Class Challenge
Justin Janice

Architecture description:

Task 1

The model uses a pre-trained VGG16 as an initial layer. Images of size 224 X 224 X 3, which is 224 by 224 pixels and then 3 RGB channels were input into VGG16. Initially ResNet152V2 was used as a base layer, but it did not perform as well as VGG16 after further testing. When training, the VGG16 layer is frozen so as to not unlearn the imagenet weights. The reason for this is VGG16 is already a very effective image classification model, so it should be able to perform well on these new images. The top layers of VGG16 are removed because those are necessary for a different classification problem requiring 10,000 classes, so therefore new top layers are included that will work for Covid classification.

The VGG16 outputs a size of 512 7X7 feature maps. This layer has 14714688 parameters, but these are not trainable since VGG16 is frozen.

The model then flattens the output from VGG16 to a fully connected layer of 25088 neurons.

The model feeds this fully connected layer into a dense layer that uses the relu activation function and outputs a fully connected layer of 256 neurons, and has 6422784 trainable parameters. This layer is meant to add capacity to the model, so more complex relationships can be found between the values. Relu is used for the hidden layer activation function because it is really easy to compute and does not saturate the input.

The model finally feeds the fully connected layer into another dense layer that uses the sigmoid activation function and outputs 1 neuron, which will be our final probability. This layer produces 257 trainable parameters. Sigmoid is used to normalize the results between 0-1 which produces a probability for if that X-ray represents someone with Covid or not.

No dropout layers were included because the model did not seem to be overfitting to the training data since both training and validation accuracy stayed consistently around .90 and 1.

Parameters:

Task 1

The model uses the optimizer function adam, with a learning rate of 0.0001. 0.0001 was the optimal value for the learning rate because it prevented any major oscillations in accuracy and loss. Adam seemed like the best choice because it is essentially an improved version of stochastic gradient descent. For the loss function, the model used binary cross entropy. Cross

entropy is one of the best loss functions for classification problems, and since this model is trying to classify for 2 classes, binary cross entropy seemed like the best option. There was no need for any regularization techniques in this model because it does not appear to overfit the training data, and works very well on the validation set. I train the model for 40 epochs because it seems the model converges on an optimal accuracy by that point in training. The model uses batch sizes of 10 because this worked well for training.

Architecture description:

Task 2

The model used a pre-trained VGG16 as an initial layer. Images of size 224 X 224 X 3, which is 224 by 224 pixels and then 3 RGB channels were input into VGG16. Initially ResNet152V2 was used as a base layer, but it did not perform as well as VGG16 after further testing. When training, the VGG16 layer is frozen so as to not unlearn the imagenet weights. The reason for this is VGG16 is already a very effective image classification model, so it should be able to perform well on these new images. The top layers of VGG16 are removed because those are necessary for a different classification problem requiring 10,000 classes, so therefore new top layers are included that will work for Covid classification. The

VGG16 outputs a size of 512 7X7 feature maps. VGG16 has 14714688 parameters, but these are not trainable since the layer is frozen.

The model feeds this through 2 more convolutional layers with filters of size (3,3), which then first outputs 1024 feature maps of size 5X5 and then 1024 feature maps of size 3X3. These layers produce 4719616 and 9438208 parameters respectively. Convolutional layers were included in an attempt to increase the number of trainable parameters, in the hope of increasing accuracy, which was effective.

The model then flattens the output from VGG16 to a single fully connected layer containing 9216 nodes.

The fully connected layer is fed into a dropout layer that has a probability of 0.5 because without this the model was overfitting, which caused a low testing accuracy, after the inclusion of dropout the training accuracy is lowered but testing accuracy increases.

The model feeds the fully connected layer into a dense layer that uses the relu activation function and outputs a fully connected layer containing 256 neurons. This layer produces 2359552 trainable parameters. This layer is meant to add capacity to the model, so more complex

relationships can be found between the values. Relu is used for the hidden layer activation functions because it is really easy to compute and does not saturate the input.

Next, the model feeds the output into a final dropout layer with a probability of 0.20 to further try and combat any potential overfitting.

Finally this fully connected layer is fed into another dense layer that uses the softmax activation function and outputs 4 nodes, one for each class. This layer produces 1028 trainable parameters. Softmax is used because it will find a probability for the image for each class and then output the class with the highest probability.

