

FACE DETECTION AND RECOGNITION IN LOW RESOURCE ANDROID MOBILE DEVICES

A MAJOR PROJECT-II REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE
OF

MASTERS OF TECHNOLOGY
IN
INFORMATION TECHNOLOGY

Submitted by

NITIN (2K23/ITY/20)

Under the supervision of

PROF. (DR.) DINESH KUMAR VISHWAKARMA



**DEPARTMENT OF INFORMATION TECHNOLOGY
DELHI TECHNOLOGICAL UNIVERSITY**

(Formerly Delhi College of Engineering)
Bawana Road, Delhi 110042

MAY, 2025

DEPARTMENT OF INFORMATION TECHNOLOGY
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, NITIN, Roll No – 2K23/ITY/20 students of M.Tech (Information Technology), hereby declare that the project Dissertation titled “FACE DETECTION AND RECOGNITION IN LOW RESOURCE ANDROID MOBILE DEVICES” which is submitted by us to the Information Technology, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of degree of Masters of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi

NITIN

Date:

2K23/ITY/20

DEPARTMENT OF INFORMATION TECHNOLOGY
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled “FACE DETECTION AND RECOGNITION IN LOW RESOURCE ANDROID MOBILE DEVICES” which is submitted by NITIN, Roll No’s – 2K23/ITY/20, Department Name ,Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Masters of Technology, is a record of the project work carried out by the students under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi

Signature

Date:

Prof. (Dr.) Dinesh Kumar Vishwakarma

SUPERVISOR

Head of Department

Department Of Information Technology

Delhi Technological University

DEPARTMENT OF INFORMATION TECHNOLOGY
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

ACKNOWLEDGEMENT

I wish to express our sincerest gratitude to Dr Prof. (Dr.) Dinesh Kumar Vishwakarma for his continuous guidance and mentorship that he provided me during the project. He showed me the path to achieve our targets by explaining all the tasks to be done and explained to me the importance of this project as well as its industrial relevance. He was always ready to help me and clear our doubts regarding any hurdles in this project. Without his constant support and motivation, this project would not have been successful.

Place: Delhi

NITIN

Date:

2K23/ITY/20

Abstract

Nowadays, face detection and recognition are used in many mobile apps like attendance systems, access control, and login systems. But when we try to use these things in low-resource Android mobile phones, it becomes a bit difficult because mobile phones have less RAM, CPU, and battery, so we have to manage all these things properly. So, in my MTech thesis, I tried to make face detection and recognition work in such low-resource Android phones. In this project, for detecting faces, I used the Google ML Kit library. It is a ready-made Android library, which works fast and is already optimized for mobile phones. It can find faces in real time, draw boundaries on them, and crop the faces also. For recognizing the faces, I used two models — FaceNet and Mobile FaceNet. I converted both of these into TFLite format so that they can run easily on Android devices. I created a full Android app from scratch. In the app itself when a person registers their face by using face detection and face recognition model, his face embeddings are saved and when he recognizes it, his embeddings are compared with the registered faces and whose embeddings are less is considered as the detected face.

Contents

Candidate's Declaration	i
Certificate	ii
Acknowledgement	iii
Abstract	iv
Content	vi
List of Tables	vii
List of Figures	ix
List of Symbols, Abbreviations	1
1 INTRODUCTION	2
1.1 Background	2
1.2 Importance of Face Detection and Recognition	2
1.3 Real-World Use Cases	3
1.4 Challenges in Low-Resource Mobile Environments	3
1.5 Why We Need Optimized Face Recognition for Low-End Android Phones	4
1.6 What This Thesis Tries to Do	4
2 LITERATURE REVIEW	6
3 METHODOLOGY	13
3.1 Proposed Approach	13
3.2 Face Detection	13
3.3 Face Recognition	13
3.4 Registration Activity	14
3.5 Recognition Activity	15

3.6	TensorFlow Lite	17
3.7	Quantization	19
3.8	Face Detection using ML Kit	20
3.9	Detect faces with ML Kit on Android	21
3.9.1	Configure the face detector	21
3.9.2	Get an instance of FaceDetector	22
3.9.3	Prepare the input image	22
3.9.4	Process the image	23
3.9.5	Get information about detected faces	23
3.10	Dataset Used(LFW Dataset)	24
3.11	Face Recognition	24
3.11.1	The FaceNet approach	25
3.11.2	MobileFaceNet approach	27
3.11.3	How face recognition system works in my app	29
3.11.4	Adding the face recognition step	30
4	RESULTS and DISCUSSION	31
5	CONCLUSION AND FUTURE SCOPE	44

List of Tables

2.1	Comparison of Face Detection Models	11
2.2	Comparison of Face Recognition Models	12
4.1	Comparison of Face Recognition Models: FaceNet vs Mobile-FaceNet on LFW Dataset	38
4.2	Comparison of various Face Recognition Models	38

List of Figures

1.1	Face Registration and Recognition Model	4
3.1	Face Registration	14
3.2	Saving face and embedding in to database	15
3.3	Face Recognition	16
3.4	Comparing Embeddings value	16
3.5	Face Registration and Recognition	17
3.6	TensorFlow Lite Model	18
3.7	TensorFlow Lite Architecture	18
3.8	Quantization	19
3.9	Face Detection	22
3.10	Face Recognition	25
3.11	CNN Architecture	27
3.12	Face Recognition Flow	28
3.13	Face Recognition on Image	29
4.1	Main Activity User Interface	31
4.2	Teacher Login Activity	32
4.3	Registered Student Details Activity	32
4.4	Student Login Activity	33
4.5	Student Dashboard Activity	33
4.6	Registration and Recognition Activity User Interface	34
4.7	Face Detection of app	34
4.8	Dialouge box for face Registration	35
4.9	Face Recognition with name and Distance	36
4.10	Present Student Details Activity	37
4.11	Accuracy FaceNet(TFlite) vs MobileFaceNet(TFlite)	39
4.12	Precision FaceNet(TFlite) vs MobileFaceNet(TFlite)	40
4.13	Recall FaceNet(TFlite) vs MobileFaceNet(TFlite)	40
4.14	F1 Score FaceNet(TFlite) vs MobileFaceNet(TFlite)	41
4.15	Inference Time FaceNet(TFlite) vs MobileFaceNet(TFlite)	41
4.16	Model Size FaceNet(TFlite) vs MobileFaceNet(TFlite)	42
4.17	Accuracy of various face recognition models	42

4.18 Inference time of face recognition model	43
---	----

List of Abbreviations

AI	Artificial Intelligence
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DTU	Delhi Technological University
FPS	Frames Per Second
GPU	Graphics Processing Unit
LBP	Local Binary Pattern
ML	Machine Learning
MTCNN	Multi-task Cascaded Convolutional Networks
PCA	Principal Component Analysis
RAM	Random Access Memory
SDK	Software Development Kit
TFLite	TensorFlow Lite
UI	User Interface
YOLO	You Only Look Once

Chapter 1

INTRODUCTION

1.1 Background

Face detection and recognition are now very common parts of computer vision systems. They help the computer to find and recognize human faces from photos or videos. These are used in many areas like security, login systems, and even when humans interact with machines. Earlier, these systems were using old methods like Viola–Jones, which worked with Haar features and cascade classifiers. But these had problems like they didn’t work well in different lighting conditions, or if the face is turned or covered. Later, deep learning and CNNs came, which changed everything. Now the models can learn from data automatically and work better, giving more accurate and strong results in face recognition.

1.2 Importance of Face Detection and Recognition

Face detection and recognition are very useful in many real-world areas:

- **Security and Surveillance:** Police and security teams use it to find suspects and keep an eye on public areas[1].
- **Mobile Authentication:** In smartphones, face recognition is used for unlocking the phone instead of using passwords or patterns[2].
- **Access Control:** Many offices use face recognition to allow only authorized people to enter certain rooms or areas[3].
- **Social Media:** Apps like Facebook and Instagram use face detection to auto-tag people in photos[4].

- **Healthcare:** In hospitals, face recognition is tested to identify patients and reduce mistakes in admin work[5].

1.3 Real-World Use Cases

In recent years, many real-life places started using face recognition:

- **Law Enforcement:** Police in London are using live face recognition cameras to catch people who are on their watchlist[6].
- **Airports:** Many airports use it to check passengers quickly and speed up boarding[7].
- **Retail:** Shops use it to see customer age group, gender, and behavior so they can improve their marketing and customer experience.

1.4 Challenges in Low-Resource Mobile Environments

When we try to use face detection and recognition on low-end Android phones, there are some big problems:

- **Low Processing Power:** These phones don't have fast CPUs like laptops or desktops, so running deep learning models becomes slow[8].
- **Low RAM and Storage:** Mobile phones have limited memory, so we can't use big models[9].
- **Battery Usage:** If face recognition keeps running, it will drain the battery very fast[10].
- **Heating Issues:** If the phone works hard for long time, it gets hot and affects user experience[11].

So, because of these problems, we have to use small and optimized models that can work well on mobile devices.

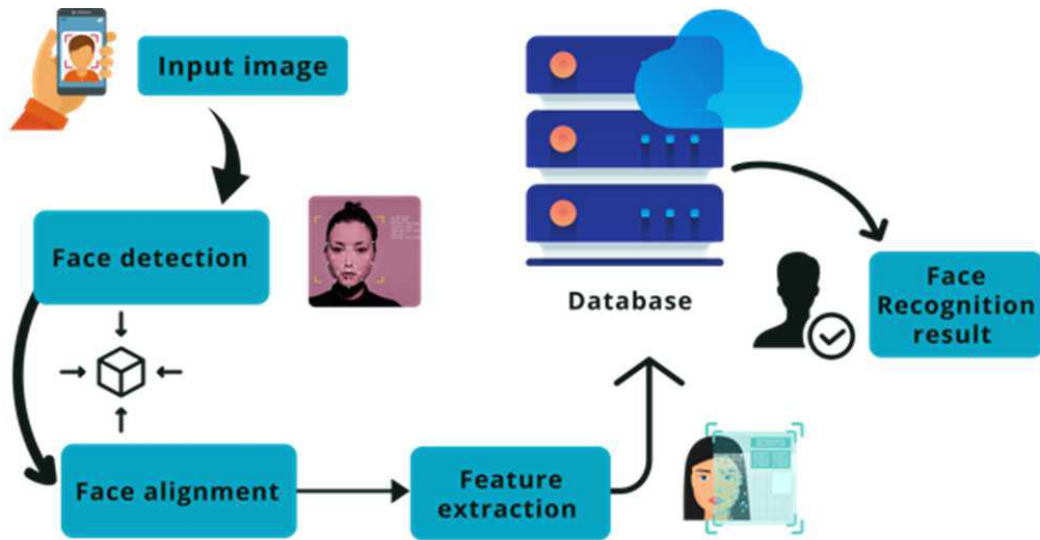


Figure 1.1: Face Registration and Recognition Model

1.5 Why We Need Optimized Face Recognition for Low-End Android Phones

Now many people use smartphones, especially in countries where people mostly use budget phones. So we need face recognition systems that can run even on these cheap or low-end devices. If we make our models small and optimized, more people can use them for taking attendance, secure logins, or personal apps — without needing costly phones[12].

