Report of face-recognition by finetuning ResNet and Haorui-Net

Haorui Li
61518407
Chien-Shiung Wu College
haoruili@seu.edu.cn

Abstract

For face recognition, first, I use MTCNN and face.evoLVe for automatic data cleansing and change parameters in MTCNN to avoid dirty data. Then I trained two models, one is self-modified Resnet called Haorui-Net which use Cov2d layers in ResNet for fracture extraction and use pooling and softmax layers to do classifications, another is InceptionResNetV1 with pre-trained weight, and fine-tuning the model on classmates' data. During the training process, I compare several different optimizers and combination of batch and epoch and use the best one. Finally the best model recognizes 86/104 classmates in 48s and it is Haorui-Net. At last, when it comes to why my model is better than ResNet, perhaps it is due to deeper network need more data size and my Haorui-Net is simpler so it can get its best with small data.

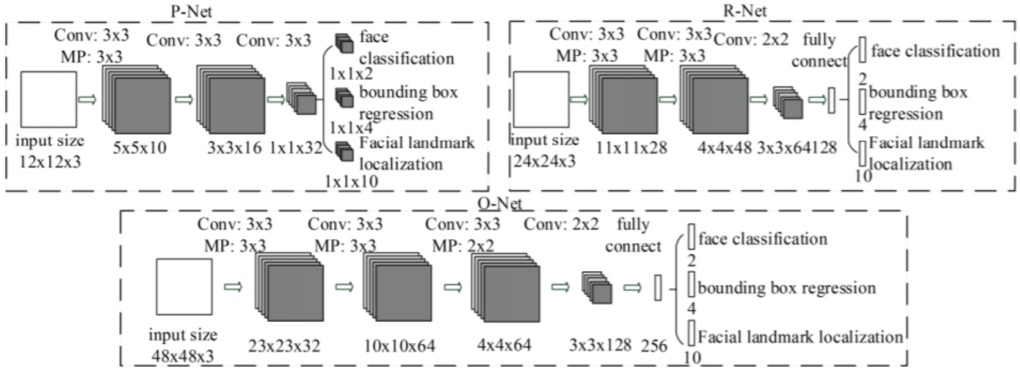
1 Data prepare

1.1 Face alignment

To begin with, I use MTCNN[1] and *face.evoLVe.PyTorch* for automatic face alignment.

MTCNN propose a deep cascaded multi-task framework which exploits the inherent correlation between them to boost up Resnet's performance on face alignment, the architecture is as follows:

Figure 1: MTCNN's architecture



But I find though MTCNN is very fast, but it sometimes go wrong and bring in dirty data, like the Figure2, and these dirty data will definitely bring catastrophe for model trainning.

Figure 2: Samples of dirty data by MTCNN

So I turn to *face.evoLVe's face-align tools* and finally get good data. This tool can be find at:

`https://github.com/ZhaoJ9014/face.evoLVe.PyTorch`

This tool is about 4-times slower than MTCNN, but brings no dirty data.

But I am wandering why MTCNN get these wrong results, because it is almost at state-of-the-art. And the face.evoLVe tool is designed base on MTCNN. So I test several parameters, It shows that when the default minim-window-size is undefined, mtcnn starts from 10x10 and tends to get wrong faces. So after I set the minimum size at 40x40, all results are good.

1.2 Rebuild folder architecture

For quick detect image labels, I use *torchvision.datasets.ImageFolder* to automatically read classmates name. To use this function, I rebuild the data folder's architecture by code.

