10.039 Deep Learning Small Project

Team

Keith Ng 1003515
Li Yuxuan 1003607
Ng Jia Yi 1003696

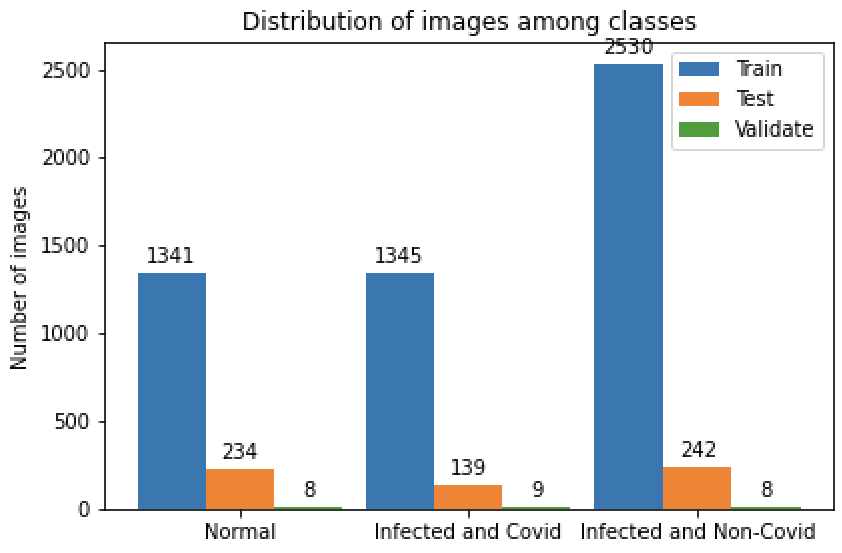
Introduction

In this project, we attempt to design a deep learning model, whose task is to assist with the diagnosis of pneumonia, for COVID and non-COVID cases, by using X-ray images of patients. We engineered a lightweight convolutional neural network model that can collectively train/test the given x-ray scans. In this report, we will cover our design process and the challenges that come along with it.

Proposed Methodology

Selected Classifier

Before we decide between multi-class classifiers or 2 binary classifiers, we first take a look at the composition of our dataset.



A multi-class classifier usually favours the largest classes and in this case, it is pretty obvious that there is a bias within our training dataset. The infected, non-covid case, has almost 2x the number of images as compared to the normal and infected and covid training datasets each. Due to the inherit biasness of the given dataset, we have chosen to use the 2 binary classifier instead of the multi-class classifier.

Image Pre-Processing

We aim to reduce noise from our images with preprocessing techniques. One of the fundamental techniques is edge detection. It locates the borders of objects in images by identifying disjointedness in brightness. Edge detection methods can produce a new set of features. Edges are noted by serious or local changes in the image, and edges normally occur on the borders between two distinct areas in an image. Edges are important features for analysing digital images.

The edge detection method we applied here is called Enhanced Canny edge detection(Ceed-Canny). Our code implementation goes as follows

```
def ceed_canny(image):
    im = io.imread(image)
    enh = enhance_contrast(im, disk(3))
    edges = feature.canny(enh)
    final = img_as_ubyte(edges)
    return final
```

The image first undergoes morphological contrast enhancement. Morphology image processing are processes that manipulate images based on structuring elements. Morphological operations accept an image as input, apply a structuring element to the image, and outputs the resulting image. The pixel of the output image is determined from a comparison between the corresponding pixel of the input image with its neighbours.

Next, it undergoes the canny edge detection. The Canny filter is a multi-stage edge detector. First, noise reduction was performed by implementing Gaussian smoothing. Then, the intensity gradient of the image is computed so that non- maximum suppression can be employed. Likely edges are reduced to 1-pixel curves by discarding non- maximum pixels of the gradient magnitude. Lastly, hysteresis thresholding is performed to determine whether or not to keep the edge pixels.

The results of image preprocessing are as follows.

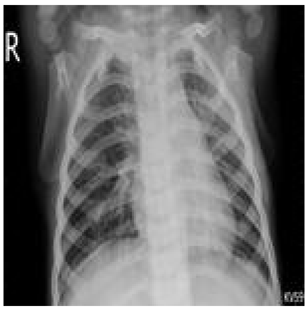
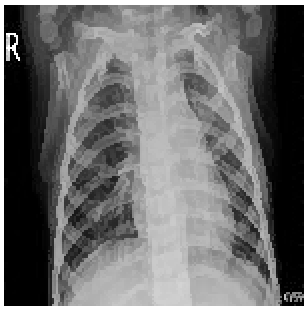
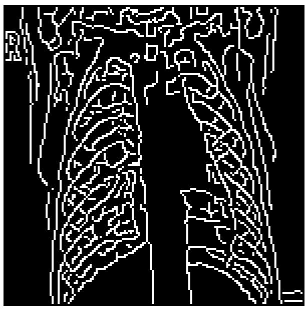
ORIGINAL	**MCE**	**CANNY**
		

