HoloGuard

Ofir Florentz

Yaniv Holder

contents

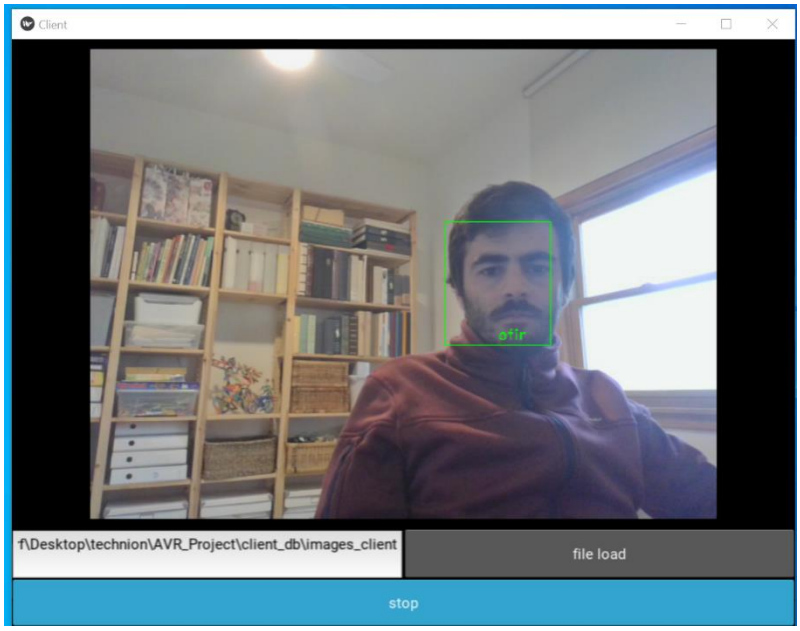
Introduction

In our project we wanted to take advantage of the latest face recognition and face detection developments to improve real world cam-based application by adding augmented information on the picture.

First, we explored several ideas of those application. Finally, we decided to focus on security cameras and the guard in the entrance of busy places.

For example –
At the Technion entrance there is a guard that checks the people who come in. Though most of these people are students who come in on regular bases, and the Technion already have their face in his data base.
There are a lot more use cases - security cameras, automatic gates and so on.

We built an app that shows for each person in the video his name if he is recognize by the system.



Technologies

The system built from two stages:

First, we take a single image and use "Face detection" to create boxes on each face in the image, next we crop the box resize it and send each face separately to a "Face recognition", that check similarity to a user defined data base of faces.

Server and Client

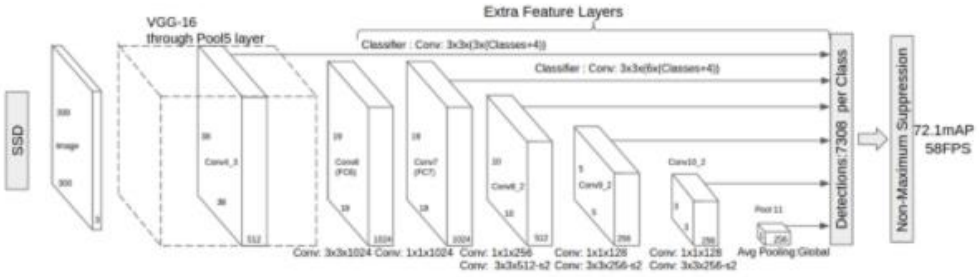
We use python for our backend and front end. As we wanted to make and android application, we use Kivy model. Which is highly recommend.

We combined it with twisted to get Web responses.

More information can be found in the code, and here https://kivy.org/

Face detection

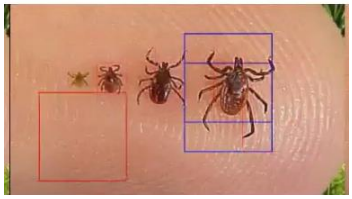
For the face detection we used 300x300 SSD (single shot detection) deep neural net that used for object detection and was fine-tuned on faces.



The architect first proposed in 2016 in the following paper. https://arxiv.org/pdf/1512.02325.pdf.

In the SSD approach we use a fix anchor (boxes) in different sizes and we slide them throw the picture, we use a net to get confidence score for each position of the box. And then we remove overlaps boxes with the confidence score.

In blue high confidence boxes, in red low confidence.



Time – running the ssd on a single image takes ~0.02 secs

Face recognition

We explored several face recognitions models, and finally chose faceNet512 which was first proposed by google in 2015 https://arxiv.org/pdf/1503.03832.pdf.

