FaceNet

A Unified Embedding for Face Recognition and Clustering

The mean squares
Param Pujara 2020202008
Sanyam Jain 2020201006
Shubham Singh 2020202002

Problem Statement

Despite significant recent advances in the field of face recognition, implementing face verification and recognition efficiently at scale presents serious challenges to current approaches.

Solution Approach

- A system is described which converts an image to a fixed size embedding.
- Distances between any two embeddings is a measure of the similarity of the faces.
- The embedding is produced by training a siamese network with triplet loss.

Introduction to FaceNet

- A unified system for face verification (is this the same person), recognition (who is this person) and clustering (find common people among these faces).
- The embedding generated is an Euclidean embedding per image using a deep convolutional network.
- The network is trained such that the squared L2 distances in the embedding space directly correspond to face similarity.
- Faces of the same person have small distances and faces of distinct people have large distances.
- Face verification simply involves thresholding the distance between the two embeddings, recognition becomes a k-NN classification problem; and clustering can be achieved using off-the-shelf techniques such as k-means or agglomerative clustering.
- Dataset to be used for training:- Labeled Faces in the Wild (LFW).

Dataset Description

- Database of face photographs designed for studying the problem of unconstrained face recognition.
- For training, we are considering a subset of classes with 3 images each. That comes up to be 610 labels.
- All the images from LFW dataset are split into 2 sets -
  - For validation, to find appropriate threshold for distances of similar faces and different faces.
  - For testing, the images we are taking all labels in the LFW dataset and performing verification/recognition tasks.

