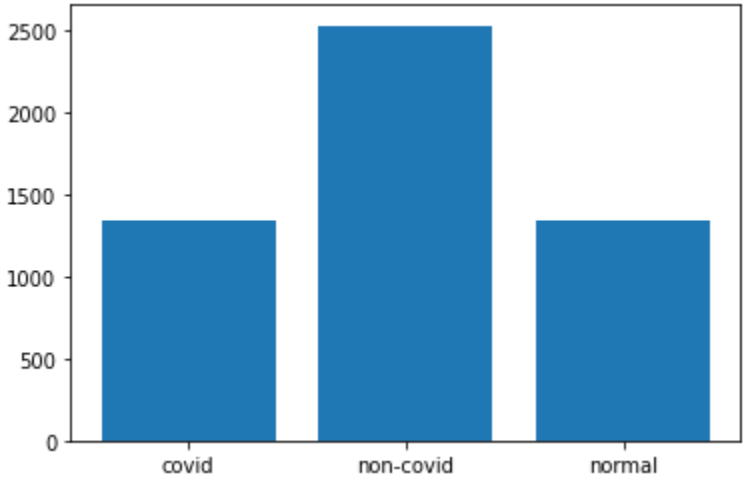
Deep Learning Small Project
Rahul Bhattacharjee, 1003719

Exploration of Data

The data provided are X-ray chest images, which will be used to predict whether the patient has COVID, a non-COVID infection, or is healthy.

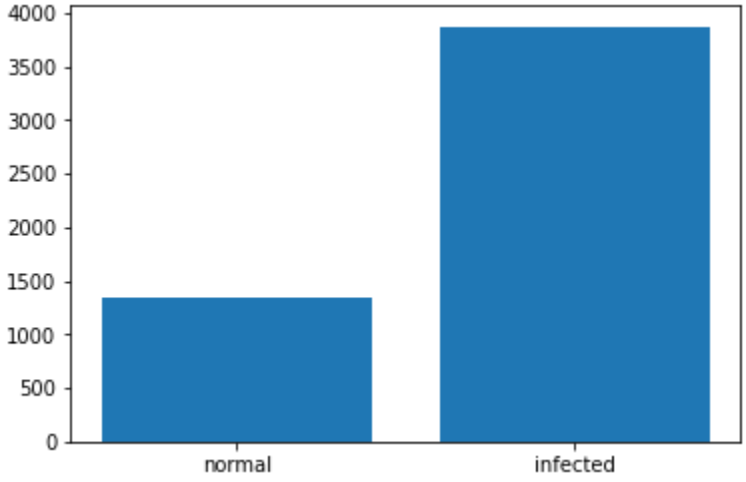
Lets look at a bar graph with the number of images provided for training:

1. COVID, non-COVID and Normal



a.

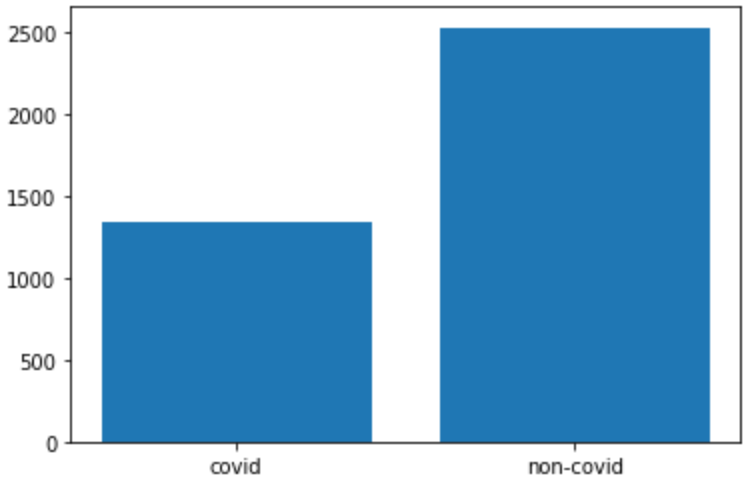
2. Normal and Infected



a.

3. COVID and non-COVID

a.



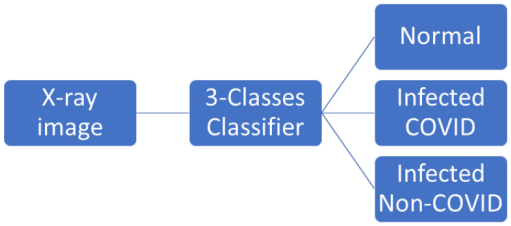
We notice the number of infected training images to be almost 3 times as large as the normal images, so we should try to avoid any training bias that would enable the model to blindly predict input images as infected rather than normal. However, we must note that errors in predicting infected patients are far more dangerous than predicting normal patients correctly. There is a similar consideration to be noted in covid vs non-covid.

