Small Project

50.039 Deep Learning, Y2021

Group members:
Phang Teng Fone (1003296), Joey Yeo Kailing (1003846), Yeh Swee Khim (1003885)

Dataset and Dataloader

Distribution of Images among Classes

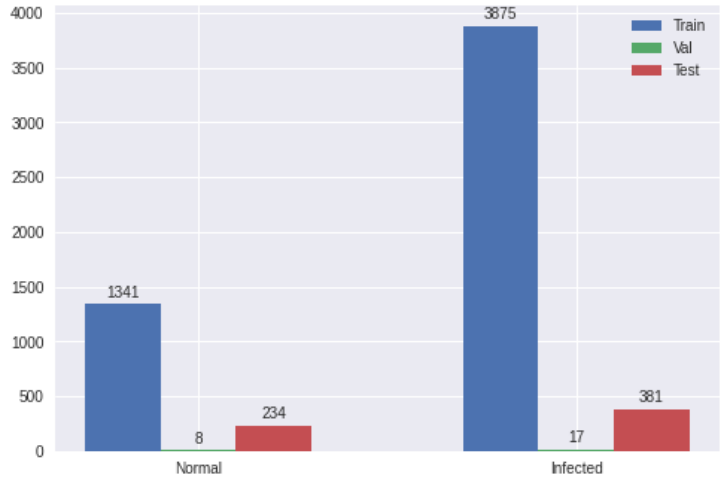


Figure 1. Distribution of Images between Normal and Infected

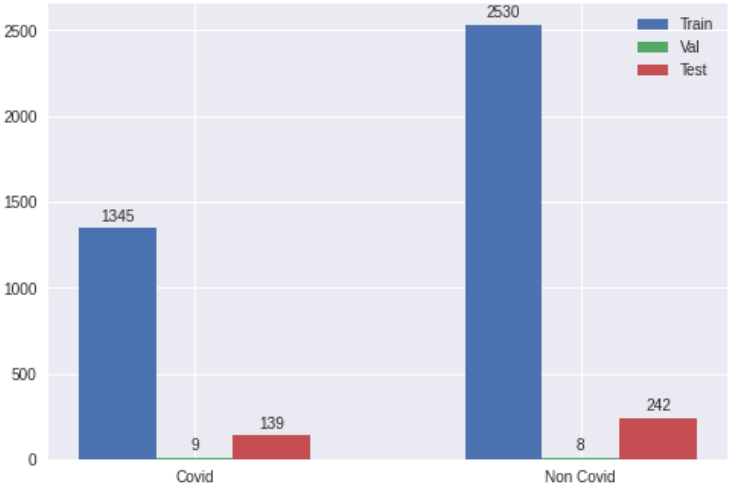


Figure 2. Distribution of Images between Covid and Non-Covid

The graph above shows the distribution of images among classes. It can be seen that the dataset is not balanced. There is significantly more training data for the infected as compared to normal (Figure 1.) and the non-covid as compared to covid (Figure 2.).

Data Processing Operations

With reference to the demo notebook, the normalization was done by dividing each pixel by 255. Neural networks process inputs with small weighted values as larger values can disrupt or slow down the learning process. Hence it is good practice to normalize the pixel values between 0 and 1, which can be achieved by dividing all pixel values by the largest pixel value of 255. This allows faster convergence when training our model, ensuring computational efficiency.

More sophisticated methods can be used such as dividing by the standard deviation of each pixel value. Another method is by subtracting the mean from each pixel and then dividing the result by the standard deviation. The distribution of such data would resemble a Gaussian curve centered at zero to ensure that each input pixel has a similar data distribution. When training our model, this allows each feature to have a similar range to achieve stable gradients.

The bonus task on data augmentation will be discussed at a later section.

Proposed Model

Differences between a Three-Class Classifier and Two Binary Classifiers Model

The three-class classifier model will be performing a multi-class classification where the classifier will receive training data for all three classes, normal , infected COVID and infected non-COVID. For testing, all testing dataset will be passed into this classifier, which will classify into the three classes.

The two binary classifiers model consists of two classifiers which are performing binary classifications. The first binary classifier is trained to classify between normal and infected. The second binary classifier is trained to classify between infected COVID and infected non-COVID. For testing, all testing dataset is passed into the first binary classifier, which classifies into normal and infected. The predicted infected instances are then passed into the second binary classifier which further classifies the instances into COVID or non-COVID.

Chosen Model - Two Binary Classifiers Model

The difference between the training instances in the normal and infected classes are obvious and would most likely be easy for the classifier to pick up features that can differentiate between the two classes. Whereas, the difference between the training instances in the infected COVID and infected non-COVID classes are subtle.

A three-class classifier will have to pick up features that can help to differentiate between normal, infected COVID and infected non-COVID, but a lot of the features for infected COVID and infected non-COVID overlap. Since a three-class classifier might only learn the general features that can help to distinguish between the three classes, it might not be able to detect and learn more detailed features that can help to differentiate between the infected COVID and infected non-COVID. The overlapping features might simply be ignored and will not be learnt by the model since they are not able to differentiate between the infected COVID and infected non-COVID from normal.

