SMART ATTENDANCE SYSTEM WITH IOT BASED FACE RECOGNITION USING DEEP LEARNING APPROACH

SEAH YOU

UNIVERSITI TUNKU ABDUL RAHMAN

SMART ATTENDANCE SYSTEM WITH IOT BASED FACE RECOGNITION USING DEEP LEARNING APPROACH

SEAH YOU

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Mechatronics Engineering with Honours

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

May 2023

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature	:	[illegible]
Name	:	Seah You
ID No.	:	18UEB03927
Date	:	30th April 2023

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"SMART ATTENDANCE SYSTEM WITH IOT BASED FACE RECOGNITION USING DEEP LEARNING APPROACH"** was prepared by **SEAH YOU** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Mechatronics Engineering with Honours at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	[illegible]
Supervisor	:	Kwan Ban Hoe
Date	:	19 May 2023

Signature	:	[illegible]
Co-Supervisor	:	Ng Oon-Ee
Date	:	21 May 2023

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Dr Kwan Ban Hoe and Dr Ng Oon-Ee for their invaluable advice, guidance and enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my loving parents and friends who had helped and given me encouragement to complete my prototypes and reports when I faced obstacles.

ABSTRACT

Attendance management system is an indispensable practice in which every institution or organisation adopts to mark the attendance of their employees or members. The manual process of marking attendance by using a paper-based or file-based system is riddled with flaws such as the risk of information loss, falsification or disasters. The current norm delineates the deployment of smart attendance system using RFID tags, fingerprints, iris scans, voice recognition, etc. Nowadays, technological developments propagate the practical utilisation of face recognition approach for a more efficient attendance management system. The face recognition-based attendance system is convenient with additional advantages that it can avoid human intervention and thus assisting to control the spread of viruses. In this project, a real-time attendance management system that employs face recognition approach is proposed to recognize individuals. Two face recognition models were developed: the first model used Deep Neural Network (DNN) for face detection, FaceNet for feature extraction, and Support Vector Machine (SVM) for face classification, and the second model utilised Convolutional Neural Network, specifically the trained VGG16 model, with the ImageNet dataset as its pretrained weights. Transfer learning was employed to apply the pretrained network for recognizing faces. The proposed systems' effectiveness was demonstrated through a comparison of both face recognition models, and the first model with testing accuracy of 97.62 % was integrated into a designated graphical user interface (GUI). In conclusion, the project's aim and objectives were successfully accomplished, which included developing a facial recognition system designed specifically for attendance tracking and conducting a literature review covering current approaches and results in facial recognition algorithms. Furthermore, the GUI with essential features such as creating new databases, face recognition, and attendance monitoring for users was developed to ease attendance monitoring for end-users. The system's performance and usability were analyzed to provide insights for future enhancements.

TABLE OF CONTENTS

DECLARATION		**i**
APPROVAL FOR SUBMISSION		**ii**
ACKNOWLEDGEMENTS		**iv**
ABSTRACT		**v**
TABLE OF CONTENTS		**vi**
LIST OF TABLES		**ix**
LIST OF FIGURES		**x**
LIST OF SYMBOLS / ABBREVIATIONS		**xii**
LIST OF APPENDICES		**xiv**
CHAPTER		
1	**INTRODUCTION**	**1**
	1.1 General Introduction	1
	1.2 Importance of the Study	3
	1.3 Problem Statement	4
	1.4 Aim and Objectives	5
	1.5 Scope and Limitation of the Study	5
	1.6 Contribution of the Study	6
	1.7 Outline of the Report	6
2	**LITERATURE REVIEW**	**7**
	2.1 Smart Attendance System	7
	2.2 Face Detection	8
	2.3 Face Recognition	11
	2.3.1 Multi-level Face Recognition Taxonomy	11
	2.3.2 Face Structure	12
	2.3.3 Feature Extraction	14
	2.3.4 Local Approach	17
	2.3.5 Holistic Approach	20
	2.3.6 Learning-based Approach	23

		2.3.7 Transfer Learning	29
	2.4	Evaluation Metrics	29
	2.5	Database	31
	2.6	Comparison between Traditional Approaches and Deep Learning Approaches	33
3	**METHODOLOGY AND WORK PLAN**		**37**
	3.1	Introduction	37
	3.2	Project Planning	37
	3.3	Project Development	38
	3.4	Resources	42
		3.4.1 Hardware	42
		3.4.2 Software	43
	3.5	First Approach	43
		3.5.1 Face Detection	43
		3.5.2 Feature Extraction	44
		3.5.3 Face Classification	45
	3.6	Second Approach	46
		3.6.1 Model Architecture	47
		3.6.2 Data Augmentation	47
		3.6.3 Training	47
	3.7	Attendance Update	48
	3.8	User Interface Design	48
	3.9	Problems and Solutions	48
4	**RESULTS AND DISCUSSION**		**51**
	4.1	Introduction	51
	4.2	Face Detection Evaluation	51
		4.2.1 Labeled Faces In The Wild	51
		4.2.2 Performance Evaluation	52
	4.3	Face Recognition Evaluation	55
		4.3.1 Face Datasets	56
		4.3.2 Parameters	57
		4.3.3 Evaluation Metrics	58
	4.4	Smart Attendance System	64
		4.4.1 Graphical User Interface	64

4.5 Summary 68

5 CONCLUSIONS AND RECOMMENDATIONS 69

5.1 Conclusions 69

5.2 Recommendations for Future Work 70

REFERENCES 72

APPENDICES 82

LIST OF TABLES

Table 2.1: Comparison of Different Approaches for Attendance System. 8

Table 2.2: Comparison Between CNN, SSD and YOLO (Srivastava et al., 2021). 28

Table 2.3: Performance Comparison for SVM and Deep learning. 34

Table 2.4: Comparison Between Conventional and Deep Learning Approach. 36

Table 3.1: Specifications of Benewy Full HD Webcam. 42

Table 4.1: Confusion Matrix for the Performance Evaluation of Face Detectors. 52

Table 4.2: Face Detection Approaches on Image Datasets. 53

Table 4.3: Comparison of Accuracy and Speed on LFW datasets. 54

Table 4.4: Face Detection Approaches on Webcam for Frontal Faces. 54

Table 4.5: Face Detection Approaches on Webcam for Side Faces. 54

Table 4.6: Face Detection Approaches on Webcam in Low-light Environment. 55

Table 4.7: Evaluation Comparison between Two Models. 59

Table 4.8: Test Accuracy On Different Classifiers. 59

Table 4.9: Classification Report for DNN, FaceNet and SVM. 62

Table 4.10: Classification Report for VGG16 Transfer Learning Model. 63

LIST OF FIGURES

Figure 2.1: Testing Results on Each Face Detection Algorith (Arora, Naithani and Areeckal, 2022). 10

Figure 2.2: Existing Multi-level Face Recognition Taxonomy (Sepas- Moghaddam, Pereira and Correia, 2020). 12

Figure 2.3: Global Representation (Sepas-Moghaddam, Pereira and Correia, 2020). 13

Figure 2.4: Component and Structure Representation (Sepas-Moghaddam, Pereira and Correia, 2020). 13

Figure 2.5: Component Representation (Sepas-Moghaddam, Pereira and Correia, 2020). 13

Figure 2.6: Local Spatial Support (Sepas-Moghaddam, Pereira and Correia, 2020). 14

Figure 2.7: FaceNet Model Structure (Schroff and Philbin, n.d.). 15

Figure 2.8: Comparison Between Conventional and Deep Learning Approach on RMFRD Dataset (Bharat Chandra and Karthikeya Reddy, 2020). 16

Figure 2.9: LBP Calculation (Chang-yeon, 2008). 18

Figure 2.10: Comparison of Performances for HOG Approach and LBP Approach (Amraee, Chinipardaz and Charoosaei, 2022). 20

Figure 2.11: Recognition Rate for PCA and KPCA Approaches (Anon., 2014). 23

Figure 2.12: Comparison Graph of Different CNN Models (Gwyn, Roy and Atay, 2021). 24

Figure 2.13: Framework of SSD (Shi, Bao and Tan, 2019). 25

Figure 2.14: YOLO Algorithm Based on DarkNet Architecture (Weng, 2018). 26

Figure 2.15: Comparison of Different Versions of YOLO Model (Francesco and Solawetz, 2023). 28

Figure 2.16: ROC and Inverted Precision-Recall Curves (Nguyen Meidine et.al., 2018). 30

Figure 2.17: Performance of Various Techniques on ORL Database (Singhal et al., 2021). 35

Figure 3.1: Project Gantt Chart. 38

Figure 3.2: Workflow for Face Recognition Approaches. 39

Figure 3.3: Block Diagram for Smart Attendance System. 40

Figure 3.4: Flowchart of Smart Attendance System. 41

Figure 3.5: Asus VivoBook S15 S510U Laptop. 43

Figure 3.6: VGG16 Model Architecture (Bansal, 2020). 46

Figure 4.1: Transfer Learning VGG16 Model Accuracy at Each Epoch. 60

Figure 4.2: Transfer Learning VGG16 Model Loss at Each Epoch. 61

Figure 4.3: Confusion Matrix for DNN, FaceNet and SVM. 64

Figure 4.4: Confusion Matrix for VGG16. 64

LIST OF SYMBOLS / ABBREVIATIONS

3D	three-dimensional
ANN	Artificial Neural Network
CMC	cumulative match characteristic
CNN	Convolution Neural Network
COVID	corona-virus disease
CPU	Central Processing Unit
DET	Detection Error Tradeoff
DET	Detection Error Tradeoff
DNN	Deep Neural Network
DOM	Document Object Model
EER	Equal Error Rate
FAR	False Accept Rate
FAR	False Alarm Rate
FCNN	Fully Convolutional Neural Network
FN	False Negative
FP	False Positive
FPR	False Positive Pate
FPS	frames per second
FRR	False Reject Rate
GLM	Generalized Linear Model
GUI	graphic user interface
HOG	Histogram of Oriented Gradients
IoT	Internet of Things
KPCA	Kernel Principal Component Analysis
LBP	Local Binary Patterns
LDA	Local Binary Patterns
LFW	Labeled Faces in the Wild
MAE	Mean Absolute Error
mAP	mean Average Precision
MSE	Mean Squared Error
MTCNN	Multi-Task Cascaded Convolutional Neural Networks

OpenCV	Open Source Computer Vision
PCA	Principal Component Analysis
RAM	random access memory
RFID	Radio-frequency identification
RMS	Root Mean Square
ROC	Receiver Operating Characteristic
ROI	regions of interest
SIFT	scale-invariant feature transform
SSD	Single-shot Multibox Detector
SSD	solid-state drive
SVM	Support Vector Machine
TAR	True Acceptance Rate
TN	True Negative
TP	True Positive
VGG	Visual Geometry Group
YOLO	You Only Look Once

LIST OF APPENDICES

Appendix A: Face Detector 82

Appendix B: Feature Extractor 83

Appendix C: Smart Attendance System 84

Appendix D: User Interface Configuration Files 109

Appendix E: Evaluation for Two Different Models 110

CHAPTER 1

INTRODUCTION

1.1 General Introduction

Attendance is illustrated as the action of frequently being present at an occasion (Akshaya et al., 2021). It is an indispensable system for many organisations or institutions, such as schools, universities, factories or government working places to keep track of the performance and monitor the quality of their members. In educational institutions such as schools, universities or colleges, it is mandatory to mark the presence of the students by respective lecturers, whereas in the industrial sector, the attendance management system is adopted by companies to evaluate the day-to-day monitoring, leave records and overtime calculations of the employees for the facilitation of payroll systems.

Many institutes still utilise paper-based or file-based approaches for attendance management, and these manual systems are riddled with flaws such as the risk of information loss, lack of consideration, falsification, or disasters such as floods or fire outbreaks. Concurrently, some organisations have implemented automated attendance system strategies based on biometric techniques such as fingerprints, iris, voice recognition, or face recognition. However, face recognition remains a major area of research due to its non-invasive nature and its status as the primary means of identification used by humans (Abdalkarim and Akgün, 2022).

Face recognition is defined as a biometric technology that can identify or recognise someone from an image or a continuous image frame from a video (Sabeenian et al., 2020). The first implementation of face recognition started in the 1960's with a semi-automated system. A person's facial features will be automatically identified by the recognition equipment. This technology comprises face detection, face position, image pre-processing, image enhancement and face recognition. The algorithm involves the process of scanning the entire image of a person by finding out the coordinate system of all faces in the image, followed by the output of the face recognition algorithm with shapes such as rectangular or square. The DeepMind team developed an

Artificial Intelligence (AI) product called AlphaGo Zero in 2017 with its great performance of finding the ideal solution on the chessboard with the essence of face recognition by beating the No.1 player, Ke Jie at Go level (Li et al., 2020).

On top of that, the market for facial recognition systems is expected to reach a value of 8.5 billion U.S. dollars by 2025, indicating significant growth potential of this technology. Several industries across Southeast Asia have already implemented facial recognition systems. For example, in Singapore, citizens use facial recognition technology to access government services, public transportation, and even for election voting. The Singapore government has positioned itself at the forefront of this technology, allowing citizens to access government services in a contactless and efficient manner (Raj, 2021).

Additionally, it was confirmed that the COVID-19 pandemic will result in long-term effects on the aviation sector eventhough the vaccine had been globally accessible at the end of 2021 (Madzou, 2020). In this context, there is an increasing demand on face recognition technology in order to ensure contactless passenger check-in process while avoiding virus transmission. In Malaysia, physical travel documents such as tickets and boarding passes were also being slowly replaced with facial recognition at the Kuala Lumpur International Airport (Raj, 2021). It is significant that the application of this sub-domain of computer vision has emerged as a vast business around the world.

Face recognition systems have been successfully embedded in various application areas owing to the requirements such as high acceptability, collectability, and universality. Several vendors have integrated face recognition into many applications. In this context, Amazon had developed a system to allow users to make payments by using face recognition systems with the introduction of “Amazon Rekognition Image” and “Amazon Rekognition Video” which were built for analysing images and videos respectively (Bally, 2002). Besides, Apple had provided a state-of-the-art face authentication system with an advanced TrueDepth camera to accurately map the face geometry of an individual by projecting and analyzing thousands of invisible dots to create a depth map of the face for unlocking users’ devices. The Face ID can automatically adapt to the changes in appearance such as growing facial hair and wearing cosmetic makeup and it works in indoors or outdoors and even in total darkness (Apple, 2017).

Face recognition itself can be integrated with deep learning in order to show its strength and accuracy in handling vast quantities of data. Nowadays, deep learning is the main framework for image recognition process with the use of neural networks in which multiple layers are used for feature extractions. The conventional features extraction process involves using varied image descriptors such as scale-invariant feature transform (SIFT), histogram of oriented gradients (HOG) or a hybrid descriptor which are more time consuming as compared to modern approach with the implementation of deep learning which automates the selection of filters to extract best features from an image to produce better accuracy (Othman Hammadi, Abdulkarim Dawah Abas, 2018).

With the advancement of technology and deep learning, the performance of these face recognition algorithms has significantly increased, and thus allowing its commercial use in modern life. This project proposes a smart attendance system with face recognition using deep learning approach to overcome shortcomings of the conventional methods that are more time consuming and complex.

1.2 Importance of the Study

The conventional attendance system includes biometrics, cards, or iris systems. In recent years, biometric-based techniques have emerged as a useful approach for recognizing individuals, instead of relying on virtual or physical domains such as passwords, smart cards, keys, and tokens. For instance, the magnetic cards may become unreadable or be stolen by others, whereas the passwords can be easily hacked or guessed by third parties. The card system complies with the risk of misplacement of attendance records, in which fake attendance can be taken by third parties. These approaches are accompanied with the risks of being misplaced, forgotten or duplicated, which has caused the inefficiency of the attendance system (Parmar and Mehta, 2014).

However, an individual's biological traits are impossible or hard to duplicate by others. The biological traits, which include face, fingerprints, iris, palm, and voice, involve interactions by the user, except for the face recognition approach, which can be completed passively by the user from a distance by a camera or video recorder. In this context, the face recognition algorithm is utilised for identification purposes, in which an unknown individual's image will be processed by comparing the image of the person with a database of known individuals' images (Zeng, Veldhuis and Spreeuwers, 2021). The face recognition technology is slowly evolving into a universal biometric solution as it requires zero effort from the users compared with other biometric solutions, and many industries are already reaping its advantages by deploying it in the current markets.

There are a few applications in which face recognition can be applied, as the technology is not limited to attendance management. Firstly, it is widely used for security at airports, seaports, border checkpoints, building access, network security, or electronic devices. Its second significant application is surveillance, which can be used for tracking offenders or criminals with the installation of CCTVs. Existing security and surveillance cameras that are embedded with deep learning and computer vision can be deployed with face recognition augmentation for security purposes (Wang and Deng, 2021).

Nowadays, identity verification for newborns, national identities, passports, or driving licences has been normalised with face recognition technology. Additionally, the technology aids the investigations in cases involving missing individuals, immigrants, forensics, and many more scenarios. Although there are concerns about privacy raised by the public, it cannot be denied that the deployment of facial recognition technology has brought convenience, efficiency, and security to the public.

1.3 Problem Statement

The management of attendance systems is a mandatory task in many organisations, such as universities, factories, companies, or government working places. Although manual attendance taking can be performed by humans, the human memory is less adaptable at memorising a substantial dataset of faces, and the task is tedious and time-consuming. Besides, there are potentials for errors in manual data entry and difficulties in verifying attendance data across multiple locations or departments (Chatrati, Naidu and Prasad, 2013). This leads to the rapid development of automatic face recognition systems, which could bring about transformations such as real-time attendance tracking with promising accuracy.

With the conventional method of clocking in and out using a card system or a biometric system such as a fingerprint, the user tends to make physical contact with the devices. Unlike other biometric systems, the face recognition-based attendance system is convenient and safe as it can avoid human intervention. Direct contact with an infected individual and indirect contact with the surfaces in the environment or with the object will result in the transmission of the diseases virus (Higuera, 2011). Therefore, it is the right time to switch the contact-based attendance system to an indirect contact approach by face recognition so that the spread of diseases can be controlled in the event of the devastating COVID-19 pandemic or other infectious disease outbreaks.

1.4 Aim and Objectives

This project aims to develop an IoT-based face recognition smart attendance system using the deep learning approach.

The objectives of this project are:

(i) To conduct a literature review covering current methods and results in facial recognition algorithm.

(ii) To design and implement a facial recognition system that is suitable for attendance tracking.

(iii) To develop a software for the ease of attendance monitoring by the end-user.

(iv) To evaluate the performance and usability of the developed system.

1.5 Scope and Limitation of the Study

This project is mainly focused on face recognition systems for updating attendance records with the integration of machine learning and deep learning. The developed system is designed for an indoor environment in which excessive exposure to light in the outer environment may affect the performance of the system. The proposed algorithm may not cover holistic factors such as illumination, pose variation, occlusion, and facial expression. Additionally, there is no cloud server for storing image databases, and as a result, there is a lack of flexibility, a low level of security, and a fear of data recovery for this smart attendance system.

1.6 Contribution of the Study

This study can prove that recent developments in deep learning and neural networks have the possibility of yielding promising results in varied fields such as image processing and pattern recognition. The integration of face recognition systems and deep learning architectures will boost performance by facilitating the learning process and increasing the efficiency of the existing models. While there are concerns regarding privacy and accuracy, the benefits of the technology cannot be denied. There are lots of potentials with the use of face recognition technology to create a convenient and efficient attendance management system and other applications.

1.7 Outline of the Report

This report is divided into five main chapters, including an introduction, literature review, methodology and work plan, results with discussion, and conclusions with recommendations.

Chapter 1 presents an overview of the project by including a general introduction, problem statement, aim and objectives, study scope and limitations, and the contribution of the study and report outline. While the previous relevant research will be presented in Chapter 2. Chapter 3 illustrates the whole development of the smart attendance system using face recognition approaches. Next, Chapter 4 interprets the results obtained, which include face detection, face recognition, and a graphical user interface for the attendance system. Lastly, the conclusion and recommendations for future work are discussed under Chapter 5.

CHAPTER 2

LITERATURE REVIEW

2.1 Smart Attendance System

Attendance tracking plays a critical role in many organizations including educational institution, corporations or government. In higher education systems, there may be a requirement for students to maintain a certain level of attendance in order to be eligible to sit for final exams. The attendance system becomes a mandatory process in ensuring academic integrity and students' success. In this context, there are still many manual and paper-based attendance systems that come with a lot of issues. With the advancement of technology, various authors have proposed automated attendance marking systems as a promising solution to eliminate the conventional attendance taking approach.

A study implemented by Chatrati, Naidu and Prasad (2013) on attendance tracking systems was conducted using RFID readers for lecturers to monitor students' attendance in Indian educational institutions. However, its major drawbacks include inappropriate attendance marking, as students can cheat by taking others attendance, and there are risks of missing the RFID tags. Dey et al. (2014) also proposed another attendance system that makes use of a speech recognition biometric feature. However, the system's inefficiency due to the surrounding environment in crowded areas such as educational institutions or companies, where sound recognition can be mimicked to pretend to be fake attendance marking, has been highlighted.

On top of that, another biometric approach using fingerprints has been proven to be successful in eliminating fake attendance systems, but it also engages contact between the users in the organisations (Mohamed and Raghu, 2012). Furthermore, Amirulloh et al. (2020) investigated an attendance monitoring system using QR code with the Android platform, which had a 100 % success rate for functional testing of the system. However, this implementation is accomplished with the risk of manipulating attendance manipulation and a waste of time and energy.

Additionally, a research had integrated the use of Convolutional Neural Network (CNN) to recognise individuals' faces as conventional approaches, such as Fisher Faces or Eigen Faces are more sensitive to issues such as light, illumination, posture, noise, and obstruction (Arya, Mesariya and Parekh, 2020) . Another proposed system used FaceNet model for face recognition and Support Vector Machine (SVM) classifier for attendance records, achieving an accuracy of 99.6 % for multi-face recognition. The experimental results display that the performance of FaceNet model and SVM is better than VGG16 model (Nyein and Oo, 2019).

Therefore, considering the limitations displayed by the attendance systems mentioned earlier, face recognition techniques have become increasingly popular in recent years, making it a viable option for automated attendance marking systems. Table 2.1 illustrates a comparison of different attendance system approaches that are commonly used by the public.

Table 2.1: Comparison of Different Approaches for Attendance System.

Method	**Hardware**	**Main Advantages**	**Main Disadvantages**	**References**
Biometric Attendance	Fingerprint Reader, Retina Scan Machine	Automatic	Cost of machine, Maintenance issue	(Mohamed and Raghu, 2012)
RFID	RFID Reader, RFID Tag	Automatic	Risk of losing RFID Tag, Fake attendance	(Chatrati, Naidu and Prasad, 2013)
QR Code	Smartphone Camera, Barcode Scanner	Sub-automatic, cheap	Fake attendance	(Amirulloh et al., 2020)
Facial Recognition	Camera, Server	Automatic	Maintenance	(Nyein and Oo, 2019)

2.2 Face Detection

Face detection serves as a fundamental component for all facial analysis algorithms such as face alignment, face recognition, facial expression detection, and gender recognition. In 2001, Paul Viola and Michael Jones introduced a learning-based algorithm for detecting faces in images (Hazim et al., 2016). This Viola-Jones object detection framework is notable for its powerful performance in real-time face detection with impressive speed. According to Deshpande and Ravishankar (2016), the detector has illustrated its efficiency on frontal images of faces in which it can cope with 45 degree of face rotation in both vertical and horizontal axis.

However, a study has displayed that the Viola-Jones algorithm has limitations when it comes to face detection under poor lighting conditions. A study implemented by Chaudhari et al. (2018) with the use of the Viola Jones algorithm only obtained a success rate of 78.6 % due to poor lighting conditions and high false detection rate. To address these issues, Deshpande and Ravishankar (2016) presented an efficient face detection and recognition system using a fusion of Viola-Jones, Principal Component Analysis (PCA) and Artificial Neural Network (ANN), achieving an accuracy of 94 %. Furthermore, the combination of the Viola-Jones algorithm and neural network obtained an accuracy of 90.31 % as proposed by Da'San, Alqudah and Debeir (2015). Both studies were implemented using a similar database named Bio ID-Face-Database.

