

CS 542 Project Report

Task1: binary classification

Architecture

In this task, I use VGG16 architecture (16 layers in total) for the convolution part, then substitute the fully connection layers and classifier in VGG16 with customized layers and binary classifier.

Input: 224x224x3 RGB image, the image is normalized before feeding into neural network.

Convolution layer:

First and Second Layers: The input passes through first and second convolutional layers, each with 64 feature filters of size 3×3 and a stride of 1. The data dimensions changes to 224x224x64. Then the VGG16 applies maximum a stride of 2. The resulting image dimensions will be reduced to 112x112x64.

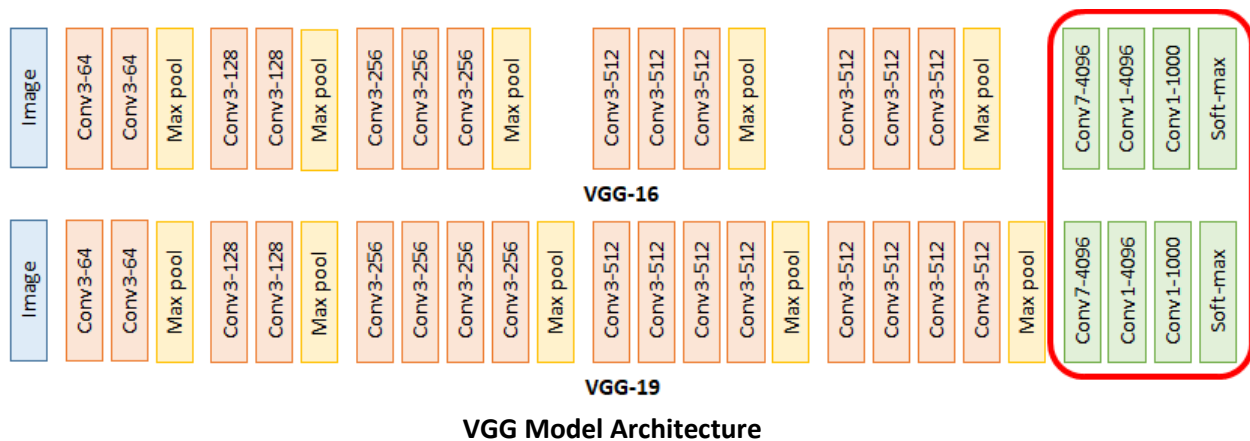
Third and Fourth Layer: Follow with two convolutional layers, each has 128 feature filters of size 3×3 and a stride of 1. Then the same max pooling, output data to 56x56x128.

Fifth and Seventh Layer: Follow with three convolutional layers, each has 256 feature filters of size 3×3 and a stride of 1. Then the same max pooling, output data to 28x28x256

Eighth to Thirteen Layer: Next are the two sets of 3 convolutional layers followed by a maximum pooling layer. All convolutional layers have 512 filters of size 3×3 and a stride of 1. The final output is reduced to 7x7x512.

Customized full connected layer: The convolutional layer output is flattened to 25088 features and feed into a dropout layer then a fully connected layer with 256 units.

Output layer (Classifier): Finally use a sigmoid output layer to perform binary classification.



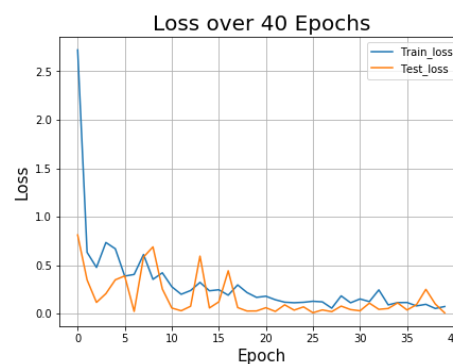
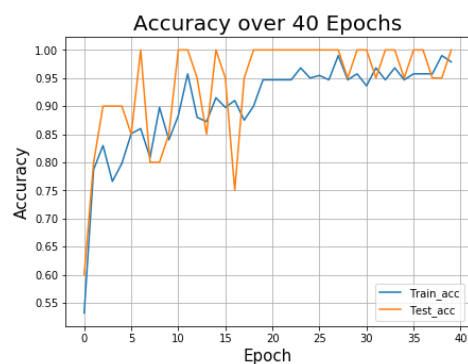
Parameter Details

