ENM 531 Final Project Report
Face Recognition Based on the FaceNet Method Using Pytorch

Haoxiong Ma*
School of Engineering and Applied Science
University of Pennsylvania
(Dated: May 15, 2019)

FaceNet is a kind of neural network coding method that directly learns a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity. This method was developed and presented by Google Inc. in 2015. In other words, FaceNet inputs images of face in color or grayscale through neural network coding (Embedings) and outputs 128-dimensions-data, the data represent the characteristics of the face, take two face image the square of the Euclidean distance L2 paradigm can directly corresponds to the similarity between the two face image. In this project, I implemented the FaceNet neural network structure using pytorch and trained a FaceNet model and perform a face register and prediction process. This project basically realized the whole process from face detection to model training then to face embedding and face recognition.

I. INTRODUCTION

Face recognition is a complex process, it relies on a huge amount of data. If we only use the traditional CNN network to train the model, we need to label each face and perform a training process for each time of registration. That is way the FaceNet is developed by google. The face net use the new method of called the triplet loss to compare the same or different faces and use that as a label to train the model. This method gives us a way to train a model in advance to transform the face information to a 128digit vector. And by storing the vector we can predict whether the other face is close to the registered faces and how close they are so as to perform the face recognition.

In this report I introduce the whole process to implement the face net method. Firstly is using the pretrained model of MTCNN to perform the face recognition and face alignment to prepare the training, registering and the predicting faces. Then is using the structure of the FaceNet to build a deep neural network with the triplet loss function. Using the network to train the model and use a proper way to use the model to register and predict.

By the original author, FaceNet's accuracy was 0.9963 on the LFW dataset and 0.9512 on the YouTube Faces DB dataset. Since the author use the super computers with great GPUs to train the model for several days, the accuracy is relatively high. Due to time and hardware constraints, my implementation only shows the feasibility of this method and the accuracy of the LFW dataset is around 0.8 on my validation dataset.

* haoxiong@seas.upenn.edu

II. METHOD

A. Data Set Preparation: Face recognition and face alignment

MTCNN is a deep cascaded multi-task framework which exploits the inherent correlation between detection and alignment to boost up their performance. In particular, this framework leverages a cascaded architecture with three stages of carefully designed deep convolutional networks to predict face and landmark location in a coarse-to-fine manner[1]. Fig. 1 shows the schematic graph of the MTCNN.

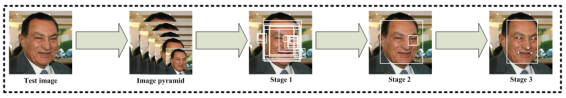

FIG. 1. Schematic graph of MTCNN

As a result the MTCNN can detect the face in a graph and shows the boundary and important point of the face. In this project, I used the code provided by MIT of this method to extract and clip the faces in the dataset, and the face images are transformed into 160*160 face images. The code is in `Face_Register.py` and the code are complex for this process but since the model of MTCNN is provided, the process is relatively simple. One example of face aligning is shown in Fig. 2.

In the project, I used the MTCNN to prepare the LFW dataset. The LFW dataset is a data set contains more than 13,000 images of faces collected from the web. Each face has been labeled with the name of the person pictured. Using this dataset to find out the face and the corresponding label and make a csv file to show the relationship between them then the data is prepared.

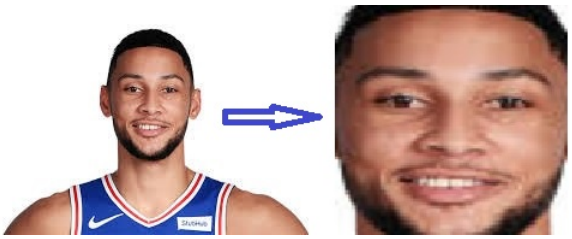

FIG. 2. Example of face aligning

B. FaceNet Structure and Implementation

The FaceNet structure[3] is as below, which is just a deep CNN architecture with the Triplet loss. The structure is a inception network shown in Fig. 3 and Fig. 4. The dimensions are shown in Fig. 5.