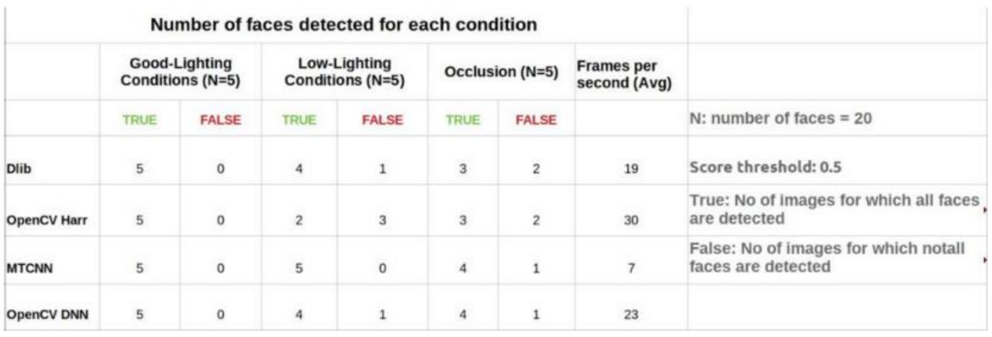
Apart from that, a component-based face detector with the use of Support Vector Machine (SVM) classifiers was implemented by Kukenys and McCane (2008) to detect eyes in grayscale images. However, this detector misses some eyes in the evaluation set due to the skipping of the image pyramid over the scale at which eyes would be detected. As SVM classifiers highly depend on training data, it is necessary to have sufficient variations in the training samples to cover different possible angles, lighting conditions, and partial occlusions in each detected image.

Another study carried out by Goyal, Agarwal and Kumar (2017) for the detection of faces in HD video was built with OpenCV and Haar-like features. The Haar classifier face detector uses a single feature to define a certain image as a face or not. However, the writer concluded that the Viola and Jones detectors are more efficient for real-time detection due to their shorter time duration and fewer CPU resource requirements.

Furthermore, the boosting algorithm named Adaboost was proposed by Schapire to enhance the accuracy of a given learning algorithm. It adjusts the training set and combines the weak classifier to form a strong classifier (Li et al., 2020). The challenge of this classifier can be investigated through the work of Meynet (2003) as it misclassified roughly 13 % of the example sets and the error rate rose quickly until more than 40 for the last selected features.

In a recent paper by Arora, Naithani and Areeckal (2022), performance analysis was conducted on four different face detection algorithms, namely Deep Neural Network (DNN), Multi-task Cascaded Convolutional Neural Network (MTCNN), Haar Cascade Classifier, and Histogram of Gradient-based (HOG) frontal face detector. The Haar Cascade Classifier works as fast as a basic CNN model by extracting multiple features and using Adaboost to select the most significant features and reduce their size. On the other hand, the HOG-based frontal face detector focuses on face landmark detection. The MTCNN, introduced by Kaipeng Zhang in 2016, comprises a three-stage cascade structure of CNN. OpenCV's deep neural network module includes DNN, which is a Caffe model that employs a ResNet-10 architecture for its operation.

After testing five images with four different face detection algorithms as shown in Figure 2.1, the results reveal that DNN performed the best, followed by MTCNN, in good lighting conditions. However, the HOG-based face detector is unable to detect small or extremely large faces, and the Haar Cascade fails to detect faces in most of the images with occlusions. DNN and MTCNN have been proven to be more robust in poor lighting conditions and images with occlusion. In conclusion, deep learning algorithms such as DNN and MTCNN have displayed better performance in various environmental setups.



Number of faces detected for each condition								
	Good-Lighting Conditions (N=5)		Low-Lighting Conditions (N=5)		Occlusion (N=5)		Frames per second (Avg)	
	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE		N: number of faces = 20
Dlib	5	0	4	1	3	2	19	Score threshold: 0.5
OpenCV Harr	5	0	2	3	3	2	30	True: No of images for which all faces are detected
MTCNN	5	0	5	0	4	1	7	False: No of images for which notall faces are detected
OpenCV DNN	5	0	4	1	4	1	23	

Figure 2.1: Testing Results on Each Face Detection Algorith (Arora, Naithani and Areeckal, 2022).

2.3 Face Recognition

Face recognition has been developed primarily for two crucial tasks: verification and identification. In this context, the verification refers to one-to-one matching in determining whether the individual claims to be when presenting with a face image of an unknown individual. The latter refers to one-to-many matching, in which an individual's identity is determined by comparing with a database of images of known individuals (Zeng, Veldhuis and Spreeuwers, 2021).

The study of face recognition has become prominent since the introduction of the historical Eigenface approach in the early 1990s. Many holistic approaches were introduced, but they failed to account for uncontrolled facial changes. This led to the development of local-feature-based face recognition in the early 2000s. However, these approaches suffered from a lack of compactness and distinctiveness. Thus, learning-based descriptors were then introduced in the face recognition field. The major evolution occurred when the deep learning approach emerged in 2012, when AlexNet won the ImageNet competition using convolutional neural networks (Wang and Deng, 2021).

Despite the advancements in face recognition technology, there are still many challenges to overcome. One of the main concerns with a face recognition system is that all images of the same face are heterogeneous. Sometimes, the detected face is unfocused and too small, resulting in a failure of the recognition process. Other issues include posing variation, lighting illuminations, occlusion, and makeup during the comparison of individuals' images with the database (Solomon, Meena and Kaur, 2021). Therefore, a multitude of face recognition systems using different algorithms have been proposed in recent years to solve the problems and achieve promising results in addressing the challenges posed by unconstrained environments.

2.3.1 Multi-level Face Recognition Taxonomy

The multi-level face recognition taxonomy, as proposed in Figure 2.2, serves as an effective tool for organising and planning face recognition solutions. It is aimed at guiding researchers to understand the technological landscape in this field, apart from exploring new possible solutions in the creation of more effective face recognition systems. The taxonomy encompasses three key face structures, namely global representation, component and structure representation, and component representation. The global representation is a face recognition approach that represents the whole face as a single entity and outputs feature extraction from the entire face region. Next, feature support considers the selection of an approach based on facial structure and the spatial support region for feature extraction. While the feature extraction level comprises a more complete set of classes and a more profound technological landscape in feature extraction (Sepas-Moghaddam, Pereira and Correia, 2020).

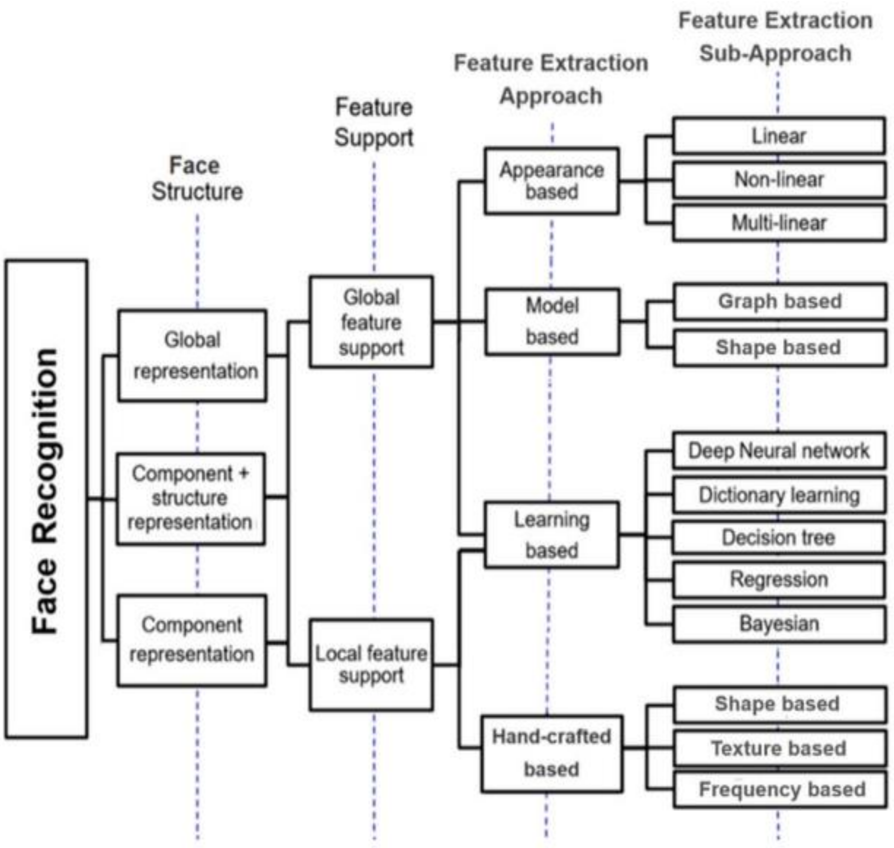


Figure 2.2: Existing Multi-level Face Recognition Taxonomy (Sepas-Moghaddam, Pereira and Correia, 2020).

2.3.2 Face Structure

In the context of face recognition, the face structure level refers to how different aspects of the face are handled by the facial recognition system. At the global level, the system focuses on the face as a whole entity, considering its overall shape and features, as illustrated in Figure 2.3. However, facial recognition systems also need to consider the component structure of the face, which involves the different facial features such as the eyes, mouth, and nose, as well as their relationships to each other. This can be seen in Figure 2.4, where each feature is represented separately. In certain conditions, the recognition system may need to select a specific facial component without considering the others. This is referred to as component representation, as shown in Figure 2.5. This level of detail is crucial for accurate facial recognition, as different facial components can provide valuable information for identifying an individual.

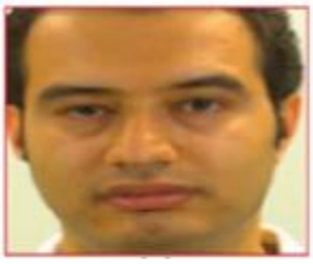


Figure 2.3: Global Representation (Sepas-Moghaddam, Pereira and Correia, 2020).

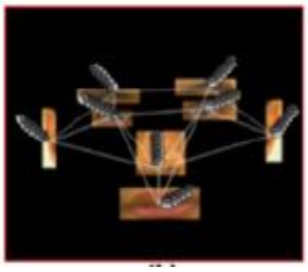


Figure 2.4: Component and Structure Representation (Sepas-Moghaddam, Pereira and Correia, 2020).

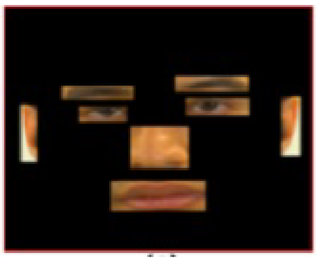


Figure 2.5: Component Representation (Sepas-Moghaddam, Pereira and Correia, 2020).

In addition to the global and component structure levels, the face structure level can be further divided into global feature support and local feature support. In global feature support, the entire selected facial structure area is considered a region of support for feature extraction. This can be either the entire face or a full-face component, and the system extracts features from this larger region of support. On the other hand, local feature support refers to a smaller portion of either the whole face or the face component that is used to extract features. This smaller region of support may vary in terms of topological standard, size, and overlapping characteristics. To achieve this, the partitioning task is accomplished by dividing the face components into squares, as shown in Figure 2.6. Overall, the division of the face into varied support regions, either globally or locally, is one of the key aspects of the face recognition process.

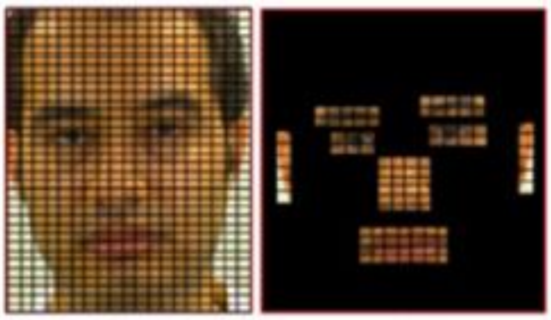


Figure 2.6: Local Spatial Support (Sepas-Moghaddam, Pereira and Correia, 2020).

2.3.3 Feature Extraction

Feature extraction is an indispensable stage in data mining and pattern recognition. It aims to shorten the machine learning duration and complexity of space by reducing irrelevancy and redundancy in the image to achieve the dimension reduction. In this context, the input data is converted into a set of features that consist of the critical information from the original data by using feature extraction algorithms. By eliminating the greatest number of irrelevant features, feature extraction maintains acceptable classification accuracy (Sufyanu et al., 2016).

As illustrated in Figure 2.7, a group of Google researchers proposed a deep learning model for face recognition namely FaceNet to produce a vector embedings of 128 numbers for each single person’s face. This model deploys a technique known as one-shot training, in which small embeddings or arrays from a couple of faces are sufficient to classify new faces. In other words, fewer photos are required to train the network with promising results. Its notable performance had been benchmarked on several famous face recognition datasets, such as Labeled Faces in the Wild (LFW), YouTube Faces (YTF), and MegaFace.

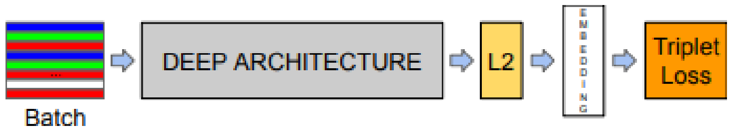


Figure 2.7: FaceNet Model Structure (Schroff and Philbin, n.d.).

FaceNet achieved 99.63 % accuracy on the LFW dataset, which consists of 13,000 face images from 5,700 individuals. On the YTF dataset, which contains videos of people speaking in unconstrained settings, FaceNet achieved 95.12 % accuracy. On the MegaFace dataset, which contains over 1 million images of over 690,000 individuals, FaceNet obtained an accuracy of 98.87 % when comparing 1 million pairs of faces and an accuracy of 99.83 % when comparing 10,000 pairs of faces, setting a new record in face recognition performance (Schroff and Philbin, n.d.).

To enhance the recognition performance, MTCNN was initially used for face detection and then its results were used as the input of FaceNet for face recognition (Jin et al., 2021). The authors also mentioned that FaceNet is able to retain face alignment and directly make use of CNN to train end-to-end after face alignment, but another deep learning model such as DeepFace is not feasible. An accuracy of over 99 % was obtained through the combination of FaceNet and K-Nearest Neighbors (KNN) as the face classifier. The proposed system was tested on self-collected datasets containing a total of 35 people with different facial expressions (Lei, Oo and Oo, 2019).

From the studies done by Brandon Amos, Bartosz Ludwiczuk (2016), another highly accurate face recognition model using deep learning techniques such as CNN was developed out namely OpenFace. One of its advantages over other models is that it had been optimized to run on mobile devices, enabling real-time face recognition on such platforms. The system's performance had been evaluated on image pairs in the LFW dataset, achieving an accuracy of 92.92 %. Another deep neural network model, DeepFace, was developed by Facebook researchers and trained with 4 million images from over 4000 people, obtaining an accuracy of 97.47 %.

Figure 2.8 provides further analysis of the performance of these deep learning models on masked face images from the RMFRD dataset, which consists of 5000 images. The analysis shows that VGG Face achieved the highest accuracy with 68.17 %, followed by FaceNet with an accuracy of 67.48 %, while OpenFace obtained an accuracy of 63.18 %. However, it was concluded that none of these pre-trained models had a great performance on masked face images, but their performance could be improved through transfer learning and fine-tuning (Bharat Chandra and Karthikeya Reddy, 2020).

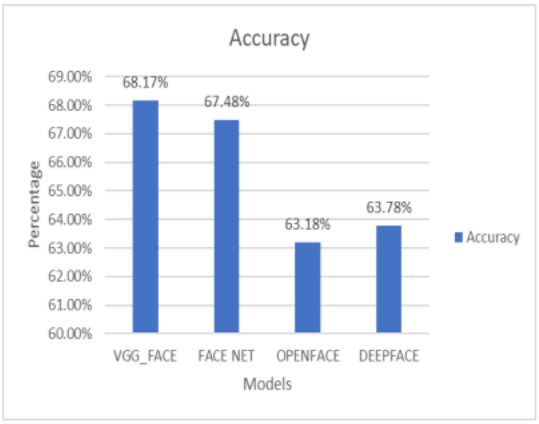


Figure 2.8: Comparison Between Conventional and Deep Learning Approach on RMFRD Dataset (Bharat Chandra and Karthikeya Reddy, 2020).

This feature extraction process can be further separated into three common approaches: local, holistic or appearance-based, and learning-based. Some famous techniques for each approach include Histogram of Oriented Gradients (HOG) and Local Binary Patterns (LBP) under local approaches, Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and Kernel Principal Component Analysis (KPCA) under holistic approaches, and CNN, Yolo, and SSD under deep learning approaches. These techniques will be discussed in the next few sessions.

2.3.4 Local Approach

In feature extraction, the local approach refers to an approach that focuses on local features or pattern within an image or dataset instead of considering the whole image or data as a whole. One prominent approach in local feature extraction is the Histogram of Oriented Gradients (HOG) descriptor, originally proposed by Dalal and Triggs, which has evolved from the scale invariant feature transform (Tan, Yang and Ma, 2014). HOG is a sophisticated framework that emphasizes facial features by using 68 facial focus descriptors that remain unaffected by changes in illumination and rotation. At the initial stage of objects detection, HOG was widely recognized as a prominent method.

The fundamental theory behind HOG features is that the distribution of local intensity gradients or edge orientations can represent local object appearance and shape, even in the absence of precise knowledge of matching gradients or edge placements. For each key point in an image, the HOG features are produced by partitioning the neighboring area into multiple evenly spaced cells and computing a histogram of edge orientations for each cell. A local one-dimensional histogram of gradient directions is then accumulated over all pixels in the cell's adjacent area around each key point (Shu, Ding and Fang, 2011).

Numerous studies have shown the effectiveness of using HOG features in combination with other classifiers for various applications. As proposed by Dadi and Mohan Pillutla (2016), a designated face recognition with the combination use of HOG features and Support Vector Machine (SVM) classifier had achieved an improved rate of 8.75 % when compared with Principal Component Analysis (PCA) on the ORL database. Furthermore, on the F-MNIST dataset, the combination of HOG features with SVM achieved an accuracy of 86.53 %, whereas another model using HOG features, Local Binary Pattern (LBP), and SVM obtained an accuracy of 87.4 % (Greeshma, College and Gripsy, 2020).

Furthermore, the combination of HOG, Adaboost, and SVM were deployed for real-time vehicle detection on the GTI vehicle datasets, and the results show that the combination of HOG and AdaBoost achieved a higher accuracy of 97.24 % compared to approaches using HOG and SVM classifier with a recognition rate of 96.89 % (Singhal et al., 2021). The use of HOG features in combination with other classifiers has proven to be effective in various applications, demonstrating the potential of the local approach in feature extraction.

Apart from that, the second local approach to be discussed is Local Binary Pattern (LBP), which has good performance in varied applications such as image retrieval, texture classification and segmentation, and surface inspection (Chang-yeon, 2008). The working principle of LBP is very easy. Originally, the LBP operator labels the image pixel by performing a threshold on the 3 × 3 neighbourhood of each pixel and comparing it with the centre pixel value. If the neighbouring value is greater than the centre value, a binary number of one will be produced. Conversely, a binary number of zero will be assigned to a value that is smaller than the centre value. Figure 2.9 displays the basic calculation for LBP.

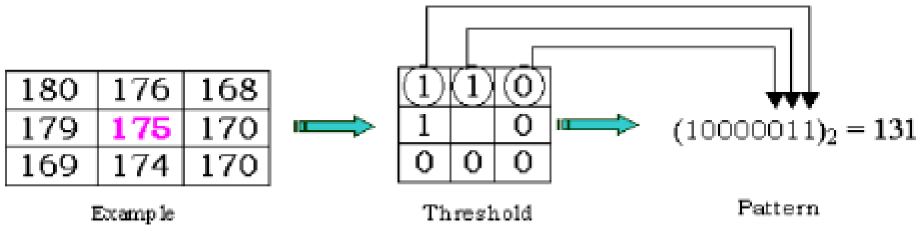


Figure 2.9: LBP Calculation (Chang-yeon, 2008).

The LBP code is computed for every pixel in the image, and the frequency of each possible pattern is recorded. These pattern frequencies are used as labels to construct a histogram, which forms a feature vector and serves as a representation of the image texture. This allows for the measurement of similarity between different images (Wahid, 2013).

From the research studied by Dalali and Suresh (2016), LBP was used to reduce the information in given images by comparing results of images with noises and images without noise on MIT face dataset. As a result, the images without noise reached up to 99.3 % of accuracy and the images with noise obtained an accuracy of 98.28 %. Additionally, LBP had been widely used with the integration of PCA and SVM and achieved good recognition rate.

In the context of facial recognition, LBP has several advantages, including its great computational efficiency and the ability to handle grayscale and colour images. These advantages make LBP an ideal choice for measuring similarities between images in facial recognition, object detection, and image segmentation applications. To further enhance its performance in texture classification, LBP has been expanded to accommodate more complex texture features, such as rotation-invariant LBP and uniform LBP. Recent studies have shown that the combination of LBP and other deep learning techniques can further enhance its capabilities in image processing tasks.

Histograms of Oriented Gradients (HOG) and Local Binary Pattern (LBP) are both recognised as effective descriptors for facial recognition. Both methods utilise gradients around a pixel to compute histograms for matching the similarity between images. HOG is known for capturing image edges and corners, while LBP is complementary for capturing local patterns. Moreover, LBP makes use of all eight directions for each image pixel compared to HOG, which only requires one direction for each image pixel, but this can cause LBP to lose information compared to HOG.

Based on the ROC curve with Support Vector Machines (SVM) classifier as displayed in Figure 2.10, the combination of HOG and LBP algorithms yielded the highest rate of 99.1 %, whereas the HOG algorithm itself outperformed LBP with a rate of 0.01 (Amraee, Chinipardaz, and Charoosaei, 2022). Furthermore, another research in 2021 conducted by Lakshmi and Ponnusamy (2021) had merged the HOG approach with LBP and SVM, and it achieved an accuracy of 90.83 % and 97.66 % on JAFFE and CK+ datasets, respectively.

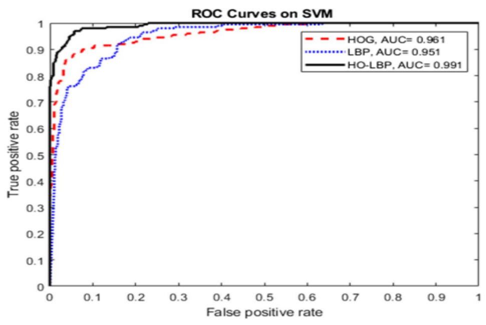


Figure 2.10: Comparison of Performances for HOG Approach and LBP Approach (Amraee, Chinipardaz and Charoosaei, 2022).

In conclusion, HOG and LBP are both effective descriptors for facial recognition, with HOG capturing image edges and corners, and LBP being better at capturing local patterns. While HOG is more efficient than LBP, the combination of both approaches outperforms the algorithm alone.

2.3.5 Holistic Approach

The holistic approach makes use of global information from an individual's face to carry out face recognition. In this context, global information is represented by features which are directly extracted from the face image pixels (Karamizadeh and Abdullah, 2013). The holistic approach is further divided into linear and non-linear approaches.

The Principal Component Analysis (PCA) or eigenface approach is a linear holistic approach and is being introduced as one of the most effective techniques for image recognition and compression. It was proposed by Pearson and Hotelling in the early 20th century (Huang, 2012). This approach aimed to reduce the dimensionality of the data space to the smaller intrinsic dimensionality of the feature space. PCA converts the large one-dimensional vector of pixels constructed from a two-dimensional facial image into the principal components of the feature space. It has advantages over other techniques due to its simplicity, speed, and insensitivity to mild or progressive changes in the face.

PCA considers a grayscale image as an $M \times N$ matrix of pixels and transforms it into a high-dimensional vector. The principal components are then calculated for the 'k' faces of the population, resulting in the minimum distance between the new image and any of the 'k' library images. This approach retains variance information for analyzing images based on similarities and differences and is widely used in face recognition. Although PCA has benefits such as lack of data redundancy, smaller database representation, noise reduction, and reduced complexity in image grouping, it has limitations in capturing the simplest invariance unless the required information is provided by training data, and the covariance matrix evaluation process is difficult (Karamizadeh et al., 2013).