- Optimizer: 'Adam' with learning rate 0.0005
- Loss function: BinaryCrossentropy
- Batch size:10 / epoch:40 / validation_split:0.2

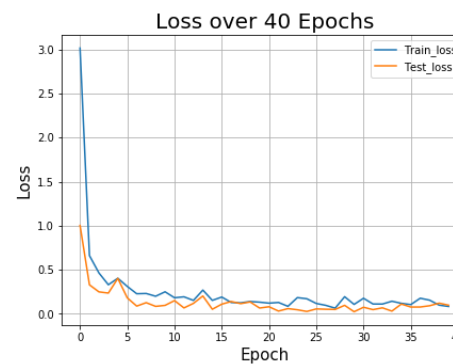
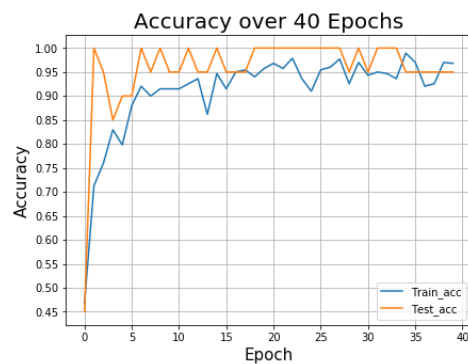
- Activation: intermediate layers use 'Relu', output layer use 'Sigmoid'
- Regularization: combine batch training and dropout layer with rate=0.2. It has small regularization effect.

• Result

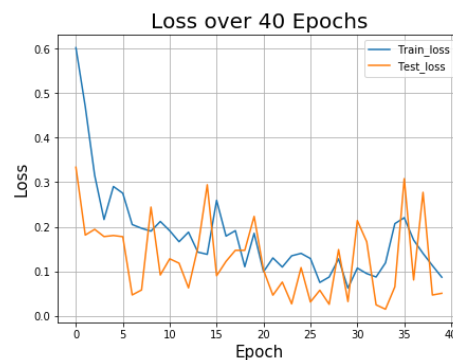
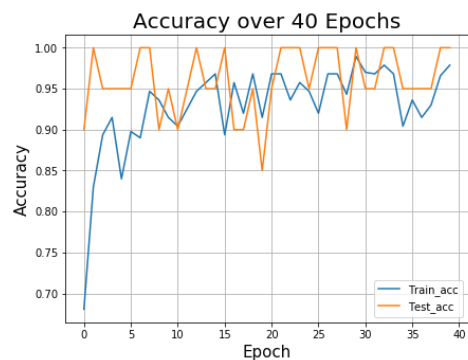
I run neural network with 3 different learning rates (0.001, 0.0005, 0.0001). Larger learning rate can help the scheme converge faster, but if it is too big the result might diverge. If learning rate is too small, it will take longer time to converge or will stuck in local optima. In this task, we already freeze the weights of convolution network, the algorithm only needs to learn the weight of fully connected layer, therefore less parameters to learn. Plus the effect of randomized input (shuffling), so the difference of learning rate is not shown significantly on the plot.



learning rate 0.001

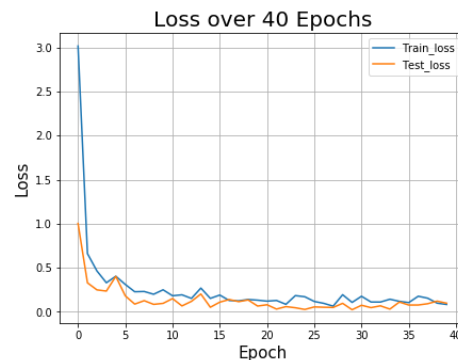
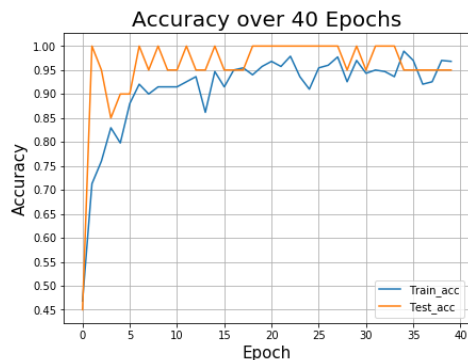