FIG. 3. FaceNet structure

FIG. 4. Network structure

For convenient, I used the ResNet34 predefined network, which is a deep architecture by Microsoft[**?**], in Pytorch and did some modification to this network. The network structure of ResNet34 is shown in Fig. 6 And this network is easier to optimize also can gain accuracy from considerably increased depth. By modifying the network, I use the network to output the 128D vector embedding of the face and used this embedding to calculate the triplet loss function and the registration or the prediction.

As for the triplet loss, the basic idea is getting three graphs at one time: one is the anchor, which is the original image, one is the positive image, which is the same persons image and a negative image from the other person as shown in Fig. 7

Feeding the three pictures through the neural network and comparing the three forward result by calculation the L2 norm. if the norm of the anchor with the positive is significantly smaller than the anchor with the negative, the model is better since it can distinguish the true images or the false images. As a result the triplet loss function is:

type	output size	depth
conv1 (7×7×3, 2)	112×112×64	1
max pool + norm	56×56×64	0
inception (2)	56×56×192	2
norm + max pool	28×28×192	0
inception (3a)	28×28×256	2
inception (3b)	28×28×320	2
inception (3c)	14×14×640	2
inception (4a)	14×14×640	2
inception (4b)	14×14×640	2
inception (4c)	14×14×640	2
inception (4d)	14×14×640	2
inception (4e)	7×7×1024	2
inception (5a)	7×7×1024	2
inception (5b)	7×7×1024	2
avg pool	1×1×1024	0
fully conn	1×1×128	1
L2 normalization	1×1×128	0

FIG. 5. Network dimensions

$$\sum_{i=1}^{N}\left[\|f(x_i^a)-f(x_i^p)\|_2^2+\|f(x_i^a)-f(x_i^n)\|_2^2+\alpha\right]_+ \quad (1)$$

By optimizing this loss function, we are trying to make the model to out put close value of the embedding for the same person while output far value for different person. By setting a specific threshold I can use the embedding to perform the clustering, which means, face recognition.

C. Training and validating

For training, I used the first 2/3 of the LFW dataset. For each training iteration, I used the code to get the minibatch and randomly choose the anchor, positive and negative images from the training dataset and store them to the data loaders. Then use the model to calculate the forward pass and then find the triplet loss and use the optimizer to optimize the loss. as for the validation, similarly, use the last 1/3 of the dataset to find the three images and calculate the loss and see whether the loss is rational.

```
Initialize the parameters
Define the model
Load checkpoint
Epochs loop
     If training,
          get batches
```

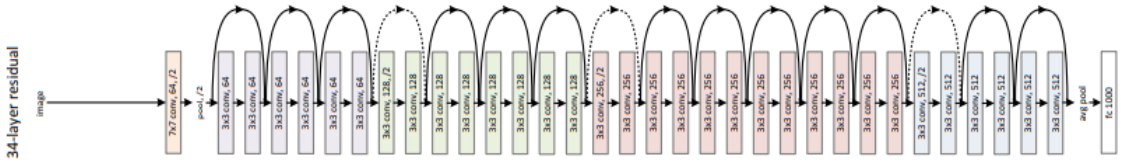

FIG. 6. ResNet34 structure

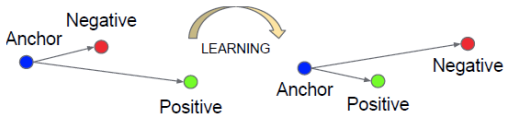

FIG. 7. Triplet loss

```
        get anchor, positive, negative
        triplet loss
        triplet loss.backwards
        optimizer.step
    if validating
        get anchor, positive, negative
        validation
    write information
save model
```

After each epoch, the program will show the accuracy of the training set and the validation set and store the model file as checkpoint for future use.

D. Face register and prediction

Face register is an embedding process, which is a simple way of putting the face images through the forward path of the trained deep network, the net work will output a 128D embedding information of the face, which represent the location of the face in the hidden space. The distance, which is the norm L2 norm of the two vectors represent the similarity of the two faces.

