# Evaluation of Python FaceNet Implementation

Ibrahim Ahmed
Michigan State University
ahmedibr@msu.edu

## ABSTRACT

*The goal of this project is to assess the performance of the FaceNet facial recognition neural network designed by Google. The project is implemented entirely in a Jupyter Notebook to make it interactive and visual. Facial recognition plays a significant role in security today. Apple's release of the iPhone X back in 2017 abandoned the fingerprint sensor and implemented what is known as FaceID. FaceID uses neural networks to authenticate iPhone owners and is said to have a probability of 1 in 1,000,000 of producing a false positive authentication result [1]. Because of the large amount of time, computational resources, and data required to train a FaceNet network from scratch, this project uses a pre-trained "Inception" image classification neural network [2] as a base model to build off of.*

## 1. INTRODUCTION

Facial Recognition has come a long way over the last decade. Thanks to advancements in neural networks, facial recognition systems are able to reach extremely high accuracy of up to 99% [3]. One of the biggest advancements in Facial Recognition comes from the release of FaceNet, a neural network created at Google. FaceNet is able to map images of faces down to 3 dimensional euclidean space and minimizes the distance between similar faces.

### 1.1 Motivation

Facial Recognition sounds like a daunting task that would require lots of domain expertise to implement. To combat this notion, this project aims to show how easy it can be to interface with these neural networks and prove their accuracy. Although it takes quite a bit of time and resources to train a network like FaceNet, it is common and efficient practice to reuse the models and weights generated by previous neural networks when developing novel ideas to build off of previous work; for that reason, this project aims to also show how simple it is to build FaceNet off of an

existing model, which in this case is the "Inception" image classification neural network.

Facial recognition will play a crucial role in humanity's future. Common applications include:

- **Threat detection** - Facial recognition can be used to build a database of known criminals and provide a monitoring service for law enforcement to track down criminals using public surveillance systems.
- **Authentication** - Given that a face is a very unique piece of data attached to each human being, it can be used to authenticate users with online services, government services, home security tools, and much more.

## 2. PROBLEM STATEMENT

In order to implement FaceNet, there are a few areas of background knowledge necessary to understand all of the working pieces involved. The sections below will outline most of the background knowledge needed to implement FaceNet.

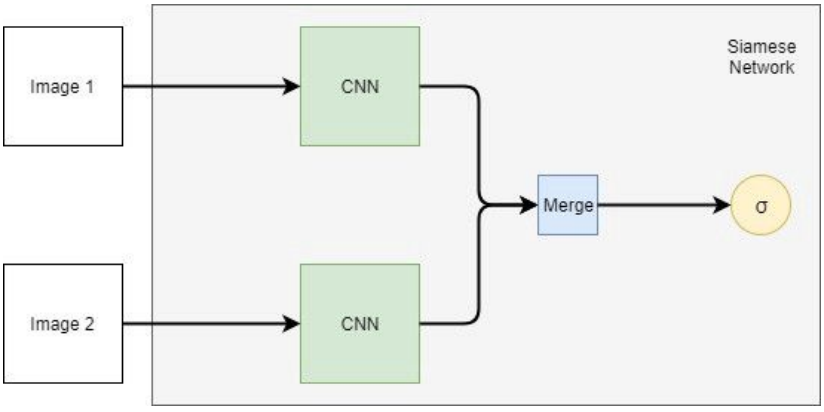
### 2.1 Neural Networks

FaceNet is a type of neural network. Neural networks are computer systems modeled after the human brain and nervous system which makes them adept at learning tasks. Neural networks generally require a large amount of data and are run on GPU's (graphics processing units) to maximize their efficiency.

### 2.2 Siamese Networks

FaceNet is a type of Siamese Network. A Siamese Network is a neural network that requires two separate inputs, $I_1$ and $I_2$ and one specific model, $M_0$. The two inputs, $I_1$ and $I_2$, and passed through the model $M_0$ and their outputs are combined to produce the result, as demonstrated in the figure below.

### 2.3 Inception Neural Network




The *Inception* neural network designed by Szegedy and co-authors is a network optimized for image recognition [2]. It is common practice in the machine

learning and data science field to build off of previous work to conserve efforts and improve existing work. FaceNet is able to be built off of the "Inception" neural network by modifying the loss function used by the model and feeding in input matrices with the correct dimensions. "Inception" uses 96x96 dimensional RBG (red, blue, green) images as its input and outputs a 128-dimensional vector to represent the image.

## 3. Related Work

The original FaceNet paper [3] outlines the original architecture of FaceNet and their evaluation results on the LFW (Labeled Faces in the Wild) dataset. The authors of the paper were able to achieve a new record (at the time) accuracy of **99.63%.** The authors also evaluated the model on the Youtube Faces DB and achieved an accuracy of **95.12%** (at the time).

