50.039-Deep-Learning

GitHub Link: https://github.com/Oxiang/50.039-Deep-Learning/tree/main

For Module 50.039

Students:

- Ong Xiang Qian - 1002646
- Glenn Chia Jin Wee - 1003118

1. Table of contents

2. Directory structure

```
- notebooks
  |_ colab
    |_ boilerplate
      |_ custom_dataset_dataloader_cascade.ipynb
      |_ custom_dataset_dataloader.ipynb
    |_ experiments
      |_ batch_size
        |_ COVINet_classifier.ipynb
      |_ binary_selection
        |_ COVINet_classifier.ipynb
        |_ naive_classifier.ipynb
        |_ resnet_classifier.ipynb
      |_ cross_entropy_weights
        |_ COVINet_classifier.ipynb
    |_ final
      |_ train_binary_classifier_model.ipynb
      |_ test_binary_classifier_model.ipynb
      |_ binary_classifier_log.txt
      |_ checkpoint_model_5_binary_L0_net_21_03_2021_20_42_56 # layer 0
model
      |_ checkpoint_model_10_binary_L1_net_21_03_2021_20_39_57 # layer 1
model
    |_ tuning
      |_ Hyperparameters Experiment.ipynb
  |_ local
```

```
    |_ final
      |_ test_binary_classifier_model.ipynb
      |_ train_binary_classifier_model.ipynb
  |_ references
    |_ custom_dataset_dataloader_demo.ipynb
- instructions # contains the small project instructions
  |_ Small_Project_instructions.pdf
README.md # contains the overview of the project and explanations for the
different requirements
```

3. Final files and instructions on running them

3.1 Training the final model

The file for this can be found in `./notebooks/local/final/train_binary_classifier_model.ipynb`

To train the final model from scratch, simply follow the steps provided below:

Step 1. Move to the folder

```
cd notebooks/local/final
```

Step 2. Open **train_binary_classifier_model.ipynb** in Jupyter Notebook

Step 3. Run every cells

3.2 Loading and testing the trained model

The file for this can be found in `./notebooks/local/final/test_binary_classifier_model.ipynb`

Running on Jupyter Notebook locally

The pre-trained weights are stored in the same folder as the final model. Follow the steps to recreate the testing model.

Step 1. Move to the folder

```
cd notebooks/local/final
```

Step 2. Open **test_binary_classifier_model.ipynb** in Jupyter Notebook

Step 3. Run every cells

The logs for the pre-trained model can be found in the same folder: **binary_classifier_log.txt**

4. Data analysis

4.1 Custom Datasets and Dataloader

The custom dataset requires two core methods to work: **len** and **getitem**. For **len** method, it simply returns the total size of the datasets. As for **getitem**, it will match the index provided to the size of each classes in the dataset and tweak the path and final index to retrieve the image from the targeted folder.

4.1.1 Binary cascade problem

The dataset class in the binary cascade problem is split into 2 separate classes, L0_Lung_Dataset and L1_Lung_Dataset. The L0_Lung_Dataset class is used to provide images and labels of normal vs infected datasets. L1_Lung_Dataset on the other hand is used to provide images and labels of infected COVID and infected non-COVID.

```
def __len__(self):
        """
        Length special method, returns the number of images in dataset.
        """

        # Length function
        return sum(self.dataset_numbers.values())


    def __getitem__(self, index):
        """
        Getitem special method.

        Expects an integer value index, between 0 and len(self) - 1.

        Returns the image and its label as a one hot vector, both
        in torch tensor format in dataset.
        """

        # Get item special method
        first_val = int(list(self.dataset_numbers.values())[0])
        if index < first_val:
            class_val = 'normal'
            label = torch.Tensor([1, 0])
            covid_status = ""
        else:
            class_val = 'infected'
            index = index - first_val
            label = torch.Tensor([0, 1])
            infected_covid_numbers =
int(list(self.infected_sub_class_numbers.values())[0]) # covid
            if index < infected_covid_numbers:
                class_val = 'infected'
                covid_status = 'covid'
            else:
                class_val = 'infected'
                index = index - infected_covid_numbers
                covid_status = 'non-covid'
        im = self.open_img(self.groups, class_val, covid_status, index)
        im = transforms.functional.to_tensor(np.array(im)).float()
        return im, label
```

4.1.2 Three-class problem

For the three-class dataset, it is much simpler. The Lung_Dataset class is used to provide images and labels for normal, infected COVID and infected non-COVID.

```
def __len__(self):
        """
        Length special method, returns the number of images in dataset.
        """

        # Length function
        return sum(self.dataset_numbers.values())


    def __getitem__(self, index):
        """
        Getitem special method.

        Expects an integer value index, between 0 and len(self) - 1.

        Returns the image and its label as a one hot vector, both
        in torch tensor format in dataset.
        """

        # Get item special method
        first_val = int(list(self.dataset_numbers.values())[0])
        second_val = int(list(self.dataset_numbers.values())[1])
        if index < first_val:
            class_val = 'normal'
            label = torch.Tensor([1, 0, 0])
        elif index < (first_val+second_val):
            class_val = 'infected_covid'
            index = index - first_val
            label = torch.Tensor([0, 1, 0])
        else:
            class_val = "infected_non_covid"
            index = index - (first_val+second_val)
            label = torch.Tensor([0, 0, 1])
        im = self.open_img(self.groups, class_val, index)
        im = transforms.functional.to_tensor(np.array(im)).float()
        return im, label
```

4.2 Distribution of data among classes and analysis

Discuss whether or not the dataset is balanced between classes, uniformly distributed, etc.

Training set

The train data for the different classes are imbalanced. From the graph plotted below, the `infected_non_covid` class has significantly more data points than the other classes. Overall, the ratio `normal:infected_covid:infected_non_covid` is approximately `1:1:2`.

This could present more complications if the model is trained in a stacking manner by first training normal vs infected. The ratio of `normal:infected` would be a ratio of `1:3` which is more imbalanced. However, if there are distinct differences in the data of the normal and infected then this large difference could be negated. Alternatively, data augmentation techniques could be used to increase the size of the training data for the `normal` class. Another possible workaround is to adjust the weights assigned to the different classes in the loss function.

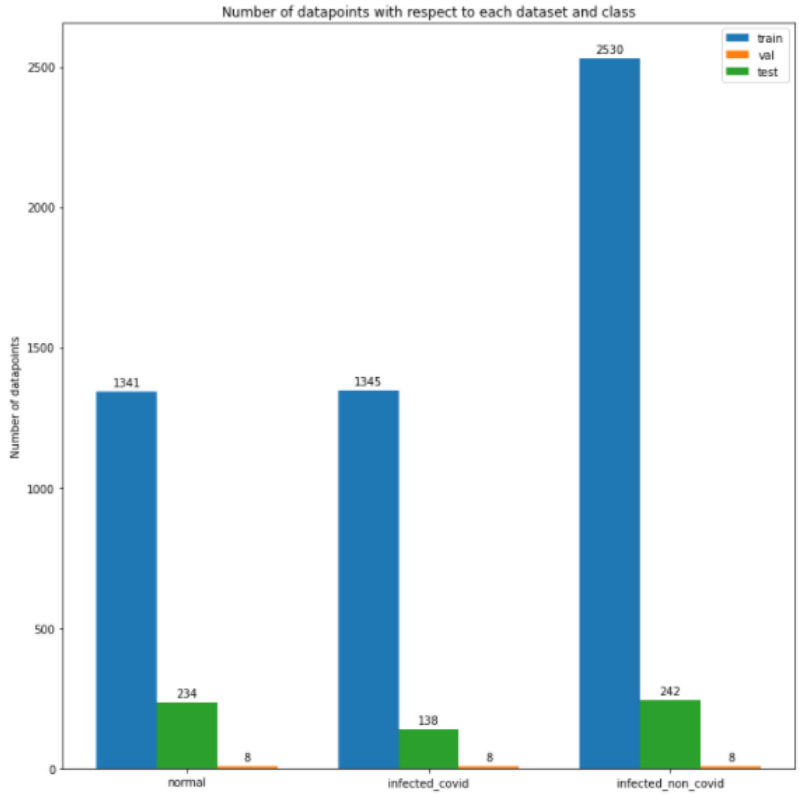
Testing set

The test set is also slightly imbalanced with the ratio `normal:infected_covid:infected_non_covid` being approximately `2:1:2`. However, this is not as bad as an imbalanced training set because the test set will not only be used to get a gauge of the model's performance after each epoch and will not directly affect the model's parameter tuning.

Validation set

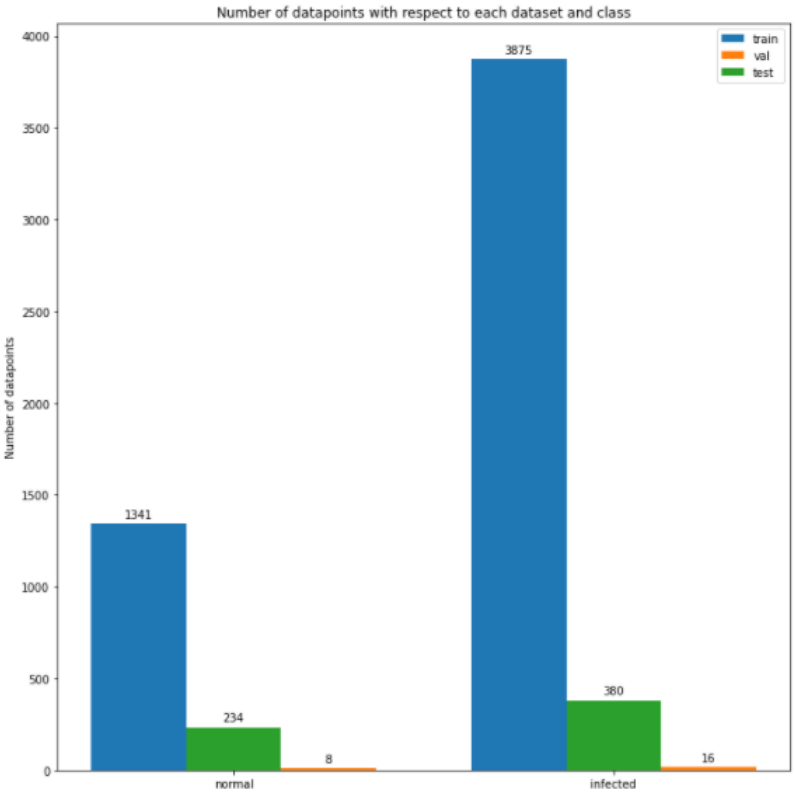
The val set is uniformly distributed between the three classes. However, it is glaring that there are only 8 validation samples for the 3 classes. However, this is acceptable as it is used only for validation and will not affect the model's parameter tuning.

