Report for Deep Learning-week2

11712617

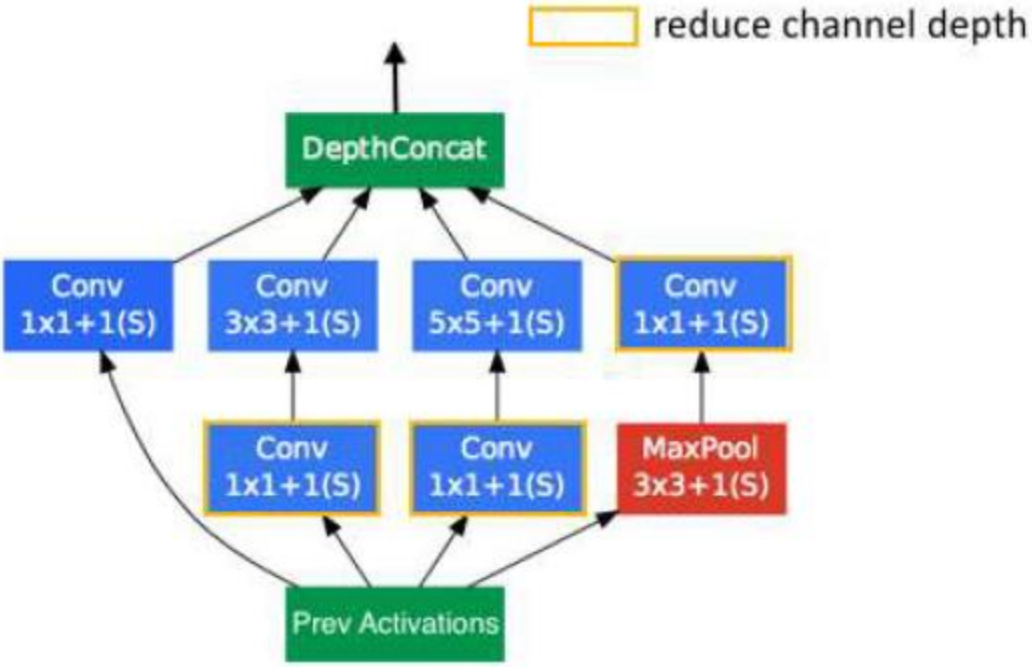
向昕昊

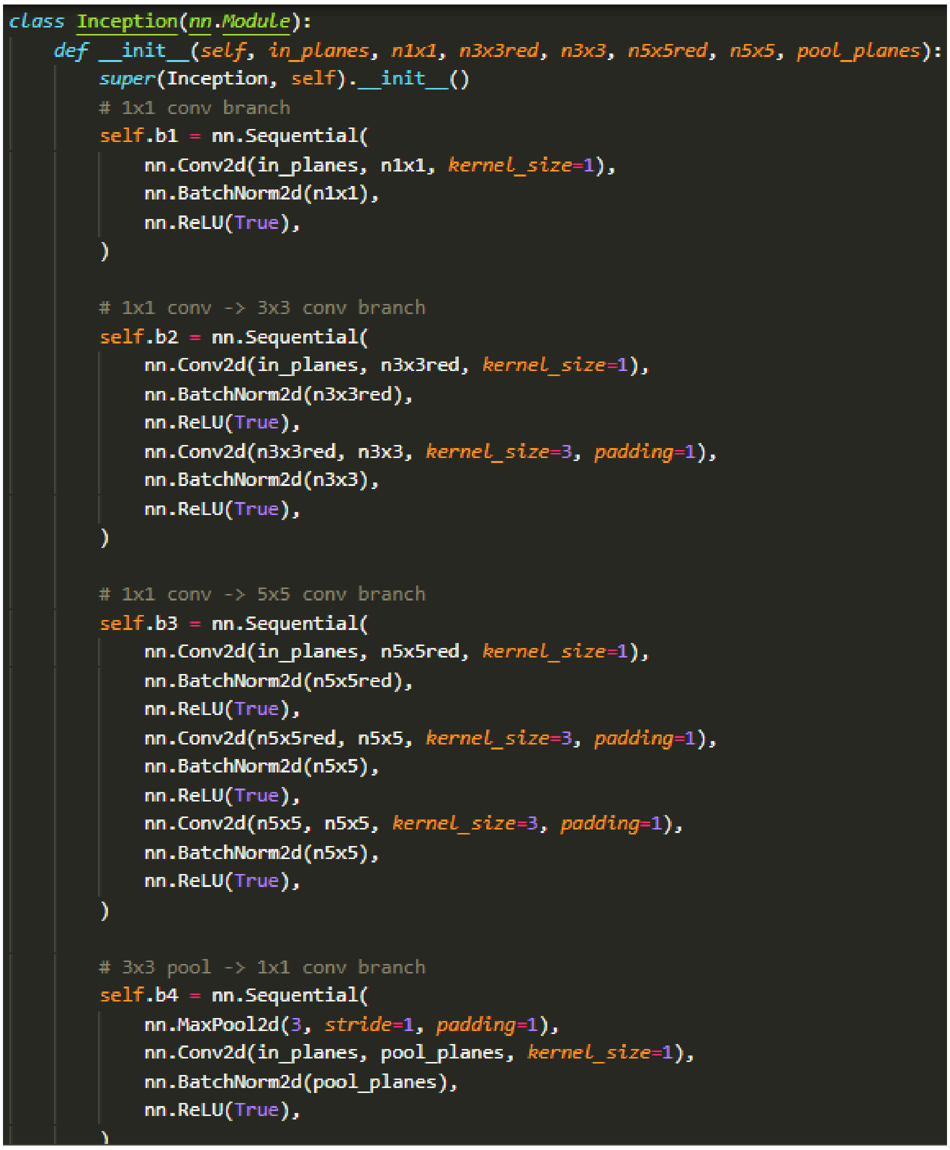
This week is the last week of the project. I have successfully implemented a GooLeNet and use it to train a model base on the dataset supplied. Generally speaking, it performs not bad.

In the project, I am asked to complete a binary classification to judge if a given Melanoma by a picture is benign or malignant. By the time I did presentation last week, I had decided to implement a VGG network. I indeed did it. When I trained it for the first time, however, the best accuracy within 200 epochs is only 73%. It seems not to be a suitable network to do the tasks. Therefore, I consider changing the network.

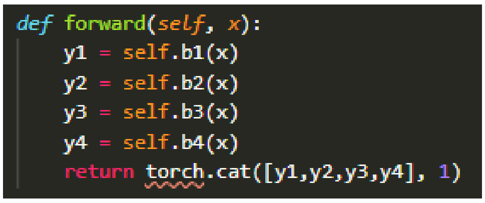
Through the history of networks which used to do image classification, I see that GooLeNet is a network invented almost at the same time with VGG. Instead of making the network longer, I am considering make it wider. Therefore, I chose GooLeNet to do this task.

The GooLeGet is assembled by a number of inception networks. I first assembled the inception network based on this model:



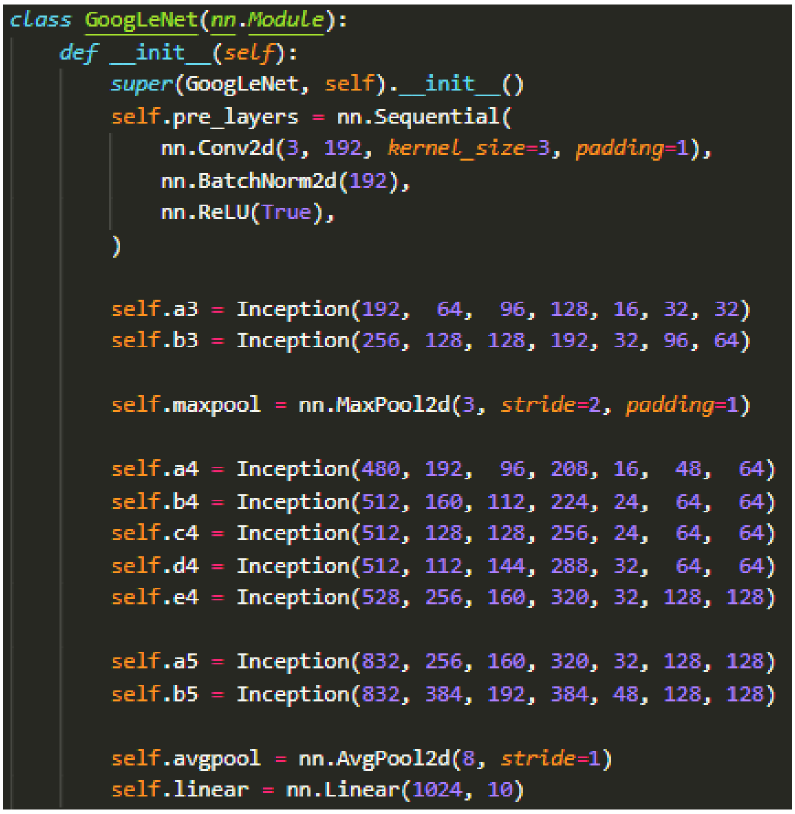


```python
class Inception(nn.Module):
    def __init__(self, in_planes, n1x1, n3x3red, n3x3, n5x5red, n5x5, pool_planes):
        super(Inception, self).__init__()
        # 1x1 conv branch
        self.b1 = nn.Sequential(
            nn.Conv2d(in_planes, n1x1, kernel_size=1),
            nn.BatchNorm2d(n1x1),
            nn.ReLU(True),
        )

        # 1x1 conv -> 3x3 conv branch
        self.b2 = nn.Sequential(
            nn.Conv2d(in_planes, n3x3red, kernel_size=1),
            nn.BatchNorm2d(n3x3red),
            nn.ReLU(True),
            nn.Conv2d(n3x3red, n3x3, kernel_size=3, padding=1),
            nn.BatchNorm2d(n3x3),
            nn.ReLU(True),
        )

        # 1x1 conv -> 5x5 conv branch
        self.b3 = nn.Sequential(
            nn.Conv2d(in_planes, n5x5red, kernel_size=1),
            nn.BatchNorm2d(n5x5red),
            nn.ReLU(True),
            nn.Conv2d(n5x5red, n5x5, kernel_size=3, padding=1),
            nn.BatchNorm2d(n5x5),
            nn.ReLU(True),
            nn.Conv2d(n5x5, n5x5, kernel_size=3, padding=1),
            nn.BatchNorm2d(n5x5),
            nn.ReLU(True),
        )

        # 3x3 pool -> 1x1 conv branch
        self.b4 = nn.Sequential(
            nn.MaxPool2d(3, stride=1, padding=1),
            nn.Conv2d(in_planes, pool_planes, kernel_size=1),
            nn.BatchNorm2d(pool_planes),
            nn.ReLU(True),
        )
```

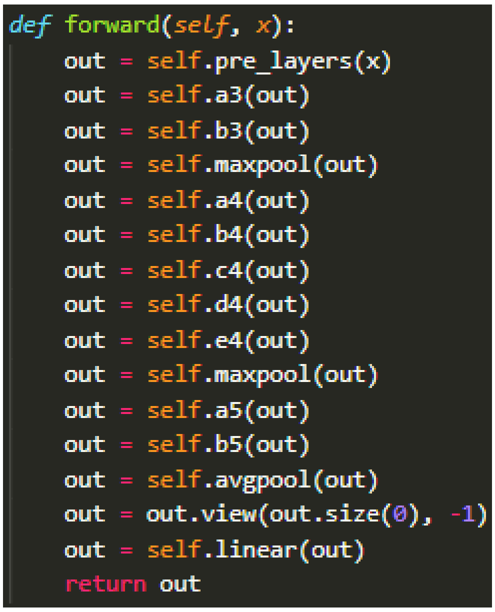


```python
def forward(self, x):
    y1 = self.b1(x)
    y2 = self.b2(x)
    y3 = self.b3(x)
    y4 = self.b4(x)
    return torch.cat([y1,y2,y3,y4], 1)
```