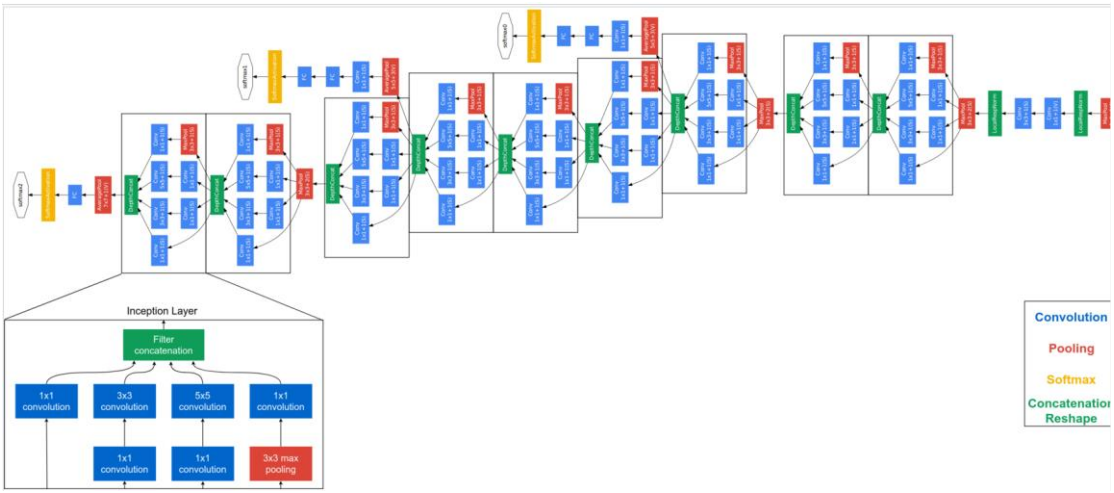
There were alternatives that their performance was better such as VGG-Face and ArcFace, but the computation time of these nets were too long.

Nets we checked and the number of parameters they have.

model	***#params***	***Time (on local cpu)***	***Performance (on lfw dataset)***
ArcFace	*85,500,000*	*~0.6 sec*	*99.83%*
DeepFace	*137,774,071*	*~0.7 sec*	*98.37%*
FaceNet	*22,808,144*	*~0.3 sec*	*99.63%*
openFace	*3,743,280*	*~0.25 sec*	*92.92%*

FaceNet512 is a CNN , which extract embedding of size 512 of a single face.

The network is trained s.t the L2 distance between the embedding correspond to face similarity, it uses stochastic gradient descent and adagrad optimizer.

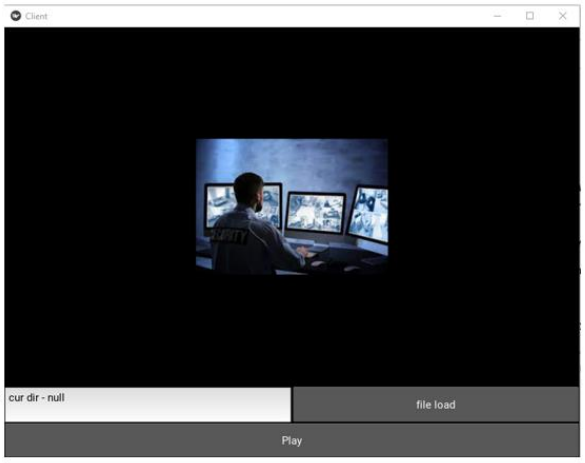


In our project – we get a dataset from the user, we run facenet one time each of the picture in the data set and save the embedding results.

Then during the run of the camera each face from the video is compared to the known embedding images from the data set with cosine distance and a threshold to decide if this person is known.

App –

Open screen –



Currents folder of images that were loaded

Button – choose folder of images

Button Play video

Play screen –

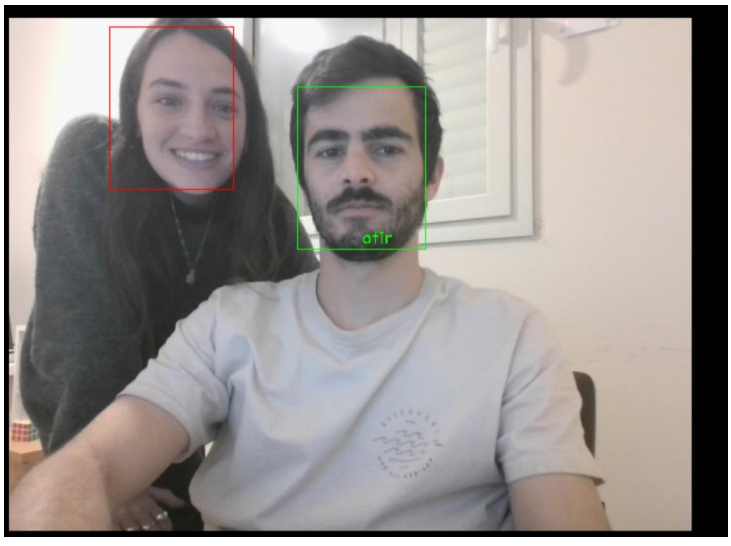
While playing, each face in the image will be bounded by a box.