Figure: Step by step image preprocessing result

CLASS	**BEFORE**	**AFTER**

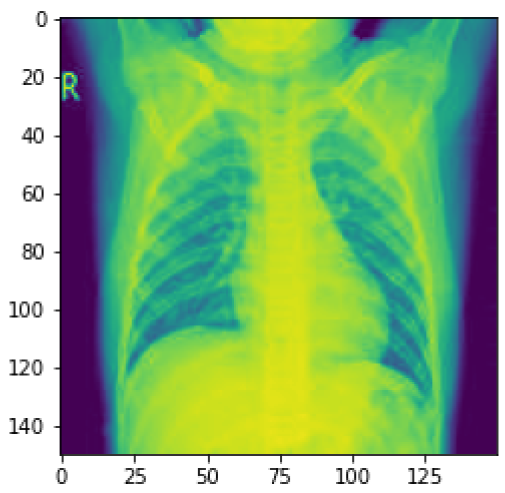
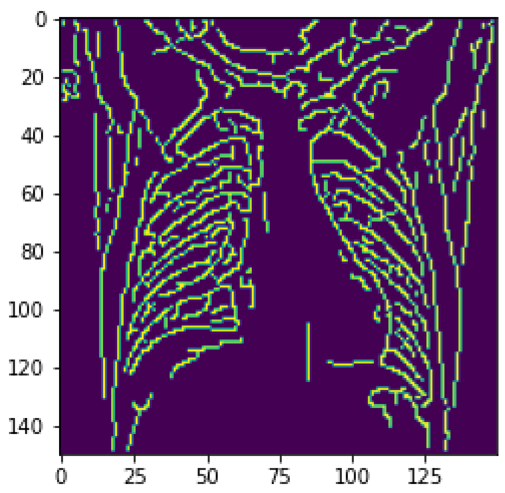
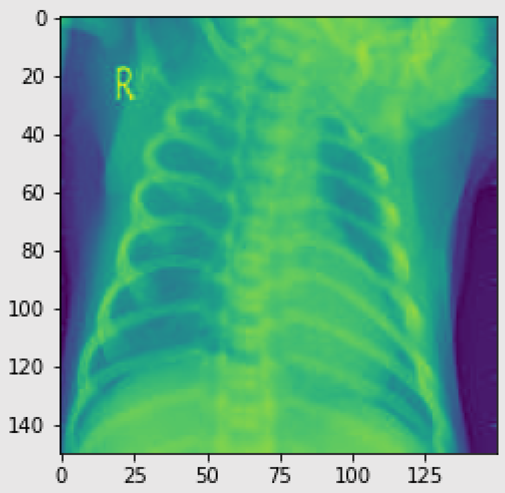
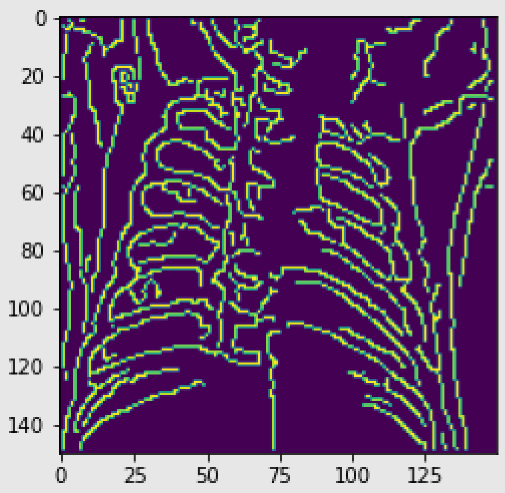
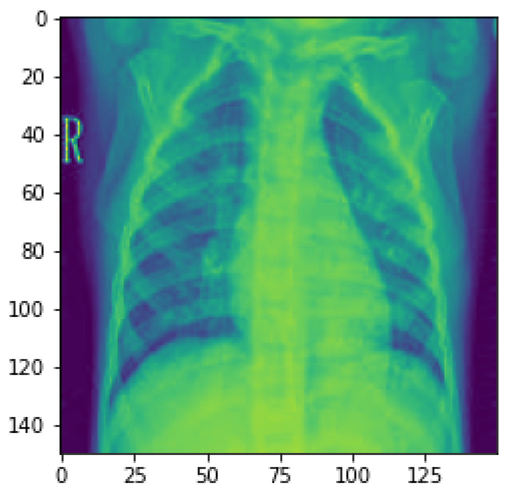
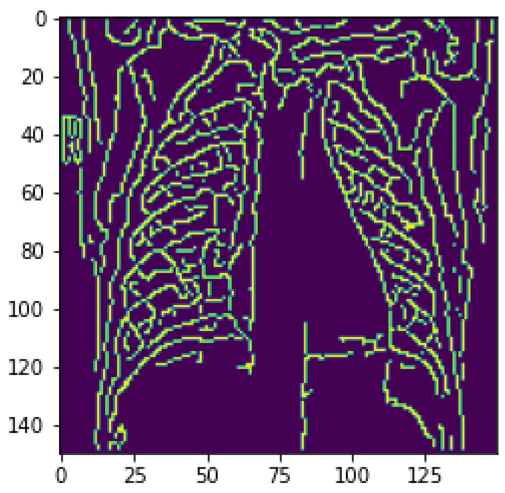
CLASS	BEFORE	AFTER
Normal		
Covid		
Non-Covid		

Figure: Before vs After per class

From the contour lines, we can clearly differentiate the three different classes as most noises are removed and the outlines are clearly presented.

Image Augmentation

We decided that the given dataset was too small and we would not be able to train our model sufficiently. To cope with this, data augmentation was utilized to expand our training and validation sets. The following code depicts our code implementation.

```
train_transformer = transforms.Compose([transforms.RandomApply([
        transforms.RandomRotation(20, fill=(0,)),
        ],0.7)
  ])
```

Each imaged passed to our transformer has a 70% chance of going through a **random rotation**. We initially implemented other techniques such as horizontal flipping and zooming, but we realized that did not work so well.

```
if self.groups[0] == "train":
    for _ in range(2):
        img_Temp = img
        for _ in range(5):
            img_Temp = train_transformer(img_Temp)
            image = np.asarray(img_Temp) / 255
            image =
transforms.functional.to_tensor(np.array(image)).float()
            imgs.append(image)
```

Every image will go through the transformer 2x5 times, and at each iteration, we save the resultant image. Hence, every 1 image produces 10 augmented images in total. Below is an example of how 10 augmented images(derived from the same original image) turns out.

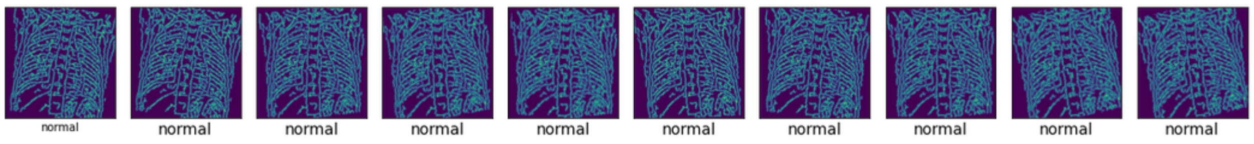
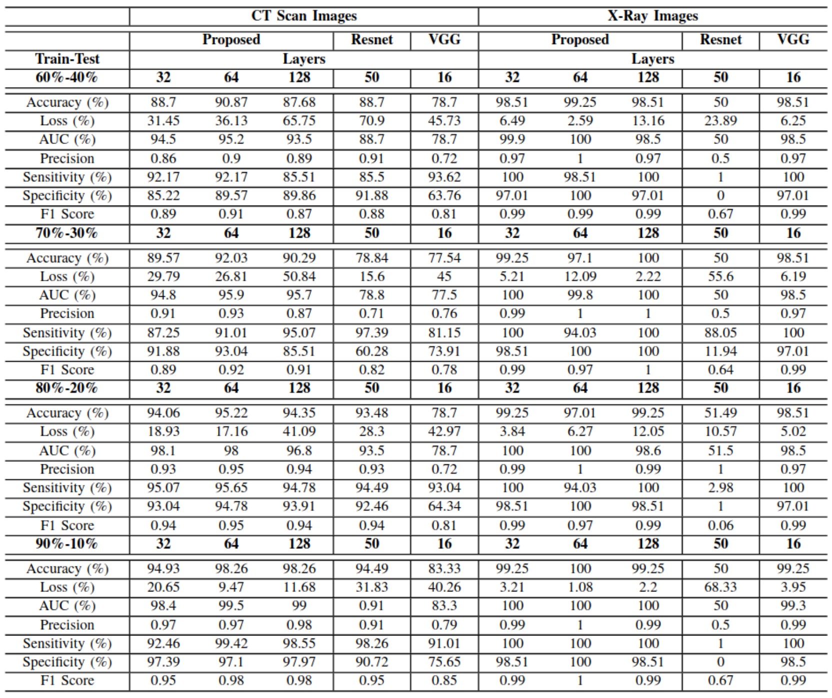