Whereas using a two binary classifiers model allows for the second classifier to focus and learn the subtle features that can help to differentiate between the infected COVID and infected non-COVID.

Our Model Architecture

Our model was inspired by the Densenet architecture. It consists of 2 convolutional layer blocks. Similar to the Dense Block, each convolutional layer block consists of the following layers and activation function (Figure 3.):

- Convolution layer allows for the learning of features from the images.
- Batch normalization layer makes optimization easier by smoothing the loss surface of the network. It helps to fix the mean and variance of the layer inputs by normalization, reducing internal covariate shifting. It also helps to improve the gradient flow through the network, reducing the need for dropout and allows for use of higher learning rate.
- ReLU is a non-linear activation function. The linearity in the positive dimension prevents non-saturation of gradients abstaining from the vanishing gradient problem although the negative dimension suffers from a gradient of zero.

In addition, our model also utilises skip connection to jump over a layer. This is done by concatenating the output of the second convolutional layer block with the output of the first convolutional layer block. When a layer receives feature maps from preceding layers, it can achieve feature reuse and improve backpropagation of the gradient, making it easier to train the model. This allows our model to be more robust and learn much faster.

The output from the concatenated layer will then be fed into our fully connected layer. The fully connected last layer allows our model to classify the output from our convolutional layer blocks into our desired number of classes, in our case two.

```
Net(
  (conv1): ModuleList(
    (0): Conv2d(1, 64, kernel_size=(3, 3), stride=(2, 2))
    (1): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (cnn_layers): ModuleList(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (output): Linear(in_features=147968, out_features=2, bias=True)
)
```

Figure 3. Our model architecture
(cnn_layers): one convolutional layer block
(output): fully connected layer

Evaluation Metrics

We have decided to use F1 score as our primary evaluation metric. This is because the F1 score takes into account the class imbalance. It is the harmonic mean of Precision and Recall, which will give us a better evaluation of the incorrectly classified classes and gives a better measure of the incorrectly classified cases.

Whereas an accuracy metrics will favour the class with the most instances in the dataset. This means that a model would score very well by always predicting '1' when evaluated on a dataset with majority of class '1', but this biased model will perform badly in real life.

Model Parameters

Cross entropy loss has achieved great success and been the norm for training many classification problems. It is a good loss function for classification problems since it can help to minimize the distance between the two probability distributions: the predicted output from our model and the actual label. The cross-entropy loss function is optimized to make the features extracted from the neural network more representative.[1] For pytorch, this criterion combines log-softmax and negative log-likelihood (NLL) in one single class. Since we have implemented the log-softmax as our model output, we used the NLL loss as our choice of loss function to simulate the cross entropy loss function.

In general, it is observed that Adam is one of the recent and popular variants of gradient descent that is able to perform reasonably well in practice as compared to the other adaptive learning methods.[2] Adam computes adaptive learning rates for each parameter by utilizing both first and second moments. It also accumulates the exponentially decaying average of past gradients similar to SGDM (Stochastic Gradient Descent with momentum) as the first moment. Moreover, it also accumulates the exponentially decaying average of squares of past gradients similar to AdaDelta and RMSProp as a second moment. Given the features of the Adam optimizer, it is able to rectify the vanishing learning rate and high variance. From a research paper on image classification using convolutional neural networks, it mentions that Adam is the best performing optimizer that can achieve the lowest mean squared error for shallow net architecture.[3] Given the benefits of the Adam optimizer and since our model has a shallow net architecture, we decided to use the Adam as our choice of optimizer.

To find the best combination to initialize our model parameters for our batch size and learning rate, we created a function to test different ranges of hyperparameter values. We tested with different batch sizes of 16, 32, 64 and learning rates of 0.1, 0.001, and 0.0001 as shown in the table below. Given that a batch size of 64 and learning rate of 0.0001 is able to achieve the best F1 score of 0.707, we used it to initialize and train our model. Moreover, to prevent our model from overfitting, we also introduced early stopping to stop training at the point when the performance on the validation set starts to degrade.

---

[1] https://www.mdpi.com/2076-3417/10/8/2950/pdf
[2] https://arxiv.org/pdf/1909.11015
[3] https://www.mdpi.com/2313-433X/6/9/92/pdf

Batch size / Learning Rate	0.1	0.001	0.0001
16	F1 Score: 0.417	F1 Score:0.597	F1 Score: 0.493
32	F1 Score: 0.238	F1 Score: 0.658	F1 Score: 0.265
64	F1 Score: 0.394	F1 Score: 0.590	F1 Score: **0.707**

Table 1: Fine Tuned Parameters

Batch size determines how fast a network converges. A lower batch size typically means a slower convergence, as the weights have to be updated more frequently in one epoch. Although a larger batch size typically results in faster convergence, too large a batch size might cause overfitting and numerical instability.

