CS 542 Class Challenge

Image Classification of COVID-19 X-rays

Shuo Liu
BUID: U28149952
email: liushuo@bu.edu

Ji Zhao
BUID: U23540628
email: zhaoji@bu.edu

Task 1:

Based on the hint of task 1 we tried two different models VGG16, VGG19 for the classification as the pre-training model and it turned out that the accuracy of VGG16 is a little higher than VGG19 if we keep same other parameters and layers and finally we decided to use VGG16 and tuned certain hyperparameters in order to achieve the best accuracy. The deep neural network model VGG16 contains 16 weighted layers. The input of the network has a (224,224,3) shape and begins with two convolution layers, with a kernel size of (3x3), and the number of output channels is 64. Then a (2,2) max-pooling layer is followed to reduce the dimensionality. Two convolutional layers are applied again, with the (3x3) kernel size, this time the number of output channels is 128. As we go further into the network, the number of convolutional layers increases from two to three as a block, and the number of output channels increases from 128 to 256, and finally 512. Among these CNN blocks, (2,2) max-pooling is applied after every block. After the last pooling layer, we flatten the 512 output channels and two dense layers having 256 and 1 neuron were added with sigmoid acting on the last dense layer since we only have two classes. We tried adding Dropout layers as regulation after the flattening layer and before the output layer with a variety of rates from 0.1 to 0.25 and found this value did not cause high performance. Figure 1 is the architecture of our model:

```
Model: "model_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten_5 (Flatten)         (None, 25088)             0

 dense_feature (Dense)       (None, 256)               6422784

 dense_5 (Dense)             (None, 1)                 257

=================================================================
Total params: 21,137,729
Trainable params: 6,423,041
Non-trainable params: 14,714,688
_________________________________________________________________
```

Fig 1. The architecture of the task 1 model

When we compile the model, we combine the Focal Loss function (with k.binary_Crossentropy) in our model (alpha=0.25, gamma=2). A Focal Loss function addresses class imbalance during training in tasks like object detection. Focal loss applies a modulating term to the cross-entropy loss in order to focus learning on hard misclassified examples. It is a dynamically scaled cross-entropy loss, where the scaling factor decays to zero as confidence in the correct class increases. Intuitively, this scaling factor can automatically down-weight the contribution of easy examples during training and rapidly focus the model on hard examples. The optimizer we set is Adam, which involves a combination of two gradient descent methodologies: (1) Momentum: This algorithm is used to accelerate the gradient descent algorithm by taking into consideration the 'exponentially weighted average' of the gradients. Using averages makes the algorithm converge towards the minima at a faster pace; (2)Root mean square: Prop or RMSprop is an adaptive learning algorithm that tries to improve adaptive gradient. Instead of taking the cumulative sum of squared gradients, it takes the 'exponential moving average'.
Our learning rate was 0.0005 (which is better than 0.0001 in terms of accuracy). We also chose to use Epochs as 40. Based on figure 2, there is no overfitting so Early stopping is not needed. Note that when we increased the number of epochs to a much higher number, overfitting happened. Figure 2 is the accuracy and loss of our model over 40 epochs:

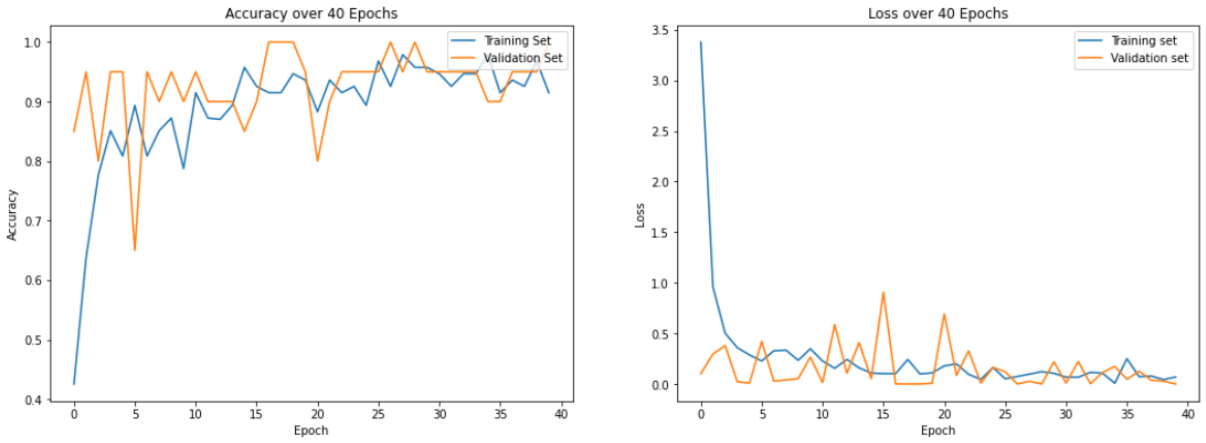

Fig 2. The accuracy and loss of our model over 40 epochs

For both the training and validation set accuracies, we can see that they are increasing at a sharp rate at the beginning while with a slow rate after around 4 epochs, but there are still some vibrations in between, especially for the validation set. This is most likely due to the minibatch training that is being used. This also explains why the validation accuracy is better than the training accuracy: during training, the model has access to all parameters so does validation but all parameters are only trained under part of the training set. The loss for the training and validation sets are all decreasing at a sharp rate at the beginning and after around 2 epochs the loss values are all less than 1, and for most epochs after 2, the loss values are less than 0.5. The usage of minibatch training might also be the reason that the training set loss is smoother than the validation set loss. To see the model's performance on test data, we evaluated the model on data from a test directory and received a more than 99% accuracy for

COVID19 Pneumonia and 99% accuracy for Normal, which means that the model has learned to generalize between two classes like shown in figure 3.

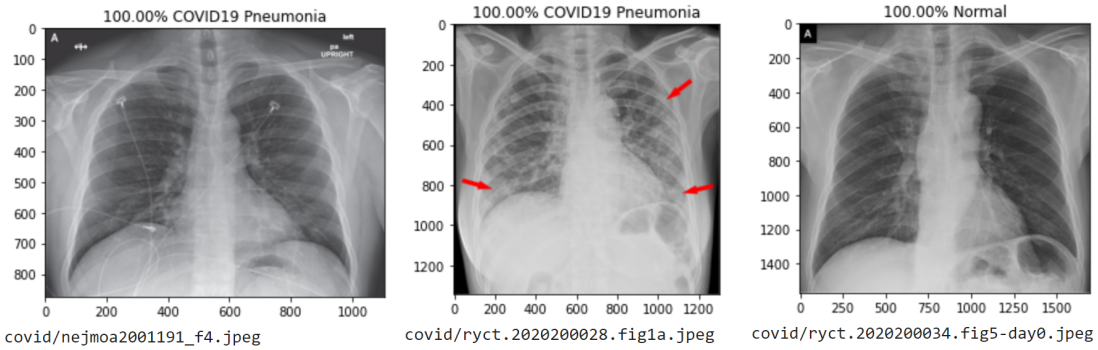

Fig 3. The accuracy for COVID19 and Normal

Finally, we show the visualization of the features of the test data in figure 4. The visualization plot was made using T-SNE to reduce the dimensionality of the data into a 2-dimensional feature space. The Blue data points are COVID-19 X-rays, and the orange data points are Normal X-rays. These features were evaluated from the input of the model to the "dense_feature" output. From this visualization, we can see besides the few Normal X-rays (several orange points) that are overlapping the COVID-19 X-rays (one blue point), the two classes are separable and this explains the high accuracy of our model: the model was able to create features that would be able to distinguish between the two classes.

