Hw4

清大光電所 **112066701** 王柏涵

6. Supplementary

I actually also have analyzed the VGG model but the result turns out that the model is basically too large and is not suitable for our task, which is relatively simple. The structure is as follows:

Vgg16 (13 Conv + 3 FC) Model (Each Conv2d is concatenated with **ReLU**):

Layer		Activation Shape	(in, out)
0	Input	(256, 256, 1)	
1	Conv2d (f=3, s=1, p=1)	(256,256,64)	(1, 64)
2	Conv2d (f=3, s=1, p=1)	(256,256,64)	(64, 64)
3	MaxPool2d (f=2, s=2, p=0)	(128, 128, 64)	(64, 64)
4	Conv2d (f=3, s=1, p=1)	(128, 128, 128)	(64, 128)
5	Conv2d (f=3, s=1, p=1)	(128, 128, 128)	(128, 128)
6	MaxPool2d (f=2, s=2, p=0)	(64, 64, 128)	(128, 128)
7-9	Conv2d (f=3, s=1, p=1) * 3	(64, 64, 256)	(128, 256)
10	MaxPool2d (f=2, s=2, p=0)	(32, 32, 256)	(256, 256)
11-13	Conv2d (f=3, s=1, p=1) * 3	(32, 32, 512)	(256, 512)
14	MaxPool2d (f=2, s=2, p=0)	(16, 16, 512)	(512, 512)
15-17	Conv2d (f=3, s=1, p=1)	(16, 16, 512)	(512, 512)
18	MaxPool2d (f=2, s=1, p=0)	(8, 8, 512)	(512, 512)
19	AvgPool2d (output size=7x7)	(7, 7, 512)	(512, 512)
20	Linear (FC, fully connected)	in_feat=25088, out_feat=4096	
21	Linear (FC)	in_feat=4096, out_feat=4096	
22	Linear (FC)	in_feat=4096, out_feat=1000	

I changed the Last layer (22) to make it suitable for our task:

22	Linear (FC)	in_feat=4096, out_feat=64
23	Linear (FC)	In_feat=64, out_feat=1

And apply a **sigmoid function** to diagnose the patient's syndrome.

The hyperparameter:

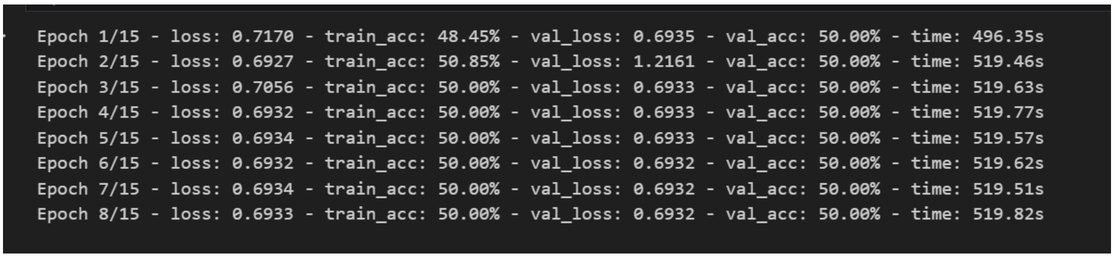
Learning rate	0.001
Weight decay	0.0001

Epochs	15
Optimizer	Adam
Loss function	BCE loss

The pretrain result with freezing all layers except last layer is fine:

	Vgg16
Train Accuracy	92.55%
Train Loss	0.2024
Val Accuracy	68.75%
Val Loss	0.6306
Test Accuracy	79.11%
Test Loss	0.4177
Computation time	500s

But by unfreezing all layers, the performance significantly drops and consume A LOT of time.



```
Epoch 1/15 - loss: 0.7170 - train_acc: 48.45% - val_loss: 0.6935 - val_acc: 50.00% - time: 496.35s
Epoch 2/15 - loss: 0.6927 - train_acc: 50.85% - val_loss: 1.2161 - val_acc: 50.00% - time: 519.46s
Epoch 3/15 - loss: 0.7056 - train_acc: 50.00% - val_loss: 0.6933 - val_acc: 50.00% - time: 519.63s
Epoch 4/15 - loss: 0.6932 - train_acc: 50.00% - val_loss: 0.6933 - val_acc: 50.00% - time: 519.77s
Epoch 5/15 - loss: 0.6934 - train_acc: 50.00% - val_loss: 0.6933 - val_acc: 50.00% - time: 519.57s
Epoch 6/15 - loss: 0.6932 - train_acc: 50.00% - val_loss: 0.6932 - val_acc: 50.00% - time: 519.62s
Epoch 7/15 - loss: 0.6934 - train_acc: 50.00% - val_loss: 0.6932 - val_acc: 50.00% - time: 519.51s
Epoch 8/15 - loss: 0.6933 - train_acc: 50.00% - val_loss: 0.6932 - val_acc: 50.00% - time: 519.82s
```

Therefore, I switched to EfficientNet, which is claimed to be not that complex and is suitable for our task.