learning rate 0.0005

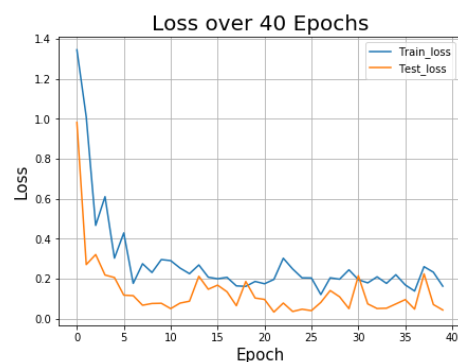
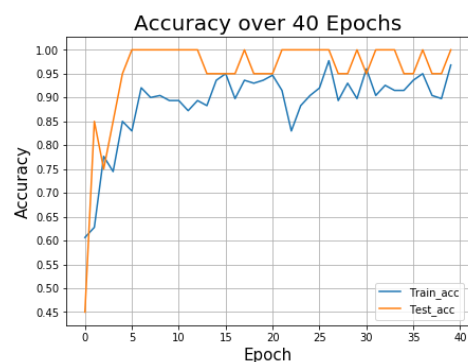


learning rate 0.0001

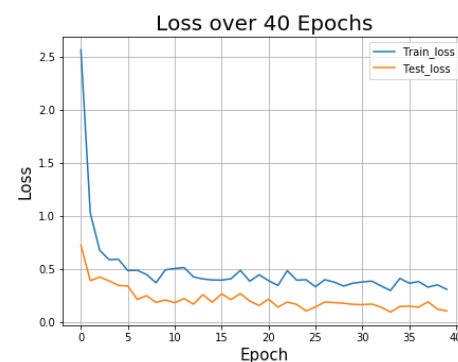
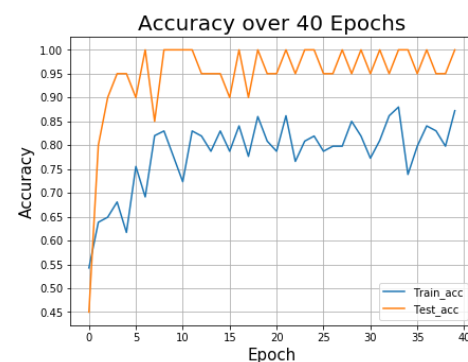
I run the model with different dropout rates (0.2, 0.5, 0.8) with learning rate = 0.0005. When using dropout, we can reduce the risk of model overfitting. Dropout set the unit to zero in a certain possibility, which inject some randomness to the feature learning, the network can't rely on one feature but will rather have to learn robust features that are useful. We need to fine tune the dropout rate. When increase dropout beyond a certain threshold, it results in the model not being able to fit properly. Intuitively, a higher dropout rate would result in a higher variance to some of the layers, which also degrades training. In the task, we also have data augmentation and batch training which also adds small regularization effects on the model.