Learning rate is also important. A learning rate that is very small will take a long time for the network to converge. Whereas a learning rate that is very large might cause the loss to fluctuate significantly, as the step taken is too large and will not be able to converge to a minimum. The learning rate is also what motivated us to choose Adam as an optimiser and not SGD. SGD typically requires a good fine tuned learning rate to perform well. Whereas Adam is typically more adaptable and works well in most cases even if the learning rate is not as fine tuned.

Output Results (Without Data Augmentation)

After calculating the best fine tuning parameter from Table 1, we used 10 epoch, 64 batchsize and 0.0001 learning rate.

Testing Accuracy	0.400
F1 Score	0.747
Confusion Matrix	

	true normal	true covid	true noncovid
predicted normal	8	1	0
predicted covid	0	1	3
predicted noncovid	0	6	6

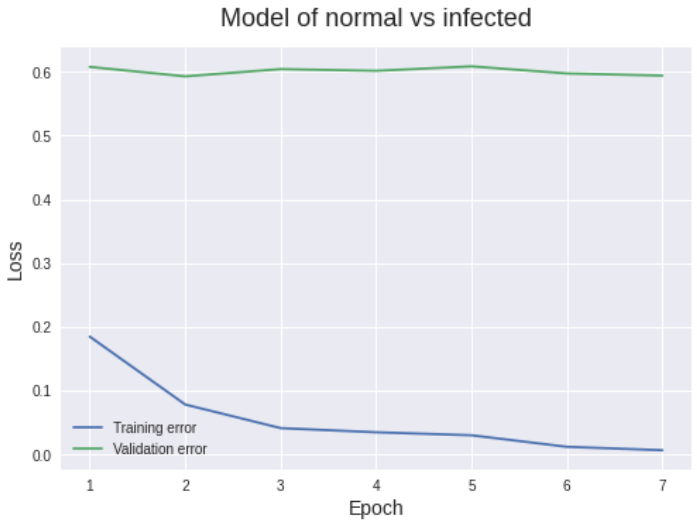


Figure 4. Training/ Validation loss per epoch on first binary classifier (normal vs infected)

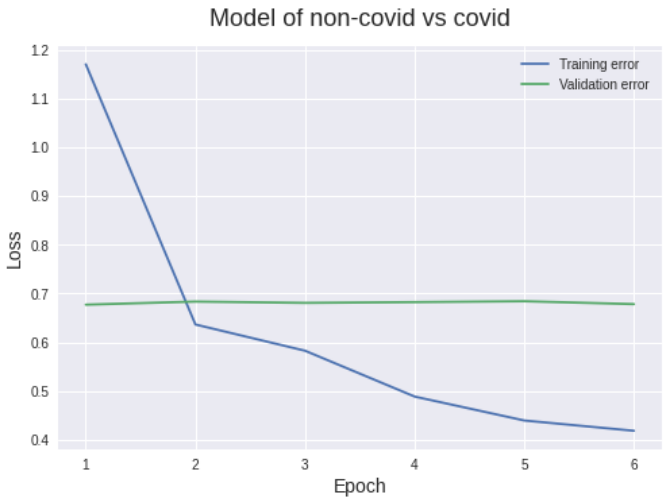
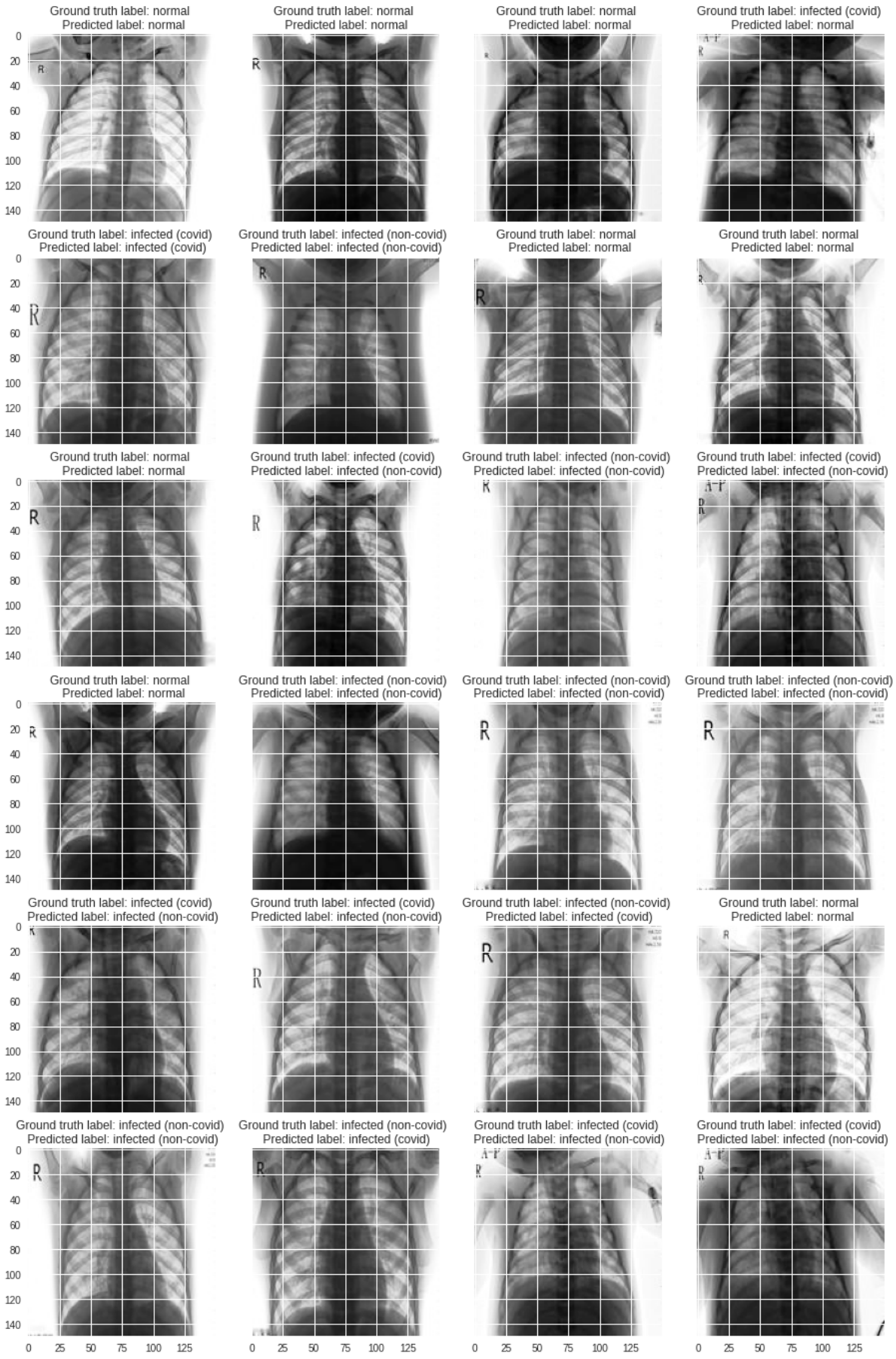