Figure: Resultant images undergone Data Augmentation

Selected Model

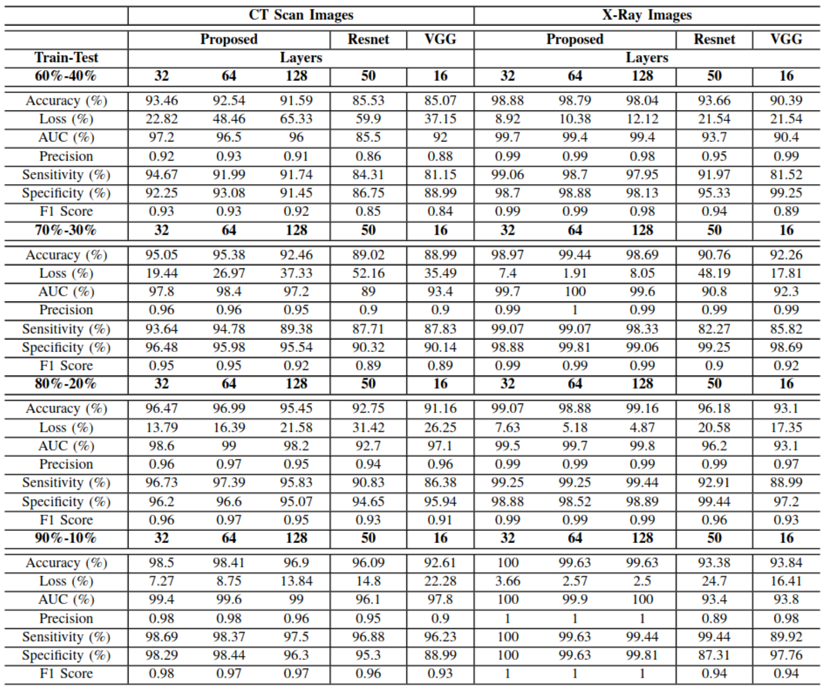
We started off by researching about the choices of models on doing COVID-19 chest X-ray image classifications. From one of the research papers that we found, we found that the comparison results show that the performance of VGG16 model on both original and augmented X-Ray dataset is better than Resnet50. (https://www.biorxiv.org/content/biorxiv/early/2020/07/17/2020.07.15.205567.full.pdf) For Train-Test ratio 90%-10%, for original, VGG16 achieves 99.25% accuracy whereas Resnet50 achieves 50% accuracy. For augmented, VGG16 achieves 93.84% accuracy whereas Resnet50 achieves 93.38% accuracy.

TABLE I: COVID-19 screening performance in Original images with CT Scan and X-Ray datasets

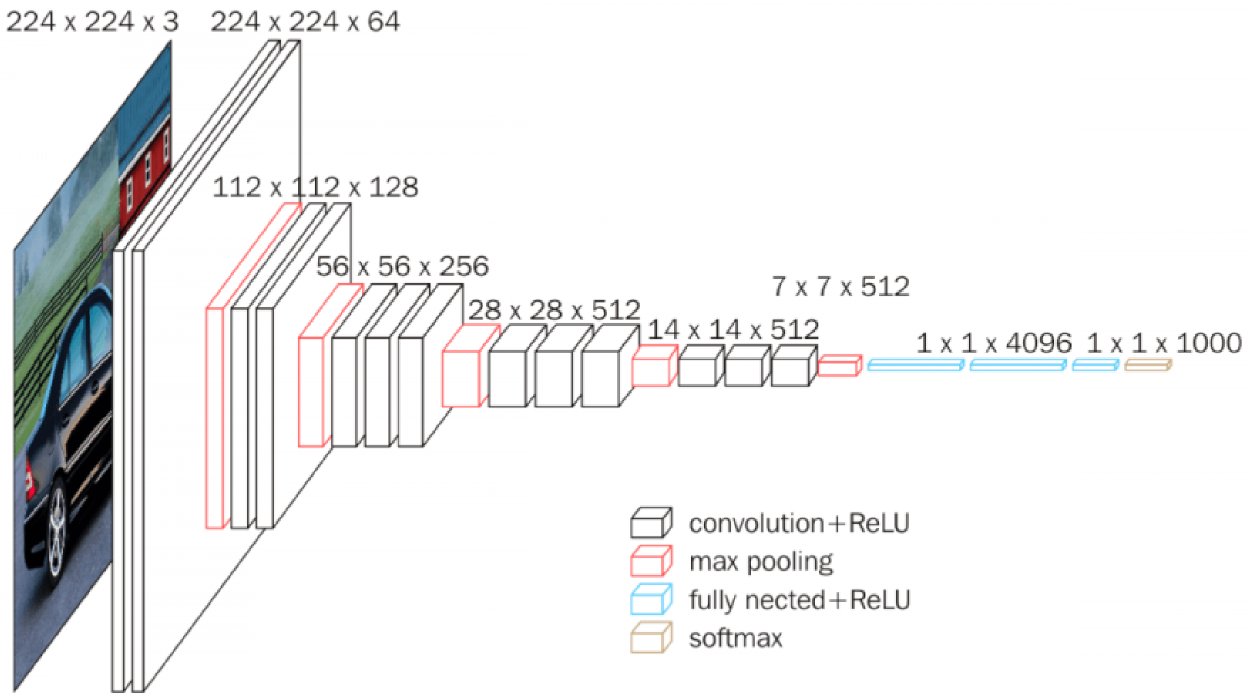


	CT Scan Images					X-Ray Images				
	Proposed			Resnet	VGG	Proposed			Resnet	VGG
Train-Test	Layers					Layers				
60%-40%	**32**	**64**	**128**	**50**	**16**	**32**	**64**	**128**	**50**	**16**
Accuracy (%)	88.7	90.87	87.68	88.7	78.7	98.51	99.25	98.51	50	98.51
Loss (%)	31.45	36.13	65.75	70.9	45.73	6.49	2.59	13.16	23.89	6.25
AUC (%)	94.5	95.2	93.5	88.7	78.7	99.9	100	98.5	50	98.5
Precision	0.86	0.9	0.89	0.91	0.72	0.97	1	0.97	0.5	0.97
Sensitivity (%)	92.17	92.17	85.51	85.5	93.62	100	98.51	100	1	100
Specificity (%)	85.22	89.57	89.86	91.88	63.76	97.01	100	97.01	0	97.01
F1 Score	0.89	0.91	0.87	0.88	0.81	0.99	0.99	0.99	0.67	0.99
70%-30%	**32**	**64**	**128**	**50**	**16**	**32**	**64**	**128**	**50**	**16**
Accuracy (%)	89.57	92.03	90.29	78.84	77.54	99.25	97.1	100	50	98.51
Loss (%)	29.79	26.81	50.84	15.6	45	5.21	12.09	2.22	55.6	6.19
AUC (%)	94.8	95.9	95.7	78.8	77.5	100	99.8	100	50	98.5
Precision	0.91	0.93	0.87	0.71	0.76	0.99	1	1	0.5	0.97
Sensitivity (%)	87.25	91.01	95.07	97.39	81.15	100	94.03	100	88.05	100
Specificity (%)	91.88	93.04	85.51	60.28	73.91	98.51	100	100	11.94	97.01
F1 Score	0.89	0.92	0.91	0.82	0.78	0.99	0.97	1	0.64	0.99
80%-20%	**32**	**64**	**128**	**50**	**16**	**32**	**64**	**128**	**50**	**16**
Accuracy (%)	94.06	95.22	94.35	93.48	78.7	99.25	97.01	99.25	51.49	98.51
Loss (%)	18.93	17.16	41.09	28.3	42.97	3.84	6.27	12.05	10.57	5.02
AUC (%)	98.1	98	96.8	93.5	78.7	100	100	98.6	51.5	98.5
Precision	0.93	0.95	0.94	0.93	0.72	0.99	1	0.99	1	0.97
Sensitivity (%)	95.07	95.65	94.78	94.49	93.04	100	94.03	100	2.98	100
Specificity (%)	93.04	94.78	93.91	92.46	64.34	98.51	100	98.51	1	97.01
F1 Score	0.94	0.95	0.94	0.94	0.81	0.99	0.97	0.99	0.06	0.99
90%-10%	**32**	**64**	**128**	**50**	**16**	**32**	**64**	**128**	**50**	**16**
Accuracy (%)	94.93	98.26	98.26	94.49	83.33	99.25	100	99.25	50	99.25
Loss (%)	20.65	9.47	11.68	31.83	40.26	3.21	1.08	2.2	68.33	3.95
AUC (%)	98.4	99.5	99	0.91	83.3	100	100	100	50	99.3
Precision	0.97	0.97	0.98	0.91	0.79	0.99	1	0.99	0.5	0.99
Sensitivity (%)	92.46	99.42	98.55	98.26	91.01	100	100	100	1	100
Specificity (%)	97.39	97.1	97.97	90.72	75.65	98.51	100	98.51	0	98.5
F1 Score	0.95	0.98	0.98	0.95	0.85	0.99	1	0.99	0.67	0.99