Then I cluster these inception networks to connect to the final GooLeNet.



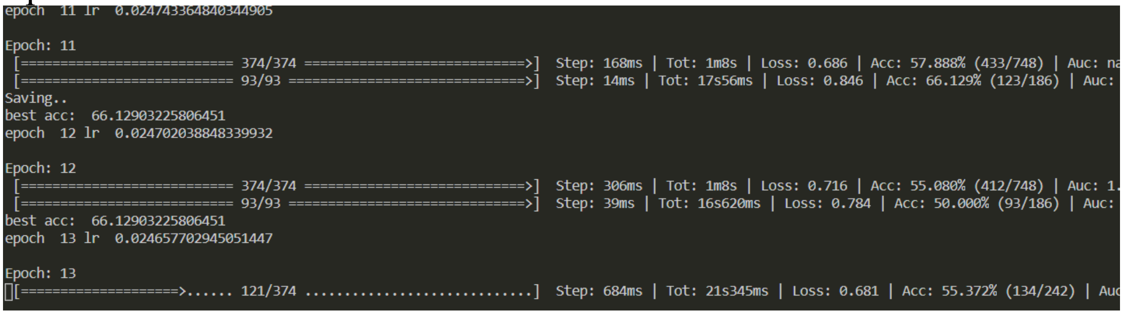
```
class GoogLeNet(nn.Module):
    def __init__(self):
        super(GoogLeNet, self).__init__()
        self.pre_layers = nn.Sequential(
            nn.Conv2d(3, 192, kernel_size=3, padding=1),
            nn.BatchNorm2d(192),
            nn.ReLU(True),
        )

        self.a3 = Inception(192,  64,  96, 128, 16, 32, 32)
        self.b3 = Inception(256, 128, 128, 192, 32, 96, 64)

        self.maxpool = nn.MaxPool2d(3, stride=2, padding=1)

        self.a4 = Inception(480, 192,  96, 208, 16,  48,  64)
        self.b4 = Inception(512, 160, 112, 224, 24,  64,  64)
        self.c4 = Inception(512, 128, 128, 256, 24,  64,  64)
        self.d4 = Inception(512, 112, 144, 288, 32,  64,  64)
        self.e4 = Inception(528, 256, 160, 320, 32, 128, 128)

        self.a5 = Inception(832, 256, 160, 320, 32, 128, 128)
        self.b5 = Inception(832, 384, 192, 384, 48, 128, 128)

        self.avgpool = nn.AvgPool2d(8, stride=1)
        self.linear = nn.Linear(1024, 10)
```



```
def forward(self, x):
    out = self.pre_layers(x)
    out = self.a3(out)
    out = self.b3(out)
    out = self.maxpool(out)
    out = self.a4(out)
    out = self.b4(out)
    out = self.c4(out)
    out = self.d4(out)
    out = self.e4(out)
    out = self.maxpool(out)
    out = self.a5(out)
    out = self.b5(out)
    out = self.avgpool(out)
    out = out.view(out.size(0), -1)
    out = self.linear(out)
    return out
```

