# PRELIMINARY REPORT

**Group 8**

**Bodhisatta Maiti, Hualiang Qin, Preetham Pareddy, Wentao Yao**

## TASK

Our work focused on Covid-19 detection in patients. It is primarily a classification project. Our classification target includes audio data and image data:

1. Audio data are the cough sound and the respiratory sound of a patient.
2. Image data are the chest and lung X-Ray/CT scans of a patient.

## DATA

The details of the data will be discussed in the subsections.

Audio Classification

1. Dataset URL:
   https://www.kaggle.com/andrewmvd/covid19-cough-audio-classification
   research paper of the dataset:
   https://www.nature.com/articles/s41597-021-00937-4
2. Dataset URL:
   https://github.com/iiscleap/Coswara-Data
   Research paper of the dataset:
   https://arxiv.org/abs/2005.10548

Image Classification

1. Dataset URL:
   https://www.kaggle.com/andyczhao/covidx-cxr2
   research paper of the dataset:
   https://www.nature.com/articles/s41598-020-76550-z
2. Dataset URL:
   https://www.kaggle.com/mehradaria/covid19-lung-ct-scans
   research paper of the dataset:
   https://www.jmir.org/2021/4/e27468/

## Image Classification

For the image classification, we will be following a dual pathway:

1. We will extract features from the images and then use these as inputs to ML techniques such as KNN and Logistic Regression.
2. For image classification tasks, our primary focus will be on using convolutional neural networks (CNN). Our plan is to initially construct a basic CNN model using our own architecture and analyze the performance. Then we would like to utilize the concepts of transfer learning and fine-tuning for classification. We would like to use pre-trained models/networks such as VGG16 and Resnet18 to understand whether we can improve upon our base CNN model

## DATASET 1:

### Data Format:

Dataset URL:

https://www.kaggle.com/andyczhao/covidx-cxr2

research paper of the dataset:

https://www.nature.com/articles/s41598-020-76550-z

For this dataset, we have the data as following:

Negative and positive values of train:

positive 16490
negative 13992
Name: class, dtype: int64

Negative and positive values of test:

positive 200
negative 200 Name: class, dtype: int64

For each of the observations, it is a jpg/png file with 200X200 pixels and 3 color channels.

### Preprocessing

For the dataset, we applied downsampling due to the unbalanced data. However, we made a mistake that caused the data to still be unbalanced. We will fix it in our final report.

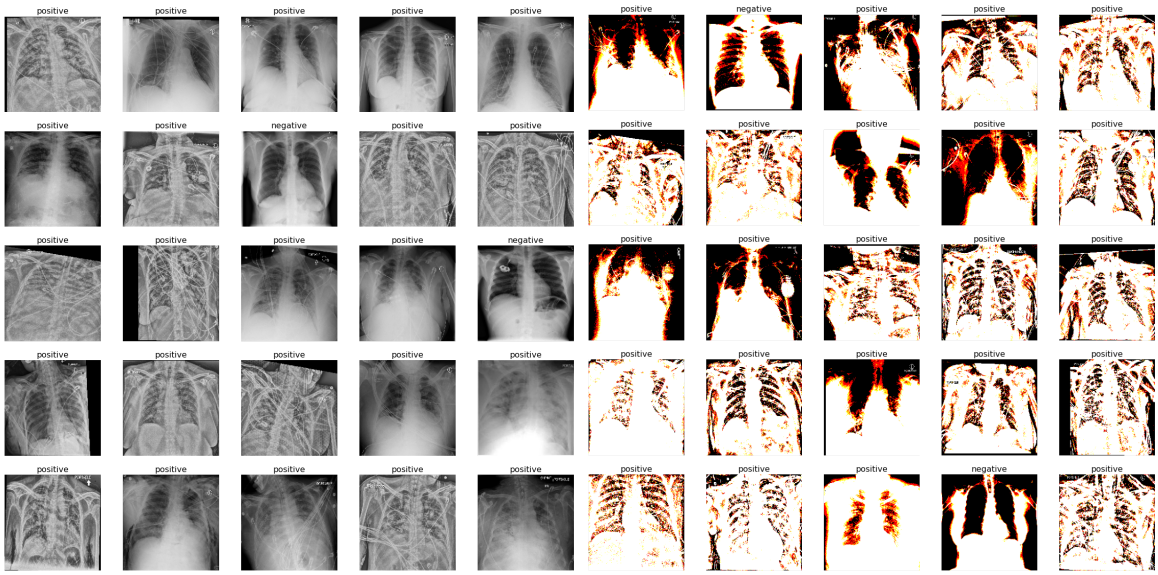
For the KNN, we only applied scale on the dataset. We did not apply any augmentation on the dataset since we want to find the shortest distance, augmentation may cause the error after shifting the image.

