Face Detection and Recognition using a smartphone

Laurențiu-Ionuț Moloman [1]
dept. CTI, Grupa 1306B
Universitatea Tehnică ”Gheorghe Asachi” din Iași
Iași, România
laurentiu-ionut.moloman@student.tuiasi.ro

Eduard-Constantin Minea [2]
dept. CTI, Grupa 1306B
Universitatea Tehnică ”Gheorghe Asachi” din Iași
Iași, România
eduard-constantin.minea@student.tuiasi.ro

Ștefan-Daniel Achirei [3]
dept. CTI
Universitatea Tehnică ”Gheorghe Asachi” din Iași
Iași, România
stefan-daniel.achirei@academic.tuiasi.ro

Vasile-Ion Manta [4]
dept. CTI
Universitatea Tehnică ”Gheorghe Asachi” din Iași
Iași, România
vasile-ion.manta@academic.tuiasi.ro

***Abstract*—The current project focuses on the development and implementation of a facial detection and recognition system using mobile devices, with a focus on smartphones. Facial detection and recognition are essential aspects of modern technology, with a variety of applications in security, authentication, and device interaction.**

***Index Terms*—face detection, face recognition, ML Kit, CameraX, TensorFlow Lite, FaceNet**

I. Introduction

The realm of facial recognition has emerged as a transformative force, permeating a wide range of applications, from streamlining access control in secure environments to unlocking digital devices and verifying identities in police records. At its core, facial recognition technology revolutionizes the way we interact with the world around us, empowering us to seamlessly navigate a world increasingly reliant on biometric authentication.

At the heart of facial recognition lies the ability to capture, analyze, and compare patterns of facial features, such as the intricate geometry of the eyes, nose, and mouth. This intricate process, meticulously executed by sophisticated algorithms, enables the technology to identify individuals with remarkable accuracy, unlocking a world of possibilities.

One of the most prominent applications of facial recognition lies in facial biometrics, a cornerstone of modern security systems. Facial biometrics stands out for its unparalleled convenience and efficiency, seamlessly integrating into our daily lives. From unlocking smartphones and accessing restricted areas to providing secure access to sensitive information, facial biometrics has transformed the way we interact with technology.

The steps involved in the process are:

1) **Detecting and locating the human faces in the image**
2) **Extracting the facial features**
3) **Comparing the detected face with the database.**

[A] KEY POINTS

Addressing a common challenge in the digital age, we seek to provide an offline solution to the seamless facial recognition and tagging capabilities currently offered by cloud-based social media platforms like Google. These platforms utilize the immense computational power of specialized servers equipped with powerful GPUs and TPUs to achieve this remarkable feat. Our goal is to replicate this functionality offline using our ARM-based processors, enabling real-time facial recognition and tagging capabilities even in offline environments.

[B] RELATED WORK

- Reference[1] :

  By referring to the first reference, we gained valuable insights into the necesity of the preprocessing part alongiside the steps of how to configure CameraX to capture the camera feed and process the frames for face detection. The guide made us understand the comprehensive pipeline and it effectively illustrated the entire process.

- Reference[2]:

  Presented similar information as Reference[1] but had a much more in depth illustration on how to actually implement the pre-facedetection step and also presented us the idea of adding an overlay over the face of each detected person alonside some code example.

- Reference[3]:

  Helped us understand the necessity of the conversion from YUV to NV21 format before converting the raw captured image into Bitmap. We encountered multiple app crashes because we didn’t know the relevance of this step. The CameraX API typically provides image data in YUV format, and certain image processing tasks, such as face recognition, require the data to be in the NV21 format before converting it to Bitmap file.

[C] *STATE-OF-THE-ART*

In the ever-evolving landscape of mobile technology, the integration of cutting-edge frameworks has ushered in a new era of possibilities. Our endeavor in real-time face recognition represents a paradigm shift, synergizing the power of Android, TensorFlow Lite, ML Kit, and CameraX to redefine the boundaries of facial recognition capabilities on mobile devices.