Figure 5. Training/ Validation loss per epoch on second binary classifier (COVID vs non-COVID)

Validation set pictures, with predicted and ground truth labels.
Average performance 15/25 = 60.0%



Bonus: Data Augmentation

Given that most machine learning models are subject to frequency bias, where they place more emphasis on learning from the data observations. In the case of an imbalance dataset, this might cause the model to perform and learn better in one class as compared to the other.

To address data imbalance for our first binary classifier training on the 'normal' and 'infected' class dataset, we performed data augmentation on the 'normal' class training dataset to increase the number of data we have in the 'normal' class. We vertically flipped the training instances in the 'normal' class to achieve twice as many data from before. This allows the data distribution between the 'normal' and 'infected' class dataset to be balanced.

Since we only want to increase the number of training data in our 'normal' class without applying data augmentation on any other training instances, we only performed the vertical flip augmentation as it is the only data augmentation technique that keeps the aspect ratio, colour, saturation, contrast and scale constant.

Similarly for the second binary classifier training on the 'COVID' and 'non-COVID' class dataset, we performed the same data augmentation technique on the 'COVID' class training dataset to increase the number of data we have in the 'COVID' class.

Using the same configurations for our binary classifiers, we integrated our newly distributed training dataset and the results are collated as shown below. From the results, we can see that the F1 score improved after increasing the number of 'normal' and 'COVID' training data using data augmentation, with more correctly predicted infected data as shown in the confusion matrix.

Output Results (With Data Augmentation)

After calculating the best fine tuning parameter from Table 1, we used 10 epoch, 64 batchsize and 0.0001 learning rate. With data augmentation, our testing accuracy and F1 score increased.

Testing Accuracy	0.440
F1 Score	0.810
Confusion Matrix	

	true normal	true covid	true noncovid
predicted normal	8	0	0
predicted covid	0	8	8
predicted noncovid	0	0	1

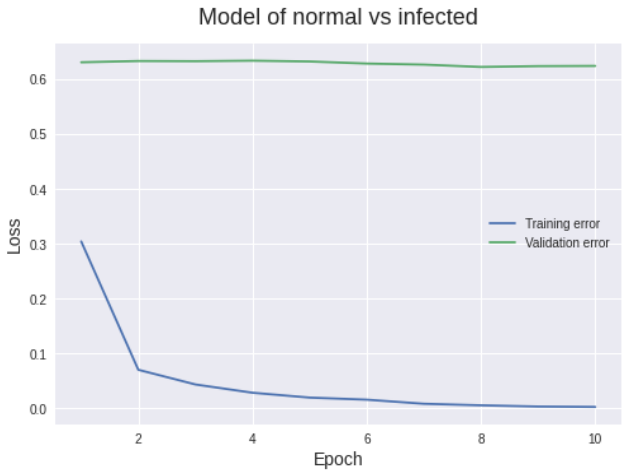


Figure 6. Training/ Validation loss per epoch on first binary classifier (normal vs infected)