For the Convolution Neural Networks, we applied scale and augmentation on our own

baseline CNN. For the transfer learning built by VGG16, we applied the VGG16 preprocessing.

The dataset looks like the following:

Scale(Left) and VGG16 Preprocessing(Right)



## Model

**KNN(baseline):**

We apply a KNN model with K = 2

**CNN**:

We applied two convolutional layers with kernel size = 5, two max-pooling layers with size = 4, and two dense layers. We applied relu as the activation function in the hidden layer and sigmoid in the output layer. We also used Dropout and Batch Normalization for avoiding overfitting. The detail of the model is shown below:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 196, 196, 32)      2432
_________________________________________________________________
batch_normalization (BatchNo (None, 196, 196, 32)      128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 49, 49, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 45, 45, 64)        51264
_________________________________________________________________
batch_normalization_1 (Batch (None, 45, 45, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 11, 11, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 11, 11, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 7744)              0
_________________________________________________________________
dense (Dense)                (None, 128)               991360
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
batch_normalization_3 (Batch (None, 64)                256
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 1,054,529
Trainable params: 1,051,521
Non-trainable params: 3,008
_________________________________________________________________
```

### VGG16

**We applied a VGG16 layer and two dense layers. We applied relu as the activation function in the hidden layer and sigmoid in the output layer. We applied sigmoid in the output layer. The detail of the model is shown below:**

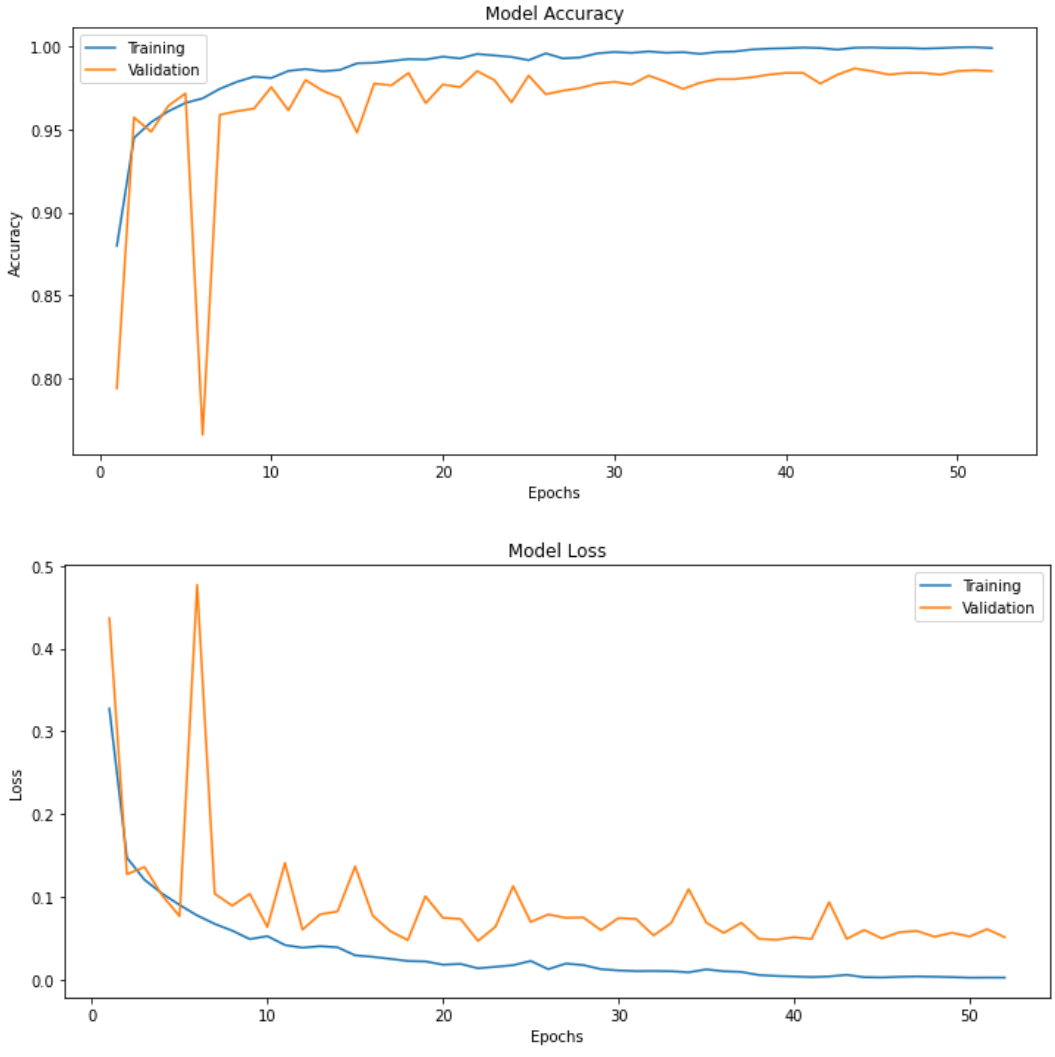
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 512)               14714688
_________________________________________________________________
dense (Dense)                (None, 128)               65664
_________________________________________________________________
batch_normalization (BatchNo (None, 128)               512
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
batch_normalization_1 (Batch (None, 64)                256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 14,789,441
Trainable params: 74,369
Non-trainable params: 14,715,072
_________________________________________________________________
```