TABLE II: COVID-19 screening performance in Augmented images with CT Scan and X-Ray datasets



	CT Scan Images					X-Ray Images				
	Proposed			Resnet	VGG	Proposed			Resnet	VGG
Train-Test	Layers					Layers				
60%-40%	**32**	**64**	**128**	**50**	**16**	**32**	**64**	**128**	**50**	**16**
Accuracy (%)	93.46	92.54	91.59	85.53	85.07	98.88	98.79	98.04	93.66	90.39
Loss (%)	22.82	48.46	65.33	59.9	37.15	8.92	10.38	12.12	21.54	21.54
AUC (%)	97.2	96.5	96	85.5	92	99.7	99.4	99.4	93.7	90.4
Precision	0.92	0.93	0.91	0.86	0.88	0.99	0.99	0.98	0.95	0.99
Sensitivity (%)	94.67	91.99	91.74	84.31	81.15	99.06	98.7	97.95	91.97	81.52
Specificity (%)	92.25	93.08	91.45	86.75	88.99	98.7	98.88	98.13	95.33	99.25
F1 Score	0.93	0.93	0.92	0.85	0.84	0.99	0.99	0.98	0.94	0.89
70%-30%	**32**	**64**	**128**	**50**	**16**	**32**	**64**	**128**	**50**	**16**
Accuracy (%)	95.05	95.38	92.46	89.02	88.99	98.97	99.44	98.69	90.76	92.26
Loss (%)	19.44	26.97	37.33	52.16	35.49	7.4	1.91	8.05	48.19	17.81
AUC (%)	97.8	98.4	97.2	89	93.4	99.7	100	99.6	90.8	92.3
Precision	0.96	0.96	0.95	0.9	0.9	0.99	1	0.99	0.99	0.99
Sensitivity (%)	93.64	94.78	89.38	87.71	87.83	99.07	99.07	98.33	82.27	85.82
Specificity (%)	96.48	95.98	95.54	90.32	90.14	98.88	99.81	99.06	99.25	98.69
F1 Score	0.95	0.95	0.92	0.89	0.89	0.99	0.99	0.99	0.9	0.92
80%-20%	**32**	**64**	**128**	**50**	**16**	**32**	**64**	**128**	**50**	**16**
Accuracy (%)	96.47	96.99	95.45	92.75	91.16	99.07	98.88	99.16	96.18	93.1
Loss (%)	13.79	16.39	21.58	31.42	26.25	7.63	5.18	4.87	20.58	17.35
AUC (%)	98.6	99	98.2	92.7	97.1	99.5	99.7	99.8	96.2	93.1
Precision	0.96	0.97	0.95	0.94	0.96	0.99	0.99	0.99	0.99	0.97
Sensitivity (%)	96.73	97.39	95.83	90.83	86.38	99.25	99.25	99.44	92.91	88.99
Specificity (%)	96.2	96.6	95.07	94.65	95.94	98.88	98.52	98.89	99.44	97.2
F1 Score	0.96	0.97	0.95	0.93	0.91	0.99	0.99	0.99	0.96	0.93
90%-10%	**32**	**64**	**128**	**50**	**16**	**32**	**64**	**128**	**50**	**16**
Accuracy (%)	98.5	98.41	96.9	96.09	92.61	100	99.63	99.63	93.38	93.84
Loss (%)	7.27	8.75	13.84	14.8	22.28	3.66	2.57	2.5	24.7	16.41
AUC (%)	99.4	99.6	99	96.1	97.8	100	99.9	100	93.4	93.8
Precision	0.98	0.98	0.96	0.95	0.9	1	1	1	0.89	0.98
Sensitivity (%)	98.69	98.37	97.5	96.88	96.23	100	99.63	99.44	99.44	89.92
Specificity (%)	98.29	98.44	96.3	95.3	88.99	100	99.63	99.81	87.31	97.76
F1 Score	0.98	0.97	0.97	0.96	0.93	1	1	1	0.94	0.94



VGG16 architecture (https://neurohive.io/en/popular-networks/vgg16/)

However, due to limited resources, especially the availability of GPUs, training the model on a full VGG16 architecture is too computationally intensive and time-consuming. Therefore, we continue our research further and found out that some papers used simpler models in their designs to achieve high accuracy of prediction as well.

One of the paper we found (https://www.preprints.org/manuscript/202007.0591/v1/download) used a simple model on Kaggle's Covid19 Chest X-ray CT images dataset and achieved promising results as follows:

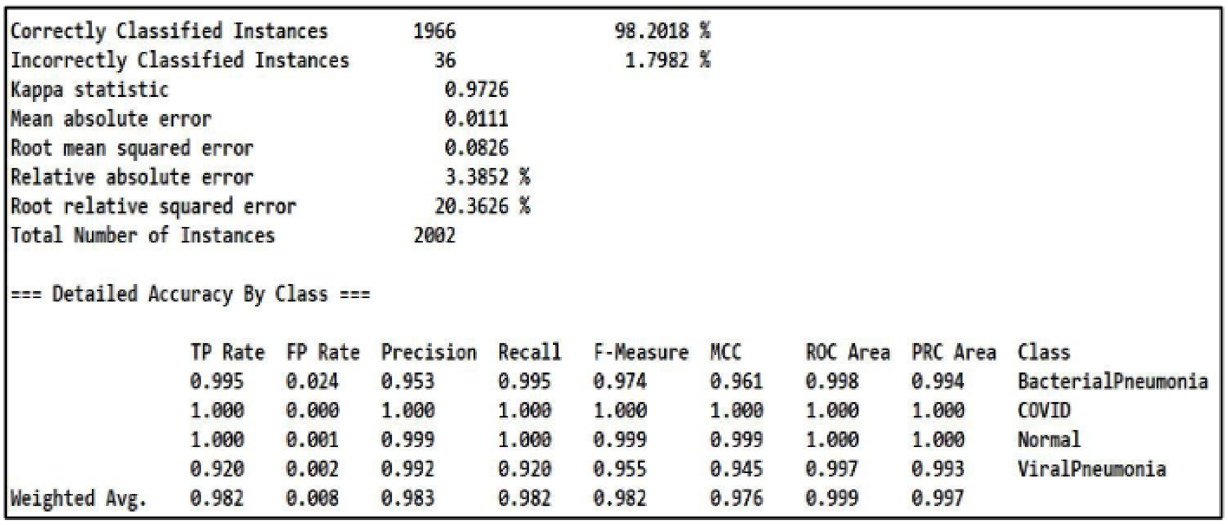
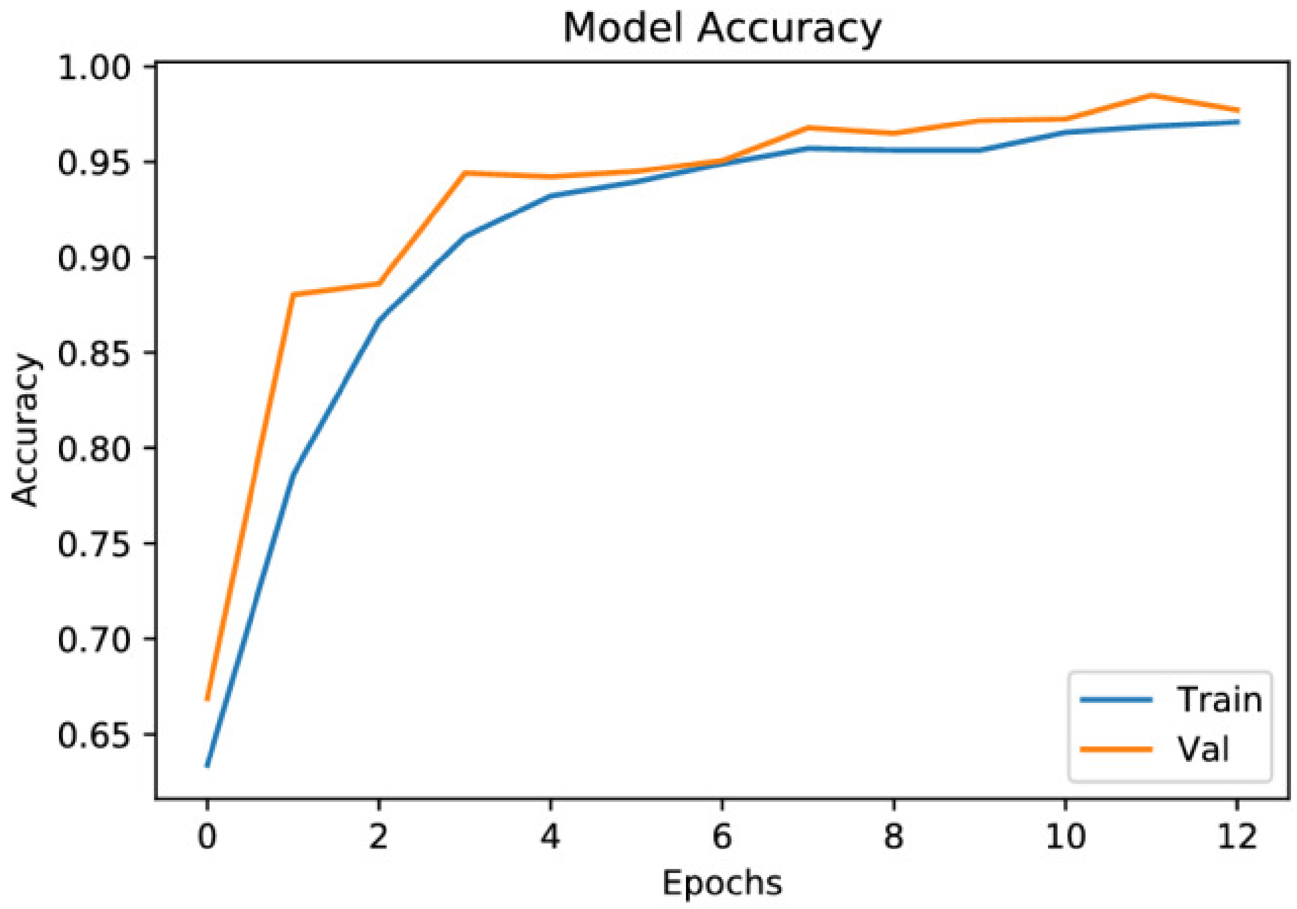
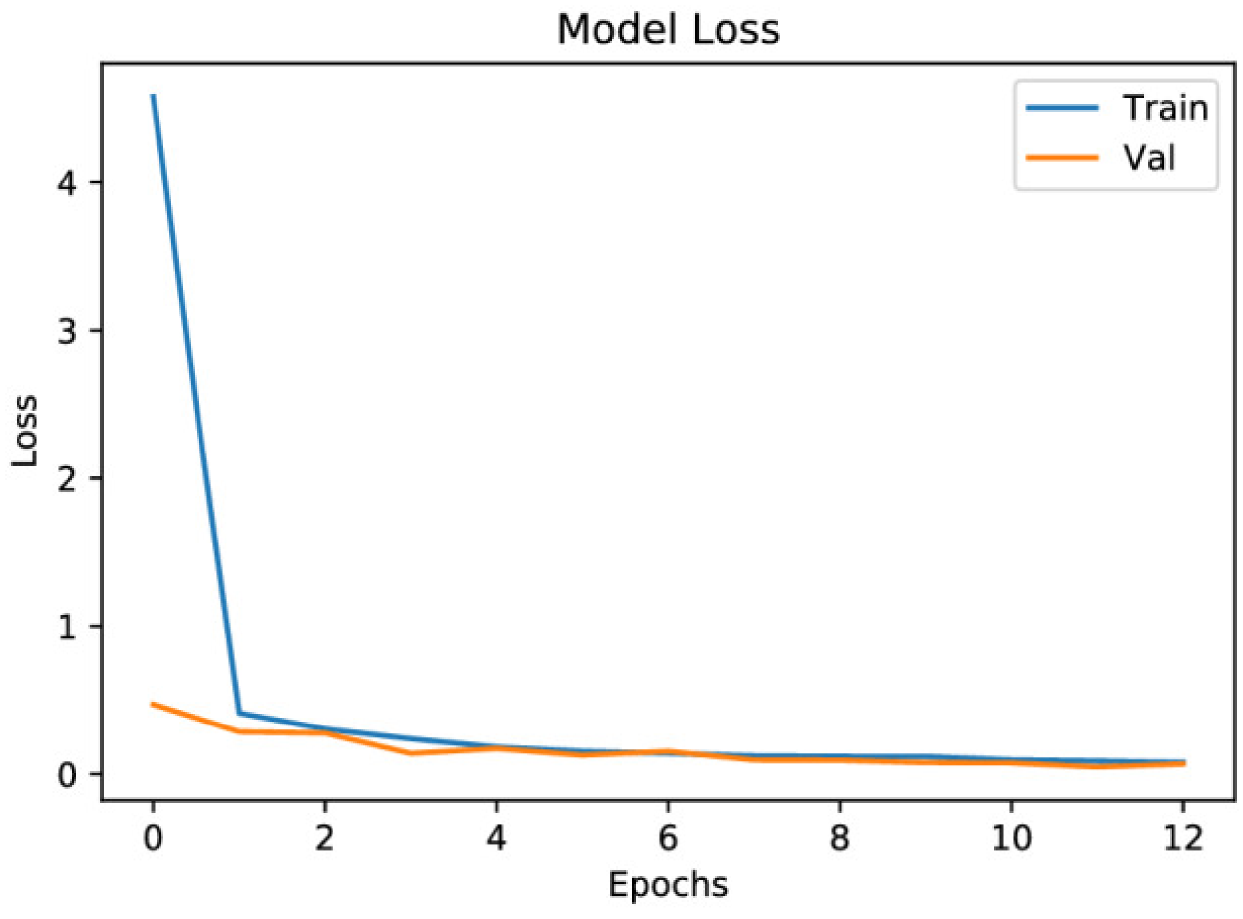

```
Correctly Classified Instances        1966               98.2018 %
Incorrectly Classified Instances        36                1.7982 %
Kappa statistic                          0.9726
Mean absolute error                      0.0111
Root mean squared error                  0.0826
Relative absolute error                  3.3852 %
Root relative squared error             20.3626 %
Total Number of Instances             2002

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.995    0.024    0.953      0.995    0.974      0.961    0.998     0.994     BacterialPneumonia
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     COVID
                 1.000    0.001    0.999      1.000    0.999      0.999    1.000     1.000     Normal
                 0.920    0.002    0.992      0.920    0.955      0.945    0.997     0.993     ViralPneumonia
Weighted Avg.    0.982    0.008    0.983      0.982    0.982      0.976    0.999     0.997
```