Initially, our contemplation involved the exploration of pre-training some existing models such as YOLOv8 and to use Roboflow for data image classification , finally converting the the format of the model into a TensorflowLite file. However, a critical realization dawned upon us – while it might be straightforward to detect individuals, training a model to discern specific names associated with each person proved to be a challenge, so we decided to stick with the already accurately trained models.

In our pursuit of refining real-time face recognition, we explored two primary methodologies, each presenting unique challenges and opportunities.

1) The first approach involved leveraging web sockets to process the detected and preprocessed image on a remote server with TensorFlow working in Python. While this method exhibited promising results, it was deemed suboptimal due to its dependency on a constant internet connection, presenting limitations for offline functionality

   As we delved into the intricacies of this server-based recognition system, it became evident that relying on external servers introduced potential latency issues and security concerns. Recognizing the need for a more self-contained and versatile solution, we shifted our focus to a second methodology, emphasizing local processing capabilities.

2) Recognizing the limitations of the initial server-based approach, we transitioned our focus to a more optimized and self-contained solution—a shift that not only aligns with real-world use cases but also embraces the versatility demanded by modern mobile applications. This second methodology, which emerged as the frontrunner in our pursuit, revolves around the integration of a pre-trained AI model specialized for mobile and embedded systems.

   In tandem with TensorFlow Lite, our system integrates MobileFaceNets, a specialized pre-trained AI model designed explicitly for face recognition on mobile and embedded systems. The choice of MobileFaceNets aligns seamlessly with our objective of creating an efficient, offline-capable face recognition solution. MobileFaceNets is optimized for low-latency inference, making it an ideal fit for real-time applications. Its architecture prioritizes accuracy while remaining lightweight.

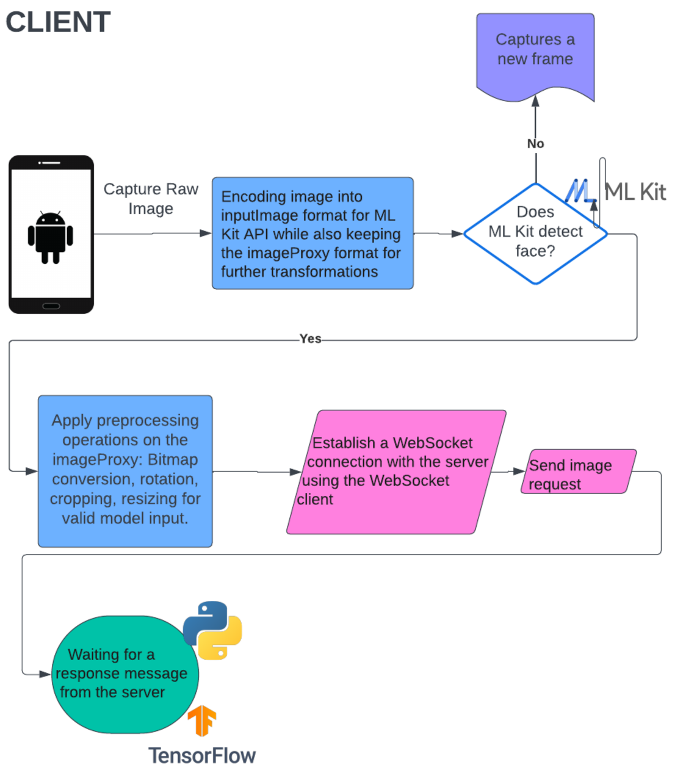
CLIENT
Captures a new frame
No
ML Kit
Capture Raw Image
Encoding image into inputImage format for ML Kit API while also keeping the imageProxy format for further transformations
Does ML Kit detect face?
Yes
Apply preprocessing operations on the imageProxy: Bitmap conversion, rotation, cropping, resizing for valid model input.
Establish a WebSocket connection with the server using the WebSocket client
Send image request
Waiting for a response message from the server
TensorFlow

Fig. 1. Self-made diagram showing the first approach using Websockets,

II. METHOD DESCRIPTION

We've splitted our proposed approach into multiple steps:

1). In the initial phase of our implementation, we embarked on understanding the foundational concepts of pre-face detection using the androidx.camera library in conjunction with CameraX. This phase was crucial for comprehending the intricacies of processing the raw input image captured from the camera preview.