Baidu, an Internet Company based in China, published a paper in 2015 on their facial recognition model using deep embedding which achieved **99.77%** accuracy on the LFW dataset.

## 4. PROBLEM STATEMENT

Building off of the Inception network, our model needs to implement the triplet loss function and then evaluate the models accuracy on the LFW dataset. Before the neural network can generate the embeddings for an image, it must be pre-processed to follow a format that the model can process. The dlib face alignment package provides some functions that we will use to perform this pre-processing. The dataset will evaluate the model on will be the LFW dataset, and a link to the specific dataset used is located in the appendix.

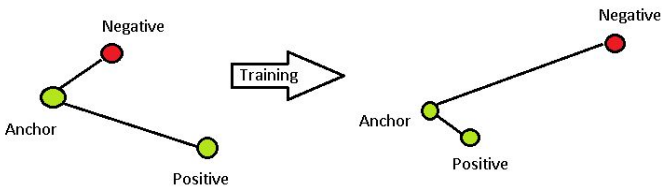
## 5. METHODOLOGY

The implementation in this project is completely self contained in a Python Jupyter Notebook which is found in the project repository. The sections that follow will describe all of the implementation details and the evaluation methods used.

### 5.1 Experimental Evaluation

The Inception network used to build the FaceNet model in this project was modified to use a Triplet Loss Function in order to generate valid embeddings for the images fed into the model.

The way FaceNet is able to recognize a face is by using what is called a Triplet Loss Function. The Triplet Loss Function is meant to make it so that when embeddings of faces are mapped to 3rd dimensional euclidean space, the distance between faces that are very similar are minimized and the distance between faces that are different are maximized. This loss function is then optimized in the Inception network which in turn results in embeddings that are very representative of the features displayed in the image.




The figure above outlines that task that the Triplet Loss Function is meant to solve. An *Anchor* image is an image of a face we would like to recognize. The *Positive* image is an image of a face which belongs to the same person found in the *Anchor* image, i.e. the same person. The *Negative* image is an image of a face which belongs to a different person.

$$Loss = \sum_{i=1}^{N} \left[ \|f_i^a - f_i^p\|_2^2 - \|f_i^a - f_i^n\|_2^2 + \alpha \right]_+$$

Above is a figure of the triplet loss function. In this formulation, *a* is the anchor image, *p* is the positive image, and *n* is the negative image. Once the loss function is minimized, the model is said to be optimized. The embeddings generated by the neural network can then be used to measure the distance between any two faces. The closer the embeddings are in space, the more similar those two faces are.

### 5.2 Experimental Setup

The Python Notebook titled "CSE841-ahmedibr-FaceNet-Project" contains the entire implementation of the FaceNet model, all of the pre-processing tasks involved, and the evaluation process.

Here is a collection of software used to implement the project and generate the results:

1. **Python 3.6** - The Python notebook is implemented in Python 3.6 due to its extensive library support.
2. **Keras and TensorFlow** - Keras and Tensorflow were used to generate the

FaceNet model. The pre-generated weights were adopted from the Inception project and their accompanying code libraries were brought along as well.

3. **Matplotlib** - For plotting, matplotlib was used to display images and graph results.
4. **dlib** - Used for image alignment, cropping, and centering.
5. **Jupyter Notebook** - Used as an environment to host and execute the project code.

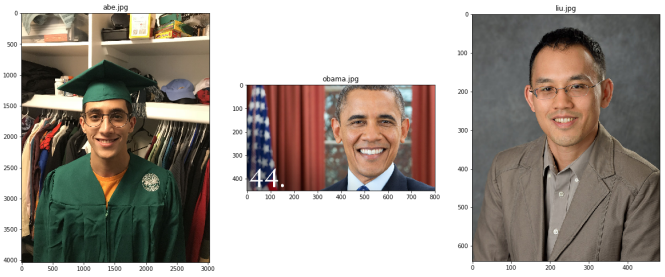
The notebook was tested on a Macbook Pro with 16GB of RAM and a 4 core i5 processor. All testing was performed on the LFW dataset, particularly the deep funneled dataset.

### 5.4 Image Pre-Processing

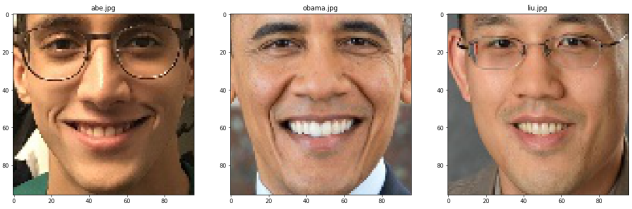
Although the original FaceNet model is designed to work on images of any dimension, it's usually a good idea to limit the size of the images fed into the model to conserve RAM and speed up training. With the savings in RAM and speed, there is a loss of information when downscaling the resolution of images fed into the model.

For this project, the chosen resolution of images fed into the model is 96x96 pixels. This means that before an image is passed into the model and a matrix representation is returned, the image must first be scaled down to 96x96 pixels.