The register process will get the embedding vector from the input faces and store the face label corresponding with the embedding information. Each face will have one corresponding embedding and increasing the face pictures will increase the accuracy of the embedding data. There are two ways of storing the information of one person, one is to calculate the mean of the 128D vectors to save it as the specific information and the other is to store all of the vectors to the dataset. In practice, I used the second method in order to know which pictures is the prediction pictures the closest to.

Face prediction is a comparison process. This process will also input the prediction pictures through the neural network and output the embedding data of the prediction pictures. Using this embedding to compare the L2 norm with all of the registered data in the dataset and the closest one is the prediction of the face. Also a threshold should be set and when the minimum L2 norm is larger than the threshold then there is the false prediction to make sure if the prediction is a stranger, the system will return the label as stranger.

E. Project structure

There are three main parts of my project. `Face_Register.py`, `train.py` and `Predict.py`.

The `Face_Register.py` used part of the code provided by PanJinquan and MTCNN to perform face aligning and data processing. This code can also perform the data preparation, which is also a face aligning and image processing. I did some modifications to make the code work with my expectations. The code for face aligning and image processing is availabel at `https://github.com/PanJinquan/Face_Detection_Recognition/blob/master/faceRecognition/create_dataset.py`

As for the `train.py`, it calls a lot of classes including my model definition class, batch generation class and triplet loss class and so on. Part of the code references the code from tbmoon, and his code is available at `https://github.com/tbmoon/facenet`.

Then is the `Prediction.py`, this code generates the face aligning pictures and load and compare the distance of the L2 norm from the register dataset. then this code can use the `OpenCV` module to plot the name and face boundary on the input images to show the prediction result. The image processing part of the code is from PanJinquan as mentioned above.

The structure of the whole project is using the function in `Face_Register.py` to process the images from the dataset such as LFW or VGG to produce face images for training and validating. Then use the `train.py` to start training and save the model files and the accuracy and loss value. After training, find some registering pictures with labels and face and run the `Face_Register.py` to perform register process. Finally use some other pictures to test the prediction.`Prediction.py` can show the prediction with the name and the face boundary on the pictures, also when there are not-register faces, the `Prediction.py` will show the stranger label on the images. The flow chart of the project is shown in Fig. 8.

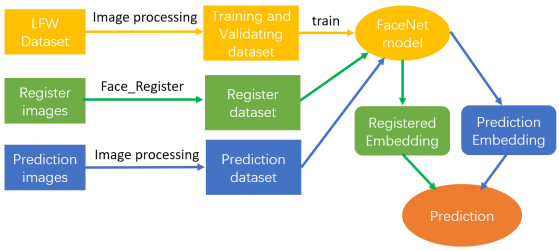

FIG. 8. Project flow chart

III. TEST RESULTS

A. Face aligning result

I used my code to prepare the dataset for training. Since the MTCNN is quite efficient and accurate, it is not difficult to find and cut the face to get the 160*160 pixels face images. One of the person in the LFW dataset is proceed to the face images as shown in Fig. 9. As a result, the LFW dataset is finally changed to 160*160 face images with a total of 5576 different people with 12110 pictures. Note that some of the pictures have multiple faces and will cause error during the process and I manually removed these images.

FIG. 9. Proceeded face images of 160*160

B. Training result

Since my hardware is not quite capable of performing such a large calculating task, it is relatively slow and runs out of memory frequently. It might because that my code has some redundant variables that takes up the memory or just my 8-Gigs of GPU ram is not enough for the project like this. To solve the problem, I decreased the batch size from 128 to 50, the memory problem has been improved. Besides, I set the total epochs to 50 in order to train a fast model just to show the program correctness. Also to save the time and memory, I used first 2/3 of the LFW to train and the rest 1/3 to perform validation. Although this setting may cause the results to be poor, but this is the best my hardware can offer. Using my NVDIA GTX1070 MaxQ GPU, the training trained 44 epochs before the CUDA ran out of memory and it takes around 6 hours to train 44epochs. The accuracy on the validation data set is around 80% and has a trend of increasing, also the triplet loss has a trend of decreasing to the 44 epochs. The loss value and accuracy change with the epochs are shown in Fig. 10.