Linear Discriminant Analysis (LDA), also known as Fisher's Discriminant Analysis, is another technique for dimensionality reduction in face recognition. LDA identifies a linear transformation to create feature clusters through scatter matrix analysis. It combines independent features linearly to produce the greatest mean difference between desired classes, with the goal of minimizing the within-class scatter matrix measure while maximizing the between-class scatter matrix measure. This approach has been proven effective in improving face recognition accuracy (Toygar and Acan, 2003).

Traditionally, the PCA algorithm is incorporated with the LDA framework, in which PCA is used as a preprocessing step for dimensionality reduction and LDA is implemented in a lower-dimensional PCA subspace (Lu, Plataniotis and Venetsanopoulos, 2003). The main difference between LDA and PCA is that LDA focuses on data classification, while PCA focuses on feature classification. While PCA changes the shape and location of original datasets when transforming them to a different space, the location remains unchanged in LDA (Hese and Banwaskar, 2013).

When solving problems related to pattern classification, LDA outperforms PCA, but its separability characteristics do not directly correspond to the classification accuracy in the output space. According to a research carried out by Patil, the performance of LDA was better than PCA, with a recognition rate of 86.07 % and 66.07 %, respectively, based on the ORL database (Patil, 2014). Another study using MATLAB found that LDA outperformed PCA with a larger training set, but PCA had better performance with a smaller training set. However, for the same number of samples, LDA generally had a higher recognition rate than PCA (Suganya and Menaka, 2014).

The standard PCA approach only supports linear dimensionality reduction. Therefore, if the data deals with more complex features that are not suitable for linear representation, the PCA technique will not be useful. However, the introduction of Kernel PCA (KPCA) allows the evolution of PCA to non-linear dimensionality reduction (Wang, 2012).

According to Cover's theorem, with the transformation of the input space into a high-dimensional feature space in the KPCA approach, the non-linear separable patterns transform into ones that are linearly separable. In other words, KPCA is the non-linear form of PCA. Although computing vector products in high-dimensional feature space is complicated, it is possible to carry out computation in low-dimensional input space by using a kernel function in KPCA. The kernel approach can solve the non-linearity problem by mapping the input face images into a higher dimensional space with simplified face linearity (Peter, Minoi and Hipiny, 2019).

By utilising the TOAM database consisting of 40 individuals, the KPCA outperformed the PCA with recognition accuracy of 80.0 % and 72.5 %, respectively (Niyi, Alagbe and Wuraola, 2019). Besides, the KPCA approach is utilized for feature extraction from the input images. The combination of KPCA and SVM had better performance with an average recognition rate of 99.05 % as compared to SVM alone which displayed an accuracy of 95.04 % based on the VISIO Multiview Face database (Timotius, Setyawan and Febrianto, 2010).

In face recognition, KPCA excels in extracting more powerful features as compared to PCA. Figure 2.11 illustrates the comparison of recognition rates for both PCA and KPCA in FERET, ORL, UMIST, and AT&T databases. The KPCA technique outperformed the PCA technique with an accuracy of 81 % to 91 % from four of the databases. It has been demonstrated that the KPCA approach is more effective in extracting low-level features of face images, which is a crucial requirement for successful face recognition (Anon., 2014).

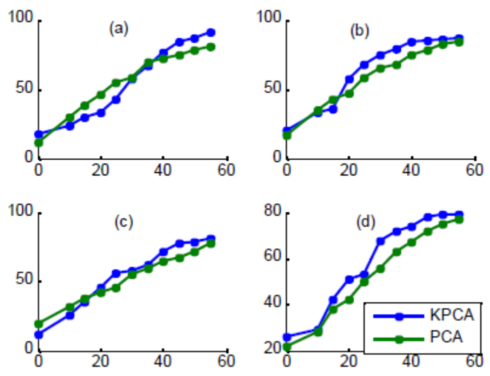


Figure 2.11: Recognition Rate for PCA and KPCA Approaches (Anon., 2014).

2.3.6 Learning-based Approach

To improve the performance of face recognition technologies in real-life applications, different deep-learning approaches have been introduced lately. In the early 2010s, learning-based local descriptors reshaped the application of face recognition, in which local filters are enhanced with better performance and more compact encoding (Wang and Deng, 2021). In this section, a comparative analysis of the Convolutional Neural Network (CNN), Single Shot Multibox Detector (SSD), and YOLO will be discussed below.

The Convolutional Neural Network (CNN) belongs to a type of neural network that is known for image classification and can be applied to face recognition. To distinguish an input image from other images, CNN tweaks the network weights based on the input image and consequently, CNN is able to recognize different faces by identifying important features in their own after training the model (Lecun et al., 1998).

In late 1998, the first popular CNN was developed for the development of handwritten digits and was applied to banking services in automated teller machines (Gopalakrishan, Arun and Sasikumar, 2021). Since 2012, scientists have concentrated on enhancing the usefulness of CNN’s architecture and methodologies, including layer design, activation function, and regularization as well as investigating the functionality in many sectors.

The introduction of Fast R-CNN as an object detection model makes use of CNN in the target detection field and has been studied by Ren et al. (2016). Kim et al. (2018) implemented a CNN-based study to create a framework for finding moving objects with Closed Circuit Television (CCTV) cameras by adding a background subtraction algorithm to the framework.

The major breakthrough of CNN was made by AlexNet with an error rate of just 15.3 % in the ILSVR challenge in 2012, and this achievement is impressive as the technology relies on truly deep learning with great computational power (Krizhevsky, Sutskever and Hinton, n.d.). From 2014 to 2020, architectural innovations were made with many CNN-based architectures, such as improved versions of RCNN and Faster RCNN, which were introduced in the subsequent years. Researchers have come up with different CNN models for face recognition. These models show the realm of image classification with different CNN models, as illustrated in Figure 2.12.

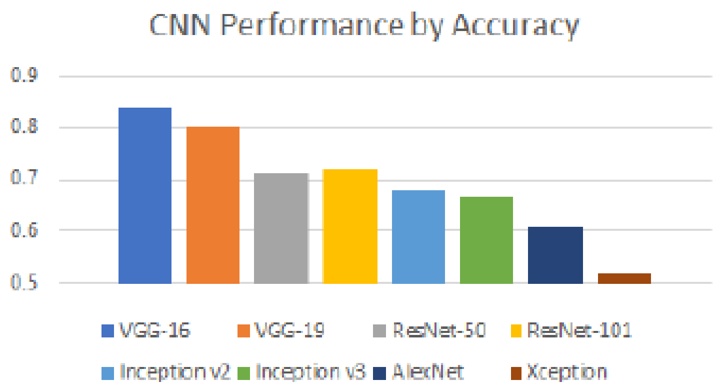


Figure 2.12: Comparison Graph of Different CNN Models (Gwyn, Roy and Atay, 2021).

A study conducted by Gwyn, Roy, and Atay (2021) compared various CNN variants, including AlexNet, Xception, Inception, ResNet, and VGG, using the Labelled Faces In The Wild (LFW) image dataset, which consisted of 4788 images from 423 individuals. The results showed that VGG-16 performed the best, followed by VGG-19, with an accuracy of 84 %. However, although VGG-16 and VGG-19 excel in image classification and object localization, their training process is slow due to their robustness.

To improve performance and accuracy with more complex features, ResNet, a prominent type of CNN, was proposed by He et al. (n.d.) with an accuracy of 72 %. Although AlexNet is the standard object recognition model in CNN, it faces challenges when applied to higher resolution images.

Inception v2 or Inception v3, on the other hand, makes use of advancements in deep learning, with more deeply stacked convolution layers. The Xception model is a transformation of the Inception model, with similar architecture and parameters but deeper separable convolutions. However, it had the lowest accuracy of 52 % among the models tested (Gwyn, Roy and Atay, 2021).

A crucial consideration in the detection pipeline is the speed of the object detection model. However, increasing the speed may lead to a decrease in accuracy. To address this challenge, the Single Shot Multibox Detector (SSD) was introduced in 2016 by utilizing VGG as the backbone of the network (Liu et al., 2016). Compared to other single-shot detectors like YOLO and R-CNN, SSD is faster and more accurate. In addition to VGG, SSD employs multiple convolutional layers for detecting smaller targets, and its deeper layers are used for detecting larger objects, as illustrated in Figure 2.13.

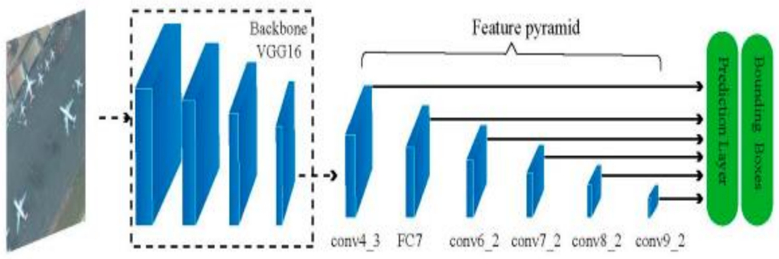


Figure 2.13: Framework of SSD (Shi, Bao and Tan, 2019).

SSD differs from other models as it does not involve pixel or feature sampling for bounding box hypotheses, making it easy to train and embed into a system for object detection. Based on a research done by Tai et al. (2018), the training process for SSD is straightforward, and alternative models such as SSD-7, SSD-300, and SSD-500 can easily balance between accuracy and speed.

When compared to R-CNN, SSD demonstrated better performance on PASCAL VOC and COCO datasets while being three times faster during computation. Furthermore, SSD excels in object localization as it directly learns to classify object categories, avoiding the use of two separate processes that can lead to more localization errors in R-CNN. Moreover, SSD outperformed YOLO, with an input image running at 59 FPS, demonstrating higher efficiency and accuracy than YOLO (Zhao et al., 2019).

The Face-SSD approach is a novel method for face recognition that uses a single shot face-related task analysis approach. It integrates a Fully Convolutional Neural Network (FCNN) as the backbone and builds upon the Single Shot Multibox Detector (SSD) method. This approach is capable of detecting multiple faces in real-time. Researchers found that it achieved a 95.76 % accuracy for smile detection and 90.26 % for attribute prediction, with a reported RMS error of 0.44, indicating high accuracy and reliability of the results (Jang, Gunes and Patras, 2019).

As shown in Figure 2.14, YOLO, or You Only Look Once, uses the Darknet architecture as its backbone and replaces Softmax classification loss. It uses neural networks to store information about people's appearance and class, with parallel bounding boxes used as detectors in images. The input image is divided into S × S grids, and object detection is successful if the object's centre points fall within the grid cell. The performance of YOLO is analyzed by estimating the prediction of the bounding box and confidence score for each grid cell, with a confidence score of zero indicating no object within the grid cell (Rana, 2022).

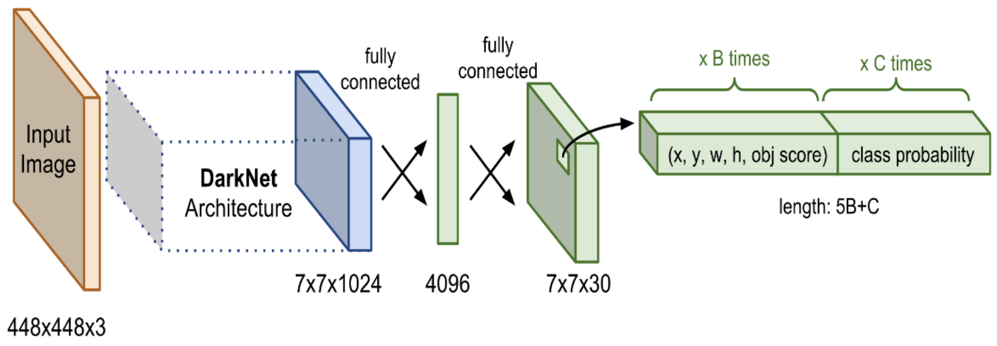


Figure 2.14: YOLO Algorithm Based on DarkNet Architecture (Weng, 2018).

With its fast and accurate performance, YOLO gains significant popularity upon its first release in 2015. The architecture improvements are continuing to advance in the field of computer vision with innovation in each version of the algorithm.

In 2020, the YOLO-v4 algorithm was updated to incorporate the CSPDarkNet 53 and PANet architectures for object detection, leading to improved face detection, even with masks. The algorithm features reduced parameters for feature extraction and increased information fusion. However, the YOLO-v4 algorithm required more frames per second (FPS) at 23.83 s as compared to YOLO-v3, which only required 21.39 s for processing the same picture during speed calculation (Yu and Zhang, 2021).

Another distinct work was implemented by researchers in Shenzhen, China by introducing YOLO5face, based on the YOLO-v5 object detector. This approach resulted in improved speed and mean-average precision performance. By testing the model on the WiderFace dataset, it achieved a mean average precision of 96.67 %, demonstrating the significant performance of the YOLO algorithm (Qi et al., 2021).

In 2022, Wang, Bochkovskiy and Liao (2022) had released the official YOLOv7 with a paper named “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors”. This model was declared to be the fastest and most precise real-time object detection model for computer vision.

Additionally, the latest object detection model, YOLOv8, has recently been released with new features and enhancements, with a strong focus on accuracy, speed, and size. In this context, there are five types of YOLOv8 models, namely YOLOv8n, YOLOv8s, YOLOv8I, and YOLOv8x. YOLOv8x has the highest accuracy but is the slowest among them, while YOLOv8 Nano is the smallest and fastest. Figure 2.15 shows that YOLOv8 significantly outperforms older versions of the YOLO model on COCO datasets (Francesco and Solawetz, 2023).

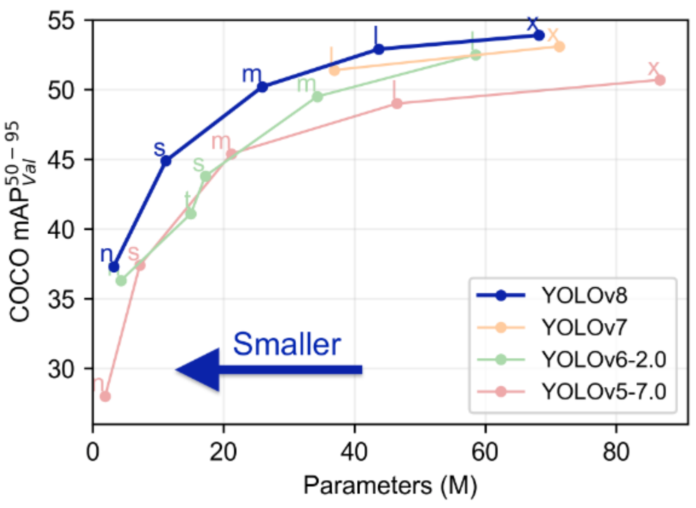


Figure 2.15: Comparison of Different Versions of YOLO Model (Francesco and Solawetz, 2023).

The Table 2.2 lists out the major differences between CNN, SSD and YOLO in terms of accuracy, speed, advantages and disadvantages.

Table 2.2: Comparison Between CNN, SSD and YOLO (Srivastava et al., 2021).

	CNN	**SSD**	**YOLO**
Accuracy	Accuracy is the highest among three models.	Accuracy is lesser as compared to CNN family.	Accuracy is the lowest as compared to CNN family and SSD.
Speed	Slower than SSD and YOLO.	Faster than CNN family but slower than YOLO.	Fastest among three algorithms.
Main Advantages	Reduced chance of overfitting due to fewer parameters to learn.	Can be trained end-to-end to improve accuracy.	Good for real-time processing. Can be trained end-to-end to improve accuracy.
Main Disadvantages	Requires lots of samples to construct a depth model.	Not suitable for small objects.	Difficult to detect objects that are small and close to one another.

2.3.7 Transfer Learning

Transfer learning is an indispensable concept in deep learning as a small amount of data is sufficient to train deep neural networks. In this context, a pre-trained model for face recognition serves as a good starting point as it has already been trained for learning low-level features such as shapes, textures, and edges. By fine-tuning the pre-trained model on a new dataset, the need for extensive training data and time is reduced, making it more robust and accurate in real-world scenarios. This approach is useful when applied to new faces or changing environmental conditions, such as changes in lighting or pose, making it more robust and accurate in real-world scenarios.

Several related works have made use of transfer learning in face recognition, such as the application of VGG16 for feature extraction and classification using a Softmax layer. The VGG16 model was trained on the ImageNet dataset, resulting in an accuracy of 83.11 % in an experiment involving 15 subjects, each with 11 images (Singh, Kansari and Sinha, 2022). Additionally, a transfer learning-based CNN model achieved an accuracy of 98.7 % and 100 % on the Yale and AT&T datasets, respectively (Meena Prakash, Thenmoezhi and Gayathri, 2019). Another work accomplished by Atabansi et al. (2021) successfully applied transfer learning in combination with the pre-trained VGG-16 network architecture to the Oulu-CASIA NIR dataset, achieving an average test accuracy of 98.11 % in facial expression recognition. These findings underscore the potential of transfer learning for improving the accuracy and robustness of face recognition in real-world scenarios.

2.4 Evaluation Metrics

To evaluate the performance of a proposed face recognition system, it is essential to consider several metrics. Four prominent metrics were discussed by Harakannanavar et al. (2019) namely False Accept Rate (FAR), False Reject Rate (FRR), Receiver Operating Characteristic (ROC) and Equal Error Rate (EER).

FAR refers to the system’s probability of incorrectly matching an input pattern to a template that does not match in the database, while FRR refers to the system’s failure to detect a match between the input pattern and a matching template in the database. The ROC curve is an useful visualization tool that displays the trade-off between FRR and FAR, while EER can be determined from the ROC curve.

In the CNN architecture, different performance metrics were utilised for face and head detection as proposed by Nguyen-Meidine et al. (2018). A ROC curve is used to measure accuracy, with the true positive rate (TPR) plotted against the false positive rate (FPR). The TPR refers to the proportion of target face regions of interest (ROI) precisely detected as faces over the entire region of interest.

For video surveillance applications, the Precision-Recall space is more appropriate for measuring detector performance under imbalanced data. Based on the FDDB dataset, three region-based CNNs, Faster R-CNN, R-FCN, and PVANET, had provided higher accuracy levels than SSD, while SSD showed more competitive performance in the Precision-Recall space, as illustrated in Figure 2.16 (Nguyen Meidine et al., 2018).

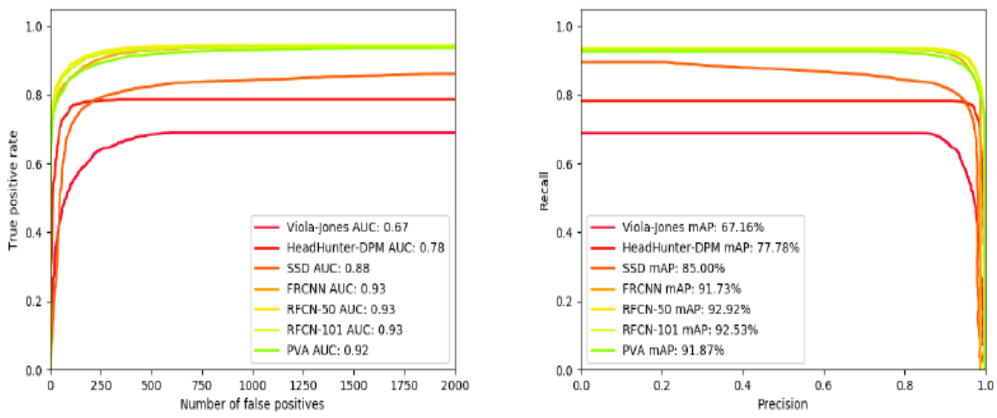


Figure 2.16: ROC and Inverted Precision-Recall Curves (Nguyen Meidine et.al., 2018).

Aside from that, Masi et al. (2019) utilized the ROC curve for verification and the True Acceptance Rate (TAR) at precision cutoff points and False Alarm Rate (FAR) to evaluate performance on the IJB dataset. The cumulative match characteristic (CMC) curve was also used to analyse the recognition rate at multiple ranks. The Detection Error Tradeoff (DET) curve, which is similar to the ROC, was used to measure quality identification.

Additionally, another study by Srivastava et al. (2021) used the average precision and F1 score as the performance metrics for the face detection model using deep learning approaches.

Next, another study on face detection by deep learning carried out by Chaves et al. (2020) utilized a regression model for the prediction of face detector speed and evaluated the model using Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). In addition, by using mean Average Precision (mAP) and the F1 score for the WIDER Face and UFDD datasets, it was found that MTCNN is the fastest but least accurate detector. Additionally, the writer also evaluated performance by using Generalized Linear Models (GLMs) for the estimation of processing time and F1 score of face detectors, as GLMs are flexible linear regression models that take into account metrics such as processing time and F1 score.

In short, evaluation metrics are essential to assess the performance of the face recognition system. In addition to testing the accuracy of the system, the metrics also facilitate the comparison and selection of the most suitable face recognition techniques for specific applications.

2.5 Database

The selection of a suitable face database is crucial for developing and evaluating robust face recognition systems. It is essential to have sufficient data and variability in the database to cover controlled variables such as illumination, pose, expression, occlusion, age, and ethnicity (Gross, 2005). For deep learning-based face recognition systems, having a large training dataset is particularly crucial to learn complex features from images. For instance, ImageNet with 14 million images significantly enhanced the development of precise deep learning object detection models.

The ORL (Olivetti Research Laboratory) face database is one of the oldest face databases, consisting of 400 face images taken between April 1992 and April 1994. This database was created for the face recognition project, and all images were taken against a dark homogeneous background, leading to inconsistent illumination conditions (Roure and Faundez-Zanuy, 2005). Another well-known face database is YALE dataset, which includes 165 grayscale images of 15 individuals. However, this database has limitations, such as a small number of individuals, and environmental factors such as the presence and absence of ambient light are not indicated (Kriegman, n.d.).

Different databases used for similar recognition approaches will affect the performance of the face recognition system. In the examples of studies carried out by Du, Su and Cai (2009) and Xie (2009) on the ORL Faces database and Yale Faces database, an accuracy of 96 % and 97.78 % was achieved respectively, using the Support Vector Machine (SVM) classifier. Furthermore, the PCA technique was applied to the ORL face database by Yi, Lei, and Li (2015) and to the Yale database by Chai, Shan, and Gao (2003). An accuracy of 70 % and 93 % was achieved, respectively.

A more recent database, ELA5 was introduced by Alexiadis et al. (2010) for the main purpose of experimentation within the face recognition domain. The database covers pose and illumination variation, different facial occlusions and expressions, and consists of a total of 1260 images from ten individuals. Some techniques, such as PCA, MPCA, and LDA, were used to simulate real-world conditions. When MPCA and LDA were used together in the first experiment, 95 % of the images were recognized. This shows that traditional PCA techniques are sensitive to changes in lighting conditions.

Other recently added face databases include the Basel Face Database, which consists of forty individuals with fourteen variations of each of them displaying different personalities (Walker et al., 2018). Another distinct face database, Labeled Faces in the Wild (LFW) is well-known as a public benchmark for face realism and consists of 13233 images. This database was specifically designed for studying unconstrained face recognition problems. The constraints include poor lighting, extreme poses, occlusions, and many groups are not represented properly, such as lack of ethnicity, few children, no babies, and people over the age of 80 (Zhang and Deng, 2016).

There are several factors that affect the quality of a face database. Firstly, the pose variance may affect the success of the face recognition system due to the inconsistency of individuals when taking a picture, as people may pose differently and there is no standardisation for photo taking. Hence, the pose variance may degrade the performance of face recognition systems due to their inflexible imaging conditions.

Secondly, occlusion refers to any obstacles in an image, such as hands, hair, sunglasses, or other items. Partial occlusion refers to occlusion that is less than 50 % of the face. Additionally, factors such as shadows can also contribute to the category of occlusion. These can be problematic for face recognition systems as they obstruct parts of the face that are important for accurate recognition.

Lastly, lighting is an essential consideration that depends on whether the photos were taken in a controlled environment with a homogeneous background. Illumination can greatly affect the face appearance and lead to inconsistencies between images. These factors highlight the importance of having a sufficient and diverse face database for training and testing a robust face recognition system (Lal et al., 2018).

In summary, the selection of a suitable face database is essential for developing a robust face recognition system. A large training dataset with sufficient variability can enhance the accuracy of deep learning-based face recognition systems.

2.6 Comparison between Traditional Approaches and Deep Learning Approaches

The field of face recognition has undergone significant transformation over the years. Initially, conventional approaches relied on edge and texture descriptors combined with machine learning techniques such as Support Vector Machines (SVM), Principal Component Analysis (PCA), and Linear Discriminant Analysis (LDA). However, these traditional techniques were not robust when facing variations in unconstrained environments. In recent years, deep learning approaches based on convolutional neural networks (CNNs) have been successfully adapted for image dimensional reduction and recognition due to their outstanding accuracy and fast computation.

