Final project: Facial Recognition using Deep Learning in Tensorflow

Student Name: Junyu Ke (jk5643)/ Jinghong Miao(jm7012)

Github: https://github.com/Cantokk/medium-facenet-tutorial

Objective

This project aims at learning and building the process of convolution neural network to implement facial recognition by using Tensorflow. In this project, we will regenerate the result from Medium-facenet-tutorial provided by https://github.com/ColeMurray/medium-facenet-tutorial/tree/master. From this tutorial, we will learn the process of building a convolution neural network under Docker environment. We will generate the original result and tune the hyper parameters to generate our own results and do some comparisons.

Methodology

(1) Facenet

This project used a convolution neural network system based on Facenet. From the Facenet paper, the model is shown in Figure 1.

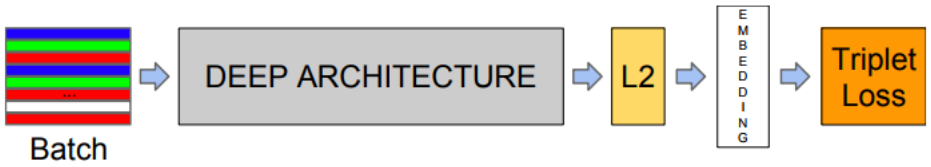


Figure 1 Facenet Model

From this model, we can see the input can be a batch of images, then the batch is input into a deep convolutional neural network. The normalization used here is L2. Then, it will do a vector embedding (explain later). Finally, it used a triplet loss function to minimize the embedding. The triplet loss function is shown in Figure 2.

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

Figure 2 Triplet function

For this function, the embedding f(x), from an image to a feature space is learned, such that the square distance between all faces of the same identity is small while the square distance between faces that belongs to different identity is large.

(2) Embedding

The other idea for this project is the vector embedding. An embedding is the collective name for mapping input images to vectors. By the experiment from the paper, a 128-dimension embedding is the best for final accuracy. Embedding can also be calculated between two vectors to indicate the similarities between the two vectors.

Tool

- Docker

- Tensorflow

- Dlib

- LFW dataset

Project Structure

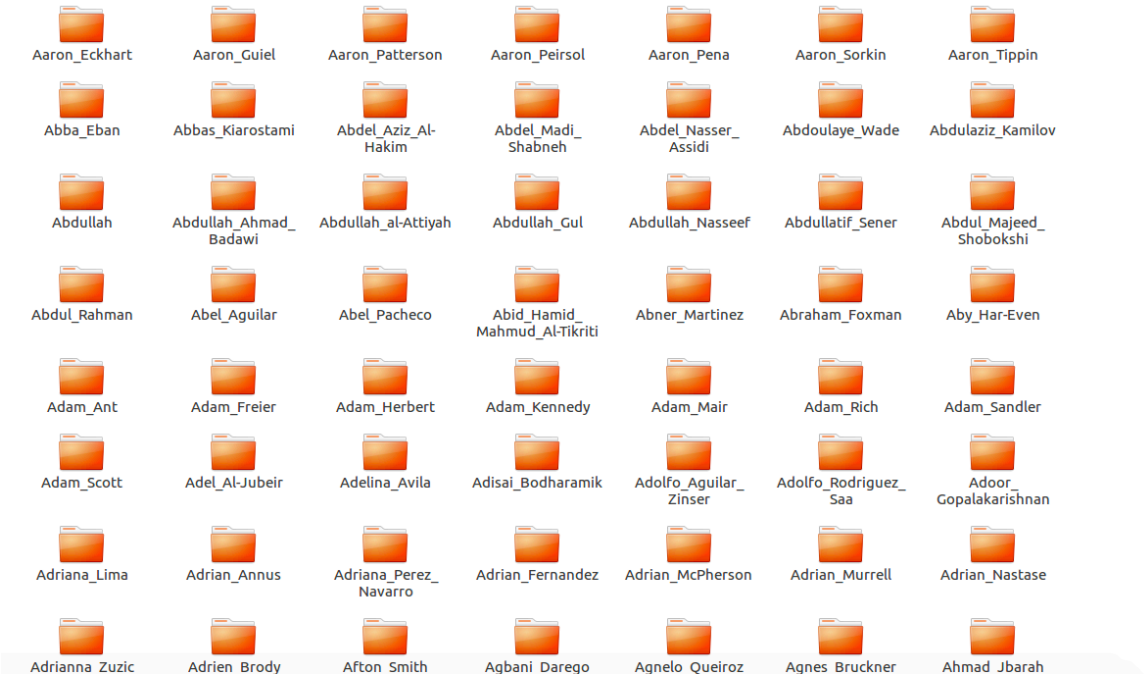
```
├── Dockerfile
├── etc
│ ├── 20170511-185253
│ │ ├── 20170511-185253.pb
├── data
├── medium_facenet_tutorial
│ ├── align_dlib.py
│ ├── download_and_extract_model.py
│ ├── __init__.py
│ ├── lfw_input.py
│ ├── preprocess.py
│ ├── shape_predictor_68_face_landmarks.dat
│ └── train_classifier.py
├── requirements.txt
```

The report will further explain some details with these documents.

Procedure

- Step 1(Load data)

  First, we download the LFW dataset into the <data> folder. The folder consists of around 1000 folders each of which contains pictures from the same person. The folder name is the name for that person which later will be marked as label when training the classifier.



- Step 2 (Environment Setup)

  We then setup the environment to run Tensorflow. Docker was installed as the environment in this project. The Docker image was provided by colemurray/medium-facenet-tutorial.