Fig12. The result of four class of Kaggle dataset after applying proposed CNN model

Another paper (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7841043/) suggested a different model than the aforementioned ones and the proposed model is trained and tested with 10,000 X-ray images of COVID-19 and normal people combined for two classes. The accuracy evolved as follows:


Model Accuracy
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
Epochs
0
2
4
6
8
10
12
Train
Val

(a)


Model Loss
Loss
4
3
2
1
0
Epochs
0
2
4
6
8
10
12
Train
Val

(b)

In the end, we decided to follow the first paper's model and designed our model as follows:

NAME	DESCRIPTION
Input	Size: 1 x 150 x 150
Convolution	in_channel=1, out_channel=20, kernel=5, stride=1
MaxPooling	kernel=3, stride=2, dilation=1
ReLU	
Convolution	in_channel=20, out_channel=50, kernel=5, stride=1
MaxPooling	kernel=3, stride=2, dilation=1
ReLU	
Dropout	rate=0.2
Fully Connected	500 neurons
Output Layer	2 classes

Implementation

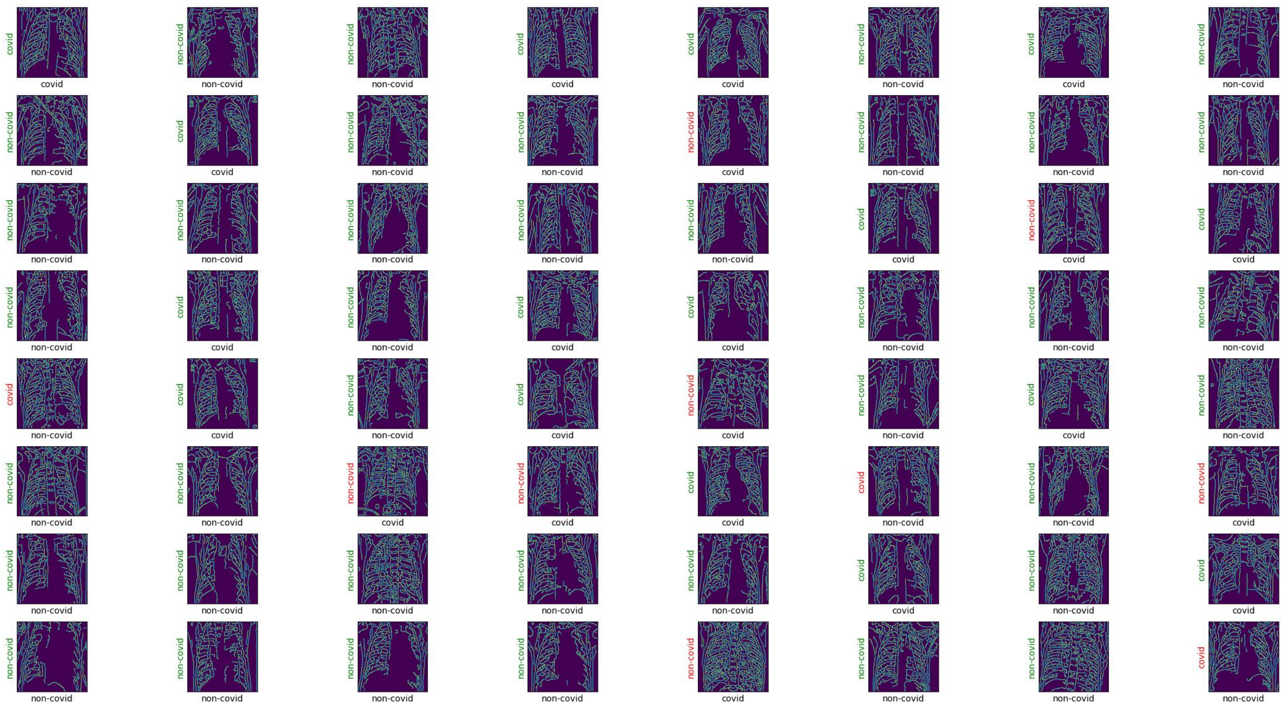
We did most of our initial model testings on Google CoLab, before the access to the school GPU was given to us.

Custom Dataset

We customized the Dataset object to perform differently for the 2 binary classifiers, one of which is **Normal**, **Infected**, and another one is **Covid**, and **Non-Covid**. The Dataset object inherits from the torch dataset class and `__getitem__` method is overwritten to output the augmented image as well as its label. For the first classifier, we use **0** to represent **Normal**, **1** to represent **Infected**. For the second classifier, we use **0** to represents **Covid** and **1** to represents **Non-Covid**.

Training

During training, we use a batch size of 32 images from the dataset for one training cycle. Each image is pre-processed and transformed randomly into 10 different images. After running 200 batches, we do one validation of the model. We wrote a custom function that will print out a graphical representation of the current performance on the validation of the model, shown as follows:



The column label represents the model prediction whereas the row label is the ground true label. If the prediction is correct, meaning that predicted label is the same as ground truth, it will be colored green. If not, it will be colored red. Additionally, we can see that the data augmentation has taken effect on some of the images, which are rotated, flipped and zoomed in.

With the 2 binary classifiers, we train each classifier independently. Each takes in a different DatasetLoader object. For **Normal-Infected** classifier as the first layer, the dataset consists of all the training dataset, but Covid and Non-Covid datasets are combined to form 1. For **Covid-NonCovid** classifier as the second layer, the dataset only consists of Covid and Non-Covid data without the Normal datasets. We customized our code such that user can supply different `mode` to activate different classifier training.

```
# IMPORTANT! Input mode -- 0: Normal-Infected 1: Covid-NonCovid
mode = 0

if mode == 0:
    train_loader = train_loader_NI
    test_loader = test_loader_NI
    val_loader = val_loader_NI
elif mode == 1:
    train_loader = train_loader_CN
    test_loader = test_loader_CN
    val_loader = val_loader_CN
else:
    print("Mode can only be 0 or 1!")
    sys.exit(1)
```