The first CNN approach for face recognition that utilized a high-capacity model, namely Facebook's DeepFace, achieved an accuracy of 97.35 % on the LFW datasets with an error reduction of 27 % (Trigueros, Meng and Hartnett, 2018). Additionally, based on a research implemented by Setiowati et al. (2017) with the YALE dataset showed that the non-deep learning algorithm can achieve up to 90.6 % for low-high complexity and 94.67 % in the deep learning approach for low to high complexity.

Most studies comparing conventional and deep learning approaches to face recognition show that the latter provides greater performance. Jayaswal and Dixit (2020) implemented the first approach by using the Viola Jones approach for image detection, followed by the Local Binary Pattern for extracting features by recognising the face. However, the real-time system took only 300 datasets and achieved an accuracy of just 50 %.

Subsequently, the deep learning approach was carried out by initially resizing the images with the MTCNN algorithm, followed by the FaceNet model to extract high-quality images. As a result, it has been proven that deep learning is suitable for models that embed larger datasets, as the accuracy obtained was 96 %. Furthermore, by applying low to high complexity to the YALE dataset, the CNN approach and the LBP approach achieved recognition rates of 93.3 % and 85.75 %, respectively (Setiowati et al., 2017). Table 2.3 summarises the comparison of SVM and deep learning approaches on MNIST datasets.

Table 2.3: Performance Comparison for SVM and Deep learning.

Approach	**SVM**	**Deep Learning**
Operating time	46.54 minutes	11 hours and 50.41 minutes
Accuracy in training set	94.09 %	100 %
Accuracy in testing set	93.92 %	98.85 %
Means for extracting features	Manually and subjective	Automatically and objective
Means for processing data	Turn images into vector	Directly using images
References	(Lai, 2019)	(Lai, 2019)

On top of that, a research from Singhal et al. (2021) compared traditional and deep learning approaches for face recognition on the ORL database. According to Figure 2.17, the combination of CNN and LBP approaches achieved a 100 % recognition rate, while the PCA with Neural Network achieved approximately 97 % accuracy. In contrast, the traditional approach, which used HOG and SVM techniques, achieved the lowest accuracy of 90 %. These results suggest that a combination of deep learning and traditional approaches can lead to better performance compared to other models.

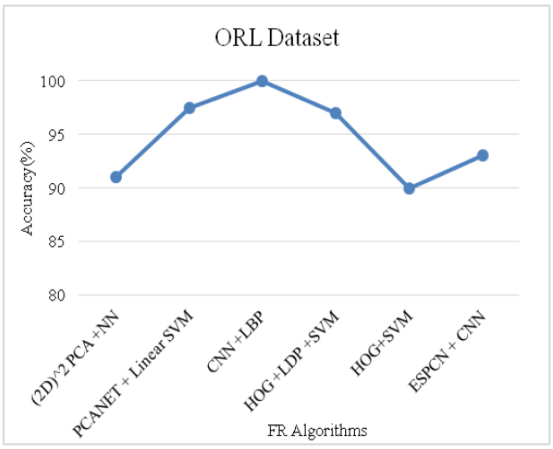


Figure 2.17: Performance of Various Techniques on ORL Database (Singhal et al., 2021).

Overall, deep learning approaches have better performance in terms of accuracy and robustness compared to traditional approaches for face recognition. This is due to its capabilities of learning high-level features directly from raw data, which makes it easier to adapt to different image variations. Additionally, large datasets are easier to handle with deep learning approaches, making them a suitable option for the development of face recognition.

Aside from good performance in terms of accuracy in most experiments, deep learning is associated with some weaknesses. The first issue is high complexity, which makes it difficult for deep learning to encounter situations such as varied facial poses, lighting and illumination, facial expressions, or partial occlusions. Next, the build-up of a deep learning approach consumes more training time and computing resources in order to achieve better generalisation ability and prediction accuracy.

The optimised solution is by integrating both conventional and deep learning approaches into the system in order to obtain results of high accuracy and low complexity. For instance, the LBP approach implemented on the ORL dataset achieved an accuracy of 97 %, whereas another experiment that integrated with the CNN technique and the LBP approach applied on a similar dataset obtained a recognition rate of 100 % (Singhal et al., 2021c). Table 2.4 illustrates the comparison between conventional approaches such as HOG, LBP or PCA and deep learning approaches such as CNN, SSD or YOLO.

Table 2.4: Comparison Between Conventional and Deep Learning Approach.

Conventional Approach	**Deep Learning Approach**
Lower Accuracy	Higher accuracy
Consume lesser training time	Requires larger training datasets with longer training time
Used for simpler application	Solve complex problems

In conclusion, while deep learning approaches have shown superior performance in face recognition, there are still limitations, such as high complexity and resource consumption. The viable solution to these limitations is to integrate traditional approaches with deep learning to further advance the face recognition field.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

In this chapter, clear and proper planning of the entire project activities will be explained in order to keep track the progress and ensure the project running successfully. Additionally, detailed methodology was covered in clear manner to ensure the results compatible with project aim and objectives. The methodology will be explained parts by parts which consist of project planning, development process, face recognition approaches followed by design of the user interface for the whole smart attendance system, and lastly, the solutions to the difficulties faced during the development process.

3.2 Project Planning

The whole project was separated into two stages which are Final Year Project 1 and Final Year Project 2. Each of them took about 14 weeks. At the beginning of the project, the project scope, problem statement and objectives were clearly stipulated to ensure the whole project aligned with the targets. Besides, a literature review was conducted to discuss on the background and few techniques leading the implementation of the project apart from providing clear perspective about the project to the audience. Research was conducted before proposing the methodology or possible solutions followed by preliminary testing and results analysis.

In FYP2, a fully working prototype was developed with performance evaluation. The initial prototype was accomplished on Week 4 as planned. Then, field test was carried out to analyse the performance of the recognition system and further enhance the prototype. Some useful features were updated in order to enhance user's experience. On Week 14, the whole project was accomplished. The project's Gantt Chart was displayed in Figure 3.1.

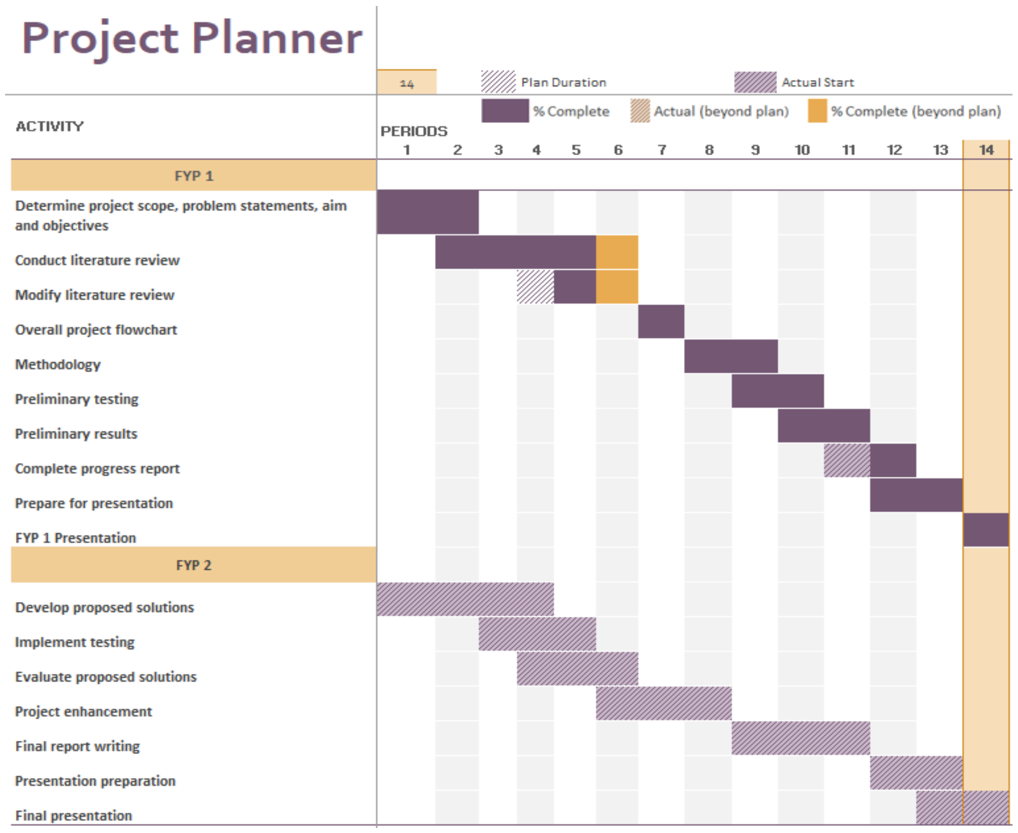


Figure 3.1: Project Gantt Chart.

3.3 Project Development

Two different face recognition approaches were tested to determine the most suitable technique for integration into the smart attendance system. The first approach combined DNN, FaceNet, and an SVM classifier, while the second approach used transfer learning with a VGG16 model. Figure 3.2 below provides a general overview of both recognition techniques.

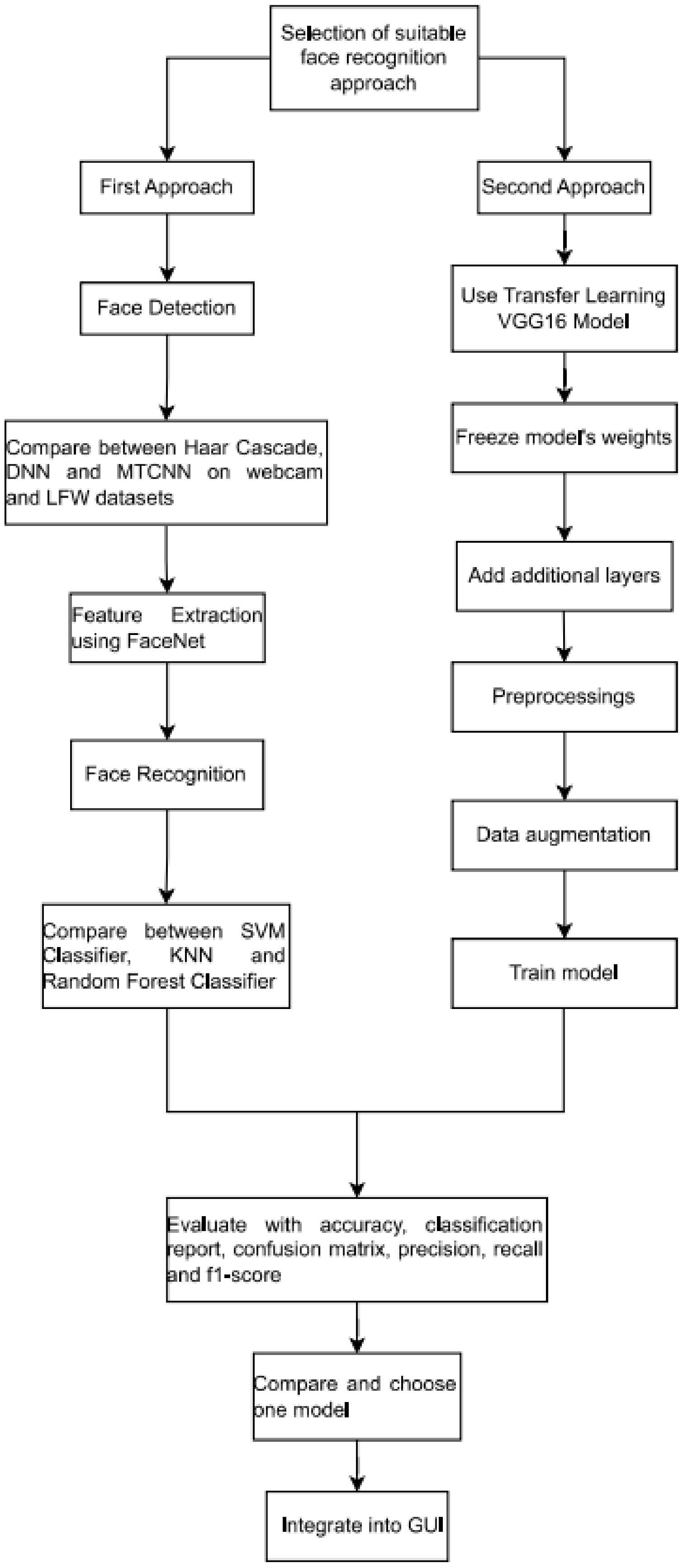

Selection of suitable face recognition approach
First Approach
Second Approach
Face Detection
Use Transfer Learning VGG16 Model
Compare between Haar Cascade, DNN and MTCNN on webcam and LFW datasets
Freeze model's weights
Add additional layers
Feature Extraction using FaceNet
Preprocessings
Face Recognition
Data augmentation
Compare between SVM Classifier, KNN and Random Forest Classifier
Train model
Evaluate with accuracy, classification report, confusion matrix, precision, recall and f1-score
Compare and choose one model
Integrate into GUI

Figure 3.2: Workflow for Face Recognition Approaches.

The designed smart attendance system is composed of three primary features: face detection, feature extraction, and face recognition, which collectively form the backbone of the system. The proposed system design is displayed in the block diagram in Figure 3.3 whereas the implementation of a more detailed flowchart is illustrated in Figure 3.4. The input image can be captured from a variety of environments such as university, company or industry by using a webcam or any video streaming device. The captured images will then be preprocessed and saved in the local files as the database. When a new face is added to the database, the model is trained to extract face encodings for the new user and associate them with their respective names. Next, face recognition takes place by classifying if the face encodings of an individual from real-time video streaming matched with the saved face encodings. If the comparison results are right, the individual's name will be displayed on the frame and the attendance excel sheet will be updated accordingly.

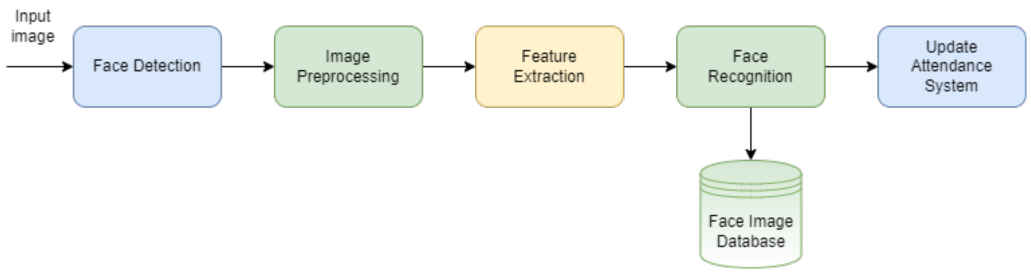


Figure 3.3: Block Diagram for Smart Attendance System.

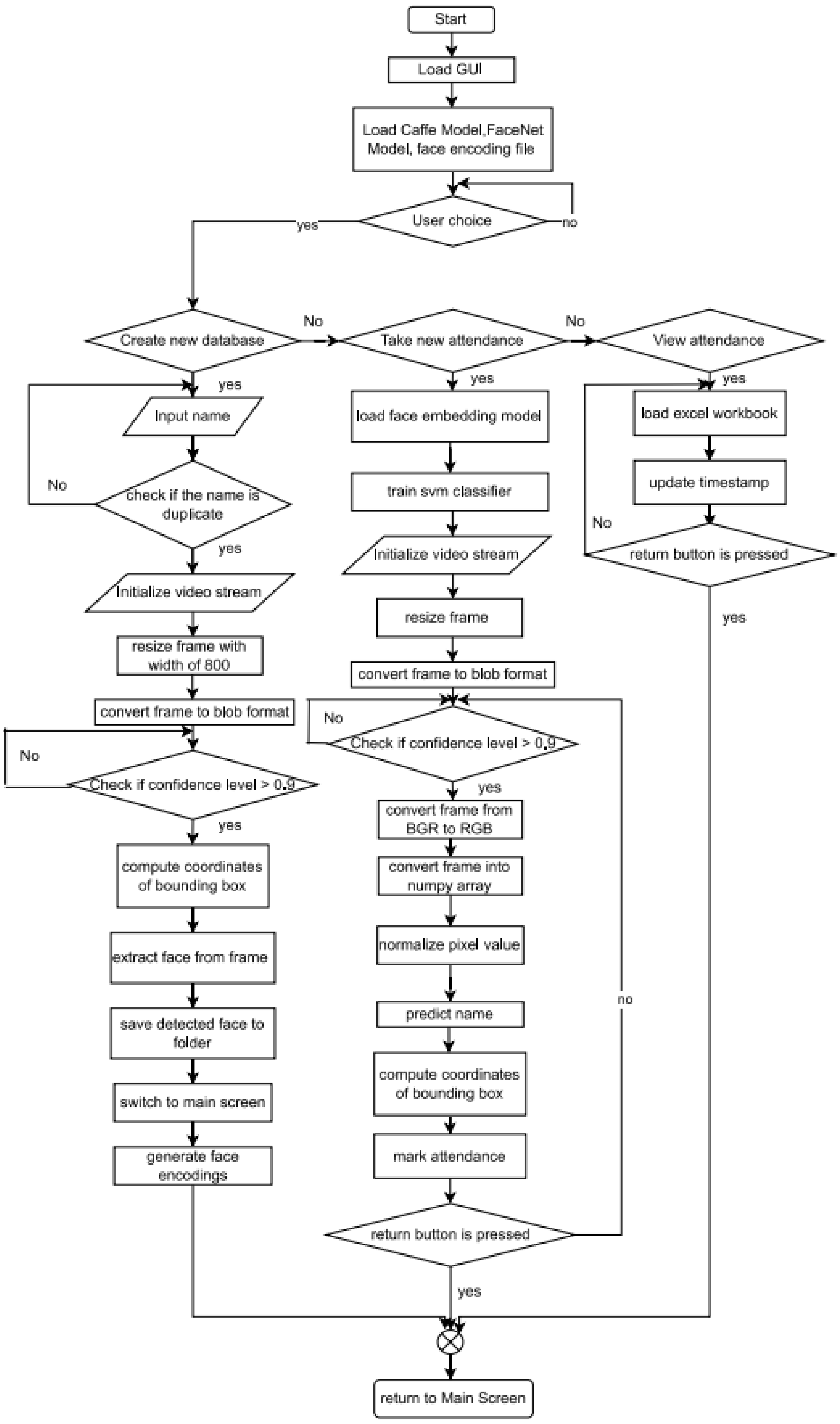


Figure 3.4: Flowchart of Smart Attendance System.

3.4 Resources

3.4.1 Hardware

The chosen webcam for the proposed system is Benewy Full High-definition (HD) webcam which can support full HD ranged from 272 pixels up to 1440 pixels with a maximum of 30 fps and 2 Megapixels. This plug and play webcam adapts advanced H.264 compression technology. Besides, it has built-in microphone and automatic focusing mode. It is being chosen as it fits the basic requirements of the face recognition system and the price is affordable. These crucial elements are considered before embedding the device to the proposed system. The specifications of this video-streaming webcam are listed out in Table 3.1.

Table 3.1: Specifications of Benewy Full HD Webcam.

Specifications	
Image	
Model Number	U01
Maximum Resolution	2560 × 1440 pixels
Pixels	2 Mega
Frame Speed	30 fps
Image Sensor	CMOS
Sensor Pixel	2 million

The developed system is processed using an Asus VivoBook S15 S510U laptop, equipped with an Intel i5-8250U CPU running at 1.60GHz and 8GB of installed Random-access memory (RAM) as shown in Figure 3.5.

Figure 3.5: Asus VivoBook S15 S510U Laptop.

3.4.2 Software

Python is the main programming language for the entire development of the smart attendance system. The development environment used is VS Code, an open-source code editor developed by Microsoft.

3.5 First Approach

3.5.1 Face Detection

Face detection is a preliminary step before proceeding into face recognition part. There are several algorithms and techniques that can be used for face detector. In this study, three widely recognized face detectors, namely OpenCV's Deep Neural Network (DNN) module, Multi-Task Cascaded Convolutional Networks (MTCNN), and Haar Cascade, were evaluated to determine the most appropriate one for the task at hand.

Firstly, OpenCV was installed as the requirements for the face detectors to work. It provides sets of useful tools for object detection and recognition. In this context, DNN makes use of 'prototxt' file and 'caffe' model as input to create a deep learning model for face detection. The architecture of the network including layers and configurations was specified in the 'prototxt' file whereas the weights for the network was defined in the 'caffe model' which can be found in Appendix A. For MTCNN, it can simply work by installing the MTCNN Library. To run Haar Cascade, the Haar Cascade XML file consisting of the face detection model was loaded to the system. Each detection technique has their own strengths and weakness and the results of comparison were analyzed in Chapter 4.

The three face detectors were compared and the most suitable one, DNN face detector was integrated into the smart attendance system. In the

Graphical User Interface (GUI), a “VideoStream” object was initialized to read frames from the camera source. The camera source was defined by a global variable named “CAMERA_INDEX” in which the user can easily change the camera source by setting “CAMERA_INDEX” to ‘1’ for external webcam or ‘0’ for the laptop’s built-in camera. In this context, a webcam was used to enhance the robustness of the system, as the webcam can produce a maximum of 2560 × 1440 pixels video stream at 25 fps whereas the built-in laptop web camera has a maximum resolution of 640 × 480 pixels.

The “start_detection()” function is called by a timer every 10 milliseconds to detect faces in the video stream. When the video stream is initialized, the frame will be resized to a width of 800 pixels. The height and width of the resized frame will be extracted out and converted to a blob, which matches the requirement for the processing of deep learning model. The blob is then passed through a pre-trained face detection model for detection. The confidence level of the current detection is extracted and if the confidence level is greater than the threshold which was preset using a variable named “CONFIDENCE_LEVEL”, the coordinates of the bounding box around the detected face will be computed. The extracted face will be saved in specific folder that is created using user’s name by using the “save_detected_face” function. Besides, the images are saved in JPEG format at maximum quality in order to preserve the sharpness and details of the image.

The timer is initialized when the video stream starts and stops when 20 faces have been captured. Collecting multiple frames of an individual’s face helps in enhancing the accuracy of the face recognition system by providing variations in lighting, angle, pose, and facial expressions. After few runs of tests, 20 frames were considered as the optimal number of frames to collect for creating a face database as it provides a good balance between capturing sufficient variation and does not overburden the system with too much data.

3.5.2 Feature Extraction

A deep learning model, FaceNet was deployed for face extraction and face recognition purpose. It takes an image of an individual’s face as input and outputs a vector of 128 numbers that represent the face embeddings. It was trained on the Tensorflow framework on a large dataset of face images for the use of face recognition. The FaceNet model and its weights are initially loaded and then compiled using the "compile" function. The model's filepath and its corresponding weights can be found in Appendix B.

After collecting 20 face images of the user, "switch_to_main()" releases the stream from a video object, creates an instance of "EncodingThread" with a batch size of 32, and starts the thread. The batch size is utilized in the "encode" method in order to loop through all collected images in batches instead of processing all the images at once. By using batches, the framework can optimize the computation graph for the specific batch size used. This can enhance memory efficiency and allow faster processing.

Face encodings will be generated from a set of training images that were saved earlier during the face detection process. Lists will be created to store the encodings and their respective names, followed by normalization of the image's pixel value. The images will be resized to 160×160 pixels and its dimensions will be expanded to fit the model. Predictions will be made on the batch of images by using the loaded FaceNet model and then the encodings and names will be appended to the lists. Lastly, the encoded data will be saved as a pickle file.

Furthermore, the face image encoding process is executed in a separate thread which means that the encoding process does not block the main thread of the application. By running the encoding process in a separate thread, the GUI remains responsive and the user can interact with the application or access other features while the encoding is in progress. In short, the encoding process was designed in a separate thread for better responsiveness and user experience without blocking the main thread of the ongoing application.

3.5.3 Face Classification

The "trainClassifier" function is called to load the encodings file path and Support Vector Machine (SVM) classifier from scikit-learn's "joblib" module to carry out face recognition. It then loads a dictionary of known face encodings and their respective names from a pickle file named "face_encoding.pickle". The known encodings and names are assigned to numpy arrays and the SVM classifier is trained on these known encodings and names. The trained classifier is then stored in “self.clf”. The “kernel” parameter in the SVM is set to “rbf” and the “tol” parameter was tuned to a value of 0.001 as stopping criterion. The regularization parameter was set with a value of 5. These parameters give the best performance for the system.