Parameters:
Task 2

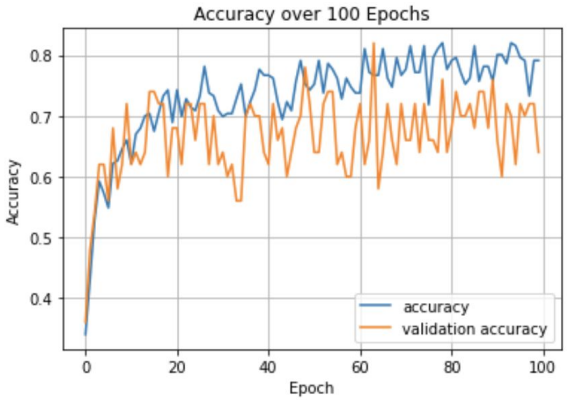
The model uses the optimizer function adam, with a learning rate of 0.0001. 0.0001 was the optimal value for the learning rate because it prevented major oscillations in accuracy and loss. Adam seemed like the best choice because it is essentially an improved version of stochastic gradient descent. For my loss function I used categorical cross entropy. Cross entropy is one of the best loss functions for classification problems, and since the model is trying to classify for more than 2 classes, categorical cross entropy was used. The regularization technique used was dropout, this is one of the most common ways to regularize a deep neural network, because by randomly dropping out neurons, it forces the model to learn and make predictions using less information, which will generalize the model. The model uses dropout layers before each of its dense layers. I train the model for 40 epochs because it seems the model converges on an accuracy by that point in training. The model uses batch sizes of 10 because this worked well for training.

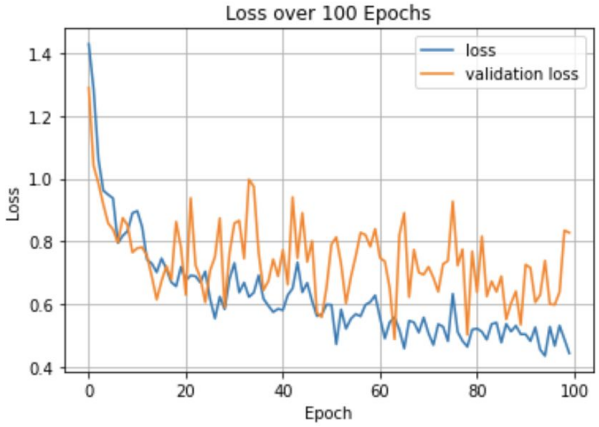
Comparison of Architectures for Task 2

Step 1: Choosing a pre-trained model.
I first tried using VGG16 to see how effective it works on the classification of chest x-rays. By simply changing the top layers of VGG16, and making the model perform a 4 way classification, rather than classifying to 10,000 classes. I wanted to try another popular pre-trained model to make sure there wasn't a better option compared to VGG16. I tried using ResNet152V2, and once again altered the top layer to work for our classification problem. I came to find ResNet's accuracy was a bit lower than VGG16, and ultimately decided on VGG16 as a base layer.

Step 2: Replicating the model from Task 1.

I decided to now try and replicate the model from Task 1. After VGG16, I start with a flatten layer, to flatten the model. Then, I add a dense layer of 256 neurons to attempt and add capacity to the model. Next, change the final dense layer to contain 4 nodes rather than one to now perform a 4 way classification, and change the final activation function from Sigmoid to Softmax. This model resulted in a testing accuracy of 58%, and it seemed the reason for low testing accuracy was because of minor overfitting. This can be shown in the plot for accuracy vs validation accuracy because the validation accuracy halfway through training seems to consistently dip below the training accuracy.





