwas, Adaeze Okeukwu, and Yihong Theis. "Masked face recognition using resnet-50." arXiv preprint arXiv:2104.08997 (2021).

[14] Mundial, Imran Qayyum, et al. "Towards facial recognition problem in COVID-19 pandemic." 2020 4rd International Conference on Electrical, Telecommunication and Computer Engineering (ELTICOM). IEEE, 2020.

[15] Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[16] Ejaz, Md Sabbir, et al. "Implementation of principal component analysis on masked and non-masked face recognition." 2019 1st international conference on advances in science, engineering and robotics technology (ICASERT). IEEE, 2019.

[17] CASIA dataset: https://github.com/SamYuen101234/Masked_Face_Recognition

[18] LFW dataset: http://vis-www.cs.umass.edu/lfw/

[19] Alzu’bi, Ahmad, et al. "Masked Face Recognition Using Deep Learning: A Review." Electronics 10.21 (2021): 2666.

[20] Szegedy, Christian, et al. "Inception-v4, inception-resnet and the impact of residual connections on learning." Thirty-first AAAI conference on artificial intelligence. 2017.

[21] Rath, Subrat Kumar, and Siddharth Swarup Rautaray. "A survey on face detection and recognition techniques in different application domain." International Journal of Modern Education and Computer Science 6.8 (2014): 34.

[22] Szegedy, Christian, et al. "Rethinking the inception architecture for computer vision." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[23] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[24] Sun, Yi, Xiaogang Wang, and Xiaoou Tang. "Deeply learned face representations are sparse, selective, and robust." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[25] Wang, Zhongyuan, et al. "Masked face recognition dataset and application." arXiv preprint arXiv:2003.09093 (2020).

[26] Golwalkar, Rucha, and Ninad Mehendale. "Masked-face recognition using deep metric learning and FaceMaskNet-21." Applied Intelligence (2022): 1-12.

[27] S-H Yooa, S-K Oha, Witold Pedrycz," Optimized face recognition algorithm using radial basis function neural networks and its practical applications", International journal on Neural Networks, volume 69, (2015), pp. 111-125.

[28] N Jindal, V Kumar," Enhanced Face Recognition Algorithm using PCA with Artificial Neural Networks", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 6, (2013),pp. 864-872.

[29] FaceNet Pretrained Model: https://github.com/davidsandberg/facenet

[30] Kaur, Paramjit, et al. "Facial-recognition algorithms: A literature review." Medicine, Science and the Law 60.2 (2020): 131-139.

[31] Vu, Hoai Nam, Mai Huong Nguyen, and Cuong Pham. "Masked face recognition with convolutional neural networks and local binary patterns." Applied Intelligence 52.5 (2022): 5497-5512.