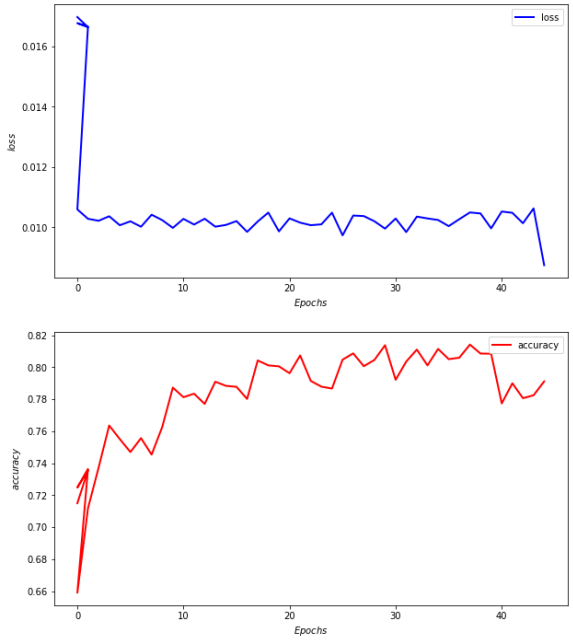

FIG. 10. Loss and Accuracy change with epochs

According to the author of FaceNet, in all their experiments we train the CNN using Stochastic Gradient Descent (SGD) with standard backprop and AdaGrad. In most experiments we start with a learning rate of 0:05 which they lower to finalize the model. The models are initialized from random, and trained on a CPU cluster for 1,000 to 2,000 hours. The decrease in the loss (and increase in accuracy) slows down drastically after 500h of training.

Compared to their result, since my network is slightly different and I used smaller batch size, so it might cause loner to reach convergence. Obviously, my training model is not converged till epoch 44, but it shows a converging trend which shows that the training process and the network is feasible.

Besides, according to the author, their maximum validation value is about 87.9%, which is significantly higher than my 80%, so my training process still has a a lot of room for optimization.

C. Register and Prediction

In the test, I chose two NBA players Lebron James and Ben Simmons. Each of them have 7 images with one face of them and use the same process to aligning the face and

perform forward pass. The original and processed images are shown in Fig. 11.

FIG. 11. Original Registered images

After getting the face images, through the forward pass, we can get the embedding vectors. The corresponding embedding vectors are saved in `faceEmbedding.npy`, the first several column of them are shown in Fig. 12.

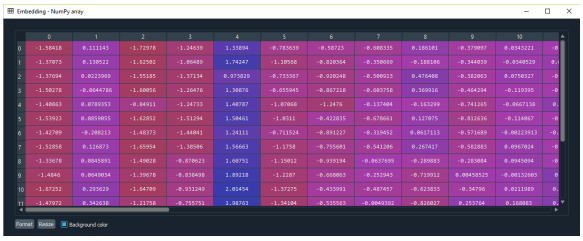

FIG. 12. Embedding values

Having the embedding information in my database, next step is to predict the face's name using the FaceNet. Firstly align the faces in one image to 160*160 images, then put these images through the forward pass and get the embedding vectors of theses faces. Comparing these information with the embedding in the database, calculating the L2 norm, find the minimum corresponding picture, then the label is the prediction.

For test one, there are only two people in the prediction images and all of them are registered in the database, the L2 norm between the faces and the database embedding is

[4.529482, 2.9258194, 4.4619975, 4.864114, 2.7946348, 4.9284544, 3.9776337, 4.791137, 1.5168058, 2.880635, 3.4881427, 2.6153257, 3.8996732, 1.8313037, 2.7024353, 1.2819384, 2.061896]

[3.053726, 1.7418196, 2.9120908, 2.537445, 3.386429, 3.061955, 3.2753685, 2.8943844, 3.0563002, 4.53195, 4.089935, 5.372189, 4.3862867, 4.084925, 4.7926326, 3.455629, 3.2465253]

We can see that they are some value that are much smaller than the others, and using the label of that minimum, we can find the closest face in the database, which is just the prediction of the face.