## Result

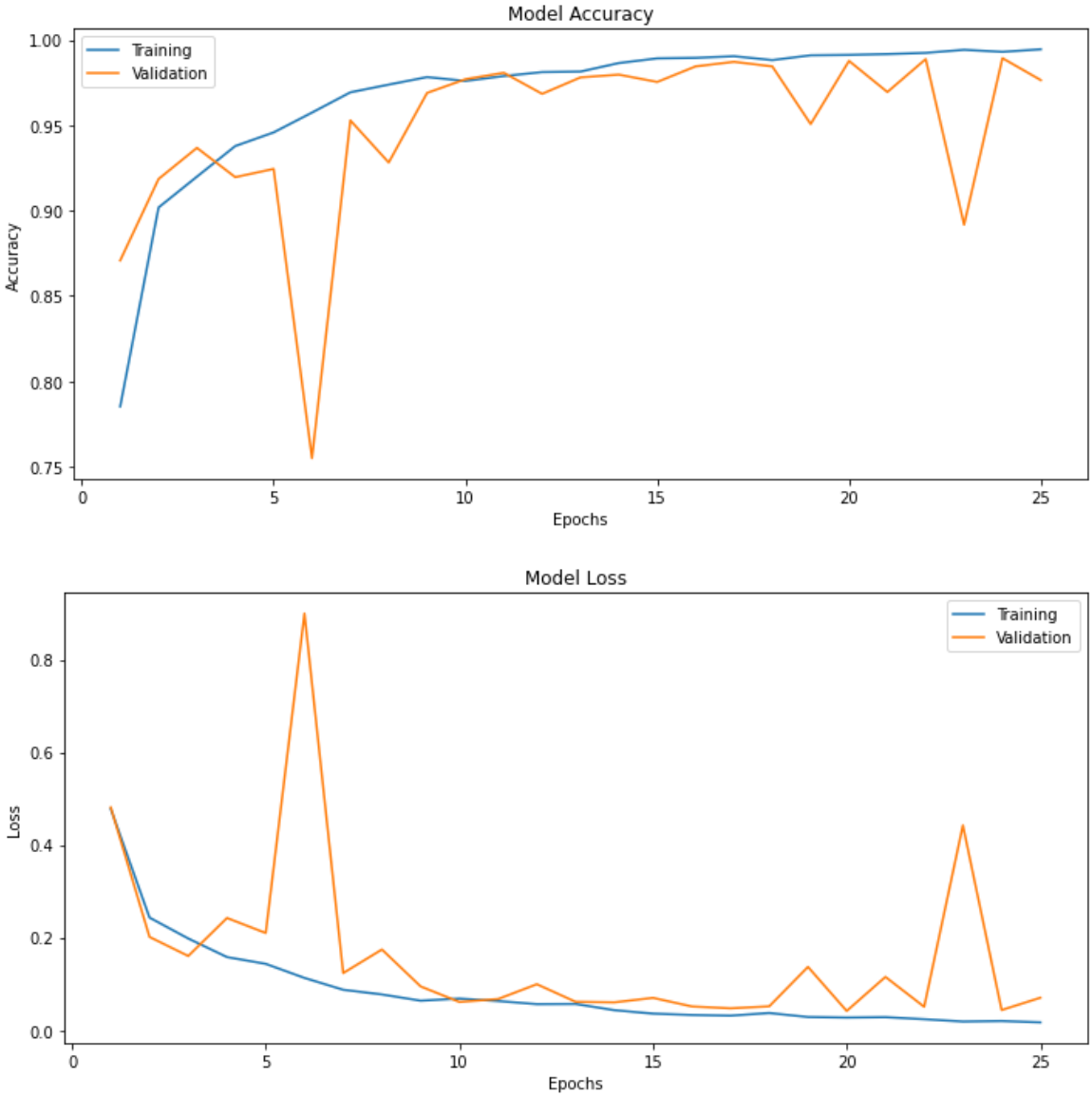
We used Accuracy as our evaluation metric.

For KNN(baseline), we have an accuracy of 0.69 on the testing set( could be improved significantly after we balanced the data). Also, due to the limitation of the memory, we cannot use every data in the data set to calculate the distance.

For CNN, we have an accuracy of 0.9993 on the training set, 0.9850 on the validation set, and 0.9525 on the testing set( could be improved after we balanced the data) after finishing the train in 12843.5s (3.6 hours). The learning curve is the following:



For VGG16 we have an accuracy of 0.9945 on the training set, 0.9764 on the validation set, and 0.9725 on the testing set( could be improved after we balanced the data)after finishing the train in 6705.2s (1.8 hours). The learning curve is the following:




Comparing the baseline KNN with our own CNN and VGG16, we can easily see a significant improvement in the performance. Comparing the CNN with VGG16, not only our test set accuracy improved, but we also saved time (1.8 hours!). This shows how powerful transfer learning is. It saves time and improves accuracy.

## Analysis

Even before the training process, I can classify by visual inspection (with my eyes), since there is a high chance that a positive image had some "shadow" in their lungs. Hence, letting the computer learn the image seems a possible approach. With the 0.9725 accuracies in the test set by VGG16, we can assume our approach to detect Covid-19 by analyzing the image of the lung X-ray scan of a patient is feasible. There is a thought that we can apply YOLO v5 to this dataset. If we are detecting Covid-19 by recognizing some "shadow" in the lungs, we may use YOLO to do it.

