Report by Mingcheng Xu

1.

For both task1 and task2, the architectures are the same: **an untrainable vgg16 followed by 2 convolutional layers, 1 flatten layer, 1 dropout layer, 1 dense layer (for feature extraction, named "dense_feature"), 1 dropout layer, and 1 final output layer (dense layer).**

All convolutional layers have a filter size of 256, a kernel size of (3, 3) and padding. Activation function is "relu".

I **do not** add a pooling layer because I do not want to further reduce the data.

Each dropout layer has a dropout rate of 0.25.

Layer "dense_feature" has a dimension of 256 for task1 and 512 for task2. I increase the dimension in task2 because it has to classify more classes. Activation function is "relu".

The output layer has a dimension of 1 and activation function "sigmoid" for task1. For task2, it has a dimension of 4 and activation function "softmax".

2.

For both tasks, the optimizer uses the Adam algorithm. Loss function is "binary cross entropy" for task1 and "categorical cross entropy" for task2. Metrics is "accuracy". Initial learning rate is 0.001. Decay step is 10000. Decay rate is 0.9 for task1 and 0.96 for task2.

For regularization, I use dropout and a decaying learning rate for both tasks.

For task2, I also use early stopping based on validation accuracy. Because sometimes a very early epoch may have very high validation accuracy by coincidence, and in fact the model is insufficiently trained at that time, I did not use the weights with the highest validation accuracy; Instead, I make the training process stop when there is no improve in validation after 50 epochs. In most cases, it will train 100 epochs.

All other parameters I use are the default values given in the templates.

3. different architectures for the second task

First, I experiment not using a pretrained model. Below is a summary of that model:

Model: "task1"

Layer (type)	Output Shape	Param #
c_1 (Conv2D)	(None, 224, 224, 16)	448
c_2 (Conv2D)	(None, 224, 224, 16)	2320
p_1 (AveragePooling2D)	(None, 112, 112, 16)	0
c_4 (Conv2D)	(None, 112, 112, 32)	4640
c_5 (Conv2D)	(None, 112, 112, 32)	9248
p_2 (AveragePooling2D)	(None, 56, 56, 32)	0
c_7 (Conv2D)	(None, 56, 56, 64)	18496
c_8 (Conv2D)	(None, 56, 56, 64)	36928
c_9 (Conv2D)	(None, 56, 56, 64)	36928
p_3 (MaxPooling2D)	(None, 28, 28, 64)	0
c_10 (Conv2D)	(None, 28, 28, 128)	73856
c_11 (Conv2D)	(None, 28, 28, 128)	147584
c_12 (Conv2D)	(None, 28, 28, 128)	147584
p_4 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten_1 (Flatten)	(None, 25088)	0
dropout_1 (Dropout)	(None, 25088)	0
dense_feature (Dense)	(None, 512)	12845568
dropout_2 (Dropout)	(None, 512)	0
dense_output (Dense)	(None, 4)	2052

Total params: 13,325,652
Trainable params: 13,325,652
Non-trainable params: 0

However, this architecture always stuck at some point with a validation accuracy that almost stayed the same. The same situation happened when tested on task1. The reason is the model itself is trapped at some local minimum (or “saddle point”). It finds a “short cut” to prevent losing accuracy by predicting all images to be normal or covid or whatever (depending on which category has the biggest amount in training dataset).

The same situation happens when I set VGG16 to be trainable. It again falls into local minimum and never gets out.

Therefore, setting VGG model untrainable greatly improves the accuracy. Since the model cannot change the weights in these convolutional layers, it avoids the local minimum by force. However, I still want my model to have some ability to alter resolution of the input images, so I add 2 convolutional layers. **Here is a summary of the architecture of my choice:**

```
Model: "task2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
conv_1 (Conv2D)              (None, 7, 7, 256)         1179904
_________________________________________________________________
conv_2 (Conv2D)              (None, 7, 7, 256)         590080
_________________________________________________________________
flatten_1 (Flatten)          (None, 12544)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12544)             0
_________________________________________________________________
dense_feature (Dense)        (None, 512)               6423040
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_output (Dense)         (None, 4)                 2052
=================================================================
Total params: 22,909,764
Trainable params: 8,195,076
Non-trainable params: 14,714,688
```

Adding the 2 convolutional layers seems to improve the performance of the model. I tried to remove the 2 convolutional layers and get a lower test accuracy. Below is the summary:

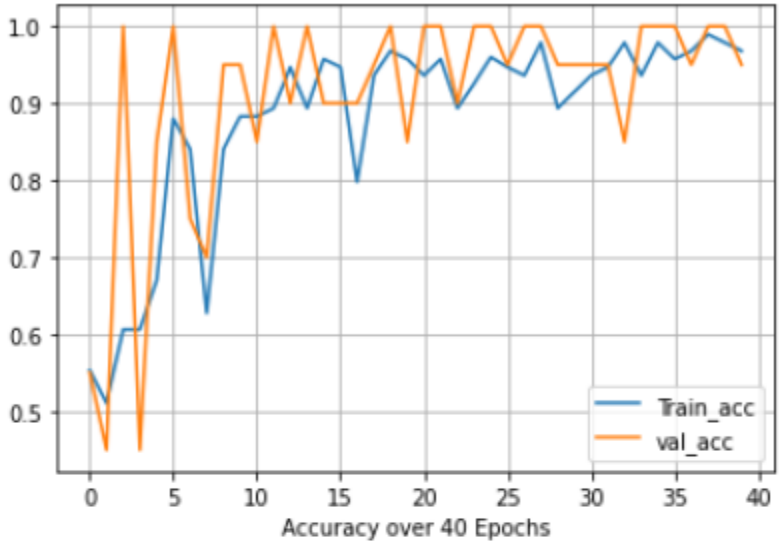
```
Model: "task2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 25088)             0
_________________________________________________________________
dense_feature (Dense)        (None, 512)               12845568
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_output (Dense)         (None, 4)                 2052
=================================================================
Total params: 27,562,308
Trainable params: 12,847,620
Non-trainable params: 14,714,688
```

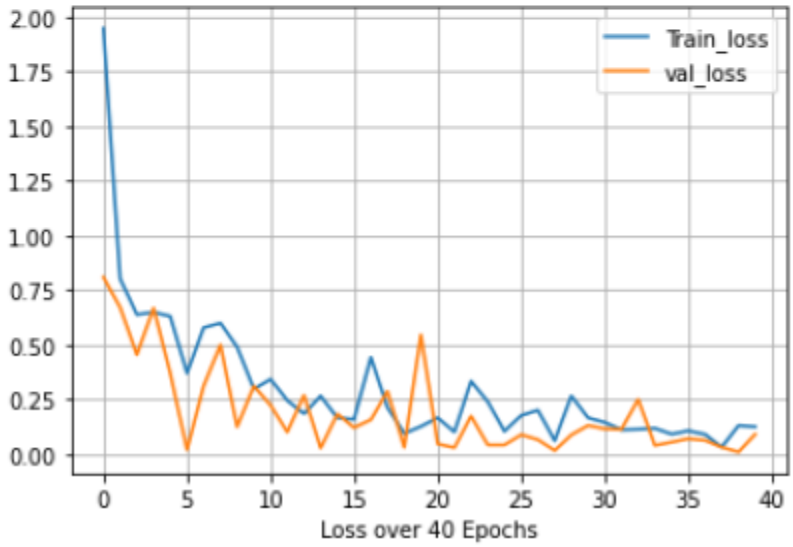
With 2 convolutional layers the test accuracy is 69.4% while without them the accuracy is around 61.1%.

Additionally, I tried different pre-trained models: VGG19 and MobileNet. For both of them, I did not add 2 convolutional layers. So, their structures and summaries are similar to the one above. For VGG19, the test accuracy is around 52.7%; for MobileNet, the test accuracy is around 58.3%. To conclude, VGG16 is the best pre-trained model.

4. the accuracy and the loss for both tasks

For task1:



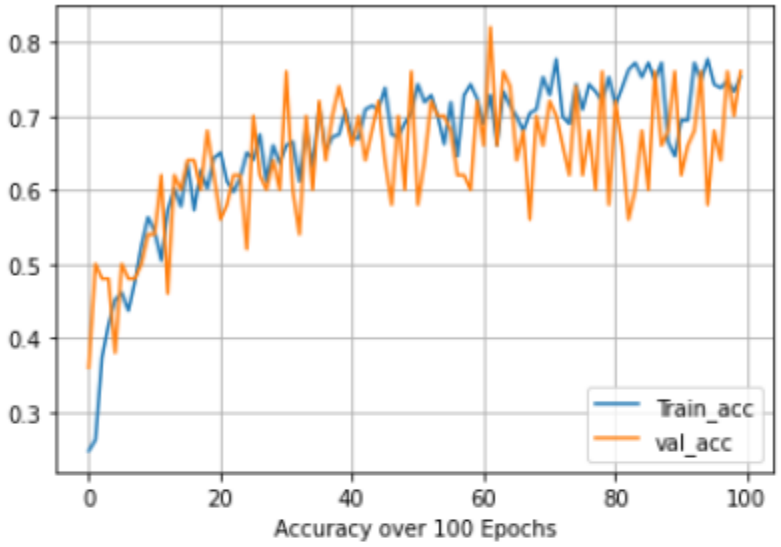


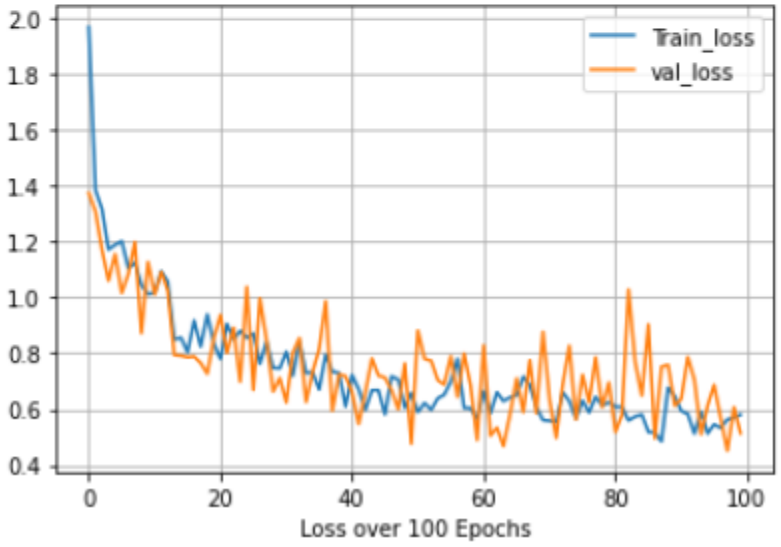
And for the test cases, it correctly predicts 18 out of 18, giving a test accuracy around 100%.

According to the graph, both training loss and validation loss tend to decrease during training (and accuracy tends to increase), though volatility exists.

In general, it is possible that the model is overfitting on training dataset or even the validation dataset, but since the test accuracy is also high, the probability of overfitting is low.

For task2:



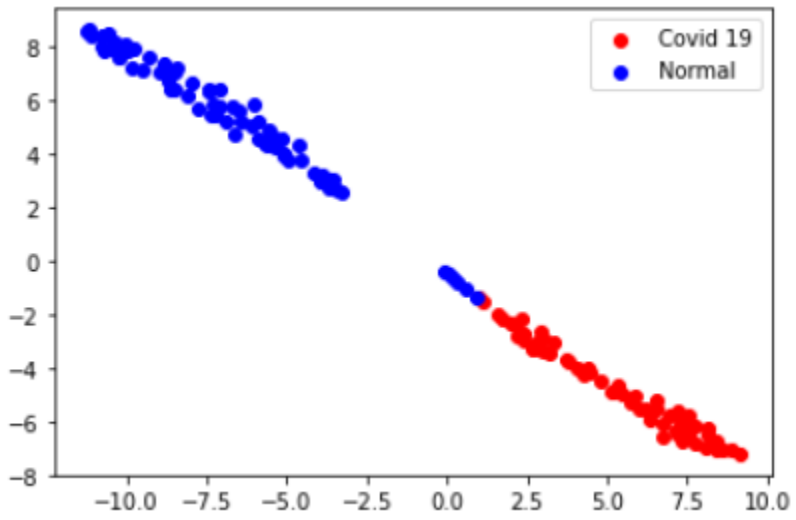



```
Found 36 images belonging to 4 classes.
36
WARNING:tensorflow:From <ipython-input-8-6e2e55ffea00>:8: Model.evaluate_generator (from tensorflow.python.ke
moved in a future version.
Instructions for updating:
Please use Model.evaluate, which supports generators.
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
36/36 [==============================] - 4s 98ms/step - loss: 1.4518 - accuracy: 0.6944
Test loss: 1.4517992918974616
Test accuracy: 0.6944444
```

The training and validation accuracy tend to increase and the loss tend to decrease, but the volatility becomes even larger. Since the training accuracy is not very high (compared to the test accuracy), I think the model is not overfitting.

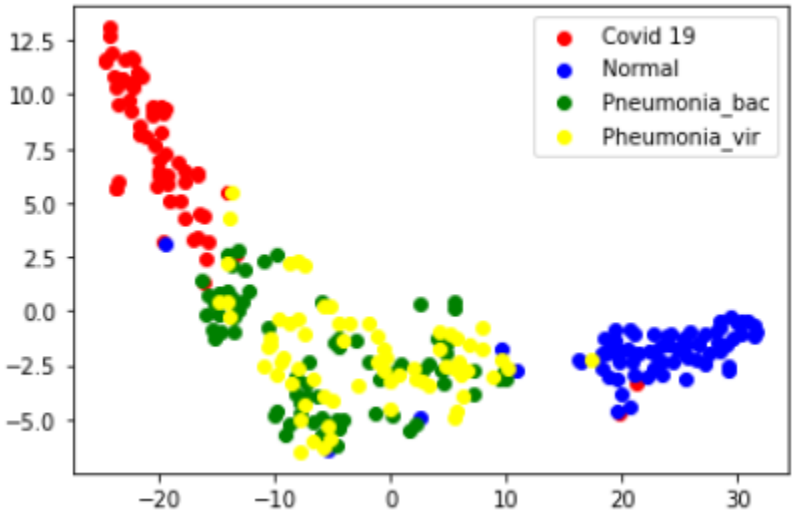
5. the t-SNE visualizations

For task1:



There are 2 clusters. So, the model has a good generalization. It is not perfect because some “normal” points are very close to “covid 19” points. This may indicate some kind of inaccuracy and misclassification. However, most of the points are correctly clustered and we can draw a diagonal line to classify them.

For task2:



The points for “covid 19” and “normal” are well clustered, while the points for two kinds of pneumonia have many overlaps. Therefore, this model will be good at finding covid 19 cases and normal cases, but may have difficulty distinguish between “pheumonia_bac” and “phenmonia_vir”.