In this context, the SVM classifier was selected instead of other classifiers such as K-Nearest Neighbors (KNN), Random Forest and Decision Tree due to its simplicity, precision and speed.

After training on the known face encodings and their respective names, the SVM classifier is used to predict the name of new face encodings. Prediction of the name of each encoding is done by calling the “predict” method on the “clf” object. After that, the face encodings detected in the current frame is stored in the “face_encodings” variable whereas the predicted name is stored in the “predictions” variable. The predicted name will be displayed on the frame when the video is streaming.

The task of the real-time face recognition is performed in separate thread from the main GUI thread to prevent GUI from freezing easily. When the application is closed by user, the “stop()” method is called to stop the thread by setting the “ThreadActive” flag to False and thus quitting the thread.

3.6 Second Approach

The model architecture for VGG16 model is illustrated in Figure 3.6. The VGG16 model, originally developed by the Visual Geometry Group (VGG), is commonly used for image classification tasks. It consists of 16 layers and is often employed in transfer learning approaches for tasks such as face detection, feature extraction, and face recognition (Bansal, 2020). In this study, the pre-trained VGG16 model’s weights were frozen, and only additional layers were trained specifically for face detection and face recognition purposes.

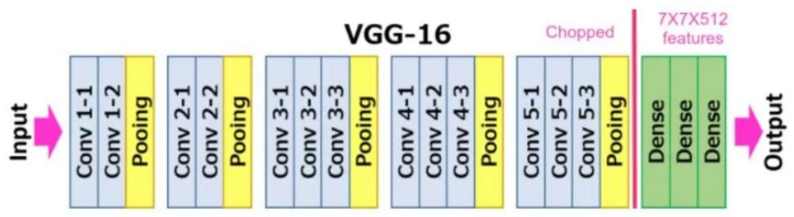


Figure 3.6: VGG16 Model Architecture (Bansal, 2020).

3.6.1 Model Architecture

The VGG16 model was loaded with pre-trained weights from the “imagenet” dataset, which comprises over 1.2 million images. The Keras functional API was used to add a new top layer to the VGG16 model. This top layer consists of a series of fully connected layers with an increasing number of units (256, 512, and 1024) and ReLU activation function. The final output layer was added with a number of units equal to the number of classes in the dataset, and softmax activation function was used for classification. The model was then compiled with “categorical_crossentropy” loss function, “adam” optimizer, and accuracy as the evaluation metric.

3.6.2 Data Augmentation

The “ImageDataGenerator” class was utilised for data augmentation on the training dataset to increase its size by applying various transformations to the images. For the training, validation and testing dataset, the images were rescaled in which each pixel value in the images is divided by 255, resulting in pixel values scaled down to the range of 0 to 1. Additionally, rotation range, width shift range, and height shift range were set to 20, 0.5, and 0.5 respectively. The horizontal flipping was set to “True” so that images can be horizontally flipped during training. “Nearest” mode was set to “fill_mode” such that empty pixels will be filled with the value of the nearest neighboring pixel.

3.6.3 Training

The model was trained using the “fit_generator()” function, which takes the training and validation data generators as inputs. During training, the weights of the pre-trained VGG16 model were frozen, so they were not updated. Only the weights of the new layers added for face recognition were trained. For the self-collected dataset, the number of epochs was set to 30 with a batch size of 16.

Overall, transfer learning with the VGG16 model, combined with pre-processing, model architecture, data augmentation, and training techniques, can enable effective face detection and recognition tasks. The evaluation results are analyzed in Chapter 4 to provide insights into the performance and effectiveness of the trained model.

3.7 Attendance Update

If the individual’s face is being recognized, an instance of the “Attendance” class will be created followed by the “mark_attendance” method. This step is carried out to mark attendance in the excel file named “Attendance_Records.xlsx”. In this context, the “xlsxwriter” library is used for creating and writing an Excel file whereas the “openpyxl” library is used for reading an existing Excel file. The “face_names_str” that is stored earlier during the face recognition process will be passed to the “mark_attendance” method followed by the “checkPreviousAttendance” method to check if a given “face_name_str” is already present in the Excel file within the last one minute. If the same name of an individual appeared within the last one minute, the name will not be recorded down. Otherwise, the name will be recorded in the Excel sheet with the timestamp.

3.8 User Interface Design

For the Graphical User Interface (GUI), PyQt5 was chosen to implement the smart attendance system due to its versatility, simplicity and powerful features that match with the objectives of this project. It provides a rich set of toolkits, widgets and other components to ease the design of a graphical user interface. For the developed smart attendance system, the GUI was designed using a variety of modules imported from PyQt5 libraries such as QPushButton, QDialog, QTableWidget, QTableWidgetItem, QLabel, and QStackedWidget.

3.9 Problems and Solutions

When running the initial prototype of the GUI, the system easily crashed, causing the GUI to become unresponsive. This is mainly due to bugs or coding errors in the application and insufficient resources as the GUI consumes too much memory and CPU. In the initial approach, the models were loaded separately in each class which resulted in loading similar models multiple times. This approach is not efficient as more memory is consumed and the processing time increases. Therefore, the codes were reviewed and few changes were made to enhance the robustness of the system.

One of the major enhancements that had been made was that all the required models for the smart attendance system are loaded once the GUI is initialized. The “models()” function is called to load FaceNet model with its pre-trained weight and the Caffe model and it ensures that all the required models are loaded and compiled successfully. In this context, when the user wants to access other features in multiple times, the models are not required to reload and this can save the processing time. As compared to initial approach of loading models in each class, the time to load all the required models had improved from 10 seconds to around 3.7 seconds.

Apart from that, the initial design of the system lacked of threading, which meant that operations had to be accomplished before proceeding to the next task. For instance, after the user captures images for creating a face database, the encoding operation must be completed before the user can access other program features. Therefore, threading is introduced to the system to allow the system become more responsive and efficient. Using threads enables the parallel processing of different tasks. By implementing threads, both tasks can be performed simultaneously so that after face capturing is completed, the user can still access other features such as face recognition or managing attendance records while the encoding process is carried out in a separate thread.

To implement threading, the signal-slot mechanism in PyQt was used to facilitate communication between different threads. A thread can emit a signal to the main thread, which can then respond to the signal by executing a slot function. This enables the main thread to be responsible for updating the GUI, while the worker thread is responsible for performing the long-running task. In short, threading is used for the execution of long-running tasks in the background, while keeping the GUI responsive and allowing the user to keep interacting with the application.

Additionally, the frames per second (FPS) of the webcam was fixed at 30 fps which induces less flexibility to the system. Adjustment of the FPS through coding was not working due to the limited capabilities of the webcam itself. Consequently, the attendance of an individual will be kept recorded based on every collected frames and this is not practical for an attendance system. To address this issue, the “checkPreviousAttendance” function is used to check if a similar name has appeared within the last minute in the attendance file. It will not create a new entry if the similar person has already checked in within the last minute. This is a good practice as it prevents duplicate records and avoiding errors so that the attendance records is up-to-date.

In short, these enhancements have greatly enhanced the system’s robustness and efficiency. By loading all required models at the initialization of GUI, implementing threading for the encoding and face recognition process and code optimization, the system can effectively handle the long-running tasks in the background while keeping the GUI responsive and thus providing a great user experience.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

The proposed systems' face recognition capabilities were tested on self-collected face datasets to evaluate their performance. A comparison was made between two approaches to choose the most suitable one for the attendance system. The face datasets were also used to fine-tune the parameters of the face recognition system. Testing the recognition system on face datasets allowed for the evaluation of metrics such as accuracy, precision, recall, F1 score, classification report, and confusion matrix. This process helped identify any limitations and possibilities in the system's ability to recognize faces for attendance records.

The graphical user interface of the smart attendance system was successfully developed using PyQT5. Face detection was performed using OpenCV's Caffe model, specifically the "Single Shot Detector" (SSD) framework. The system then used the pre-trained FaceNet model for feature extraction. The Support Vector Machine (SVM) classifier was applied to classify face encodings into names, and attendance was recorded in an Excel sheet if the user's face was successfully recognized.

4.2 Face Detection Evaluation

Face detection is an essential part of the whole recognition system, as it helps to detect face regions in the video frame so that later the recognition system can provide reliable recognition results. In this context, three famous face detector techniques, namely OpenCV Haar Cascades, OpenCV DNN, and MTCNN, had been tested out to analyze their performances in both Labeled Faces in the Wild (LFW) image datasets and real-time video captured from the webcam.

4.2.1 Labeled Faces In The Wild

LFW is a benchmark face recognition dataset that consists of more than 13000 images of faces from 5749 individuals collected from online website. The datasets are accomplished with a wide range of sizes and resolutions and variation in terms of pose, lighting and occlusion. The creation of this LFW dataset had been used in the field of face recognition by many researchers and studies because it reflected real-world scenarios where images were captured in uncontrolled environment. Each image from this LFW dataset had been detected and centered using the OpenCV with the implementation of the Viola-Jones face detector (Zhang and Deng, 2016). For the purpose of evaluating face recognition performance, a total of 300 images were randomly selected from the LFW dataset.

4.2.2 Performance Evaluation

The testing results for the face detectors are described in Table 4.1 to provide insights on the best face detector algorithms. The face detectors were tested on both LFW datasets and webcam. There are four different outcomes possible: true positive, true negative, false positive, and false negative. A true positive is achieved when all the faces in an image are correctly detected, while a true negative occurs when non-face images are correctly identified as such. False positive happens when the detection system flags an image or region as containing a face when it does not, and false negative occurs when a face is missed or not detected in the image or frame.

Table 4.1: Confusion Matrix for the Performance Evaluation of Face Detectors.

		Actual Conditions	
		Faces detected	**Faces not detected**
Predicted Conditions	**Predicted faces detected**	True Positive (TP)	False Positive (FP)
	Predicted non-faces detected	False Negative (FN)	True Negative (TN)

The accuracy of the results were evaluated with the following formula:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{4.1}$$

where

TP = True Positive

TN = True Negative

FP = False Positive

FN = False Negative

The speed of each detector was determined by measuring the elapsed time between two points: the start of model loading and the completion of image collection. The performance comparison in terms of accuracy and speed of three different approaches on the LFW dataset and laptop webcam for frontal faces, side faces, and low light environments is illustrated in Table 4.2, 4.3, 4.4, 4.5, and 4.6, respectively.

Table 4.2: Face Detection Approaches on Image Datasets.

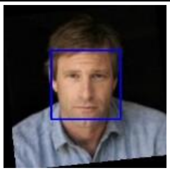
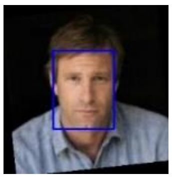
Approaches	**Test Image Example**	**Detected Image Example**	**TP**	**TN**	**FP**	**FN**
Open CV Haar Cascade			$\frac{274}{300}$	$\frac{0}{300}$	$\frac{21}{300}$	$\frac{5}{300}$
OpenCV Deep Neural Network			$\frac{296}{300}$	$\frac{0}{300}$	$\frac{2}{300}$	$\frac{2}{300}$
Multi-Task Cascaded Convolutional Neural Networks			$\frac{286}{300}$	$\frac{0}{300}$	$\frac{14}{300}$	$\frac{0}{300}$

Table 4.3: Comparison of Accuracy and Speed on LFW datasets.

Approaches	Accuracy	Speed
Open CV Haar Cascade	0.91	10.97 s
Open CV DNN	0.99	50.94 s
MTCNN	0.95	142.20 s

Table 4.4: Face Detection Approaches on Webcam for Frontal Faces.

Approaches	Detected Image Sample	Accuracy	Speed
OpenCV Haar Cascade		1.0	9 s
OpenCV Deep Neural Network		1.0	14.39 s
Multi-Task Cascaded Convolutional Neural Networks		1.0	12.80 s

Table 4.5: Face Detection Approaches on Webcam for Side Faces.

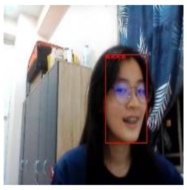
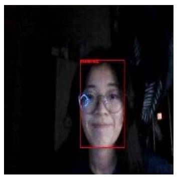
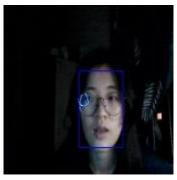
Approaches	Detected Image Sample	Accuracy	Speed
OpenCV Haar Cascade		0.97	11.75 s
OpenCV Deep Neural Network		1.0	14.88 s
Multi-Task Cascaded Convolutional Neural Networks		1.0	16.30 s

Table 4.6: Face Detection Approaches on Webcam in Low-light Environment.

Approaches	Detected Image Sample	Accuracy	Speed
OpenCV Haar Cascade		0.82	6.23 s
OpenCV Deep Neural Network		1.0	14.18 s
Multi-Task Cascaded Convolutional Neural Networks		0.95	18.01 s

From the analysis of the above results, the OpenCV Haar Cascade classifier yielded the worst results with an accuracy of 91 % when tested on the LFW dataset. On the other hand, the DNN approach outperformed the other two face detection techniques in terms of both accuracy and computational speed. The DNN module achieved 100 % accuracy in real-time webcam streaming applications, even when faced with different angles of faces and lighting conditions. When applied to the LFW dataset, the DNN module achieved an accuracy of 99 % with an average execution time of 50.94 s. Although MTCNN could perform well with good accuracy, its computational resource consumption was a constraint. Furthermore, the DNN module demonstrated its ability to detect individuals in unconstrained environments that varied in terms of lighting, pose, occlusion, and other factors, including cases where individuals were wearing face masks. Therefore, for a smart attendance system where real-time performance is critical, the DNN module was chosen as the face detector for the smart attendance system.

4.3 Face Recognition Evaluation

The purpose of carrying out the evaluation using face datasets was to evaluate the performance and accuracy of the designated face recognition system. In real-time face recognition, the system continuously processes video stream and makes predictions of the individual's name but it is impossible to determine whether the correct label of the person is being recognized. The challenge faced is that human operators have to manually verify the recognition results. Therefore, evaluation on different datasets could help identify strengths and weaknesses of the system apart from measuring its performance.

During the evaluation process, two different models were used on similar self-collected face datasets to ensure a fair comparison between the performance of the face recognition system. The first model is a combination of OpenCV DNN, FaceNet model, and SVM classifier. The second model is a VGG16 transfer learning model, which was pre-trained on a large dataset of images and fine-tuned on the face recognition task.

4.3.1 Face Datasets

The face datasets were collected with a total of 280 images from 14 subjects in an indoor environment using the video stream. The frames were resized to a width of 800 pixels for each image and then converted to a blob using the "cv2.dnn.blobFromImage()" method. The blob was passed through a deep learning model to detect faces in the frame, and only those with a confidence level greater than 0.9 were saved. The collected datasets were then separated into training, validation, and testing sets with ratios of 0.7, 0.15, and 0.15, respectively.

For the first approach, face detection was applied to all sets, and the loaded FaceNet model was used to extract encodings. This process resulted in the creation of three separate files containing the encoding information for the face datasets. For the training sets, data augmentation was applied to enhance system robustness by exposing the face images to a wider range of face variations. In this context, a rotation degree of 25, minimum scale of 0.6, and maximum scale of 1.0 were applied to the face images. The encodings were flattened and padded with zeros to ensure that all samples had the same number of features. The names of the face images were encoded using 'LabelEncoder' to transform categorical data into numerical data that machine learning models can process. The training sets' encodings, validation sets' encodings, and testing sets' encodings with their respective names were stored as 'trainX', 'trainY', 'testX', 'testY', 'valX', and 'valY', respectively.

The encodings and labels were reshaped to ensure the correct format for the machine learning algorithm to process. The SVM classifier expects labels to be in a specific shape of a one-dimensional array, so the label arrays were reshaped to have one column and as many rows as necessary. Similarly, the feature arrays were flattened to have one column and as many rows as necessary. Mean imputation was performed on the data to replace missing values with the mean of the feature column. The 'fit_transform' method was applied to the training data to compute the mean and replace missing values with it. The transform method was then used on the validation and test data to replace missing values with the same mean value calculated from the training data. The imputation was done separately on the training, validation, and test sets to avoid data leakage.

For the VGG16 transfer learning model, the image datasets were initially preprocessed by resizing each input shape of the images to 224×224 pixels with three channels RGB before being fed into the model. Data normalization was then performed to rescale the pixel values of the input images between zero and one using the rescale parameter in ImageDataGenerator. This step is important for better performance and faster convergence. Finally, data augmentation was performed on the training images by applying random transformations to the images with a rotation range of 20, width and height shift values of 0.5, and horizontal flip.

4.3.2 Parameters

The selection and tuning of model parameters are crucial aspects of building and training for machine learning and deep learning models.

In the first approach, face detection was initially performed, and the confidence threshold was adjusted to balance the trade-off between precision and recall. A low threshold could result in identifying non-faces as faces, while a high threshold could lead to missing some faces. To ensure consistency in performance evaluation between datasets and real-time face recognition, a confidence threshold of 0.9 was set, indicating that only faces with a confidence score above 0.9 were considered valid.

Regarding the SVM classifier, few parameters were tuned as the classifier's performance heavily relied on their values. The 'C' value was set to

5, which controlled the trade-off between classification error minimization and maximization. Setting it to a higher value than 5 could lead to overfitting and reduced classifier accuracy. Additionally, the ‘kernel’ parameter was set to the radial basis function (RBF) kernel, which had been shown to provide better classifier performance. The tolerance rate was set to 0.001, and the ‘probability’ parameter was set to ‘True’ to enable probability estimates for each predicted class. This is a useful feature that provides additional information about the classifier’s confidence in its predictions.

For the second approach, which utilized the VGG16 transfer learning model, two key parameters were identified: batch size and epoch, which had a significant impact on the model’s performance. Batch size is the number of training examples utilized in one forward or backward pass. In this VGG16 model, the batch size was set to 16. From the few experiments of setting different values of batch size, it was found that a larger batch size allows for faster training and better accuracy. Conversely, smaller batch sizes yield faster computation but also indicate fewer updates per training iteration.

The term epoch refers to going through the complete training dataset once. Increasing the number of epochs could result in better accuracy but carried with the risk of overfitting such that the model performs well on training sets but poorly on unseen data. In this case, the VGG16 model was trained for 30 epochs, and after each epoch, the model was evaluated on a separate validation set. To prevent overfitting, the “EarlyStopping” callback function was utilized to stop the training early if the monitored metric stopped improving. The validation accuracy was set as the “monitor” parameter for early stopping. Additionally, the model was checkpointed after each epoch to ensure that the best performing model was saved. For this self-collected face datasets, the training was stopped at epoch 25, and the model was saved.

4.3.3 Evaluation Metrics

The first metric that needs to be evaluated is the accuracy of the recognition system. Accuracy represents the percentage of correctly classified instances out of all instances in the dataset. To measure the accuracy of the classifier, the ‘accuracy_score’ function from the scikit-learn library was imported. This function takes the true target labels and the predicted classes as inputs and returns the accuracy of the classifier.

After evaluating the combined models of DNN, FaceNet, and SVM on the self-collected datasets, the training accuracy, validation accuracy, and testing accuracy were found to be 98.98 %, 100 %, and 97.62 %, respectively. The high accuracy on the training sets suggests that the SVM classifier learned the training data very well and could predict the correct class for the majority of the training instances. The validation accuracy of 100 % indicates that the model was performing perfectly on the validation dataset without any errors or misclassifications. However, a higher accuracy on validation set than training accuracy and testing accuracy could be explained by less noise in samples due to small size of the validation samples used. The overall evaluation comparison between the two models is presented in Table 4.7.

Table 4.7: Evaluation Comparison between Two Models.

	DNN, FaceNet and SVM	**VGG16 Transfer Learning Model**
Training Accuracy	0.9898	0.8622
Validation Accuracy	1.0	0.9762
Testing Accuracy	0.9762	0.9762
Training Loss	-	0.3733
Validation Loss	-	0.0916
Testing Loss	-	0.0844
Precision	0.9821	0.9821
Recall	0.9762	0.9762
F1 Score	0.9755	0.9755

Apart from SVM, other machine learning algorithms such as K-Nearest Neighbors (KNN) and Random Forest were tested to get a comparison of the accuracy score as illustrated in Table 4.8.

Table 4.8: Test Accuracy On Different Classifiers.

Classifier	**Test accuracy (%)**
Support Vector Machine (SVM)	97.62
K-Nearest Neighbors (KNN)	95.24
Random Forest Classifier	95.24

Among the three tested classifiers, SVM had exhibited superior performance in term of test accuracy. It had better accuracy of 97.62 %. On the other hand, KNN and Random Forest Classifier obtained an accuracy of 95.24 %. Therefore, it had been decided that SVM would be the optimal choice for the current face recognition project as it was suitable for multi-class classification tasks and its ability to handle high-dimensional data.

During the training process, the VGG16 model calculated the accuracy and loss at each epoch. Accuracy refers to the percentage of images that the model correctly classifies, while loss is a metric for measuring model performance during training. The goal of training was to minimize loss and maximize accuracy. The accuracy graph in Figure 4.1 shows that accuracy increased as the number of epochs increased, indicating the model improved its ability to predict class labels. While the Figure 4.2 displays the model loss on training set and validation set for each epoch. The model was trained for 30 epochs, but the training process stopped at the 25th epoch. The model achieved a training accuracy of 0.8622 and a training loss of 0.3733. Validation data was evaluated after each epoch, and the best validation loss achieved was 0.07984, but it did not improve at the 25th epoch, so the training was stopped early to prevent overfitting. The model achieved a validation loss of 0.0916 and a validation accuracy of 0.9762. The test accuracy of 0.9762 indicates that the model predicted well on new, unseen data.

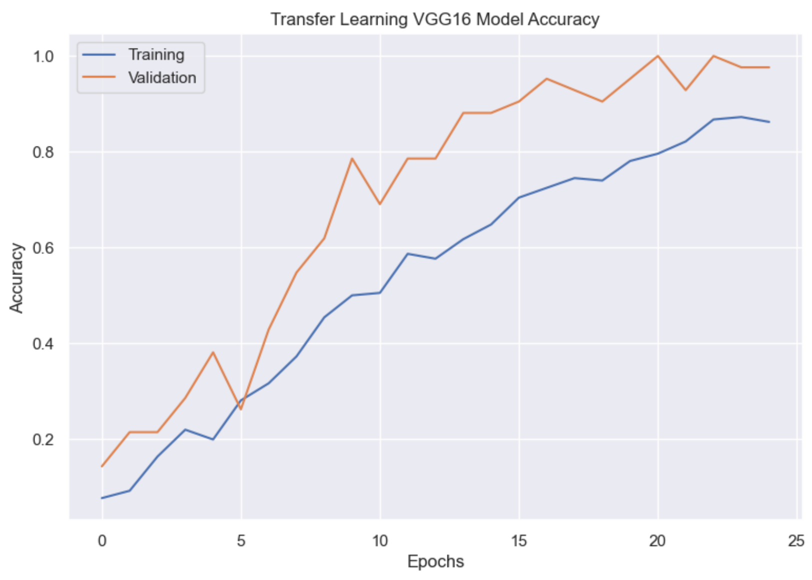


Figure 4.1: Transfer Learning VGG16 Model Accuracy at Each Epoch.

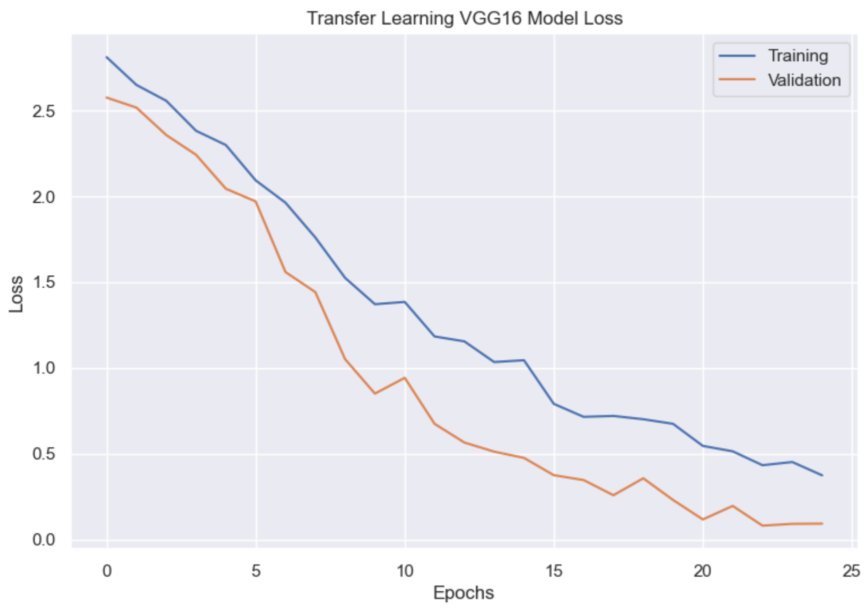


Figure 4.2: Transfer Learning VGG16 Model Loss at Each Epoch.

