# 50.039 Deep Learning Small Project Report

Jeremy Ng 1003565
Zhou Yutong 1003704

## Introduction

We are tasked to design a custom deep learning model where the model will assist with the diagnosis of pneumonia, for COVID and non-COVID cases, by using X-ray images of patients.

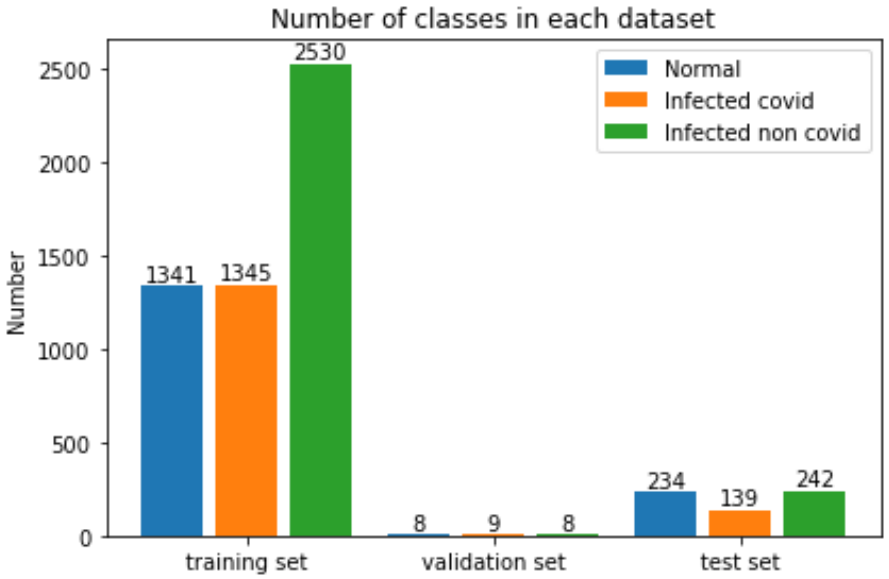
## Instructions to run code:

- Downloaded zip file and unzip
- Move the folder containing the x-ray image datasets under the "50.039-Small_Project" directory. Rename the dataset folder to "small_proj_dataset".
- Open notebook and cd to the directory of the unzip folder
- customCnn_1.py & customCnn_2.py: custom CNN models implemented for the small project.
  customDataset_1.py & customDataset_2.py: custom datasets implemented for the small project.
  utils.py: common useful functions which are shared between both models.
  Graph folder: stores accuracy and loss graphs of the models
  Models folder: stores the models
  Checkpointing.ipynb: Demonstration of checkpoint saving and reloading of model to resume training.
  Final_confusion_matrix.png: confusion matrix of output after undergoing two models

- To run the full code on training the models and displaying final results, open Small_project_sequential.ipynb and run all cells.
  ***Important Note:** The weights of the models we trained were saved under their respective folders in the "models" directory. Re-running the notebook will overwrite the saved weights.

- To run the demonstration of how our code implements checkpoint saving upon premature termination during training, reloading the checkpoint model to resume training, open checkpointing.ipynb and run all cells. When the model is being trained half way, interrupt the process. Run the cells below to load and continue training the model.