The process commenced with the initialization of the camera provider, a key entity responsible for managing access to the device's camera. Subsequently, we delved into the creation and configuration of essential camera use cases. Among these, the Preview use case facilitated real-time display of the camera feed by ”setting it's surface” to our smartphone display, while the Lens Facing Selector ensured the selection of the desired camera lens.

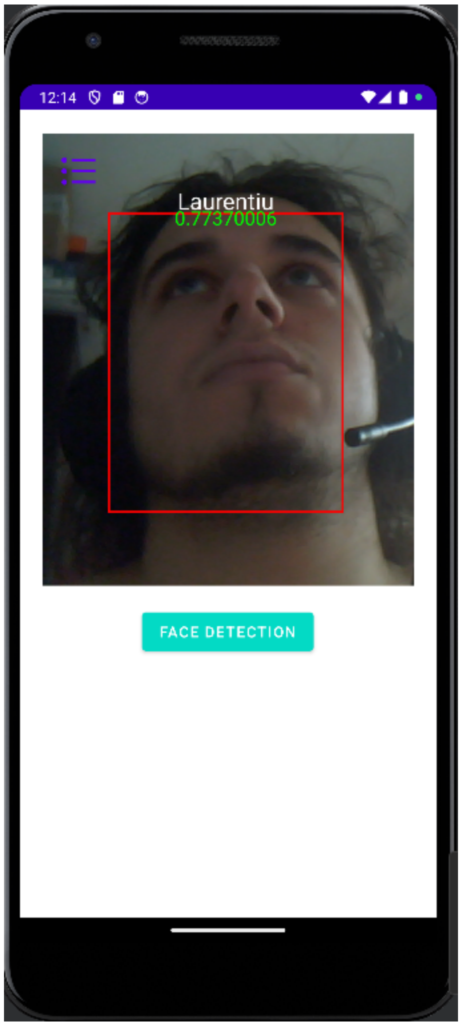

Fig. 2. Screenshot with the app recognizing a face, alongside it's confidence level

To ensure that the image analysis process, which may involve computationally intensive tasks, does not impact the responsiveness of the main (UI) thread, we opted for the use of an executor. Specifically, we chose to employ a single-threaded executor, an abstraction for thread management in Kotlin. This configuration guarantees that tasks submitted to this executor are executed sequentially on a dedicated background thread. This deliberate choice helps maintain a responsive user interface, preventing any potential delays or lags caused by heavy image analysis operations from affecting the main thread responsible for handling UI interactions.

This image analysis process ensures to perform a specific operation on each captured frame. In this case, the operation is defined by the lambda function. The ImageProxy parameter represents a frame from the camera, and the lambda function defines how each frame should be processed. Every frame is going to be processed but with a delay cause by a handler (like a thread sleep) to not overwhelm the app with rapidly instant new frames. After this conversion from Image to inputImage, we can finally start the detection while keeping further the ImageProxy for the second phase.

The camera will only be active and consuming resources when the associated lifecycle (activity) is in the "resumed" state. This helps in optimizing resource usage and avoiding potential memory leaks. This means we needed to bind the configured camera use cases to the lifecycle of the component.

2) In the post-face detection phase, the transformation of the face bitmap is a carefully orchestrated process designed to optimize its presentation for subsequent analysis. The initial rotation, mandated by the default leftward orientation of the captured image, serves to align the face correctly. Subsequently, a query regarding a potential camera switch determines whether an additional rotation, this time potentially a horizontal flip, is needed. This meticulous approach ensures that the face bitmap undergoes the necessary transformations, including rotations and optional flips, ultimately preparing it in a standardized format conducive to accurate and efficient processing by the TensorFlow model.

The next processing step is the cropping of the original image to only the facial zone, the alghoritm is pretty basic, we have the boundaries rectangle, so that makes it easy for use to create a new bitmap image using the width and height of the rectange, we make a source rectangle with the specific coordonates of where the face on the original bitmap is located, and a destination rectangle is for the coordonates where the canvas will draw the image.

