Internship Report : AjnaLens

Name: Krish Rana.
Designation: Deep Learning Intern.
Duration of Internship : 2 months

Contents

1. Literature Survey and Initial Work

At the start of the internship, I started working on the face recognition feature of the AjnaTx AR Glasses. The core of the idea was to develop a model that could accurately recognise faces covered with masks which was deployable on Android OS. We took some inspiration from the existing work by other developers and companies mainly from Hitachi Vantara's iRedact system. the iRedact product overlapped significantly with our requirements and provided an initial starting point to decide the tools and libraries to be used and begin the development process.
link - https://youtu.be/_wuyh37gO-4

There were many face recognition solutions available which worked well with non-masked faces but the accuracy decreased significantly when they were used on a person wearing a mask. There were no off-the-shelf solutions available for face recognition models that worked with masked faces, so I had to build our model from scratch. We started with two approaches 1. Training the model with partial faces (from forehead till nose) and 2. Training the model with full masked faces. Both the approaches are explained in detail in the Python Module.

I hereby, declare that all information put in the report are genuine and each of the mentioned functionalities/remarks mentioned can be observed in source code for each task/subtask listed in the content section. I acknowledge that the report may further be evaluated, for providing future references for my career-pursuit.

2. Face Recognition - Python Module

Description:

As mentioned above I started with two approaches.
In the first approach, I tried building a recognition model with a training dataset that contains only partial faces of the people that is only the part that is visible while wearing a mask (from forehead to nose). For this, I pre-processed the training images to extract the visible part (from forehead to nose) using Dlib 68 point face shape predictor (shape_predictor_68_face_landmarks.dat). And stored it in the training directory.

In the second approach, I trained the recognition model with complete faces of people in masks. As we did not have a large open-source dataset of people with masks, the masks were digitally superimposed on the existing face datasets such as VGG Face2 dataset. The dataset that I used was already preprocessed and ready to use. The mask superimposition was inspired by a blog post by Brouton Labs.

To train the recognition model, I applied one-shot learning procedure based on the triplet loss function. The process was inspired by the FaceNet paper published by Google. In this process, we take three images at a time 1. Anchor image (our input image), 2. Positive image(image of the same person) 3. Negative image (image of a different person). What the triplet loss function does is it minimises the distance between the Anchor and the Positive image and increases the distance between the Anchor and Negative image based on Euclidean norm. I used online triplet mining method where the triplets ae formed on-the-fly during training.

Model architecture: I used MobileNetv2 as the base model as it was optimised for edge devices and resulted in a faster inference time as compared to the Inception model used in the FaceNet paper. I used the MobileFaceNet research paper as a reference for the top layers of the model and other hyper-parameter optimisations. The model outputs a 128D vector embeddings which are compared to one another to assign identity.

Face detection: I used the TinyFacesDetector module to detect the faces in the video frame. The module was already developed by the team. The detection model returned bounding box co-ordinates for each face detected in the frame. The faces were then cropped based on these bounding boxes and pre-processed for the face recognition model.

Face Tracking: For face tracking, I used Dlib correlation tracker coupled with centroid tracking It was based on Adrian PyImagesearch's approach.

The identity was assigned to the person using Nearest neighbour algorithm where the distance between the embeddings is measured based on L2 norm(closer to '0.0' is better). If the distance was greater than a pre-determined threshold then it was identified as an unknown person.

I hereby, declare that all information put in the report are genuine and each of the mentioned functionalities/remarks mentioned can be observed in source code for each task/subtask listed in the content section. I acknowledge that the report may further be evaluated, for providing future references for my career-pursuit.

The flow of the recognition pipeline:

1. The known faces stored in the directory are loaded in the memory and their face embeddings are generated using the deep learning model and stored in a dictionary with {Name: embeddings} as key-value pairs.
2. The input video is loaded and the face detector (TinyFacesDetector) detects the faces in the frame and passes the face rects to the Dlib correlation tracker to track the faces.
3. For the first 30 frames of the video, the face embeddings are calculated and measured against the known faces to assign identity. The identity of a person is locked based on a voting system where the identity which is assigned the most number of times in the initial 30 frames is chosen.
4. After every 60 frames, the face detector again checks if any new faces have entered the frame and assign tracking id and identity to them.