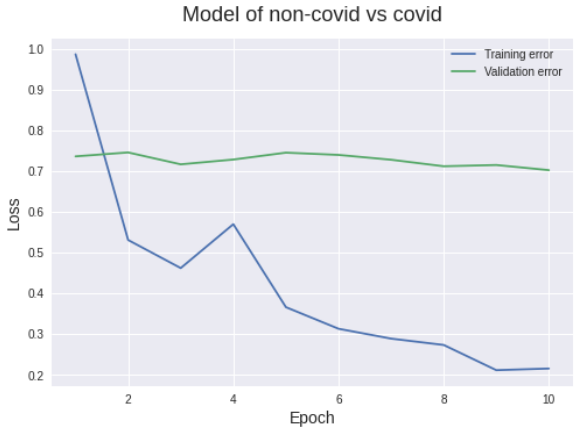
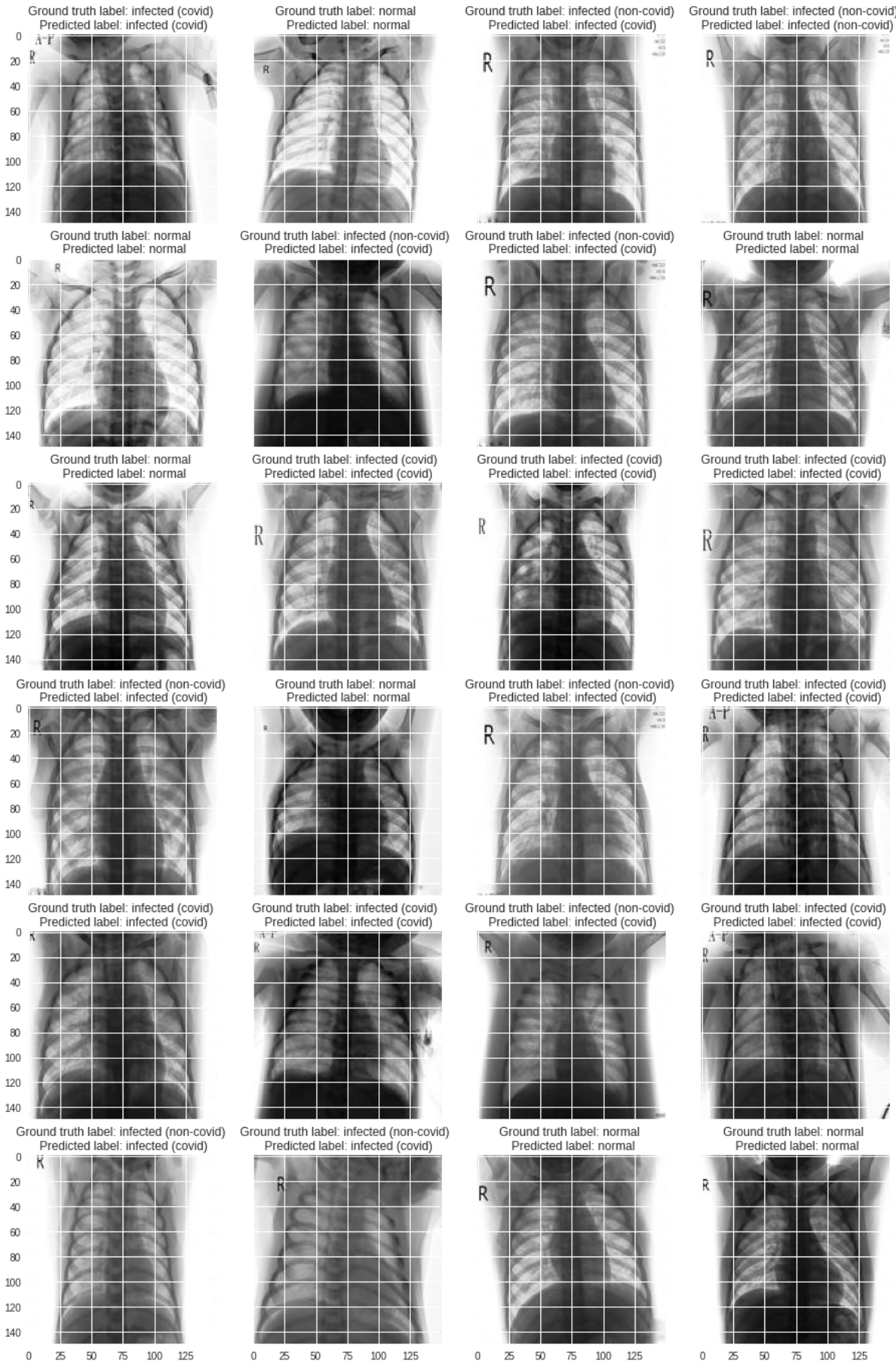


Figure 7. Training/ Validation loss per epoch on second binary classifier (COVID vs non-COVID)

Validation set pictures, with predicted and ground truth labels.
Average performance 17/25 = 68.0%



Final Thoughts

It is expected to be more difficult to differentiate between non-COVID and COVID x-rays as compared to normal and infected x-rays. The dataset for normal and infected have very distinct differences that is distinguishable by the human eye. The images in the infected dataset have whiteness in the lungs that are absent in the normal dataset. Meanwhile for infected COVID and non-COVID, the images are almost indistinguishable by the human eye. This implies that there will be more overlapping features between the infected COVID and non-COVID as compared to the normal. Therefore, for our proposed model we decided to use the two binary classifiers model as the second binary classifier will be focused on learning the distinct features that can help to differentiate between the infected COVID and non-COVID that cannot be seen by the naked eye.