Overall class distribution



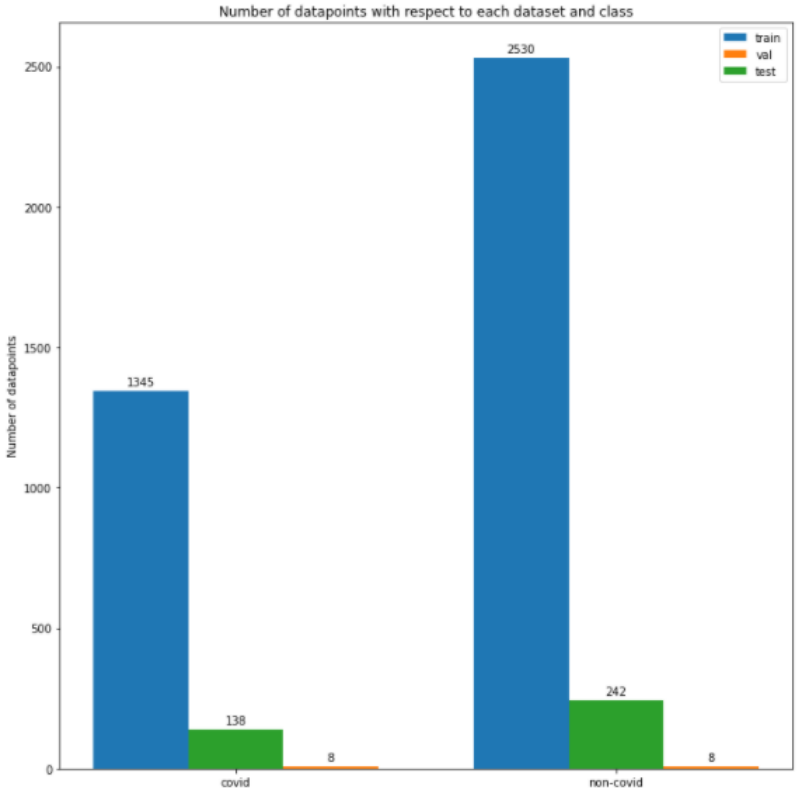
Distribution for Layer 0 of the 2 binary classifier approach - normal vs infected

The graph below gives a Clearer idea about the training imbalance where the ratio of `normal:infected` would be a ratio of `1:3` which are the ratios used when training the binary classifier for Layer 0.



Distribution for Layer 1 of the 2 binary classifier approach - covid vs non_covid

The class distribution for infected_covid vs infected_non_covid is imbalanced by about 1:2. This represents what is likely in reality. Covid is new and will probably not have as many data samples as the common infected non-covid X rays. Again, possible data augmentations techniques like mirroring could potentially be used to increase the number of training samples for covid.



4.3 Why normalize the data

To recap the normalization can be found in the `Lung_Dataset` class' `open_img` function which did the following normalization.

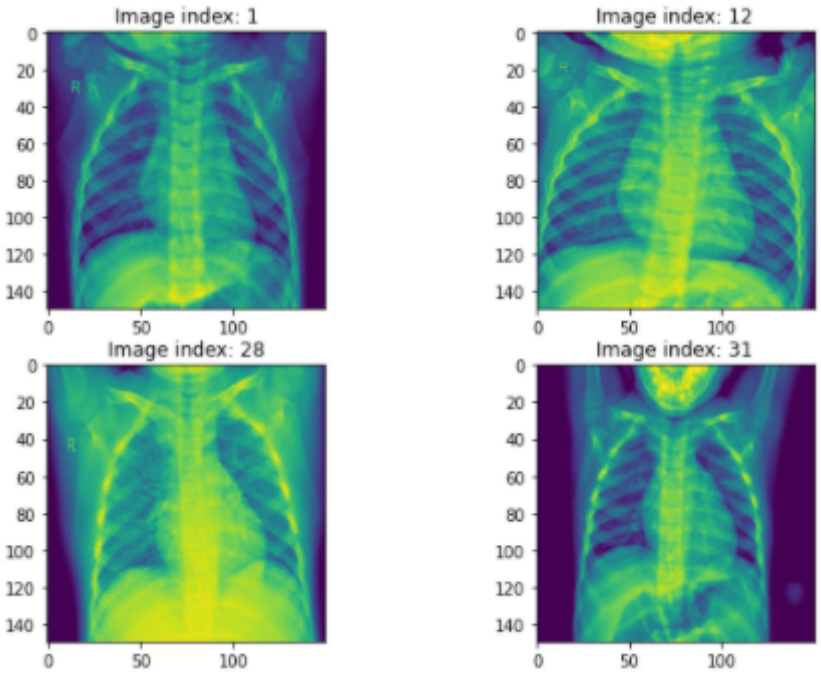
```
# Convert to Numpy array and normalize pixel values by dividing by 255.
im = np.asarray(Image.open(f))/255
```

Images have RGB ranges from 0-255. Considering various activation functions like `sigmoid` such a large range would mean that for vastly different values like 100 and 255, not much difference can be seen when passed into the `sigmoid` activation function. Both would produce a value that is close to 1.

Taking the same values as reference, if we divide by 255, for a value of 100, $\frac{100}{255}$ we get approximately 0.39. Then for a value of 255, $\frac{255}{255}$ we get 1. For the initial value of 100 that becomes 0.39 after the division, passing it into `sigmoid(0.39)` produces a value of 0.596. Meanwhile for the initial value of 255 that becomes 1 after division, passing it into `sigmoid(1)` produces a value of 0.731. This difference in value allows us to extract meaningful differences in the pixel values.

4.4 Other potential pre-processing operations

From the plot below, which is based on the Training set for normal images as reference, it is evident that there are several differences in the photo dimensions and photo environment.



For example, comparing image_index 1 and image_index 28 there is a clear difference in the lighting, Image_index 28 is a lot brighter. One pre-processing step could be to use histogram normalization. There is a paper that recommends 14 possible normalization algorithms that can be performed (Leszczynski, 2010).

Another example is comparing "skinny" images like image_index 1 and image_index 31 where there is significantly more dark backgrounds at the side compares to images like image_index 12. Perhaps a edge detection algorithm can be applied to just filter the relevant parts of the image which are the lungs.

4.5 Possible data augmentations

Reference link on how Doctors diagnose Covid-19: [The role of chest radiography in confirming covid-19 pneumonia]

Relevant insight:

- "covid-19 pneumonia changes are mostly **bilateral on chest radiographs** (72.9%, 95% confidence interval 58.6 to 87.1) and have **ground glass opacity** in 68.5% of cases (95% CI 51.8 to 85.2)"
- The paper provides several images and analyzes them. From the images, it seems that the target areas are in the 4 corners of the lungs, perhaps data augmentation can perform crops to those areas

How it can be used in the model?

- Problem: The data classes are imbalanced for covid and non-covid and that the model has more problems differentiating the 2 compared to differentiating normal and infected. Solution: Since covid-19 is `bilateral` meaning that it affects both sides of the lungs at a high probability. Perhaps mirroring images could be a possible approach to increase the size of the dataset for covid images, giving the model a more balanced dataset and more covid images to recognize

5. Proposed model

5.1 3-class classifier vs 2 binary classifiers

Differences between the 2 architectures

The 3-class classifier architecture classifies the x-rays according to normal, infected COVID and infected non-COVID cases. With this architecture, the overall classification is reliant on only one model which reduces the training time and the number of parameters that need to be tuned. This is a multi-class classifier that uses a tensor of 3 values. Example, normal can correspond to `[1 0 0]`. The model thus outputs a tensor of this shape as well.

The 2 x binary classifiers is more specific and tackles 2 sequential binary classification problems. In this project, the top layer will classify normal vs infected x-rays. If the output of the first layer is infected, it will be fed to the next layer which classifies COVID vs non-COVID x-rays. This approach is advantageous as each models are able to learn distinct features that are catered towards their primary target. For example, it seems intuitive that there would be distinct features between the normal and infected X-rays and this is elaborated in published medical papers where doctors look for certain tell-tale signs in lungs such as obscured line markings of the lings (see section 10). The second classifier is based on the hypothesis that there is a difference between infected non covid lungs and infected covid lungs. This is also briefly mentioned in section 10. In real life the efficacy of differentiating between the 2 is not that great. Hence, to improve the model's chances of differentiating the 2, it makes more sense to have a dedicated model to learn the subtle differences and hopefully have better predictions.

Why we chose the 2 binary classifier approach

As the number of dataset is generally low, using the first architecture, 2 binary classifier, would allow us to tap onto complementary datasets to train each of the models (for example, the number of infected cases comprises the covid and non covid cases, effectively "increasing" the training data for infected). Furthermore, having two layers will allow the flexibility to tune individual hyperparameter to improve the overall accuracy for that specific model.

To confirm our hypothesis, we did an exploratory analysis using both models and evaluate their effectiveness.

Empirical observations

Methodology: We setup 2 notebooks to test the 3-class classifier and the 2 x binary cascade classifier. Within those notebooks, the same models are used for all. All parameters are set as their default values. The only tuning done is to tune the number of epochs to ensure that the models are not overfitted.