It's shown that after around 50 epochs the loss and accuracy begin to separate from the validation loss and accuracy, which shows there is some overfitting happening.

```
36/36 [==============================] - 3s 75ms/step - loss: 0.7544 - accuracy: 0.5833
Test loss: 0.7543615102767944
Test accuracy: 0.5833333134651184
```

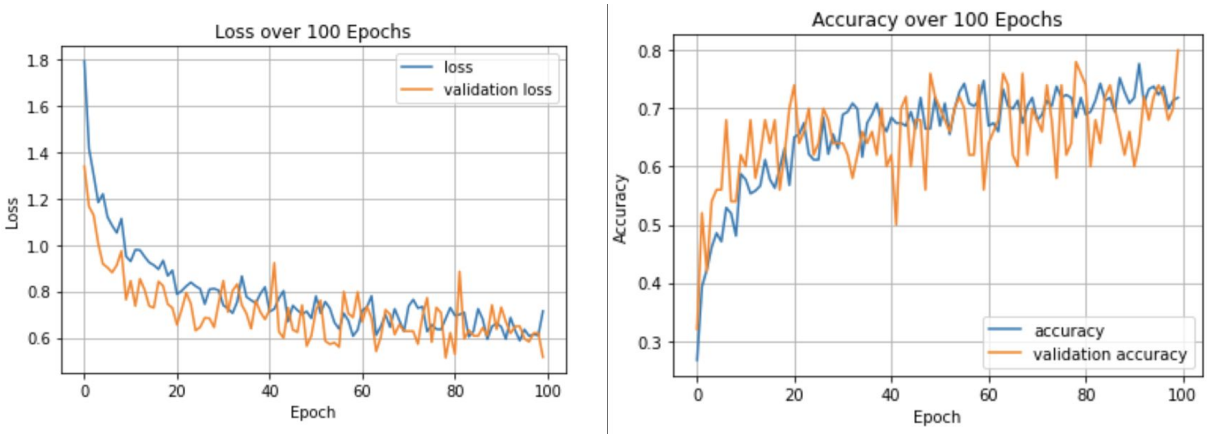
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 256)               6422784
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 1028
=================================================================
Total params: 21,138,500
Trainable params: 6,423,812
Non-trainable params: 14,714,688
_________________________________________________________________
```

This is the model I consider as my initial architecture, and build off of.

Step 3: Dropout Layers

Since the replication of the model from part one resulted in overfitting. I decided to begin adding dropout layers. I attempted many different combinations of dropout layers, which included only one dropout layer before either of the dense layers, or dropout layers before both the dense layers. I also tried this with all different probabilities for the dropout layers. I settled on dropout

layers before both the dense layers with a probability of 0.50 for the first dropout layer and 0.2 for the second.



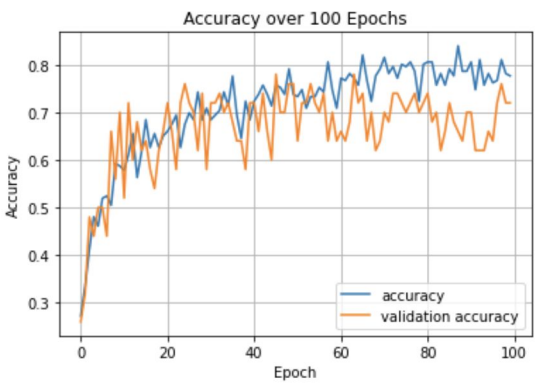
From the graphs, you can see much less separation between accuracy and validation accuracy as well as loss and validation loss. This shows our model is more generalized and this holds true in our testing accuracy, which is now around 67%.

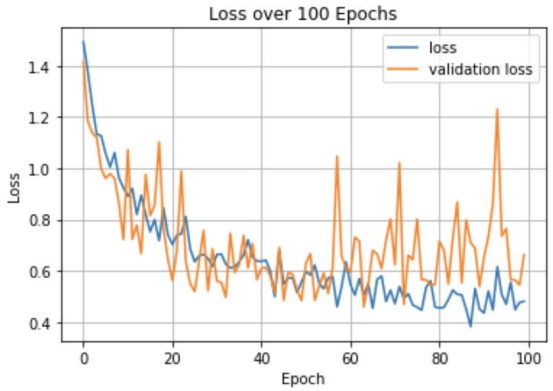
```
