Deep Learning Small Project Report

Cornelius Yap 1003358
Son Soo Han 1003525
Leong EnYi 1003463

Data

Dataset and Dataloader Classes

Taking inspiration from the Dataset tutorial uploaded along with the small project, we created our custom Dataset class in a similar way. However, we made some changes to the code provided in the Dataset tutorial such that we do not need to create a new custom class when we want to change the purpose of the dataset (whether it is used for training, testing or validation) or when we want to change the classes of the data retrieved (whether we want to retrieve normal vs infected or normal vs covid vs non_covid and etc.). Our custom Dataset class is built such that it will be able to handle those variances.

To do so, we added two arguments to the initialization of our Lung_Dataset() class. They are:

1) Purpose: Compulsory argument. Requires one of the following strings “train”, “test” or “val” to define if the dataset should retrieve the images from the train, test or val folders
2) Verbose: Optional argument that defaults to 0. It accepts integer values of either 0,1 or 2, to define the classes and labels that the dataset will return. If verbose is 0, the dataset will retrieve the normal and infected classes (the covid and non-covid images will be grouped under infected). If verbose is 1, the dataset will retrieve the 3 different classes, with the normal, covid and non-covid images all having their own labels. If verbose is 2, the dataset will only retrieve the 2 classes, covid and non-covid, from the infected category.

Data Distribution with Graphs

The dataset we are given consists of a train, test and val folder. Within which, the normal and infected images are separated. The infected images are also further splitted up into covid and non-covid images. The file structure is as shown:

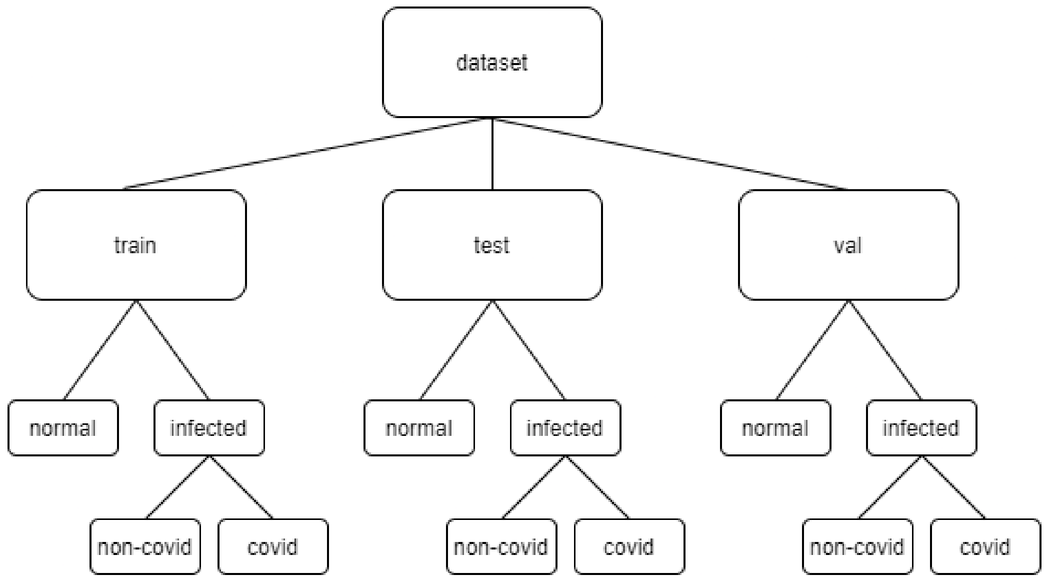
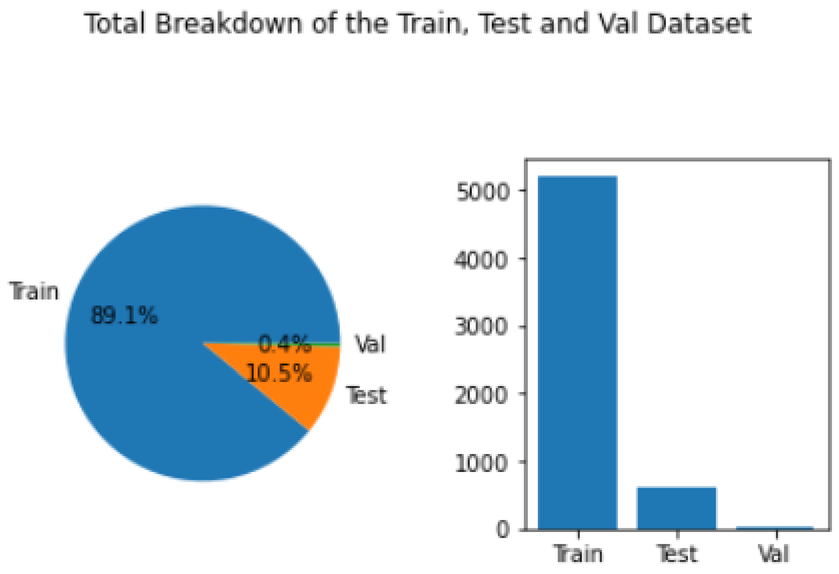


