CS 419 PROJECT REPORT

Face Recognition Using TensorFlow

PROFESSOR: ABIR DE

13 – 05 – 2021

Aayush Shah 190100003
Ashutosh Sharma 190100027
Hitvarth Diwanji 190100057
Prakarsh Kumar Yadav 190020085

Introduction

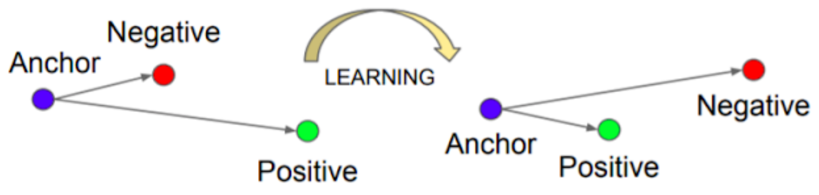
With the advent of ubiquity of mobile devices like smartphones and tablets, it has become increasingly important for the creation of technologies to facilitate quick and reliable security systems. Identification and classification of people based on their face is an efficient way to achieve the same, as seen in its widespread use in phone unlocking systems and airport monitoring. In the backdrop of this new and exciting technology, we have tried to implement our version of a facial recognition system taking inspiration from a 2015 paper published by Google researchers in which they described a system called "FaceNet".
This report is written as an overview of our work. We first briefly state and describe the literature we have referred to and mention the overall scheme of our program. The training and optimisation of the models are elaborated thereafter, ending with the results achieved and open issues which can be ironed out to refine the models.

Literature Survey

We have mostly referred to the FaceNet paper, in which they have described a novel method for carrying out tasks like facial recognition, verification and detection. Instead of using bottleneck layers, they have used a deep convolutional neural network to map the face containing images onto a 128 dimensional vector space, also known as embeddings of the images.

Embeddings are low-dimensional, learned continuous vector representations of discrete variables and they can be used in neural networks as they can reduce the dimensionality of categorical variables. CNN is a deep learning algorithm which performs convolution operation on the input using appropriate kernel size. In order to train this model, they have used "Triplet loss" which is motivated in the context of nearest neighbour classification.

This loss involves three terms, the embeddings of an "anchor", which belongs to a specific person, a "positive" which is another embedding of the same person and a "negative", which is an embedding belonging to a different person. They wanted to ensure that an image xi,a (anchor) of a specific person is closer to all other images xi,p (positive) of the same person than it is to any image xi,n (negative) of any other person. This is visualized in the following figure.



The loss that is being minimised is :

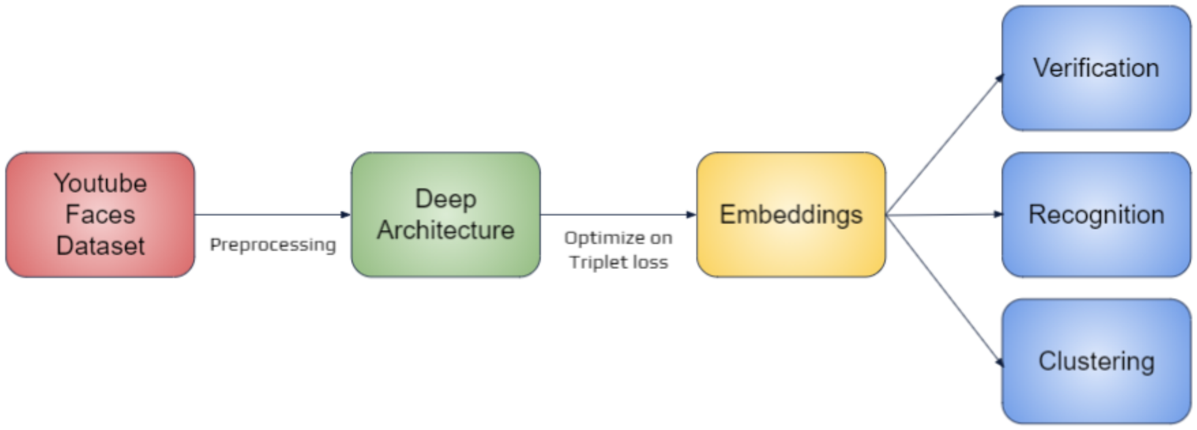
$$\sum_{i}^{N}\left[\|f(x_i^a)-f(x_i^p)\|_2^2-\|f(x_i^a)-f(x_i^n)\|_2^2+\alpha\right]_+$$