Of the two approaches mentioned above, **the model trained on full masked faces performed better** than the model trained on partial faces.

Statistics for the two models on a sample video of Justin Trudeau:
https://drive.google.com/file/d/1aB3c9NLgnpgvPyYgq_PYIaqUXO8SK29u/view?usp=sharing

Source code of each approach:

Partial faces generator:
https://drive.google.com/file/d/1aWKucBZf0cDfqO-hRpcXWB0cfYiIzv11/view?usp=sharing

Source code:
https://drive.google.com/drive/folders/1U2n5txvB5_ECgAqzEn-O6sqB8Du1zOKW?usp=sharing

Recognition model:
https://drive.google.com/file/d/1J75mlEpfgm7iYo8WPUfjcX7aggtkFHHu/view?usp=sharing

References (Blogs/Github Repositories):

1. Brouton Labs Blog: https://broutonlab.com/blog/how-facial-recognition-works-with-face-masks
2. FaceNet Paper: https://arxiv.org/abs/1503.03832
3. Tensorflow's implementation of Triplet Loss: TripletSemiHardLoss
4. MobileFaceNets Paper: https://arxiv.org/pdf/1804.07573.pdf
5. Adrian's Dlib correlation tracker blog: Dlib_correlation_tracker, Centroid_tracking
6. Face recognition pipeline: https://github.com/Alloooshe/facelib_modular_face_recognition_pipline
7. CASIA masked dataset- https://drive.google.com/open?id=1q0ibaoFVEmXrjlk3-Oyx2oYR8HpVy6jc

Scope of Improvement:

1. Scope of improvement lies in the recognition model's accuracy. It can be improved by training it with more data as the model has about 2.4M parameters. Link to a preprocessed and masked augmented CASIA dataset is attached in the References.
2. A better face tracking algorithm can be used for eg. Deep Sort.
3. Advance clustering algorithm such as DBSCAN would give a better and a faster result.

I hereby, declare that all information put in the report are genuine and each of the mentioned functionalities/remarks mentioned can be observed in source code for each task/subtask listed in the content section. I acknowledge that the report may further be evaluated, for providing future references for my career-pursuit.

3. Face Recognition - Android Module

Description:

To deploy the face recognition model on Android, there were some off-the-shelf solutions available but the custom tuning was required for it to work with the AR user interface. One such solution was available on medium link- https://medium.com/@estebanuri/real-time-face-recognition-with-android-tensorflow-lite-14e9c6cc53a5 which itself was inspired from the TensorFlow lite's demo apps.

For a faster inference and keeping the memory constraints in mind, the Face recognition model was converted into a '.tflite' model which was 9.7mb in size as compared to the 30mb '.h5' model. For face detection and face tracking, I used Google's ML Kit which was natively supported on Android.

Because the user wearing the AR glasses cannot physically press the 'Add person' button to register a person, automated cloud storage and retrieval system was developed. Whenever a new person was detected in the frame, an ID was assigned to that person and the ID, face embeddings and the cropped image of the face were automatically stored in the database. It was a smart storage system in the sense that it did not write data of the same person more than once and the ID assigned to that person was retained in future. This was achieved by keep track of the number of people that were scanned in that session and applying face recognition to faces detected in that session to determine if the person was seen before. The app is connected to Firebase's Realtime Database and Storage container. When the app is loaded, face Embeddings, ID and the name of the person is retrieved from the realtime database and is stored in a Hashmap in the memory.

The SimilarityClassifier Interface is responsible for assigning the identity to the person. When a Face is detected in the frame, embeddings are generated by the recognition model and the SimilarityClassifier finds the nearest neighbour by calculating the euclidean distances and performing a linear search to find the minimum distance..