Dropout rate = 0.2



Dropout rate = 0.5

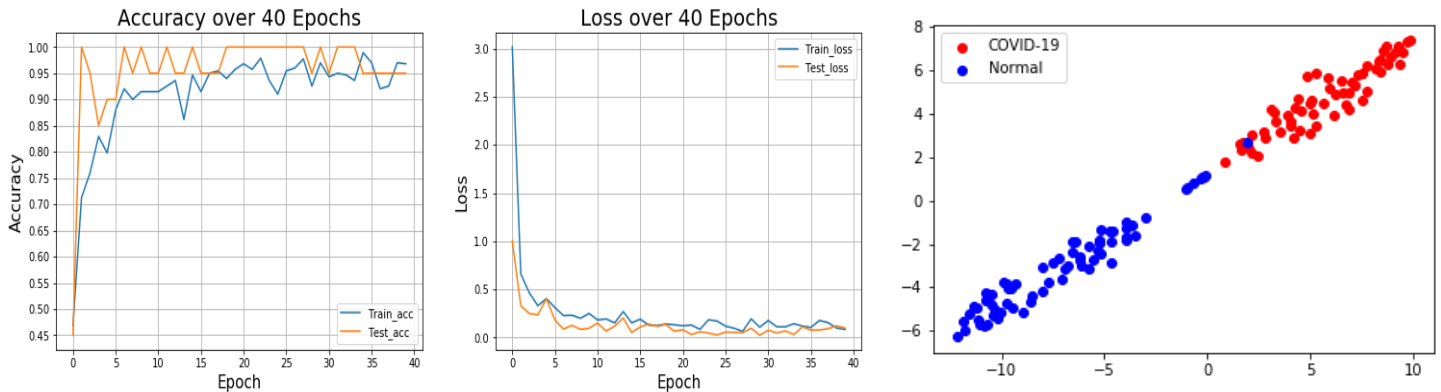


Dropout rate = 0.8

I choose the model with the hyperparameters which has best result, learning rate = 0.0005, Dropout rate = 0.2. Plotting the result as below, we can see that the VGG-16 model achieves to an 95% train and

validation accuracy after training on 40 epochs with Adam optimizer. The loss for train and validation are 0.12 and 0.15 respectively.

According to t-SNE visualization, we learn that after deep learning, we can extract features effectively. By projecting the features to a 2 dimension space, we can separate 'Normal', 'Covid-19' is separable so that our model is capable of generating a decision boundary with good generalization.



Take2: Multi-class classification

Architecture

In this task, I run the task using three different architectures.

- VGG-16: Same with Task1. Only change the output layer and the loss function.
- VGG-19: Same with VGG-16 but use the convolution part of VGG-19 instead of VGG-16. VGG-19 consists of 19 layers of deep neural network whereas VGG-16 consists of 16 layers of deep neural network, respectively. VGG-19 has a deeper structure.
- ResNet50: The ResNet-50 model consists of 5 stages each with a convolution and Identity block. Each convolution block has 3 convolution layers and each identity block also has 3 convolution layers. ResNet-50 uses skip connection to add the output from an earlier layer to a later layer. This helps it mitigate the vanishing gradient problem.

I utilize the convolution part of ResNet50 and substitute its top layers with Global Average Pooling and two Dense output layers.

Input: normalized 224x224x3 RGB image

Convolution layer: ResNet50's convolution part, output data with size 7x7x2048

Pooling layer: use GlobalMaxPooling2D layer, output is flattened with 2048 units

Fully connected layer: use two dropout layers and two fully connected layers with 512 units.