Inspiration from literature

1. We started by taking inspiration from the paper “COVINet: a convolutional neural network approach for predicting COVID-19 from chest X-ray images”. Although traditional models such as ResNet and InceptionV3 seem to achieve extremely high accuracies upto 97%, we note that it takes a lot of time to train those models, due to their large and deep architectures.
2. The architecture proposed in their paper is used to classify COVID vs normal images upto 97% accuracy, so its reasonable to assume that this model could be a good starting point for the first task: to classify infected and non-infected images.

Model Architecture Considerations and Decisions

3. Two types of pipelines
    a. Train a single classifier to classify all three categories



i.

ii. Potential Advantages

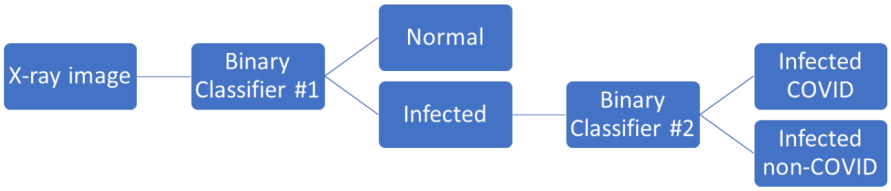
1. Since we are training the model on all the data, the model will be able to learn relative relationships, and might get a better idea of

the context of the data. This could allow the model to understand the specific features of images which differentiates the different categories, and thus could be more effective.

iii. Potential Disadvantages

1. Classifying these 3 different categories is a more difficult task than classifying 2 of these categories
2. The 3 categories are not at the same level of analysis. At a conceptual level, it seems that the distinction between COVID and non-COVID infection is more fine-grained than the distinction between infected and non-infected.

b. Train two separate classifiers for each separate task



i.

ii. Potential Advantages

1. This setup can take advantage of the different distinctions between the two sets of categories in the three.
2. It breaks up a complex task into two simpler tasks, which is considered a good general approach in deep learning. Usually this is done in terms of convolution layers rather than fully connected layers, but this task separation should also be an example of it.

iii. Potential Disadvantages

1. There is potential for error propagation, as there are two avenues for error here: once by the first model, and then by the second model. So accurate predictions are those which are necessarily classified correctly by both the first and the second models
2. The fact that both models are not connected together(for the training process) means that there is the potential for them to make different types of errors, and not learn from the errors each other is making. This could reduce the accuracy significantly, due to the reason in the previous point.

4. Architecture settled upon:
   a. Pipeline Type: Two Models, trained on 2 separate tasks. This turned out to be empirically better, after I trained models for both the 3-in-1 case and this case.

```
Net2(
  (conv1): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1))
  (dropout1): Dropout(p=0.2, inplace=False)
  (conv2): Conv2d(16, 128, kernel_size=(3, 3), stride=(1, 1))
  (pool1): MaxPool2d(kernel_size=3, stride=3, padding=0, dilation=1, ceil_mode=False)
  (dropout2): Dropout(p=0.2, inplace=False)
  (conv3): Conv2d(128, 256, kernel_size=(2, 2), stride=(1, 1))
  (pool2): AvgPool2d(kernel_size=3, stride=3, padding=0)
  (dropout3): Dropout(p=0.2, inplace=False)
  (fc1): Linear(in_features=57600, out_features=100, bias=True)
  (fc2): Linear(in_features=100, out_features=2, bias=True)
)
```

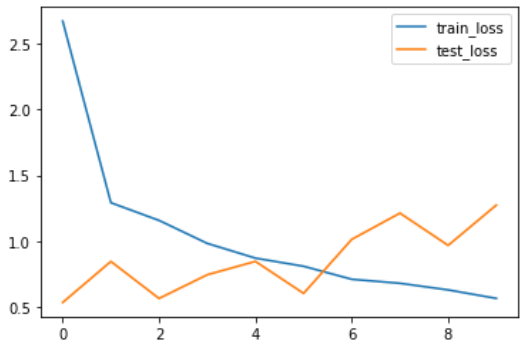
b.

Optimizer Choice and Parameters

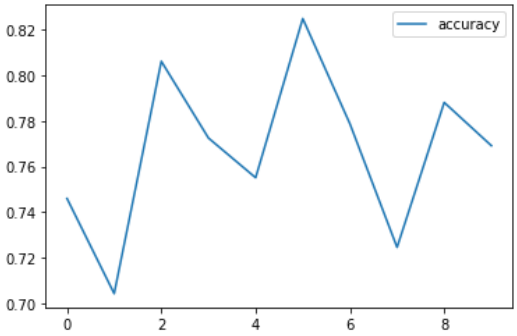
1. Optimizer Choice: Adam. I chose Adam (which we studied in class), because it consists of the best properties of both Adagrad and RMSProp gradient descent optimizers. It is also more theoretically rigorous, due to the real 'weighted to 1' properties of the functions used in optimizing weights using the gradient.
2. Learning Rate: 0.001. This is a standard baseline learning rate. I empirically observed that when the learning rate was higher than this, say 0.005, the loss did not decrease as well, and sometimes fluctuated
3. Batch size: 32. This choice was made keeping in mind the size of the dataset. Empirically, using a larger batch size (64) led to poor learning, both in terms of loss reduction, as well as test set performance.
4. Epochs: Depends on the model being trained, and the task. In order to optimize this, I kept saving a version of the model for each subsequent epoch - this way I get to keep and use the model which had the best performance out of all of these epochs.