Also, when the face recognition happens inside the phone (on-device), then we don't need to send face data over the internet. This makes it more private and secure and helps in following privacy rules.[13]

1.6 What This Thesis Tries to Do

In this thesis, my aim is to:

- Make a good face detection and recognition system that can run on low-end Android phones.

- Use small models like Mobile FaceNet (in TFLite format), which are fast and lightweight.
- Test the system and check how accurate and fast it is, and how much resources it uses on different low-end phones.
- Show how this system can be used in real apps like attendance systems.

Chapter 2

LITERATURE REVIEW

After reviewing different research papers, i observes that there are various approaches by which we can do the face detection and recognition also previously various people did research on low resources by using the various way and find out different way of doing this work.

In the research paper by Mubarak Alburaiki and his team [14], they made a mobile-based attendance app that uses face recognition and also checks the location of the user. They wanted to fix problems in old methods like QR code and fingerprint-based systems, which can be misused or don't work well in some conditions. For detecting and recognizing the face, they used Azure Face API, and for checking the location, they used Mapbox API. In their testing, 85% of users were happy with the app. They also did beta testing with 30 people and the system worked well in good lighting and stable internet. But there were some issues — like in low-light conditions, the face recognition didn't work properly, and it needed a good internet connection all the time, which is a problem for low-resource areas. Also, they didn't talk much about how much battery or CPU the app uses, which is important if we want to use it on low-end Android phones. That's why in my thesis, I focused on using lightweight and optimized models that can work without needing heavy processing or fast internet.

In the research by Amirul Mukhlis and his team [15], they made a web-based attendance system that uses face recognition along with blockchain. Their aim was to fix the problems in traditional attendance methods like using paper or QR codes, which can be misused or changed easily. They used OpenCV with Python and got 98% accuracy in detecting faces, and it took around 2 seconds to recognize one face. To keep the attendance data safe, they used smart contracts on the Ethereum blockchain, which makes the records unchangeable. They tested the system with 13 users and it worked well in

a controlled environment. But the system depends on webcams and central servers, which is not suitable for low-end Android phones. Also, they didn't discuss how much processing power or battery the system needs, which is important when we want to run such systems on mobile devices. This is why my thesis focuses on using small, optimized models that can run directly on Android phones without needing too many resources.

In another research by Arjun Raj and team [16], they developed a smart attendance system using face recognition to solve problems like fake attendance and slow manual methods. They used the LBPH algorithm, which works better than Eigenfaces and Fisherfaces, especially in different lighting. Their system gave 98.5% accuracy. They built it using Raspberry Pi with OpenCV and Dlib libraries. It also included steps like converting images to grayscale and improving contrast to make face detection better. They even added a GSM module to send messages when a student is absent. The system worked well inside classrooms, but it needed fixed camera positions and didn't perform well when faces were covered or turned too much. Because of this, it's hard to use it in real-time on mobile phones, especially in low-resource devices. Even though LBPH is fast and works on edge devices, they didn't try to make it suitable for Android phones with less power. That's why my thesis focuses on models that are optimized for mobile use, with low battery and CPU usage.

In the paper by Paul Viola and Michael Jones[17], they made a very fast and accurate face detection method. They used something called a boosted cascade of simple features, and their model could detect faces in real time, even on an old computer with only 700MHz processor, with 15 FPS speed. Their accuracy was also good, like 93.9% on the MIT+CMU dataset. They introduced three main things: one is integral image which helps to calculate features very fast, second is AdaBoost which selects the important features like eyes and nose contrast, and third is cascade classifier which helps to skip the areas where there is no face early, so the processing becomes faster. But still, this model was not working well if the face is turned or the light is too much or too low. Also, it only works on fixed size windows, like 24x24, and uses handcrafted features, so it cannot handle all types of faces. This model mainly works well for straight faces and in good lighting. But because it is very fast and doesn't need much resources, it was good for old devices like handheld systems.

In paper by Kaipeng Zhang [18], in which they introduced MTCNN, which is a deep learning based face detector and also does alignment. It uses three neural networks: P-Net, R-Net and O-Net. This system works in three steps,

like first detecting face, then rejecting wrong ones, and then improving the box. It gives very good accuracy like 95.1% on FDDB dataset and works in real-time like 16 FPS on GPU. They also used some smart techniques like training with multiple tasks together like detection and alignment, and they also used only hard samples during training to improve the learning. But the problem is, if the face is covered with sunglasses or very small, then it doesn't work well. And also it runs mainly on GPU, so it is not good for low-end Android mobiles. But still, it is better than older models because it combines both detection and alignment.

Moh. Edi Wibowo [19], used RetinaFace with some tracking to make the face detection better in CCTV-type systems, especially when light is low or image is low-quality. Their model got 91.4% precision on WIDER FACE dataset and also improved recall by 4.47%, and it runs at 25 FPS. They did this by combining many things like face detection, landmark detection, and pixel regression into one loss function. Also, they used a buffer to track faces across frames using IoU, so even if detection fails for a moment, tracking keeps it going. But still, their system doesn't work well in dark conditions or when the face is too small, like less than 20x20 pixels. Also, the tracking part needs more CPU, so it becomes heavy for real-time apps on Android. This shows that if we want high accuracy, we need more power, so for mobiles we need to find a balance.

Valentin Bazarevsky [20] introduced BlazeFace, which is made specially for mobile phones. It is super fast, like even 1000+ FPS on high-end mobiles, and they made it using GPU-based layers called BlazeBlocks. They also removed some extra feature maps to make it run faster, and they used blending to make the bounding boxes more stable. Their model is very small and fast, and gives 98.6% accuracy on frontal faces. It is even better than MobileNet SSD in speed. But still, it is mainly made for faces which are close to the camera, and doesn't work well for very small faces. Also, it needs GPU, so it may not run well on low-end Android phones that don't have GPU support. So there is a trade-off between speed and hardware compatibility.

Glenn Jocher [21] used YOLOv5 for face detection. They changed the normal YOLOv5 model and made it work well for faces. They used CSPDarknet for the backbone, and also used auto anchor boxes and other optimizations using PyTorch. On the WIDER FACE dataset, they got 96.8% mAP and the speed was very high, like 140 FPS on V100 GPU. They also made small models like YOLOv5s, which has only 1.7M parameters, so it is good for mobile use. But the smaller model has lower accuracy, like 89.3%. Also, this model

doesn't work very well when the face is too small or the light is very low. In such cases, accuracy drops by 15 to 20% compared to models like RetinaFace.

In the paper by Yaniv Taigman [22], they introduced DeepFace, a very powerful deep learning model for face recognition. This model uses 3D face alignment to turn the face straight (called frontalization) and then uses a nine-layer deep neural network with over 120 million parameters. They trained it on a huge dataset of 4.4 million face images and got really good results—97.35% accuracy on LFW and 91.4% on YouTube Faces. That was a big improvement over older models. But the problem is, the model is very heavy—it takes about 0.33 seconds to recognize one face on a CPU, and it also needs high-quality color images. Because of this, it's not really suitable for real-time face recognition on Android phones or low-power devices. The 3D part and the large size make it hard to use when memory and battery are limited.

In Anissa Lintang Ramadhani [23], study where they made a face recognition system for a robot using the Eigenface method, which is based on something called PCA (Principal Component Analysis). First, they detect the face using the Viola-Jones method, then they do some preprocessing like converting to grayscale, aligning the face using eye positions, and adjusting brightness with histogram equalization. They also collect many face images per person and add mirror images to increase data. Then the PCA model learns the main features, called eigenfaces. During recognition, it compares the new face with stored ones and checks if they match. They tested it on six people and got an average accuracy of 96.3%, and it was fast—less than 1 second per image. So it's good for simple or low-resource systems. But the downside is that it doesn't handle lighting or pose changes very well. It works best when the face is clearly visible from the front in good light.

Peter N. Belhumeur [24] came up with another method called Fisherface, which is more robust than Eigenface when lighting or facial expressions change. This method first uses PCA to reduce the size of the image data, and then applies Fisher's Linear Discriminant to separate different people's faces more clearly. The advantage is that it ignores parts of the image that vary a lot within the same person (like shadows or expressions) and focuses more on what's unique across different people. They tested this on the Yale and Harvard face datasets, and the results were much better than Eigenface—like in one case, Fisherface had only 4.6% error while Eigenface had 41.5%. On the Yale dataset, Fisherface had just 0.6% error in full-face images. Its speed is similar to Eigenface, but it's much more reliable in real-world situations,

especially on mobile devices—if the face is straight and clear.

Timo Ahonen [25] proposed a face recognition method using Local Binary Patterns (LBP). In this method, they divide the face into small regions and extract texture patterns from each part. These patterns are then combined to form a full feature set. The best thing about LBP is that it's very fast and works well even when lighting is not perfect. It also doesn't need any special adjustment to brightness. They tested this on the FERET dataset and got great results—97% accuracy for different expressions, 79% for lighting changes, and even 64% for faces that had aged. It did better than other popular methods like PCA and Bayesian methods. This method is lightweight, accurate, and works even if the face position is a little off. That's why it is suitable for mobile phones and low-resource systems.

Table 2.1: Comparison of Face Detection Models

Model	Accuracy (Dataset)	Speed	Hardware
Viola-Jones (2001)	93.9% De- tection Rate (MIT+CMU)	15 FPS	CPU
MTCNN (2016)	95.1% AP (FDDB) 94.3% AP (WIDER FACE)	16 FPS	GPU
RetinaFace (2020)	91.4% AP (WIDER FACE Hard)	5 FPS	CPU/GPU
BlazeFace (2019)	98.6% AP (Google Inter- nal Front-Face)	1000 FPS	Mobile GPU
YOLOv5-Face (2021)	96.8% mAP@0.5 (WIDER FACE) 93.2% mAP (FDDB)	140 FPS	GPU/TPU

Table 2.2: Comparison of Face Recognition Models

Model	Dataset Used	Accuracy / Key Metric	Processing Time
DeepFace	SFC (4.4M images)	97.35% (LFW)	0.33s/image
Eigenface (PCA)	6 users \times 40 images (custom)	96.3% (avg, custom set)	<1s/image
Fisherface (FLD)	Harvard, Yale Face Databases	Error: 4.6% (Harvard, lighting)	Similar to PCA
LBP (Ahonen et al.)	FERET (fb, fc, dup I, dup II)	97% (fb), 79% (fc), 66% (dup I), 64% (dup II)	Fast, efficient

Chapter 3

METHODOLOGY

3.1 Proposed Approach

In my proposed approach for face detection and face recognition in low-resource Android mobile devices, I created an Android app. In this app, there are two main modules one is registration and the second one is recognition. So in the registration module, a person can register his face. First, the face detection model detects the face from the image, and then the face recognition model creates and saves the embedding of that face. And later in the recognition module, whenever the user wants to recognize a face, then the face embedding is generated again and is compared with the previously registered face embeddings. If the embedding distance is minimum, then the face is recognized.

For the face detection part, I used Google ML Kit. It is an Android library which we can directly use in Android Studio. For face recognition, I used two models one is the FaceNet model in TFLite format, and the second one is the MobileFaceNet model also in TFLite format.

3.2 Face Detection

So face detection means finding the face from the image, like where is the face, its position, and size. After that, we can crop that part of the face and send it for recognition.