Output: use 'softmax' to perform 4-class classification

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

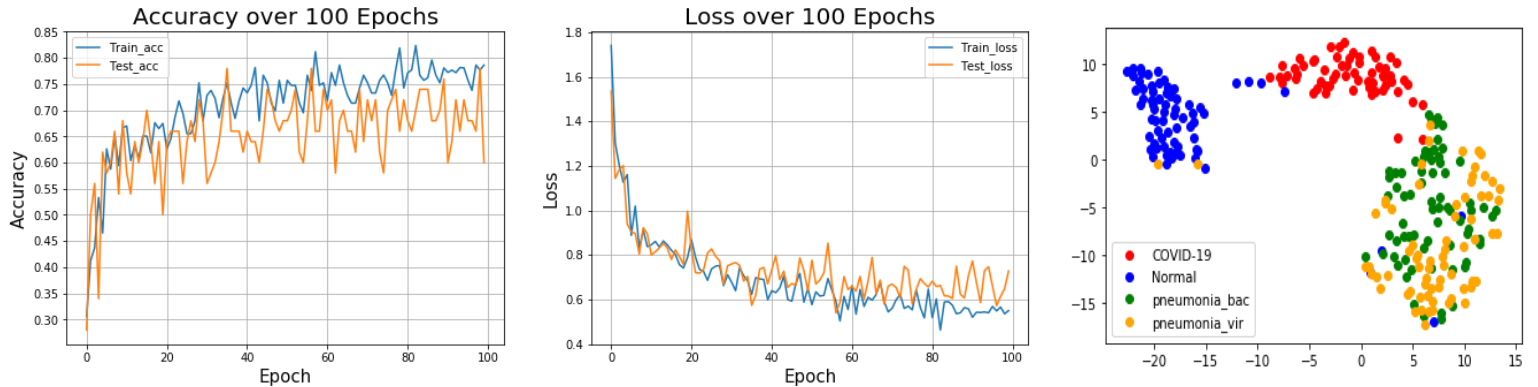
ResNet architecture

Parameter Details

- Optimizer: 'Adam' with learning rate 0.0001
- Loss function: CategoricalCrossentropy
- Batch size:10 epoch:100 validation_split:0.2
- Activation: intermediate layers use 'Relu', output layer use 'Softmax'
- Regularization: combine batch training and dropout layer with rate=0.2. It has small regularization effect.

Result

- VGG-16



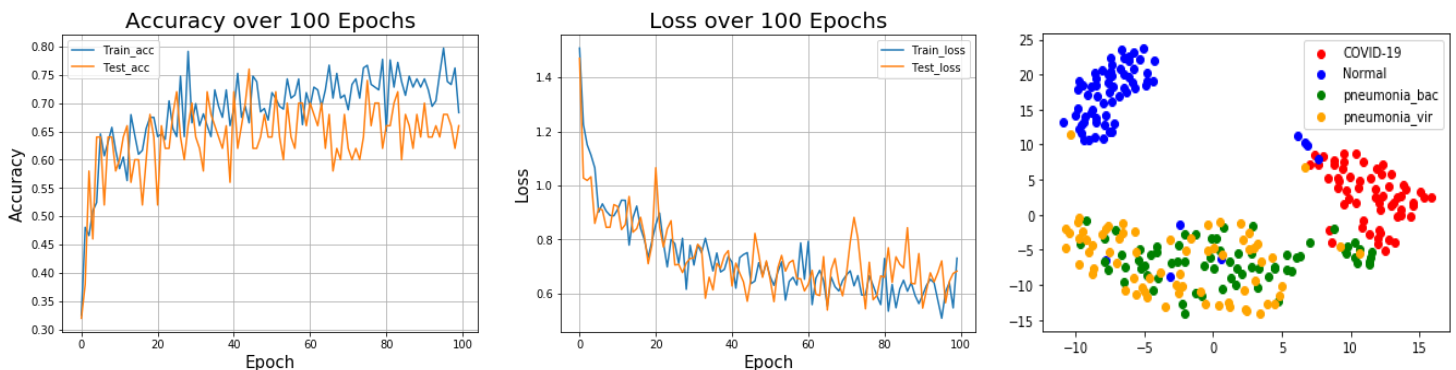
The VGG-16 model achieves to an 75% train accuracy and 70% validation accuracy after training on 100 epochs with Adam optimizer and a learning rate of 0.0001. The loss for train and validation are 0.58 and 0.7 respectively. On the test set, the test loss is 0.72 and test accuracy is 69%.

According to t-SNE visualization, we learn that after deep learning, we can extract features effectively. By projecting the features to a lower dimension space, we can separate 'Normal', 'Covid-19' and the rest

with high accuracy. But we didn't learn the feature to separate class 'pneumonia_bactaria' and 'pneumonia_virus'.

The test accuracy is slightly lower than train accuracy, it might be the reason that the model is slightly overfit, we can try to lower the batch size together with add stronger dropout layer to enhance the regularization, or to add regularization on the fully connected layer weights.