## DATASET 2:

### DATA

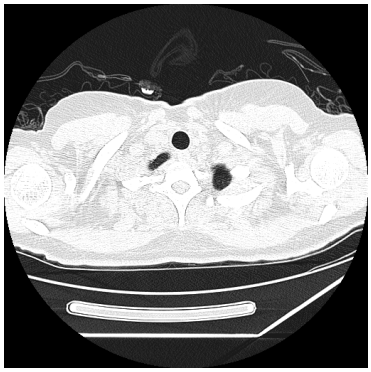
Source:

Dataset URL: https://www.kaggle.com/mehradaria/covid19-lung-ct-scans
Research paper of the dataset: https://www.jmir.org/2021/4/e27468/

Number of Observations: 8439
- Covid Data: 7495
- Non Covid Data: 944

- Training Set: 6751
- Test Set: 1688

Covid



Non Covid

For each of the observations, it is a png file with 512X512 pixels and 3 color channels (3 channels have the same value, since these are actually black/white pictures).

Preprocessing:

1. Convert the image to Grayscale (the images are black and white anyway, convert it to grey-scale simply the calculation)
2. Extract the pixels as array, each pixel is a feature for an image
3. Scaled down the array of pixels by 255 (the total number of color units)

## MODEL

**Resnet18 (baseline)**

We used the pretrained Resnet18 and froze their weight and parameters. After that, we add two layers at the end with relu as activation function, 0.2 dropout and applied softmax before the output. The Loss function chosen is NLLLoss().The learning rate was set to 0.003 and the optimizer was Adam.