36/36 [==============================] - 2s 69ms/step - loss: 0.7099 - accuracy: 0.6667
Test loss: 0.7099384069442749
Test accuracy: 0.6666666865348816
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dropout (Dropout)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 256)               6422784
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 1028
=================================================================
Total params: 21,138,500
Trainable params: 6,423,812
Non-trainable params: 14,714,688
_________________________________________________________________
```

Step 4: Improving accuracy

Now that I have handled overfitting, I wanted to see if I would be able to improve my accuracy any further. The way I did this was by trying to incorporate 2 more convolutional layers right after the output of VGG16. I thought this could give me more trainable parameters considering I don't train at all on VGG16. The VGG16 outputs a size of 7 X 7 X 512. I then run this through 2 more convolutional layers with filters of size (3,3), which then first outputs 1024 feature maps of size 5X5 and then 1024 feature maps of size 3X3. I was hoping those extra parameters could help to further generalize the model and improve accuracy. I came to discover that this was indeed the case and after adding in these extra layers, I was able to improve my testing accuracy to 72%. Since VGG16 is already so effective at image classification and outputs such small feature maps there is not too much that can be done to try and better improve the accuracy, but it does seem adding these extra convolutional layers can give a slight edge.





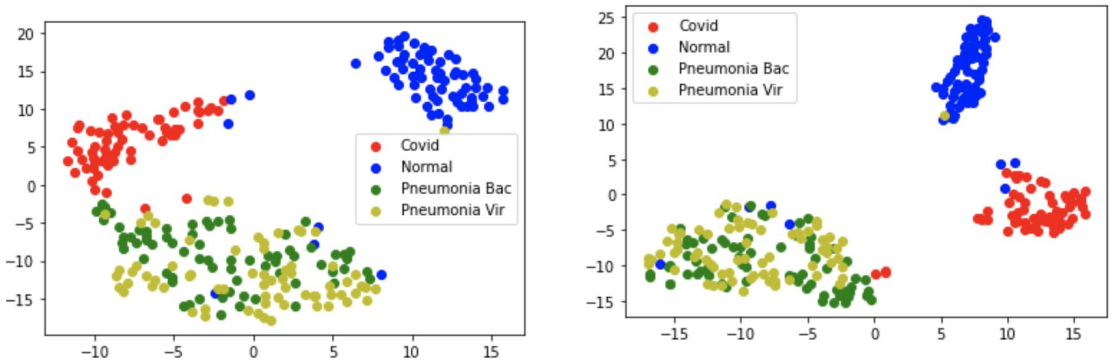