Figure 1: File Structure of the Dataset

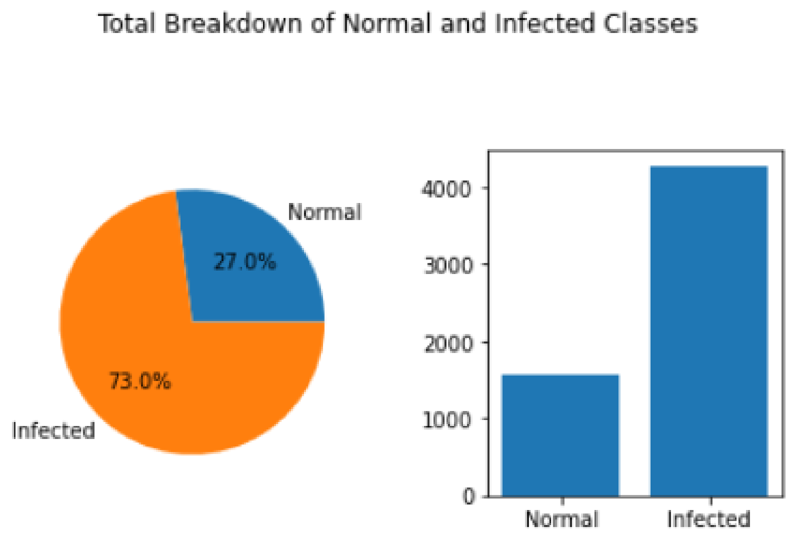
The entire Dataset consists of 5856 images, of size 150 by 150. Out of which, we have 5216 training images, 615 test images and 25 validation images:



```
Train: 5216 images
Test: 615 images
Val: 25 images
```

Figure 2: Breakdown of the Train, Test and Validation Dataset

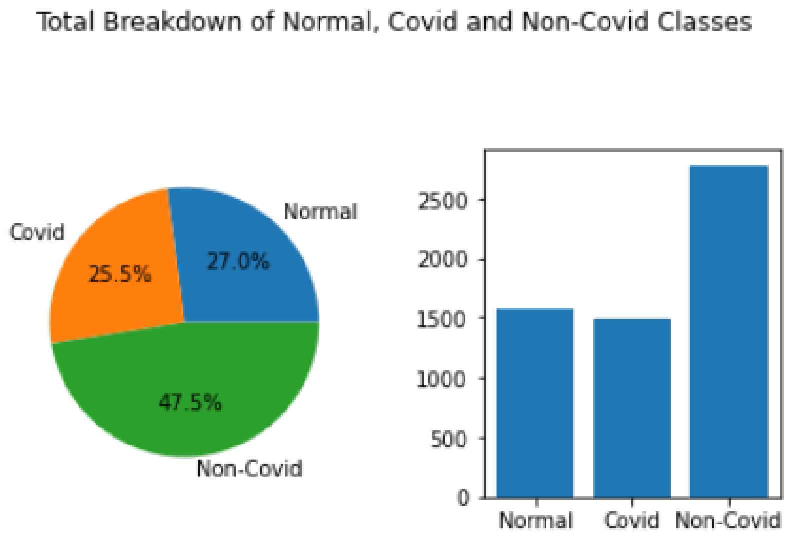
Among the entire training set, we have 1583 normal images and 4273 infected images:



```
Normal: 1583 images
Infected: 4273 images
```

Figure 3: Breakdown of the Normal and Infected Classes

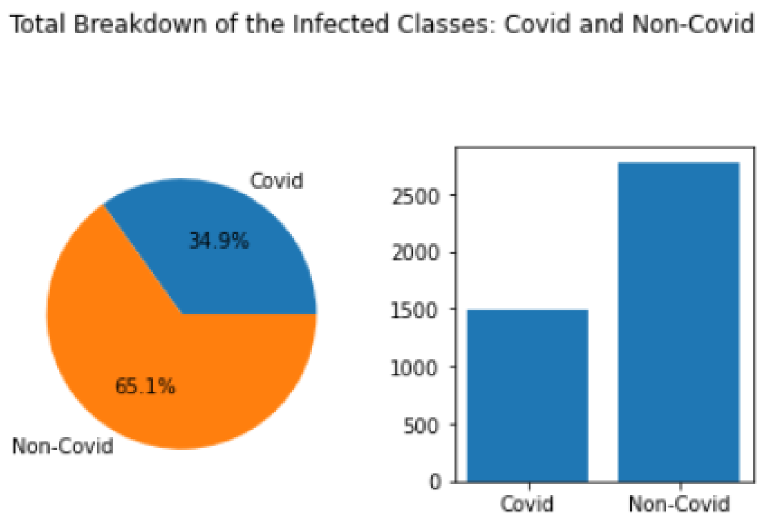
If we further classify the infected into covid and non-covid images, we have 2780 non-covid images and 1493 covid images:



```
Normal: 1583 images
Covid: 1493 images
Non-Covid: 2780 images
```

Figure 4: Breakdown of the Normal, Covid and Non-Covid Classes

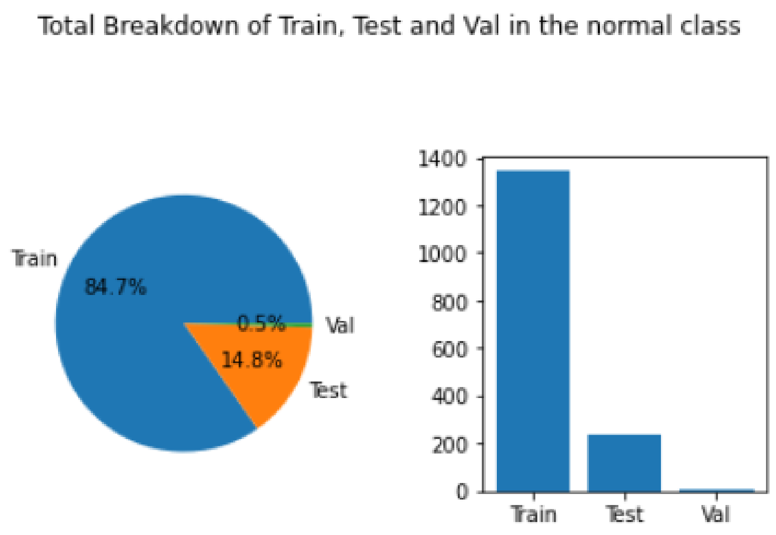
Among the infected class, the proportion of the number of covid and non-covid images is as shown:



```
Covid: 1493 images
Non-Covid: 2780 images
```

