Face Recognition using TensorFlow

HITVARTH DIWANJI 190100057

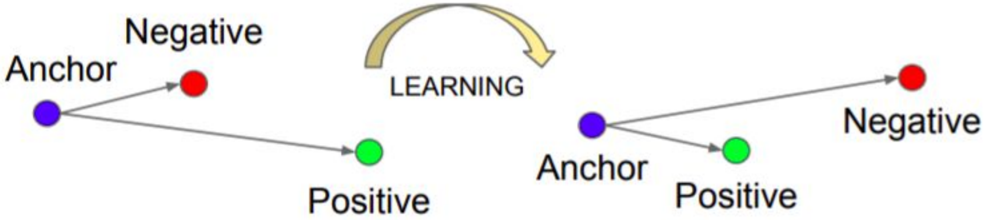
Problem Statement

- There is a need for efficient facial recognition, verification and detection systems due to their varied uses. It also gives a competitive edge to the corporations employing a better version of the system, creating a huge demand for the same.
- **Recognition**: Matching a face from a database of faces
- **Verification:** Verifying the identity of a person by their face
- **Detection:** Detecting presence of faces in images
- **Our task:** To overcome the problem of lack of generalization of previous methods which use bottleneck layers and to carry out facial recognition even when the number of classes is unknown.

Overview of the Research Paper

- FaceNet: A system which directly learns a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity.
- Embeddings: embeddings are low-dimensional, learned continuous vector representations of discrete variables.
- This method uses CNN which is trained to directly optimize the embeddings rather than an intermediate bottle-neck layer as done in other approaches.
- FaceNet is highly accurate, achieving a new record accuracy of 99.63% on the Labeled Faces in the Wild Dataset and 95.12% on Youtube Faces.

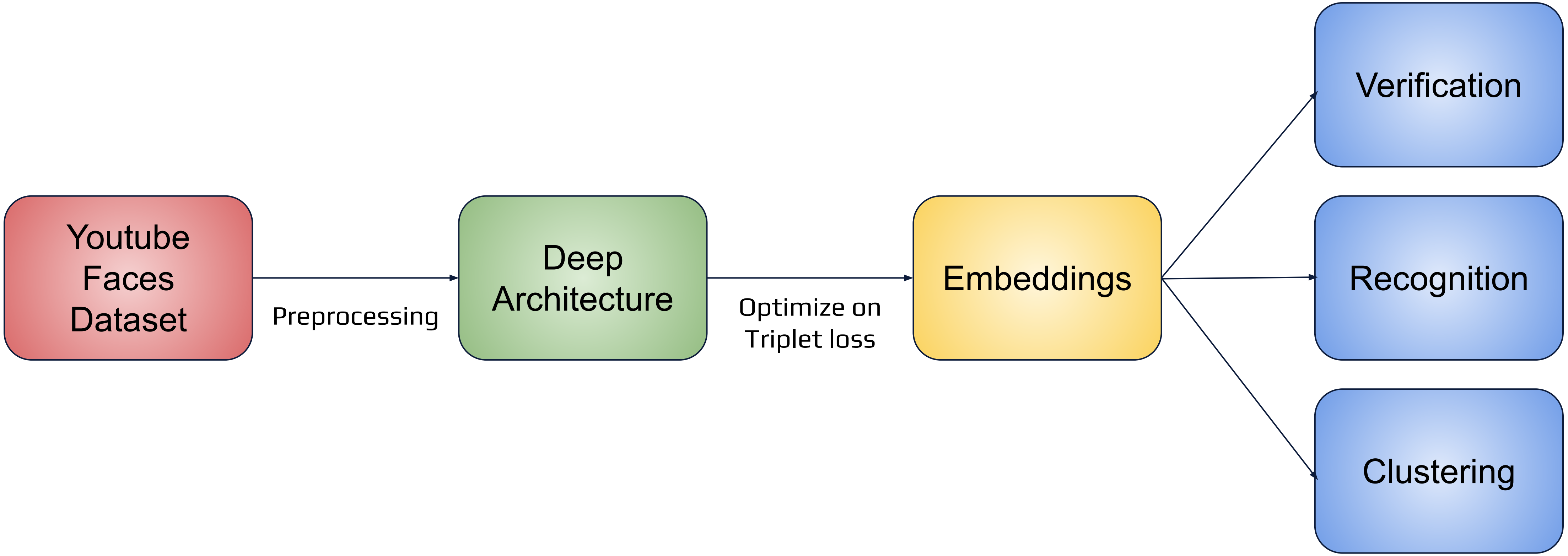
Triplet Loss



- The loss used to optimize the training algorithm is triplet loss, which essentially maximizes the distance between the anchor(image of a person) and the negative points (images of other people) and minimizes the distance between the positive points (image of the same person) and the anchor.
- The triplets chosen for the method are Semi-Hard triplets because it makes the model stable and converge faster
- Mathematical Formulation -