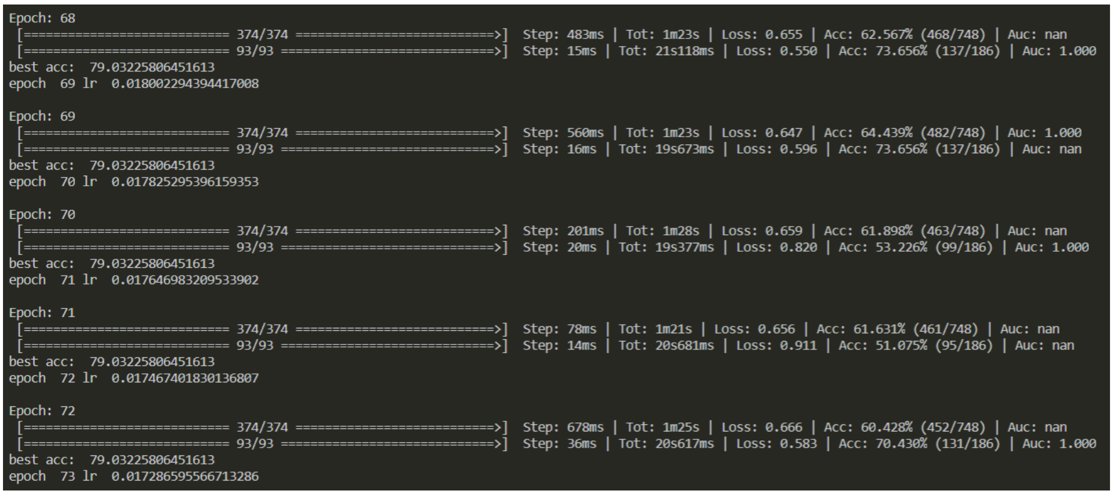
After implementing the GooLeNet, I started to train the model based on the dataset provided. After some experiment, I finally set the parameter like this: the learning rate equals 0.025, the momentum equals 0.9, the weight dacay equals 3e-4, the total epochs equal 200 and the batch size equals 2. Under this setting, the overall performance can be relatively high. The best accuracy is around 79%- 80% within 200 epochs. Here are some screenshots during the training:

Epoch 11-12:



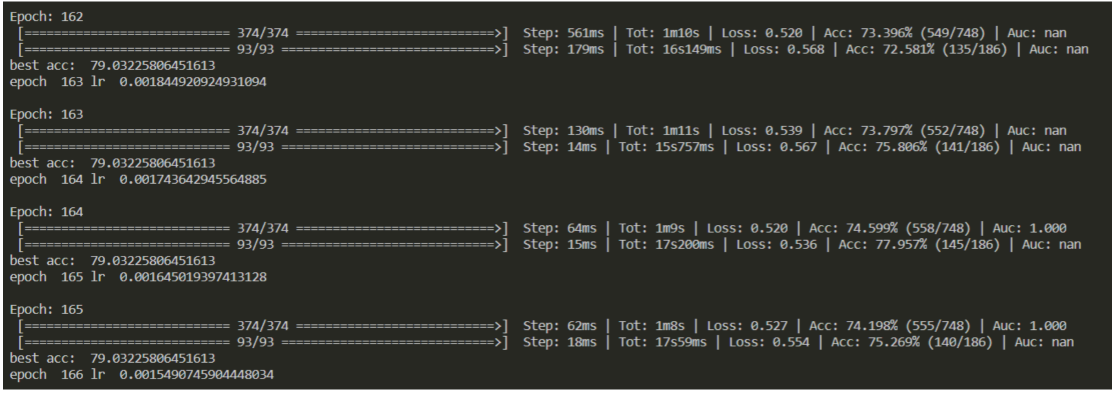
```
epoch  11 lr  0.024743364840344905

Epoch: 11
 [========================= 374/374 ===========================>]  Step: 168ms | Tot: 1m8s | Loss: 0.686 | Acc: 57.888% (433/748) | Auc: na
 [========================= 93/93 =============================>]  Step: 14ms | Tot: 17s56ms | Loss: 0.846 | Acc: 66.129% (123/186) | Auc:
Saving..
best acc:  66.12903225806451
epoch  12 lr  0.024702038848339932

Epoch: 12
 [========================= 374/374 ===========================>]  Step: 306ms | Tot: 1m8s | Loss: 0.716 | Acc: 55.080% (412/748) | Auc: 1.
 [========================= 93/93 =============================>]  Step: 39ms | Tot: 16s620ms | Loss: 0.784 | Acc: 50.000% (93/186) | Auc:
best acc:  66.12903225806451
epoch  13 lr  0.024657702945051447

Epoch: 13
▯[===================>...... 121/374 ............................]  Step: 684ms | Tot: 21s345ms | Loss: 0.681 | Acc: 55.372% (134/242) | Auc
```