## Visual Representation of Dataset

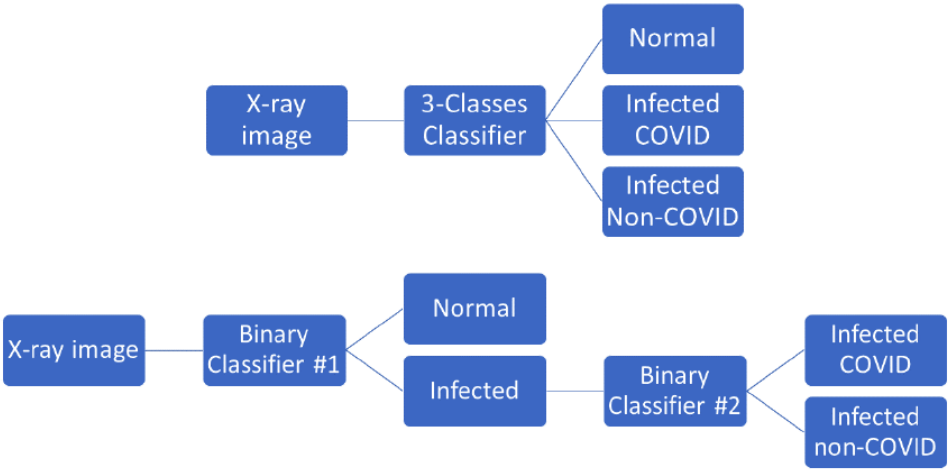



Based on the distribution of data we can see that the distribution is not equal in the training set and test set given that there are significantly more infected non covid lung images and significantly lesser infected covid lung images as compared to the rest. This might affect the final results of our model.

## Data pre-processing operations

Images which were fed into the custom datasets were normalized by scaling their image pixels from [0-255] to a range between [0-1]. This is done by simply dividing all the pixels in the images by 255 to get a normalized image. This allows the pixel numbers to be much smaller and hence computation becomes much easier and faster during training. Passing high values into a deep neural network might lead to increased complexity during computation [1]. Since the images in the dataset which are given to us are fixed at 150x150 pixels there is no need to do further image resizing before inputting them into the model for training. However since it's a neural network, one pre-processing operation that is required will be to segregate the images up randomly into batches and stored into Dataloaders before feeding it into the model. Further pre-processing operations might be required if data augmentation is applied to the images before storing them into the Dataloaders.

## Choice of Model Classifier




Two architectures were introduced in the instruction where we have to choose from. The first architecture is a multi class classification model which involves a single 3-class classifier. The X-ray images will be fed into the 3-class classifier which will then classify them into normal healthy lungs, infected covid lungs and infected non-covid lungs. While the second architecture involves two binary classifiers. The X-ray images will be fed into the binary classifier #1 which will then classify them into normal healthy lung images and infected lung images. If the lung is classified as infected, it will then be further fed into binary classifier #2 where it will further classify them into infected covid lung images and infected non-covid lung images.

We decided to go for the architecture which involves the two stacked binary model classifiers which we named Model_1 and Model_2 respectively. We decided to go with the second architecture because upon visual inspection, there is a significant difference between the images of a normal lung and that of an infected lung. However, the visual difference is a lot more smaller within the infected lungs when compared between the images of infected covid lungs and infected non-covid lungs. Hence, the set of features differentiating between normal and infected lung images will be very different from those differentiating between infected covid and infected non-covid lung images. Hence, it would be justifiable to train two different models for the different tasks.

## Data Augmentation

A visual inspection was done on the dataset and it was found that there were in fact very small differences between the images of lungs with covid and the images of lungs without covid. Furthermore, the size of the dataset which was provided to us were substantially small and hence we felt that there were too little inputs to carry out the training of our custom model from scratch. Hence, this motivated us to perform data augmentation in hopes of having a better performance on our model.

Data Augmentation is a technique that can be used to artificially increase the size of the training dataset by constructing modified versions of the training images. This can result in the ability to generate more skillful models as the augmentation techniques create variations of the image that can improve the models ability to generalise and work on other images

better. This technique is applied when the size of training data is too small and unable to train accurately and having this technique applied properly in balance will also reduce the problem of overfitting.

Since the lung images of both covid and non-covid patients were very similar to each other. We came up with a plan to implement data augmentation to increase noise within the images by altering only the brightness, contrast, saturation and hue. We did not implement any vertical or horizontal flips as we felt that it would not make sense given that x-ray images are produced upright with directional indicators indicating left and right side of the image. Below shows a snippet of code which initialises the transformation object which is then fed into the Train Dataloader after.

```
# applied brightness, contrast, saturation, hue to augment training data images
my_transforms = torchvision.transforms.RandomApply(([transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1)]), p=0.2)
```