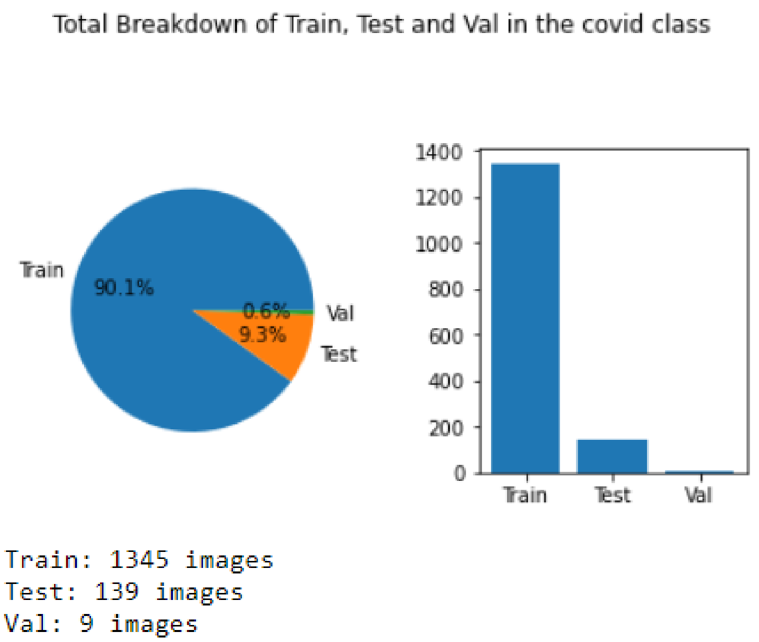
Figure 5: Breakdown of the Covid and Non-Covid Classes

As for the train, test and validation size proportion within the different classes (normal, covid and non-covid) the distribution is as shown:



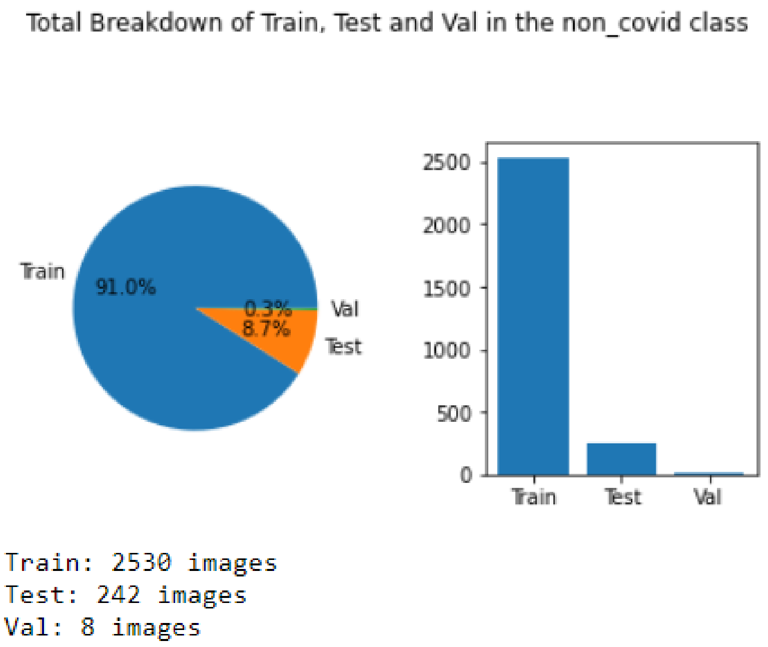
```
Train: 1341 images
Test: 234 images
Val: 8 images
```

Figure 6: Breakdown of the Train, Test and Validate in the Normal Class



```
Train: 1345 images
Test: 139 images
Val: 9 images
```

Figure 7: Breakdown of the Train, Test and Validate in the Covid Class



```
Train: 2530 images
Test: 242 images
Val: 8 images
```

Figure 8: Breakdown of the Train, Test and Validate in the Non-Covid Class

From the image distributions shown above, we can see that we have a slight imbalance among the classes. 73% of our entire dataset is made up of infected images and only 27% of images are classified as normal. If we further split the infected images into covid and non-covid, we have 47.5% of non-covid images, 25.5% of covid images and 27.0% of normal images.

A problem with the class imbalance is that the model might get exposed to more images that are classified as infected as compared to normal. Thus, the weights might get updated more towards the infected side, causing the model to be slightly biased towards classifying an image into the infected category. However, this might work to our favour. Since we are building a model for a medical use-case, we want it to be on the safe-side and have less false negatives. Thus having a slight bias towards infected images might actually be beneficial in the long run as we rather a patient get mis-classified as infected rather than being mis-classified as normal and potentially spreading the virus around.

However, this does not work to our favour for our model that was trained to classify between the covid and non-covid classes. We have 65.1% of our images that are classified as non-covid and only 34.9% of the images classified as covid. This might introduce a bias towards the non-covid class which is not ideal. Since covid is a very infectious disease, we want to reduce the number of false negative cases as much as possible. This is to prevent someone who has covid from roaming around freely in public because the model wrongly classified him as not suffering from covid.

Important

To clarify, we decided to use the images in the test folder for validation purposes, and images in the validation folder for testing purposes. As the test set has significantly more images (615) as compared to the validation set (25), we felt that it was better that we validate the model on the larger test set to ensure the generalisability of the model. The smaller validation set will then be used as the final test to measure the actual performance of the model at the end of the training.

Data processing operations

Our data processing operations involve scaling the pixel values of the images from 0-255 to 0-1. This normalizes the images to be on the same scale, ensuring that updates made during gradient descent are not dominated by the larger pixel values (255) as compared to the smaller values (0). When the gradient descent is dominated by large values, the updated weight values will oscillate back and forth, causing it to take more steps to reach the optimal point.