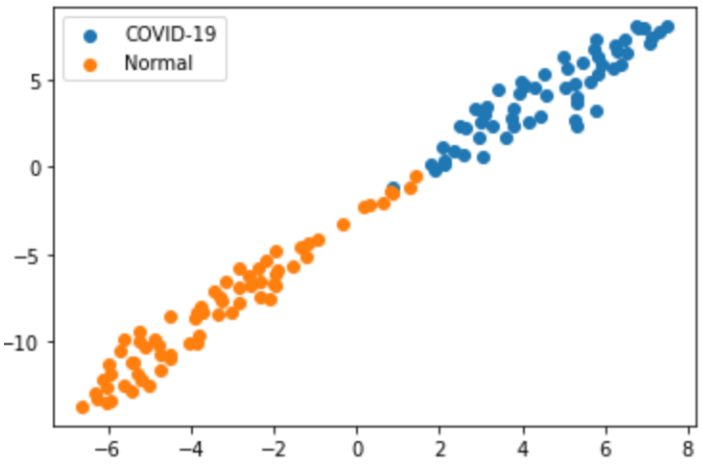

Fig 4. The visualization of COVID-19 X-rays and Normal X-rays

Task 2:

In this task, we were asked to train a deep neural model to classify an X-ray image into one of the following classes: normal, COVID-19, Pneumonia-Bacterial, and Pneumonia-Viral. VGG16 had also been applied in this task as well as an architecture called MobileNet for their outstanding image classification abilities.

Model 1:

Unlike the VGG16 model in Task 1, the output layer was changed to include 4 perceptrons, for 4 classes. For this task, the softmax function was used in the output layer and CategoricalCrossEntropy was applied as a loss function instead of the BinaryCrossEntropy loss in task 1. The feature extraction layer also contains 256 neurons. In this task, the epoch number was increased to 100 from 40 for this multi-class task. The optimizer we set is also Adam. It took about 15 mins to fit the model. The structure of the modified model in task 2 is as follows:

```
Model: "VGG16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten_14 (Flatten)        (None, 25088)             0

 feature_dense (Dense)       (None, 256)               6422784

 dense_14 (Dense)            (None, 4)                 1028

=================================================================
Total params: 21,138,500
Trainable params: 6,423,812
Non-trainable params: 14,714,688
_________________________________________________________________
```

Fig 5. The structure and parameters of the modified VGG16 model

In addition to the VGG16 model, we applied MobileNet in this task.

Model 2:

MobileNet, as shown in the below figure, has a smaller structure, with less computation and higher precision that can be used on mobile terminals and embedded devices. Based on a deep separable roll product, Mobilenet uses two global hyperparameters to maintain a balance between efficiency and accuracy. The core idea of MobileNet is the decomposition of the convolution kernel. Standard convolutions can be decomposed into depthwise convolutions and pointwise convolutions with convolution kernels by using depthwise separable convolutions. A depthwise convolution filter convolved each channel, and the convolution was used to combine the outputs of the depthwise convolutional layers.

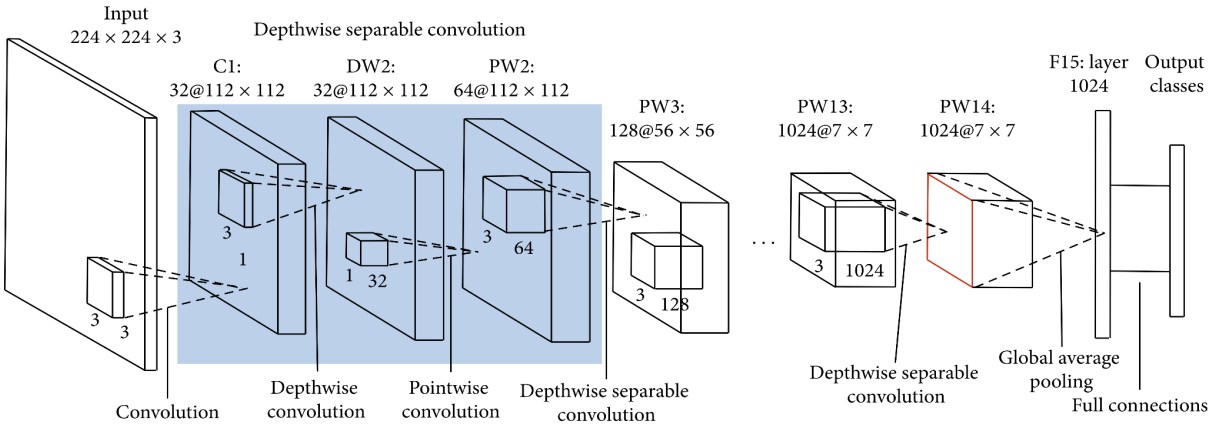

Fig 6. The structure of the MobileNetV1

The architecture of MobileNet has the same Input as the VGG16 architecture, with the image of the X-rays (224,224,3), but the rest of the architecture is very different in comparison.