3.3 Face Recognition

In this, the model checks the unique features of the face. The image is first cropped, resized, and sometimes converted into grayscale before sending it to

the model.

My app has three main screens Main Activity, Registration Activity, and Recognition Activity.

3.4 Registration Activity

This screen is used for registering the face and storing it in the database. The registration process happens in four main steps:

1. First, take the image and pass it to the face detection model.
2. The model detects the face from the image.
3. The cropped face is then passed to the face recognition model, which creates an embedding.
4. Then we ask the user to give the name of the person. We save the name, roll number, branch, and the embedding in the database.

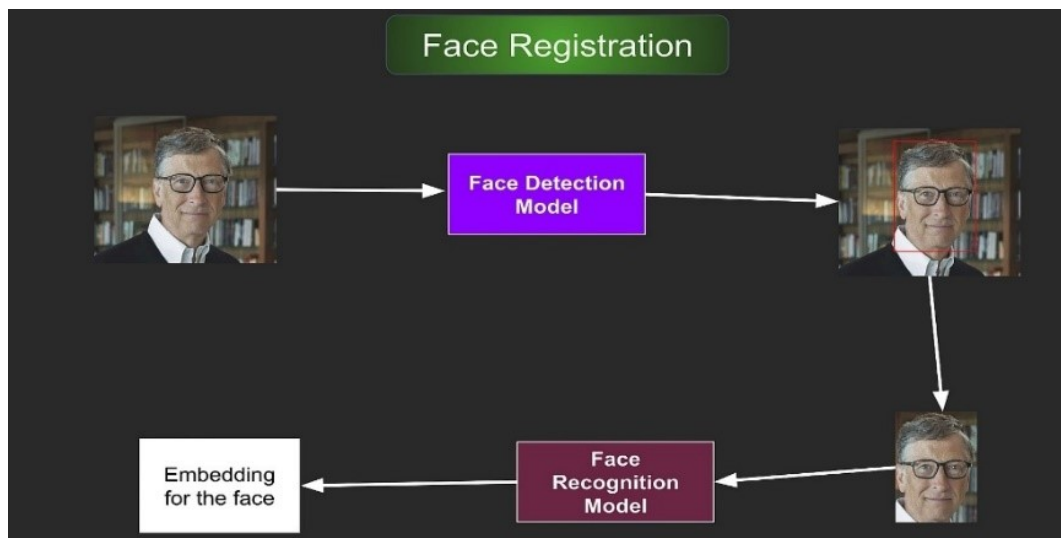


Figure 3.1: Face Registration

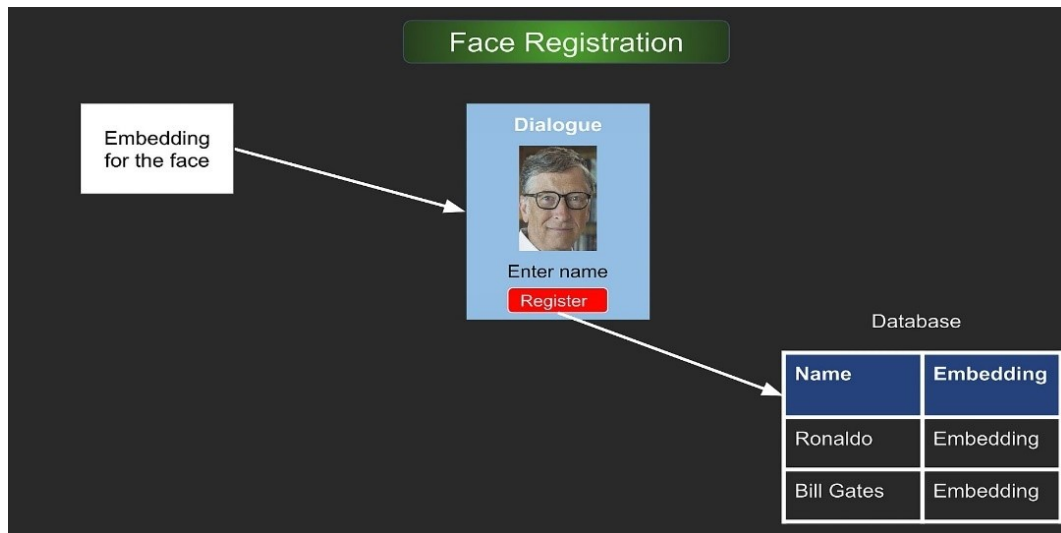


Figure 3.2: Saving face and embedding in to database

3.5 Recognition Activity

The recognition process also happens in four steps:

1. First, take the image and give it to the face detection model.
2. Detect the face from the image.
3. The cropped face is passed to the recognition model to generate the embedding.
4. Then the new embedding is compared with all the registered embeddings. The one with the smallest distance is the matched face.

When the image is captured by the camera in face registration activity it is to be converted in to the bitmap format in order to use it inside machine leaning model.

Also when we capture the image sometimes it is to be rotated by 90 degree. So in order to change the orientation of the bitmap in to portrait mode so we have to create a function that will handle this.

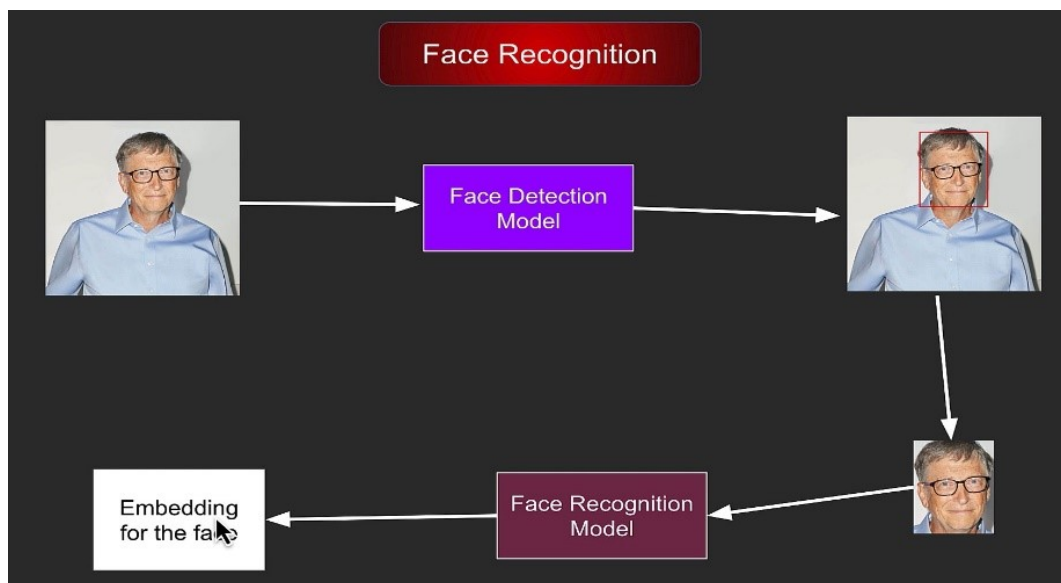


Figure 3.3: Face Recognition

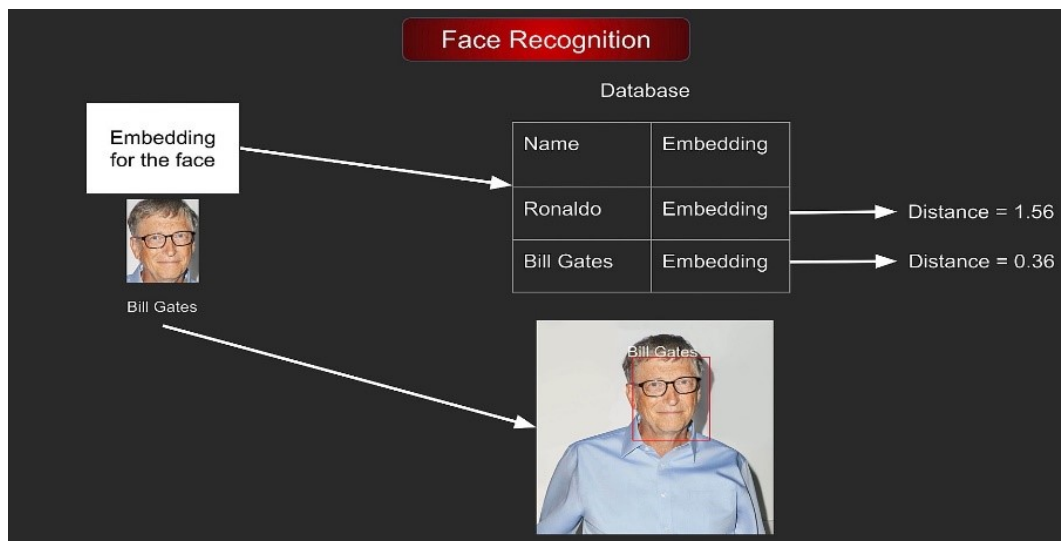


Figure 3.4: Comparing Embeddings value

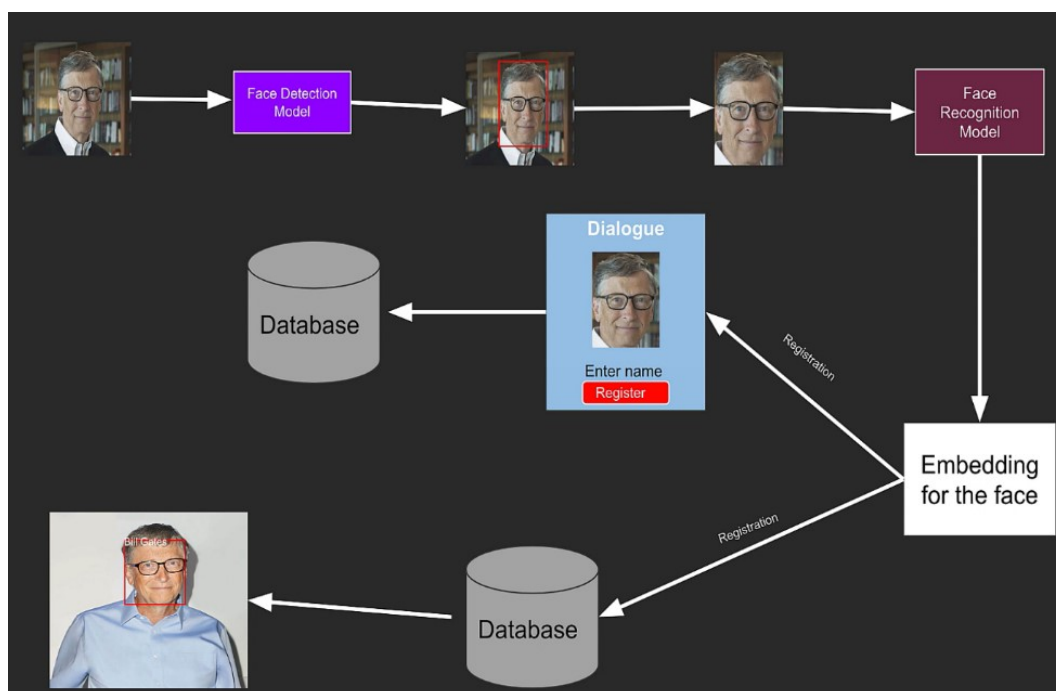


Figure 3.5: Face Registration and Recognition

3.6 TensorFlow Lite

TensorFlow Lite, or TFLite, is a small and light version of TensorFlow that is mainly made for mobile phones and small devices like Android phones and other edge devices. It is created by Google, and it helps to run machine learning models on the mobile phone itself without needing internet. So, I don't have to send the data to a server again and again.