Then using the module `OpenCV`, draw the face boundary on the image and show the prediciton label to the face. The result is shown in Fig. 13.

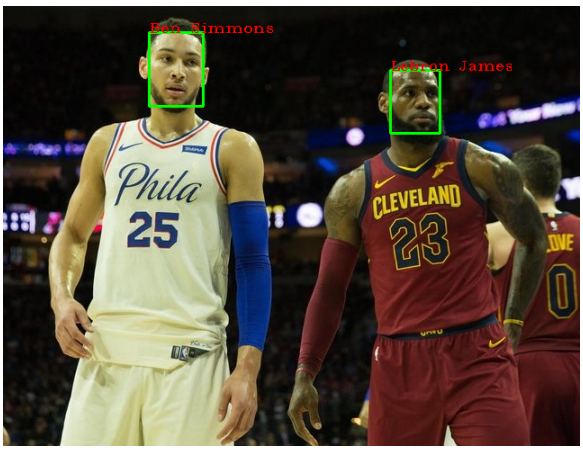

FIG. 13. Prediction without stranger

We can see that although the model is not that accurate, it can still predict the correct name of the faces, even the prediction face is from the different photo of the registered face.

In the above test, I did not add the stranger threshold since there is no stranger in the prediction photo. Thus, I did another test with the stranger in the picture and set an threshold to the recognizer. And the prediction embedding vectors norm are shown below.

[4.5659027, 3.4657204, 3.7709434, 4.338629, 3.5090516, 4.11044, 4.377149, 3.890586, 3.9394655, 4.9469438, 4.912274, 5.848147, 3.987892, 4.2113137, 6.116754, 4.445544, 4.625337]

[5.077701, 3.756811, 5.29414, 5.4823723, 4.151814, 5.864051, 4.85794, 5.7229085, 2.446993, 2.9426453, 3.7403586, 2.348097, 4.6764874, 2.7271767, 2.2662003, 1.9354942, 2.4789548]

[2.1453044, 2.8901641, 2.0869873, 2.0791588, 3.2723072, 3.4841385, 1.9658879, 4.0137205, 3.251206, 5.194651, 4.7723703, 5.5319443, 5.319653, 4.0663743, 4.7856016, 3.6204798, 3.3809352]

We can see that the first data is the face data norm from the stranger, and there is no smaller value than 2.3 like the other two does, which means the face embedding is far away from the registered faces. Setting the threshold to 2.3 we can separate this stranger face from the other registered faces.

Similarly, the last two data has some smaller value like 2.086 or 1.935 and they occur in the different location. Since in the registered database, the first 8 of the embeddings are from Ben, the other are from LeBron, which means, the second value of the norm has the closest value to LeBron and the third has the closest to Ben, then we can recognize the two faces along with the stranger faces. The resulted prediciton image is shown in Fig. 14.