Exactly, I use os.rename and string.split. Following are some codes I use to split the student number:

```
def replaceDirName(rootDir):
  #Change the folders' name under rootDir, split the student number by '-' or '_'
    num = 0
    dirs = os.listdir(rootDir)
    for dir in dirs:
        print('oldname is:' + dir)
        num = num +1
        try:
          temp = dir.split('_')[1]
        except IndexError:
          try:
            temp=dir.split('-')[1]
          except:
            print("This is not Number-Name structure", dir)
            continue
        except:
          print("This is not - or _ structure", dir)
          continue
        print('new name:',temp)
        oldname = os.path.join(rootDir, dir)
        newname = os.path.join(rootDir, temp)
        os.rename(oldname, newname)#replace
replaceDirName('align_data')
```

Listing 1: Change folder names for ImageFloder function

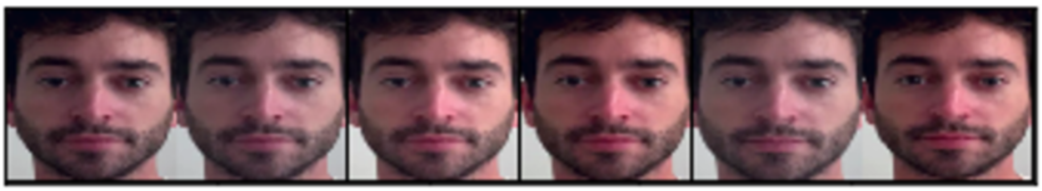
After rebuild the folder architecture, *torchvision.datasets.ImageFolder* is able to automatically read sub-folders' name as image label.

1.3 Transforms

After clean the data and align all the faces, I made some extra preparations for models robustness and these work has brought about 3-point increase in test accuracy.

When load in the data I perform some random transforms to the images to improve training. Different transforms can be attempted and I tried various ones, like Random-Color-Jitter and Random-Rotation, along with Random-Horizontal-Flip.

Figure 3: Examples of random Color Jitter



Finally I choose all these transforms to improve the model's robustness. And the random-color-jitter improves about 2 points in accuracy probably because classmates take photo at different light environment.

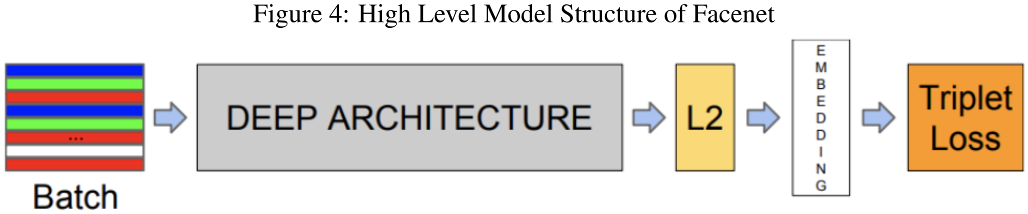
2 Design model architecture

Due to the fact that the data we have is small scale, it will be hard to train a model without over-fitting. So I think it is recognized to use some pre-trained model and do the fine-tuning. What I have to do is design the final layers.

2.1 Pre-trained ResNet

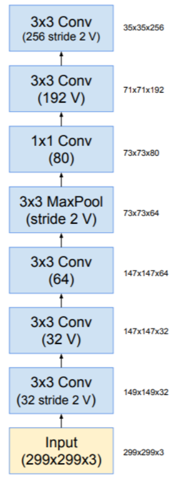
The pre-trained weight I download is the Facenet trained by Google. They use triple loss and finally get 0.997 accuracy at Lwf, the High-Level model structure of Facenet is as follow[2]:

Figure 4: High Level Model Structure of Facenet



And for the first model, I use Inception-ResNet[3] to fine-tuning the model, which is designed for fine-tuning Facenet. The architecture of Inception-ResNet is as follow:

Figure 5: Inception-ResNet



The code of final layers are:

```
        self.block8 = Block8(noReLU=True)
        self.avgpool_1a = nn.AdaptiveAvgPool2d(1)
        self.dropout = nn.Dropout(dropout_prob)
        self.last_linear = nn.Linear(1792, 512, bias=False)
        self.last_bn = nn.BatchNorm1d(512, eps=0.001, momentum=0.1, affine=True)
        self.logits = nn.Linear(512, tmp_classes)
```

Listing 2: Final layer Codes

And I will modified the final layers, then test which model is the best.