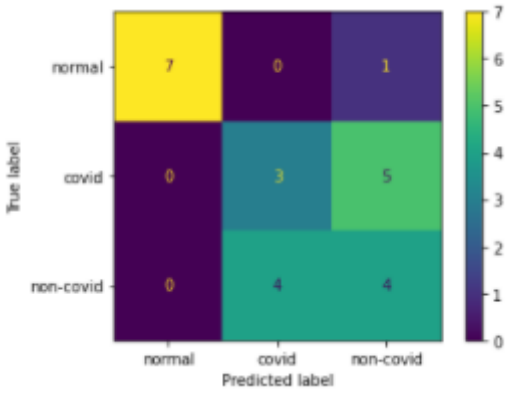
The notebooks can be found at

- 3-class classifier: `./notebooks/colab/experiments/architecture_selection/multi_class_classifier.ipynb`
- 2 x binary classifiers: `./notebooks/colab/experiments/architecture_selection/naive_classifier.ipynb`

Result for 3-class classifier

Validation Accuracy: 14/24 (58.3%)

Confusion matrix on Validation set:



	Normal	Covid	Non-covid
Recall	0.87	0.37	0.50
Precision	1.00	0.42	0.40
f1_score	0.93	0.39	0.44

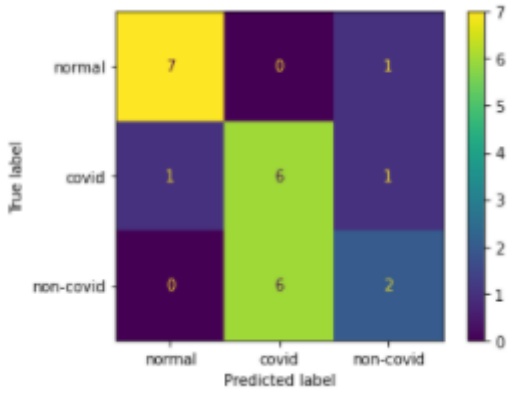
The results for loss and accuracy on the train and test set during training can be found at

- `./results/experiments/architecture_selection/multi_net.txt`

Result for 2 x binary classifiers:

Validation Accuracy: 15/24 (62.5%)

Confusion matrix on Validation set:



	Normal	Covid	Non-covid
Recall	0.87	0.75	0.25
Precision	0.87	0.50	0.50
f1_score	0.87	0.60	0.33

The results for loss and accuracy on the train and test set during training can be found at

- `./results/experiments/architecture_selection/naive_binary_L0_net.txt`
- `./results/experiments/architecture_selection/naive_binary_L1_net.txt`

Conclusion

The results without any tuning of parameters for both architectures are largely equal. It is interesting that the multi-class one predicts normal classes better and the 2 x binary one predicts Covid better. Predicting Covid well aligns more with the desired outcomes of the model. Based on our hypothesis of the proposed advantages of the 2 x binary model, we will proceed with it and propose and tune models that aim to improve on the baseline scores above.

5.2 2 2 Binary classifiers architecture design

5.2.1 Referencing literature and traditional well-performing models

Methodology

- There are many possible models to select from. To simplify the model selection process, we reference literature that propose novel architectures that solved similar problems.
- To benchmark the custom models, we also experimented using the naïve single convolution model provided in the starter notebook
- For a more robust benchmark, we modified the resnet architecture, using only a single layer of resnet for the experiments. The rationale was that the dataset was rather small and having too many layers would probably not work as well as they will likely overfit on the training data and this eventually results in poor performance for the Validation data. We chose resnet because it has generally performed well on image classification problems. For example, a paper by Talo, M. uses resnet's Convolutional Neural Networks for Multi-class Histopathology Image Classification and has achieved decent performance for the metrics they chose
- Finally, the custom network chosen was based of the paper COVINet: a convolutional neural network approach for predicting COVID-19 from chest X-ray images. The model proposed was tested on different number of classes, and has achieved a high accuracy.
- To test the performance of these models, we use defaults for the optimizer, learning rate. The only parameter that we adjust is the number of epochs, based on trial and error. This meant running the models on a large number of epochs and eventually selecting the best epoch number for each model.

Naïve single convolution model

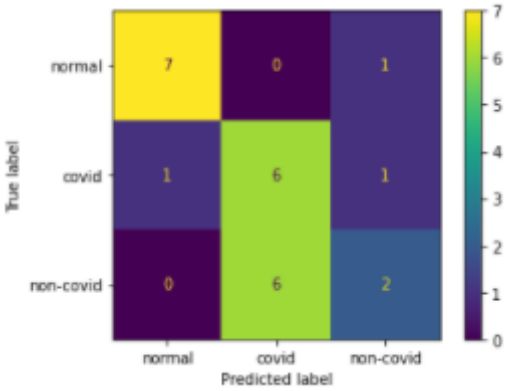
The naïve single convolution model was the model provided in the starter notebook. The notebook for this experiment can be found at `./notebooks/colab/experiments/binary_selection/naive_classifier.ipynb`. After initially experimenting using a large number of epochs, the first classifier uses 3 epochs and the second classifier uses 4 epochs before they overfit.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 4, 148, 148]              40
            Linear-2                   [-1, 2]         175,234
================================================================
Total params: 175,274
Trainable params: 175,274
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.09
Forward/backward pass size (MB): 0.67
Params size (MB): 0.67
Estimated Total Size (MB): 1.42
----------------------------------------------------------------
```

Result:

Validation Accuracy: 15/24 (62.5%)

Confusion matrix on Validation set:



	Normal	Covid	Non-covid
Recall	0.87	0.75	0.25
Precision	0.87	0.50	0.50
f1_score	0.87	0.60	0.33

The results for loss and accuracy on the train and test set during training can be found at

- `./results/experiments/binary_selection/naive_binary_L0_net.txt`
- `./results/experiments/binary_selection/naive_binary_L1_net.txt`

Re-implementing a scaled down version of resnet

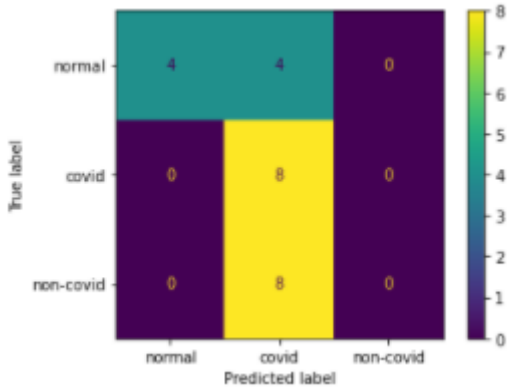
The architecture for this model was motivated by resnet. In terms of number of channels, kernel size, stride and padding, the single layer mimics the resnet architecture. A single layer was chosen because the dataset provided is small and a full resnet implementation will likely overfit on the data. The notebook for this experiment can be found at `./notebooks/colab/experiments/binary_selection/resnet_classifier.ipynb`. The model was implemented by hand and trained from scratch.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 75, 75]           3,136
       BatchNorm2d-2          [-1, 64, 75, 75]             128
            Conv2d-3          [-1, 64, 38, 38]          36,864
       BatchNorm2d-4          [-1, 64, 38, 38]             128
              ReLU-5          [-1, 64, 38, 38]               0
            Conv2d-6          [-1, 64, 38, 38]          36,864
       BatchNorm2d-7          [-1, 64, 38, 38]             128
 AdaptiveAvgPool2d-8            [-1, 64, 1, 1]               0
            Linear-9                 [-1, 1000]          65,000
================================================================
Total params: 142,248
Trainable params: 142,248
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.09
Forward/backward pass size (MB): 9.03
Params size (MB): 0.54
Estimated Total Size (MB): 9.66
----------------------------------------------------------------
```

Result: 12/24 (50%)

Validation Accuracy:

Confusion matrix on Validation set:



Relevant metrics on validation set:

	Normal	Covid	Non-covid
Recall	0.50	1.00	0.00
Precision	1.00	0.40	0.00
f1_score	0.66	0.57	0.00

Very clearly this model is not suitable because all the infected predictions are just classified as covid. In real life if this were to happen, there would be a severe strain on medical resources and facilities.

The results for loss and accuracy on the train and test set during training can be found at

- `./results/experiments/binary_selection/simple_resnet_L0_net.txt`
- `./results/experiments/binary_selection/simple_resnet_L1_net.txt`