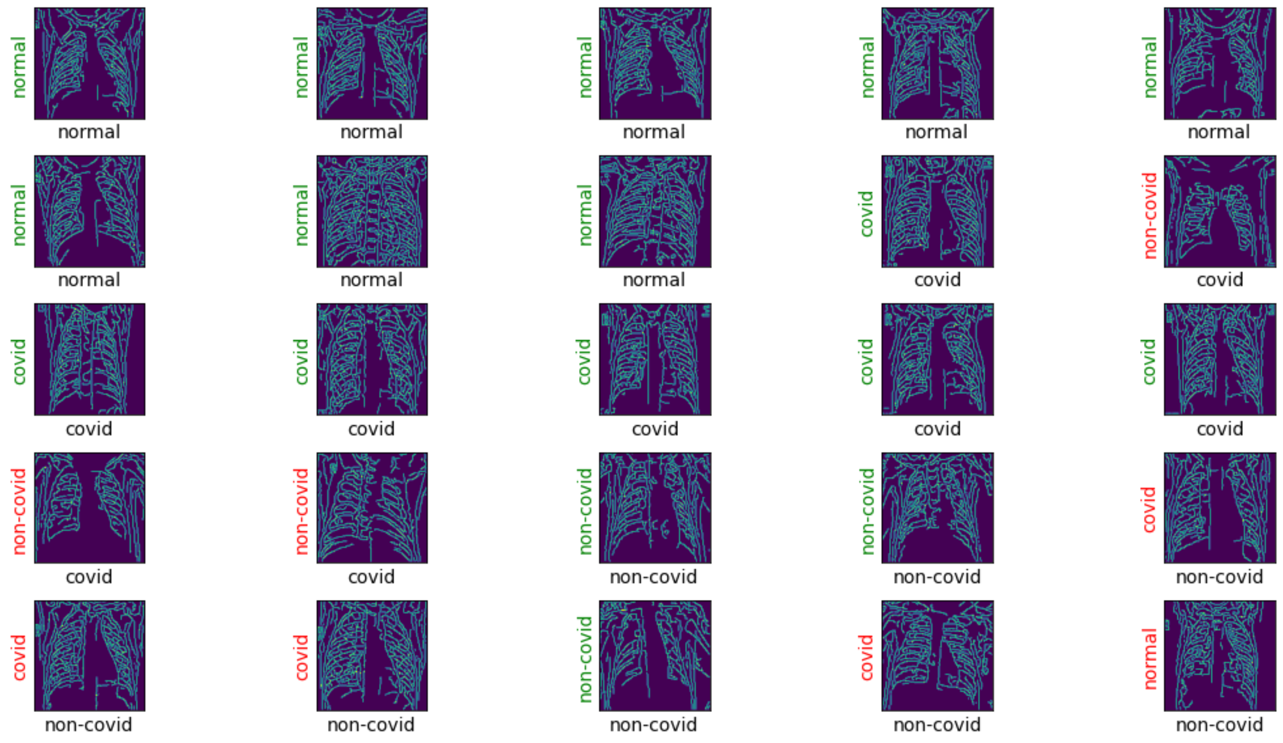
Saving and Re-training

When we migrated our training onto the school GPU, it allowed us to train with less time and larger batch size of 32 (since originally we trained on Colab with batch size of 8 only). However, due to the complicated layers and large number of parameters, our training will crash when going into the 3rd epochs as CUDA GPU ran out of memory. Hence, we implemented a saving mechanism that during each training (process until it crashed), the model is constantly saved whenever the accuracy reached a higher value. This ensures that we always have the best model to train with at the start each time we re-run the training. Moreover, for visual representation, the training loss and validation loss, and accuracy are saved for each batch and saved in separate CSV to plot graphs. In the event of crash, the re-run will append to the previous accuracy, train loss and validation loss CSV so in the end we can plot an overall graph about how the three metrics evolve over time (number of batches).

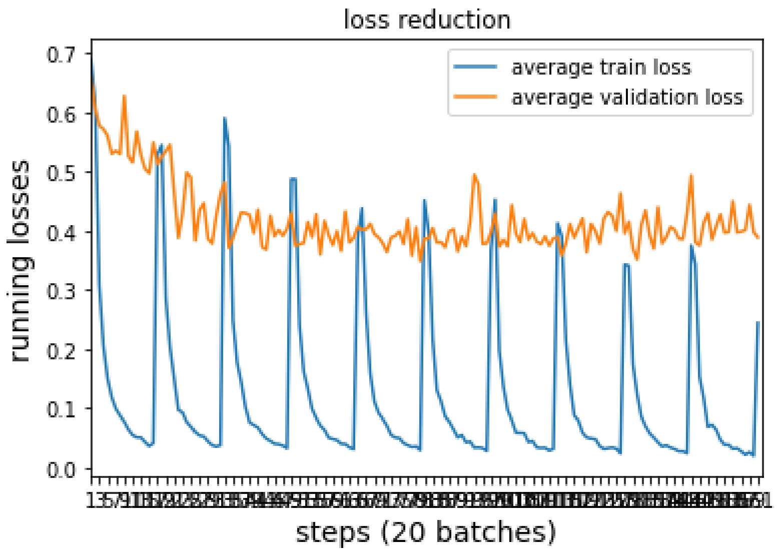
Results

In our final experiment, we ran 2 binary classifiers separately, each with batch size of **64**, with **data preprocessing** and **data augmentation** as discussed earlier, with validation after every **40 batches** of training, and running over 10 epochs. And then we tested our models with our test script, which chains the two models together. First model will predict the image is **Normal** or **Infected**, the **Infected** ones will then be passed on to the second model to be further classified into **Covid** or **Non-Covid**. Here are the results we obtained:

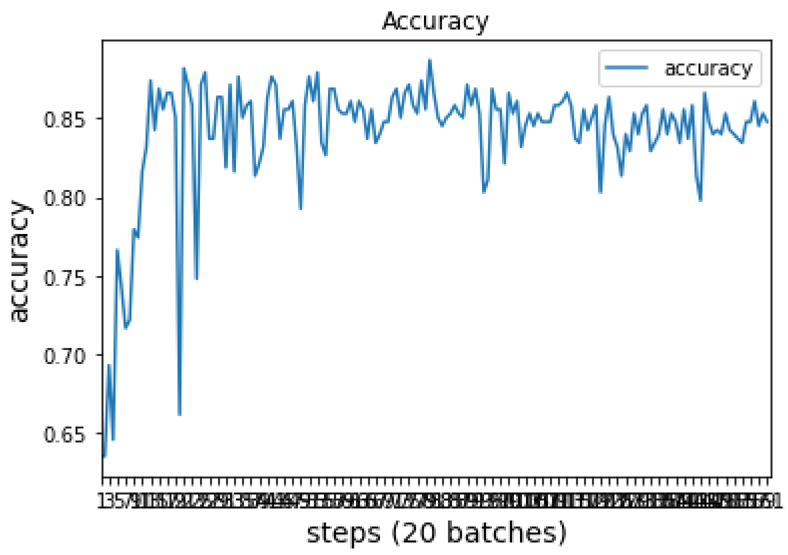
Predictions Grid



Loss Profile



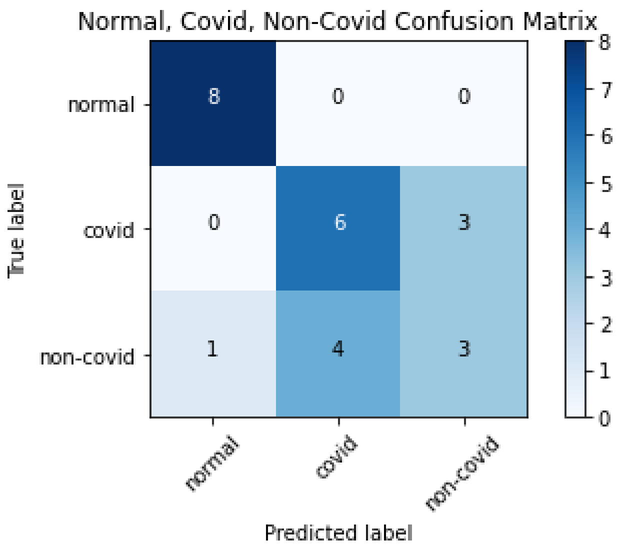
Accuracy Profile



Metrics

NAME	VALUE
F1 Score	0.665688
Recall	0.680000
Accuracy	0.680

Confusion Matrix



Results Analysis

From the confusion matrix above, we can easily visualize that our model is able to differentiate between **Normal** and **Infected** clearly except for **1** case which is non-covid but our model thinks it is normal. However, most of the inaccuracy comes at the second part of the classification, where our second model fails to differentiate between covid and non-covid cases. This phenomenon is commonly faced as discussed later in the discussion section.