TFLite is an open-source library, and it allows machine learning to run on the phone directly. So, even if there is no internet, we can still do things like classification or regression using this. And this is very useful for mobile apps because it works fast and gives results without delay.

One of the best things about TFLite is that it comes with tools that help in converting the models. Like, if I have a normal TensorFlow model in .pb or .h5 format, I can convert it into .tflite format using the TFLite converter. And during this conversion, I can apply optimizations like quantization and pruning, which help in making the model smaller and faster, and the accuracy also mostly stays the same.

Using TensorFlow Lite in Android is very helpful. It gives fast results, and it doesn't need internet, so the app can also work in offline mode. Also, it uses less CPU and memory, which is good for saving battery. And TFLite not only supports Android but also supports iOS and other small devices, so it is very flexible to use in different platforms.

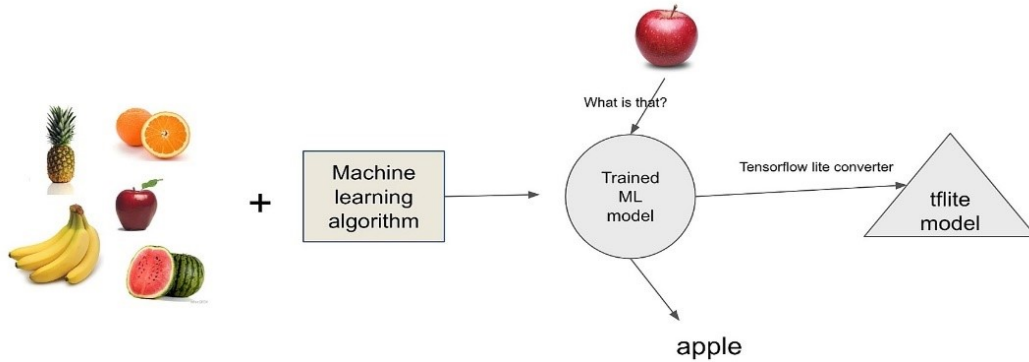


Figure 3.6: TensorFlow Lite Model

With the help of TFlite we can deploy face recognition models like FaceNet and MobileFaceNet in to the low resources android mobile devices These models are converted to the .tflite format and integrated into the Android application to perform on-device face recognition. This approach mak esure that no data reach to the round trip to the server our application is fast and secure and works fine in low resource devices.

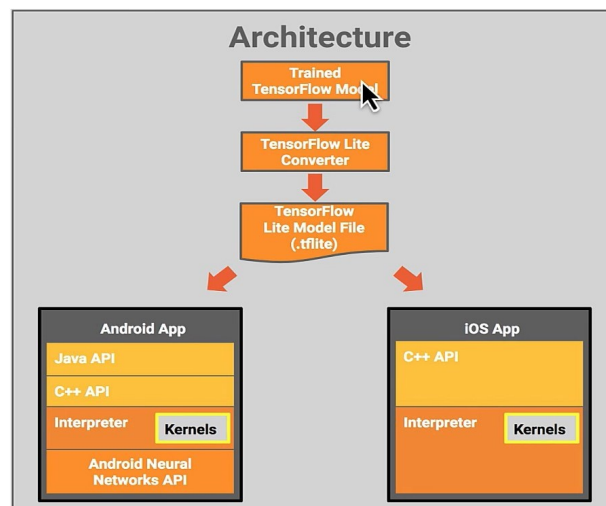


Figure 3.7: TensorFlow Lite Architecture

3.7 Quantization

Quantization is like a trick or method to make the model smaller and faster. Normally, deep learning models use 32-bit float numbers, but with quantization, we change these big numbers into smaller ones like 8-bit integers or 16-bit floats. This helps a lot when we are using the models on low-end Android phones where RAM, processing speed, and battery are all limited.

In my thesis, quantization is very important because I use it for face recognition models like FaceNet, MobileFaceNet, and ArcFaceLight when I convert them into .tflite format. After applying quantization, the size of the model becomes 4 times smaller and the speed also increases. It can run faster on mobile CPU, DSP, or NPU without losing much accuracy.

There are mainly two types of quantization. One is Post-Training Quantization, which is done after the model is already trained. It is very easy to use and works well for mobile apps. The second one is called Quantization-Aware Training, where the model is trained while keeping the quantization effects in mind. This is used when we need high accuracy and the first method doesn't give good results.

By using quantization, the face recognition models I used in my app become light and fast. It helps the app to run smoothly even on cheap or old Android phones. And this is exactly what I wanted in my thesis—to make face recognition work in real-time on low-resource Android mobile devices.

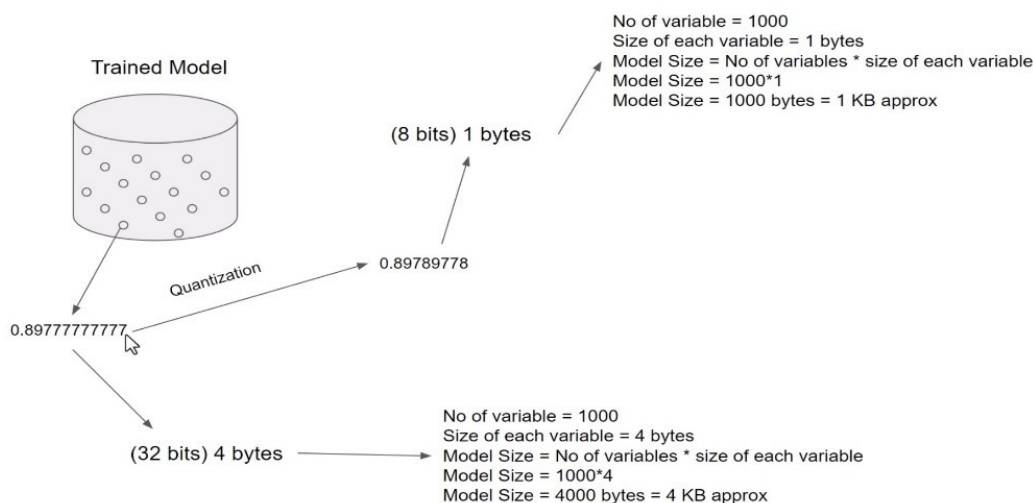


Figure 3.8: Quantization

3.8 Face Detection using ML Kit

For the face detection part, I used Google ML Kit. It is basically a light-weight library that is made specially for mobile apps. It works directly on the mobile phone, no need of internet or server, and it gives fast results even on low-end Android devices. Google already trained this library with deep learning models and provided it as a ready-made tool. We can just add it like a dependency in Android Studio and start using it.

In this library, the face detection part is done using deep learning model, but it is very optimized. So, even if the mobile phone has low RAM or slow processor, still it will work smoothly. It doesn't use the old style methods like Haar cascade or Viola-Jones, which were not good in real-world conditions like if light is low or if face is not straight. ML Kit is much better because it can detect face even if it is slightly tilted or if there is different lighting.

Now if we compare it with some other methods

Like Haar cascade method, it is not suitable for mobile phones because it is slow and not very accurate, especially if lighting or face angle changes. But ML Kit handles those problems better.

Then comes MTCNN, it is a deep learning method also, but it is heavy and slow for mobile, so it takes time. ML Kit is faster and almost same in accuracy.

RetinaFace is very good in accuracy, but it needs a lot of GPU and power, so it's not good for low-end phones. ML Kit is better here also.

BlazeFace is also mobile friendly, but it works better with front camera and straight faces. If face is from side, BlazeFace sometimes fails. ML Kit can handle side faces also better.

YOLOv5 can also detect faces but it is more for general object detection and it uses lot of memory and power. So again, ML Kit is more useful if we just want face detection in mobile apps.

ML Kit internally uses CNN model which is compressed and optimized. It uses 8-bit format (instead of 32-bit) which makes it smaller in size and faster to run. It also supports Android's hardware like GPU and NNAPI, so detection becomes faster. And all of this runs on mobile itself, so it doesn't need internet, and privacy is also maintained.

It also gives many features. Like it not only detects the face but also gives points on eyes, nose, lips, and mouth. It can also detect if the person is smiling or has eyes closed. It can track faces in videos also and give unique IDs to each person so we can identify them in next frame. That's why it is very good for real-time apps like video calls or games.

So overall, I used ML Kit for face detection in my app because it is fast, works offline, and perfect for low-resource Android devices.

3.9 Detect faces with ML Kit on Android

First u need to add this dependency below in to thr build.gradle fiel of your android application gradle file is the file where your app all dependencies and other resources that you used are there dependencies are like the APIs in android app you can use all feature of of MK kit with Ml kit dependency.

```
dependencies {  
    // ...  
    // Use this dependency to bundle the model with your app  
    implementation 'com.google.mlkit:face-detection:16.1.6'  
}
```

Once this is added, then we can start using all the features of ML Kit for detecting faces in our app.

3.9.1 Configure the face detector

Before we start detecting faces in the image, we can also configure some settings if we want better results. Like if we want more accuracy or we want to detect facial landmarks like eyes, nose, and mouth, we can use the options object.

```
// High-accuracy landmark detection and face classification  
FaceDetectorOptions highAccuracyOpts =  
    new FaceDetectorOptions.Builder()  
        .setPerformanceMode(FaceDetectorOptions.PERFORMANCE_MODE_ACCURATE)  
        .setLandmarkMode(FaceDetectorOptions.LANDMARK_MODE_ALL)  
        .setClassificationMode(FaceDetectorOptions.CLASSIFICATION_MODE_ALL)  
        .build();
```

If we want real-time face detection with face contour (like full shape of face), then we can use this:

```
FaceDetectorOptions realTimeOpts =
    new FaceDetectorOptions.Builder()
        .setContourMode(FaceDetectorOptions.CONTOUR_MODE_ALL)
        .build();
```

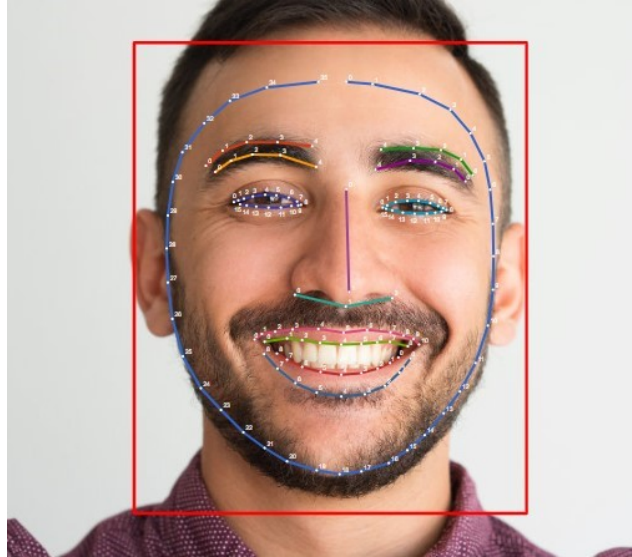


Figure 3.9: Face Detection

3.9.2 Get an instance of FaceDetector

```
FaceDetector detector = FaceDetection.getClient(options);
// Or use the default options:
// FaceDetector detector = FaceDetection.getClient();
```

3.9.3 Prepare the input image

Now, to detect face in any image, we have to convert the image into `InputImage` format. It supports different types like `Bitmap`, `media.Image`, `ByteBuffer`, or even an image file. But most of the time we use `Bitmap`, so let's go with that. Also, ML Kit recommends that the image size should be at least 480x360 pixels so that it can detect faces properly, especially if we are doing it in real-time (like from a video frame).


```
InputImage image = InputImage.fromBitmap(bitmap, rotationDegree);
```

```
this:Task<List<Face>> result =  
    detector.process(image)  
        .addOnSuccessListener(  
            new OnSuccessListener<List<Face>>() {  
                @Override  
                public void onSuccess(List<Face> faces) {  
                    // Task completed successfully  
                    // ...  
                }  
            })  
        .addOnFailureListener(  
            new OnFailureListener() {  
                @Override  
                public void onFailure(@NonNull Exception e) {  
                    // Task failed with an exception  
                    // ...  
                }  
            })  
        );
```

To create an `InputImage` object from a `Bitmap` object, make the following declaration: `InputImage image = InputImage.fromBitmap(bitmap, rotationDegree);` The image is represented by a `Bitmap` object together with rotation degrees.