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
├─Conv2d: 1-1                            [-1, 64, 112, 112]        (9,408)
├─BatchNorm2d: 1-2                       [-1, 64, 112, 112]        (128)
├─ReLU: 1-3                              [-1, 64, 112, 112]        --
├─MaxPool2d: 1-4                         [-1, 64, 56, 56]          --
├─Sequential: 1-5                        [-1, 64, 56, 56]          --
|    └─BasicBlock: 2-1                   [-1, 64, 56, 56]          --
|    |    └─Conv2d: 3-1                  [-1, 64, 56, 56]          (36,864)
|    |    └─BatchNorm2d: 3-2             [-1, 64, 56, 56]          (128)
|    |    └─ReLU: 3-3                    [-1, 64, 56, 56]          --
|    |    └─Conv2d: 3-4                  [-1, 64, 56, 56]          (36,864)
|    |    └─BatchNorm2d: 3-5             [-1, 64, 56, 56]          (128)
|    |    └─ReLU: 3-6                    [-1, 64, 56, 56]          --
|    └─BasicBlock: 2-2                   [-1, 64, 56, 56]          --
|    |    └─Conv2d: 3-7                  [-1, 64, 56, 56]          (36,864)
|    |    └─BatchNorm2d: 3-8             [-1, 64, 56, 56]          (128)
|    |    └─ReLU: 3-9                    [-1, 64, 56, 56]          --
|    |    └─Conv2d: 3-10                 [-1, 64, 56, 56]          (36,864)
|    |    └─BatchNorm2d: 3-11            [-1, 64, 56, 56]          (128)
|    |    └─ReLU: 3-12                   [-1, 64, 56, 56]          --
├─Sequential: 1-6                        [-1, 128, 28, 28]         --
|    └─BasicBlock: 2-3                   [-1, 128, 28, 28]         --
|    |    └─Conv2d: 3-13                 [-1, 128, 28, 28]         (73,728)
|    |    └─BatchNorm2d: 3-14            [-1, 128, 28, 28]         (256)
|    |    └─ReLU: 3-15                   [-1, 128, 28, 28]         --
|    |    └─Conv2d: 3-16                 [-1, 128, 28, 28]         (147,456)
|    |    └─BatchNorm2d: 3-17            [-1, 128, 28, 28]         (256)
```

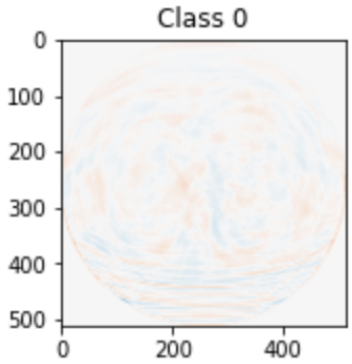
```
|    |    └─Sequential: 3-18                  [-1, 128, 28, 28]         (8,448)
|    |    └─ReLU: 3-19                        [-1, 128, 28, 28]         --
|    └─BasicBlock: 2-4                        [-1, 128, 28, 28]         --
|    |    └─Conv2d: 3-20                      [-1, 128, 28, 28]         (147,456)
|    |    └─BatchNorm2d: 3-21                 [-1, 128, 28, 28]         (256)
|    |    └─ReLU: 3-22                        [-1, 128, 28, 28]         --
|    |    └─Conv2d: 3-23                      [-1, 128, 28, 28]         (147,456)