- **VGG-19**

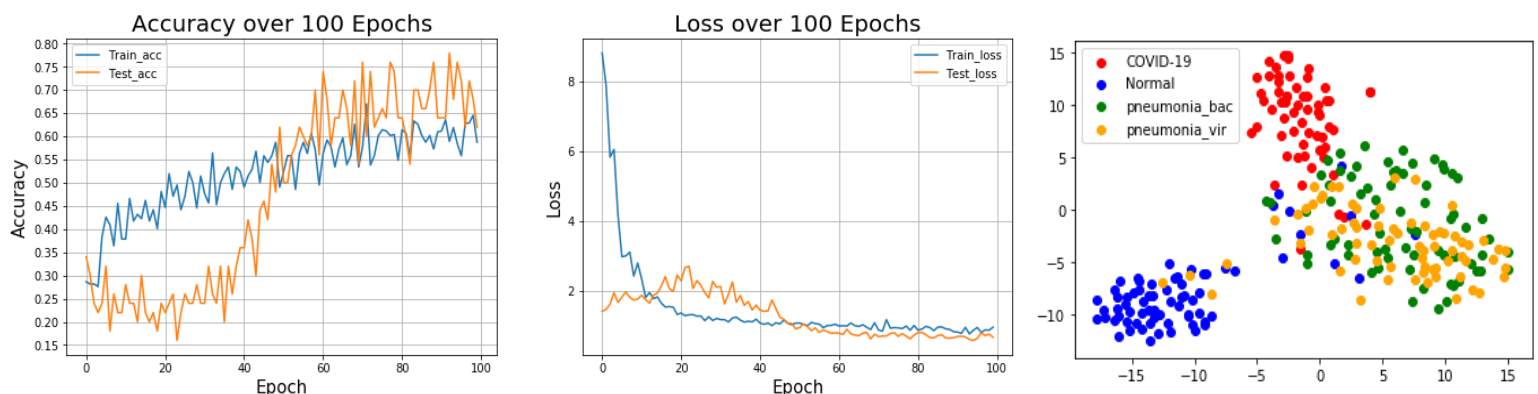


The VGG-19 model achieves to an 73% train accuracy and 67% validation accuracy after training on 100 epochs with Adam optimizer and a learning rate of 0.0001. The loss for train and validation are 0.67 and 0.7 respectively. On the test set, the test loss is 0.91 and test accuracy is 56%.

According to t-SNE visualization, the distance between 'normal' and the other class is bigger than VGG-16 model. So it has better generalization in classify the 'normal' class from the other sick classes than VGG-19. But as VGG-16, it didn't learn the features that can separate to separate class 'pneumonia_bactaria' and 'pneumonia_virus' effectively. To solve that problem, we can try to add increase the complexity of model or add more polynomial features, but it might have backfire effect on classifying 'normal' and 'covid-19' classes.

The test accuracy is slightly lower than train accuracy, it might be the reason that the model is slightly overfit, we can try to lower the batch size together with add stronger dropout layer to enhance the regularization, or to add regularization on the fully connected layer weights.

- **ResNet50**



The ResNet50 model achieves to an 65% train accuracy and 70% validation accuracy after training on 100 epochs with Adam optimizer and a learning rate of 0.0001. The loss for train and validation are 0.97 and 0.78 respectfully. On the test set, the test loss is 0.76 and test accuracy is 72%.

According to t-SNE visualization, the ResNet can learn features to classify 'Normal' and the other class with high accuracy. It also has similar effect as VGG model on classifying 'Normal', 'pneumonia_bactaria' and 'pneumonia_virus' classes. It might has untrivial misclassification when separate 'Covid-19' and 'pneumonia'. Also it has poor performance on 'pneumonia_bactaria' and 'pneumonia_virus' classification.

The test accuracy is almost equal to train accuracy. It takes more iterations for ResNet50 to converge. Before 50 epochs, there are huge gaps between train accuracy and validation accuracy.