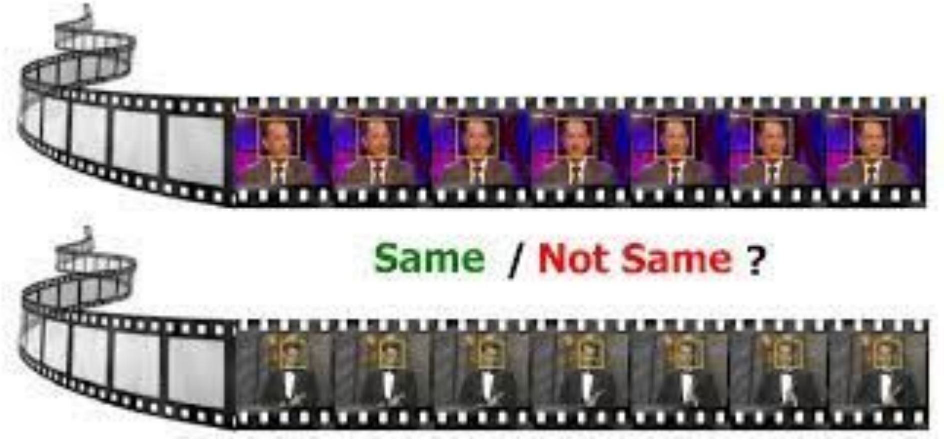
$$\sum_{i}^{N} \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+$$

The complete picture



Preprocessing

- Youtube Faces Dataset
- Resizing into 220x220x3
- Split the data : Train 70%, Val 20%, Test 10%
- Shuffled train, val, test separately
- Gave integer labels to the people and saved the mapping
- Saved the data as an npz file for further use



Deep Architecture

- To get embeddings of each image
- Embeddings from the output of the model
- Similar to Zeiler & Fergus model
- Input : Image of shape (220,220,3)
  Output : Embeddings of shape (128,)
- Sequential Conv 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

Training

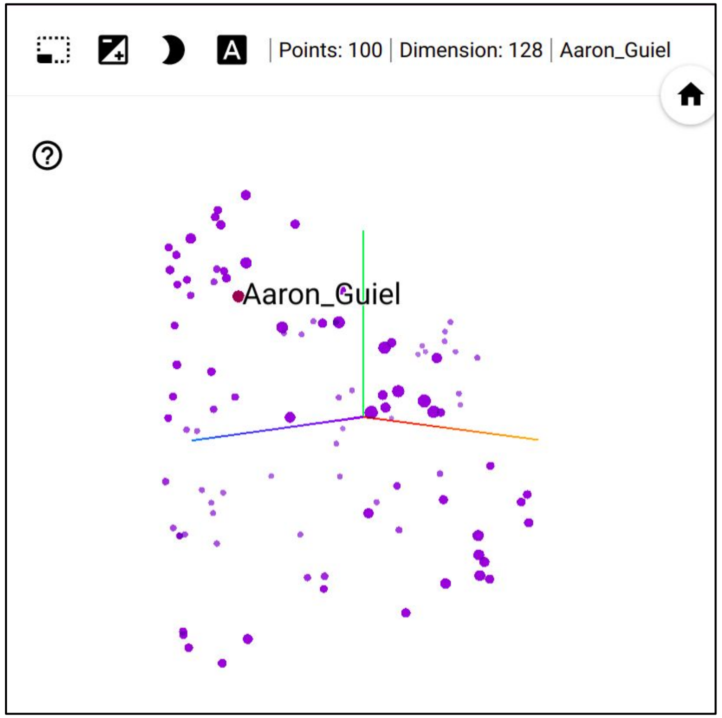
- Batch_size = 10
  Epochs = 5
- Optimizer = Adam
  Learning rate = 0.001
- Loss -> TripletSemiHardLoss

The triplet semi hard loss calculates the pairwise loss for all the images in the batch.

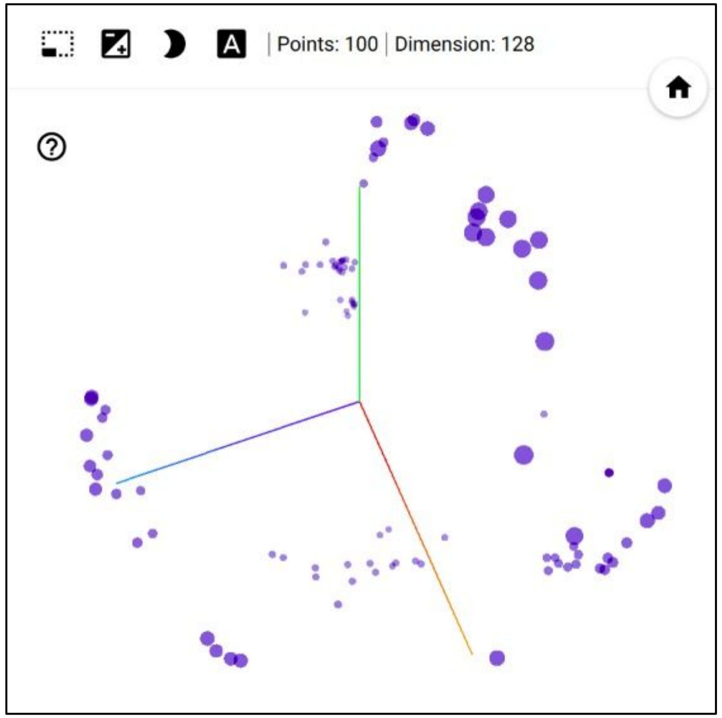
Hence the batch size should not be very small otherwise the model won't be able to learn much.

```
Epoch 1/5
75/75 [==============================] - 146s 2s/step - loss: 0.6671 - accuracy: 0.0394 - val_loss: 0.6908 - val_accuracy: 0.0291
Epoch 2/5
75/75 [==============================] - 143s 2s/step - loss: 0.3772 - accuracy: 0.0045 - val_loss: 0.5787 - val_accuracy: 0.0000e+00
Epoch 3/5
75/75 [==============================] - 144s 2s/step - loss: 0.2000 - accuracy: 0.0000e+00 - val_loss: 0.4579 - val_accuracy: 0.0000e+00
Epoch 4/5
75/75 [==============================] - 143s 2s/step - loss: 0.2028 - accuracy: 0.0000e+00 - val_loss: 0.1895 - val_accuracy: 0.0000e+00
Epoch 5/5
75/75 [==============================] - 143s 2s/step - loss: 0.1133 - accuracy: 0.0000e+00 - val_loss: 0.4456 - val_accuracy: 0.0049
<tensorflow.python.keras.callbacks.History at 0x7f36de84e810>
```