|    |    └─BatchNorm2d: 3-24                 [-1, 128, 28, 28]         (256)
|    |    └─ReLU: 3-25                        [-1, 128, 28, 28]         --
├─Sequential: 1-7                            [-1, 256, 14, 14]         --
|    └─BasicBlock: 2-5                        [-1, 256, 14, 14]         --
|    |    └─Conv2d: 3-26                      [-1, 256, 14, 14]         (294,912)
|    |    └─BatchNorm2d: 3-27                 [-1, 256, 14, 14]         (512)
|    |    └─ReLU: 3-28                        [-1, 256, 14, 14]         --
|    |    └─Conv2d: 3-29                      [-1, 256, 14, 14]         (589,824)
|    |    └─BatchNorm2d: 3-30                 [-1, 256, 14, 14]         (512)
|    |    └─Sequential: 3-31                  [-1, 256, 14, 14]         (33,280)
|    |    └─ReLU: 3-32                        [-1, 256, 14, 14]         --
|    └─BasicBlock: 2-6                        [-1, 256, 14, 14]         --
|    |    └─Conv2d: 3-33                      [-1, 256, 14, 14]         (589,824)
|    |    └─BatchNorm2d: 3-34                 [-1, 256, 14, 14]         (512)
|    |    └─ReLU: 3-35                        [-1, 256, 14, 14]         --
|    |    └─Conv2d: 3-36                      [-1, 256, 14, 14]         (589,824)
|    |    └─BatchNorm2d: 3-37                 [-1, 256, 14, 14]         (512)
|    |    └─ReLU: 3-38                        [-1, 256, 14, 14]         --
├─Sequential: 1-8                            [-1, 512, 7, 7]           --
|    └─BasicBlock: 2-7                        [-1, 512, 7, 7]           --
|    |    └─Conv2d: 3-39                      [-1, 512, 7, 7]           (1,179,648)
|    |    └─BatchNorm2d: 3-40                 [-1, 512, 7, 7]           (1,024)
|    |    └─ReLU: 3-41                        [-1, 512, 7, 7]           --
|    |    └─Conv2d: 3-42                      [-1, 512, 7, 7]           (2,359,296)
|    |    └─BatchNorm2d: 3-43                 [-1, 512, 7, 7]           (1,024)
|    |    └─Sequential: 3-44                  [-1, 512, 7, 7]           (132,096)
|    |    └─ReLU: 3-45                        [-1, 512, 7, 7]           --
|    └─BasicBlock: 2-8                        [-1, 512, 7, 7]           --
|    |    └─Conv2d: 3-46                      [-1, 512, 7, 7]           (2,359,296)
|    |    └─BatchNorm2d: 3-47                 [-1, 512, 7, 7]           (1,024)
|    |    └─ReLU: 3-48                        [-1, 512, 7, 7]           --
|    |    └─Conv2d: 3-49                      [-1, 512, 7, 7]           (2,359,296)
```