The resizing operation is indeed straightforward, utilizing the createScaledBitmap method. This method takes the original bitmap, along with specified target width and height parameters, to generate a resized bitmap. Notably, the use of the false parameter in the method call signifies that no filtering, such as interpolation, is applied during the scaling process.

In the context of our facial recognition model, a key aspect of this resizing step is tailoring the image dimensions to a specific size, often 112x112 pixels. This dimension customization is particularly significant for preparing the image as input for the facial recognition model.

3) For the final phase of facial recognition, it's essential to delve into the training process of the model. During the training phase, the MobileFaceNet model is fine-tuned on a dataset specifically curated for face recognition tasks. The primary objective is to teach the model to generate discriminative embeddings for faces, effectively learning a feature space where faces of the same individual are closely clustered, while faces of different individuals are distinctly separated.

The training process involves the following key steps:

Generate Face Representation or Encoding: Convolutional Neural Networks (CNNs) serve as powerful feature extractors in this context. The model takes face images as input and produces an array of numerical values, known as an encoding. This encoding serves as a compact representation of the facial features.

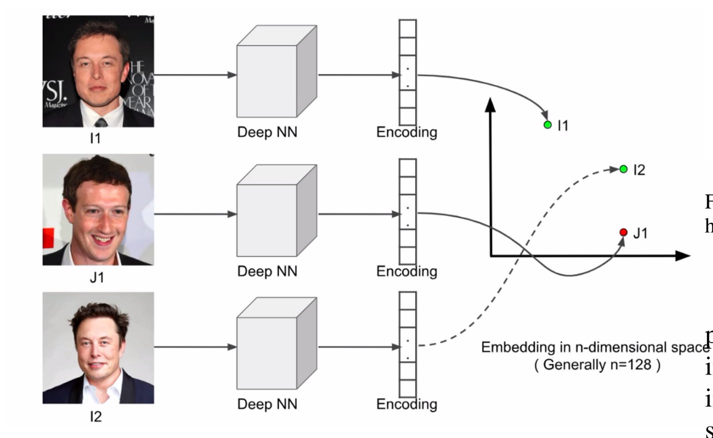

Fig. 3. The illustration above showcases the generated encodings for each image, providing a glimpse into their representation within a 2-dimensional feature space, typically with a dimensionality of n=128. The points visualized on the graph are referred to as Face Embeddings, signifying their placement within this feature space.
https://learnopencv.com/face-recognition-an-introduction-for-beginners/

In the process of learning a Face Embedding Space, the model undergoes training to enhance a specific loss metric, such as Triplet Loss. Triplet Loss plays a pivotal role by instructing the network to minimize the distance between embeddings of the same individual while simultaneously maximizing the distance between embeddings of different individuals. This iterative training of the neural network aims to optimize the overall face embedding space. Triplet Loss operates on triplets of images: an anchor image from a person, a positive image of the same person, and a negative image from a different person. The loss function encourages the network to decrease the distance between the anchor and positive images (similar faces) while increasing the distance between the anchor and negative images (dissimilar faces). This process not only refines the model for accurate face recognition but also inherently leads to the clustering of similar faces in the feature space, facilitating efficient face grouping.

Basically we conclude all this to:

Bring the face embeddings of same person closer.

Push embeddings of different persons farther.

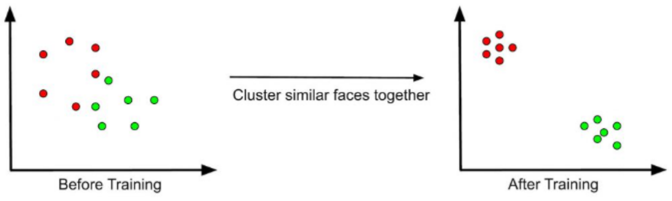

Fig. 4. Clustering process.
https://learnopencv.com/face-recognition-an-introduction-for-beginners/