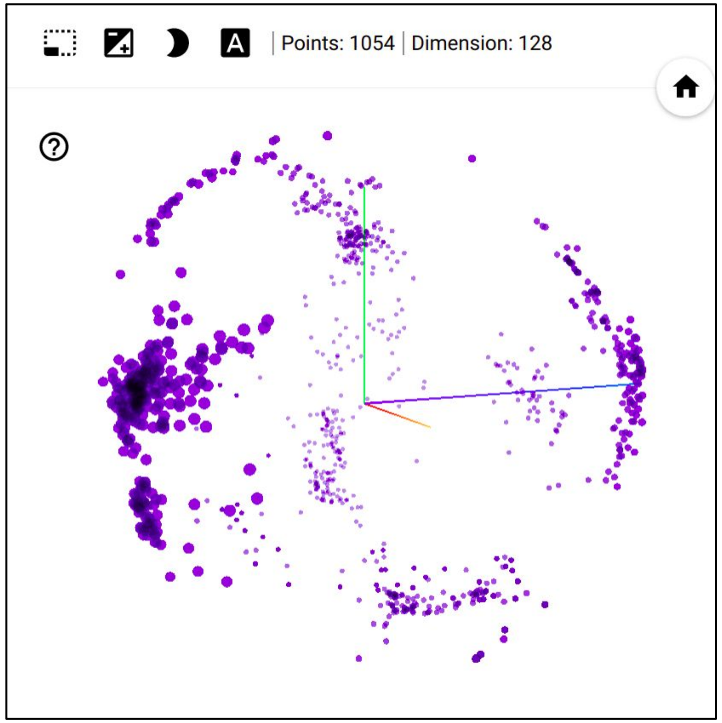
Embeddings 3D representation



Test



Test



All

Before training

After training

Face Recognition

- Embeddings derived from the images can be used for different tasks such as recognition, verification and detection.
- Here, a neural network is implemented to classify the embeddings
- Output is an array depicting the probabilities

```
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

- Sparse Categorical Loss was minimized.
- Images were classified with an accuracy of 90% on the validation set.

```
model2.fit(
    x = results[:700],
    y = results_label_int[:700],
    batch_size = 100,
    epochs = 5,
    verbose = 1,
    validation_data = (results[700:1000], results_label_int[700:1000]),
    shuffle = True, # doesn't matter if we have only one epoch or no batches,
    callbacks=[model_checkpoint_callback]
)
```

```
Epoch 1/5
7/7 [==============================] - 1s 31ms/step - loss: 1.6006 - accuracy: 0.2408 - val_loss: 1.4750 - val_accuracy: 0.6167
Epoch 2/5
7/7 [==============================] - 0s 10ms/step - loss: 1.4370 - accuracy: 0.7008 - val_loss: 1.3381 - val_accuracy: 0.7900
Epoch 3/5
7/7 [==============================] - 0s 8ms/step - loss: 1.3092 - accuracy: 0.8198 - val_loss: 1.2048 - val_accuracy: 0.8700
Epoch 4/5
7/7 [==============================] - 0s 9ms/step - loss: 1.1625 - accuracy: 0.8933 - val_loss: 1.0702 - val_accuracy: 0.9000
Epoch 5/5
7/7 [==============================] - 0s 8ms/step - loss: 1.0292 - accuracy: 0.9012 - val_loss: 0.9370 - val_accuracy: 0.9033
<tensorflow.python.keras.callbacks.History at 0x7fc6ca286090>
```

Results

The model was used to make predictions on the test set.

This is an example of a test set example being correctly recognized.

```
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 to be tackled

1. 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.
2. Online triplet mining techniques can be utilized to generate embeddings on the fly
3. The embeddings will also be further used to implement the task of face detection and verification.
4. Training can be customized by defining our own loss functions and optimizers