Source code of each approach:

Link: https://drive.google.com/drive/folders/1N2yj0vy9AbA27AUISlRMRqNQ5T7BfBiI?usp=sharing

APK link: https://drive.google.com/file/d/15DofaqYTx8Bsgy5YcVTBcK8IoQSNB9qM/view?usp=sharing

Tflite model: https://drive.google.com/file/d/1_TCFvsr4J5jv4PH9GSeF73s-GCw1D6pZ/view?usp=sharing

References (Blogs/Github Repositories):

Medium blog:
https://medium.com/@estebanuri/real-time-face-recognition-with-android-tensorflow-lite-14e9c6cc53a5

Github link:
https://github.com/estebanuri/face_recognition

I hereby, declare that all information put in the report are genuine and each of the mentioned functionalities/remarks mentioned can be observed in source code for each task/subtask listed in the content section. I acknowledge that the report may further be evaluated, for providing future references for my career-pursuit.

Google ML Kit:
https://developers.google.com/ml-kit/vision/face-detection/android

Scope of Improvement:

1. Opening the app from the saved state for eg. From the multitasking view or a saved instance causes the app to crash.

2. The scope of improvement for cloud connectivity and large-scale implementation of the recognition system considering the memory constrains of the edge device is explained in greater detail in the next section.

I hereby, declare that all information put in the report are genuine and each of the mentioned functionalities/remarks mentioned can be observed in source code for each task/subtask listed in the content section. I acknowledge that the report may further be evaluated, for providing future references for my career-pursuit.

4. Future Work and Scope of Improvement

The problem that we would eventually face for large scale face recognition on edge device will be the memory and the compute constraints of that device. The state-of-art method to assign the identity to a person once the face embeddings are generated is to perform a nearest neighbour search on the entire database to find the most similar face. The KNN algorithm requires entire dataset to be in the memory to work efficiently and effectively. This quickly becomes a problem once the database increase to a certain limit where the size of our dataset becomes greater than the internal memory of the edge device.

One way to tackle this issue is instead of retrieving the entire database to the edge device we send the face embeddings of the person detected in the frame to the cloud and perform Nearest Neighbour in the cloud. But this can lead to latency issues between the cloud server and the mobile device which can hamper the real-time performance. The **Cloud-Vision: Real-time Face Recognition Using a Mobile-Cloudlet-Cloud Acceleration Architecture** paper deals with this issue.

Sending raw data to the cloud is very network intensive. So instead of sending the cropped face images of the person, the face embeddings can be generated on the mobile device itself and they can be sent to the cloud. The paper introduces an mobile-cloudlet-cloud architecture. The cloudlet is a high compute device capable of massively parallel processing which acts as a secondary compute device once the compute load on the edge device increases. The device can then be used to capture the video and send it to the cloudlet for further processing (face detection and recognition).

The paper suggests implementing a mobile-cloudlet-cloud architecture to minimise the latency and improve the performance. It also proposes algorithms to dynamically partition the data amongst the cloudlets and accessing the servers such that the overall latency is reduced. The paper proposes a greedy approach where we first order the servers (and the cloudlet) by their (known) response times and give the first task to the server (cloudlet) that can complete this task in the minimum amount of time. We then give the second task to the server (cloudlet) that can complete this task in the minimum amount of time (note that this may be the same server as given the first task if the time for the first server to complete both tasks one and two are less than the time for the second server to complete just task two).

There is also some research in modelling edge-servers as caches for compute-intensive recognition applications where some of the inferences are cached in the cloudlets. The researchers have built Cachier, a system that applies adaptive load-balancing between the cloud and the cloudlet. Their initial results show a 3x speedup in responsiveness for face recognition tasks on the edge, as compared to non-caching systems.

Link- https://www.cs.rit.edu/~jmk/papers/cloud-vision.pdf
Link- https://people.eecs.berkeley.edu/~kubitron/courses/cs262a-F18/projects/reports/project9_report_ver2.pdf

I hereby, declare that all information put in the report are genuine and each of the mentioned functionalities/remarks mentioned can be observed in source code for each task/subtask listed in the content section. I acknowledge that the report may further be evaluated, for providing future references for my career-pursuit.