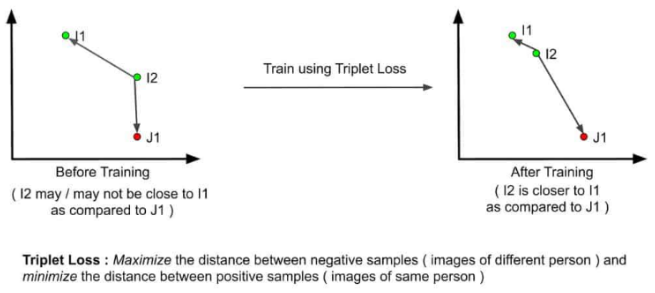
I1
I2
J1
Train using Triplet Loss
I1
I2
J1
Before Training
(I2 may / may not be close to I1
as compared to J1)
After Training
(I2 is closer to I1
as compared to J1)
Triplet Loss : Maximize the distance between negative samples (images of different person) and
minimize the distance between positive samples (images of same person)

Fig. 5. Triplet loss process.
https://learnopencv.com/face-recognition-an-introduction-for-beginners/

During the inference phase, the trained model is utilized to perform facial recognition. When presented with a new face image, the model produces a face embedding for that specific image. The identification process involves predicting the person's identity by assessing the distance between this generated embedding and those stored in the database. Typically, the L2 distance, also known as Euclidean distance, serves as the common measure for comparing embeddings.

$$Similarity(F1, F2) = \|DNN(F1) - DNN(F2)\|_2 = \sqrt{\sum_{i=1}^{D}(E1_i - E2_i)^2}$$

Fig. 6. Similarity between faces can be computed as the euclidean distance between its embeedings.
https://medium.com/@estebanuri/real-time-face-recognition-with-android-tensorflow-lite-14e9c6cc53a5

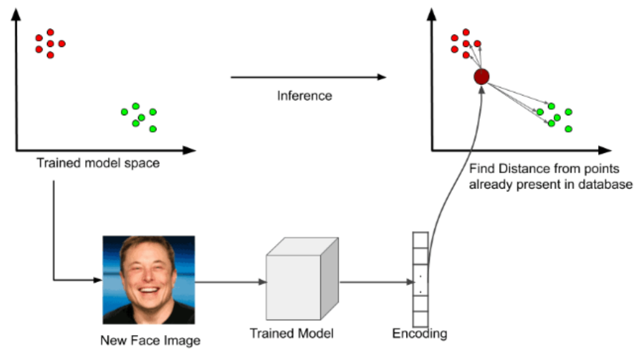

Fig. 7. Inference process: Upon the completion of network training and the generation of embeddings for the training images, the model becomes reliably equipped to predict the identity of a new face.
https://learnopencv.com/face-recognition-an-introduction-for-beginners/

Now if we were to talk strictly about the pre-trained model that we used for the app, MobileFaceNet, Fig.8 can illustrate the process, after we transform the original image into the resized 112x112 cropped image, the CNN starts doing it's job.

The 7x7 feature map is typically found in the **Convolutional layer** of a face recognition network. Convolutional layers are responsible for extracting features from an image,This information is then passed through a **Pooling layer** to reduce the dimensionality of the feature map and make it more manageable for the next layers in the network.After the pooling layer, the feature map may go through a **ReLU activation layer**. ReLU layers introduce non-linearity to the network, which can help the network to learn more complex features.

Earlier, 7x7 feature maps were typically used in the early stages of face recognition networks. Now, the encodings array is more commonly used in later stages. 7x7 feature maps are used to extract basic features from an image, such as the location of the eyes, nose, and mouth. These basic features are then combined into a more complex representation of the face using the encodings array.

For our approach recognition with this model we start by normalizing the image with a mean and standard deviation. This normalization is part of the preprocessing steps and serves several purposes:

Subtracting the mean (128 in this case) centers the pixel values around zero. Zero-centered data is often preferred because it simplifies the training process for neural networks, making it easier for them to converge during training.

Dividing by the standard deviation (also 128 in this case) scales the pixel values, ensuring that they fall within a similar range. This normalization) is a form of scaling that bring the pixel values to a range where they are not overly large or too small.

