Hw4

清大光電所 **112066701** 王柏涵

Github: git@github.com:icesplendent/NTHU_2024_DLBOI_HW.git

1. **Selected Model: EfficientNet_b0 & ResNet18**

EfficientNet Model with b0 weight with **SiLu** activation function

(Skipping the squeezeExcitation layer)

Block	Layer		Activation Shape
	0	Input	(256, 256, 1)
	1	Conv2d (f=3, s=2, p=1)	(128, 128, 32)
1	2	Conv2d (f=3, s=1, p=1)	(128, 128, 32)
	3	Conv2d (f=1, s=1, p=0)	(128, 128, 16)
2	4	Conv2d (f=1, s=1, p=0)	(128, 128, 96)
	5	Conv2d (f=3, s=2, p=1)	(64, 64, 96)
	6	Conv2d (f=1, s=1, p=0)	(64, 64, 24)
	7	Conv2d (f=1, s=1, p=0)	(64, 64, 144)
	8	Conv2d (f=3, s=1, p=1)	(64, 64, 144)
	9	Conv2d (f=1, s=1, p=0)	(64, 64, 24)
3-7	Similar structure as block 2		
	49	Conv2d (f=1, s=1, p=0)	(, , 1280)
	50	AvgPool2d (outputsize=1)	(1, 1, 1280)
	51	Linear (FC, fully connected)	in_feat=1280, out_feat=1000

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

51	Linear (FC)	in_feat=1280, out_feat=64
52	Linear (FC)	In_feat=64, out_feat=1

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

The hyperparameter used for EfficientNet:

Learning rate	0.001
Weight decay	0.0001
Epochs	15
Optimizer	Adam
Loss function	BCE loss
Dropout for the output sequential NN	0.5

ResNet18 (18 layers with weight) Model

(Activation function: **ReLU**):

	Stage	Layer		Activation Shape
	0	0	Input	(256, 256, 1)
		1	Conv2d (f=7, s=2, p=2)	(127, 127, 64)
		2	MaxPool2d (f=3, s=2, p=1)	(64, 64, 64)
	1	3-6	Conv2d (f=3, s=1, p=1)	(64, 64, 64)
	2	7	Conv2d (f=3, s=2, p=1)	(32, 32, 128)
		8	Conv2d (f=3, s=1, p=1)	(32, 32, 128)
downsample	downsample		Conv2d (f=1, s=2, p=0)	(32, 32, 128)
	2	9	Conv2d (f=3, s=1, p=1)	(32, 32, 128)
		10	Conv2d (f=3, s=1, p=1)	(32, 32, 128)
	3	11	Conv2d (f=3, s=2, p=1)	(16, 16, 256)
		12	Conv2d (f=3, s=1, p=1)	(16, 16, 256)
downsample	downsample		Conv2d (f=1, s=2, p=0)	(16, 16, 256)
	3	13-14	Conv2d (f=3, s=1, p=1)	(16, 16, 256)
	4	15	Conv2d (f=3, s=2, p=1)	(8, 8, 512)
		16	Conv2d (f=3, s=1, p=1)	(8, 8, 512)
downsample	downsample		Conv2d (f=1, s=2, p=0)	(8, 8, 512)
	4	17-18	Conv2d (f=3, s=1, p=1)	(8, 8, 512)
	5	19	AvgPool2d (output size 1x1)	(1, 1, 512)
	6	20	Linear (FC, fully connected)	in_feat=512, out_feat=1000

I changed the Last layer (20) to make it suitable for our task:

20	Linear (FC)	in_feat=512, 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 used for ResNet18:

Learning rate	0.001
Weight decay	0.0001
Epochs	15
Optimizer	Adam
Loss function	BCE loss
Dropout for the output sequential NN	0.5

Pre-Train Result (NOT fine-tuned yet)

	EfficientNet	ResNet18
Train Accuracy	97.05%	96.4%
Train Loss	0.0826	0.1089
Val Accuracy	68.75%	62.5%
Val Loss	0.6357	0.9404
Test Accuracy	84.22%	80.36%
Test Loss	0.369	50.4644
Computation time	500s	500s

The computation time for both models is the same. This doesn't mean they are both as efficient as each other. It is because I have frozen all layers except the last layer to see the performance of pre-trained model, which means the other part of layer doesn't consume any resource.

Both models show a decent result for training accuracy, which indicates they are a good candidate for our task. However, the validation loss and the test accuracy imply further variance techniques such as **regularization should be applied (Task C)**.

2. Task B: Fine-tuning the ConvNet

By Unfreezing all the layer, the following are the performance:

	Efficient	ResNet18
Train Accuracy	99.8%	99.55%
Train Loss	0.0053	0.0121
Val Accuracy	75%	100%
Val Loss	0.8019	0.0176
Test Accuracy	87.66%	87.81%
Test Loss	0.4930	0.5567
Computation time	500s	500s

By training all the parameters inside the model. The test accuracy has significantly improved for both models. This is the expected result since we have more parameters used for tuning. This also further proves that both models are good candidates for our task.