The reason why validation accuracy and test accuracy are higher than train accuracy is due to the validation set, which is too small to adequately represent the probability distribution of the data. Besides, a smaller splitting ratio of validation and testing ratios may cause this, as the model could perform prediction easily on the unseen data. Another reason is the data augmentation techniques such as rotation, flipping, and scaling used to increase the dataset's size, which helped the model generalize better on new data.

Apart from that, the classification report was used to provide a summary of the precision, recall and F1 score for each sample in the datasets. The metrics were measured using the true positive (TP), false positive (FP), true negative (TN), and false negative (FN) values for each class. The classification report for both models are displayed in Table 4.9 and Table 4.10. From the analysis, the first model achieved a good accuracy of 0.9762 However, it had low scores for some classes, particularly for class 2, which had a precision of 0.75 and f1-score of 0.86. Class 5 also obtained a recall score of 0.67 and f1-score of 0.80. On the other hand, all the classes in VGG16 transfer learning model had perfect precision, recall, and F1-score, except for class 7 and class 12, which had slightly lower scores. Class 7 obtained precision score of 0.75 and f1-score of 0.86 whereas Class 12 had a recall score of 0.67 and f1-score of 0.80. Therefore, the model appeared to perform very well in this classification task.

Overall, the differences in the comparison of results were not significant, the classification report suggested that both had the similar average accuracy of 0.98.

Table 4.9: Classification Report for DNN, FaceNet and SVM.

Name labels	**precision**	**recall**	**f1-score**	**support**
0	1	1	1	3
1	1	1	1	3
2	075	1	0.86	3
3	1	1	1	3
4	1	1	1	3
5	1	0.67	0.80	3
6	1	1	1	3
7	1	1	1	3
8	1	1	1	3
9	1	1	1	3
10	1	1	1	3
11	1	1	1	3
12	1	1	1	3
13	1	1	1	3
accuracy	-	-	0.98	42
macro avg	0.98	0.98	0.98	42
weighted avg	0.98	0.98	0.98	42

Table 4.10: Classification Report for VGG16 Transfer Learning Model.

Name labels	precision	recall	f1-score	support
0	1	1	1	3
1	1	1	1	3
2	1	1	1	3
3	1	1	1	3
4	1	1	1	3
5	1	1	1	3
6	1	1	1	3
7	0.75	1	0.86	3
8	1	1	1	3
9	1	1	1	3
10	1	1	1	3
11	1	1	1	3
12	1	0.67	0.80	3
13	1	1	1	3
accuracy	-	-	0.98	42
macro avg	0.98	0.98	0.98	42
weighted avg	0.98	0.98	0.98	42

The resulting heatmap, as displayed in Figure 4.3 and Figure 4.4 showcases the confusion matrix generated by the SVM classifier and VGG16 classification model on the face dataset. The rows and columns in the heatmap represent the actual and predicted class labels, respectively. The diagonal matrix indicates that the number of instances that were correctly classified for each class, while the off-diagonal values indicated misclassified instances. The confusion matrix included 14 classes (0 to 13), and the labels were displayed in numeric form due to their encoding using the 'Label Encoder' method. Figure 4.3 reveals that the SVM model accurately classified all instances except for class 6. For instance, one instance that actually belonged to class 6 was misclassified as class 3. In contrast, the VGG16 classification model had correctly classified all instances in each class, except for class 13 as shown in Figure 4.4. One instance that belonged to class 13 was predicted as class 8. The results indicate that both the VGG16 model and the SVM model performed well on this dataset. However, it is possible that the dataset used to generate this confusion matrix was very small, which causes the possibility that it may not be representative of the entire population.

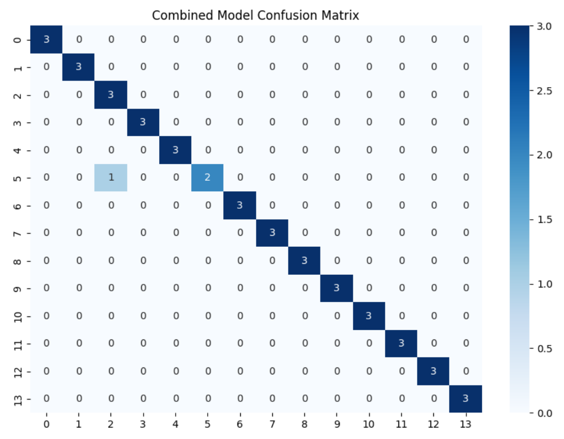


Figure 4.3: Confusion Matrix for DNN, FaceNet and SVM.

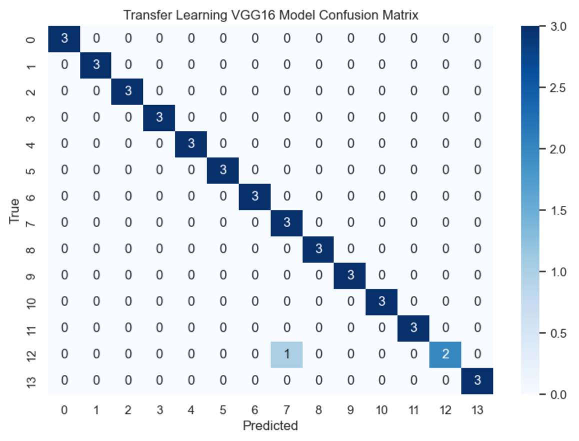


Figure 4.4: Confusion Matrix for VGG16.

4.4 Smart Attendance System

When initializing the GUI, the smart attendance system required approximately 7 seconds to load the necessary models such as FaceNet and OpenCV's Deep Neural Network module for its operation for the first time. After the initial load, the average loading time decreased to 3.8 seconds.

4.4.1 Graphical User Interface

The Figure 4.5 displays an User Interface (UI) which was designed to associate with three main features which are creating new face database or identity, face recognition and attendance monitoring. It can be accessed on the main window screen named “WelcomeScreen”. It was implemented using a stacked widget to switch between different screens. Buttons were used to connect to different screens or implement the methods. For instance, the “pushButton”, “pushButton_2” and “pushButton_3” buttons were defined in the “HomeScreen.ui” file and connected to the “gotoadd”, “gotoadd2” and “gotoadd3” methods, respectively, using the “clicked.connect” method. The methods will prompt screen switching or events triggering. In the homescreen, the “pushButton”, “pushButton_2” and “pushButton_3” buttons were linked to “CreateIDScreen”, “FaceRecogScreen” and “AttendanceScreen” respectively.

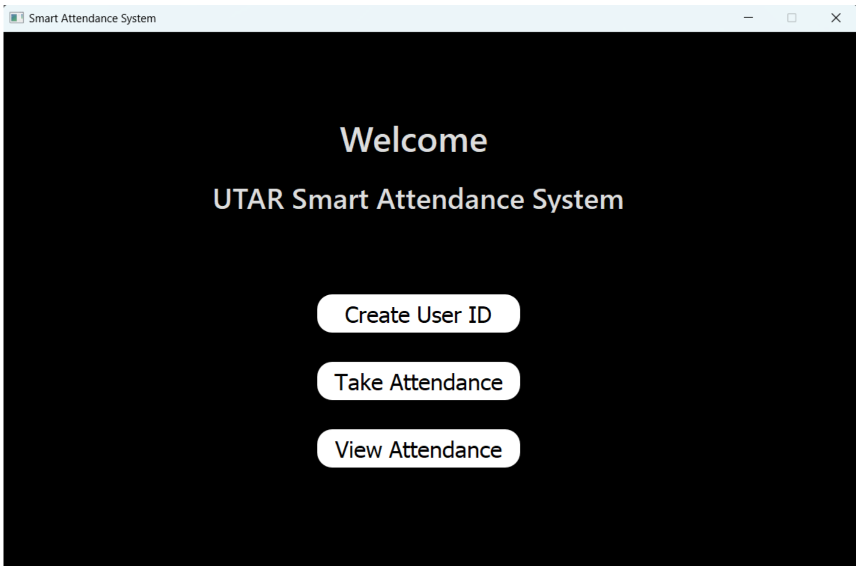


Figure 4.5: UI Design for Main Screen.

As shown in Figure 4.6, the ‘CreateIDScreen’ class was used to input the user’s name for creating new database and it was defined using the ‘QDialog’ widget. It was equipped with two ‘QPushButton’ widgets, one to proceed to the next screen and the other to return to the main screen. On top of that, the ‘Next’ button was connected to the ‘check_name_input’ function, which checks if the name is valid or not. A warning message, ‘Please enter name to proceed’ will be displayed if the user does not enter any alphabet and just presses the button to proceed, whereas a warning message of ‘Name already exists’ will be shown if the input name is duplicate.

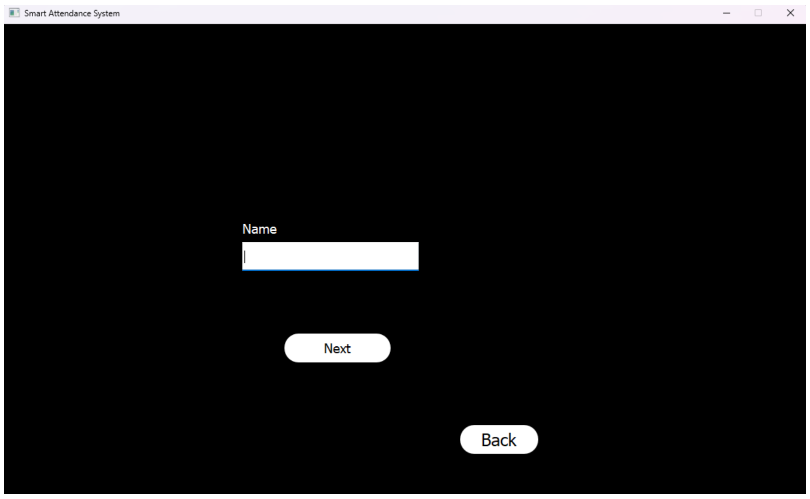


Figure 4.6: Create ID Screen

If the input informations are valid, the program will prompt the user to switch to the screen for capturing their image and extracting their face encodings. Once the first face is detected, a timer will start running and will stop once the 20th face is captured. While 20 images may be considered a small number for a face recognition system, it can still be sufficient to recognize an individual and help to reduce storage requirements. A message will indicate that the face has been captured, and the screen will switch to the main window where the face encodings process will be initiated. On average, it takes approximately 12 seconds to train 20 images with face encodings.

The second feature of the GUI is the face recognition system as displayed in Figure 4.7. In order to implement this, the code was structured into two classes named “FaceRecogScreen” which is responsible for setting up the GUI and managing the video stream and “StartThread” for processing the video stream and recognizing the faces. The face recognition process is continuous until the “Cancel” button is pressed by the user to stop the video stream and it will prompt the screen to return to the main window.

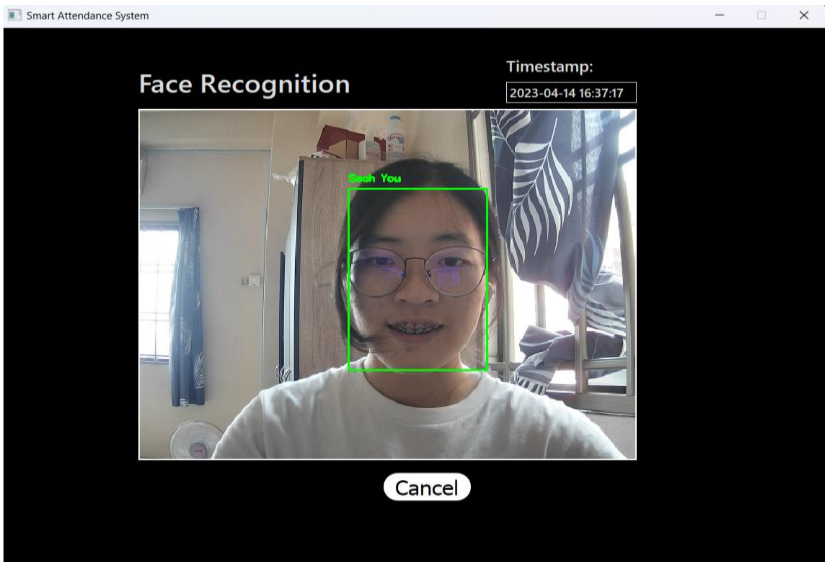
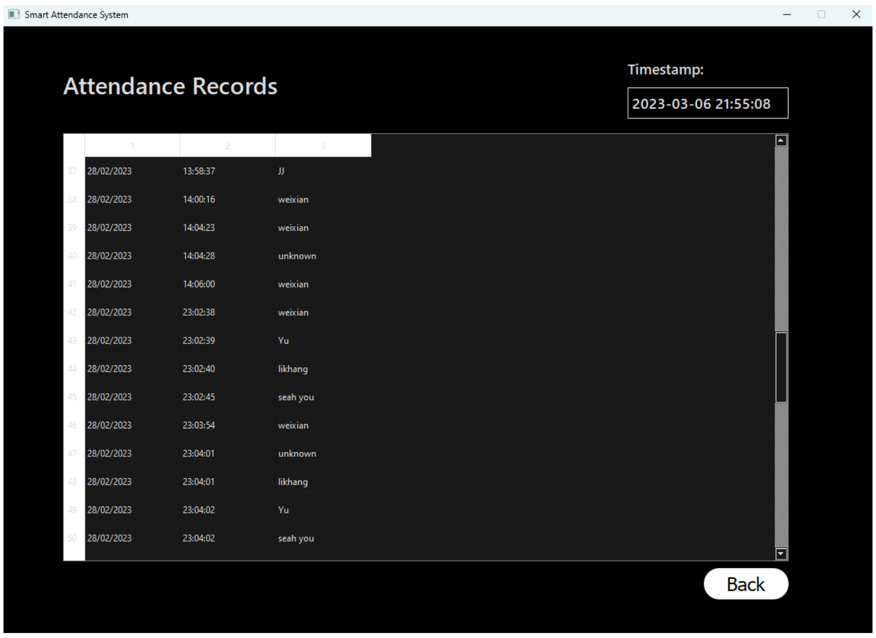


Figure 4.7: Face Recognition Screen.

In the GUI, the "update_date_time" method was called to update the current date and time in the format of "YYYY-MM-DD HH:MM:SS" on a text label widget. By displaying the timestamp on the user interface, it provides transparency to the system user, allowing users to verify whether the system is functioning properly and capturing the attendance information in a timely manner. Furthermore, the user can easily check the current time without accessing it through external devices such as a smartphone or watch.

The third main feature was implemented on the "Attendance Screen" in which user gets to view the attendance records as shown in Figure 4.8. A "QTableWidget" object from the "PyQt5.QtWidgets" module was used to represent the table for displaying data. This feature provides convenience to the user or admin to visualize and manage the attendance data. Furthermore, necessary changes or edits can be made by just accessing the excel sheet if required. It would be helpful in reports generation or data analyzing for the organisations that implement the smart attendance system.



Attendance Records

Timestamp:

2023-03-06 21:55:08

	1	2	3
37	28/02/2023	13:58:37	JJ
38	28/02/2023	14:00:16	weixian
39	28/02/2023	14:04:23	weixian
40	28/02/2023	14:04:28	unknown
41	28/02/2023	14:06:00	weixian
42	28/02/2023	23:02:38	weixian
43	28/02/2023	23:02:39	Yu
44	28/02/2023	23:02:40	likhang
45	28/02/2023	23:02:45	seah you
46	28/02/2023	23:03:54	weixian
47	28/02/2023	23:04:01	unknown
48	28/02/2023	23:04:01	likhang
49	28/02/2023	23:04:02	Yu
50	28/02/2023	23:04:02	seah you

Figure 4.8: Attendance Screen.

4.5 Summary

Both the combined model consisting of a DNN module, FaceNet, and SVM classifier, and the VGG16 transfer learning model, exhibited excellent results with the testing accuracy of 97.62 % and 97.619 % respectively on the self-collected face dataset. The minimal difference between the two models suggested that pre-trained models could be effectively utilized to achieve outstanding performance for face recognition tasks. On top of that, the deep learning model would require retraining to recognize a new individual's face if a new database was created. This could be a time-consuming process, especially for larger organizations that regularly add many new faces to their attendance system. Therefore, the recognition task utilized a combination of SSD, FaceNet, and SVM and integrated them into the designated GUI for the smart attendance system.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

In conclusion, all the project’s aim and objectives were successfully accomplished, which included developing a facial recognition system designed specifically for attendance tracking. A comprehensive literature review covering current approaches and results in facial recognition algorithms was conducted. Furthermore, an intuitive GUI with essential features was created to simplify attendance monitoring for end-users. Finally, the system’s performance and usability were analyzed, providing insight for future enhancements to enhance the system’s robustness.

The study proposed and tested three different face detection approaches, which are OpenCV Haar cascades, OpenCV DNN, and MTCNN. Their performances were evaluated on both the Labeled Faces in the Wild (LFW) image datasets and real-time video captured from the webcam. The results showed that the DNN approach outperformed the other two techniques with an accuracy of 99 % and computational time of 57.94 s. As a result, it was chosen as the face detector for the face recognition system. To enable face recognition, two distinct models were utilized in which the first model combined OpenCV DNN, FaceNet model, and SVM classifier, while the second model is a VGG16 transfer learning model. The evaluation results showed minimal differences between the two models with test accuracies of 97.62 % and 97.619 % respectively, indicating that pre-trained models and the use of machine learning algorithms, such as SVM, can achieve outstanding performance for face recognition tasks. Finally, the recognition task was implemented using a combination of DNN, FaceNet, and SVM, which were integrated into a designated GUI with features such as creating a new database, taking attendance, and monitoring attendance.

5.2 Recommendations for Future Work

Although the presented testing accuracy of the face recognition system is high, it is important to note that the self-collected dataset used to train and test the system is small. Therefore, it is possible that the system's performance may not generalize well to other datasets. To address this limitation, future work could involve collecting more diverse and large training datasets to evaluate the scalability and generalizability of the system's performance. Additionally, the robustness of the face recognition system can be enhanced for future work, enabling it to recognize faces under different conditions, such as varying illumination, occlusion, pose, and masked faces.

Ensemble learning and adaptive learning techniques can be effectively employed to enhance the accuracy and robustness of a face recognition system. By using ensemble learning, multiple models can be used to improve the system's ability to learn from the strengths and weaknesses of each model. Conversely, adaptive learning can be integrated into the system to ensure that recognition remains effective even as the user ages. Furthermore, incremental learning can be applied to deep learning models, enabling them to continuously learn from data and adapt to changes in that data over time.

On top of that, the ethical issues surrounding face recognition technology should be considered to protect users privacy and information. Therefore, collaborating with experts in related fields such as social science, government, and police and having a great understanding of the law concerning humanity's privacy can provide valuable insight into these ethical concerns.

For the attendance system, the current work involves saving new users databases into local files. One way to improve the system's effectiveness and data security is to upload the users databases to a centralised cloud server. For instance, Amazon Web Services (AWS), which supports MySQL and Oracle, Microsoft Azure, which supports Microsoft SQL Server, and Firebase, which enables the storing of real-time data. All of these cloud server options guarantee high levels of security, ensuring that only authorised personnel can access the database (Rashid, 2022). Additionally, this can facilitate data sharing and management for the attendance monitoring of an organisation.

Finally, the current graphical user interface for the attendance system was developed using PyQt5, which is suitable for desktop applications. To enhance the efficient use of the smart attendance system at public organizations, other frontend frameworks such as Angular and React can be utilized. Angular, developed by Google, offers a range of features that make it convenient for building complex applications. React, developed by Facebook, allows for an efficient user interface with a virtual Document Object Model (DOM) that updates only the necessary components, resulting in faster performance (Patel and Tere, 2014). Both frameworks provide a wide range of useful features that are suitable for developing an efficient smart attendance system.

REFERENCES

Abdalkarim, B.A. and Akgün, D., 2022. A Literature Review on Smart Attendance Systems. *International Conference on Applied Engineering and Natural Sciences A Literature Review on Smart Attendance Systems*. [online] Available at: <https://www.researchgate.net /publication/327019394_Attendance_ Management_System_for_Educational_Sector_Critical_Review> [Accessed 20 April 2023].

Akshaya, R.S., Devi, K., Juhidha, J., Priyanka, R. and Kanimozhi, R., 2021. IOT based Face Recognition Smart Attendance System using ESP 32 cam. *International Journal of Innovative Research in Science, Engineering and Technology*, [e-journal] 10(5). https://doi.org/10.15680/IJIRSET.2021.1005214.

Alexiadis, D., Syrris, V., Papastergiou, A., Hatzigaidas, A. and Mariuta, L., 2010. A new face database and evaluation of face recognition techniques. *International Conference on Systems - Proceedings*, pp.590–595.

Amirulloh, I., Iskandar, I.D., Apriyani, Y., Warnilah, A.I., Purnia, D.S. and Surahman, M., 2020. Teacher Attendance Monitoring System Teaching with QR-Code and Geo Location using Android Platform. *Journal of Physics: Conference Series*, [e-journal] 1641(1), pp.1-2. https://doi.org/10.1088/1742-6596/1641/1/012030.

Amraee, S., Chinipardaz, M. and Charoosaei, M., 2022. Analytical study of two feature extraction methods in comparison with deep learning methods for classification of small metal objects. *Visual Computing for Industry, Biomedicine, and Art*, [e-journal] 5(1), pp.9-11. https://doi.org/10.1186/s42492-022-00111-6.

Anon. 2014. Face Recognition Using Kernel PrincipalComponent Analysis. *Advances in Vision Computing: An International Journal*, 1(1), pp.1–9.

Apple, 2017. *Face ID Security*. [online] Available at: <https://www.apple.com/ca/business-docs/FaceID_Security_Guide.pdf> [Acc essed 3 October 2022].

Arora, M., Naithani, S. and Areeckal, A.S., 2022. A web-based application for face detection in real-time images and videos. *Journal of Physics: Conference Series*, [e-journal] 2161(1), pp2-9. https://doi.org/10.1088/1742-6596/2161/1/012071.

Arya, S., Mesariya, H. and Parekh, V., 2020. Smart Attendance System Usign CNN. *CoRR* [e-journal] (1), pp.2–5. https://doi.org/https://doi.org/10.48550/arXiv.2004.14289.

Atabansi, C.C., Chen, T., Cao, R. and Xu, X., 2021. Transfer Learning Technique with VGG-16 for Near-Infrared Facial Expression Recognition. *Journal of Physics: Conference Series*, [e-journal] 1873(1), pp.2-9. https://doi.org/10.1088/1742-6596/1873/1/012033.

Bally, M., 2002. Technology trends. *Glass Digest*, 81(7–8), p.12.

Bansal, M., 2020. *Face recognition using Transfer learning and VGG16 | by Megha Bansal | Analytics Vidhya | Medium*, [online] Available at: <https://medium.com/analytics-vidhya/face-recognition-using-transfer-learning-and-vgg16-cf4de57b9154> [Accessed 7 April 2023].

Bharat Chandra, Y. and Karthikeya Reddy, G., 2020. A Comparative Analysis Of Face Recognition Models On Masked Faces. *International Journal of Scientific & Technology Research*, [online] Available at: <www.ijstr.org> [Accessed 3 October 2022].

Brandon Amos, Bartosz Ludwiczuk, M.S., 2016. Openface: uniwersalna biblioteka rozpoznawania twarzy z aplikacjami mobilnymi. *National Science Foundation (NSF),* [online] Available at: <http://reports-archive.adm.cs.cmu.edu/anon/anon/usr0/ftp/2016/CMU-CS-16-118.pdf>[A ccessed 3 October 2022].