Epoch 68-72:



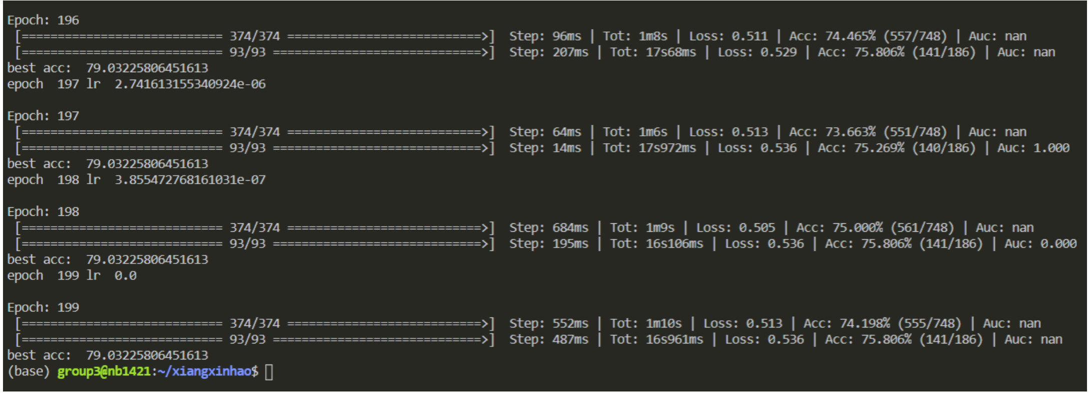
```
Epoch: 68
 [========================== 374/374 ==========================>]  Step: 483ms | Tot: 1m23s | Loss: 0.655 | Acc: 62.567% (468/748) | Auc: nan
 [========================== 93/93 ============================>]  Step: 15ms | Tot: 21s118ms | Loss: 0.550 | Acc: 73.656% (137/186) | Auc: 1.000
best acc:  79.03225806451613
epoch  69 lr  0.018002294394417008

Epoch: 69
 [========================== 374/374 ==========================>]  Step: 560ms | Tot: 1m23s | Loss: 0.647 | Acc: 64.439% (482/748) | Auc: 1.000
 [========================== 93/93 ============================>]  Step: 16ms | Tot: 19s673ms | Loss: 0.596 | Acc: 73.656% (137/186) | Auc: nan
best acc:  79.03225806451613
epoch  70 lr  0.017825295396159353

Epoch: 70
 [========================== 374/374 ==========================>]  Step: 201ms | Tot: 1m28s | Loss: 0.659 | Acc: 61.898% (463/748) | Auc: nan
 [========================== 93/93 ============================>]  Step: 20ms | Tot: 19s377ms | Loss: 0.820 | Acc: 53.226% (99/186) | Auc: 1.000
best acc:  79.03225806451613
epoch  71 lr  0.017646983209533902

Epoch: 71
 [========================== 374/374 ==========================>]  Step: 78ms | Tot: 1m21s | Loss: 0.656 | Acc: 61.631% (461/748) | Auc: nan
 [========================== 93/93 ============================>]  Step: 14ms | Tot: 20s681ms | Loss: 0.911 | Acc: 51.075% (95/186) | Auc: nan
best acc:  79.03225806451613
epoch  72 lr  0.017467401830136807

Epoch: 72
 [========================== 374/374 ==========================>]  Step: 678ms | Tot: 1m25s | Loss: 0.666 | Acc: 60.428% (452/748) | Auc: nan
 [========================== 93/93 ============================>]  Step: 36ms | Tot: 20s617ms | Loss: 0.583 | Acc: 70.430% (131/186) | Auc: 1.000
best acc:  79.03225806451613
epoch  73 lr  0.017286595566713286
```