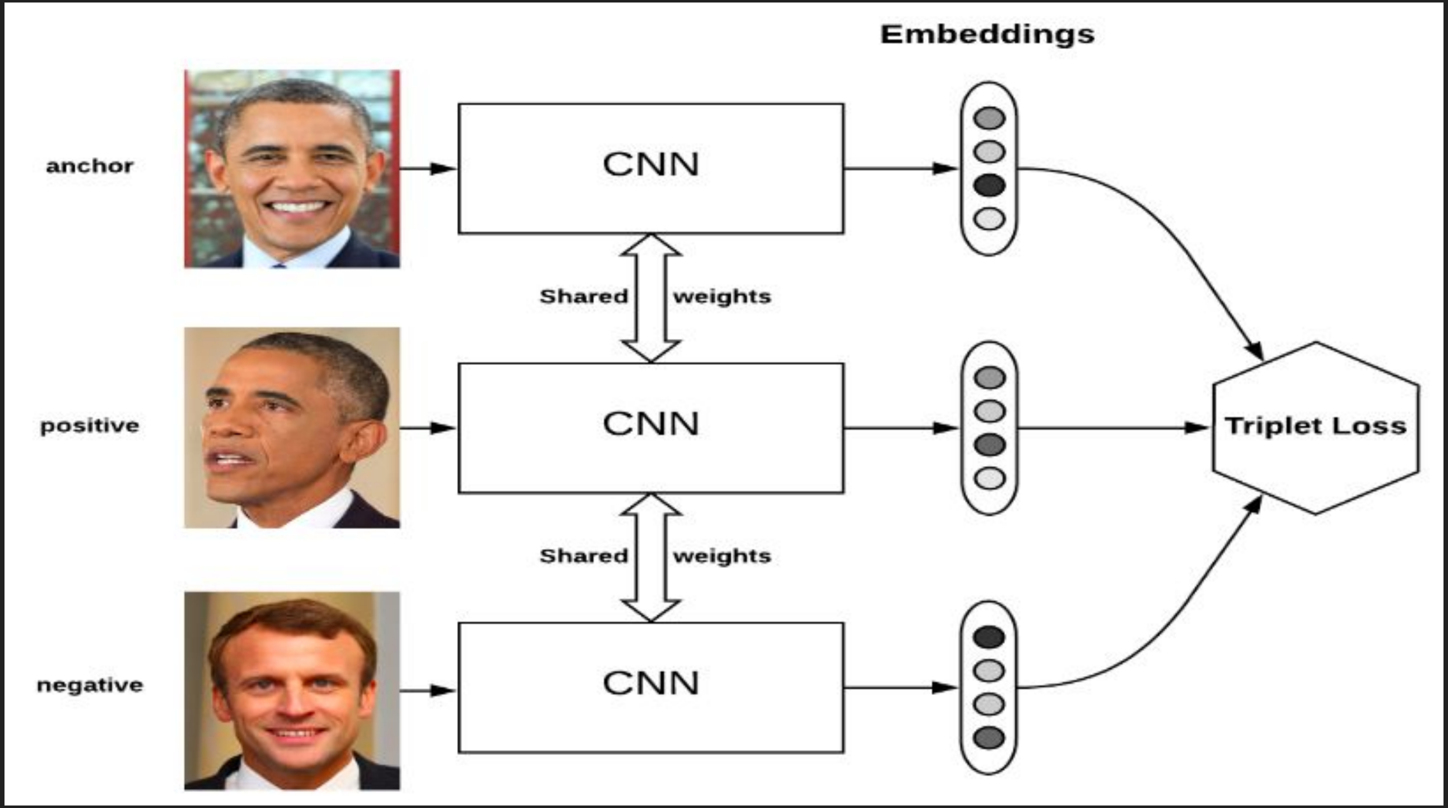
Expected Results

- The model will produce a fixed size embedding for a test data.
- If the test data belongs to a particular label then the distance between the embedding generated and stored embedding for that class should be less than a threshold.

Siamese Networks

- Siamese Networks are basically two same neural networks, the output of which are passed through the a function that calculates the distance between the inputs.
- For inputs belonging to same class, the outputs are close (i.e., distance is less) and for inputs belonging to different classes, the distance between outputs should be large.
- Triplet loss can be used to train a siamese network.

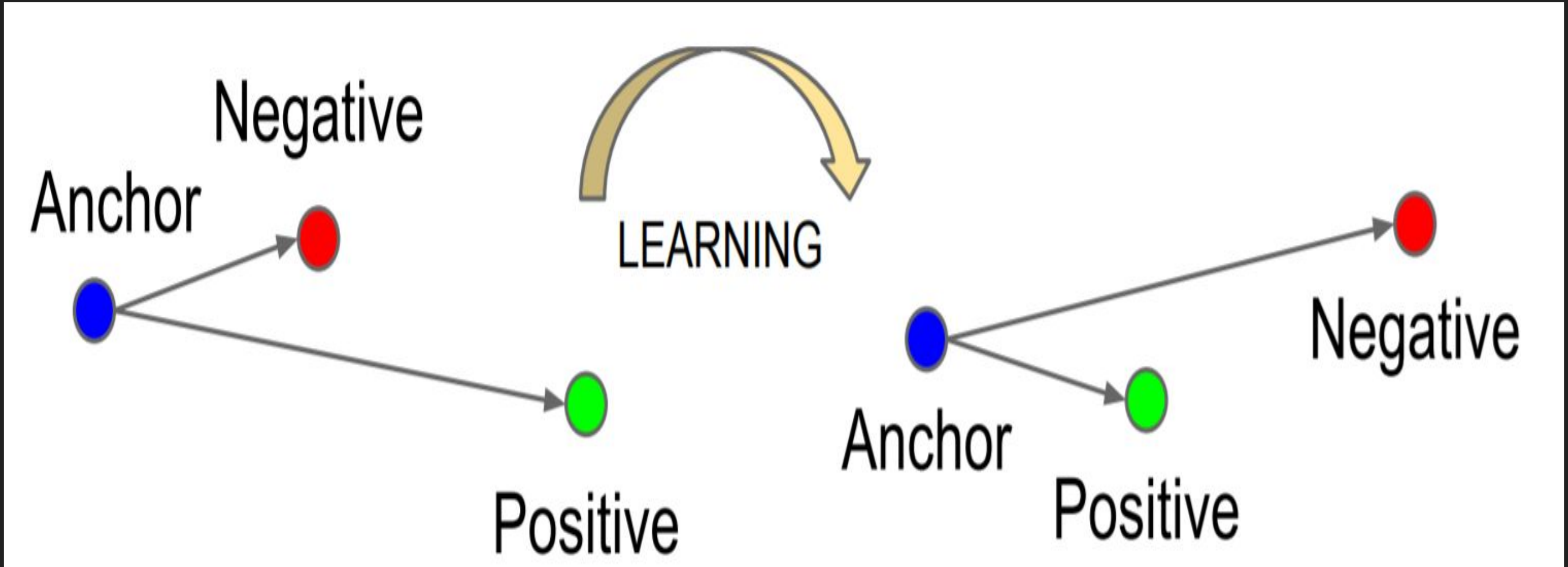
Siamese Network(Cont..)