3.9.4 Process the image

Now to detect the face, we just need to pass the image to the face

3.9.5 Get information about detected faces

Inside the success block, we get a list of `Face` objects. Each `Face` object contains all the details like the position of the face in the image and the things we selected in options like landmarks, expressions, etc. For example, to get the position (bounding box) of each detected face:

```
for (Face face : faces) {  
    Rect bounds = face.getBoundingBox();  
}
```

In Now if we want to draw a rectangle around the detected face in the image, then we need to use Canvas to draw on the bitmap. But before doing that, we must make the bitmap mutable, because by default bitmaps are immutable (means we can't change them). Once we make the bitmap mutable, then we can create a canvas and draw on it, including rectangles on the face locations.

3.10 Dataset Used(LFW Dataset)

LFW dataset, which means Labeled Faces in the Wild, is a popular dataset that is mostly used for testing face recognition models. It was created to check how face recognition models perform in real-world situations, like when the lighting is different, or the face is from a different angle, or the person is making some other facial expressions. It has more than 13,000 images of people's faces, and around 5,749 different people are there in this dataset. Some people have only one image, and some have more than one, so we can use it for both identification and verification tasks. The images are collected from the internet, and it contains real-life variations, so it is good for checking how well our model works in practical conditions.

In my thesis work, I used this dataset to test my FaceNet and Mobile FaceNet models, which I used for face recognition in Android mobile devices. Before giving the face to the model, I resized all the images based on the input size of the model and passed them through the models to generate embeddings. Then I used those embeddings for training and testing using SVM classifier, and I calculated the accuracy, precision, recall, F1 score, inference time, and model size. This helped me to compare both the models and decide which one is better to use in my app. FaceNet gives better accuracy but takes more time and space, and MobileFaceNet gives slightly less accuracy but it is faster and smaller, so I used MobileFaceNet in my attendance app.

3.11 Face Recognition

Face recognition: given an image of a person's face, identify who the person is (from a known dataset of registered faces) therefore Face Recognition is the process of differentiating faces and classifying them to identify a person. Therefore, to distinguish the two identical (but not the same).

In my app, I used face recognition to identify people by their face image, like matching it from already registered faces. Basically, face recognition is used to find out whose face it is by comparing it with known ones, even if two faces look almost the same. In my Android app, I used two models for face recognition:

FaceNet and MobileFaceNet. I wanted to check which one works better in Android phones, and whichever works best, I use that in my app. So first, I added the FaceNet model into the assets folder of the Android project, and then I created some classes like FaceClassifier interface and TLiteFaceRecognition class. Same steps I did for the MobileFaceNet model also.



Figure 3.10: Face Recognition

3.11.1 The FaceNet approach

FaceNet is a deep learning model made by Google. It takes face images and changes them into a 128-number vector which we call embeddings. The main idea of this model is it tries to keep similar faces close together and different faces far from each other in that number space. It uses something called triplet loss to do that. This helps in face matching, verification, and grouping faces properly.

In my project, I used FaceNet to create embeddings of each face. These embeddings are then compared using distance formula like Euclidean distance. If the distance is small, then it's the same person, otherwise not. FaceNet is very accurate, but the problem is that the full model is heavy and not suitable for low-end Android phones. So I converted the model into TFLite format and used optimization methods like quantization to reduce size and make it run better on mobile. Still, it takes around 3.5 seconds to run. But it gives good results and works offline also.

FaceNet is a CNN based model So, CNN full form is Convolutional Neural Network. It is used for image-based problems like face detection and recognition. It works better than normal neural networks (ANNs) when we are

dealing with images. In CNN, we don't give the full image directly to the model like in ANN, instead CNN uses filters which scan the image part by part and extract features like edges, corners, etc.

Now, if we talk about how CNN actually works, the first step is convolution layer. In this layer, we use a small filter which moves across the image and does multiplication with small parts of the image and gives output. This output is called feature map, and it shows where the important features are in the image.

After this, we apply ReLU, which is a function that just removes negative values from the feature map and keeps only the positive ones, because they are more useful for detecting features.

Then comes the pooling layer. This layer is used to reduce the size of the feature map. The most common one is max pooling, which just picks the largest value from a small block of the image. This helps the model to work faster and makes it focus on only the important parts of the image.

At the end, we have fully connected layers. These are like normal neural networks where all the values are combined and final prediction is done — like whether the face is present or not, or who the person is.

So, this is how CNN works step by step — first convolution for feature extraction, then ReLU for non-linearity, then pooling for reducing size, and finally fully connected layers for output.

In my thesis, I used CNN-based models like FaceNet and MobileFaceNet for face recognition. These models convert the face into embeddings and match them to recognize the person. Since they are based on CNN, they are able to extract useful features from face images and give accurate results.

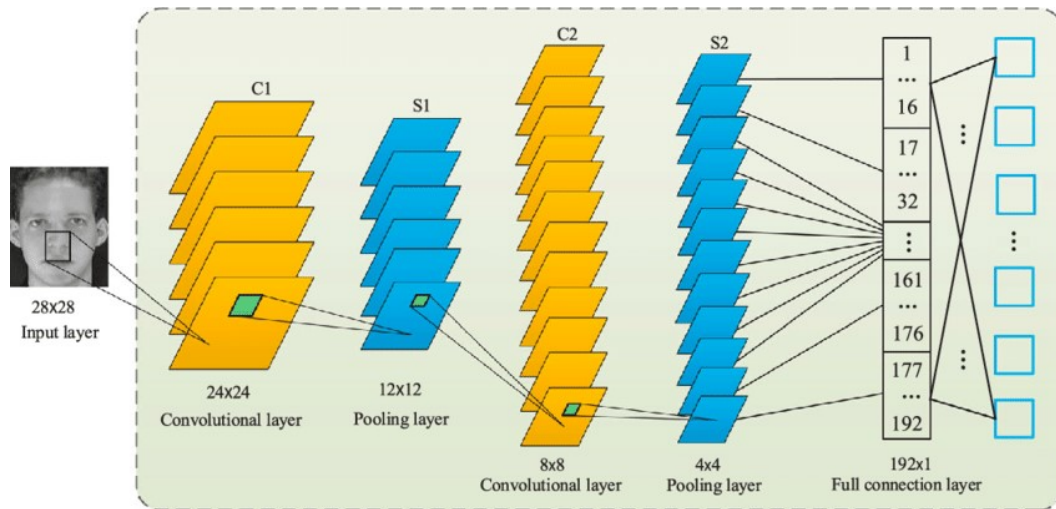


Figure 3.11: CNN Architecture

Using FaceNet with TFLite:

- Compare faces offline
- Accurate results
- Execution time ≈ 3.5 seconds

3.11.2 MobileFaceNet approach

MobileFaceNet is another face recognition model, but it's made especially for mobile and small devices. It's developed by Watchdata Inc. from China. This model is based on lightweight layers, similar to MobileNet, and it replaces heavy layers with something called depthwise separable convolutions. Because of this, the model becomes very small and fast.

Even though MobileFaceNet is only around 4 MB in size, it still gives accuracy similar to big models like FaceNet. So in my thesis, I used MobileFaceNet as a better option for real-time face recognition in low-resource Android devices. I took the model, converted it to .tflite format, and applied quantization for optimization. Then I added this model in my Android app. It runs fast, takes less memory and power, and the accuracy is also fine. So it's a good balance between speed and performance, and that's why I chose it for my final app.

First of all we need to add the dependencies of TensorFlow Lite Model

implementation ("org.tensorflow:tensorflow-lite:+")

with the help of this we can load the tensorflow Lite into our model. after that you need to create an object of face classifier interface so that you can add the value of the embeddings and then you can add them in to the table with respect to the name present in to the table when the dialog is open.

A dialog box is open and soon as the registered face is detected by the face detector it asks for some things like student name, roll no, branch and there is also a register button also there along with image of the person in the image view.

By clicking on the register the face along with the embedding value is saved in to the android local database that can be used further in order to get result and compare embeddings.

The value of the embedding are saved in to hash map type string and embedding first, This hash map is present in Main Activity file so that the value of name and embedding data do not change and deleted when app is not in working condition or not active. In that case also the data is saved.

After saving the data when the cropped image is given as a parameter to recogniseImage method it will compare the embedding of the image face with the embeddings of other faces present in the database and whose distance is minimum it will give us that face name and distance.

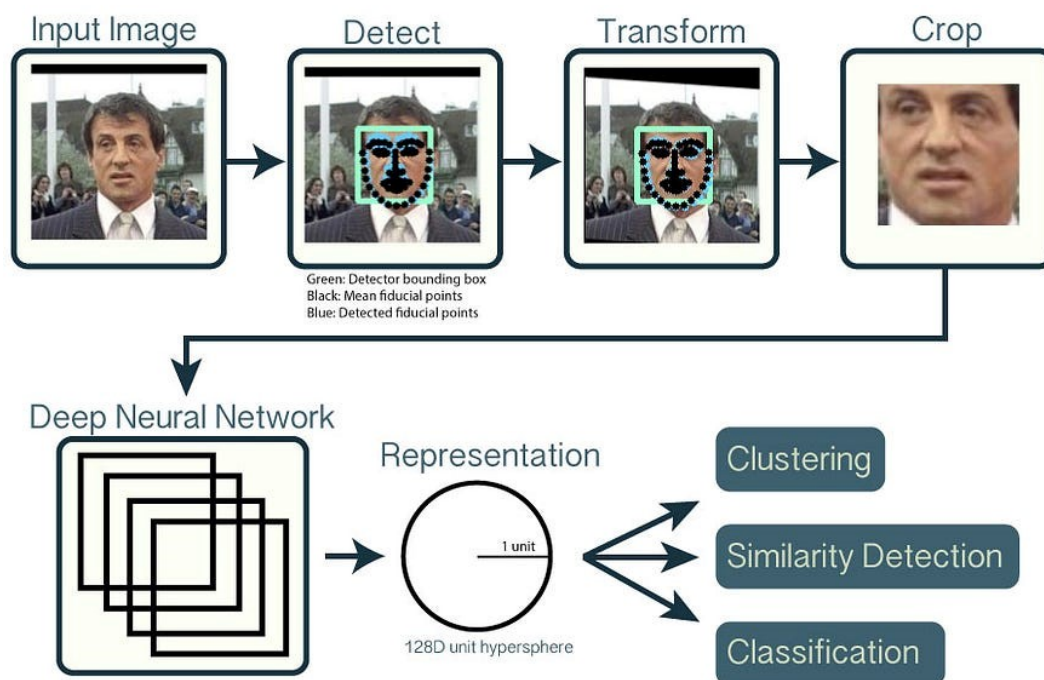


Figure 3.12: Face Recognition Flow

3.11.3 How face recognition system works in my app

So first, how this whole face recognition thing works step-by-step is like this:

- First of all, we detect the face from the input image.
- After that, we align the face properly. We use the face landmarks (like eyes, nose) to adjust the face so that in all images, eyes and everything are at the same position.
- Then we crop the face and resize it, so we can give it to the recognition model. In this step, we also do some image cleaning like normalization or whitening to make it ready for the model.
- After this, the main part comes — the Deep Neural Network (DNN). This model takes the face as input and gives us a 128-number vector called embedding. This embedding is used to compare with other embeddings. So like, if we have two faces F1 and F2, we convert them into E1 and E2 embeddings, and then compare both using Euclidean distance. If the distance is small, it means both are of the same person.