Models from literature that tackled similar problems - COVINet

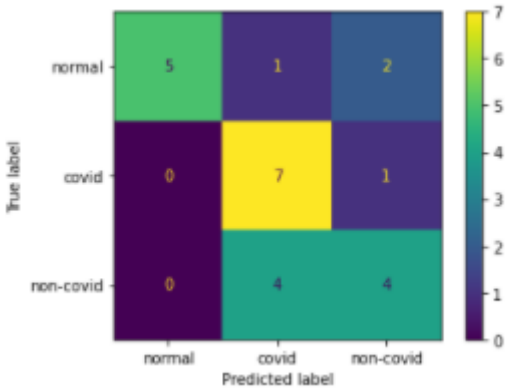
The architecture for this model was motivated from the following paper: [COVINet: a convolutional neural network approach for predicting COVID-19 from chest X-ray images](). This model performed the best among the models. From the architecture, it is around the same complexity as the modified resnet but it has more layers than the naïve classifier. The increase in number of convolution layers means that lower layers can learn lower level features and subsequent layers the higher level features. Dropout layers at different parts of the architecture also help to reduce overfitting. The notebook for this experiment can be found at `./notebooks/colab/experiments/binary_selection/COVINet_classifier.ipynb`. The model was implemented by hand and trained from scratch. A slight tweak was made to model 2 to add a stride of size 2 to the max pooling layer. This was found to improve the model's performance significantly.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 8, 148, 148]              80
         Dropout2d-2          [-1, 8, 148, 148]               0
              ReLU-3          [-1, 8, 148, 148]               0
            Conv2d-4         [-1, 16, 146, 146]           1,168
              ReLU-5         [-1, 16, 146, 146]               0
         MaxPool2d-6           [-1, 16, 73, 73]               0
         Dropout2d-7           [-1, 16, 73, 73]               0
            Conv2d-8           [-1, 32, 71, 71]           4,640
              ReLU-9           [-1, 32, 71, 71]               0
        AvgPool2d-10           [-1, 32, 23, 23]               0
        Dropout2d-11           [-1, 32, 23, 23]               0
             ReLU-12           [-1, 32, 23, 23]               0
           Linear-13                  [-1, 2]          33,858
================================================================
Total params: 39,746
Trainable params: 39,746
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.09
Forward/backward pass size (MB): 13.36
Params size (MB): 0.15
Estimated Total Size (MB): 13.60
----------------------------------------------------------------
```

Result:

Accuracy on validation set: 16/24 (66.7%)

Confusion matrix on Validation set:



Relevant metrics on validation set:

	Normal	Covid	Non-covid
Recall	0.62	0.875	0.50
Precision	1.00	0.58	0.57
f1_score	0.76	0.70	0.53

The results for loss and accuracy on the train and test set during training can be found at

- `./results/experiments/binary_selection/COVINet_L0_net.txt`
- `./results/experiments/binary_selection/COVINet_L1_net.txt`

Conclusion

Overall it is evident that the custom COVID model that was specifically designed for this type of problem performs better than the other 2 baselines. The better performance can be seen in most metrics used such as recall, precision, and accuracy. In the context, of Covid, the important metrics to consider would be the Recall and Precision scores for Covid. Both of these metrics are better than the other models. Moving forward, we will tune the parameters for COVINet.

5.2.2 Model parameters

The architecture was motivated by the following paper: COVINet: a convolutional neural network approach for predicting COVID-19 from chest X-ray images. This section seeks to understand the choice of layers and parameters chosen by this paper. To recap, the model is as follows:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 8, 148, 148]              80
         Dropout2d-2         [-1, 8, 148, 148]               0
              ReLU-3         [-1, 8, 148, 148]               0
            Conv2d-4        [-1, 16, 146, 146]           1,168
              ReLU-5        [-1, 16, 146, 146]               0
         MaxPool2d-6          [-1, 16, 73, 73]               0
         Dropout2d-7          [-1, 16, 73, 73]               0
            Conv2d-8          [-1, 32, 71, 71]           4,640
              ReLU-9          [-1, 32, 71, 71]               0
        AvgPool2d-10          [-1, 32, 23, 23]               0
        Dropout2d-11          [-1, 32, 23, 23]               0
             ReLU-12          [-1, 32, 23, 23]               0
           Linear-13                   [-1, 2]          33,858
================================================================
Total params: 39,746
Trainable params: 39,746
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.09
Forward/backward pass size (MB): 13.36
Params size (MB): 0.15
Estimated Total Size (MB): 13.60
----------------------------------------------------------------
```

```
class L1_Net(nn.Module):
    def __init__(self, num_layers=1):
        super(L1_Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 8, 3, 1)
        self.dropout1 = nn.Dropout2d(0.2)
        self.relu = nn.ReLU()
        self.conv2 = nn.Conv2d(8, 16, 3, 1)
        self.conv3 = nn.Conv2d(16, 32, 3, 1)
        self.avgpool = nn.AvgPool2d(3,3)
        self.maxpool = nn.MaxPool2d(2, stride=2)
```

```
        self.fc1 = nn.Linear(16928, 2)

    def forward(self, x):
        x = self.conv1(x)
        x = self.dropout1(x)
        x = self.relu(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.maxpool(x)
        x = self.dropout1(x)
        x = self.conv3(x)
        x = self.relu(x)
        x = self.avgpool(x)
        x = self.dropout1(x)
        x = self.relu(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        output = F.log_softmax(x, dim = 1)
        return output
```

Convolution layers

One of the approaches to designing Convolution layers is to stack convolution layers as mentioned in class. The initial convolution layers will learn the low level features of the images. Subsequent layers can interpret these extracted features and could have a deeper feature representation.

The original model proposed by the paper uses 1->16->128->256 transitions between convolution layers. This has a large number of parameters which makes sense for their problem because their dataset consists of 10,000 images. In this problem, the dataset we are using only has about 5854 images. Hence, having a smaller output size and thus lesser parameters would likely reduce the likelihood of overfitting.

In terms of kernel size, the standard 3x3 kernel is used. The stride is set as 1 as recommended by the paper.

Dropout layers

Dropout means that at training time, we randomly set 1-p of all neurons to 0. This is a way of adding noise to the learning problem. For instance, if there is a certain pattern that consistently occurs in the training data but not in any other dataset, then setting the neurons that identify this pattern to 0 will allow the model to forget these training set specific patterns. This can help to model to generalize better to reduce the likelihood of overfitting. For this model the paper recommended setting the `p` value as `0.2` which means that the probability of a neuron to be zeroed is 0.2.

Relu layers

Relu layers help to overcome the vanishing gradient problem. Relu outputs a true 0 value for values <= 0. This true 0 values are known as a spare representation and can help speed up learning and simplify the model. For values >0, they are output in a linear fashion. Gradients thus remain proportional to node activations and help to prevent vanishing gradients.

Pooling layers

Pooling has no parameters to learn and it serves to aggregate parameters at each "window" where the kernel slides. This is faster than convolutions because pooling layers have no parameters to learn. There is also argument that max pooling helps to extract the sharpest images from an image. In the case of covid images, some parts of the lungs are identified. According to Cleverly et al. (2020), doctors look out for abnormalities in the **heart, mediastinum, lungs, diaphragm, and ribs** for x-rays. Perhaps these could result in sharp features that are picked up by the pooling layers.

For the average pooling the standard kernel size of 3 is used. The stride used is 3 as recommended in the paper. Perhaps this is to capture more independent, non overlapping windows. This ensures that each output is not so dependent on the adjacent windows.

Similarly for max pooling the same logic is applied except for a smaller kernel size as recommended by the paper.

Linear layers

The last linear layer simply outputs a tensor with shape [1 2] corresponding to the probability of a data point being classified as either of the 2 classes.

5.2.3 Mini-batch size

Advantage of using a batch size

- Reduce computations since the mean and variance of the whole training set does not have to be computed. This speeds up training.
- Batch normalization also means that the gradient with respect to the inputs does not depend on the scale of the weights

Some considerations is that batch size should not be below 8 because then the mean and variance will not be stable.

At the same time, the link at https://stats.stackexchange.com/questions/164876/what-is-the-trade-off-between-batch-size-and-number-of-iterations-to-train-a-neu suggests the pitfalls of having a batch size that is too large. If the batch size is too large, this could degrade the model's ability to generalize as they converge to "sharp minimizers of the training and testing functions"

Recommendations

Reference links

- https://ai.stackexchange.com/questions/8560/how-do-i-choose-the-optimal-batch-size
- https://stats.stackexchange.com/questions/164876/what-is-the-trade-off-between-batch-size-and-number-of-iterations-to-train-a-neu

The general rule of thumb for batch size would be `32, 64, 128`

Since the dataset is small, we will experiment with `16,32,64`

- 32 was already done in the pervious section

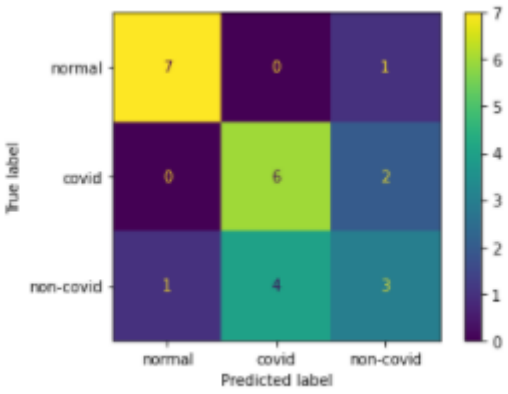
Batch size of 16:

The notebook for this experiment can be found at `./notebooks/colab/experiments/batch_size/COVINet_classifier.ipynb`. This notebook is configured to test batch size 16.

Result:

Accuracy on validation set: 16/24 (66.7%)

Confusion matrix on Validation set:



Relevant metrics on validation set:

	Normal	Covid	Non-covid
Recall	0.87	0.75	0.375
Precision	0.87	0.60	0.50
f1_score	0.87	0.66	0.42

The results for loss and accuracy on the train and test set during training can be found at

- `./results/experiments/batch_size/COVINet_L0_net_16.txt`
- `./results/experiments/batch_size/COVINet_L1_net_16.txt`

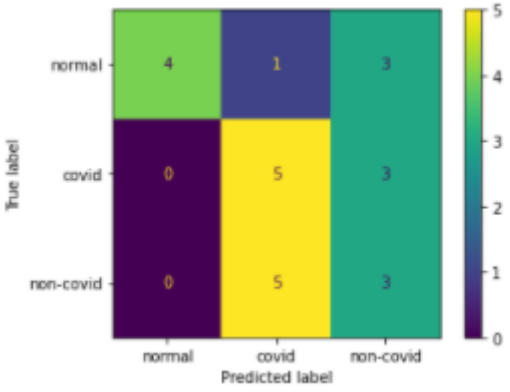
Batch size of 64:

The notebook for this experiment can be found at `./notebooks/colab/experiments/binary_selection/COVINet_classifier.ipynb`, just edit the batch size to 64.

Result:

Accuracy on validation set: 12/24 (50.0%)

Confusion matrix on Validation set:



Relevant metrics on validation set:

	Normal	Covid	Non-covid
Recall	0.50	0.62	0.37
Precision	1.00	0.45	0.33
f1_score	0.66	0.52	0.35

The results for loss and accuracy on the train and test set during training can be found at

- `./results/experiments/batch_size/COVINet_L0_net_64.txt`
- `./results/experiments/batch_size/COVINet_L1_net_64.txt`

Overall

In terms of model performance, batch_size 16 and 32 do not make any difference. batch_size 64 results in a poorer model. Hence batch_size 32 is eventually chosen moving forward because it is the more common batch_size and also because it can result in faster training without much tradeoff in terms of model performance.

5.2.4 Loss function

Why cross-entropy loss?