Triplet loss

- It is a distance based loss function that operates on three inputs:
  1. anchor (a) : is any arbitrary data point,
  2. positive (p) : which is the same class as the anchor
  3. and negative (n) : which is a different class from the anchor
- Mathematically, it is defined as: $L=\max(d(a,p)-d(a,n)+margin,0)$.
- We minimize this loss, which pushes $d(a,p)$ to 0 and $d(a,n)$ to be greater than $d(a,p)+margin$.
- After the training, the positive examples will be closer to the anchor while the negative examples will be farther from it.

Triplet Loss(Cont..)



Triplet loss - loss function

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

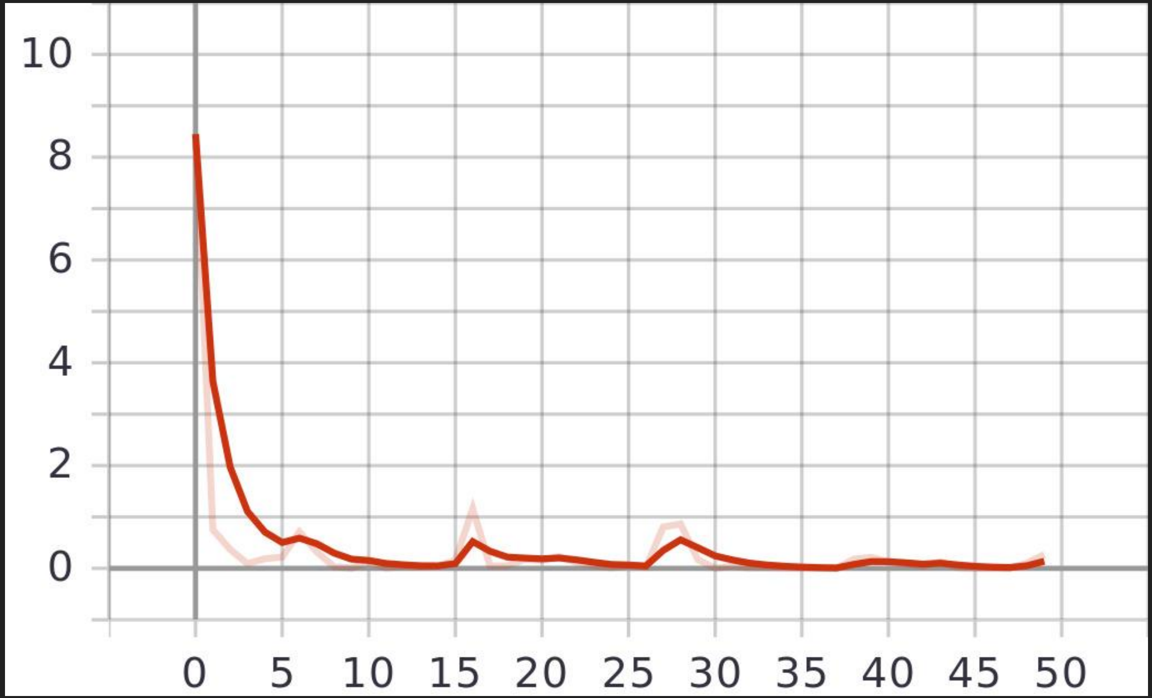
Model Summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 30, 30, 64)        1792      
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 15, 15, 128)       73856     
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 64)          73792     
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 64)          4160      
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0         
_________________________________________________________________
dense (Dense)                (None, 512)               524800    
_________________________________________________________________
dropout (Dropout)            (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328    
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896     
=================================================================
Total params: 842,624
Trainable params: 842,624
Non-trainable params: 0
_________________________________________________________________
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to                     
==================================================================================================
anchor_input (InputLayer)       [(None, 60, 60, 3)]  0                                            
__________________________________________________________________________________________________
positive_input (InputLayer)     [(None, 60, 60, 3)]  0                                            
__________________________________________________________________________________________________
negative_input (InputLayer)     [(None, 60, 60, 3)]  0                                            