Figure 3.13: Face Recognition on Image

3.11.4 Adding the face recognition step

First, I added the .tflite model file (FaceNet or MobileFaceNet) inside the assets folder of the Android project. Then I changed some settings in the DetectorActivity class. Like I set the input size to 112 and kept quantized to false. Also, I changed the Classifier interface name to SimilarityClassifier because now the model is giving similarity results, not confidence. So instead of saying how confident the model is, we check how close the faces are using distance. Smaller the distance, better the match. If the distance is 0, that means both images are exactly the same. After that, I also changed the way I store the data. I used a simple dictionary where the name of the person is saved along with the recognition details (which includes the embeddings). In the recognizeImage() method, we take the face, get its embeddings, and then compare it with all the saved ones by doing a simple linear search — means we check one by one to find the closest match.

Chapter 4

RESULTS and DISCUSSION

In results shows that face detection and recognition in low resources android mobile devices is implemented in this app its kind of a attendance app which is using the ML kit for face detectiin and FaceNet/MobileFaceNet model in TFlite format for the detection purpose and keep the record of all the detected faces inside device itself.

This is the Main Activity consist of two buttons one button for login as a teacher and second button for login as a student which will progress you towards respective activities.

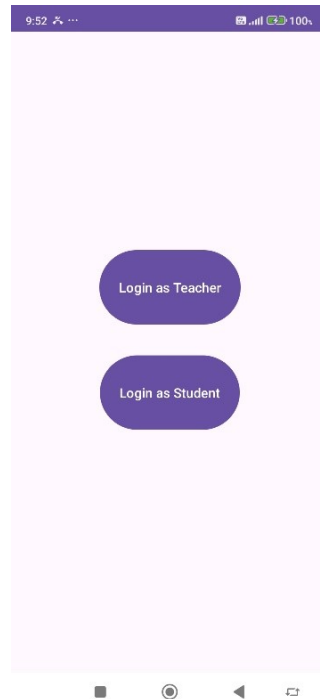


Figure 4.1: Main Activity User Interface

After pressing the button for login as a teacher you are directed towards the Teacher Login Activity in which teacher have to fill his login ID and password there is an option of view password as well. On clicking on login button Registered student Details Activity opens up.

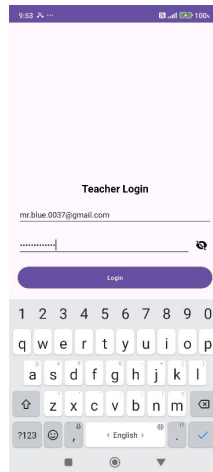


Figure 4.2: Teacher Login Activity

Registered Student Details Activity contains all the student details who registered themselves by detecting there face in the app and filling there roll no and branch

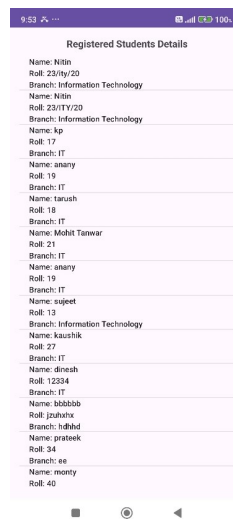


Figure 4.3: Registered Student Details Activity

On clicking login as student in the Main Activity you are directed towards

the Student Login Activity in which student have to fill his login ID and password there is an option of view password as well. On clicking on login button Registered student Details Activity opens up.

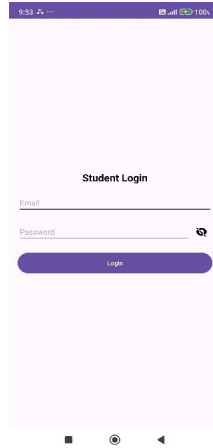


Figure 4.4: Student Login Activity

On successful login of student, Student Dashboard Activity opens in which there is an image view which take the image of the person and displayed over here. There are Two button are also created named Registration and Recognition. On clicking registration the Registration Activity will open and on clicking recognition the Recognition Activity is opened.

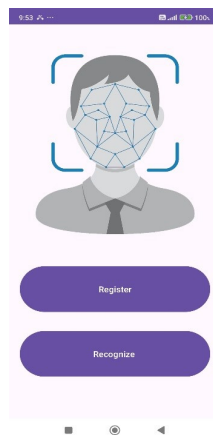


Figure 4.5: Student Dashboard Activity

On clicking the Registration and Recognition buttons in Main Activity Registration and Recognition Activity are opened. Both the activities have the same layout. They contains two clickable card views and an image view

one clickable card view is to open gallery and take image from the gallery and other card view is for opening the camera and click the image and then this image is to be displayed over the image view

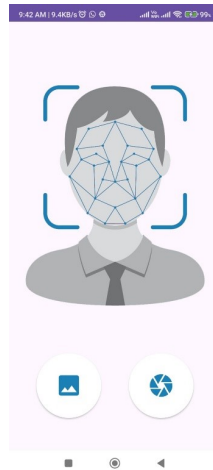


Figure 4.6: Registration and Recognition Activity User Interface

As the image is displayed over the image view the ML kit face detection detect the face present in the image id there are multiple image present in the image than multiple images of faces are detected and a red square are appeared over the images which indicated that the face is detected.

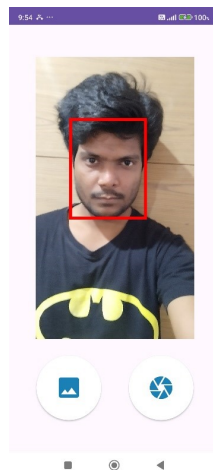


Figure 4.7: Face Detection of app

In this image the registration dialogue is open as soon as face is detected by the face detector. This registration dialogue is opened multiple times for

multiple faces and it contain the student information like student name , student roll no and student branch than on clicking the register button all this information is saved in to the database along with the embeddings of the image in this way the face and the other information are saved in to the database after detecting it

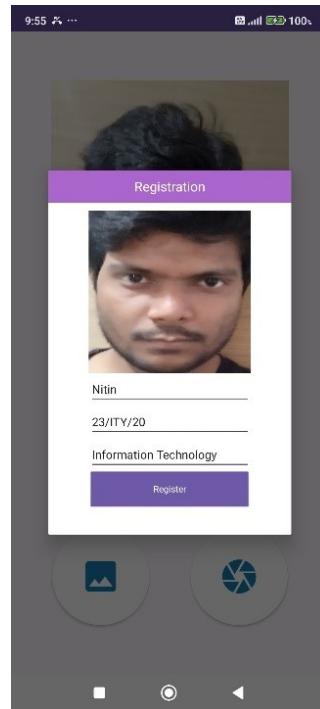


Figure 4.8: Dialouge box for face Registration

This how the face registration works after face registration we have to recognize the regsitered face from the dataset for that the steps will be the same we have to click an iamge using camera or select an image from the gallary the we have to recognize.

As soon as we select the image it shows over the image view and the embeddings of that image is compared with the embeddings of the images that are present in to data dataset that we created after registering mutliple images of face.

After subtracting the embedding values of the face image with each of embedding value of other registered faces images the value whose distance is minimum is the face of the person that is how we get the recognised facealong with the distance.

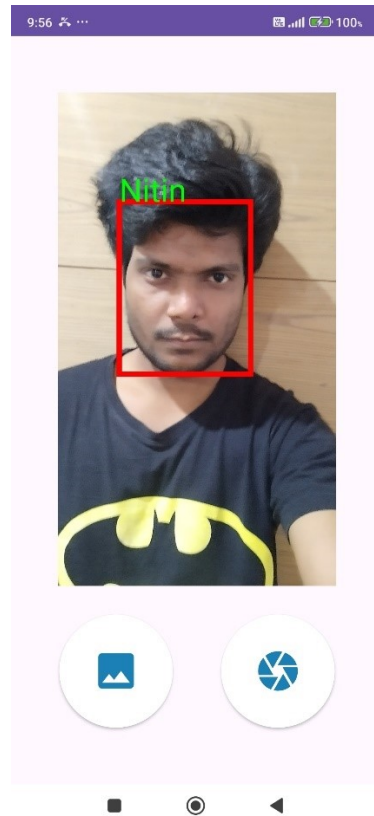


Figure 4.9: Face Recognition with name and Distance

In last as soon as the face is detected the name of the person in the image is saved inside the list view of Present Student Details Activity in which all the students details who recognised there faces will be add in the list as soon as there face is recognised



Figure 4.10: Present Student Details Activity

when comapre the FaceNet model with Mobile FaceNet model by evaluate on the LFW dataset we can see that the accuracy of the FaceNet model is high but the Time it takes for the inference is also high as compare to the MobileFaceNet model also size of the model is also big.

In table 4.1 we can clearly see that the FaceNet(TFlite) model Accuracy, Precision, Recall, F1 Score is better as comapre to teh MobileFaceNet(TFlite) but otehr parameters like Inference time and Model Size is high as compare to MobileFaceNet which is required for low resource android mobile devices.