We used cross-entropy loss when we were experimenting with the 3 class classifier as cross entropy loss is ideal for multi-class networks. We decided to thus preserve the code across to the 2 x binary classifier model. Cross entropy loss also allows us to set weights on the different classes to account for their class imbalances. For example, for the first binary classifier for normal:infected, the ratio is about 1:2.88. To account for this, we can set the same ratios using the following line:

```
l0_class_weights = torch.tensor([2.88, 1.0]).to(torch.device(device))
criterion = nn.CrossEntropyLoss(l0_class_weights)
```

Experimenting with weighted cross-entropy to account for imbalanced classes

The ratios of the classes are as follows

- Normal:infected = 1341:3875 = approx 1:2.88
- covid:non_covid = 1345:2530 = approx 1:1.88

For normal:infected, it makes sense to favor the infected case a bit more in predictions since it is safer to misclassify a normal person as an infected person rather than an infected person as a normal person. In the latter case, this is dangerous for the public as that person may spread the infection to other people. Hence, the weights chosen are 2.83:1

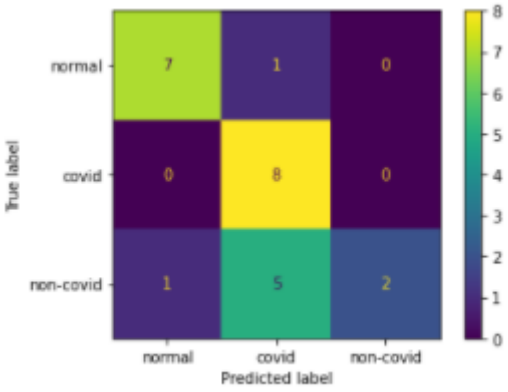
For covid:non_covid, we favor covid since covid is more deadly than non_covid infections. Hence, we choose a 1.9:1 ratio for the weighs

While experimenting, the epoch counts were tweaked to ensure optimum performance.

Results

Accuracy on validation set: 17/24 (70.8%)

Confusion Matrix on Validation set::



Relevant metrics on validation set:

	Normal	Covid	Non-covid
Recall	0.87	1.0	0.25
Precision	0.87	0.57	1.00
f1_score	0.87	0.72	0.40

The results for loss and accuracy on the train and test set during training can be found at

- `./results/experiments/cross_entropy_weights/COVINet_L0_net.txt`
- `./results/experiments/cross_entropy_weights/COVINet_L1_net.txt`

There is slight improvement once the weights are added. Hence, these weights will be used in the final model once the optimizers and learning rates have been optimized

5.3 Optimizer

5.3.1 Optimizer selection

Adam vs AdamW theory

For Optimizers like Adam and AdamW, the L2 Regularization is implemented on the optimizer using `weight_decay`

The parameter for weight decay in pytorch is as follow: weight_decay (float, optional) – weight decay (L2 penalty) (default: 0)

The lectures from week 5 also mentions that for SGD, the effect of weight decay is the same as doing l2-regularization. This can potentially help reduce overfitting.

The core difference between AdamW and Adam is the regularization pattern. For Adam, the weight decay ends up with the moving average while AdamW ensures that the regularization term does not, which makes the regularization proportional to the weight itself.

Theoretically, AdamW should yield better training loss and that the models generalize much better than models trained with Adam.

Adam vs AdamW empirical

Log: results/experiments/final_tuning_layer_0_hyperparameters.log

Epochs	Learning Rate	Scheduler Gamma	Weight Decay	Optimizer	Test Accuracy	Val Average Precision	Val Average Recall
10	0.0001	1	0	Adam	0.7495	0.9700	0.9375
10	0.0001	1	0	AdamW	0.8125	0.9700	0.9375

From the table, we can see that with AdamW optimizer, the test accuracy is slightly higher than Adam in layer 0. We pick AdamW as the optimizer for layer 0 for initial fine tuning.

5.3.2 Regularization - Weight Decay

The weight decay parameter is used as a L2 regularization in the Adam optimizer. L2 regularization is used to alleviate overfitting of the model.

Log: results/experiments/final_tuning_layer_0_hyperparameters.log

Epochs	Learning Rate	Scheduler Gamma	Weight Decay	Optimizer	Test Accuracy	Val Average Precision	Val Average Recall
10	0.0001	1	0	Adam	0.7495	0.9700	0.9375
10	0.0001	1	0.00001	Adam	0.7870	0.9211	0.8125
10	0.0001	1	0.01	Adam	0.7729	0.9444	0.875

With all other hyperparameters equal, it is found that both weight decay of 0.00001 and weight decay of 0.01 have a lower average precision and average recall than the weight decay of 0. However, one advantage of having the weight decay is to prevent overfitting. We will be using decay weight of 0 for initial fine tuning.

5.3.3 Learning rate

Experimenting with different learning rates

Log: results/experiments/final_tuning_layer_0_hyperparameters.log

Epochs	Learning Rate	Scheduler Gamma	Weight Decay	Optimizer	Test Accuracy	Average Precision	Average Recall
10	0.00001	1	0	Adam	0.8260	0.7945	0.7187
10	0.0001	1	0	Adam	0.7495	0.9700	0.9375
10	0.001	1	0	Adam	0.7568	0.9063	0.9063

From the experiment, we can see that both the average precision and average recall for learning rate 0.00001 and 0.001 are lower than learning rate of 0.0001. Since we prioritize precision and recall, we will choose learning rate of 0.0001 for initial fine tuning. This give us a rough number to get started when implementing the final model.

5.3.4 Scheduled learning rate

Log: results/experiments/final_tuning_layer_0_hyperparameters.log

Under the hyperparameter folder, we have experimented with the scheduled learning rate to gauge the effectiveness of implementing the scheduler. In the experiment, pytorch's StepLR was used with step size of 5 with variance of gamma of [1, 0.001] . The gamma determines the decay of the rate of the learning rate at after every predetermined step size.

As the epoch increases, it moves closer the to convergence point. The learning rate scheduler helps to reduce the learning rate and the convergence speed of the model, which prevents the model from jumping out of the optima.

Epochs	Learning Rate	Scheduler Gamma	Weight Decay	Optimizer	Test Accuracy	Average Precision	Average Recall
10	0.0001	1	0	Adam	0.7495	0.9700	0.9375
10	0.0001	0.001	0	Adam	0.7896	0.9444	0.875

From the table, we can see that scheduler gamma of 1 (no decay) has better average precision and average recall. As such we will not be using scheduled learning rate in our model for initial fine tuning.

5.4 Model parameters

After further fine-tuning to an even greater extend on top of our initial experimental values, we have decided to pick the following parameters.

Layer 0

- Number of Epochs: 5
- Learning Rate: 0.001
- Weight Decay: 0
- Optimizer: AdamW
- Learning Rate Scheduler: None

Layer 1

- Number of Epochs : 10
- Learning Rate: 0.001
- Weight Decay: 0.00001
- Optimizer: AdamW
- Learning Rate Scheduler: None

5.5 Implementing checkpoints

Checkpoints are saved at every epoch to ensure that the models can be loaded for evaluation. In the event of crashing midway during training, the checkpoints can provide the state to continue training again.

The Checkpoint stores the following variables:

- Current epochs
- Model state dict
- Optimizer state dict
- training loss
- test loss
- test accuracy
- training accuracy

6. Model Summary

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 8, 148, 148]              80
         Dropout2d-2          [-1, 8, 148, 148]               0
              ReLU-3          [-1, 8, 148, 148]               0
            Conv2d-4         [-1, 16, 146, 146]           1,168
              ReLU-5         [-1, 16, 146, 146]               0
         MaxPool2d-6           [-1, 16, 73, 73]               0
         Dropout2d-7           [-1, 16, 73, 73]               0
            Conv2d-8           [-1, 32, 71, 71]           4,640
              ReLU-9           [-1, 32, 71, 71]               0
        AvgPool2d-10           [-1, 32, 23, 23]               0
        Dropout2d-11           [-1, 32, 23, 23]               0
             ReLU-12           [-1, 32, 23, 23]               0
           Linear-13                    [-1, 2]          33,858
================================================================
Total params: 39,746
Trainable params: 39,746
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.09
Forward/backward pass size (MB): 13.36
Params size (MB): 0.15
Estimated Total Size (MB): 13.60
----------------------------------------------------------------
```

```python
class L1_Net(nn.Module):
    def __init__(self, num_layers=1):
        super(L1_Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 8, 3, 1)
        self.dropout1 = nn.Dropout2d(0.2)
        self.relu = nn.ReLU()
        self.conv2 = nn.Conv2d(8, 16, 3, 1)
        self.conv3 = nn.Conv2d(16, 32, 3, 1)
        self.avgpool = nn.AvgPool2d(3,3)
        self.maxpool = nn.MaxPool2d(2, stride=2)
        self.fc1 = nn.Linear(16928, 2)