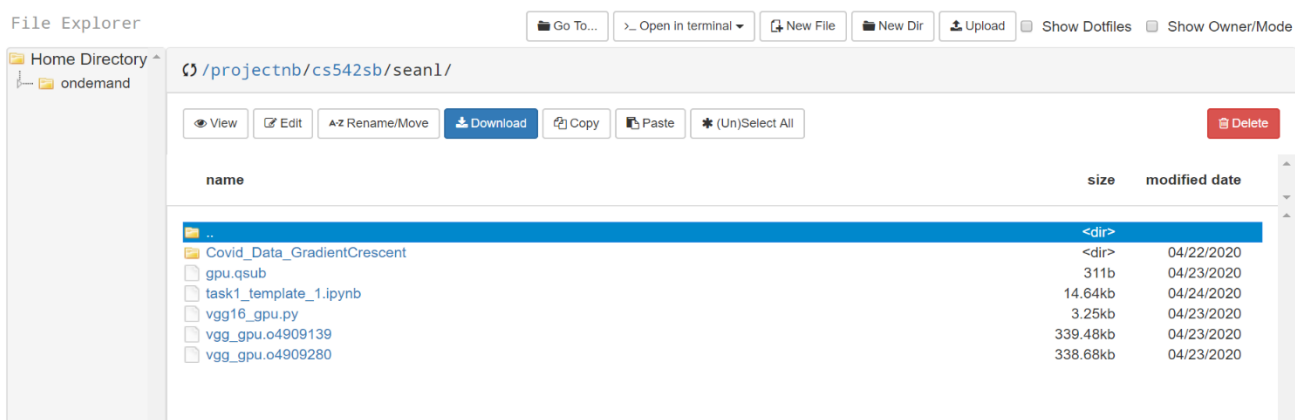
- **Model comparison**

Among three models, ResNet50 has largest depth and more complicated than the other two, while VGG-19 has least layers and complexity. Inherently, ResNet is harder to train and converge than VGG model. In addition to that, we add one more fully connected layer to ResNet than VGG model, so it takes more epochs and longer time to converge. VGG-19 has bigger overfitting effect than VGG-16, which can be alleviate by decreasing the model complexity, use stronger dropout layer or add regularization penalty on layer weights.

All the three models has similar generalization effects which is that they can separate 'Normal' class from the rest with high accuracy. When it comes to separate 'Covid-19' and 'pneumonia', they have decent performance but may have untrivial misclassification issue on some 'boundary data'. All the models can't generate a clear decision boundary for separating 'pneumonia_bactaria' and 'pneumonia_virus'.

Deep networks are hard to train because of the vanishing gradient problem — as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient extremely small. As a result, with the network depth increasing, its performance gets saturated or even starts degrading rapidly. That's part of the reason that the performance of VGG-19 is not noticeably better than VGG-16, but is even slightly worse. Increasing network depth does not work by simply stacking layers together. But ResNet uses a totally different architecture. Among all of three models, it achieves highest test accuracy.

