The Report for Facial Recognition

58119213 Danying Xu

Abstract: In this report, I will address on how I accomplish the facial recognition programme. I will give out the requirements in this work, how I solve the problem by using convolutional neural networks (CNN) and the results of predicting accuracy in my experiment.

Keywords: DNN, CNN, facial recognition, machine learning

1. Introduction

Facial recognition is a very popular topic in machine learning area in recent years. With the rapid developement in auto identify verification, the biology feature recognition technology grows fast. The general process of dealing facial recognition problem is to use the following steps: obtaining images, detecting faces, preprocessing facial images, feature extractions, comparasions and output results.

This experiment requires to finish the programming of facial recognition using knowledge in machine learning. The detailed requirements are as follows:

1) Each student is asked to give in 6 photos taken from the front.
2) Use DNN learnt in machine learning to train a model. Wrap up the recognition function as "facial_recognition", and facial verification function as "facial_verification".
3) Improve the accuracy of running time as much as possible.

2. Related Work

Considering that the data collected is far not enough to train a complete model, therefore I use the pretrained model downloaded from github. It has pretrained model and parameters based on relatively much bigger dataset. The model consists of 2 parts, MTCNN and FaceNet. MTCNN is used for face detection and FaceNet is used for facial recognition.

2.1. MTCNN

The pretrained model used MTCNN to pre-process the data, cutting them into 160×160px images, which performs better. It can accomplish both face detection and face alignment at the same time.

2.2. FaceNet

FaceNet learns a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity. Once the space is created, facial recognition and verification can be easily implemented using standard techniques with FaceNet embedding as feature vectors which is trained by triple loss $L(A, P, N) = \sum_{i=1}^{M} L(A^{(i)}, P^{(i)}, N^{(i)})$, where A is anchor, P is positive and N is negative.

3. Method

The method in my experiment mainly contains image cropping and model training. I will talk about them in details below.

3.1. Images Cropping

I use MTCNN to finish the cropping of the photos which is an important step of image processing. Since the pretrained resnet model was trained on 160×160 px images, so it performs better if we choose to apply all images resizing to this shape.

Before training the model, a pipeline is needed for the face detection. First, I utilize an imported package named Datasets.ImageFolder to read original images after resizing them into the same size. Then I use 8 batches to load datasets. At last, the batches are put into the MTCNN() one by one, saving the cropped images in file "mydata_cut".

Then I divide "mydata_cut" into another 2 files "mytrain_cut" and "mytest_cut". The former file is used to train the model and the test file is used to test the accuracy of the model. Then at last, I choose the parameters with best accuracy during training as the final model's parameters.

3.2. Model Training

The InceptionResnetV1 is trained with editable function parameters in tensorflow facenet repo which returns a well-trained model. I use VGGFace2 as the dataset for InceptionResnetV1. Therefore I connect VPN to train the model and download the pre-trained model named "myresnet1".

The model is trained within 8 epochs with cross entropy used as the loss function. Use writerto record the process of training.

3.3. Output

There are two functions for this project which are used for facial recognition and facial verification. Use torch.max() to save the predict results. For facial verification, if the output is same as id in names vector, then the verification is true, else is false.

3.4. Programming

First use the ported function InceptionResnetV1() and save the model as initial model by using torch.load(), names as "myresnet1.pt". The parameters in InceptionResnetV1 is classify="True", pretrained="vggface2", num_classes=41 and dropout_prob=0.5-0.8(will be discussed later in the results part).

Then train our model with the basic model "myresnet1.pt". Identify the parameters optimizer by optim.Adam(), scheduler by MultiStepLR(), epochs and batch_size. Use transforms.Compose to preprocess the images.

Next load the dataset "mytrain_cut" and loss function uses cross entropy to evaluate. Train the model one by one epoch.

At last, save the model also with torch.save() so that we can use it directly next time without training it again.

4. Results and Discussion

This part mainly discusses different results during adjusting the parameters in my codes.

4.1. Parameter Settings

When we directly use pretrained model to implement facial verification, we need to set several parameters: batch_size, epochs, learning rate and dropout_prob. Batch size is the number of data that put into the model for training. Considering the total number of images collected from students is 246, so I set batch_size=25. Epoch represents the process which all data is put into the net while finishing the feed forward and back propagation. Therefore I set epochs=8. Learning rate is another hyperparameter that plays an key role in whterther the object function can converge to the local

minimum and when it can converge to the minimum. I chose several learning rate for the model, lr=0.01/0.001/0.0001. When doing classification task, the dropout layer is gernerally added to the fully connected layer to prevent overfitting and imporve the genralization ability of the model. In my model, I set batch_size=25. Considering the total samples in "mytrain_cut" are 205 and the true samples in the final model are 246(all data should be used when training the final model), so the batch_size for the final model is 30.

4.2. accuracy in models

The parameters for the model I trained are as follows. I chose to adjust these parameters according to their last performance, so the differences can be observed easily between every two side-to-side rows.

The results are in table 1 and fig 1.

TABLE I
Different Accuracy with Differernt Parameters

model	1	2	3	4	5	6	7	8	9
batch_size	30	25	25	25	45	25	25	25	25
epochs	20	8	8	8	8	8	8	8	8
lr	0.01	0.0001	0.001	0.001	0.001	0.001	0.01	0.0001	0.001
dropout_prob	0.6	0.6	0.6	0.5	0.6	0.7	0.7	0.8	0.8
running time	3s	3s	3s	3s	4s	4s	4s	3s	4s
accuracy	80.488	87.805	63.415	85.366	82.927	90.244	70.732	80.488	63.415

Fig. 1. The best result during training with batch_size=25, epochs=8, lr=0.001, dropout_prob=0.7

From the results shown above, it can be easily seen that the model 5 has the best performance. Therefore I used these parameters to train the final model with only batch_size changed and named it as "myresnet_final_30.pt". It is trained on the dataset named "mydata_cut".

4.3. Questions I Met

1) Use VPN to train "myresnet1.pt" for the first time.
2) I tried to use the eigenvectors between different students to identify different perople, but the effects don't seem to be good.
3) During these different versions of models, I found it really hard to actually find the hidden pattern between these models.I implemented facial recognition many times and could just train the final model with parameters in best performed model 6.
4) I found the performances vary different in the models with "mytrain_cut" and "mydata_cut" as the training set. In fact, with many times real face recognition, I found that model 2 actually have the best predictions. I suppose this is because it has less variance than other models. And model 6 might be a little overfitting to some extent.

5. Conclusion

Through this experiment, I have a better understanding of facial recognition by using deep learning methods. I've learned how to download and use a pretrained model as well as fine tune with new data. Meanwhile, I've also got the hang of some useful operations in pytorch. Actually, at some points

it was totally a mess. I spent a lot time searching for the right model for solving this problem and many time to adjust the code to fit the specific conditions. And when practicing, there are many detailed things that we should pay attention to.

However, I think I didn't do enough work in image preprocessing, which may lead to the low accuracy under real time recognition. I only used methods of normalization and prewhiten in image preprocessing stage. That means I still have a lot of room for improvement.

In conclusion, I will learn more related knowledge in machine learning and try to put it in good use.

Acknowledgement

Big thanks to Dr. Beilun Wang and his marvelous machine learning class!

References

[1]https://www.cnblogs.com/zyly/p/9703614.html
[2]http://www.uml.org.cn/ai/201806124.asp?artid=20840
[3]https://blog.csdn.net/weixin_32974061/article/details/113995265