    def forward(self, x):
        x = self.conv1(x)
        x = self.dropout1(x)
        x = self.relu(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.maxpool(x)
        x = self.dropout1(x)
        x = self.conv3(x)
        x = self.relu(x)
        x = self.avgpool(x)
        x = self.dropout1(x)
        x = self.relu(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        output = F.log_softmax(x, dim = 1)
        return output
```

Layer 0

- Number of Epochs: 5
- Learning Rate: 0.001

- Weight Decay: 0
- Optimizer: AdamW
- Learning Rate Scheduler: None

Layer 1

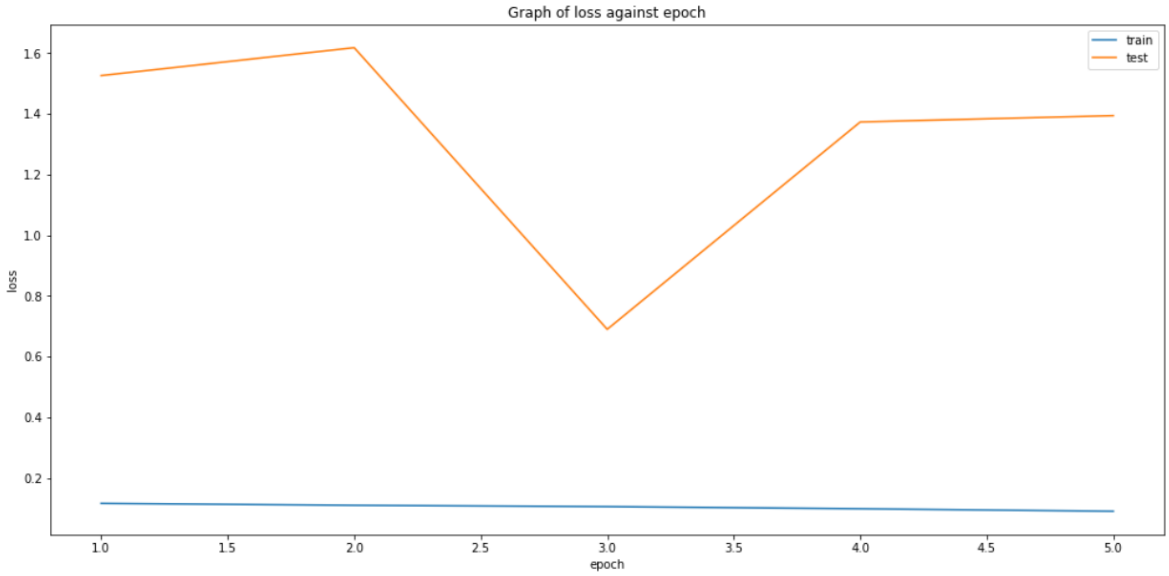
- Number of Epochs : 10
- Learning Rate: 0.001
- Weight Decay: 0.00001
- Optimizer: AdamW
- Learning Rate Scheduler: None

7. Evaluation

7.1 Learning curves

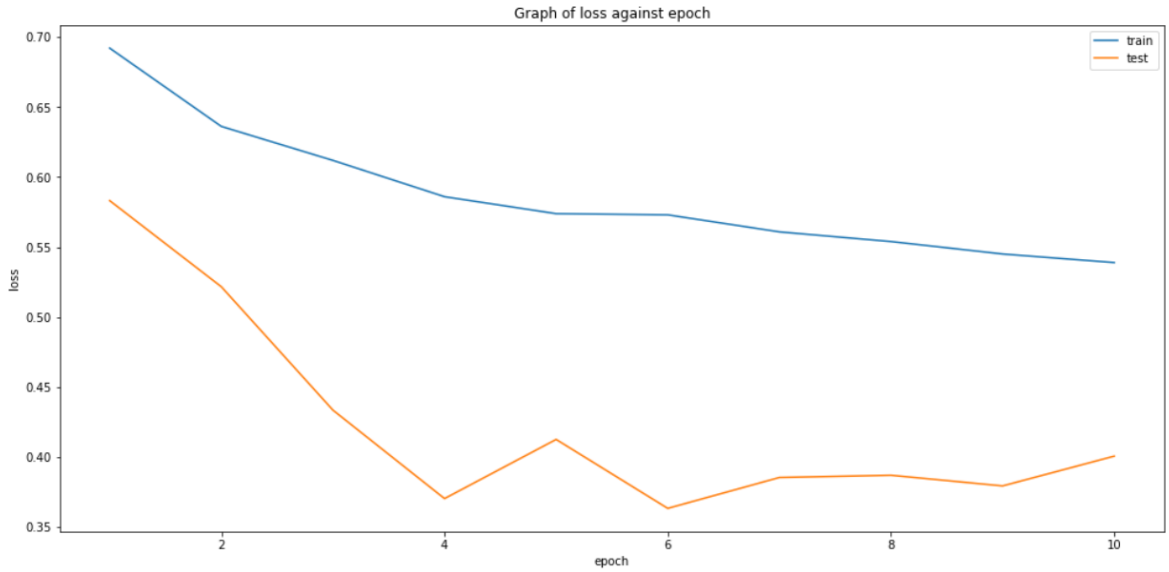
Loss vs epochs

Layer 0



We started with large number of epochs. As we proceed, we found that the optimum number of epochs ranges from 4 - 6. In this graph, we notice that the loss increases after epoch 3 and this is because the Dataloader have randomly shuffled the data. Thus, the results are non-deterministic.

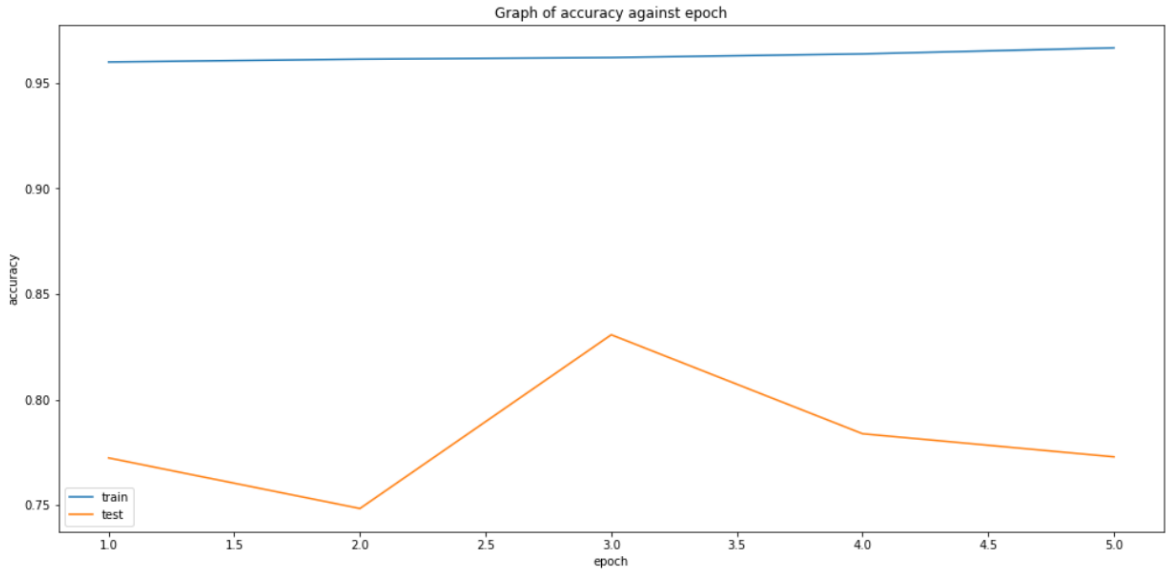
Layer 1



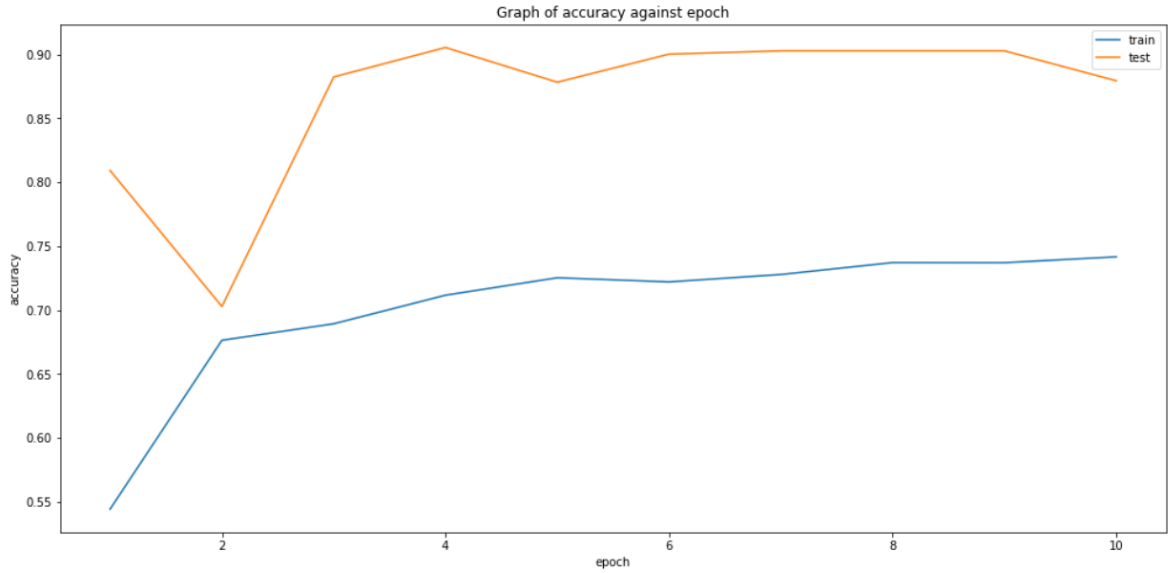
For layer 1, we found the optimum number of epochs ranges from 5 - 10.

Accuracy vs epochs

Layer 0



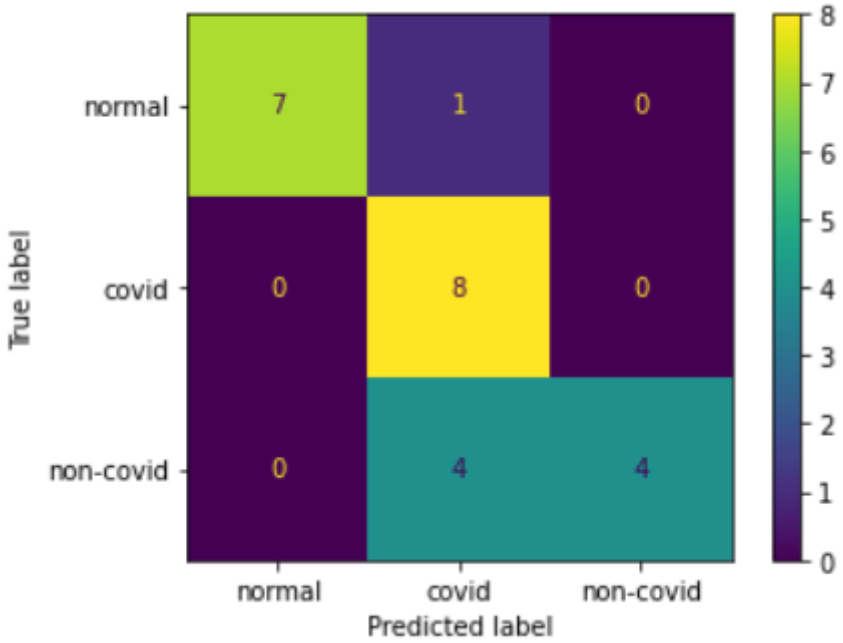
Layer 1



7.2 Key metrics and considerations

Confusion matrix

Combined Accuracy: 0.7916666666666666



Recall, Precision and F1 Score

	Precision	Recall	F1 Score
COVID	0.6153846153846154	1.0	0.761904761904762
non-COVID	1.0	0.5	0.6666666666666666
Normal	1.0	0.875	0.9333333333333333

Precision = $\frac{tp}{tp+fp}$

Recall (also Sensitivity) = $\frac{tp}{tp+fn}$

Specificity = $\frac{tn}{tn+fp}$

For this project, the metrics can be applied differently considering the needs of different stakeholders.

Considering COVID, we cite 2 stakeholders who could have different concerns

For example, the general public could be concerned about the spread of the virus, hence, they could be more worried about Recall because false negatives could mean that people with the virus in the public space and mingling with others.

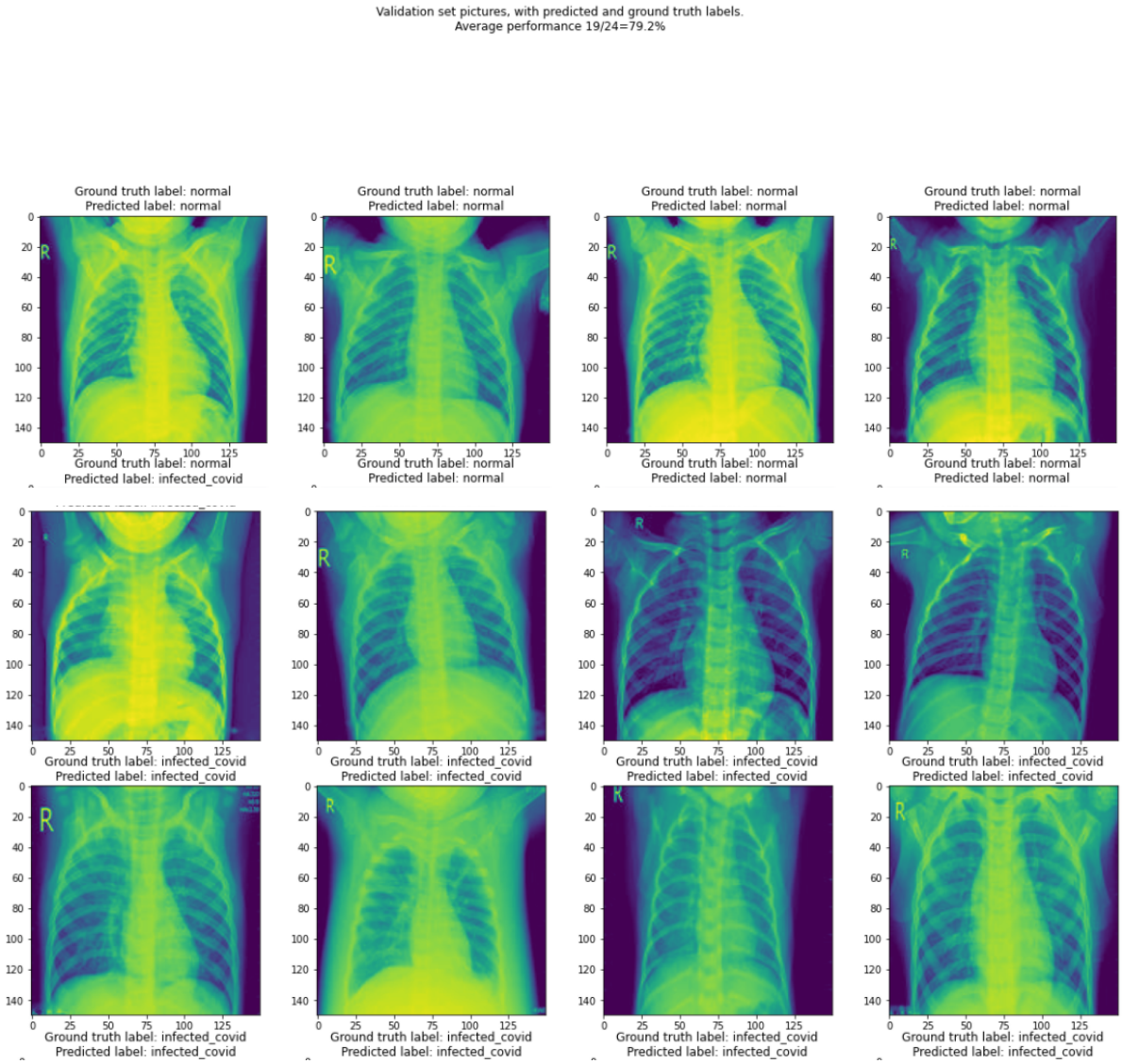
On the other hand, the clinicians could be more concerned about Precision. This is because there are limited hospital beds and it would be a waste of resources to allocated scarce resource to people who do not have covid but were predicted to have covid. This can be a problem if there is a model that predicts everyone to have covid. While this model successfully identifies all covid cases, there will be a lot of people who do not actually have covid being admitted to the hospitals for treatment. This puts a strain on resources.

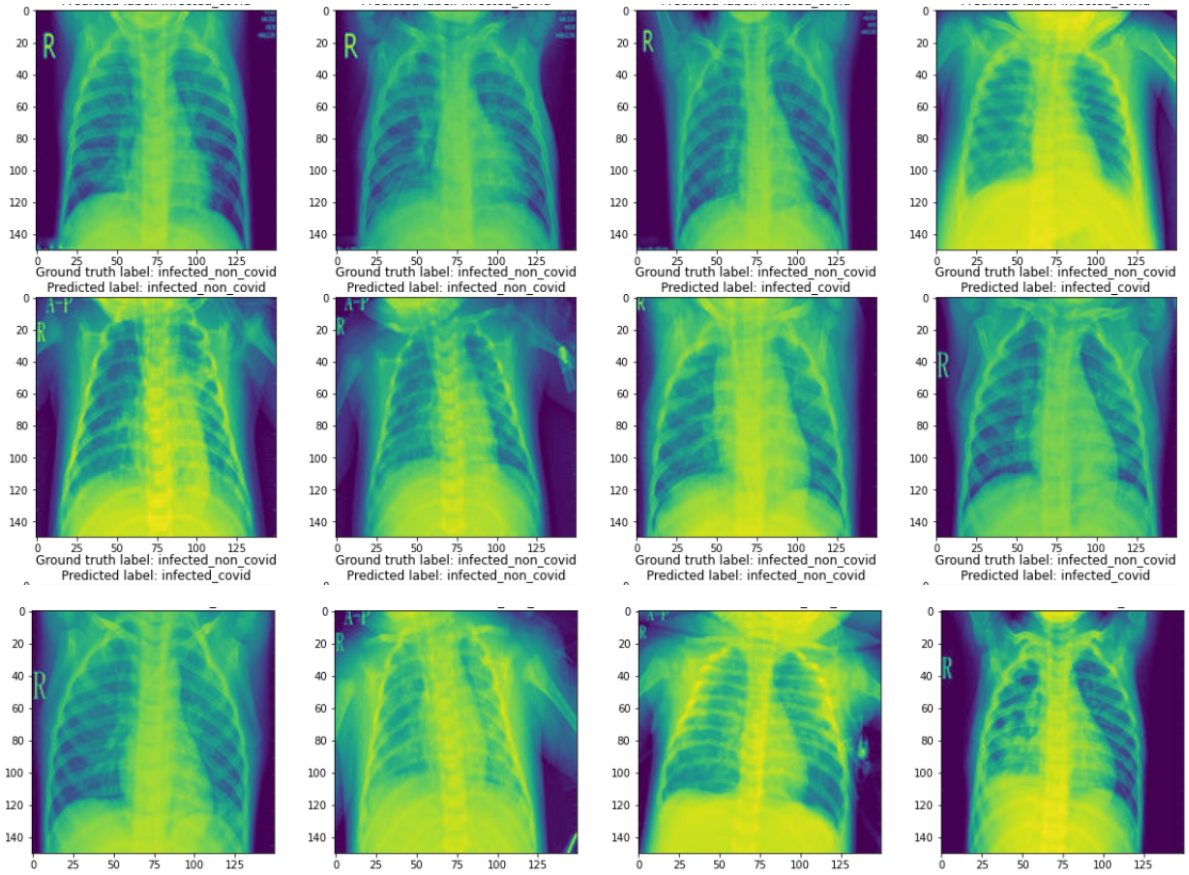
Similarly, they would be interested in high specificity which is the percentage of people who are predicted negative among all the predications cases who are truly negative. A higher value for this would similarly reduce the strain of hospital resources as it implies that those who really do not have covid are likely predicted as not having covid and will not need treatment.

For the analysis below we focus on `recall` as there was a time when the world was focused on containing the spread of covid which is aligned with the saying that "prevention is better than cure". We also output `precision` and the `f1` score to consider that other stakeholders are important as well.

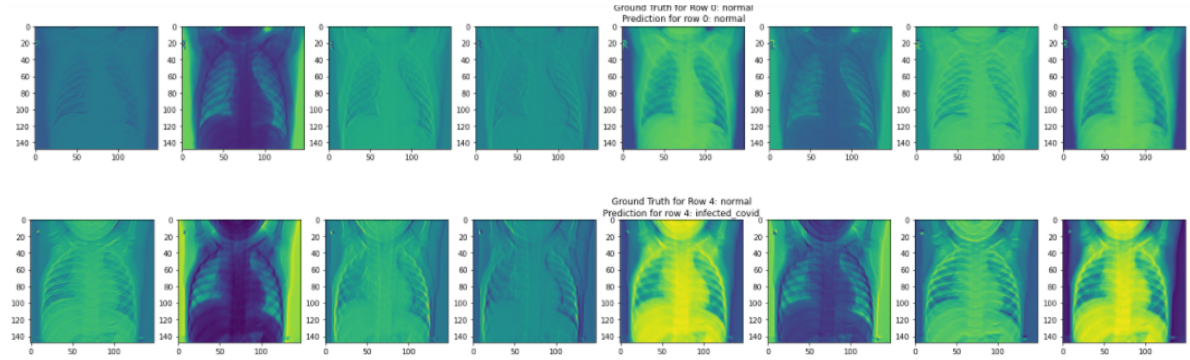
From the metrics, the important classification is that the model successfully predicted that all the people who have covid are detected as having covid. In addition, the precision is still acceptable as it is able to detect people who have infections but are non_covid. Hence, from the public's perspective, the high recall on Covid would make them feel more assured. However, a potential room for improvement would be to better differentiate the infected covid from infected non_covid as the false positives would mean that precious hospital resources would be taken up. In this case, 1 normal patient and 4 infected non-covid patients were detected to have covid. This generally speaks of the inherent difficulty of differentiating covid and non covid patients. Perhaps from the X-rays the difference between the 2 are only very minor. Considering the days since infection, early stages of covid could barely be any different from the non-covid cases.

7.3 Accuracy and image diagrams



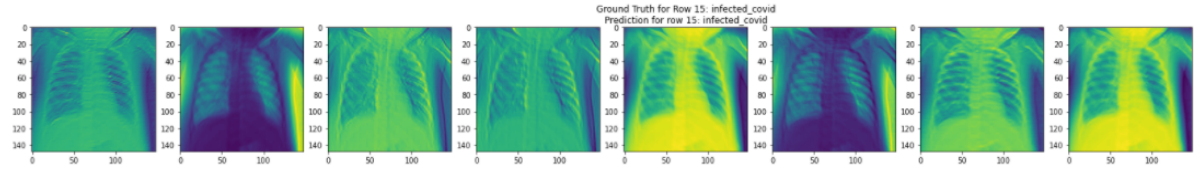


7.4 Investigating failures with feature maps



The validation image above (normal) was misclassified as an infected-COVID. We can see from above that the misclassified image has very similar feature map as a correctly classified infected-COVID image. An example will be the extreme left image.

Considering that the feature maps was created based on the first convolution layers, we can see that the low lying features will influence the result and the output.



8. Challenges of predictions

8.1 Why is 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

Intuitively infected lungs should have features that differentiate them from non-infected lungs. Clinicians would look out for tell-tale signs like the heart and diaphragm and inflammation. The model can also pick up on these features from the X-Ray and thus is able to differentiate them fairly accurately. However, differentiating covid from non covid infected lungs is much more challenging. This is even a problem for well-trained clinicians. If humans with many years of experience have trouble detecting it then machine learning models which mimic the human brain will also have difficulty.