```
WARNING:tensorflow:From <ipython-input-7-6e2e55ffea00>:8: Model.evaluate_generator (from tensorflow.python.keras.engi
ne.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.evaluate, which supports generators.
36/36 [==============================] - 3s 79ms/step - loss: 0.8610 - accuracy: 0.7222
Test loss: 0.8610367178916931
Test accuracy: 0.7222222089767456
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
conv2d (Conv2D)              (None, 5, 5, 1024)        4719616
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 3, 3, 1024)        9438208
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
dropout (Dropout)            (None, 9216)              0
_________________________________________________________________
dense (Dense)                (None, 256)               2359552
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 1028
=================================================================
Total params: 31,233,092
Trainable params: 16,518,404
Non-trainable params: 14,714,688
_________________________________________________________________
```

This represents my final architecture.

Final Comparison between Step 2 Model and Step 4 model.
The original architecture consisted of only VGG16, a flatten layer and then 2 dense layers. This gave good accuracy, but the validation accuracy was a bit lower than the training accuracy, which seemed to indicate there was some overfitting. Dropout layers were implemented as a way to improve testing accuracy, this was effective, but there seemed to be room for further improvements. I realized we only had around 6 million trainable parameters because the majority of the parameters come from the VGG16 base model, and these parameters are not trainable, since the model is frozen. I decided if I add 2 extra convolutional layers, this will give me more parameters to train my model, and could potentially increase the accuracy. This did prove to be true because the testing accuracy increased from 66% without extra convolutional layers to 72% with them. This is what brought me to my final model which is shown in “Step 4: Improving Accuracy” .

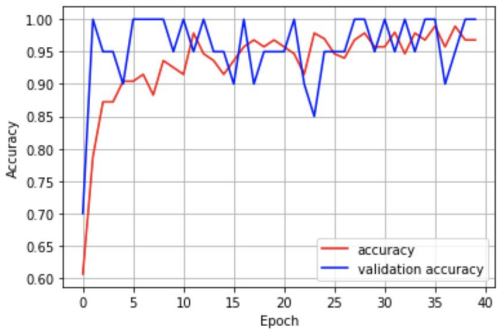
A final way to show concrete improvement between the initial model and the final model is the TSNE mapping.

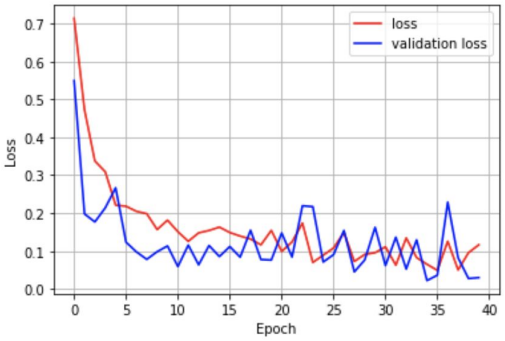


The TSNE for the initial model is on the left and the final model is on the right. Although both of these models do a good job clustering. It is obvious that the final model has tighter clusters, and leaks less into the other clusters. In the initial model, the pneumonia and covid clusters are almost beginning to blend together, whereas the final model shows very clear clusters between the Pneumonia variants, Covid, and Normal. This shows the final model does a better job of classifying these images into their respective groupings.