Epoch 162-165:



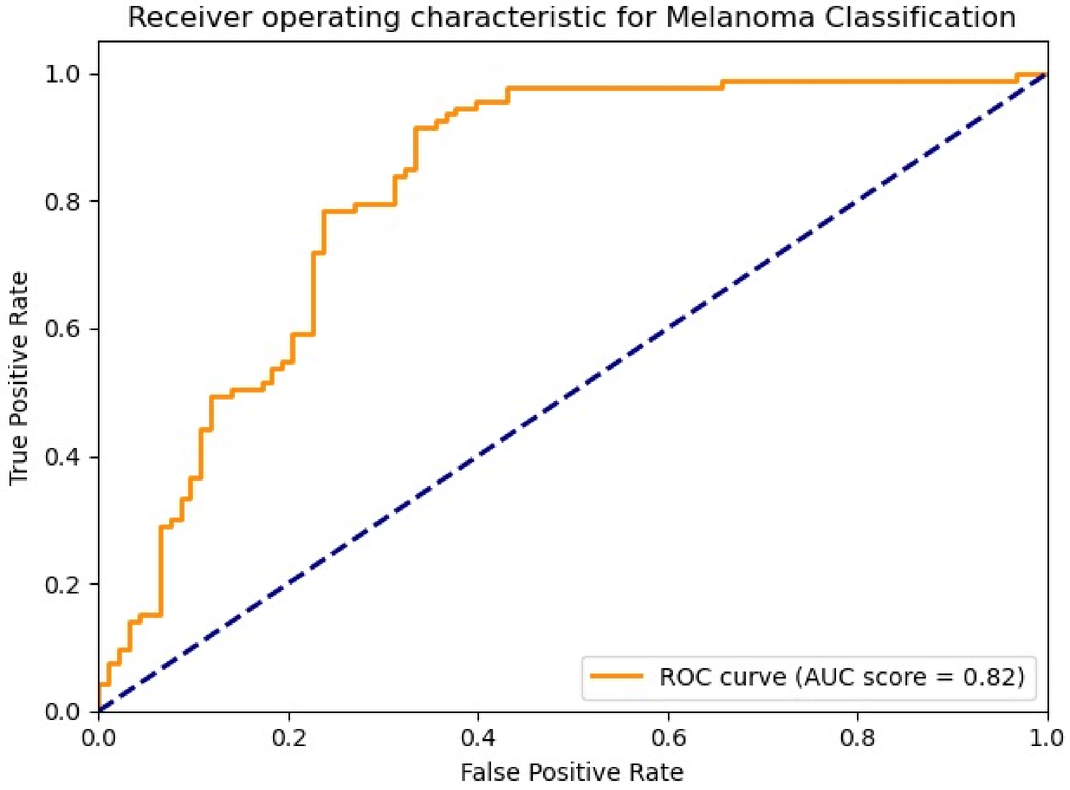
```
Epoch: 162
 [========================== 374/374 ==========================>]  Step: 561ms | Tot: 1m10s | Loss: 0.520 | Acc: 73.396% (549/748) | Auc: nan
 [========================== 93/93 ============================>]  Step: 179ms | Tot: 16s149ms | Loss: 0.568 | Acc: 72.581% (135/186) | Auc: nan
best acc:  79.03225806451613
epoch  163 lr  0.001844920924931094

Epoch: 163
 [========================== 374/374 ==========================>]  Step: 130ms | Tot: 1m11s | Loss: 0.539 | Acc: 73.797% (552/748) | Auc: nan
 [========================== 93/93 ============================>]  Step: 14ms | Tot: 15s757ms | Loss: 0.567 | Acc: 75.806% (141/186) | Auc: nan
best acc:  79.03225806451613
epoch  164 lr  0.001743642945564885

Epoch: 164
 [========================== 374/374 ==========================>]  Step: 64ms | Tot: 1m9s | Loss: 0.520 | Acc: 74.599% (558/748) | Auc: 1.000
 [========================== 93/93 ============================>]  Step: 15ms | Tot: 17s200ms | Loss: 0.536 | Acc: 77.957% (145/186) | Auc: nan
best acc:  79.03225806451613
epoch  165 lr  0.001645019397413128

Epoch: 165
 [========================== 374/374 ==========================>]  Step: 62ms | Tot: 1m8s | Loss: 0.527 | Acc: 74.198% (555/748) | Auc: 1.000
 [========================== 93/93 ============================>]  Step: 18ms | Tot: 17s59ms | Loss: 0.554 | Acc: 75.269% (140/186) | Auc: nan
best acc:  79.03225806451613
epoch  166 lr  0.0015490745904448034
```