The first layer of this MobileNet is a (3x3) standard convolution, followed by the accumulation of Depthwise convolutions. As can be seen, some depthwise convolutions will continue to downsample with strides=2. Then, the features are changed to (1x1) by average pooling, a fully connected layer is added according to the predicted class size, and the last Softmax layer. The entire network has 28 layers (excluding Average-Pool and Softmax). For parameters, it mainly focuses on (1x1) convolutions. In addition, the fully connected layer accounts for part of the parameters. Instead of using the last three layers of the standard model, we use a flatten and three fully connected layers to extract features and get the output. We also added a global-average pooling layer in the model which preserved the spatial information extracted by the previous convolutional layers and pooling layers. The modified model structure in Task 2 is as follows:

```
Model: "MobileNet"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 mobilenet_1.00_224 (Functio  (None, 7, 7, 1024)       3228864
 nal)

 global_average_pooling2d_5   (None, 1024)             0
 (GlobalAveragePooling2D)

 flatten_11 (Flatten)         (None, 1024)             0

 dense_9 (Dense)              (None, 512)              524800

 feature_dense (Dense)        (None, 256)              131328

 dense_10 (Dense)             (None, 4)                1028

=================================================================
Total params: 3,886,020
Trainable params: 3,864,132
Non-trainable params: 21,888
_________________________________________________________________
```

Fig 7. The structure and parameters of the modified MobileNet model

The number of channels increases after each Dense block, going from 32 to 64, to 128, to 512, and finally 1024. Unlike VGG16, which requires 6.4 million parameters to train, the MobileNet architecture requires fewer parameters to train, which means less time cost. Batch Normalization exists after almost every convolutional layer in MobileNet to speed up training convergence and improve accuracy.

Accuracy:

The learning rate in task 2 for both models was 0.0001 because our model's accuracy dropped when we tried a learning rate greater than 0.0001. When we set the learning rate to be less than 0.0001, the training process became too slow and not efficient enough.

We tried many methods to avoid overfitting. Firstly, we used the early-stop method, but every training session stopped early, we would always have a model that would underfit. Secondly, we tried adding dropout layers to our models, but they performed worse than models without them. Moreover, as is known to us, a higher epoch might lead to a worse accuracy because of overfitting. Therefore, we tried to change the number of epochs to increase our models' accuracy.

Accuracy and Loss of Model 1: VGG16;

Parameters: Epoch = 100; Activation function = Softmax;

For this model, we used 100 epochs because when we tried higher epochs, the training and validation set began to overfit, and the performance of the model was significantly worse on the test set. When we used fewer epochs, there would be underfitting and the model performed worse or had no change on the test set too.
In order to get higher accuracy, we compared the accuracies of VGG16 and VGG19. We finally chose VGG16 because of its better performance even though VGG19 is more complex than VGG16.

Here is the accuracy of VGG16:

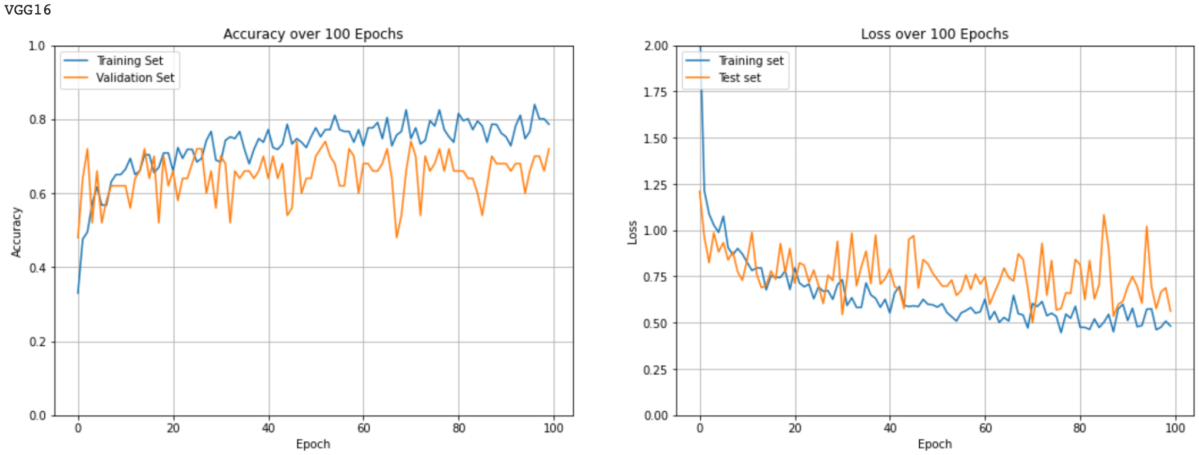

Fig 8. The accuracy and loss of VGG16

In task 1, the training accuracies were always lower than the validation accuracies because of the small amount of validation data. However, in task 2, the training accuracies were higher than the ones in the validation set. Around epoch 65 we could see that the model started to overfit the training set. From the loss graph on the right side of figure 8, there is a peak around epoch 65 which is consistent with the accuracy graph on the right. After training VGG16, it could be overfitting. To possibly improve performance, we can add more layers to the network so that it can constantly learn new nonlinear features and thus be able to classify.

Accuracy and Loss of Model 2: MobileNet;
Parameters: Epoch = 100; Activation function = Softmax;

For this model, we also used 100 epochs. We finally chose MobileNetV1 among MobileNetV1, V2, and V3 because of its better performance.

Here is the accuracy of MobileNet:

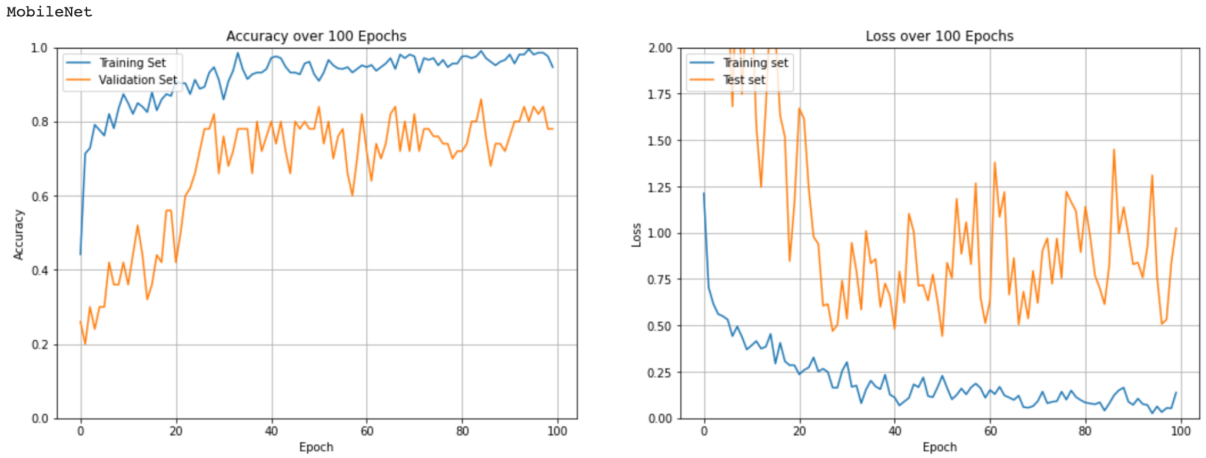

Fig 9. The accuracy and loss of MobileNet

Unlike VGG16, the trajectory of the accuracy of the training and validation improves over the number of epochs. This model performed well on the training set with really high accuracies, but the fluctuation of accuracy was too large for the validation set. Although the model didn't perform very well graphically, even after we added the dropout layer, the accuracy fluctuated greatly for the validation set and decreased for the test set. After repeated comparison and modification of model parameters, the current model performed best on the test set with an accuracy of about 0.8333. It performed better than VGG16 in task 2 with an accuracy of about 0.7500, which is shown in table 1.