2.2 Modified ResNet

From the upper section we can see the final six layers are:

```
[Block8(
   (branch0): BasicConv2d(
     (conv): Conv2d(1792, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
     (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
     (relu): ReLU()
   )
   (branch1): Sequential(
     (0): BasicConv2d(
       (conv): Conv2d(1792, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
       (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
       (relu): ReLU()
     )
     (1): BasicConv2d(
       (conv): Conv2d(192, 192, kernel_size=(1, 3), stride=(1, 1), padding=(0, 1), bias=False)
       (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
       (relu): ReLU()
     )
     (2): BasicConv2d(
       (conv): Conv2d(192, 192, kernel_size=(3, 1), stride=(1, 1), padding=(1, 0), bias=False)
```

```
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True,
    track_running_stats=True)
        (relu): ReLU()
      )
    )
    (conv2d): Conv2d(384, 1792, kernel_size=(1, 1), stride=(1, 1))
  ),
  AdaptiveAvgPool2d(output_size=1),
  Linear(in_features=1792, out_features=512, bias=False),
  BatchNorm1d(512, eps=0.001, momentum=0.1, affine=True,
    track_running_stats=True),
  Linear(in_features=512, out_features=8631, bias=True),
  Softmax(dim=1)]
```

Listing 3: Final layers

Because earlier layers as containing the base-level information needed to recognize face attributes and base level characteristics, so I want to cut the layers after Conv2d and use some my own code, and just updating the final layers to include another 104 faces.

Put all beginning layers in an nn.Sequential:

```
model_ft = nn.Sequential(*list(model_ft.children())[:-5])
```

Listing 4: Keep the conv2d layers

Now, model modified is a torch model but without the final linear, pooling, batchnorm, and sigmoid layers.

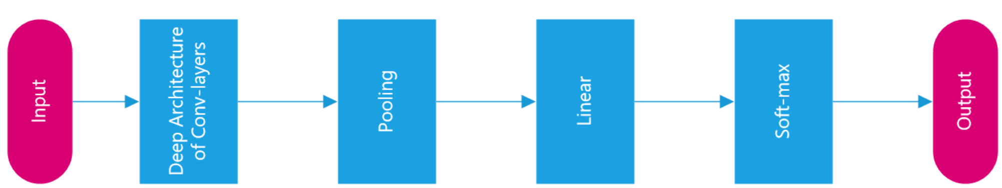
After this, I design another final layers class includes sample Flatten and Normalize layers in a gesture to use features extracted by Cov2d layers, the codes are:

```
#Change the final layers as follows
model_modified.avgpool_1a = nn.AdaptiveAvgPool2d(output_size=1)
model_modified.last_linear = nn.Sequential(
    Flatten(),
    nn.Linear(in_features=1792, out_features=512, bias=False),
    normalize()
)
model_modified.logits = nn.Linear(layer_list[4].in_features,104)
model_modified.softmax = nn.Softmax(dim=1)
model_modified = model_modified.to(device)
```

Listing 5: Haorui Net

So the architecture is:

Figure 6: Haorui-Net Architecture



We can name it Haorui-Net. In the next section I will train these two models and show some details to pick the winner.

3 Training and select parameters

After design the model, I begin the training step. Tried different epoch, batch size, learning rate and models.

3.1 Check GPU Memory

The options of batch size are often limited by GPU memory.

On my machine, I have a single Tesla-P-100 with 16280 MiB memory, which means I have more choice on batch size and epochs.

Use '!nvidia-smi' I get the following in formations of GPU memeory, it shows that 6869 MiB memory is located at device and I still have space to test.