Chai, X., Shan, S. and Gao, W., 2003. Pose normalization for robust face recognition based on statistical affine transformation. *ICICS-PCM 2003 - Proceedings of the 2003 Joint Conference of the 4th International Conference on Information, Communications and Signal Processing and 4th Pacific-Rim Conference on Multimedia*, [e-journal] pp.1413–1417. https://doi.org/10.1109/ICICS.2003.1292698.

Chang-yeon, J., 2008. Face Detection using LBP features. *Final Project Report*, *77*, pp.1-4.

Chatrati, S., Naidu, S. and Prasad, C.R., 2013. RFID based student monitoring and attendance tracking system. *2013 4th International Conference on Computing, Communications and Networking Technologies, ICCCNT 2013*, [e-journal] pp.3. https://doi.org/10.1109/ICCCNT.2013.6726702.

Chaudhari, M.N., Deshmukh, M., Ramrakhiani, G. and Parvatikar, R., 2018. Face Detection Using Viola Jones Algorithm and Neural Networks. *Proceedings - 2018 4th International Conference on Computing, Communication Control and Automation, ICCUBEA 2018*, [e-journal] pp.1-6. https://doi.org/10.1109/ICCUBEA.2018.8697768.

Chaves, D., Fidalgo, E., Alegre, E., Alaiz-Rodríguez, R., Jáñez-Martino, F. and Azzopardi, G., 2020. Assessment and estimation of face detection performance based on deep learning for forensic applications. *Sensors (Switzerland)*, [e-journal] 20(16), pp.1–21. https://doi.org/10.3390/s20164491.

Da'San, M., Alqudah, A. and Debeir, O., 2015. Face detection using Viola and Jones method and neural networks. *2015 International Conference on Information and Communication Technology Research, ICTRC 2015*, [e-journal] (1), pp.40–43. https://doi.org/10.1109/ICTRC.2015.7156416.

Dadi, H.S. and Mohan Pillutla, G.K., 2016. Improved Face Recognition Rate Using HOG Features and SVM Classifier. *IOSR Journal of Electronics and Communication Engineering*, [e-journal] 11(04), pp.34–44. https://doi.org/10.9790/2834-1104013444.

Dalali, S. and Suresh, L., 2016. Daubechives Wavelet Based Face Recognition Using Modified LBP. *Procedia Computer Science*, [e-journal] 93, pp.344–350. https://doi.org/10.1016/j.procs.2016.07.219.

Deshpande, N.T. and Ravishankar, S., 2016. Face Detection and Recognition using Viola-Jones algorithm and fusion of LDA and ANN. *IOSR Journal of Computer Engineering*, [online] Available at: <https://pdfs.semanticscholar.org/c5cf/c1f5a430ad9c103b381d016adb4cba20ce4e.pdf>.

Dey, S., Barman, S., Bhukya, R.K., Das, R.K., Haris, B.C., Prasanna, S.R.M. and Sinha, R., 2014. Speech biometric based attendance system. *2014 20th National Conference on Communications, NCC 2014*, [e-journal] pp.1-6. https://doi.org/10.1109/NCC.2014.6811345.

Du, G., Su, F. and Cai, A., 2009. Face recognition using SURF features and SVM Classifier. *MIPPR 2009: Pattern Recognition and Computer Vision*, [e-journal] 7496(1), p.749628. https://doi.org/10.1117/12.832636.

Francesco and Solawetz, J., 2023. *What is YOLOv8? The Ultimate Guide.* [online] Available at: <https://blog.roboflow.com/whats-new-in-yolov8/> [Accessed 7 April 2023].

Gopalakrishan, V., Arun, R. and Sasikumar, L., 2021. Handwritten Digit Recognition for Banking System. *International Journal of Scientific & Technology Research,* 9(5), pp.313–314.

Goyal, K., Agarwal, K. and Kumar, R., 2017. Face detection and tracking: Using OpenCV. *Proceedings of the International Conference on Electronics, Communication and Aerospace Technology, ICECA 2017*, [e-journal] pp.474–478. https://doi.org/10.1109/ICECA.2017.8203730.

Greeshma, K. V, College, C. and Gripsy, J.V., 2020. Image Classification using HOG and LBP Feature Descriptors with SVM and CNN. *International Journal of Engineering Research & Technology*, 8(04), pp.4–7.

Gross, R., 2005. Face Databases. *Handbook of Face Recognition*, pp.301–327.

Gwyn, T., Roy, K. and Atay, M., 2021. Face recognition using popular deep net architectures: A brief comparative study. *MDPI*, [e-journal] 13(7), pp.1–15. https://doi.org/10.3390/fi13070164.

Harakannanavar, S.S., R, P.C., Kanabur, V., Puranikmath, V.I. and Raja, K.B., 2019. Technical Challenges, Performance Metrics and Advancements in Face Recognition System. *International Journal of Computer Sciences and Engineering*, [e-journal] 7(3), pp.836–847. https://doi.org/10.26438/ijcse/v7i3.836847.

Hazim, N., Sameer, S., Esam, W. and Abdul, M., 2016. Face Detection and Recognition Using Viola-Jones with PCA-LDA and Square Euclidean Distance. *International Journal of Advanced Computer Science and Applications*, [e-journal] 7(5), pp.371-376. https://doi.org/10.14569/ijacsa.2016.070550.

Hese, S.K. and Banwaskar, M.R., 2013. Performance Evaluation of PCA and LDA for Face Recognition. *International Journal of Engineering Research & Technology*, 2(2), pp.149–154.

Higuera, V., 2011. How Are Diseases Transmitted? *Delaware Health and Social Services*, [online] Available at: <https://www.healthline.com/health/disease-transmission> [Accessed 3 October 2022].

Huang, K., 2012. Principal Component Analysis in the Eigenface Technique for Facial Recognition. [online] Available at: <https://digitalrepository.trincoll.edu/cgi/viewcontent.cgi?article=1221&context=theses> [Accessed 23 October 2022].

Jang, Y., Gunes, H. and Patras, I., 2019. Registration-free Face-SSD: Single shot analysis of smiles, facial attributes, and affect in the wild. *Computer Vision and Image Understanding*, [e-journal] 182(0), pp.17–29. https://doi.org/10.1016/j.cviu.2019.01.006.

Jayaswal, R. and Dixit, M., 2020. Comparative analysis of human face recognition by traditional methods and deep learning in real-time environment. *Proceedings - 2020 IEEE 9th International Conference on Communication Systems and Network Technologies, CSNT 2020*, [e-journal] pp.66–71. https://doi.org/10.1109/CSNT48778.2020.9115779.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S. and Darrell, T., 2014. Caffe : Convolutional Architecture for Fast Feature Embedding Categories and Subject Descriptors∗. *22nd ACM International Conference on Multimedia*, [e-journal] pp.675–678. https://doi.org/10.48550/arXiv.1408.5093.

Karamizadeh, S. and Abdullah, S.M., 2013. An Overview of Holistic Face Recognition. *International Journal of Research in Computer and Communication Technology*, 2(9), pp.738–741.

Karamizadeh, S., Abdullah, S.M., Manaf, A.A., Zamani, M. and Hooman, A., 2013. An Overview of Principal Component Analysis. *Journal of Signal and Information Processing*, [e-journal] 04(03), pp.173–175. https://doi.org/10.4236/jsip.2013.43b031.

Kim, C., Lee, J., Han, T. and Kim, Y.M., 2018. A hybrid framework combining background subtraction and deep neural networks for rapid person detection. *Journal of Big Data*, [e-journal] 5(1), pp.1–24. https://doi.org/10.1186/S40537-018-0131-X/TABLES/6.

Kriegman, D., n.d. Instructions : The Yale Face Database Recognition Using Eigenfaces. pp.2–4.

Krizhevsky, A., Sutskever, I. and Hinton, G.E., n.d. *ImageNet Classification with Deep Convolutional Neural Networks*. [online] Available at: <http://code.google.com/p/cuda-convnet/> [Accessed 4 July 2022].

Kukenys, I. and McCane, B., 2008. Support vector machines for human face detection. *New Zealand Computer Science Research Student Conference, NZCSRSC 2008 - Proceedings*, pp.226–229.

Lai, Y., 2019. A Comparison of Traditional Machine Learning and Deep Learning in Image Recognition. *Journal of Physics: Conference Series*, 1314(1). https://doi.org/10.1088/1742-6596/1314/1/012148.

Lakshmi, D. and Ponnusamy, R., 2021. Facial emotion recognition using modified HOG and LBP features with deep stacked autoencoders. *Association for Computing Machinery*, [e-journal] 82, p.103834. https://doi.org/10.1016/j.micpro.2021.103834.

Lal, M., Kumar, K., Arain, R.H., Maitlo, A., Ruk, S.A. and Shaikh, H., 2018. Study of face recognition techniques: A survey. *International Journal of Advanced Computer Science and Applications*, [e-journal] 9(6), pp.42–49. https://doi.org/10.14569/IJACSA.2018.090606.

Lecun, Y., Bottou, L., Bengio, Y. and Ha, P., 1998. LeNet. *Proceedings of the IEEE*, pp.1–46.

Lei, S., Oo, M. and Oo, A.N., 2019. Child Face Recognition System with FaceNet. *MURC*, (1), pp.2–5.

Li, L., Mu, X., Li, S. and Peng, H., 2020. A Review of Face Recognition Technology. *IEEE Access*, [e-journal] pp.139110–139120. https://doi.org/10.1109/ACCESS.2020.3011028.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y. and Berg, A.C., 2016. SSD: Single shot multibox detector. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, [e-journal] pp.21–37. https://doi.org/10.1007/978-3-319-46448-0_2.

Lu, J., Plataniotis, K.N. and Venetsanopoulos, A.N., 2003. Face recognition using LDA-based algorithms. *IEEE Transactions on Neural Networks*, [e-journal] 14(1), pp.195–200. https://doi.org/10.1109/TNN.2002.806647.

Masi, I., Wu, Y., Hassner, T. and Natarajan, P., 2019. Deep Face Recognition: A Survey. *Proceedings - 31st Conference on Graphics, Patterns and Images, SIBGRAPI 2018*, [e-journal] pp.471–478. https://doi.org/10.1109/SIBGRAPI.2018.00067.

Meynet, J., 2003. Fast face detection using adaboost. *University of Trier*, [online] Available at: <https://dtpapers.googlecode.com/files/Meynet2003_923.pdf> [Accessed 3 April 2023].

Madzou, L., 2020. *Facial recognition technology and travel after COVID-19 | World Economic Forum*. [online] Available at: <https://www.weforum.org/agenda/2020/12/facial-recognition-technology-and-travel-after-covid-19-freedom-versus-privacy/> [Accessed 5 September 2022].

Meena Prakash, R., Thenmoezhi, N. and Gayathri, M., 2019. Face Recognition with Convolutional Neural Network and Transfer Learning. *Proceedings of the 2nd International Conference on Smart Systems and Inventive Technology, ICSSIT 2019*, [e-journal] pp.861–864. https://doi.org/10.1109/ICSSIT46314.2019.8987899.

Mohamed, B.K.P. and Raghu, C. V., 2012. Fingerprint attendance system for classroom needs. *2012 Annual IEEE India Conference, INDICON 2012*, [e-journal] pp.433–438. https://doi.org/10.1109/INDCON.2012.6420657.

Solomon, M.M., Meena, M.S. and Kaur, J., 2021. Challenges in Face Recognition Technique. *Journal of University of Shanghai for Science and Technology*, [e-journal] 23(07), pp.1201–1204. https://doi.org/10.51201/jusst/21/07253.

Nelson, J., 2021. *Your Comprehensive Guide to the YOLO Family of Models*, [online] Available at: <https://blog.roboflow.com/guide-to-yolo-models/> [Accessed 6 July 2022].

Nguyen-Meidine, L.T., Granger, E., Kiran, M. and Blais-Morin, L.A., 2018. A comparison of CNN-based face and head detectors for real-time video surveillance applications. *Proceedings of the 7th International Conference on Image Processing Theory, Tools and Applications, IPTA 2017*, [e-journal], pp.1–7. https://doi.org/10.1109/IPTA.2017.8310113.

Niyi, T., Alagbe, K. and Wuraola, Y.A., 2019. Performance Assessment of Principal Component Analysis and Kernel Principal Component Analysis Using TOAM Database. *Asian Journal of Research in Computer Science*, 3(2), pp.1–10.

Nyein, T. and Oo, A.N., 2019. University Classroom Attendance System Using FaceNet and Support Vector Machine. *2019 International Conference on Advanced Information Technologies, ICAIT 2019*, [e-journal] pp.171–176. https://doi.org/10.1109/AITC.2019.8921316.

Othman Hammadi, Abdulkarim Dawah Abas, K.H.A., 2018. Face recognition using deep learning methods a review. *International Journal of Engineering &Technology*, [e-journal] 7(4), pp.6223–6228. https://doi.org/10.14419/ijet.v7i4.23127.

Patel, S. and Tere, G., 2014. Analysis and Comparison of Different Approaches. *Manipal Institute of Technology*, 5(10), pp.779–783.

Patil, S.A., 2014. Principle Component Analysis (PCA) and Linear Discriminant Analysis (LDA) based Face Recognition. *International Journal of Computer Applications*, pp.1–5.

Peter, M., Minoi, J.L. and Hipiny, I.H.M., 2019. 3D face recognition using kernel-based PCA approach. *Lecture Notes in Electrical Engineering*, [e-journal] 481, pp.77–86. https://doi.org/10.1007/978-981-13-2622-6_8.

Qi, D., Tan, W., Yao, Q. and Liu, J., 2021. YOLO5Face: Why Reinventing a Face Detector. [online] Available at: <http://arxiv.org/abs/2105.12931> [Accessed 3 April 2023].

Raj, A., 2021. *Increasing demand for facial recognition system technology in Malaysia*, [online] Available at: <https://techwireasia.com/2021/10/increasing-demand-for-facial-recognition-system-technology-in-malaysia/> [Accessed 22 June 2022].

Rana, A., 2022. Face Detection Using YOLO. *International Journal of Engineering and Techniques*, 8(3), pp.36–39.

Rashid, B., 2022. *(4) A Survey of Comparing Different Cloud Database Performance: SQL and NoSQL | Bilal Rashid - Academia.edu*, [online] Available at: <https://www.academia.edu/74820433/A_Survey_of_Comparing_Different_Cloud_Database_Performance_SQL_and_NoSQL> [Accessed 14 April 2023].

Ren, S., He, K., Girshick, R. and Sun, J., 2016. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. [online] Available at: <http://image-net.org/challenges/LSVRC/2015/results> [Accessed 4 July 2022].

Roure, J. and Faundez-Zanuy, M., 2005. Face recognition with small and large size databases. *Proceedings - International Carnahan Conference on Security Technology*, [e-journal] pp.153–156. https://doi.org/10.1109/ccst.2005.1594843.

Sabeenian, R.S., Aravind, S., Arunkumar, P., Harrish Joshua, P. and Eswarraj, G., 2020. Smart attendance system using face recognition. *Journal of Advanced Research in Dynamical and Control Systems*, [e-journal] 12(5 Special Issue), pp.1079–1084. https://doi.org/10.5373/JARDCS/V12SP5/20201860.

Schroff, F., Kalenichenko, D. and Philbin, J., 2015. FaceNet: A unified embedding for face recognition and clustering. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, [e-journal] pp.815–823. https://doi.org/10.1109/CVPR.2015.7298682.

Sepas-Moghaddam, A., Pereira, F.M. and Correia, P.L., 2020. Face recognition: A novel multi-level taxonomy based survey. *IET Biometrics*, [e-journal] 9(2), pp.58–67. https://doi.org/10.1049/iet-bmt.2019.0001.

Setiowati, S., Zulfanahri, Franita, E.L. and Ardiyanto, I., 2017. A review of optimization method in face recognition: Comparison deep learning and non-deep learning methods. *2017 9th International Conference on Information Technology and Electrical Engineering, ICITEE 2017*, [e-journal] pp.1–6. https://doi.org/10.1109/ICITEED.2017.8250484.

Shi, W., Bao, S. and Tan, D., 2019. FFESSD: An accurate and efficient single-shot detector for target detection. *Applied Sciences (Switzerland)*, [e-journal] 9(20). https://doi.org/10.3390/app9204276.

Shu, C., Ding, X. and Fang, C., 2011. Histogram of the oriented gradient for face recognition. *Tsinghua Science and Technology*, [e-journal] 16(2), pp.216–224. https://doi.org/10.1016/S1007-0214(11)70032-3.

Singh, A., Kansari, J. and Sinha, V.K., 2022. Face Recognition Using Transfer Learning by deep VGG16 model. 9(4), pp.121–127.

Singhal, N., Ganganwar, V., Yadav, M., Chauhan, A., Jakhar, M. and Sharma, K., 2021. Comparative study of machine learning and deep learning algorithm for face recognition. *Jordanian Journal of Computers and Information Technology*, [e-journal] 7(3), pp.313–325. https://doi.org/10.5455/JJCIT.71-1624859356.

Srivastava, S., Divekar, A.V., Anilkumar, C., Naik, I., Kulkarni, V. and Pattabiraman, V., 2021. Comparative analysis of deep learning image detection algorithms. *Journal of Big Data*, [e-journal] 8(1), pp.1-25. https://doi.org/10.1186/s40537-021-00434-w.

Sufyanu, Z., Mohamad, F., Yusuf, A. and Nuhu, A., 2016. Feature Extraction Methods for Face Recognition. *International Journal of Applied Engineering Research*, [e-journal] 5(3), pp.5658–5668. Available at: <https://www.researchgate.net/publication/313360566_FEATURE_EXTRACTION_METHODS_FOR_FACE_RECOGNITION>.

Suganya, S. and Menaka, D., 2014. Performance Evaluation of Face Recognition Algorithms. *International Journal on Recent and Innovation Trends in Computing and Communication*, 2(1), pp.135–140.

Tai, D.N., Kim, S., Lee, G., Yang, H., Na, I. and Oh, A., 2018. Tracking by Detection of Multiple Faces using SSD and CNN Features. *Smart Media Journal*, [e-journal] 7(4). https://doi.org/10.30693/SMJ.2018.7.4.61.

Tan, H., Yang, B. and Ma, Z., 2014. Face recognition based on the fusion of global and local HOG features of face images. *IET Computer Vision*, [e-journal] 8(3), pp.224–234. https://doi.org/10.1049/iet-cvi.2012.0302.

Timotius, I.K., Setyawan, I. and Febrianto, A.A., 2010. Face recognition between two person using kernel principal component analysis and support vector machines. *International Journal on Electrical Engineering and Informatics*, [e-journal] 2(1), pp.53–61. https://doi.org/10.15676/ijeei.2010.2.1.5.

Toygar, Ö. and Acan, A., 2003. Face Recognition Using PCA , LDA and ICA Approaches on Colored Images. *Journal of Electrical & Electronic Engineering*, 3(1), pp.735–743.

Trigueros, D.S., Meng, L. and Hartnett, M., 2018. Face Recognition: From Traditional to Deep Learning Methods. CoRR [e-journal]. https://doi.org/10.48550/arXiv.1811.00116.

Wahid, T., 2013. Face Recognition using Local Binary Patterns (LBP) Face Recognition using Local Binary Patterns LBP. *Journal of Computer Science and Technology Graphics & Vision*, [online] Available at: <https://globaljournals.org/GJCST_Volume13/1-Face-Recognition-using- Local.pdf> [Accessed 3 April 2023].

Walker, M., Schönborn, S., Greifeneder, R. and Vetter, T., 2018. The basel face database: A validated set of photographs reflecting systematic differences in big two and big five personality dimensions. *PLoS ONE*, [e-journal] 13(3), pp.1–20. https://doi.org/10.1371/journal.pone.0193190.

Wang, M. and Deng, W., 2021. Deep face recognition: A survey. *Neurocomputing*, [e-journal] 429, pp.215–244. https://doi.org/10.1016/j.neucom.2020.10.081.

Wang, Q., 2012. Kernel Principal Component Analysis and its Applications in Face Recognition and Active Shape Models. [online] Available at: <http://arxiv.org/abs/1207.3538> [Accessed 3 April 2023].

Weng, L., 2018. *Object Detection Part 4: Fast Detection Models | Lil'Log*, [online] Available at: <https://lilianweng.github.io/posts/2018-12-27-object-recognition-part-4/> [Accessed 31 July 2022].

Xie, J., 2009. Face Recognition Based on Curvelet Transform and LS-SVM. *Proceedings of the International Symposium on Information Processing*, 2, pp.140–143.

Yi, D., Lei, Z. and Li, S.Z., 2015. Shared representation learning for heterogenous face recognition. *2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition, FG 2015*. [e-journal] https://doi.org/10.1109/FG.2015.7163093.

Yu, J. and Zhang, W., 2021. Face mask wearing detection algorithm based on improved yolo-v4. *MDPI*, [e-journal] 21(9). https://doi.org/10.3390/s21093263.

Zeng, D., Veldhuis, R. and Spreeuwers, L., 2021. A survey of face recognition techniques under occlusion. *IET Biometrics*, [e-journal] 10(6), pp.581–606. https://doi.org/10.1049/bme2.12029.

Zhang, N. and Deng, W., 2016. Fine-grained LFW database. *2016 International Conference on Biometrics, ICB 2016*, [e-journal] pp.1–11. https://doi.org/10.1109/ICB.2016.7550057.

Zhao, Z.Q., Zheng, P., Xu, S.T. and Wu, X., 2019. Object Detection with Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems*, [e-journal] 30(11), pp.3212–3232. https://doi.org/10.1109/TNNLS.2018.2876865.