Apart from the accuracy metric, the confusion matrix for the models are also computed. Accuracy can be a useful measure if we have a uniformly distributed dataset. However, with imbalance data, the standard classification metrics will not adequately represent our model performance. Given imbalance data, the model might be achieving a high overall accuracy by choosing the more common class. Although our dataset might be unbalanced, by using a confusion matrix, we can overlook the performance for each class individually.

Depending on the application, it might be better to have a model with low true negatives/false positives rates on certain classes as compared to the high overall accuracy. In our case, it is better to have a high false positive (as compared to the false negative) for COVID and non-COVID classification. This is because COVID is an extremely dangerous disease which can be fatal and highly contagious thus having a high false positive rate will allow more patients to be tested physically instead of relying on the prediction of a neural network. It is better to be safe than sorry, and to "over classify" patients as COVID than wrongly classifying them as non-COVID.

Bonuses

Implementing and discussing a learning rate scheduler and discussing its appropriate choice of parameters and benefits for your model.

For our model, we implemented the 1cycle learning rate policy. This learning rate scheduler is experimented on and said to converge the network quickly. This scheduler changes the learning rate after every batch, and oscillates the learning rate from a maximum value to a lower value, back to the maximum value, and so on. The motivation for this is that at steps with large learning rates, it works as a regulariser, hence keeping the network from overfitting. Additionally, the oscillation of learning rate per step helps the network to avoid regions of steep loss. Thus, having a higher chance of landing in a flatter minima, which is generally more stable and better.[4]

Since this scheduler helps to lower the chance of overfitting the network and that learning rates are set to oscillate in search for a more stable minima, we used more epochs and allowed early stopping to stop training should there be signs of overfitting (stagnant validation losses).

Output result:

Without scheduler

When training without a scheduler, we used 20 epoch, 64 batchsize and 0.0001 learning rate.

Testing Accuracy	0.600
F1 Score	0.741
Confusion Matrix	

	true normal	true covid	true noncovid
predicted normal	8	1	2
predicted covid	0	2	1
predicted noncovid	0	5	6

[4] https://towardsdatascience.com/finding-good-learning-rate-and-the-one-cycle-policy-7159fe1db5d6

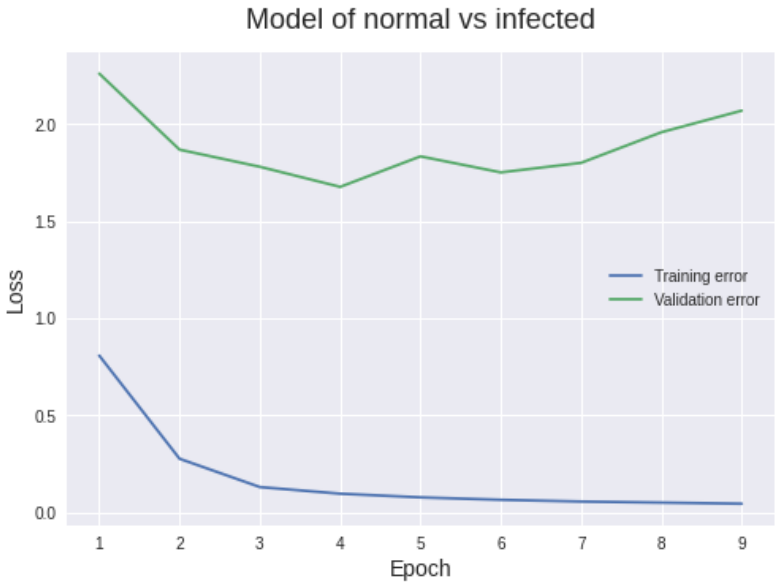


Figure 8. Training/ Validation loss per epoch on first binary classifier (normal vs infected)

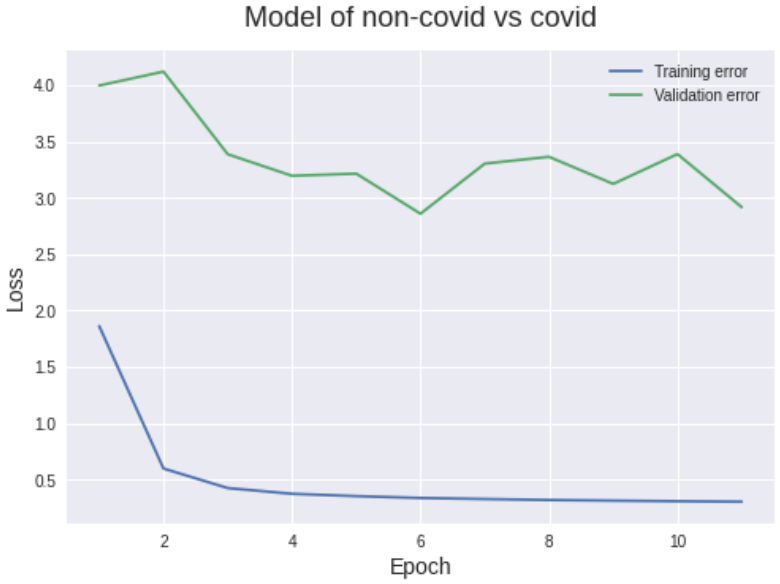


Figure 9. Training/ Validation loss per epoch on first binary classifier (normal vs infected)

With scheduler

When training with a scheduler, we used 20 epoch, 64 batchsize and 0.0001 learning rate.