By normalising all values to 0-1, we will have a more balanced gradient descent, allowing for faster convergence as well. [1]

Bonus

For the bonus, we added the following augmentations:

1. Center Crop
2. Translation of the image along its width
3. Scaling
4. Horizontal Flip
5. Standardization

For the augmentation of the medical images, we have to be careful in choosing what augmentations we should add. This is because medical images are usually very standardized and we have to ensure that the augmented images do not change the images too much such that it is not plausible to be seen on the test set. We go into the reasons for each augmentation in the section below:

Center Crop

After looking through the dataset, we realised that some of the images contain features that are not very helpful to the classification of normal, covid and non-covid images. We did some research on how clinicians diagnose the chest x-ray and we found out that they only focus on the lungs. Thus we decided to perform center-cropping to remove any irrelevant features from the image

Translation of Image

In addition, we found that not all the dataset have the chest centered in the image, but is slightly to the side. Thus we decided to perform some translation such that the model does not overfit on images that has the chest right at the center of the image

Scaling

We perform scaling for the same reason as the center cropping. We want to remove any irrelevant noise from the sides of the images and just get it to focus on the lungs in the middle.

Horizontal Flip

The reason for the horizontal flip is to prevent the model from overfitting to any specific parts of the lungs. Through our research into how clinicians diagnose the chest x-ray, we learn that they based it on the cloudiness of the lungs in general. As such, flipping the lungs horizontally should not make a difference in the classification of whether the individual is normal, has covid or no covid.

Standardization

We perform standardization to help the model converge faster.

Benefits

The benefits of the augmentation will be discussed in the Bonus section of the Results further down in the report.

Model Architecture

Model Layers

There are two proposed model architectures, one being a flat multiclass classifier and the other being a stack of hierarchical binary classifiers. For the first architecture, a single model will be trained with the final layer consisting of three output neurons, each corresponding to one of the three classes of images: Normal, COVID, and non-COVID. The predicted class label will be retrieved by calculating the argmax of the output. For the second architecture, two separate binary classifiers will be trained, the first on the classes Normal and Infected, and the second on the classes COVID and non-COVID. Only the samples labelled as Infected by the first classifier will be passed on to the second classifier for further classification. Our group has chosen to implement the second model architecture.

The provided dataset of x-rays have a natural hierarchy of classes, as visualized in Figure 9. Looking at sample images from the three classes in Figure 10, it can be seen that the features that distinguish the infected images from the normal images, such as cloudiness, are different from those that distinguish between the two types of infections, which are harder to immediately perceive. Breaking the classification into smaller chained sub-tasks can help to preserve the natural hierarchy of the data, allowing each classifier to focus on more specific identifying features for each sub-task. One concern with chaining classifiers is error propagation, where mistakes made by the first classifier will propagate to the second classifier. However, in our case the model only has two levels of classification, and hence this effect is minimized.

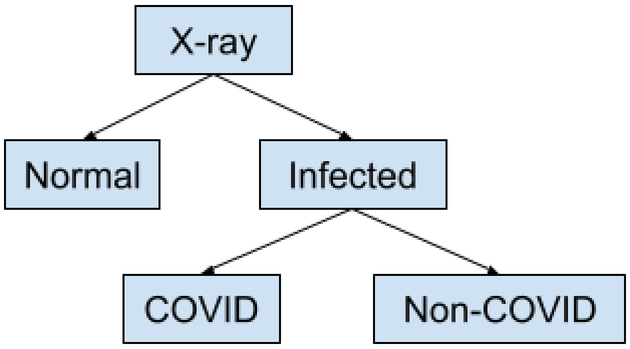


Figure 9: Hierarchy of provided dataset

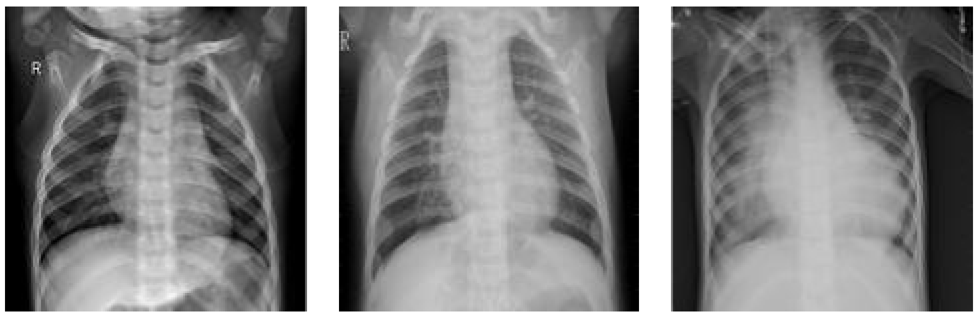


Figure 10: Normal, COVID, and non-COVID (respectively) image samples

Each of our two binary classifiers will be a Convolutional Neural Network (CNN) with the following structure:

Layer	**Input Size**	**Output Size**
3x3 Convolution 2x2 Max Pooling ReLU Batch Normalization Dropout	150 x 150	75 x 75 x 64
3x3 Convolution 2x2 Max Pooling ReLU Batch Normalization Dropout	75 x 75 x 64	37 x 37 x 128
3x3 Convolution 2x2 Max Pooling ReLU Batch Normalization Dropout	37 x 37 x 128	18 x 18 x 256
3x3 Convolution 2x2 Max Pooling ReLU Batch Normalization Dropout	18 x 18 x 256	9 x 9 x 512
Global Average Pooling	9 x 9 x 512	512
Fully Connected Sigmoid	512	1

Table 1: Model architecture for each binary classifier

CNNs are the current state of the art for image classification tasks [2]. Their use of convolutional kernels allows for the extraction of spatial features by capturing the relationship between neighboring input pixels. CNNs also employ weight sharing, where each kernel has a fixed set of weights that it uses over the entire image. This helps to reduce the number of learnable model parameters as compared to a fully connected model, which is ideal as our dataset only comprises a few thousand images. 3x3 convolution with 1 pixel padding and a stride of 1 followed by 2x2 max pooling with a stride of 2 is a pattern commonly seen in state-of-the-art models such as VGG [3] and ResNet [4]. The small kernel size helps to keep the number of model parameters small, and can be stacked to mimic larger convolutions. Padding is used to preserve information at the edges of the image. Pooling is used to summarize extracted features in different areas of the image, helping to reduce the sensitivity of the model to differences in location of features.