Accuracy and Loss

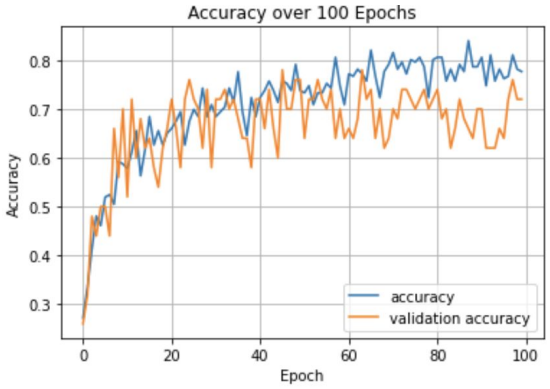
Task 1

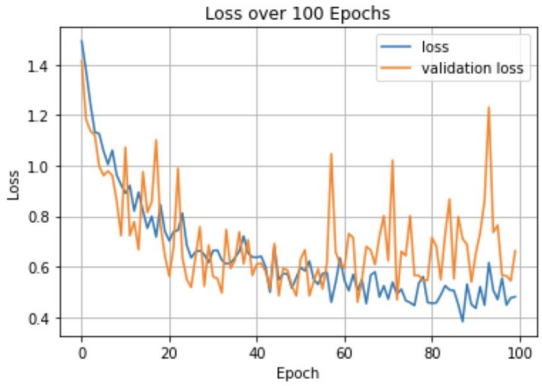




Here is the accuracy and loss for task 1 across 40 epochs. The validation accuracy is consistently between 0.9 and 1.0 the entire time, and even seems to be a bit higher than the training accuracy throughout the entire training process. This shows the model is very good at determining if someone has covid or not based on their xray. As for the loss we see a similar situation, the training and validation loss are very similar throughout training. These graphs show we can make predictions with around 90 - 95 percent accuracy and we reach this accuracy in less than 5 epochs.

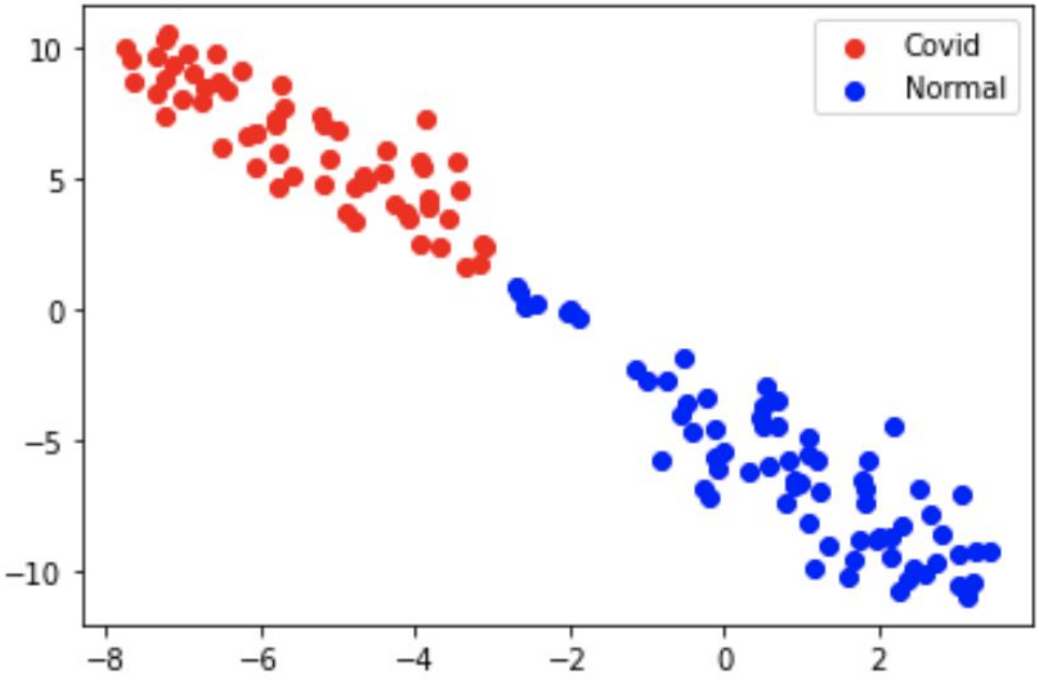
Task 2





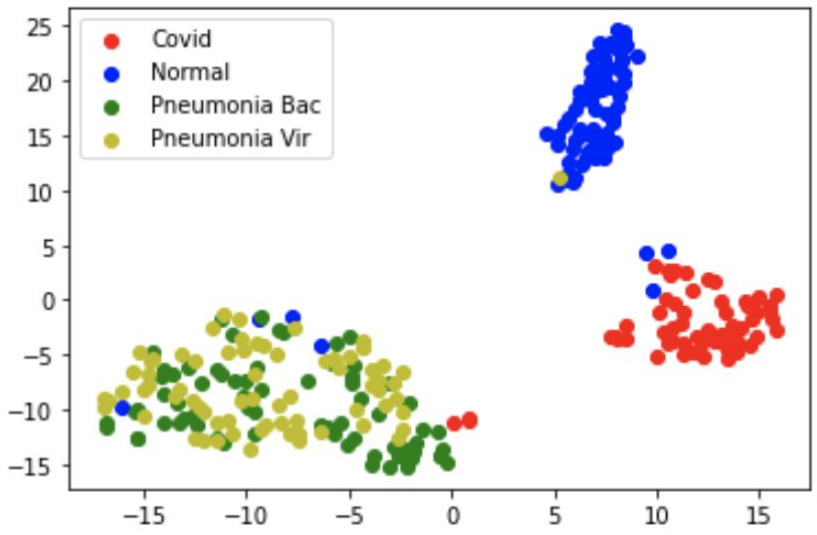
Here is the accuracy and loss for the task 2 model across 100 epochs. You can see the validation accuracy averages around 0.7 after it reaches epoch 20. The training accuracy at the end seems to go a bit higher than the validation accuracy peaking around 0.8. The validation loss almost perfectly maps the training loss until around epoch 60 where the validation loss begins getting some spikes in loss, which could just be the result of randomness in the model. The accuracy graph shows we can make predictions with around 0.7 accuracy when using this model after around 50 epochs of training.

TSNE
Task 1



The TSNE plot for Task 1 shows very clear clustering between Covid and Normal, which means that the model does a good job classifying the images that were input . This gives a really good visual representation of the two classifications. Based off of this image set, the model was able to perfectly cluster these two classes. It is also shown that these two classes are linearly separable, which means we are able to draw a straight line and perfectly separate the two clusters, and this shows the model has extracted valuable feature maps.

Task 2



The TSNE for task 2 shows very defined clusters, and although a few data points may make it into the incorrect clusters, it seems the majority are clustered correctly. This gives a great visual to see how well the model is performing and how well it is distinguishing between the different X-rays. I also notice that the clustering between Covid, Normal, and Pneumonia are very well defined, but it was more difficult for the model to tell the difference between the two types of pneumonia, and therefore those are clustered together. The reason for this is the two types of pneumonia probably have much more subtle differences compared to the differences in x rays of someone who has Covid vs. Normal.