Figure 7: 24 Epochs and 64 Batch-size

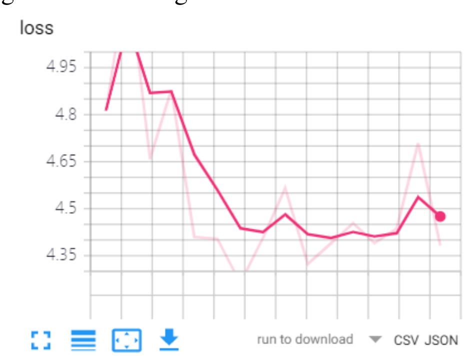
```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.82       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   49C    P0    36W / 250W |   6869MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

3.2 Should I use Adam?

Optimizer plays an important role in deep-learning, and different optimizer can have totally performance.

As we all know, "Adam" is honoured as an excellent optimizer, but should I use it too in my work? So I test another theoretically-good optimizer which is called RMS-prop, and the results in Tensorborad-X are as follows:

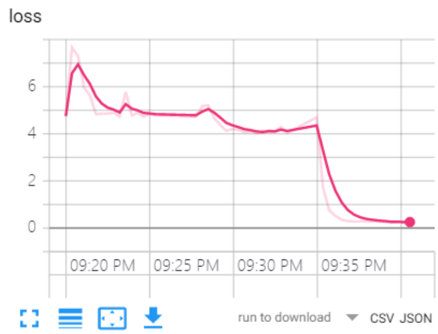
Figure 8: Trainning loss of RMS in TensorboradX



It shows that the loss of RMS optimizer finally convergences at about 4.5, and in the preliminary stage it really decreased fast.

But with the same epochs and batch-size, which is 32 and 128, the Adam optimizer performs really better:

Figure 9: Trainning loss of Adam in TensorboradX



It shows that the loss of Adam optimizer finally convergences at about 0.2, even though in the preliminary stage it decrease slower than RMS but finally it convergences at a better point.

I also test the FPS of training and testing, but it shows that this two optimizer are almost the same:

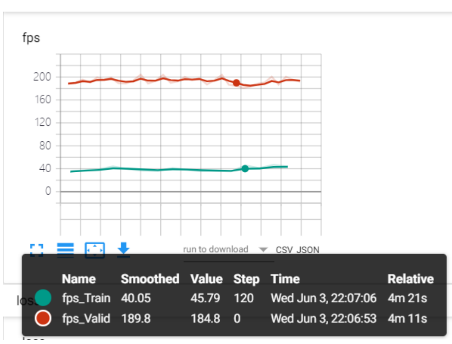


Figure 10: FPS of RMS

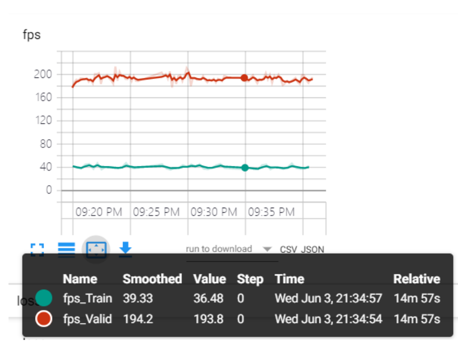


Figure 11: FPS of Adam

As its shown above, Adam optimizer performs better and I will use it in trainning my model.

3.3 Epochs and batch-size

After choose several combinations of epochs and batch size, I get the results as follows on Inception-ResNet:

Table 1: Records of combination for ResNet

Epochs	Batch size	TP	Train FPS
10	16	21	427.4
24	16	26	420.7
24	32	41	279.6
24	64	75	153.4
32	64	71	161.5
24	128	80	149.5
32	128	77	233.9
24	256	70	183.3
32	256	77	192.8
64	256	76	155.3

From the chart we can see, more batch size often means better performance, but with more batch size, sometimes it need more epochs to minimize the loss, just like 256 batch size performs weaker than 128 batch size in 24 epochs, and become better in 32 epochs.

So finally, the ResNet performs its best at 24 epochs, 128 batch size and reaches 82 true positive. This model was saved as '24-epoch-128bz-VGGFACE2-TEST80ACC.pb'.

With the chart above, I can qiuckly choose some combinations for Haorui-Net, and the results are as follows:

Table 2: Records of combination for Haorui-Net

Epochs	Batch size	TP	Train FPS
24	64	71	153.9
24	128	82	171.4
32	128	86	255.5
32	256	77	210.4
64	256	77	195.7

Luckily, the Haorui-Net performs better than ResNet its best at 24 epochs, 128 batch size and reaches 82 true positive. This model was saved as '32-epoch-128bz-MODIFIED-TEST86ACC.pb'.

So I'm proud to announce that Haorui-Net becomes the winner in this combination, with ten more ture-positive!

But what I want to point out is that, Haorui-Net is weaker in the decrease of loss, for ResNet, the minimum of loss is about 0.27 while training, but for Haorui-Net, the minimum loss is about 3.8, it probably means ResNet is designed more smarter in track and reduce the loss.

4 Test and Conclusion

Because in the training stage I use Face.LVe to process face images, now when test, using this tool will be slow, so I turn to MTCNN and by change its parameters it seldom detect wrong images.