The chaining of convolutional layers while increasing the number of kernels allows for a few basic features to be extracted at the lower layers, which can then be combined into a larger number of complex features at higher layers.

The Rectified Linear Unit (ReLU) activation function was used for each hidden layer in order to mitigate the vanishing gradient problem. The ReLU is a popular activation function used in many state-of-the-art CNN architectures such as VGG [3] and ResNet [4]. Other versions of the ReLU were not chosen as they would add parameters to tune without proving to provide significant performance increases [5].

A combination of Batch Normalization and Dropout layers were used after each hidden layer's activation to improve regularization, training stability, and convergence rate [6].

As a bridge between the final convolutional layer and the fully connected layer, Global Average Pooling was chosen instead of simply flattening the tensor in order to reduce the number of parameters of the fully connected layer.

Loss Function

Since each of our classifiers will only be predicting between two classes, we will be using the Binary Cross-entropy loss function. This loss penalizes based on the confidence of the prediction, which in this case is the predicted probability. It penalizes both wrong but confident predictions as well as correct but unconfident predictions, leading to a more robust model.

Optimizer

We used the Adam optimizer commonly found in most state-of-the-art computer vision models today [3, 4]. The Adam optimizer combines the best of RMSProp and momentum based stochastic gradient descent (SGD) optimizers. It adjusts weights based on both an exponentially moving average of past gradients and past square gradients, and has shown to outperform other optimizers in convergence stability and speed [7].

Initialization

We used the Kaiming initialization method [8] which is the current state-of-the-art for models with ReLU activation functions. This initialization method helps to maintain the means and standard deviations of activation layer outputs to 0 and 1 respectively, mitigating the exploding or vanishing gradient problem. It does so by scaling initial weight values from a random normal distribution according to the number of input features to each layer. The randomness helps to avoid symmetry in neural network weights.

Hyperparameters

We chose 4 different hyperparameters to tune, namely the dropout probability, learning rate, batch size, and weight decay. In the interest of time, reasonable defaults were chosen for the other hyperparameters required in the Adam optimizer, which were all taken from the recommendations by the original paper [7] and are listed as follows:

Beta1 (first moment decay): 0.9
Beta2 (second moment decay): 0.999
Epsilon (for numerical stability): 10^{-8}

We used the optimization framework Optuna [9] to tune our hyperparameters. Instead of a random search over the parameter space, Optuna leverages a bayesian optimization technique which guides the search according to previously found results [10]. The search space for each hyperparameter was defined as follows, and were roughly based on values commonly found in literature:

Dropout [6, 11] - linearly from 0 to 0.2
Learning Rate [7, 12] - logarithmically from 0.00001 to 0.1
Weight Decay [13] - linearly from 0 to 0.2
Batch Size [7, 12] - one of 16, 32, or 64

We ran separate searches for the first and second layer models, and ran 30 trials each. For each trial, the model was trained for 200 epochs but with an early stopping condition with a patience of 5. If the validation loss does not decrease for more than 5 epochs, the training is stopped. The original training and validation datasets were used.

Results

Hyperparameter	Best Value
Learning Rate	0.015482051772781362
Batch Size	16
Dropout Probability	0.1258685286441622
Weight Decay	0.018373534139734887

Table 2: First layer classifier best hyperparameters

Hyperparameter	Best Value
Learning Rate	0.00022040717935498526
Batch Size	64
Dropout Probability	0.07216065625585566
Weight Decay	0.08267768486777832

Table 3: Second layer classifier best hyperparameters

Results

Training

We trained our final models for 200 epochs with an early stopping patience of 10 based on the validation loss, obtaining the following loss curves:

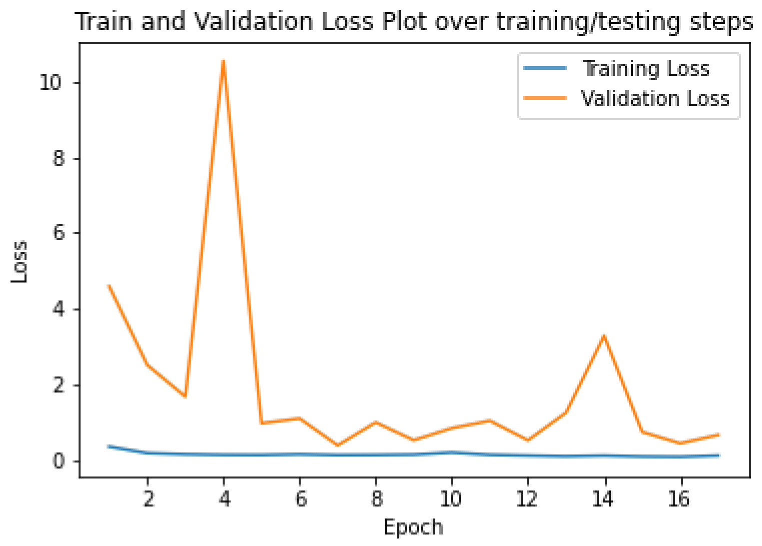


Figure 11: Training and validation loss for first layer classifier

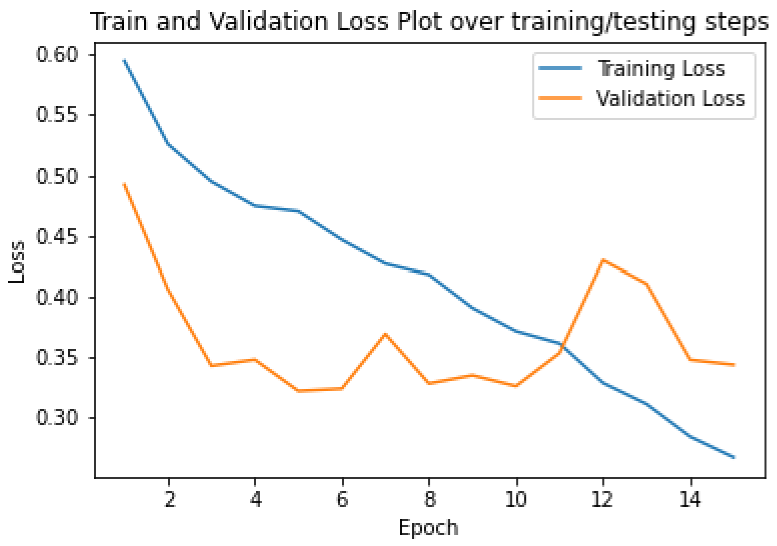


Figure 12: Training and validation loss for second layer classifier

Both models trained relatively quickly and started to overfit after a relatively small number of epochs (7 and 5 for the first and second layers respectively). This could be an indication that more regularisation is needed, or that the model architecture is too deep. For the second layer classifier, the validation loss is lower than the training loss at the start, possibly meaning that the validation set is too easy for the classifier. However as training goes on the training loss continues to drop while the validation loss does not, indicating that there is no increase in generalizable performance.

Testing

Before combining the classifiers, we checked their individual validation and testing accuracies.

Classifier	**Validation Accuracy**	**Testing Accuracy**
Normal vs Infected	0.867	0.880
COVID vs Non-COVID	0.919	0.647

Table 4: Individual classifier results

The significantly lower testing accuracy of the second layer classifier as compared to its validation accuracy indicates that the second layer model might not be generalizing well. It also alludes to the relatively higher difficulty of differentiating between COVID and Non-COVID images as compared to Normal and Infected images. This was expected to a certain extent due to the fact that it is not immediately obvious just by looking at the images what the differences are between a COVID and Non-COVID image, while the cloudiness of the image is usually a tell that the image is Infected as opposed to Normal (as in figure 10).

For the final combined model, apart from accuracy we have chosen two other metrics by which to evaluate our model, namely the precision and recall, as calculated in equations 1 and 2.

$$Precision = \frac{TP}{TP + FP}$$

Equation 1: Calculation of precision

$$Recall = \frac{TP}{TP + FN}$$

Equation 2: Calculation of recall

TP stands for the number of true positive predictions, while FN and FP stand for the number of false negatives and positives respectively. The metrics will be calculated on a per-class basis. A high precision indicates that out of all of the positive samples identified by a model, many of them are true positive. However, it does not comment on how many positive samples were missed out (false negatives). On the other hand, a high recall indicates that the model was able to identify most of the positive samples, but makes no comment on how many negative samples were mistakenly identified (false positives). The two metrics therefore complement each other and prevent trivial classifiers from being selected when used together. If given equal weightage, they can be combined into a single metric which is the f1-score, however we have decided to keep them distinct because we believe that in the context of our classification problem the relative weight of the two metrics will vary based on the class. For example, for the COVID class we believe that the recall is more important than the precision. For medical diagnosis purposes, the potential consequences of false negative predictions are greater than that of false positive predictions. Particularly in this case of identifying potential COVID infections, a single false negative prediction could endanger the community of the infected individual whereas false positives can be mitigated by administering additional tests on identified patients. The opposite is true for the Normal class, where precision is more important in order to minimize the false positive classifications.

Metric	**Validation Dataset Result**	**Test Dataset Result**
Accuracy	0.8243902325630188	0.6399999856948853
Normal class precision	0.8877551020408163	1.0
COVID class precision	0.68125	0.5384615384615384
Non-COVID class precision	0.8648648648648649	0.5714285714285714
Normal class recall	0.7435897435897436	0.625
COVID class recall	0.7841726618705036	0.7777777777777778
Non-COVID class recall	0.9256198347107438	0.5