Epoch 196-199:



```
Epoch: 196
 [========================== 374/374 ==========================>]  Step: 96ms | Tot: 1m8s | Loss: 0.511 | Acc: 74.465% (557/748) | Auc: nan
 [========================== 93/93 ============================>]  Step: 207ms | Tot: 17s68ms | Loss: 0.529 | Acc: 75.806% (141/186) | Auc: nan
best acc:  79.03225806451613
epoch  197 lr  2.741613155340924e-06

Epoch: 197
 [========================== 374/374 ==========================>]  Step: 64ms | Tot: 1m6s | Loss: 0.513 | Acc: 73.663% (551/748) | Auc: nan
 [========================== 93/93 ============================>]  Step: 14ms | Tot: 17s972ms | Loss: 0.536 | Acc: 75.269% (140/186) | Auc: 1.000
best acc:  79.03225806451613
epoch  198 lr  3.855472768161031e-07

Epoch: 198
 [========================== 374/374 ==========================>]  Step: 684ms | Tot: 1m9s | Loss: 0.505 | Acc: 75.000% (561/748) | Auc: nan
 [========================== 93/93 ============================>]  Step: 195ms | Tot: 16s106ms | Loss: 0.536 | Acc: 75.806% (141/186) | Auc: 0.000
best acc:  79.03225806451613
epoch  199 lr  0.0

Epoch: 199
 [========================== 374/374 ==========================>]  Step: 552ms | Tot: 1m10s | Loss: 0.513 | Acc: 74.198% (555/748) | Auc: nan
 [========================== 93/93 ============================>]  Step: 487ms | Tot: 16s961ms | Loss: 0.536 | Acc: 75.806% (141/186) | Auc: nan
best acc:  79.03225806451613
(base) group3@nb1421:~/xiangxinhao$ ▯
```

And the ROC curve is like this:



The training process is not propitious at all. GooLeNet is such a large model that I usually encountered the runtime error which shows “CUDA out of memory”. Last Friday, I attempted to deploy my code on the server to train. The training is not success until Monday this week, even if I have set the argument “batch size” to 2. On Monday, I run my code on one of my friend’s server. Only then have I trained the model. The training spends more than 5 hours, which is also relatively time-consuming.

After training. I shared my implemented GooLeNet and the trained .pth file to my groupmates. We combined each one’s model to generate the final model version of our group. It shows that after adding my model, the overall accuracy is improved by round 0.6 percent.

All in all, this is a meaningful project. I have learnt a lot and enjoy in it.