__________________________________________________________________________________________________
sequential (Sequential)         (None, 128)          842624      anchor_input[0][0]               
                                                                 positive_input[0][0]             
                                                                 negative_input[0][0]             
__________________________________________________________________________________________________
merged_layer (Concatenate)      (None, 384)          0           sequential[0][0]                 
                                                                 sequential[1][0]                 
                                                                 sequential[2][0]                 
==================================================================================================
Total params: 842,624
Trainable params: 842,624
Non-trainable params: 0
```

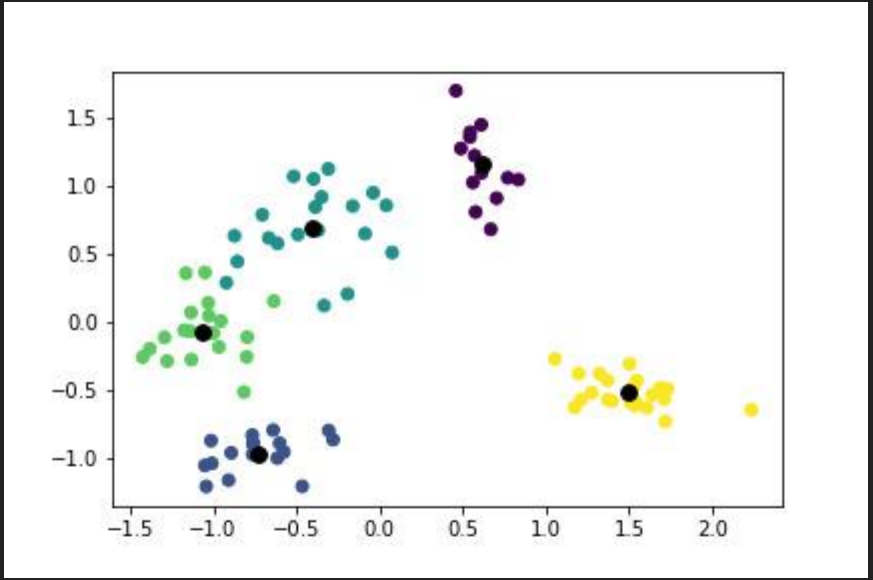
Training and Validation

- For training and validation, we have considered a small subset of the original dataset.
- For validation accuracy, we have used K-NN for better accuracy.

Epoch vs Loss



Scatter Plot for validation data and clusters



Validation Accuracy

K	Accuracy
1	83.33%
3	83.33%
5	79.16%
7	70.83%

Code Snippets

Triplet Generation

```
def generate_triplets(x, y, num_same = 4, num_diff = 4):
    anchor_images = np.array([]).reshape((-1,)+ x.shape[1:])
    same_images = np.array([]).reshape((-1,)+ x.shape[1:])
    diff_images = np.array([]).reshape((-1,)+ x.shape[1:])

    for i in range(len(y)):
        point = y[i]
        anchor = x[i]

        same_pairs = np.where(y == point)[0]
        same_pairs = np.delete(same_pairs , np.where(same_pairs == i))
        diff_pairs = np.where(y != point)[0]

        same = x[np.random.choice(same_pairs,num_same)]
        diff = x[np.random.choice(diff_pairs,num_diff)]

        anchor_images = np.concatenate((anchor_images, np.tile(anchor, (num_same * num_diff, 1, 1, 1) )), axis = 0)

        for s in same:
            same_images = np.concatenate((same_images, np.tile(s, (num_same, 1, 1, 1) )), axis = 0)

        diff_images = np.concatenate((diff_images, np.tile(diff, (num_diff, 1, 1, 1) )), axis = 0)

    return anchor_images, same_images, diff_images
```

Code Snippets

Triplet Loss Function