Table 5: Combined classifier results

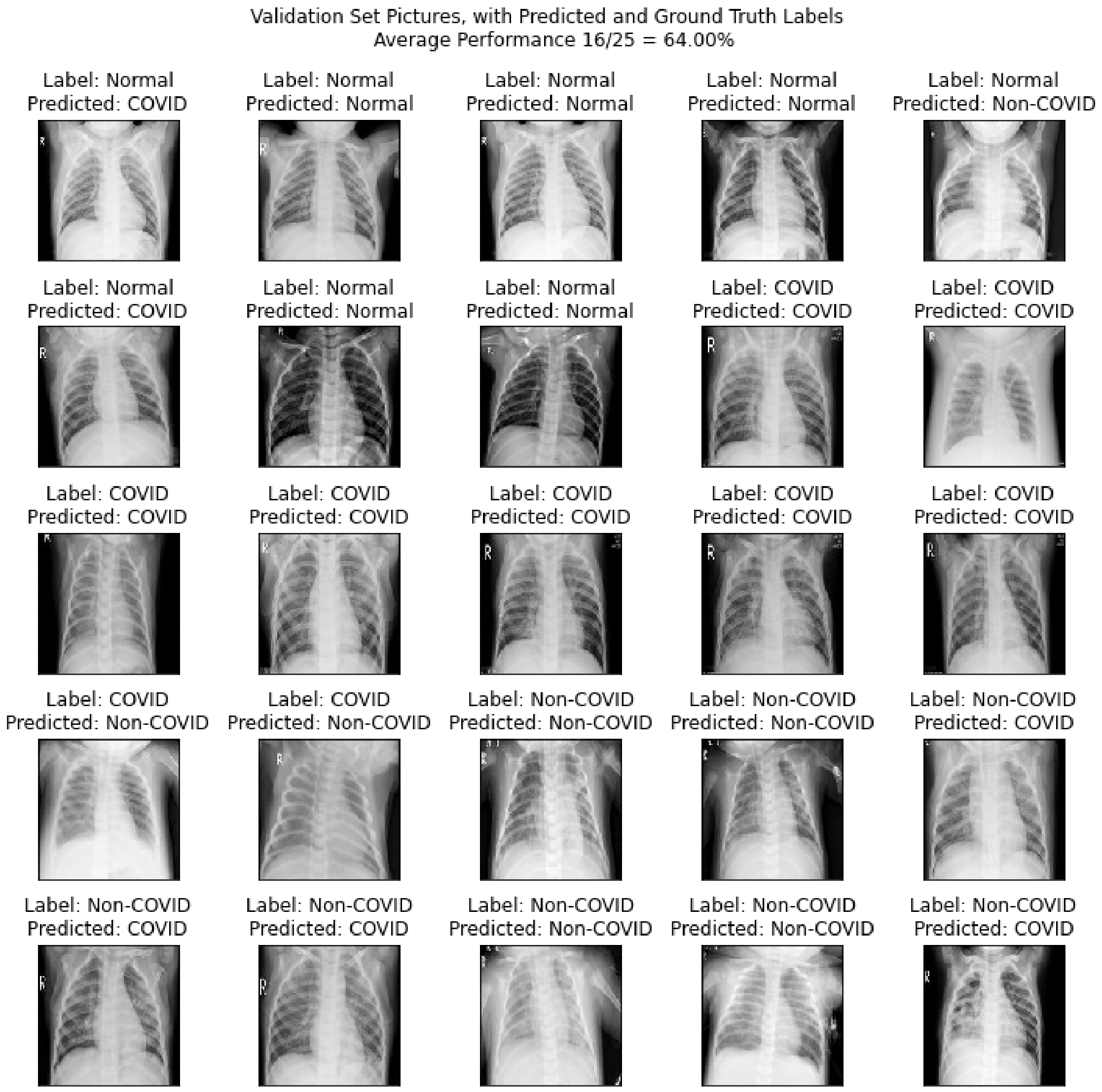


Figure 13: Test dataset images, labels, and predicted labels

As alluded to during the testing of the individual classifiers, overall the performance of the model on the testing dataset is worse than on the validation dataset. However there are a few notable exceptions. The model achieved a perfect score on the precision of the Normal class on the test dataset. This means that whenever it classified an image as Normal, it was correct, and no Infected image was classified as Normal. As explained above, this result is desirable in this particular application. Another desirable result is that the COVID class recall was also only slightly worse on the test set as compared to the validation set, and was the highest compared to the other class recalls. This indicates that most of the COVID images were correctly identified. While the low overall performance on the test dataset still leaves room for improvement before it is ready to be deployed in a medical setting, and the size of the test set is too small to be conclusive, the results show that a deep learning approach could be a viable solution for making COVID diagnoses from chest x-ray images that can help alleviate the burden of doing it manually for medical professionals.

Bonus

1. Implementing and discussing a learning rate scheduler and discuss its appropriate choice of parameters and benefits for your model

Our group used the AdamW variant of Adam, proposed from the paper 'Decoupled Weight Decay Regularization' [14] for the purposes of regularization (bonus question 2, explained after this section). Within the paper, the author made the following observation that Adam optimizer can 'substantially benefit' from using learning rate schedulers. The paper cited other papers that used cosine annealing alongside the decoupled weight day, which led to a more separable hyperparameter search space. Cosine annealing also showed clear outperformance than other learning rate schedulers, which led our group to try implementing a cosine annealing learning rate scheduler as well (shown in the Jupyter Notebook : Model Training and Testing - Learning Rate Scheduler Implementation). Furthermore, the paper showed that warm restart for cosine annealing has been observed to not have any observable impacts to the performance of the model. Hence we did not implement the warm restart.

We used the CosineAnnealingLR available through Pytorch lr_scheduler class. There were some improvements shown in the learning curves

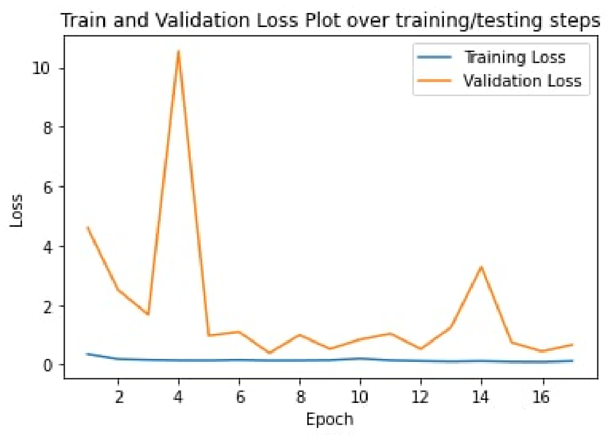


Figure 14:
Without LR Scheduler (Test Accuracy 88%)

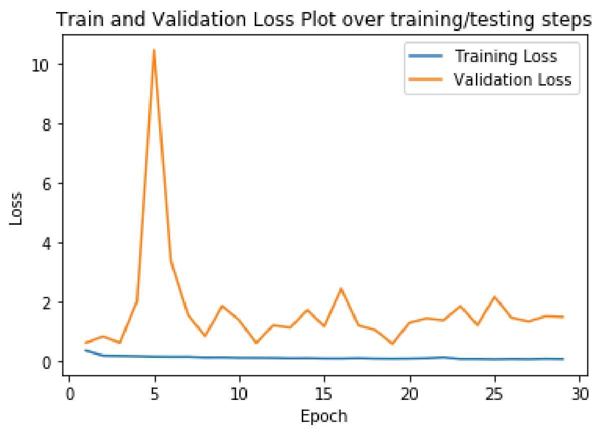
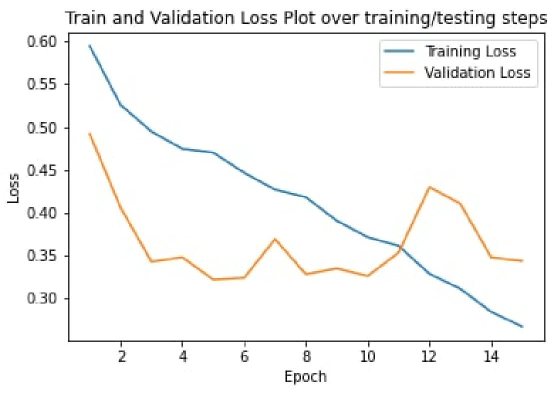