Run model on SCC



name	size	modified date
..	<dir>	
Covid_Data_GradientCrescent	<dir>	04/22/2020
gpu.qsub	311b	04/23/2020
task1_template_1.ipynb	14.64kb	04/24/2020
vgg16_gpu.py	3.25kb	04/23/2020
vgg_gpu.o4909139	339.48kb	04/23/2020
vgg_gpu.o4909280	338.68kb	04/23/2020

```
← → ↻ scc-ondemand2.bu.edu/pun/sys/file-editor/edit/projectnb/cs542sb/seanl/g
Save /projectnb/cs542sb/seanl/gpu.qsub Key Bindings
1 #!/bin/bash -l
2
3 # Specify a project
4 # $ -p cs542sb
5
6 # Give job a name
7 # $ -N vgg_gpu
8
9 # Join error and output stream
10 # $ -j y
11
12 # Specify time limit
13 # $ -l h_rt=1:00:00
14
15 # Send email when the job is done
16 # $ -m e
17
18 # $ -l gpus=1
19 # $ -l gpu_c=6
20
21 module load python3/3.6.9
22 module load tensorflow/2.1.0
23
24 python vgg16_gpu.py
```

```
Save /projectnb/cs542sb/seanl/vgg_gpu.o4909139 Key Bindings Default Font Size 12px Mode Text Theme Solarized Light Word Wrap
2334 12/21 [=====>.....] - ETA: 1s - loss: 0.5296 - accuracy: 0.7500
2335 13/21 [=====>.....] - ETA: 1s - loss: 0.5223 - accuracy: 0.7460
2336 14/21 [=====>.....] - ETA: 1s - loss: 0.5086 - accuracy: 0.7574
2337 15/21 [=====>.....] - ETA: 1s - loss: 0.5159 - accuracy: 0.7329
2338 16/21 [=====>.....] - ETA: 0s - loss: 0.5515 - accuracy: 0.7244
2339 17/21 [=====>.....] - ETA: 0s - loss: 0.5429 - accuracy: 0.7349
2340 18/21 [=====>.....] - ETA: 0s - loss: 0.5482 - accuracy: 0.7386
2341 19/21 [=====>.....] - ETA: 0s - loss: 0.5772 - accuracy: 0.7258
2342 20/21 [=====>.....] - ETA: 0s - loss: 0.5732 - accuracy: 0.7296
2343 21/21 [=====] - 5s 238ms/step - loss: 0.5701 - accuracy: 0.7379 - val_loss: 0.6935 - val_accuracy: 0.6400
2344 Epoch 100/100
2345
2346 1/21 [>.....] - ETA: 4s - loss: 0.6196 - accuracy: 0.6000
2347 2/21 [=>.....] - ETA: 3s - loss: 0.6687 - accuracy: 0.6000
2348 3/21 [===>.....] - ETA: 3s - loss: 0.6312 - accuracy: 0.6667
2349 4/21 [====>.....] - ETA: 4s - loss: 0.6700 - accuracy: 0.6500
2350 5/21 [=====>.....] - ETA: 3s - loss: 0.6549 - accuracy: 0.6800
2351 6/21 [=====>.....] - ETA: 3s - loss: 0.6220 - accuracy: 0.6786
2352 7/21 [=====>.....] - ETA: 2s - loss: 0.5915 - accuracy: 0.7121
2353 8/21 [=====>.....] - ETA: 2s - loss: 0.6299 - accuracy: 0.6842
2354 9/21 [=====>.....] - ETA: 2s - loss: 0.6540 - accuracy: 0.6744
2355 10/21 [=====>.....] - ETA: 2s - loss: 0.6422 - accuracy: 0.6875
2356 11/21 [=====>.....] - ETA: 1s - loss: 0.6386 - accuracy: 0.6981
2357 12/21 [=====>.....] - ETA: 1s - loss: 0.6335 - accuracy: 0.6983
2358 13/21 [=====>.....] - ETA: 1s - loss: 0.6226 - accuracy: 0.7143
2359 14/21 [=====>.....] - ETA: 1s - loss: 0.5961 - accuracy: 0.7279
2360 15/21 [=====>.....] - ETA: 1s - loss: 0.6083 - accuracy: 0.7192
2361 16/21 [=====>.....] - ETA: 0s - loss: 0.5939 - accuracy: 0.7244
2362 17/21 [=====>.....] - ETA: 0s - loss: 0.6183 - accuracy: 0.6988
2363 18/21 [=====>.....] - ETA: 0s - loss: 0.6092 - accuracy: 0.7102
2364 19/21 [=====>.....] - ETA: 0s - loss: 0.6191 - accuracy: 0.7097
2365 20/21 [=====>.....] - ETA: 0s - loss: 0.6085 - accuracy: 0.7092
2366 21/21 [=====] - 5s 224ms/step - loss: 0.6029 - accuracy: 0.7136 - val_loss: 0.5360 - val_accuracy: 0.7400
2367 /share/pkg.7/tensorflow/2.1.0/install/lib/SCC/./python3.6-gpu/site-packages/keras_preprocessing/image/image_data_generator.py:716: UserWarning: This ImageDataGenerator specifies `fo
2368 warnings.warn('This ImageDataGenerator specifies `
2369 /share/pkg.7/tensorflow/2.1.0/install/lib/SCC/./python3.6-gpu/site-packages/keras_preprocessing/image/image_data_generator.py:735: UserWarning: This ImageDataGenerator specifies `zo
2370 warnings.warn('This ImageDataGenerator specifies `
2371 Training Time: 497.39s
2372
```

Running Task2 VGG_16 model

Training time	
Run on CPU	7453.10s
Run on GPU	497.39s