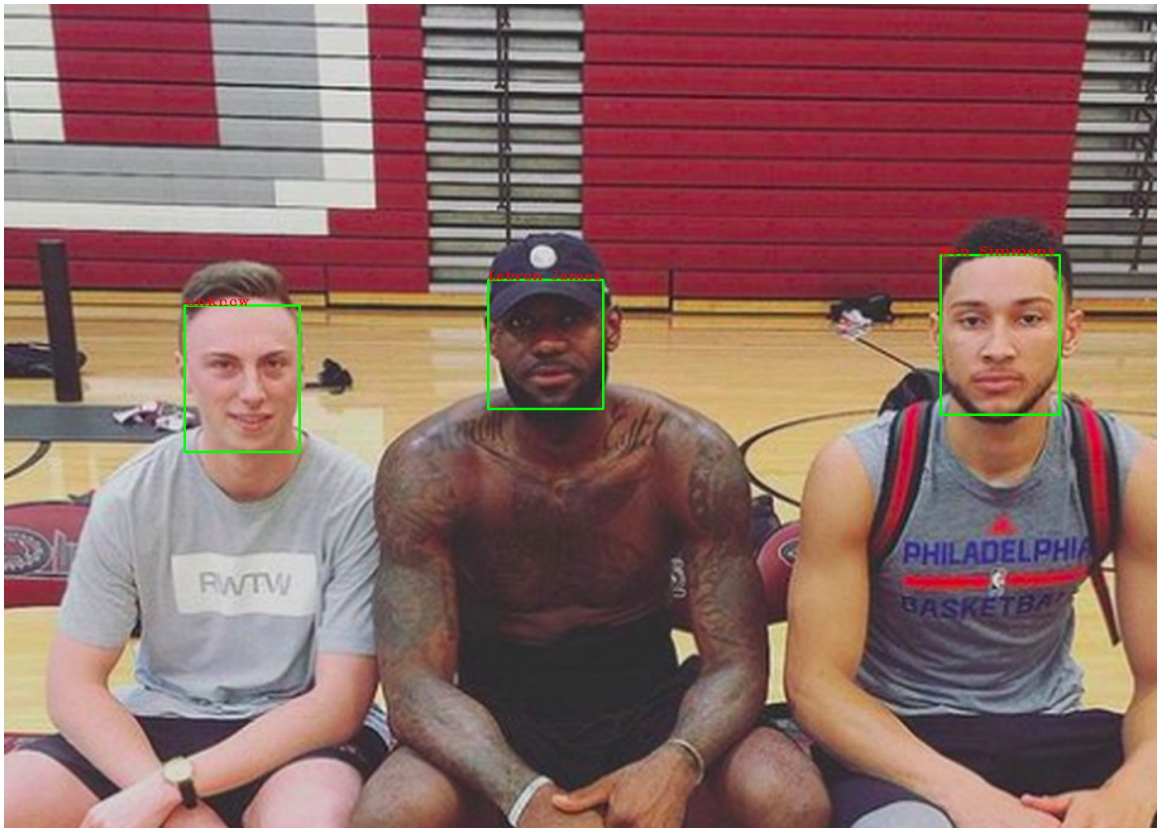

FIG. 14. Prediction with stranger

As a result, we can see that adding the stranger threshold, the result turns out to be quite good, as three of the people are labeled with the correct name. Even in the image, LeBron wears a hat and that hat will definitely decrease the accuracy of the prediction.

IV. FUTURE IMPROVEMENT

Firstly, there are some basic improvement that can be carried out. The most important one is that the current model is not trained well, that is caused by the low capability of the hardware and the irrationality of some code. I can try to define the variables, expecially those required grad, more careful to ensure there is no redundant use of the memory. After that, I can try using different batch settings and training dataset to train the model and see the accuracy of the well-trained model.

Also, the net work structure might not be perfect. I should try the google definition of the network if possible, or even develop a new better network.

During this project, I leaned a lot of new knowledge of Pytorch modeling, such as using dataloader, using different loop other than using iteration value to run a loop. I need to learn these knowledge better and try to do more practice and use the efficient code to solve the mathemetical problems.

V. SUMMARY

I provide a whole process to train the FaceNet model and use it to recognize faces. This is an interesting project since it can be developed to perform video face recognize or other practical use.

VI. ACKNOWLEDGMENTS

I would like to thank Professor Perdikaris for his well-organized and clear-taught lectures as well as his kindness and patience for my questions about the courses. Also I would like to thank Shuzhan Yang, Xinlong Zheng and Yuzhi Wang for their valuable discussion support for my assignment and final project.

Besides, I would like to thank the GitHub user, Pan-Jinquan and tbmoon for their great sample code, as well as the author of MTCNN and FaceNet for their code and paper.

VII. REFERENCES

[1] K. Zhang and Z. Zhang and Z. Li and Y. Qiao, Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks, IEEE Signal Processing Letters, 2016: 1499-1503, arXiv:1604.02878.

[2] Schroff. F, Kalenichenko. D, Philbin. J, FaceNet: A Unified Embedding for Face Recognition and Clustering, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2015, arXiv:1503.03832.

[3] He. K, Zhang. H, Ren. S, Sun. J, Deep Residual Learning for Image Recognition, 2015, arXiv:1512.03385.