In addition to the scaling applied to the images, it is important that the image of a face is aligned correctly and is cropped in such a way that the face is the main view of the image. This is where the dlib library was used to crop, center, and align the rotation of input images prior to passing them to the model. Below is an example of the preprocessing performed on a pair of test images used while experimenting with the model.




These three images were then pre-processed and the output of the preprocessing is found below.




As you can see, the images are much smaller in size and the focus of the image is on the faces contained in the original images. The size is also downscaled to 96x96 pixels.

### 5.3 Testing Procedure

The testing procedure was executed based on the test cases in the *lfw_evalutation_pairs.txt* text file contained in the project repository. The format of each line is as follows:

*If the line contains 3 tab separated values, the first value is the persons name, the second value is the id of the anchor image, and the third value is the id of the image to test on.*

Any line containing 3 values is considered a "positive test" i.e a test where the two images provided are of the same person. The model should produce representations where these two images have a small euclidean distance from each other.

*If the line contains 4 tab separated values, the first value is person A's name, the second value is the id of an image of person A, the third value is the person B's name, and the fourth value is the id of an image of person B.*

Any line containing 4 values is considered a "negative test" i.e a test where the two images provided are of different people. The model should produce representations where these two images have a large euclidean distance from each other.

In order to test if two images are a match, the following procedure is run:

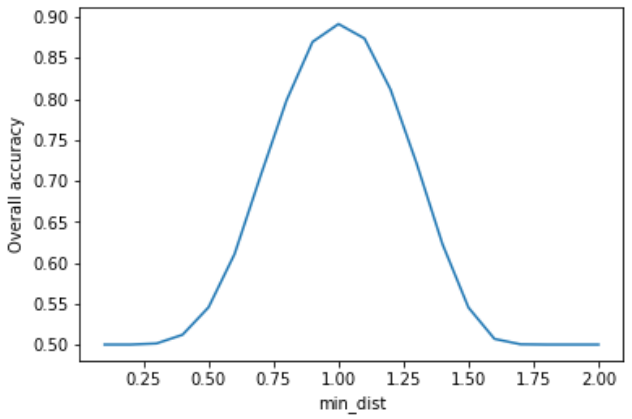
1. Generate matrix representations for both images using the FaceNet model.
2. Calculate the distance between the generated representations using classical euclidean distance.
3. Compare the distance to a threshold defined by the user.

The important part here is the threshold. If the threshold is very large, it will result in lots of false-positives because images which have a large distance from each other will result in a "true" being

predicted when checking for a match. If the threshold is too small, images with the same face will only match if they are extremely similar images.

In order to see which threshold was best suited for the LFW deep funneled dataset, the accuracy was plotted as a function of the threshold to see what trend the accuracy would follow as the threshold was changed.

The figure below is a plot of the overall accuracy of the model as it performed on the *lfw_evalutation_pairs.txt* dataset:




As we can see from the figure above, the optimal distance threshold is approximately 1.0 which results in an overall accuracy score of 89.14%.

It's important to note that approximately 59 testing pairs were omitted from the testing result because the dlib library was not able to properly pre-process their images.

### 5.3 Experimental Results

Using a threshold of 1.0, the FaceNet model was able to achieve an overall accuracy of 89.14% on the LFW deep funneled testing pair set with a total of 2970 sample pairs. The negative test cases had an accuracy of 89.69% and the positive test cases had an accuracy of 88.58% with a total of 2970 sample pairs.

## 6 RESULTS

The reason why the accuracy is not as high as the original FaceNet paper is because of the chosen dimension for the input images into the model, which is 96x96 pixels. When the images in this project are pre-processed, they lose a certain amount of information that would have been useful to the model when making predictions. This choice in dimension is due to computational resource restrictions.

Overall, the model performed very well given the size of the input images and the computational resource restrictions.

## CONCLUSIONS

This project was a great example as to how simple it is to interface with a FaceNet deep neural network model and use it for facial recognition. The model was able to produce image embeddings that were very representative of their contained features and performed well on the test data.

Future work could include increasing the dimension of input images to improve accuracy, retraining the model on new higher resolution images to produce weights that are more robust, and implementing a real time video stream to test the model with input data from the real world.

## APPENDIX

Here is a list of links referenced throughout the report:

1. Click here for a link to the LFW dataset used for evaluation.
2. Click here for a link to the text file which contains the person-pairs used for evaluation.mme
3. Click here for a link to download a copy of the project code.

## REFERENCES (at least 5 references)


[1] "Face ID Security." *Apple*, Nov. 2017, www.apple.com/ca/business-docs/FaceID_Security_Guide.pdf.

[2] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[3] Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[4] Liu, Jingtuo, et al. "Targeting ultimate accuracy: Face recognition via deep embedding." arXiv preprint arXiv:1506.07310 (2015).