```python
def triplet_loss(y_true, y_pred, alpha = 0.2):
    total_length = y_pred.shape.as_list()[-1]
    anchor, positive, negative = y_pred[:,:int(1/3*total_length)], \
        y_pred[:,int(1/3*total_length):int(2/3*total_length)], y_pred[:,int(2/3*total_length):]

    pos_dist = tf.reduce_sum(tf.square(anchor - positive), axis=-1)
    neg_dist = tf.reduce_sum(tf.square(anchor - negative), axis=-1)
    basic_loss = pos_dist - neg_dist + alpha
    loss = tf.reduce_sum(tf.maximum(basic_loss,0.0))
    return loss
```

Models and experiments

- We have tried different deep learning structures and experimented with them.
  - Experiments on Batch Normalisation
  - Max/Min/Avg pooling
  - Dropout
  - optimizers
  - epochs and different batch sizes
- We tried different values of K in KNN classification.
- We trained our model to different numbers of triplets.
- We tried different subsets of LFW dataset for training model.

Accuracy on LFW

Model	Accuracy
Margin: 0.4, K=7, w/o dropout	52.70%
Margin: 0.4, K=9, with dropout	51.45%
Margin: 0.5, k = 1, w/o dropout	55.96%
Margin: 1, k = 7, w/o dropout, Batch normalization	52.74%

Final Model & Accuracy

Margin: 0.6, k=9, w/o dropout

Accuracy = 61.32%

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 64)	1792
max_pooling2d (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_1 (Conv2D)	(None, 15, 15, 128)	73856
max_pooling2d_1 (MaxPooling2	(None, 15, 15, 128)	0
conv2d_2 (Conv2D)	(None, 8, 8, 64)	73792
conv2d_3 (Conv2D)	(None, 4, 4, 64)	4160
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 64)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 512)	524800
dense_1 (Dense)	(None, 256)	131328
dense_2 (Dense)	(None, 128)	32896

Total params: 842,624
Trainable params: 842,624
Non-trainable params: 0

Timeline

WEEK 1

Understanding the problem statement, Studying Research paper, Solution approach in research paper, Initial evaluation

WEEK 2

More exploration in research paper solution, Existing solutions for the problem statement, Finding different datasets

WEEK 3

Writing code of solution approach, Debugging code, Evaluation our model with parameters, Mid evaluation

WEEK 4

Tuning Parameters, Finalizing dataset, Preprocessing dataset, Final run on dataset

WEEK 5

Testing model, Experiments, Feedback from testing and make changes accordingly, Final results, Final evaluations

Individual contribution

- Data Preprocessing & face detection - Param and Sanyam
- Generating triplets - Shubham and Sanyam
- Siamese Network and Triplet Loss function - Param and Shubham
- Experimenting with Different parameters - All

References

1. https://arxiv.org/abs/1503.03832
2. https://machinelearningmastery.com/how-to-develop-a-face-recognition-system-using-facenet-in-keras-and-an-svm-classifier/
3. https://towardsdatascience.com/siamese-network-triplet-loss-b4ca82c1aec8
4. https://towardsdatascience.com/understand-and-implement-resnet-50-with-tensorflow-2-0-1190b9b52691
5. https://www.coursera.org/lecture/convolutional-neural-networks/triplet-loss-HuUtN
6. https://medium.com/analytics-vidhya/facenet-architecture-part-1-a062d5d918a1
7. https://medium.com/@anilmatcha/summary-for-facenet-a-unified-embedding-for-face-recognition-and-clustering-cfe878027a3d