Hard triplets are those triplets which have the maximum value of the loss in the batch. The triplets chosen for their method are Semi-Hard triplets in which instead of picking the hardest positive, the loss is calculated over all the anchor-positive pairs in a mini-batch while selecting the hard negatives. They used this method because it was more stable and converged slightly faster at the beginning of training.

The Facenet system achieved a remarkable accuracy of 99.63% on the dataset “Labelled Faces in the Wild” and 95.12% on YouTube Faces DB.

Overview

We have used the dataset “Youtube Faces DB”, which contains images of faces picked up from videos uploaded on Youtube. We implement a CNN based on the one mentioned in the paper to generate embeddings, and another neural network to conduct face recognition using those embeddings.

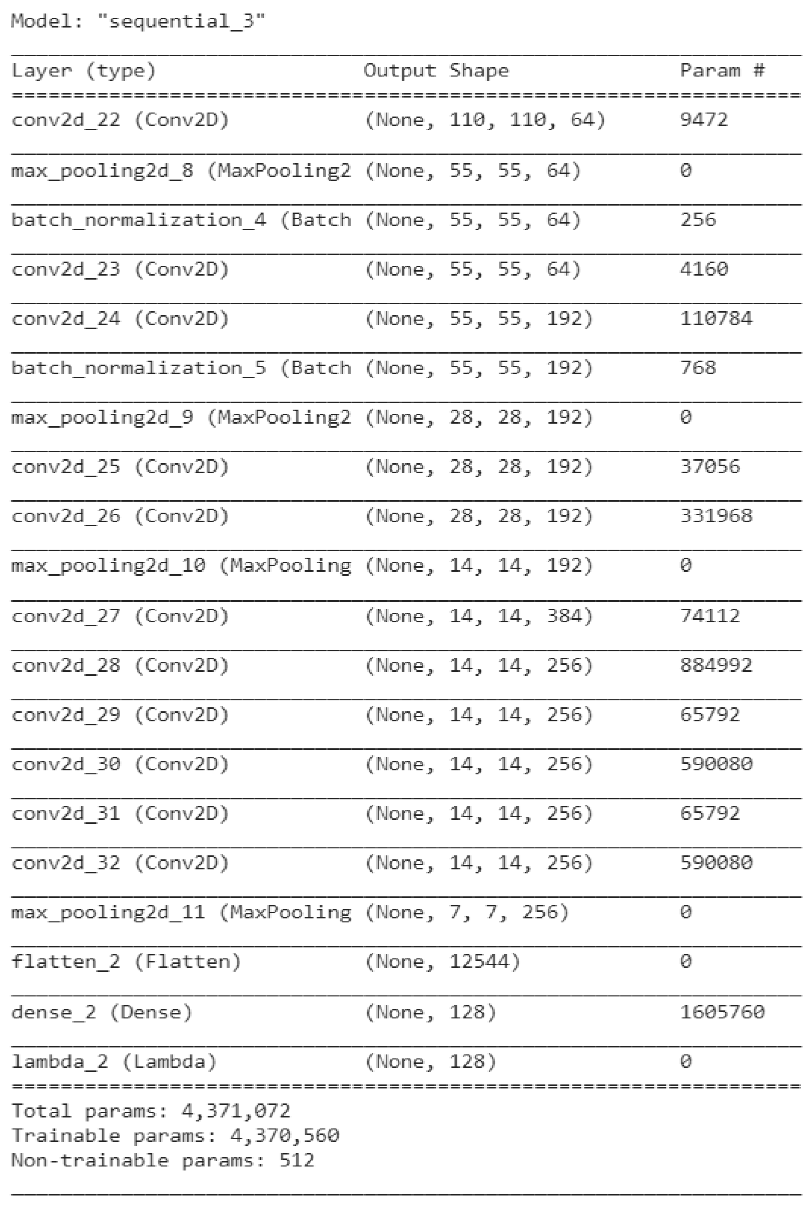


Preprocessing

Preprocessing involves cleaning of data and bringing it in a format which can be fed into the neural network.
The youtube faces dataset had images of varying sizes and our model required the image to be of the shape (220,220,3). Hence, while loading the dataset, we resized all the images using OpenCV. We also carried out splitting the dataset here itself. The chosen ratio of Train, Validation and Test is 70:20:10. The data is stored as a numpy array whose each element is an array of mat image and the name of the person. The train, test and val are shuffled to avoid any batch from having images of the same person. The string labels are then converted to integer labels and the mapping from name to person is saved for further use (in recognition). Finally, all this data is stored as an .npz file so that we do not have to load the dataset again.

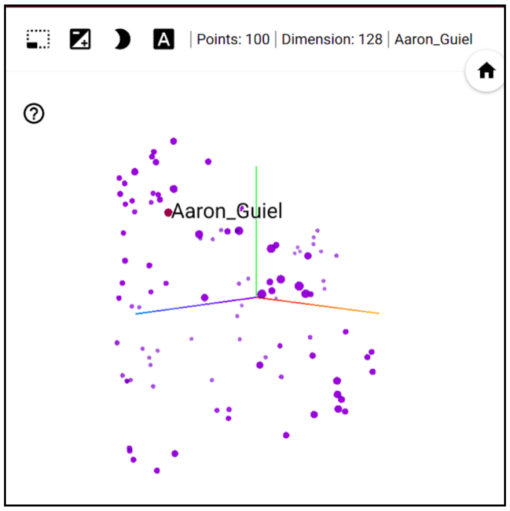
Deep Architecture

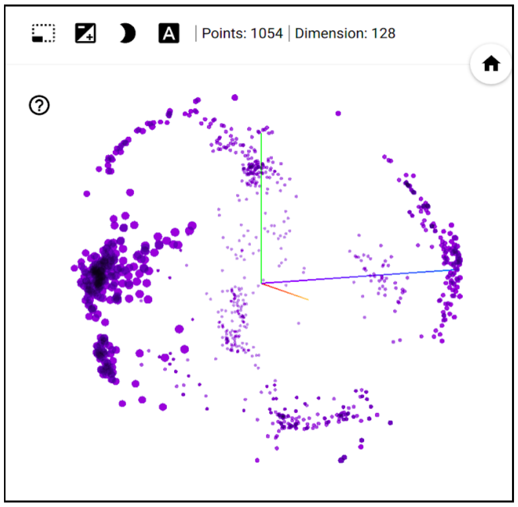
The model used generates embeddings of length 128 for input images of size 220 x 220 x 3. It is based on the Zeiler & Fergus model mentioned in the paper and uses sequential convolutional and pool layers with a fully connected layer and L2 Normalisation in the end.



```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_22 (Conv2D)           (None, 110, 110, 64)      9472      
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 55, 55, 64)        0         
_________________________________________________________________
batch_normalization_4 (Batch (None, 55, 55, 64)        256       
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 55, 55, 64)        4160      
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 55, 55, 192)       110784    
_________________________________________________________________
batch_normalization_5 (Batch (None, 55, 55, 192)       768       
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 28, 28, 192)       0         
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 28, 28, 192)       37056     
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 28, 28, 192)       331968    
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 14, 14, 192)       0         
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 14, 14, 384)       74112     
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 14, 14, 256)       884992    
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 14, 14, 256)       65792     
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 14, 14, 256)       590080    
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 14, 14, 256)       65792     
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 14, 14, 256)       590080    
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 7, 7, 256)         0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 12544)             0         
_________________________________________________________________
dense_2 (Dense)              (None, 128)               1605760   
_________________________________________________________________
lambda_2 (Lambda)            (None, 128)               0         
=================================================================
Total params: 4,371,072
Trainable params: 4,370,560
Non-trainable params: 512
_________________________________________________________________
```