We validated our plan by comparing our models performances multiple times with and without data augmentation where the architectures of both models remained constant throughout. Model_1 is the model classifier which classifies normal and infected lung images while Model_2 is the model classifier which classifies infected covid and infected non-covid lung images. Our performance metric used for comparison was the lowest validation loss observed during training and we took its mean value from five training sessions of the models and recorded them in a table below.

| Lowest validation loss | Model_1 | Model_1 with augmentation | Model_2 | Model_2 with augmentation |
|---|---|---|---|---|
| Train 1 | 0.349 | 0.494 | 0.361 | 0.362 |
| Train 2 | 0.427 | 0.414 | 0.376 | 0.352 |
| Train 3 | 0.424 | 0.429 | 0.352 | 0.367 |
| Train 4 | 0.469 | 0.617 | 0.368 | 0.366 |
| Train 5 | 0.389 | 0.424 | 0.367 | 0.336 |
| Avg | 0.412 | 0.476 | 0.365 | 0.357 |

We can see that Model_1 has a lower average validation loss when trained without data augmentation. This means that it performs better without any data augmentation when classifying between normal and infected lung images. However Model_2 has a lower average validation loss when trained with data augmentation. It performs better with data augmentation when classifying infected covid and infected non-covid lung images. This results agrees with our earlier plan of implementing data augmentation of the covid and non covid lung images for Model_2 since the images were very similar to each other, applying augmentation will increase the variation of the images.
Based on our initial plan backed up by the comparisons we made, we decided to go without data augmentation for Model_1 and with data augmentation applied to the Dataloders for Model_2.

# Model Architecture and Parameters

## Model_1 (normal/infected):

```
class NN_Classifier_1(nn.Module):
    def __init__(self, output_size, drop_p=0.75):
        """
        Initialising parameters used in the custom CNN model to classify healthy and infected lungs
        """
        super(NN_Classifier_1,self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1, 1)
        #input channels, output channels, kernel size, stride, padding
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(3)
        # shape: 64 32, 75 75

        self.conv2 = nn.Conv2d(32, 64, 3, 1, 1)
        self.relu2 = nn.ReLU()

        self.fc1 = nn.Linear(64*50*50, 256)
        self.dropout = nn.Dropout(drop_p)
        self.fc2 = nn.Linear(256, output_size)

    def forward(self, x):
        """
        Building the custom model with the initialised parameters
        """

        x = self.conv1(x)
        x = self.relu1(x)
        x = self.pool1(x)
        x = self.dropout(x)

        x= self.conv2(x)
        x = self.relu2(x)

        # to flatten into 1D
        x = x.view(x.size(0), -1)

        x = self.fc1(x)
        x = self.dropout(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

## Model_2 (covid/non-covid):

```
class NN_Classifier_2(nn.Module):
    def __init__(self):
        """
        Initialising parameters used in the custom CNN model to classify covid and non_covid lungs
        """
        super(NN_Classifier_2, self).__init__()
        self.conv1 = nn.Conv2d(1, 96, 7, stride=4, padding=1)
        # input channels, output channels, kernel size, stride, padding
        self.conv2 = nn.Conv2d(96, 128, 5, stride=4, padding=1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        # 64 256 2 2
        self.fc1 = nn.Linear(512, 64)
        self.fc2 = nn.Linear(64, 2)

    def forward(self, x):
        """
        Building the custom model with the initialised parameters
        """
        x = self.conv1(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)


        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

The architecture (NN_Classifier_1) is used to differentiate normal and infected x-ray images. This architecture is mainly constructed via trial and error with the main consideration to make the model simple as it is relatively simple to differentiate them and hence we do not want to overfit the model. The model consists of two convolutional layers, a single dropout layer and lastly two fully connected layers to classify the images. To further simplify the model, we went to include a relatively high dropout probability in between the two fully connected layers.