It should also be noted that the validation accuracy doesn't seems good is just because the number of validation data set is small. Miss classification of one image can significantly decrease the rate. The tendency shown in the result oscillates between values higher than 70%, which I believe is ideal for the amount of validation set we have. **Overall, the test accuracy improved proves the model can be fitted more perfectly with more parameters tuned with back propagation.**

3. Task C: ConvNet as Fixed Feature Extractor

From Section one, the variance is large since the validation loss is relatively high. Therefore, to see better performance, regularization should be fine-tuned. I primarily focus on the weight decay, and the following tables are the result.

By choosing a suitable weight decay value, the test accuracy can indeed be improved (EfficientNet: 84.22% -> 86.15%, ResNet18:80.36% -> 85.05%).

Note: lr: Learning rate. wd: weight decay

Lr	0.0001	wd	0.0005	epochs	15
Dropout	0.5	optimizer	Adam		

	EfficientNet	ResNet18
Train Accuracy	96.65%	95.85%
Train Loss	0.0987	0.5499
Val Accuracy	87.5%	68.75%
Val Loss	0.2674	0.5499
Test Accuracy	80.83%	**87.71% (> 80.36%)**
Test Loss	0.4754	0.3171
Computation time	500s	500s

lr	0.0001	wd	0.0002	epochs	15
Dropout	0.5	optimizer	Adam		

	EfficientNet	ResNet18

Train Accuracy	96.20%	97.4%
Train Loss	0.1069	0.0796
Val Accuracy	81.25%	75%
Val Loss	0.3925	0.7620
Test Accuracy	**86.15% (> 84.22%)**	85.05% (> 80.36%)
Test Loss	0.3329	0.3433
Computation time	500s	500s

Intuitively, **different models have their own suitable hyper parameter.** This is further proven with the above two results. With different setups, different models have their own best value respectively. For EfficientNet, weightdecay = 0.0002 is optimized, For ResNet18, weightdecay = 0.0005 is better.

4. Task D: Comparison and Analysis

To further conduct the comparison, I adopt the best parameter set in Section 3 (Task C) to Section 2 (Task B) to see if fully tuned would be better than fixed feature.

lr	0.0001	epochs	15
Dropout	0.5	optimizer	Adam

	EfficientNet (wd=0.0002)	ResNet18 (wd = 0.0005)
Train Accuracy	100%	99.95%
Train Loss	0.0021	0.0032
Val Accuracy	100%	87.5%
Val Loss	0.0952	0.1694
Test Accuracy	**84.06% < 86.15%**	**90.08% (> 87.71%)**
Test Loss	0.8016	0.2612
Computation time	500s	500s

The above result shows a different tendency. For EfficientNet, it has become worse, but it improves for ResNet18. This indicates that **fixed feature and full tuning has different suitable hyperparameters, requiring further tuning afterwards if necessary**.

FYI, EfficientNet can be improved to have test accuracy of 89.02% in my case.

Besides, all the training performed have similar computation time. I infer this is because **all the tasks don't exceed my computation resource**. In fact, by using VGG18 model without freezing, it would take 500s just an epoch.

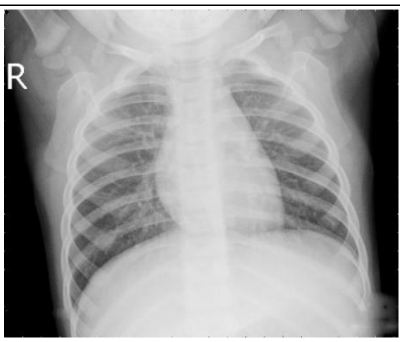
5. <u>Task E: Test Dataset Analysis</u>

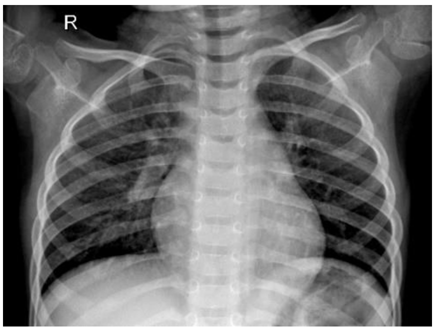
The Test accuracy is hard to exceed 90% due to the data quality itself. By checking each image one by one in person, I found that each image is slightly different from others in terms of angle, the actual range taken by the camera. The letter "R" and "L" also place at some different places for different images, adding some unwanted features for each image. These unwanted features will be taken into consideration and therefore reduce the accuracy for the model.

I have though about cropping the image to get ride of the "letter" factor. But some letter is actually overlapped with the lung and therefore it is not feasible to directly crop the image before reading it to GPU.

I also perform some random rotation about 10 degrees before processing, but the accuracy doesn't improve.

In short, the data quality (angle, photo range, letter added) may have limited accuracy. But overall, achieving accuracy above 85% is feasible with suitable models and corresponding parameters.





The image extracted from the data set. The angle is different. The range shot by camera is slightly different. The letter is placed at different places. These are the factors that distract the model, reducing the accuracy.