We run our model with the normalized input and the result is stored in 192-dimensions array with each dimension meaning a feature, after we save a face, it's encodings are going to be saved in a database(for us is just a map with the name and the encodings of the person. When we show a new face for comparision, the alghoritm will iterate through each person stored in the database and find the nearest match if it does exist, if the distance between the existing embeedings and the new embeedings it's too large, it will pass the threshold and on the detected person face it will ouput "Unknown". If the distance is less than the threshold, we calcultate the confidence level being inversely proportional to the distance, the closer the distance, the larger the confidence value is.

III. PRELIMINARY RESULTS

Very promising results concluding the first and the second phase, both pre-facedetection and post-facedection stages work as expected, we successfully made a suitable format for the ML Detector from just capturing the raw image of the smartphone. We also managed to process the image after the detection, which will give the future the model's desirable input.

We could also mention that we tried to integrate a gallery saving mechanism which will store the processed image and the posibility to add a face from the phone's gallery as a registered face. Both features contain a lot of bugs and we think it might be related on the thread workflow.

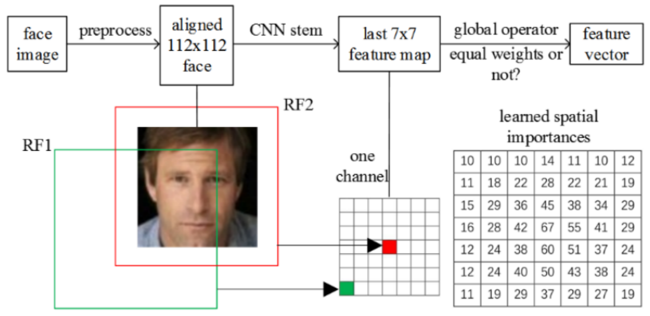

Fig. 8. How MobileFaceNet actually works
https://arxiv.org/ftp/arxiv/papers/1804/1804.07573.pdf

IV. PRELIMINARY CONCLUSIONS

Our app has shown promising results in its early development stages, and we believe it has the potential to be a valuable tool for users. As we continue to refine the app, we are exploring the possibility of integrating face recognition technology to enhance its capabilities and provide a more personalized user experience.

III. FINAL RESULTS

As for the final results, the model does very well and succesfully recognizes faces in various bad conditions such as poor lighting. We do have a problem with the fact that the model from certain angles can't really recognize very well and it ouputs "Unknown" sometimes. Another problem will be the fact that on some devices the face tracking has bugs and it's mostly because of some sensors of orientation of the smartphones, but on every emulator device we tested it works well. For a more intriguing showcase we graphed an overlay around the recognized face with name and the confidence level.

IV. FINAL CONCLUSIONS

In its current iteration, our facial recognition app has successfully achieved its fundamental objectives, showcasing impressive capabilities in face detection, recognition, and proximity-based confidence level calculations.While the app's core functionalities are solid, there is ample room for future refinement and enhancement.

REFERENCES

[1] Google AI. (2023, March 22). Android Jetpack CameraX: Face Detection and Recognition.
[2] Hechio, Steve. "Android ML Face Detection with CameraX." Medium, 15 Mar. 2023,
[3] Gonzalez and Woods, "Digital Image Processing".
[4] Vasco Correia Veloso , "Hands-On Artificial Intelligence for Android: Understand Machine Learning and Unleash the Power of TensorFlow in Android Applications with Google ML Kit".
[5] Dawn Griffiths and David Griffiths, "Head First Android Development".

[6] John Omokore, ”Android Camera2 API”.
[7] Bharath Ramsundar and Reza Bosagh Zadeh, ”TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning”.
[8] Raja Kannan, ”Android Machine Learning with TensorFlow Lite”.
[9] Anum Haroon, Sadaf Iqbal, ”Face detection,classification and transformation android application”, July 28, 2013.
[10] Li Ma, Lei Gu and Jin Wang, ”Research and Development of Mobile Application for Android Platform”, Vol. 9, No.4, pp.187-198, (2014).
[11] Ex-Sight, ”Face Recognition - Technology Overview”, 2009.