The architecture (NN_Classifier_2) is used to differentiate covid and non-covid x-ray images and it is inspired by a research we came across. The dataset used in the article is also x-ray images similar to our case. Our model is adopted and modified (slightly) from the proposed model discussed in the article which consists of 2 convolution and 1 fully connected layer. Apart from the current architecture, we had also explored architectures which are modifications of well known architectures such as Lenet and VGG16 along with other architectures discussed in the research we found. The architecture we are currently adopting produces the best result among all the models we tried. [2]

We explored various initialization methods. One general guideline discussed by Prof. Liu Jun was to use the Kaiming He's method. We went ahead to test on the initialization methods namely kaiming_uniform and kaiming_normal. Although those were the well known initialization methods, we found that they did not perform as well for our case. We realised the validation accuracy after 10 epochs of training was slightly lower with Kaiming's initialization method as compared to the default initialization with Pytorch. Hence, we decided to just use the default initialization of the model parameters.

## Use of Mini Batches

We trained both our models in batches of size 32. The benefit of training using mini-batches is that it generally tends to converge faster as the computation is based on the randomly selected mini-batch and not the entire dataset. This also reduces computational costs and increases training speed since computing on the full dataset is often too costly due to its large size. We are also not required to load all the data into the computer memory at one goal which takes up alot of memory resources. [3]

## Choice of Loss Function

We have chosen to use CrossEntropyLoss as our loss function in both our models. Since both Model_1 and Model_2 contain only two classes during classification this can be considered as a binary classification. However we label encoded our classes (0:normal,1:infected for Model_1; 0:covid,1:non_covid for Model_2) where our classes are encoded into integer labels 0 or 1. Hence the appropriate choice for integer labels would be to use pytorch's nn.CrossEntropyLoss.

## Choice of Optimizer

We have chosen Adam as the optimizer. Adam is an adaptive learning rate optimization algorithm that is a combination of RMSprop with Momentum Term , but with improvements over RMSprop. The reason behind the choice is that Adam is one of the state of art optimizers used currently. Hence, it is definitely worth a try. We tried to use different other optimizers for our model such as Adagrad, RMSprop and SGD. The final validation loss and accuracy produced with different optimization algorithms are very close. We tried to train the model with each algorithm 5 times and we found that Adam optimizer produced a better result in general. The initialization of learning rate for Adam is set to be 0.001. This is because if we set the learning rate to be too large, it may lead to divergence.On the other hand, if we set the learning rate to be too small, it may lead to slow convergence. We tried different values of learning rate namely 0.1, 0.01, 0.001 and 0.0001. We found that the training and validation loss (both about 0.65), as well as the training and validation accuracies (both about 0.65) are not improving after the very first update. Using a learning rate of 0.001 and 0.0001 would lead to a similar result but it took more epochs to train for a learning rate of 0.0001 to reach the similar result as the learning rate of 0.001. This leads to our final decision of learning rate of 0.001.

## Additional Discussions

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

Yes, we generally found it much more difficult to differentiate between non-covid and covid lung images as compared to normal lung and infected lung images. This is because both pneumonia and covid19 are lung diseases that will cause the lung to look very differently from a normal healthy lung, hence it will be easier to detect if a lung is normal in this case against an infected one. However, within the images of the infected lungs, there are very subtle differences between images of the covid infected lung and the infected non-covid lung. This can be validated further by inspecting the images in the dataset where its quite obvious to see the difference between a normal and infected lung while its much more harder to differentiate the lung images between the infected covid and infected non-covid class.