Table 4.1: Comparison of Face Recognition Models: FaceNet vs MobileFaceNet on LFW Dataset

Metric	FaceNet	MobileFaceNet
Accuracy	99.44%	71.60%
Precision	99.54%	70.54%
Recall	99.44%	71.60%
F1 Score	99.42%	68.01%
Inference Time	0.063222 sec	0.012811 sec
Model Size	22.54 MB	5.00 MB

Table 4.2: Comparison of various Face Recognition Models

Model	Dataset Used for validation	Accuracy	Processing Time
DeepFace	LFW	97.35%	0.33s/image
Eigenface (PCA)	LFW	78.41%	0.00038s/image
Fisherface (FLD)	LFW	80.55%	1.1997s/image
LBP	LFW	29.05%	5.28233s/image
FaceNet	LFW	99.4%	0.063222s/image
MobileFaceNet	LFW	71.60%	0.012811s/image

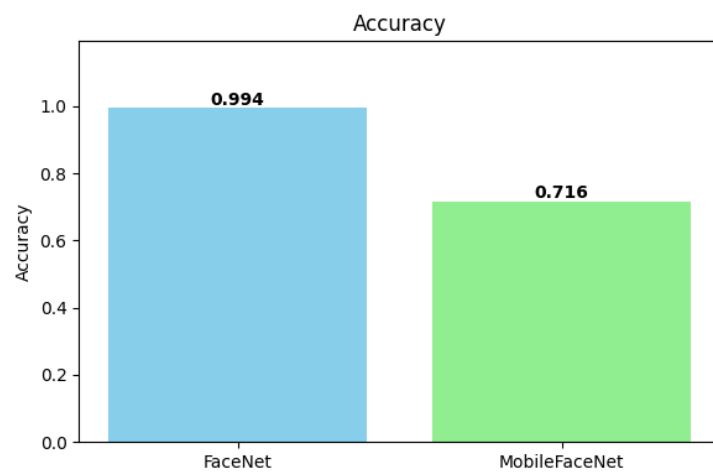


Figure 4.11: Accuracy FaceNet(TFlite) vs MobileFaceNet(TFlite)

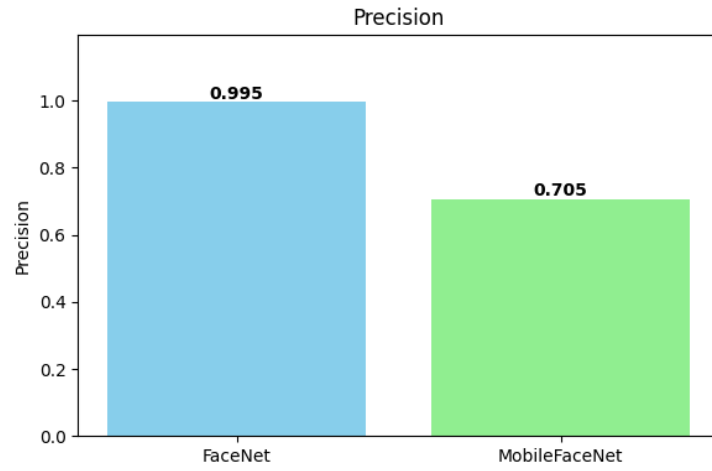


Figure 4.12: Precision FaceNet(TFlite) vs MobileFaceNet(TFlite)

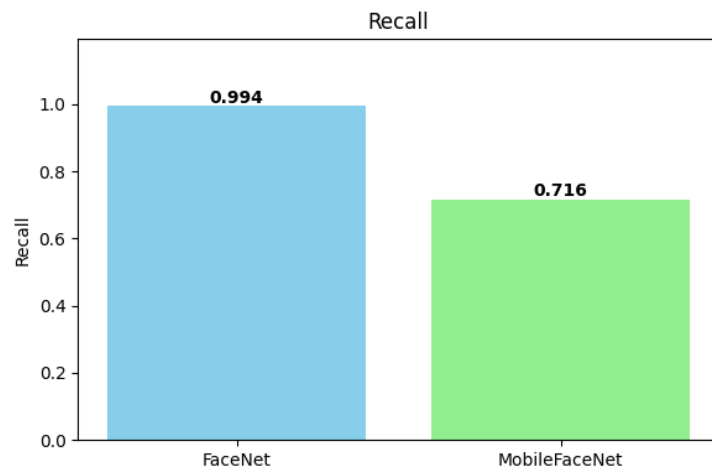


Figure 4.13: Recall FaceNet(TFlite) vs MobileFaceNet(TFlite)

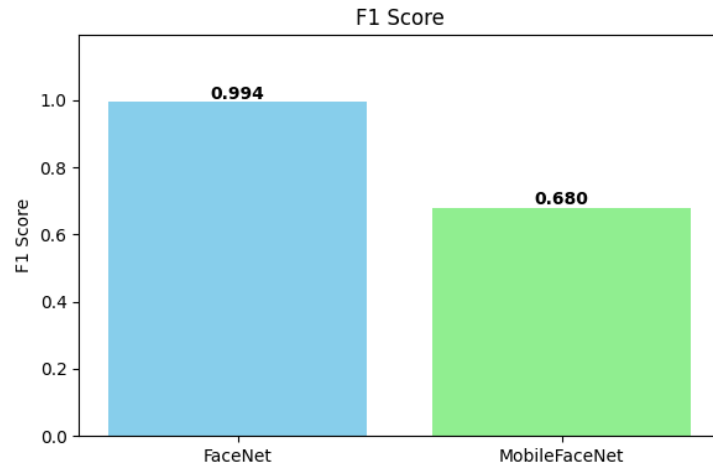


Figure 4.14: F1 Score FaceNet(TFlite) vs MobileFaceNet(TFlite)

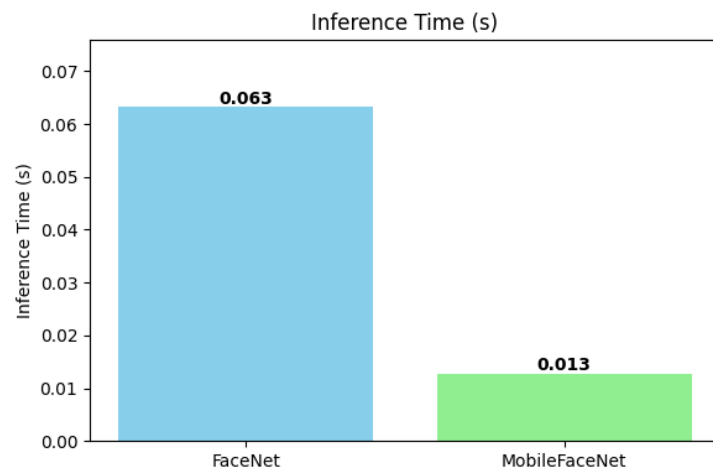


Figure 4.15: Inference Time FaceNet(TFlite) vs MobileFaceNet(TFlite)

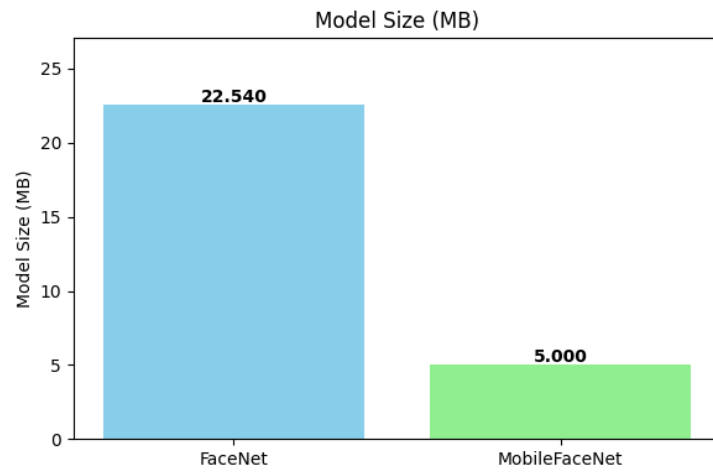


Figure 4.16: Model Size FaceNet(TFlite) vs MobileFaceNet(TFlite)

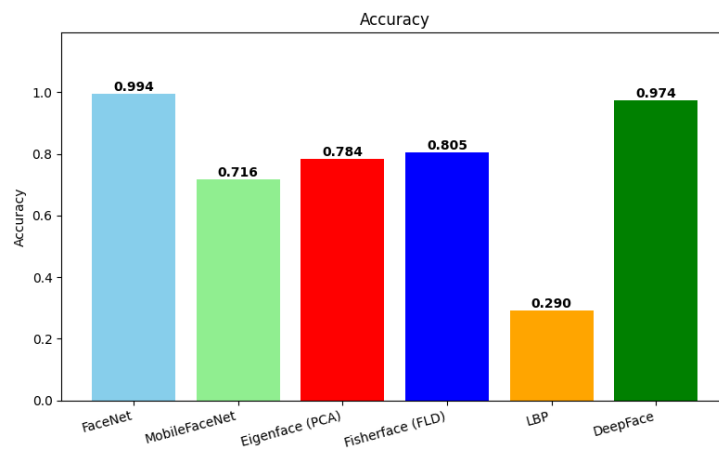


Figure 4.17: Accuracy of various face recognition models

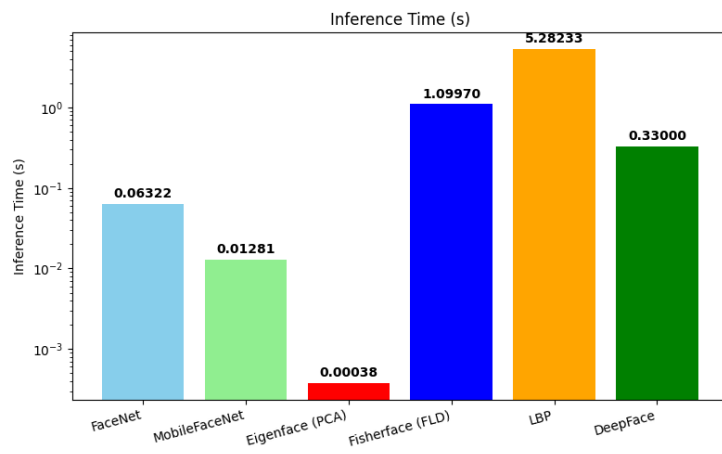


Figure 4.18: Inference time of face recognition model

Chapter 5

CONCLUSION AND FUTURE SCOPE

In today's time, everyone is using mobile phones. Mobile phones are not just for calling now, people are using them for so many things like banking, shopping, learning, and even to verify their identity. So in my thesis, I tried to bring the face detection and face recognition system into low-end Android mobile phones. These mobile phones don't have much RAM or fast processor or big battery, so the challenge was to make it work properly on them.

The main aim of my project was to create one system that can detect and recognize faces fast and accurately, and everything should run inside the mobile only. No internet or cloud or big hardware should be needed. So for face detection, I used the Google ML Kit, because it is already made library by Google for Android, and it works very smoothly. And for face recognition, I used lightweight versions of FaceNet and Mobile FaceNet models, both are in TFLite format so they can work on mobile.

This type of face recognition system can be used in real life also, like for student attendance, checking who is entering a building (access control), or even for verifying identity. So overall, I proved that even low-end Android phones can do this type of work properly.

So in conclusion we can say that Face detection and recognition can run inside the phone itself, no need for internet or server. The system works fine in normal light and normal face conditions, but if the light is bad or the face is turned or covered, then sometimes it may not work properly. Mobile FaceNet model is small and fast, so it is good for phones, especially in those places where people don't have very expensive phones or fast internet.

Future work can focus on developing even more lightweight models that maintain high accuracy while reducing computational requirements. Advanced quantization techniques and neural architecture search could help create bet-

ter mobile-optimized models. In future the feature can be enhanced by Improve the system to handle multiple faces simultaneously in crowded environments, Add anti-spoofing mechanisms to prevent photo-based attacks, Implement on-device learning to adapt to changing facial features over time. At the same time model can be used for broader applications like Integration with IoT devices for smart home applications, Development of privacy-preserving federated learning approaches, Extension to other biometric modalities for multi-modal authentication.