APPENDICES

Appendix A: Face Detector

i. Haar Cascade Files
haar-cascade-files/haarcascade_frontalface_default.xml atmaster · an austinbeing/haar-cascade-files · GitHub

ii. OpenCV DNN Model Configuration File
https://github.com/aakashjhawar/face-detection/blob/master/res10_300x300_ssd_iter_140000.caffemodel

iii. OpenCV DNN Model's Weights
https://github.com/aakashjhawar/face-detection/blob/master/deploy.prototxt.txt

Appendix B: Feature Extractor

i. FaceNet Model File
https://drive.google.com/file/d/1PZ_6Zsy1Vb0s0JmjEmVd8FS99zoMCiN1/view?usp=share_link

ii. FaceNet Weights File
https://drive.google.com/file/d/1e6PHRlIeayAsvRGpYUwvstklvJy-3H5B/view?usp=share_link

Appendix C: Smart Attendance System

i. Main Code

```
"""
Libraries
"""
import cv2
import os
import time
import numpy as np

from os import listdir, path, mkdir
from sys import argv, exit
from datetime import datetime
from numpy import asarray, expand_dims, vstack, array
from pickle import dumps, load
from openpyxl import load_workbook
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder

from PyQt5 import uic
from PyQt5.QtGui import QPixmap, QImage, QFont
from PyQt5.QtCore import QTimer, pyqtSignal, QThread, Qt, QSize
from PyQt5.QtWidgets import (
    QApplication,
    QPushButton,
    QDialog,
    QTableWidget,
    QTableWidgetItem,
    QLabel,
    QStackedWidget,)
```

```
from sklearn import svm
from tensorflow.keras.models import load_model

from imutils import paths, resize
from imutils.video import VideoStream

from Attendance import Attendance



"""
Configurations
"""
CAMERA_INDEX = 1  # set 0 for laptop default webcam
NUM_FRAMES = 20
CONFIDENCE_LEVEL = 0.9



"""
File paths
"""

MODEL_PATH = ".\\models\\FaceNetModel.h5"
WEIGHTS_PATH = ".\\models\\facenet_keras_weights.h5"
ENCONDINGS_PATH = ".\\models\\face_encoding.pickle"
PROTOTXT_PATH = ".\\models\\deploy.prototxt.txt"
CAFFE_MODEL = ".\\models\\res10_300x300_ssd_iter_140000.caffemodel"
TRAINING_FOLDER = ".\\Detected images"
ATTENDANCE_SHEET = "Attendance_Records.xlsx"



"""-
UI paths
"""
```

```
HomeScreen = ".\\UI\\HomeScreen.ui"
FaceRecog_Screen = ".\\UI\\FaceRecog.ui"
Camera_Sreen = ".\\UI\\Camera.ui"
CreateUserID_Screen = ".\\UI\\CreateUserID.ui"
Attendance_Screen = ".\\UI\\Attendance.ui"

EMBED_MODEL = ""
DETECT_MODEL = ""



"""
Function to load all models
"""
def models():
   print("[INFORMATION] Loading models....")
   global EMBED_MODEL, DETECT_MODEL
   EMBED_MODEL = load_model(MODEL_PATH)
   EMBED_MODEL.load_weights(WEIGHTS_PATH)
   DETECT_MODEL    =    cv2.dnn.readNetFromCaffe(PROTOTXT_PATH,
CAFFE_MODEL)
   print("[INFORMATION] Models loaded successfully")
   print("[INFORMATION] Compiling models....\n")
   EMBED_MODEL.compile()
   end_time = time.time()
   # calculate elapsed time
   elapsed_time = end_time - start_time
   print('Time to load all models: {:.2f} seconds\n'.format(elapsed_time))
   print("[INFORMATION] Models compiled successfully")
   print("[INFORMATION] Starting application....")


"""
```

```
Main Screen
"""


class WelcomeScreen(QDialog):
    def __init__(self):
        super(WelcomeScreen, self).__init__()
        uic.loadUi(HomeScreen, self)

        self.setMinimumSize(QSize(400, 300))
        flags = Qt.WindowFlags(Qt.Window)
        self.setWindowFlags(flags)

        self.pushButton.clicked.connect(self.gotoadd)
        self.pushButton_2.clicked.connect(self.gotoadd2)
        self.pushButton_3.clicked.connect(self.gotoadd3)

    def gotoadd(self):
        pushButton = CreateIDScreen()
        widget.addWidget(pushButton)
        widget.setCurrentIndex(widget.currentIndex() + 1)

    def gotoadd2(self):
        pushButton_2 = FaceRecogScreen()
        widget.addWidget(pushButton_2)
        widget.setCurrentIndex(widget.currentIndex() + 1)

    def gotoadd3(self):
        pushButton_3 = AttendanceScreen()
        widget.addWidget(pushButton_3)
        widget.setCurrentIndex(widget.currentIndex() + 1)
```

```
"""
Create User ID Screen
Allow user to input name
"""

class CreateIDScreen(QDialog):
    def __init__(self):
        super(CreateIDScreen, self).__init__()
        uic.loadUi(CreateUserID_Screen, self)

        # Connect buttons to functions
        self.nextbutton.clicked.connect(self.check_name_input)
        self.pushButton_back.clicked.connect(self.back_to_main)

    # Check input name
    def check_name_input(self):
        i = 5
        j = 0

        while i > j:
            # Retrieve the input name from user
            self.get_name()

            if not self.name:
                # Display warning message if name is not being input
                self.label_warning.setVisible(True)
                self.label_warning.setStyleSheet("color: red")
                self.label_warning.setFont(QFont("Yu Gothic UI Semibold", 12))
                self.label_warning.setText("Please enter name to proceed")
                return  # Return to wait for valid input

            elif self.check_duplicate():
                # If name already exists, loop back to get valid input
                j += 1
                continue

            else:
                # If name is input and valid, proceed to next screen
                self.duplicate_warning.setVisible(False)
                self.label_warning.setVisible(False)
                self.nextfunction()
                # Exit the loop
                break

    # Retrieve name from user
    def get_name(self):
        self.name = self.name_input.text()

    # Check if the input name is duplicate
    def check_duplicate(self):
        # Display warning message if name already exists
        for file in listdir("Detected images"):
            if file == self.name:
                self.duplicate_warning.setVisible(True)
                self.duplicate_warning.setStyleSheet("color: blue")
                self.duplicate_warning.setFont(QFont("Yu Gothic UI Semibold",
12))
                self.duplicate_warning.setText("Name already exists")
                return True  # Return True to indicate duplicate name found
        return False  # Return False to indicate no duplicate name found

    # Proceed to next screen
    def nextfunction(self):
        nextbutton = CreateIDScreen2(self.name)
        widget.addWidget(nextbutton)
        widget.setCurrentIndex(widget.currentIndex() + 1)
    # Return to main screen
    def back_to_main(self):
        pushButton_back = WelcomeScreen()
        widget.addWidget(pushButton_back)
        widget.setCurrentIndex(widget.currentIndex() + 1)


"""
Screen for capturing user image
As Database for Training Purpose
"""

class CreateIDScreen2(QDialog):
    def __init__(self, name):
        super(CreateIDScreen2, self).__init__()
        uic.loadUi(Camera_Sreen, self)

        self.counter = 0  # number of face captured
        self.ctr = 0  # sample face counter

        self.name = name  # name of the user
        self.encoding_thread = EncodingThread(self)  # thread for encoding the
face
        self.encoding_thread.encodingFinished.connect(self.encoding_finished)

        # load imported webcam
        print("[INFORMATION] Start Video Stream....")
        self.video = VideoStream(src=CAMERA_INDEX).start()

        # Create a timer to trigger face detection function
        # start_detection function, which will be triggered every 10 milliseconds
when the timer is started
```

```
        self.timer = QTimer(self)
        self.timer.timeout.connect(self.start_detection)
        self.timer.start(10)

        # Initialize variables to store the start and end times for capturing 20
images
        self.start_time = None
        self.end_time = None
        self.end_time_encoding = None

    def start_detection(self):
        # read frame from video stream
        frame = self.video.read()

        # resize frame
        frame = resize(frame, width=800)

        # grab frame and convert it to bloob
        (h, w) = frame.shape[:2]

        # convert image to required format
        blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0, (300,
300), (104, 177, 123))

        # pass blob through the network and obtain detections and predictions
        DETECT_MODEL.setInput(blob)
        detections = DETECT_MODEL.forward()

        # look for face with confidence level bigger than 0.5
        for i in range(0, detections.shape[2]):
            confidence = detections[0, 0, i, 2]
            if confidence > CONFIDENCE_LEVEL:
                # compute the coordinates of the bounding box for the face
            box = detections[0, 0, i, 3:7] * array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")
            face = frame[startY:endY, startX:endX]

            # save extracted face
            self.save_detected_face(face)
            self.ctr += 1
            self.counter += 1

        if self.counter == 1:
            # Start the timer when the first face is detected
            self.start_time = time.time()

        if self.ctr >= NUM_FRAMES and self.counter == NUM_FRAMES:
            # Stop the timer when 20 faces are captured
            self.end_time = time.time()
            elapsed_time = self.end_time - self.start_time
            print(f"Time to capture 20 images: {elapsed_time:.2f} seconds\n")

            # break
            self.video.stop()
            self.timer.stop()

            # show message after face is captured
            self.msg.setVisible(True)
            self.msg.setText("Face Captured")
            self.msg.setStyleSheet("color: white")
            self.msg.setFont(QFont("Yu Gothic UI Semibold", 16))

            # switch to main screen after 1ms
            QTimer.singleShot(1, lambda: self.switch_to_main())

    def save_detected_face(self, face):
```

```
# create new folder by naming with input name
new_folder = path.join(TRAINING_FOLDER, self.name)

# avoid FileExistsError Error
if not path.exists(new_folder):
    mkdir(new_folder)
    print("Directory ", new_folder, " Created \n")
    # write detected image to created folder
    # set jpeg quality to highest and save
    cv2.imwrite(
        path.join(new_folder, f"{self.name}_{self.ctr}.jpg"),
        face,
        [int(cv2.IMWRITE_JPEG_QUALITY), 100],
    )
else:
    cv2.imwrite(
        path.join(new_folder, f"{self.name}_{self.ctr}.jpg"),
        face,
        [int(cv2.IMWRITE_JPEG_QUALITY), 100],
    )

# create QImage object to RGB format
rgb_image = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)

# obtain height, width, and number of channels in the image
h, w, ch = rgb_image.shape
bytesPerLine = ch * w
convertToQtFormat = QImage(rgb_image.data, w, h, bytesPerLine, QImage.Format_RGB888)

# store image data as variable p, resize to fit the size of label widget, and set as pixmap on label widget
```

```
        p = convertToQtFormat.scaled(self.label_cam.width(),
self.label_cam.height(), Qt.KeepAspectRatio)
        self.label_cam.setPixmap(QPixmap.fromImage(p))
        # keep the aspect ratio of image during scaling
        self.label_cam.setScaledContents(True)

    # switch to main screen and generate encodings from trained images
    def switch_to_main(self):
        self.video.stream.release()

        # show encoding in progress message
        self.msg_encode.setVisible(True)
        self.msg_encode.setText("Encoding in progress..")
        self.msg_encode.setStyleSheet("color: white")
        self.msg_encode.setFont(QFont("Yu Gothic UI Semibold", 12))

        self.encoding_thread = EncodingThread(parent=self, batch_size=32)
        self.encoding_thread.start()
        self.encoding_thread.encodingFinished.connect(self.encoding_finished)

    def encoding_finished(self):
        # switch to the welcome screen
        widget.addWidget(WelcomeScreen())
        widget.setCurrentIndex(widget.currentIndex() + 1)

        # print the time it took to capture 20 images with encodings
        self.end_time_encoding = time.time()
        elapsed_time_encoding = self.end_time_encoding - self.start_time
        print(f"Time to capture 20 images with encodings:
{elapsed_time_encoding:.2f} seconds\n")
        print("Encoding finished.")
```

```
"""
generate encodings from trained images in seperate thread
"""

class EncodingThread(QThread):
    encodingFinished = pyqtSignal()

    def __init__(self, parent=None, batch_size=32):
        super(EncodingThread, self).__init__(parent)
        self.batch_size = batch_size

    def run(self):
        # Load existing encodings
        if os.path.exists(ENCONDINGS_PATH):
            with open(ENCONDINGS_PATH, "rb") as f:
                data = load(f)
            knownEncodings = data["encodings"]
            knownNames = data["names"]
            existingName = set(knownNames)
        else:
            knownEncodings = []
            knownNames = []
            existingName = set()

        # Get list of images in training folder
        imagepaths = list(paths.list_images(TRAINING_FOLDER))

        # Loop through images in batches
        for i in range(0, len(imagepaths), self.batch_size):
            batch_paths = imagepaths[i: i+self.batch_size]
            batch_images = []
            batch_names = []
            # Loop through images in the batch
            for imagepath in batch_paths:
                name = imagepath.split(path.sep)[-2]
                if name in existingName:
                    continue  # Skip if person's images have been added
                image = cv2.imread(imagepath)
                # Convert to RGB format
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                # Normalize pixel values
                pixels = asarray(image)
                cv2.normalize(image, None, 0, 1.0,
                              cv2.NORM_MINMAX, dtype=cv2.CV_32F)
                # Resize image to 160x160
                image = cv2.resize(image, (160, 160))
                # Expand dimension to fit the model
                pixels = expand_dims(image, axis=0)
                # Add image and name to batch lists
                batch_images.append(pixels)
                batch_names.append(name)

            # Make prediction on batch of images
            if not batch_images:
                continue  # If no new images to encode
            batch_images = vstack(batch_images)
            batch_encodings = EMBED_MODEL.predict(batch_images)

            # Append encodings and names to lists
            for encoding, name in zip(batch_encodings, batch_names):
                knownEncodings.append(encoding)
                knownNames.append(name)
                existingName.add(name)

        # Write encoding data as pickle file
```

```
        data = {"encodings": knownEncodings, "names": knownNames}
        with open(ENCONDINGS_PATH, "wb") as f:
            f.write(dumps(data))
            print("Encoding complete.\n")

        self.encodingFinished.emit()


"""
Face Recognition Screen

"""

class FaceRecogScreen(QDialog):
    def __init__(self):
        super(FaceRecogScreen, self).__init__()
        uic.loadUi(FaceRecog_Screen, self)

        # initialize the clf variable
        self.clf = None

        # define fram that showing video stream as qlabel
        self.label_cam = QLabel()
        self.label_cam = self.findChild(QLabel, "label_cam")

        # stop the thread if 'cancel' button is pressed
        self.pushButton_cancel = self.findChild(
            QPushButton, "pushButton_cancel")
        self.pushButton_cancel.clicked.connect(self.stop_video)

        # define label for date and time
        self.label_datetime = self.findChild(QLabel, "label_datetime")
        # initialize the video stream with thread
        self.thread = StartThread(clf=self.clf)
        self.thread.start()
        self.thread.ImageUpdate.connect(self.ImageUpdateSlot)

        # define timer to update date and time
        self.update_date_time()

    def ImageUpdateSlot(self, Image):
        self.label_cam.setPixmap(QPixmap.fromImage(Image))

    def stop_video(self):
        self.thread.stop()
        self.back_to_main()

    def back_to_main(self):
        pushButton_cancel = WelcomeScreen()
        widget.addWidget(pushButton_cancel)
        widget.setCurrentIndex(widget.currentIndex() + 1)

    def update_date_time(self):
        date_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        self.label_datetime.setText(date_time)


class StartThread(QThread):
    ImageUpdate = pyqtSignal(QImage)

    def __init__(self, clf):
        super().__init__()
        self.clf = clf
        self.label_encoder = None
        self.video = None
```

```
 def run(self):
     self.ThreadActive = True
     # load the face embedding model
     with open(ENCONDINGS_PATH, "rb") as f:
         data = load(f)

     # load encodings and names
     encodings = data["encodings"]
     names = data["names"]

     # convert class labels to numeric labels using LabelEncoder
     self.label_encoder = LabelEncoder()
     self.label_encoder.fit(names)
     numeric_labels = self.label_encoder.transform(names)

     # convert encodings and names to numpy arrays
     knownEncodings = np.array(encodings)
     knownNames = np.array(numeric_labels)

     # replace NaN values with mean
     imputer = SimpleImputer(strategy='mean')
     knownEncodings = imputer.fit_transform(knownEncodings)

     # flatten encodings
     knownEncodings = knownEncodings.reshape(knownEncodings.shape[0],
-1)

     # pad the training encodings with zeros to ensure all training samples
have the same number of features
     max_features = knownEncodings.shape[1]
     knownEncodings = np.pad(knownEncodings, ((
         0, 0), (0, max_features - knownEncodings.shape[1])), mode='constant')
```

```
# train the classifier using known encodings and names
clf = svm.SVC(kernel='rbf', C=5, tol=0.001, probability=True)
clf.fit(knownEncodings, knownNames)
self.clf = clf

# initialize video stream
video = cv2.VideoCapture(CAMERA_INDEX)
video.set(cv2.CAP_PROP_FPS, 60)

while self.ThreadActive:
    ret, frame = video.read()
    if ret:
        (h, w) = frame.shape[:2]

        # Pass the input frame to the face detection model
        blob = cv2.dnn.blobFromImage(
            cv2.resize(frame, (300, 300)),
            1.0,
            (300, 300),
            (104.0, 177.0, 123.0),
        )
        DETECT_MODEL.setInput(blob)
        detections = DETECT_MODEL.forward()

        # initialize predicted_names outside the loop
        predicted_names = []

        # record start time
        self.start_time = time.time()

        # Loop over the detected faces
        for i in range(0, detections.shape[2]):
            confidence = detections[0, 0, i, 2]

            # Only consider detections above a certain confidence level
            if confidence > CONFIDENCE_LEVEL:
                # Get the coordinates of the detected face
                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")

                # Extract the face from the input frame
                face = frame[startY:endY, startX:endX]
                face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
                cv2.normalize(face, None, 0, 1.0, cv2.NORM_MINMAX,
dtype=cv2.CV_32F)

                # Resize the face to the input size of the embedding model
                face_resized = cv2.resize(face, (160, 160))
                pixels = np.expand_dims(face_resized, axis=0)

                # Use Facenet to predict
                face_encodings = EMBED_MODEL.predict(pixels)

                # Use the self.clf classifier to predict face names
                predicted_names = clf.predict(face_encodings)
                predicted_names                                        =
self.label_encoder.inverse_transform(predicted_names)

                # Draw bounding box and label around face
                y = startY - 10 if startY - 10 > 10 else startY + 10
                cv2.rectangle(frame, (startX, startY),(endX, endY), (0, 255, 0),
2)
                cv2.putText(
                    frame,          predicted_names[0],          (startX,          y),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 255, 0), 2)

                # mark attendance
                self.attendance          =          Attendance(interval=1,
attendance_file=ATTENDANCE_SHEET)
                self.attendance.mark_attendance(predicted_names[0])

            # record end time
            self.end_time = time.time()
            elapsed_time_attendance = self.end_time - self.start_time
            print("\nTime  taken  for  whole  recognition  process:  {:.3f}
seconds\n".format(
                elapsed_time_attendance))

            # convert frame to QImage to display video stream
            Image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            FlippedImage = Image.copy()
            ConvertToQtFormat = QImage(
                FlippedImage.data,
                FlippedImage.shape[1],
                FlippedImage.shape[0],
                QImage.Format_RGB888,
            )
            Pic = ConvertToQtFormat.scaled(
                800, 520, Qt.KeepAspectRatio)
            self.ImageUpdate.emit(Pic)

    def stop(self):
        self.ThreadActive = False
        self.quit()
```

```
"""
Attendance Screen
"""

class AttendanceScreen(QDialog):
    def __init__(self):
        super(AttendanceScreen, self).__init__()
        uic.loadUi(Attendance_Screen, self)

        # Initialize instance variables
        self.workbook = None
        self.sheet = None

        # Find the table widget in the UI and store a reference to it
        self.tableWidget = self.findChild(QTableWidget, "tableWidget")
        # Load the attendance workbook and populate the table
        self.load_workbook()
        self.label_attendance_3 = QLabel()

        # Connect the back button to the back_to_main function
        self.pushButton_back.clicked.connect(self.back_to_main)

    def load_workbook(self):
        # Load the attendance workbook and get the sheet
        self.workbook = load_workbook(ATTENDANCE_SHEET)
        self.sheet = self.workbook["Sheet1"]

        # Get the number of rows and columns
        rows = self.sheet.max_row
        columns = self.sheet.max_column

        # Set the number of rows and columns in the table
        self.tableWidget.setRowCount(rows)
        self.tableWidget.setColumnCount(columns)

        # Iterate over each cell in the sheet and add the value to the table
        for row in range(1, rows + 1):
            for col in range(1, columns + 1):
                value = self.sheet.cell(row=row, column=col).value
                item = QTableWidgetItem(str(value))
                self.tableWidget.setItem(row - 1, col - 1, item)

        # Update the table in the workbook
        self.workbook.save(ATTENDANCE_SHEET)
        # Update date and time
        self.update_date_time()

    def update_date_time(self):
        # Get the current date and time
        date_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        self.label_attendance_3.setText(date_time)

        # Update the date and time in the table
        self.timer = QTimer()
        self.timer.timeout.connect(self.update_date_time)
        self.timer.start(100)

    def back_to_main(self):
        # Return to the main welcome screen
        pushButton_back = WelcomeScreen()
        widget.addWidget(pushButton_back)
        widget.setCurrentIndex(widget.currentIndex() + 1)


# Main
if __name__ == "__main__":
    start_time = time.time()
    models()
    app = QApplication(argv)
    window = WelcomeScreen()
    widget = QStackedWidget()
    widget.addWidget(window)
    widget.setFixedWidth(1143)
    widget.setFixedHeight(798)
    widget.setWindowTitle("Smart Attendance System")
    widget.show()
    try:
        exit(app.exec_())
    except:
        print("Exiting")
```

The entire code for this main program is also accessible at the link below:

https://drive.google.com/file/d/1jfwyXvYptoHwIhiXP0x2G2WHDbtEyhhZ/view?usp=share_link

ii. Attendance Marking Code

```
import datetime
import openpyxl
import xlsxwriter

class Attendance:
   def __init__(self, interval=1, attendance_file=""):
      # Create a workbook and add a worksheet.
      self.workbook = xlsxwriter.Workbook(attendance_file)
      self.worksheet = self.workbook.add_worksheet()
      self.date_time = datetime.datetime.strptime("2023-01-10", "%Y-%m-%d")
      self.interval = interval
      self.attendance_file = attendance_file

      # Start from the first cell below the headers.
      self.row = 0
      self.col = 0

   def checkPreviousAttendance(self, face_name_str):
      # Check if the face_names_str is already in the excel file within the last 1
minute
      self.workbook = openpyxl.load_workbook(self.attendance_file)
      self.worksheet = self.workbook.active

      for i in range(1, self.worksheet.max_row + 1):
         if self.worksheet.cell(i, 3).value == face_name_str:
            # if exists, get the date and time of the last attendance
            self.date_time = datetime.datetime.strptime(
               self.worksheet.cell(i, 1).value, "%d/%m/%Y"
            )
            self.time = datetime.datetime.strptime(
               self.worksheet.cell(i, 2).value, "%H:%M:%S"
```

```
        )
        # check if the last attendance is within the last 1 minute
        if datetime.datetime.now() - datetime.timedelta(
            minutes=self.interval
        ) < datetime.datetime.combine(self.date_time, self.time.time()):
            return True
    return False

def mark_attendance(self, face_names_str):
    if self.checkPreviousAttendance(face_names_str):
        return
    # mark attendance on excel
    self.workbook = openpyxl.load_workbook(self.attendance_file)
    self.worksheet = self.workbook.active
    # Write some data headers.
    self.worksheet.cell(1, 1).value = "DATE"
    self.worksheet.cell(1, 2).value = "TIME"
    self.worksheet.cell(1, 3).value = "NAME"
    # get current date
    date = datetime.datetime.now().strftime("%d/%m/%Y")
    # get current time
    time = datetime.datetime.now().strftime("%H:%M:%S")
    # get the row number
    self.row = self.worksheet.max_row
    # get the column number
    self.col = self.worksheet.max_column
    # write date, time and name in excel
    self.worksheet.cell(self.row + 1, 1).value = date
    self.worksheet.cell(self.row + 1, 2).value = time
    self.worksheet.cell(self.row + 1, 3).value = face_names_str
    # save the excel file
    self.workbook.save(self.attendance_file)
```

The entire code for this attendance marking script is accessible at the link below:

https://drive.google.com/file/d/17Fx5uyyO0bSzDDJV9ge7p7_RNkbVgMJL/view?usp=share_link

The up-to-date attendance records excel sheet is accessible at the link below:

https://docs.google.com/spreadsheets/d/1Wj7PYMPT9-MvynNJp6i_C8r0Ni4ovdzP/edit?usp=share_link&ouid=109643314267715793484&rtpof=true&sd=true

Appendix D: User Interface Configuration Files

i. Homescreen
https://drive.google.com/file/d/1TG8rJSJiHm9OS6SE3J1qY914H7D8nmsV/view?usp=share_link

ii. Create User ID First Screen
https://drive.google.com/file/d/1TwrmVmXPGYWcShQyhfrhqDWAb9IupcVe/view?usp=share_link

iii. Create User ID Second Screen
https://drive.google.com/file/d/1oDbv4Um40PNesogAs313Guzi5XqFxovc/view?usp=share_link

iv. Face Recognition Screen
https://drive.google.com/file/d/116ExDYvz4fVEx3nyQ92MG0fYaYOwJmok/view?usp=share_link

v. Attedance Monitoring Screen
https://drive.google.com/file/d/1KreD3JiSRqFeaAWSTWdYFsxBSiNbPuAT/view?usp=share_link

Appendix E: Evaluation for Two Different Models

i. Collected Images Used for Models Evaluation
https://drive.google.com/drive/folders/13JxuxAe3FBQl19nxHQsgF9gj
eriZpeR9?usp=share_link

ii. Code to Split Datasets into Train, Validation and Test Datasets
https://drive.google.com/file/d/1PGx3_ul8nfhqVoFXaftP27inkVjTIAJ
W/view?usp=share_link

iii. Feature Extraction using FaceNet Code for Training Datasets
https://drive.google.com/file/d/14cxTW5CpV2IWt0h_9rLAosrx3HY3
9xdW/view?usp=share_link

iv. Feature Extraction using FaceNet Code for Validation and Testing Datasets
https://drive.google.com/file/d/1yVBq0JJeIoTaJ2fHS9pF76755sJdOcF
K/view?usp=share_link

v. VGG 16 Model File
https://drive.google.com/file/d/1ZEEPRYzoOyOYORuzYnQa-
TW4aqPl0BZQ/view?usp=share_link

vi. Evaluation Code for DNN, FaceNet and SVM
https://drive.google.com/file/d/16U_-lrdwYcwlwUXq50SJQqU-9vsno-
n5/view?usp=share_link

vii. Evaluation Code for VGG16
https://drive.google.com/file/d/1MA7srMjtwSxaIgMFZVL6qv9zaglcs
FIz/view?usp=share_link