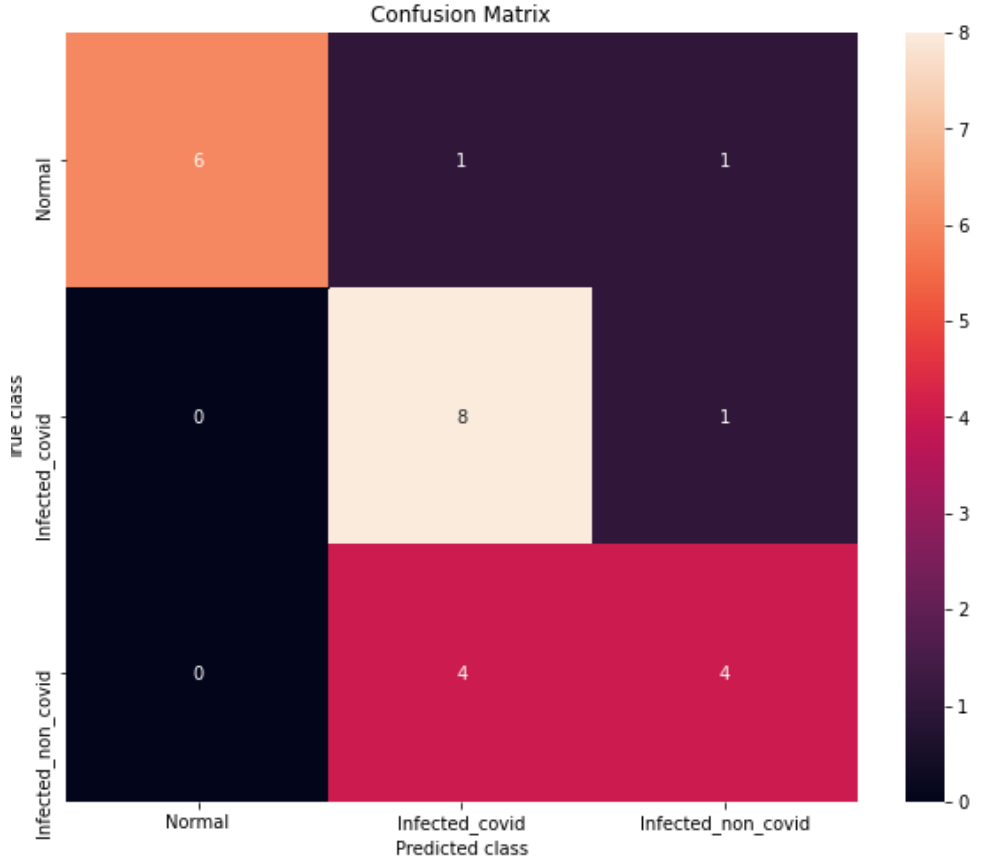
**Final question: would it be better to have a model with high overall accuracy or low true negatives/false positives rates on certain classes? Discuss.**

Accuracy provides a probability of the number of correct predictions over the total predictions made on the test Dataset. It is considered a general metric to compare how the model performs however it does not always give the full picture. It would not be an accurate metric when our dataset is not evenly distributed as shown in our case of the small project where

the training and validation datasets are generally unevenly distributed. This will skew our training and validation results and hence will not be an accurate representation on how the model is actually performing. We can see in the Small_project_sequential.ipynb that the training, validation accuracies during training differs from the test accuracies during testing. Instead a much better and accurate approach to evaluate model performance will be to gather the low true negatives/false positives rates on certain classes which can be taken from the confusion matrix. For example if we were to take infected non-covid to be the positive class and infected covid to be the negative class, we would most definitely want an extremely low or even zero false positive rate because we want to minimize errors of a patient who actually has covid but is diagnosed as non-covid. Wrongly classifying such cases will have severe consequences.

In the context of Singapore, there are currently much fewer people who are suffering from covid than people who are healthy or suffering from an ordinary lung infection. Hence if we use the accuracy metrics to determine the model performance, it might not be the best representation. The model can wrongly classify covid cases as non-covid and the model accuracy will still be deemed high as long as there is a high number of non-covid cases predicted correctly. Therefore due to the imbalanced classes, accuracy might not be the best metric to evaluate performance.

Below is the confusion matrix which show the performance of both models (Model_1 and Model_2) combined :



## References:

[1]https://medium.com/analytics-vidhya/a-tip-a-day-python-tip-8-why-should-we-normalize-image-pixel-values-or-divide-by-255-4608ac5cd26a

[2]https://journals.physiology.org/doi/full/10.1152/physiolgenomics.00084.2020

[3]https://medium.com/analytics-vidhya/when-and-why-are-batches-used-in-machine-learning-acda4eb00763#:~:text=Another%20reason%20for%20why%20you,decrease%20in%20speed%20of%20training.