Bibliography

- [1] A. Jain, A. Ross, and S. Prabhakar, “An introduction to biometric recognition,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14.
- [2] H. Farooq, A. B. Kahng, and M. S. J. S. Lim, “Biometric authentication in smartphones,” *IEEE Consumer Electronics Magazine*, vol. 6, no. 2, pp. 59–65, 2017.
- [3] S. Zafeiriou, C. Zhang, and Z. Zhang, “A survey on face detection in the wild: Past, present and future,” *Computer Vision and Image Understanding*, vol. 138, pp. 1–24, 2015.
- [4] A. Bhattacharya *et al.*, “Smart hospital management system using face recognition,” in *Proc. IEEE ICCCA*, 2020.
- [5] S. Biswas, M. S. Kankanhalli, and A. Roy-Chowdhury, “Privacy-preserving face recognition,” *IEEE Transactions on Image Processing*, vol. 27, no. 4, pp. 2118–2131, 2018.
- [6] L. Fussey and D. Murray, “Independent report on the london metropolitan police service’s trial of live facial recognition technology,” University of Essex, 2019.
- [7] D. J. Phillips and E. Curry, “Retail surveillance and the face,” *International Journal of Communication*, vol. 11, pp. 2994–3014, 2017.
- [8] T. Howard, “Embedded deep learning for real-time facial recognition on mobile devices,” in *Proc. IEEE ICRA*, 2018.
- [9] M. Sandler *et al.*, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proc. IEEE CVPR*, 2018.
- [10] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, “A survey of model compression and acceleration for deep neural networks,” *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 126–136, 2018.

- [11] A. Ignatov *et al.*, “Ai benchmark: Running deep neural networks on android smartphones,” in *Proc. ECCV Workshops*, 2018.
- [12] S. Mehta and S. E. Sarma, “Edge computing for real-time facial recognition in low-resource environments,” in *Proc. IEEE ICMLA*, 2021.
- [13] D. A. Das, N. Borisov, and M. Caesar, “Do you hear what i hear? fingerprinting smart devices through embedded acoustic components,” in *Proc. ACM CCS*, 2014.
- [14] M. S. Mubarak Alburaiqi, G. M. Johar, R. A. Abbas Helmi, and M. H. Alkawaz, “Mobile based attendance system: Face recognition and location detection using machine learning,” in *2021 IEEE 12th Control and System Graduate Research Colloquium (ICSGRC)*, Shah Alam, Malaysia, 2021.
- [15] A. M. B. M. Azli, H. K. Mammi, M. M. Din, and A. Abdul-Samad, “Face-recognition based attendance authentication system,” in *2023 International Conference on Data Science and Its Applications (ICoDSA)*, Bandung, Indonesia, 2023.
- [16] A. A. Raj, M. Shoheb, K. Arvind, and K. S. Chethan, “Face recognition based smart attendance system,” in *2020 International Conference on Intelligent Engineering and Management (ICIEM)*, London, UK, 2020.
- [17] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Kauai, HI, USA, 2001.
- [18] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, “Joint face detection and alignment using multitask cascaded convolutional networks,” *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.
- [19] M. E. Wibowo, A. Ashari, A. Subiantoro, and W. Wahyono, “Human face detection and tracking using retinaface network for surveillance systems,” in *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, Toronto, ON, Canada, 2021.
- [20] V. Bazarevsky, Y. Kartynnik, A. Vakunov, K. Raveendran, and M. Grundmann, “Blazeface: Sub-millisecond neural face detection on mobile gpus,” arXiv preprint arXiv:1907.05047, 2019.
- [21] D. Qi, W. Tan, Q. Yao, and J. Liu, “Yolo5face: Why reinventing a face detector,” arXiv preprint arXiv:2105.12931, 2022.

- [22] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “Deepface: Closing the gap to human-level performance in face verification,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Columbus, OH, USA, 2014.
- [23] A. L. Ramadhani, P. Musa, and E. P. Wibowo, “Human face recognition application using pca and eigenface approach,” in *Second International Conference on Informatics and Computing (ICIC)*, Jayapura, Indonesia, 2017.
- [24] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, “Eigenfaces vs. fisherfaces: recognition using class specific linear projection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 711–720, 1997.
- [25] T. Ahonen, A. Hadid, and M. Pietikainen, “Face description with local binary patterns: Application to face recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 2037–2041, 2006.



DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Shahbad Daultpur, Main Bawana Road, Delhi-42

PLAGIARISM VERIFICATION

Title of the Thesis Face Detection And Recognition In Low Resource Android Mobile Devices

Total Pages 51 Name of the Scholar Nitin

Supervisor (s)

- (1) Prof. Dinesh Kumar Vishwakarma
(2) _____
(3) _____

Department Information Technology

This is to report that the above thesis was scanned for similarity detection. Process and outcome is given below:

Software used: turnitin Similarity Index: 7%, Total Word Count: 8176

Date: 28/05/2025

Nitin
Candidate's Signature

Signature of Supervisor(s)

Nitin_23ITY20_MTech_Thesis_Face_detection_and_recogniti...



Delhi Technological University

Document Details

Submission ID

trn:oid::27535:98152118

Submission Date

May 28, 2025, 2:45 PM GMT+5:30

Download Date

May 28, 2025, 2:49 PM GMT+5:30

File Name

Nitin_23ITY20_MTech_Thesis_Face_detection_and_recognition_in_low_resources_android_mobile_....pdf

File Size

1.6 MB

41 Pages

8,176 Words

39,417 Characters





7% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.




Filtered from the Report

- Bibliography
- Quoted Text

Match Groups

-  **46 Not Cited or Quoted 6%**
Matches with neither in-text citation nor quotation marks
-  **9 Missing Quotations 1%**
Matches that are still very similar to source material
-  **0 Missing Citation 0%**
Matches that have quotation marks, but no in-text citation
-  **0 Cited and Quoted 0%**
Matches with in-text citation present, but no quotation marks

Top Sources

- 2%  Internet sources
- 2%  Publications
- 5%  Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- 46** Not Cited or Quoted 6%
Matches with neither in-text citation nor quotation marks
- 9** Missing Quotations 1%
Matches that are still very similar to source material
- 0** Missing Citation 0%
Matches that have quotation marks, but no in-text citation
- 0** Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

- 2% Internet sources
- 2% Publications
- 5% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

- Submitted works**
Hofstra University on 2022-05-17 <1%
- Submitted works**
Istanbul Aydin University on 2021-06-14 <1%
- Internet**
developers.google.com <1%
- Submitted works**
University of Wolverhampton on 2021-05-05 <1%
- Submitted works**
University of Greenwich on 2015-02-03 <1%
- Publication**
Aditya Yadav, Shauryan Singh, Muzzamil Siddique, Nileshkumar Mehta, Archana ... <1%
- Submitted works**
Coventry University on 2023-04-11 <1%
- Submitted works**
Technical University of Liberec on 2024-05-12 <1%
- Submitted works**
Terna Engineering College on 2023-04-23 <1%
- Publication**
Dheeraj Patel, Kumar Setu, Ramanand Sharma, Dinesh K. Vishwakarma. "Android... <1%

11	Internet	firebase.google.com	<1%
12	Internet	www.mdpi.com	<1%
13	Submitted works	The Indian Institute Of Management And Engineering Society on 2025-02-17	<1%
14	Internet	jivp-urasipjournals.springeropen.com	<1%
15	Submitted works	Manchester Metropolitan University on 2024-10-04	<1%
16	Submitted works	Troy University on 2025-03-03	<1%
17	Submitted works	University of Hertfordshire on 2022-07-16	<1%
18	Submitted works	University of Wollongong on 2023-12-08	<1%
19	Internet	dr.ntu.edu.sg	<1%
20	Internet	mediatum.ub.tum.de	<1%
21	Internet	repository.unika.ac.id	<1%
22	Internet	www.ee.oulu.fi	<1%
23	Publication	Bashir, Sirosh. "Face Detection and Recognition in Low Illumination Environment ...	<1%
24	Publication	Mahmul Huq, Javier Garrigos, Jose Javier Martinez, JoseManuel Ferrandez, Edu...	<1%

25	Submitted works	National College of Ireland on 2022-05-15	<1%
26	Publication	Thangaprakash Sengodan, Sanjay Misra, M Murugappan. "Advances in Electrical ...	<1%
27	Submitted works	University of Birmingham on 2023-03-15	<1%
28	Submitted works	University of Stirling on 2024-06-14	<1%
29	Submitted works	University of Ulster on 2012-05-10	<1%
30	Submitted works	Xiamen University on 2022-07-02	<1%
31	Submitted works	Xiamen University on 2024-07-01	<1%
32	Internet	datahacker.rs	<1%
33	Internet	hdl.handle.net	<1%
34	Internet	herkules oulu.fi	<1%
35	Internet	www.researchgate.net	<1%
36	Submitted works	University of Greenwich on 2014-04-23	<1%
37	Submitted works	University of Lancaster on 2025-04-07	<1%
38	Submitted works	University of Warwick on 2024-03-11	<1%

39

Publication

Prajwol Chhetri, Sunil Raut Kshetri. "Decoding Facial Recognition: Analyzing Stan... <1%

40

Submitted works

University of Macau on 2023-05-12 <1%

41

Submitted works

universititeknologimara on 2025-01-27 <1%

e-Receipt for State Bank Collect Payment



REGISTRAR, DTU (RECEIPT A/C)

BAWANA ROAD, SHAHABAD DAULATPUR, , DELHI-110042
Date: 28-May-2025

SBCollect Reference Number :	DUO1242997	Category :	Miscellaneous Fees from students
Amount :	₹3000		
University Roll No :	23/ITY/20		
Name of the student :	Nitin		
Academic Year :	2024-25		
Branch Course :	Master of Technology in Information Technology		
Type/Name of fee :	Others if any		
Remarks if any :	M.Tech Dissertation Fees		
Mobile No. of the student :	8368099458		
Fee Amount :	3000		
Transaction charge :	0.00		
Total Amount (In Figures) :	3,000.00	Total Amount (In words) :	Rupees Three Thousand Only
Remarks :	M.Tech Dissertation Fees	Notification 1:	Late Registration Fee, Hostel Room rent for internship, Hostel cooler rent, Transcript fee (Within 5 years Rs.1500/- & \$150 in USD. More than

& \$150 in USD, More than
5 years but less than 10
years Rs.2500/- & \$250 in
USD, More than 10 years
Rs.5000/- & \$500 in USD)
Additional copies Rs.200/-
each & \$20 in USD each,
I-card fee,Character
certificate Rs.500/-.

Notification 2: Migration Certificate
Rs.500/-, Bonafide
certificate Rs.200/-,
Special certificate (any
other certificate not
covered in above list)
Rs.1000/-,Provisional
certificate Rs.500/-,
Duplicate Mark sheet
(Within 5 years Rs.2500/-
& \$250 in USD, More than
5 years but less than 10
years Rs.4000/- & \$400 in
USD, More than 10 years
Rs.10000/- & \$1000 in
USD)

Thank you for choosing SB Collect. If you have any query / grievances regarding the transaction, please contact us

Toll-free helpline number i.e. 1800-1111-09 / 1800 - 1234/1800 2100

Email -: sbcollect@sbi.co.in