```
|    |    └─BatchNorm2d: 3-50                 [-1, 512, 7, 7]           (1,024)
|    |    └─ReLU: 3-51                        [-1, 512, 7, 7]           --
├─AdaptiveAvgPool2d: 1-9                      [-1, 512, 1, 1]           --
├─Sequential: 1-10                            [-1, 2]                   --
|    └─Linear: 2-9                            [-1, 64]                  32,832
|    └─ReLU: 2-10                             [-1, 64]                  --
|    └─Dropout: 2-11                          [-1, 64]                  --
|    └─Linear: 2-12                           [-1, 2]                   130
|    └─LogSoftmax: 2-13                       [-1, 2]                   --
==========================================================================
Total params: 11,209,474
Trainable params: 32,962
Non-trainable params: 11,176,512
Total mult-adds (G): 1.84
==========================================================================
```

**Logistic Regression**

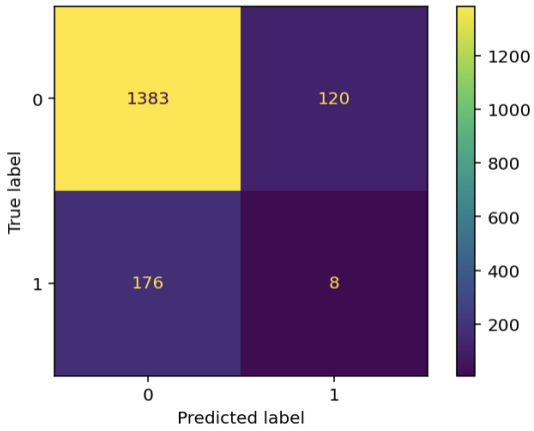
We applied logistic regression from Scikit Learn. Since this is simply a binary classification task. For the multi_class parameter, we chose ‘ovr’ (for binary classification), with tolerance = 0.1, no penalty.

We want to know what features (which pixels in this case) are important in determining the diagnosis of Covid. The following plot shows the coefficient vector for Covid, with positive coefficients in blue and negative coefficients in red.



# Result

### Resnet18 (baseline)




```
f1_score(labels_test, pred_test, average='macro')
```

```
0.4773066036945854
```

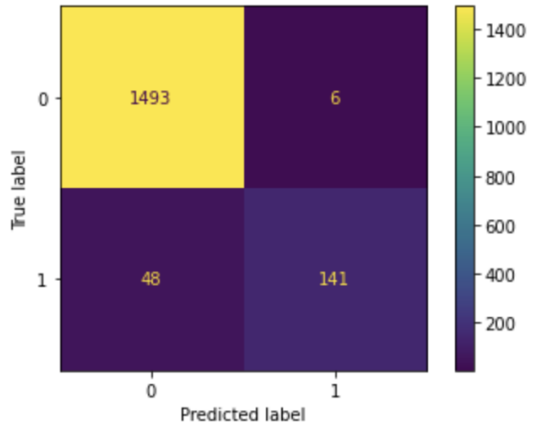
```
f1_score(labels_test, pred_test, average='micro')
```

```
0.8245406046235921
```

```
f1_score(labels_test, pred_test, average='weighted')
```

```
0.8103987107675744
```

### Logistic Regression




```
a = f1_score(test_label, test_pred, average='macro')
b = f1_score(test_label, test_pred, average='micro')
c = f1_score(test_label, test_pred, average='weighted')
print(a, b, c)
```

```
0.9107612781954888 0.9680094786729858 0.9662310582439512
```

For the evaluation matrix, we chose the confusion matrix here. Confusion matrix is a matrix of size 2×2 for binary classification with actual values on one axis and predicted on another. In the case of this classification task, Covid observations were labeled as 0 and Non Covid observations were labeled as 1. Compared with Resnet18, the logistic regression approach has a higher f1 score while calculating with macro, micro or weighted average. According to the results, we can see that Logistic Regression provides better performance in classifying CT scan from Covid to Non Covid and moreover, the training time was less.

## Analysis

As we have seen in the Result section, Logistic regression has better performance in identifying whether the CT scans came from a Covid patient or not. More importantly, logistic regression has its benefit of knowing what features are significant for diagnosis, which might provide researchers/physicians more insights on correlative study. While analyzing the results in the confusion matrix, we can observe that logistic regression did better in avoiding the False Positives case, which is important particularly in the case of pandemic since we don't want to miss any positive case, this will aid us in the objective of lowering the infectivity as much as possible.