This model was trained using the triplet semi-hard loss function available in TensorFlow which calculates the pairwise loss for all the images in the batch hence the batch should not be very small. The batch size used was 10 and training was done for 5 epochs. The loss reduced considerably and we got a loss of around **0.44** on the validation set. A 3D representation of the embeddings (done using Tensorflow embedding projector) before training and after training is shown. The first figure shows that the test set vectors are diffused but the second figure shows the vectors of all data sets are getting “clustered”, each cluster depicting one person.





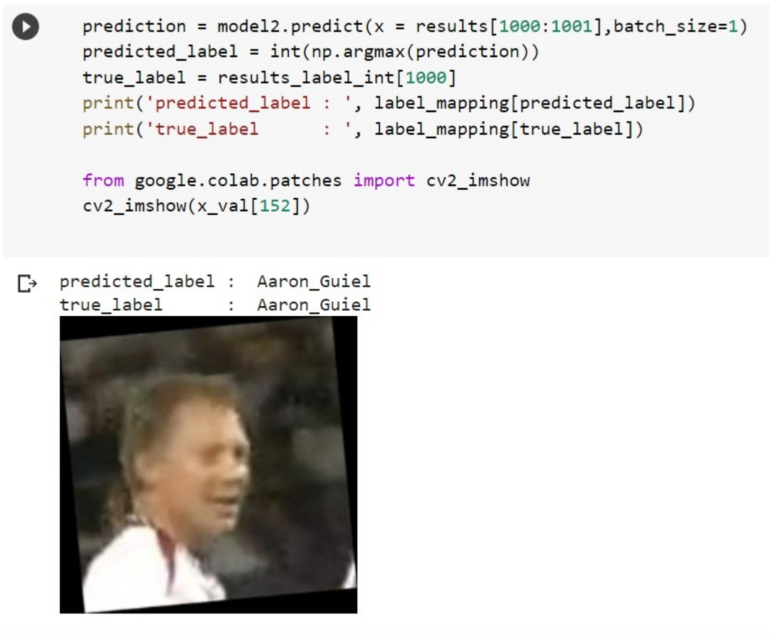
Face Recognition

The embeddings generated by training the previous model will now be used as an input for another neural network which will provide the probabilities that the particular embedding belongs to a particular person. The summary of this model is given.

```
  model2.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
layer2 (Dense)               (None, 64)                8256
_________________________________________________________________
layer3 (Dense)               (None, 5)                 325
=================================================================
Total params: 8,581
Trainable params: 8,581
Non-trainable params: 0
_________________________________________________________________
```

Training was done using the "Sparse Categorical Loss" for 5 epochs. An accuracy of **90%** was achieved on the validation set which proves our method was a success. A test example being correctly classified is shown.



```
prediction = model2.predict(x = results[1000:1001],batch_size=1)
predicted_label = int(np.argmax(prediction))
true_label = results_label_int[1000]
print('predicted_label : ', label_mapping[predicted_label])
print('true_label      : ', label_mapping[true_label])

from google.colab.patches import cv2_imshow
cv2_imshow(x_val[152])
```

```
predicted_label :  Aaron_Guiel
true_label      :  Aaron_Guiel
```

Open Issues

- The accuracy can be improved by using advanced training techniques like learning rate scheduler, early stopping etc. Using more examples in the training set will also lead to better embeddings.
- Online triplet mining techniques can be utilized to generate embeddings on the fly
- The embeddings will also be further used to implement the task of face detection and verification.
- Training can be customized by defining our own loss functions and optimizers