```
mtcnn = MTCNN(image_size=160,
          margin=0,
          min_face_size=60,
          thresholds=[0.6,0.7,0.7],
          factor=0.709,post_process=True,device=device)
```

Listing 6: MTCNN Parameter

I load the best model of Haorui-Net and the test of Face-Recognize shows:

Figure 12: Face Recognize Test

人脸识别的考察结果：
人脸识别的准确率是：0.8269230769230769
整个人脸识别的运行时间是： 48 s

It takes about 0.46 second per student for face recognize and the accuracy is 82.7% for the best model of "Haorui Net", not so bad.

But this result is slower than ResNet:

Figure 13: Face Recognize Test

人脸识别的考察结果：
人脸识别的准确率是：0.7884615384615384
整个人脸识别的运行时间是： 38 s

For Face-Verification, I find that it takes too long to run the function because it have to check all the faces, so I just check the first 40 faces and get the results below:

Figure 14: Face Verification Test

人脸认证的考察结果:
精度：0.875
回归率：0.875
特异性：0.9987864077669902
F1值：0.875

In conclusion, I test the Resnet and hand-modified Haorui-Net, all based on pretrained weights, finally Haorui-Net win the competition in accuracy. I use Adam optimizer because it performs best in minimising loss. For the best model, it takes about 0.46 second per student for face recognize and the accuracy is 82.7 %.

Why my model can performers better than this champion model? (though the resnet model in paper get 99.5% accuracy and only 76% in my work) I think perhaps it because our database is small and only need to classify 104 people, when the neuronal network is more and more deep, it needs more data to get its best accuracy, and my Haorui-Net is simpler, which means with small data it is more easy to be trained at its best. Last but not least, the gap between these two model is small, with more experiment of combination of epochs and batchsize, perhaps ResNet can give better results.

5 Expectations

Though my model get a good result in accuracy, but there still remains something I want to explore.

For example, my face-verification function runs too slow to verified all pictures and names, I think it perhaps due to my algorithm is O(n^2) and I write too many works to move data between GPU and CPU which is time-consuming. And I think perhaps use B+ tree or some other data structure can speed up the searching process, also, keep all the data on one device can avoid moving them.

Moreover, though my model works great on our classmate-dataset, but for actual industrial demand, sometimes the faces in picture is really small, slant, and only have side faces, like surveillance videos. To recognize faces in these scenes, perhaps we have to made a 3D-model for faces[4], and use more skills to avoid overfitting like knowledge-distillation.[5]

In conclusion, there are still large space to modify this work for specific context.

References

[1] Zhang, K., Zhang, Z., Li, Z. Qiao, Y. (2016). Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks.. CoRR, abs/1604.02878.

[2]Schroff, F., Kalenichenko, D. Philbin, J. (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering (cite arxiv:1503.03832Comment: Also published, in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2015)

[3]Szegedy, C., Ioffe, S., Vanhoucke, V. Alemi, A. A. (2017). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. Proceedings of the 31st AAAI Conference on Artificial Intelligence (p./pp. 4278–4284), : AAAI Press.

[4]Dou, P., Zhang, L., Wu, Y., Shah, S. K. Kakadiaris, I. A. (2015). Pose-robust face signature for multi-view face recognition.. BTAS (p./pp. 1-8), : IEEE. ISBN: 978-1-4799-8776-4

[5]Luo, P., Zhu, Z., Liu, Z., Wang, X. Tang, X. (2016). Face Model Compression by Distilling Knowledge from Neurons.. In D. Schuurmans M. P. Wellman (eds.), AAAI (p./pp. 3560-3566), : AAAI Press. ISBN: 978-1-57735-760-5