Reference link: [How accurate is chest imaging for diagnosing COVID-19?]

- Defines X-Rays as "X-rays (radiography) use radiation to produce a 2-D image"
- This paper pooled results from several sources (if there were 4 or more results). For Chest X-rays, it found that the Chest X-ray "correctly diagnosed COVID-19 in 80.6% of the people who had COVID-19. However it incorrectly identified COVID-19 in 28.5% of the people who did not have COVID-19"

9. Overall - what is the better model, accuracy vs low true negatives/false positives rates on certain classes

Precision = $\frac{tp}{tp+fp}$

Recall (also Sensitivity) = $\frac{tp}{tp+fn}$

Specificity = $\frac{tn}{tn+fp}$

For this project, the metrics can be applied differently considering the needs of different stakeholders.

Considering Covid, we cite 2 stakeholders who could have different concerns

For example, the general public could be concerned about the spread of the virus, hence, they could be more worried about Recall because false negatives could mean that people with the virus in the public space and mingling with others.

On the other hand, the clinicians could be more concerned about Precision. This is because there are limited hospital beds and it would be a waste of resources to allocated scarce resource to people who do not have covid but were predicted to have covid. This can be a problem if there is a model that predicts everyone to have covid. While this model successfully identifies all covid cases, there will be a lot of people who do not actually have covid being admitted to the hospitals for treatment. This puts a strain on resources.

Similarly, they would be interested in high specificity which is the percentage of people who are predicted negative among all the predictions cases who are truly negative. A higher value for this would similarly reduce the strain of hospital resources as it implies that those who really do not have covid are likely predicted as not having covid and will not need treatment.

For the analysis below we focus on `recall` as there was a time when the world was focused on containing the spread of covid which is aligned with the saying that "prevention is better than cure". We also output `precision` and the `f1` score to consider that other stakeholders are important as well.

10. How doctors diagnose infections based on x-rays.

Reference link: How accurate is chest imaging for diagnosing COVID-19?

- Defines X-Rays as "X-rays (radiography) use radiation to produce a 2-D image"
- This paper pooled results from several sources (if there were 4 or more results). For Chest X-rays, it found that the Chest X-ray "correctly diagnosed COVID-19 in 80.6% of the people who had COVID-19. However it incorrectly identified COVID-19 in 28.5% of the people who did not have COVID-19"

Critical insight

- It seems that classifications for X-rays have room for improvement, the numbers indicated above would provide a baseline for any model's performance

Citation:

> Islam N, Ebrahimzadeh S, Salameh J-P, Kazi S, Fabiano N, Treanor L, Absi M, Hallgrimson Z, Leeflang MMG, Hooft L, van der Pol CB, Prager R, Hare SS, Dennie C, Spijker R, Deeks JJ, Dinnes J, Jenniskens K, Korevaar DA, Cohen JF, Van den Bruel A, Takwoingi Y, van de Wijgert J, Damen JAAG, Wang J, McInnes MDF, Cochrane COVID-19 Diagnostic Test Accuracy Group. Thoracic imaging tests for the diagnosis of COVID-19. Cochrane Database of Systematic Reviews 2021, Issue 3. Art. No.: CD013639. DOI: 10.1002/14651858.CD013639.pub4.

Reference link 2: The role of chest radiography in confirming covid-19 pneumonia

- "No single feature of covid-19 pneumonia on a chest radiograph is specific or diagnostic, but a combination of multifocal peripheral lung changes of ground glass opacity and/or consolidation, which are most commonly bilateral, may be present"
- "Diagnosis might be complicated as covid-19 pneumonia may or may not be visible on chest radiograph"
- "Most patients with covid-19 infection have a mild illness and do not develop pneumonia"
- Diagnose
    1. **"Like other pneumonias, covid-19 pneumonia causes the density of the lungs to increase**. This may be seen as **whiteness in the lungs** on radiography which, depending on the severity of the pneumonia, **obscures the lung markings** that are normally seen; however, this may be delayed in appearing or absent."
    2. "Review the radiograph systematically, looking for abnormalities of the **heart, mediastinum, lungs, diaphragm, and ribs**,9 and remembering that radiographic changes of covid-19 pneumonia can be **subtle or absent**."
    3. "covid-19 pneumonia changes are mostly **bilateral on chest radiographs** (72.9%, 95% confidence interval 58.6 to 87.1) and have **ground glass opacity** in 68.5% of cases (95% CI 51.8 to 85.2)"
    4. "common manifestations and patterns of **lung abnormality** seen on portable chest radiography in covid-19 patients"
- How it can be used in the model?

1. The paper provides several images and analyzes them. From the images, it seems that the target areas are in the 4 corners of the lungs, perhaps data augmentation can perform crops to those areas
2. In addition, covid-19 is "bilateral" meaning that it affects both sides of the lungs at a high probability. Perhaps mirroring images could be a possible approach to increase the size of the dataset.

Citation:

```
@article {Cleverleym2426,
    author = {Cleverley, Joanne and Piper, James and Jones, Melvyn M},
    title = {The role of chest radiography in confirming covid-19
pneumonia},
    volume = {370},
    elocation-id = {m2426},
    year = {2020},
    doi = {10.1136/bmj.m2426},
    publisher = {BMJ Publishing Group Ltd},
    URL = {https://www.bmj.com/content/370/bmj.m2426},
    eprint = {https://www.bmj.com/content/370/bmj.m2426.full.pdf},
    journal = {BMJ}
}
```

11. References

Leszczynski, M. (2010). Image Preprocessing for Illumination Invariant Face Verification. Journal of telecommunications and information technology, 19-25.

12. Common bug fixes

Bug fix 1

Issue

Erroneous Code

```
train_loss  = criterion(output, y_train_batch)
```

Error: `RuntimeError: 1D target tensor expected, multi-target not supported`

Background information

`criterion` is defined as `criterion = nn.CrossEntropyLoss()`

Tensors were initialized as `[1 0 0]` in the dataset class' `__getitem`. This was then passed to the data loader in `train_loader = DataLoader(ld_train, batch_size = bs_val, shuffle = True)` for the respective train, validation, test.

Explanation

The following link at Pytorch discussion - RuntimeError: multi-target not supported explains it

This issue is caused by because `CrossEntropyLoss` does not expect a one-hot encoded vector as the target, but class indices: The input is expected to contain scores for each class. Input has to be a 2D Tensor of size (minibatch, C). This criterion expects a class index (0 to C-1) as the target for each value of a 1D tensor of size minibatch

Example inputs

```
loss = nn.CrossEntropyLoss()
# Input: (N, C). C is the number of classes
input = torch.randn(3, 5, requires_grad=True)
'''
tensor([[-0.9296, -0.6807,  0.1294,  0.0215,  1.1662],
        [-2.0247, -1.5964,  1.4188, -1.4765,  0.7276],
        [-0.3432, -2.3248,  0.2816, -0.4983, -0.7904]], requires_grad=True)
'''
target = torch.empty(3, dtype=torch.long).random_(5)
'''
tensor([2, 4, 1])
'''
output = loss(input, target)
'''
tensor(2.1256, grad_fn=<NllLossBackward>)
'''
```

Fix

Possible fix 1:

This fix was shown in the link. Change the loss to the following:

```
loss = criterion(outputs, torch.max(y_train_batch, 1)[1])
```

What `torch.max` does. Refer to PyTorch docs TORCH.MAX

```
for k, v in train_loader:
    print(v)
    '''
    tensor([[0., 1., 0.],
        [1., 0., 0.],
        [0., 1., 0.],
        [0., 1., 0.]])
    '''
    print(torch.max(v))
    '''
    tensor(1.)
    '''
    print(torch.max(v,1))
    '''
    values=tensor([1., 1., 1., 1.]),
    indices=tensor([1, 0, 1, 1]))
    '''
    print(torch.max(v,1)[1])
    '''
    tensor([1, 0, 1, 1])
    # this is the index of the max value per sample in the batch
    # example: first sample, the max is at index 1. Hence the first element
in the tensor
    indicates that sample 1's max is at index 1
    '''
    # Forced stop
    break
```