Model(s) Performance

1. Model1 (Normal / Infected)
    a. Class Wise Testing Accuracy
        i. `tensor([0.5726, 0.9685])`
    b. Confusion Matrix
        i.
```
tensor([[134., 100.],
        [ 12., 369.]])
```
    c. Loss Curves



d. Accuracy Curve



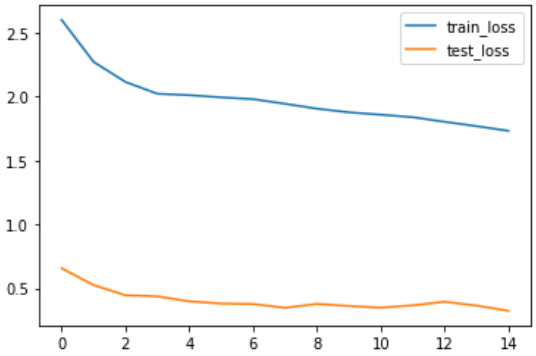
2. Model2 (COVID / Non-COVID Infection)
   a. Class Wise Testing Accuracy

```
tensor([0.7770, 0.9793])
```

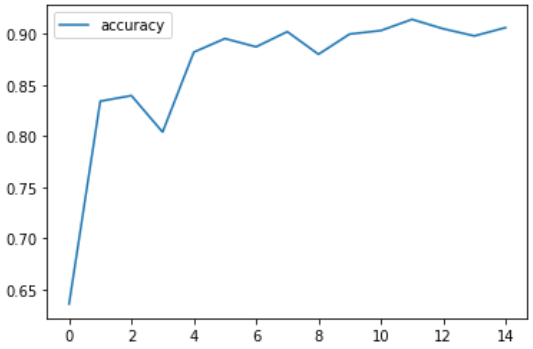
   b. Confusion Matrix

```
tensor([[108.,  31.],
        [  5., 237.]])
```

   c. Loss Curves



   d. Accuracy Curve

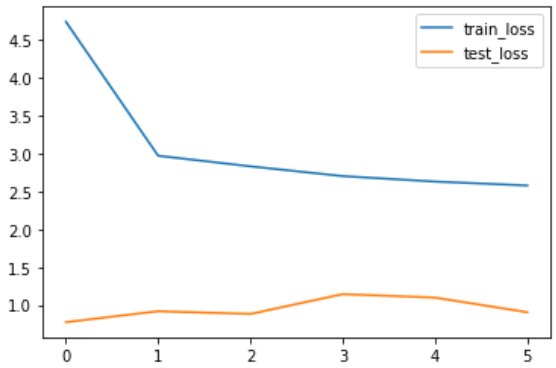


3. Model3 (COVID / Non-COVID Infection / Normal)
   a. Class Wise Testing Accuracy
      i. `tensor([0.7122, 0.9628, 0.4145])`

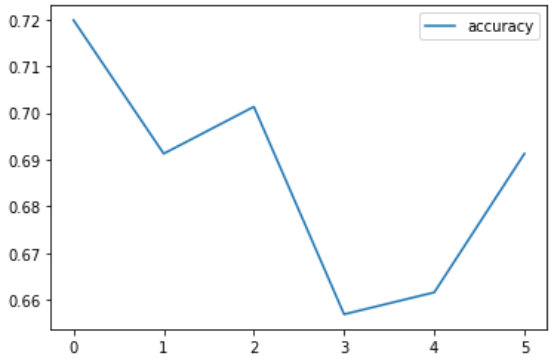
b. Confusion Matrix

```
tensor([[ 99.,  38.,   2.],
        [  8., 233.,   1.],
        [100.,  37.,  97.]])
```

c. Loss Curves



d. Accuracy Curve



Validation Prediction Visuals

1. Present in the notebook: model_evaluation.ipynb

Instructions on using code

1. Training model from scratch and saving it
    a. Set the hyperparameters you want
    b. Set the model name
    c. Run the notebook model_training.ipynb
2. Evaluating saved models
    a. Ensure that the model names in the model loading cell are correct
    b. Ensure training log file names in the dataframe loading cells are correct
    c. Run the notebook model_evaluation.ipynb