Model	Epoch	#Dropout Layer	Activation function	Test Accuracy
VGG16	**100**	**0**	**Softmax**	**0.7500**
MobileNet	**100**	**0**	**Softmax**	**0.8333**

Table 1. Some information on models

T-SNE of two Models: VGG16 and MobileNet;

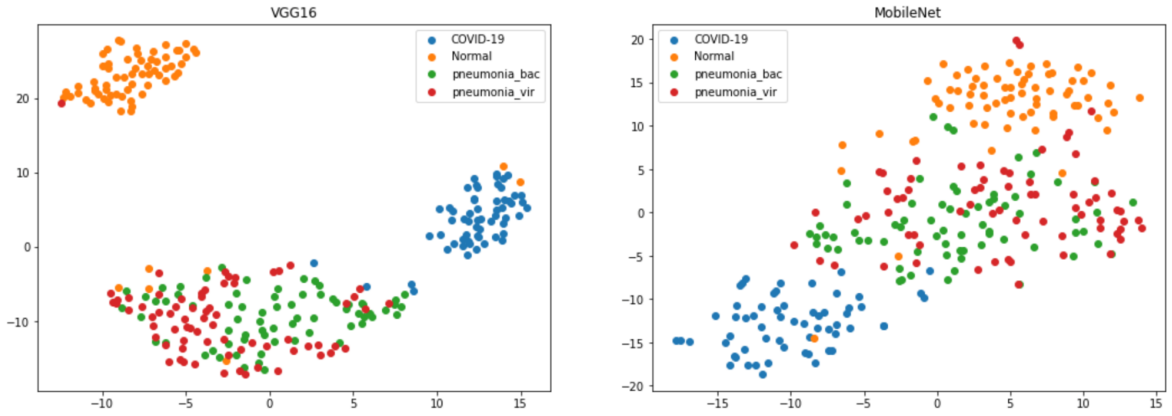

Fig 10. T-SNE of VGG16 (left) and MobileNet (right)

The visualizations of the features learned by the two models are shown in figure 10. Like in task 1, we also used T-SNE to reduce the dimensionality to two models to visualize the test data. In the two plots above, the blue data points are COVID-19 X-rays, orange data points are normal X-rays, green data points are pneumonia-bacterial X-rays, and red represents pneumonia-viral X-rays. The left side is the modified VGG16 model while the right side shows the modified MobileNet model. For the VGG16, These features were evaluated from the input of the model to the "feature_dense" output. We can see that in this model, features extracted from the test set were able to clearly separate normal and COVID-19 X-rays, though there is a small overlap of normal by pneumonia-viral data points and a small overlap of COVID-19 by pneumonia-bacterial data points. We speculate that few normal people have similar cinematographic signs to pneumonia-viral patients, and so do COVID-19 patients with normal people. We also see that the VGG16 model lacks the ability to distinguish the Pneumonia-bacterial and Pneumonia-viral X-rays, which may be due to the prime similarity of cinematographic signs with minor details that the model can not learn. Compared to the VGG16 model, the MobileNet model shows obviously better performance in terms of distinguishing the normal and COVID-19 from Pneumonia-bacterial and Pneumonia-viral X-rays for fewer overlap regions. It seems that for both models it is not difficult to separate X-rays of COVID-19 patients and normal people. We might select Pneumonia-bacterial and Pneumonia-viral X-rays for a binary classification case and try to use transfer learning for more clear feature extraction.

Compared to the T-SNE visualization in task 1, the modified VGG16 model in task 2 has the ability to learn how different the normal X-rays and COVID-19 X-rays are due to the long distance between two clusters. Nonetheless, the VGG16 lacks the ability to extract features to clearly separate Pneumonia-bacterial and Pneumonia-viral X-rays clusters.