Testing Accuracy	0.680
F1 Score	0.805
Confusion Matrix	

	true normal	true covid	true noncovid
predicted normal	8	2	0
predicted covid	0	3	2
predicted noncovid	0	3	7

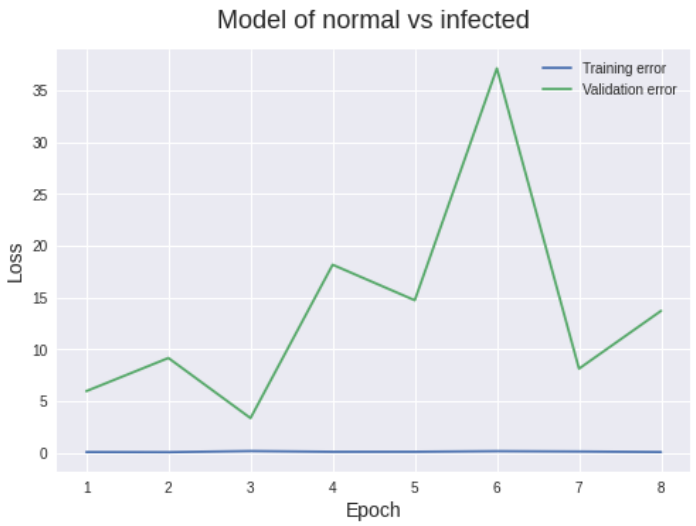


Figure 10. Training/ Validation loss per epoch on first binary classifier (normal vs infected)

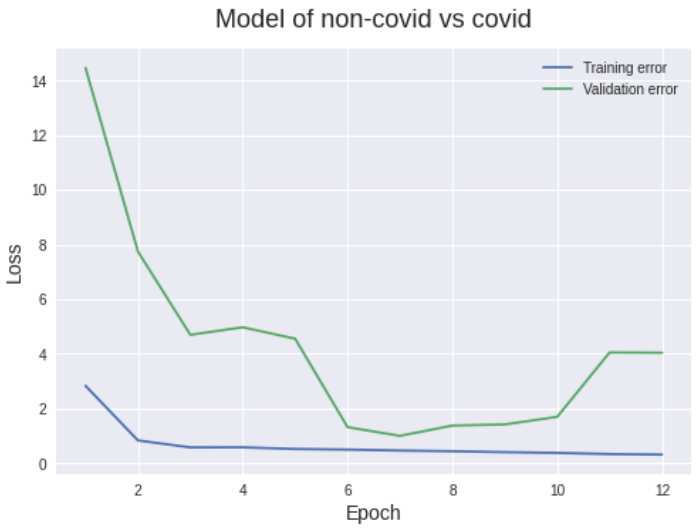
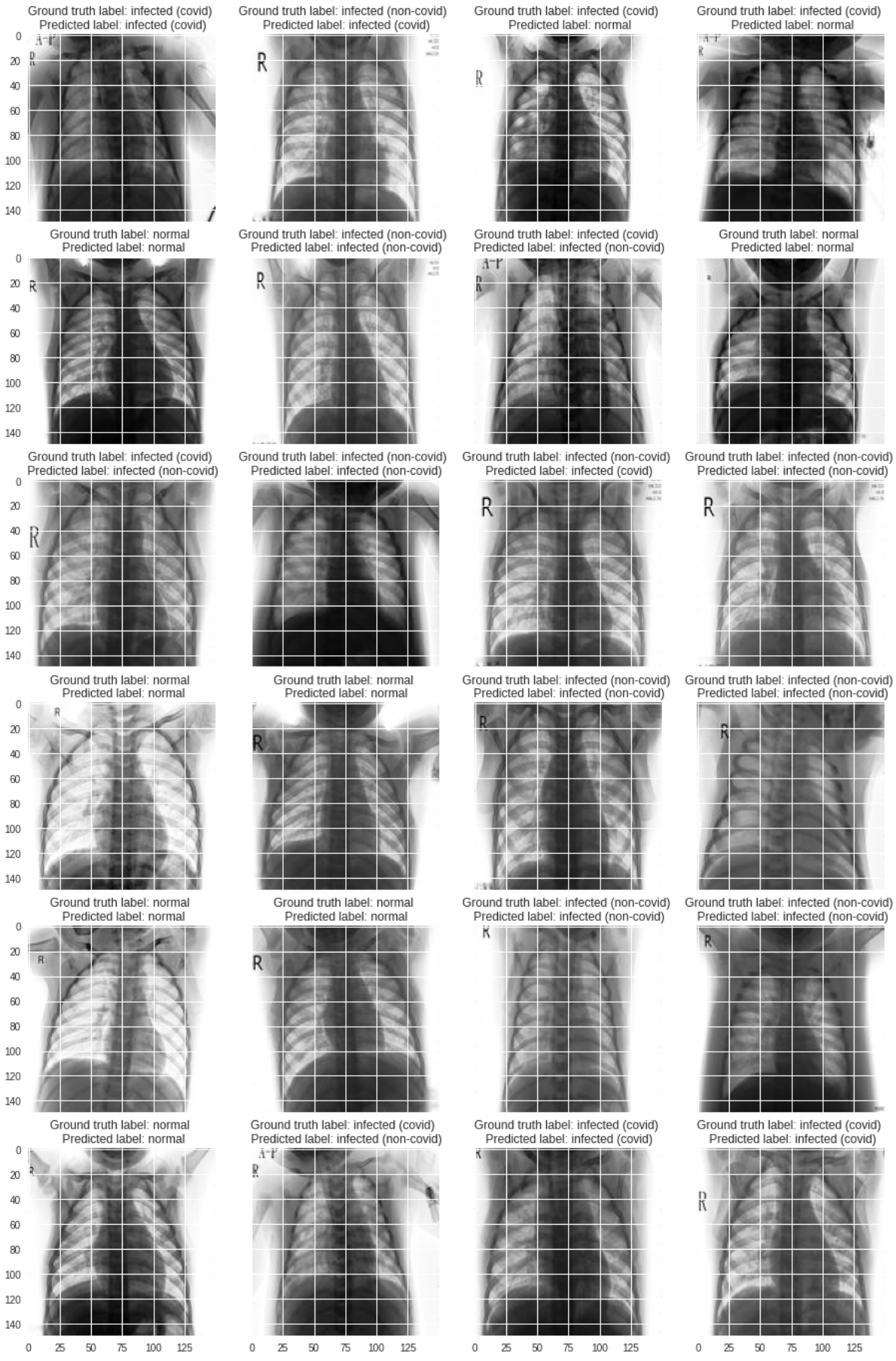