When people from the data set appear the box color will be green, while for people that are not in the data set the color will be red.

The user can run the app without any people, in that case all boxes will be in red.

The user can also pause the video with "stop" button and reload the images.

In the picture "ofir" is part of the database while the other figure is not



Results

Computation time –

Usually, videos are sampled 32 times a second, we don't have to run our code every frame as the movement of people is usually slower, and we can settle for running it only 4-6 times a second and still get a clean flow of the video.

This was still a limitation (at least for our computation power) as from time reason we did not use a better but bigger recognition model such as deepface and VGG.

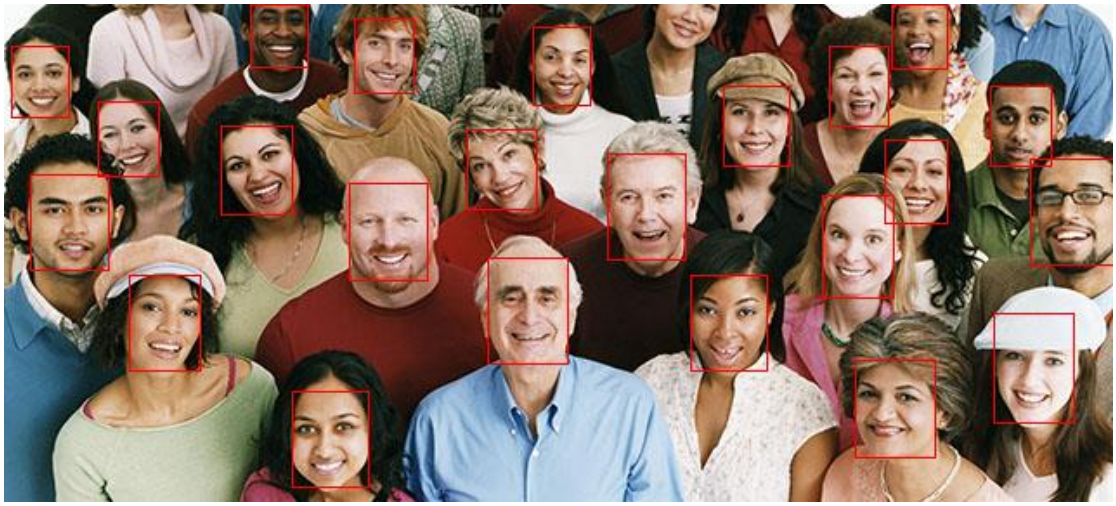
On a CPU the server takes 0.25- 0.3 sec to handle one frame, while detection time is very quick (only 0.03 sec) the recognition take most of the time.

while on GPU it is 2.5 times quicker around 0.1 sec for a frame.

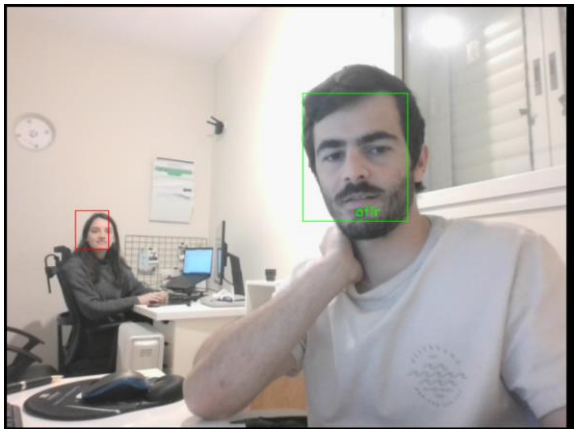
Multiple figures in frame – one can ask if the calculation time depends on the number of people.
For the detection time it is not, while in the recognition every face is cropped into a box and **handled separately** so as long as there are enough cores the number of faces in the pictures will not slow us down.

Detection -

Our detection model works almost perfect when people look straight to the camera. And can detect multiple people even at a crowd.



From our experiment it ranges of detection from a regular laptop computer is around ~9 meters. When detection of faces that are further starts to become unstable.



Recognition –

We evaluate several different scenarios –

- Old images - the recognition is pretty robust and can recognized people from images that are more the 5 years old.

  e.g.: the left pictures took four and half years ago.

- Range – the recognition range is around ~5m and smaller than the detection range.
- False positive – while when a person is in the database is recognition is accurate, most of our mistakes happen from people **outside the database** that are wrongly classified.
  For example, we trying to check our brothers (which are not in the data set) they were misclassified. While human eye could tell the different.