Discussion

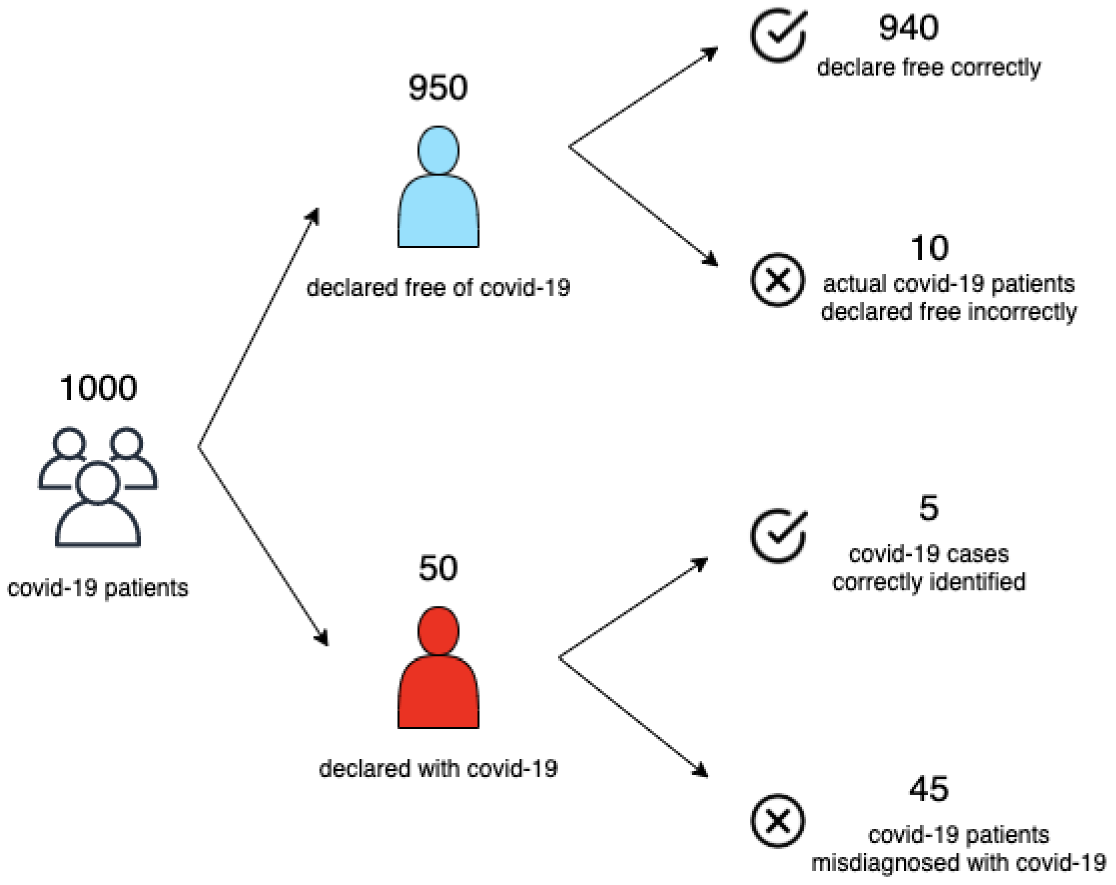
Q: You might find it more difficult to differentiate between non-covid and covid x-rays, rather than between normal x-rays and infected (both covid and non-covid) people x-rays. Was that something to be expected? Discuss.

A: This is expected as for covid vs non-covid, it is not recommended to be tested with x-rays. This is because a imaging findings are not specific enough to confirm Covid-19. They can only point to signs of an infection, that could be due to other causes such as seasonal flu. [https://blog.radiology.virginia.edu/covid-19-and-imaging/]

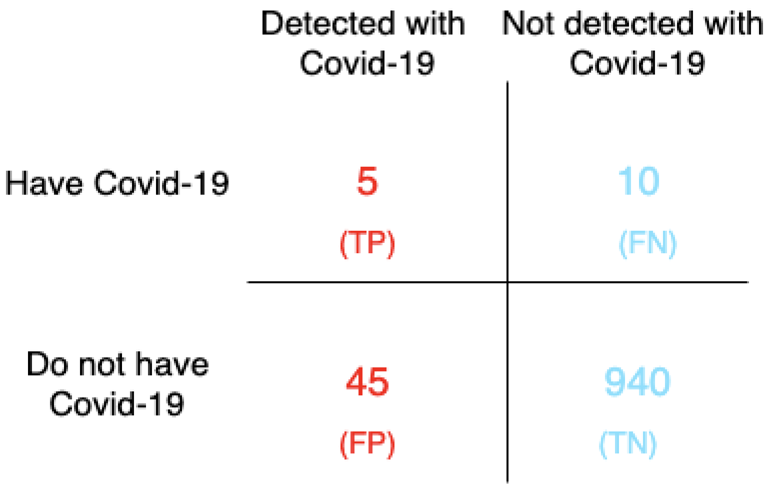
On the other hand, however, x-rays is an important test for making diagnosis of pneumonia. It can reveal areas of opacity (seen as white) which represent inflammation.

Q: Would it be better to have a model with high overall accuracy or low true negatives/false positives rates on certain classes? Discuss.

A: It will be good for prediction like our small project to have a low true negatives/false positive rates. For example:


1000
covid-19 patients
950
declared free of covid-19
50
declared with covid-19
940
declare free correctly
10
actual covid-19 patients declared free incorrectly
5
covid-19 cases correctly identified
45
covid-19 patients misdiagnosed with covid-19

Will give a confusion matrix as follows:



	Detected with Covid-19	Not detected with Covid-19
Have Covid-19	5 (TP)	10 (FN)
Do not have Covid-19	45 (FP)	940 (TN)

As for the accuracy, it will give:

$$correct = TP + TN = 5 + 940 = 945$$
$$incorrect = TP + FN = 45 + 10 = 55$$
$$accuracy = \frac{correct}{correct + incorrect} = \frac{945}{945+55} = 0.945$$

As such, the accuracy seems like a high number, however, it will leave out a lot of patients susceptible to extreme losses. Take for example the FN cases where 10 people would be relieved that the doctor has told them that they did not get covid-19 but they actually did, and will do around spreading. Therefore, in this case, a low FP/FN will be more useful than a high accuracy score.

Conclusion

We have preprocessed our image with **edge detection** to reduce noise and produce a new set of features. On top of that, we have done **data augmentation** to increase the size of our train dataset. We have picked a binary class classifier, where there are two separate models. For the first one, it is trained with **normal** vs **infected**, while for the second one, the model is trained with **covid** vs **non-covid**. After-which, we have obtained our results in terms of F1 score, recall, accuracy as well as confusion matrix. Due to the nature of our project (to predict infected vs normal and covid vs non-covid), we have concluded that it is better to present our data in a confusion matrix, shown in the results above.