Figure 11. Training/ Validation loss per epoch on first binary classifier (normal vs infected)

Validation set pictures, with predicted and ground truth labels.
Average performance 18/25 = 72.0%



Briefly look up online how doctors diagnose infections based on x-rays. Does our AI seem to be able to reproduce this behavior correctly? Show typical samples of the dataset on which your AI failed and discuss what might have been the reasons.

From an article, a group of specialised medical professionals gave advice on how to look for changes on chest radiograph that may be suggestive of COVID-19 pneumonia.[5] Similar to other pneumonias, COVID-19 pneumonia causes the density of the lungs to decrease. (Figure 12.) Depending on the severity of the pneumonia, this reflects the whiteness in the lungs on the radiography, obscuring the lung markings that should be normally seen as shown in the image below. However, it might also be delayed in appearing or absent.

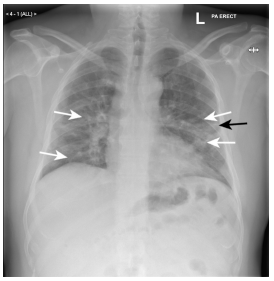


Figure 12. Radiography of a patient with COVID-19 pneumonia

From our test output predicted by both models trained with and without data augmentation as shown in the sections above, while we are able to classify between normal and infected pretty well, we are unable to correctly classify most infected COVID/non-COVD cases. Below shows 2 test instances that have been predicted by our model. Although one of the images has a ground truth label of infected non-COVID and the other with infected COVID, they have both been classified under infected COVID. (Figure 13.)

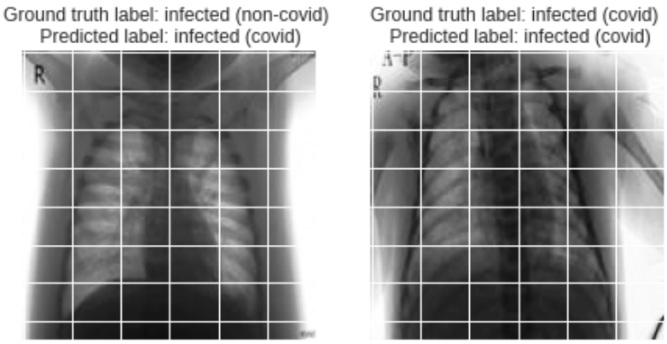


Figure 13. Test instances predicted by our model (includes data augmentation)

There could be multiple reasons that cause our model to fail. Firstly, the symptoms are similar for COVID-19 pneumonia and other pneumonias as mentioned in the article above. Depending on the severity of either pneumonias, the radiography might be very similar for the two and hence making it very difficult to distinguish between them. Secondly, the whiteness in the lungs might be delayed in appearing or absent from the radiography. This might cause some confusion to our model when it is learning the features for infected COVID and non-COVID cases.

[5] https://doi.org/10.1136/bmj.m2426