- Step 3 (Dlib predictor)

  Dlib predictor (provided by CMU) was used in this project to help preprocessing the pictures before training. Called by preprocess.py, Dlib predictor can detect the largest face within a picture and try to align the face to the center of the pictures.

  Inside preprocess.py, we modify the crop_dim parameter from 180 to 120 to try to make the pictures contains larger fraction of human face which we think can increase the

accuracy.

Figure 3 shows the result of preprocessing.

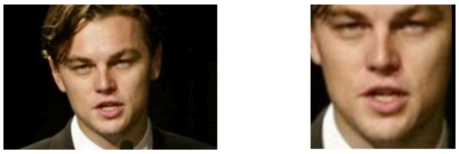


Figure 3 Result of preprocessing

- Step 4 (Model extraction)

Download and extract model from

https://github.com/davidsandberg/facenet/blob/master/src/download_and_extract_model.py

In this project, we used Inception Resnet V1 as our convolution neural network. After downloading, the pre-trained weights can be used to train the LFW dataset in this project.

- Step 5 (Embedding)

We used Inception Resnet V1 to get the embeddings and the embeddings will be used to train the classifier.

Ifw_input.py is responsible for embedding the dataset and we tuned some modification in the parameters that were used in this file.

In function read(), we tune the brightness and contrast to smaller range by setting as shown in Figure 4.

```
if random_brightness:
    image = tf.image.random_brightness(image, max_delta=0.6)

if random_contrast:
    image = tf.image.random_contrast(image, lower=0.1, upper=2.0)
```

Figure 4 Tune the random brightness and contrast

Our idea is to make the range larger to give more different embedding to train the classifier.

In function split_dataset(), we tuned the split ratio to 0.9 which was larger than that in default. We tried to train more data to improve the accuracy.

```
def split_dataset(dataset, split_ratio=0.9):
    train_set = []
    test_set = []
    min_nrof_images = 2
    for cls in dataset:
        paths = cls.image_paths
        np.random.shuffle(paths)
        split = int(round(len(paths) * split_ratio))
        if split < min_nrof_images:
            continue  # Not enough images for test set. Skip class...
        train_set.append(ImageClass(cls.name, paths[0:split]))
        test_set.append(ImageClass(cls.name, paths[split:-1]))
    return train_set, test_set
```

Figure 5 Split ratio

- Step 6 (Train classifier)
- After the embedding vectors are created, we used them as feature inputs to train an scikit-learn's SVM classifier.

  Train_classifier.py was responsible for training the dataset. Each picture will be embedded into 128-dimension vector and each batch was tuned to contains 256 vectors which indicated that each batch will return a 128*256 matrix during the training process. Also, the training process will randomly transform the pictures to allow more pictures to be trained. Finally, each label with less than 15 pictures will be dropped as we try to

allow those with more training samples to be accepted.

```
if __name__ == '__main__':
    logging.basicConfig(level=logging.INFO)
    parser = argparse.ArgumentParser(add_help=True)
    parser.add_argument('--model-path', type=str, action='store', dest='model_path',
                        help='Path to model protobuf graph')
    parser.add_argument('--input-dir', type=str, action='store', dest='input_dir',
                        help='Input path of data to train on')
    parser.add_argument('--batch-size', type=int, action='store', dest='batch_size',
                        help='Input path of data to train on', default=256)
    parser.add_argument('--num-threads', type=int, action='store', dest='num_threads', default=16,
                        help='Number of threads to utilize for queue')
    parser.add_argument('--num-epochs', type=int, action='store', dest='num_epochs', default=3,
                        help='Path to output trained classifier model')
    parser.add_argument('--split-ratio', type=float, action='store', dest='split_ratio', default=0.9,
                        help='Ratio to split train/test dataset')
    parser.add_argument('--min-num-images-per-class', type=int, action='store', default=15,
                        dest='min_images_per_class', help='Minimum number of images per class')
    parser.add_argument('--classifier-path', type=str, action='store', dest='classifier_path',
                        help='Path to output trained classifier model')
    parser.add_argument('--is-train', action='store_true', dest='is_train', default=False,
                        help='Flag to determine if train or evaluate')
```

Figure 6 Parameters that can be tuned to train the classifier

In the main function, we try to tune the default value for batch size to be 256, the ratio for splitting training and testing dataset to be 0.9 as we have explained before, minimum number of images to be dropped to be 15 while it was set to 10 before.

- Step 7 (Result)

The original result is shown in Figure 7:

```
1139  Winona_Ryder: 0.990
1140  Yoriko_Kawaguchi: 0.801
Accuracy: 0.908
INFO:__main__:Completed in 95.54084134101868 seconds
```

Figure 7 Origin result

Our result is shown in Figure 8:

```
1139  Winona_Ryder: 0.654
1140  Yoriko_Kawaguchi: 0.957
Accuracy: 0.911
INFO:__main__:Completed in 167.0648844242096 seconds
```

Figure 8 Result after tuning some parameters

We can see by training more samples for each label, enlarge the batch size, the accuracy is increased.

Future Work

In the future work, we will find a new dataset with gender as labels. Thus, we can re-train the dataset to predict the gender of each input pictures. While in the meantime, we will try again tuning some parameters to get higher accuracy.

References

1. Ieeexplore.ieee.org. (2017). *FaceNet: A unified embedding for face recognition and clustering - IEEE Conference Publication*. [online] Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7298682 [Accessed 15 Dec. 2017].