Figure 15:
With LR Scheduler (Test Accuracy 92%)

As seen in the above 2 graphs for training of the first layer model, LR Scheduler resulted in a smoother convergence (after the first peak), and overfitted later than the one without LR Scheduler (as seen in the early stopping being activated later to end at epoch 30, rather than epoch 17.

Similarly, we are able to see a similar pattern for the 2nd layer model.



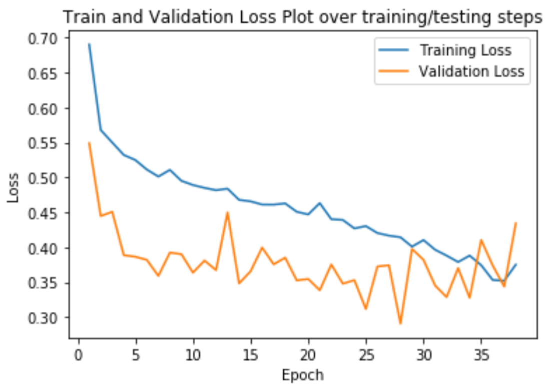


Figure 16: Without LR Scheduler (Test Accuracy 64.7%)

Figure 17: With LR Scheduler (Test Accuracy 64.7%)

Although test accuracy showed no difference, the convergence for the model with LR Scheduler was better and the model overfitted later than the model without the LR Scheduler.

In overall, however, the test accuracy only showed a slight improvement of 68% than 64% by using the LR Scheduler, which is a very minimal increase in performance.

2. Regularization for loss function, parameters and proof of benefits

As mentioned, we implemented the AdamW variant, which was initially proposed in the paper ‘Decoupled Weight Decay Regularization’. This paper reported observations that L2 regularization is not effective in Adam (and the weight decay is actually implemented wrongly in Pytorch and other libraries), and suggested the AdamW variant which improves regularization in Adam by decoupling the weight decay (which is equivalent to L2 regularization for SGD) from the gradient-based update. This AdamW variant improved the regularization in Adam for our models, in reducing the degree of overfitting.

In order to find the most optimal parameters, which was the weight decay value for the AdamW, we used Optuna hyperparameter optimisation framework as mentioned earlier in the report.

3. Compare AI solutions to how doctors diagnose infections based on xrays. Show some typical samples of failure cases and discuss what might have been the reasons.

As COVID-19 is a respiratory disease, doctors use chest imaging to diagnose people with COVID-19 symptoms while waiting for the PCR test results. In a medical review done by Louisiana State University Health Sciences Center, one of the doctors commented that "The presence of patchy and/or confluent, band-like ground glass opacity or consolidation in a peripheral and mid-to-lower lung zone distribution on a chest radiograph is highly suggestive of SARS-CoV-2 infection and should be used in conjunction with clinical judgment to make a diagnosis," [15]. The doctors observe these patches and opaque areas of the chest x-rays, which our AI is capable of detecting such cases as well through convolution. From the feature map (in the next section), we are able to see various contrasts and transparency that could potentially mean that our model may pick up these patches of opaque areas.

However, our model is not exceptionally well-performant which also indicates that our model lacks in many areas. To better understand, we analyse some wrongly classified images, especially pertaining to the second-layer model classification error (the model that differentiates covid and non-covid for infected patients.

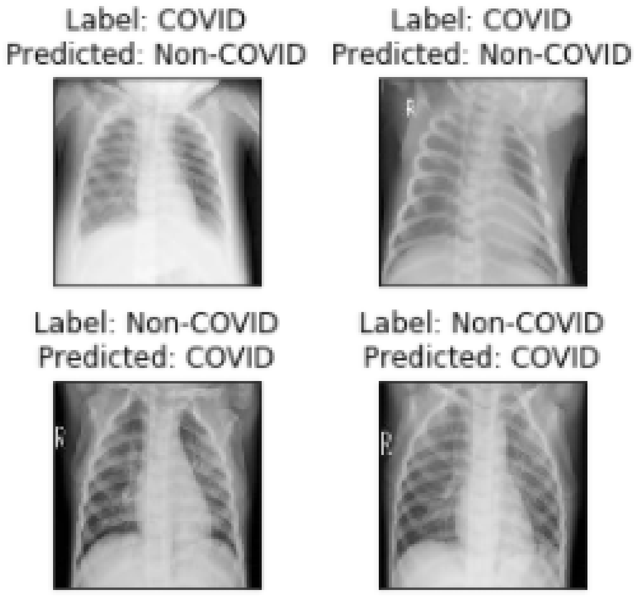


Figure 18: Samples of Wrongly Classified Images

Visually speaking, one of the most obvious reasons can be that the difference between a pneumonia symptom (which may **not** be COVID-19) and a COVID-19 symptom are highly alike, as pneumonia is one of the signs of COVID-19. And that is why doctors do not entirely rely on x-ray imaging but a PCR test to make sure that it is COVID-19. For the AI, AI can face the exact same problem that there may be very little to no differentiating semantic features between an infected covid vs infected non-covid image.

4. Feature maps visuals

An example of a feature map visual (layer 4 of the second layer model) is as follows:

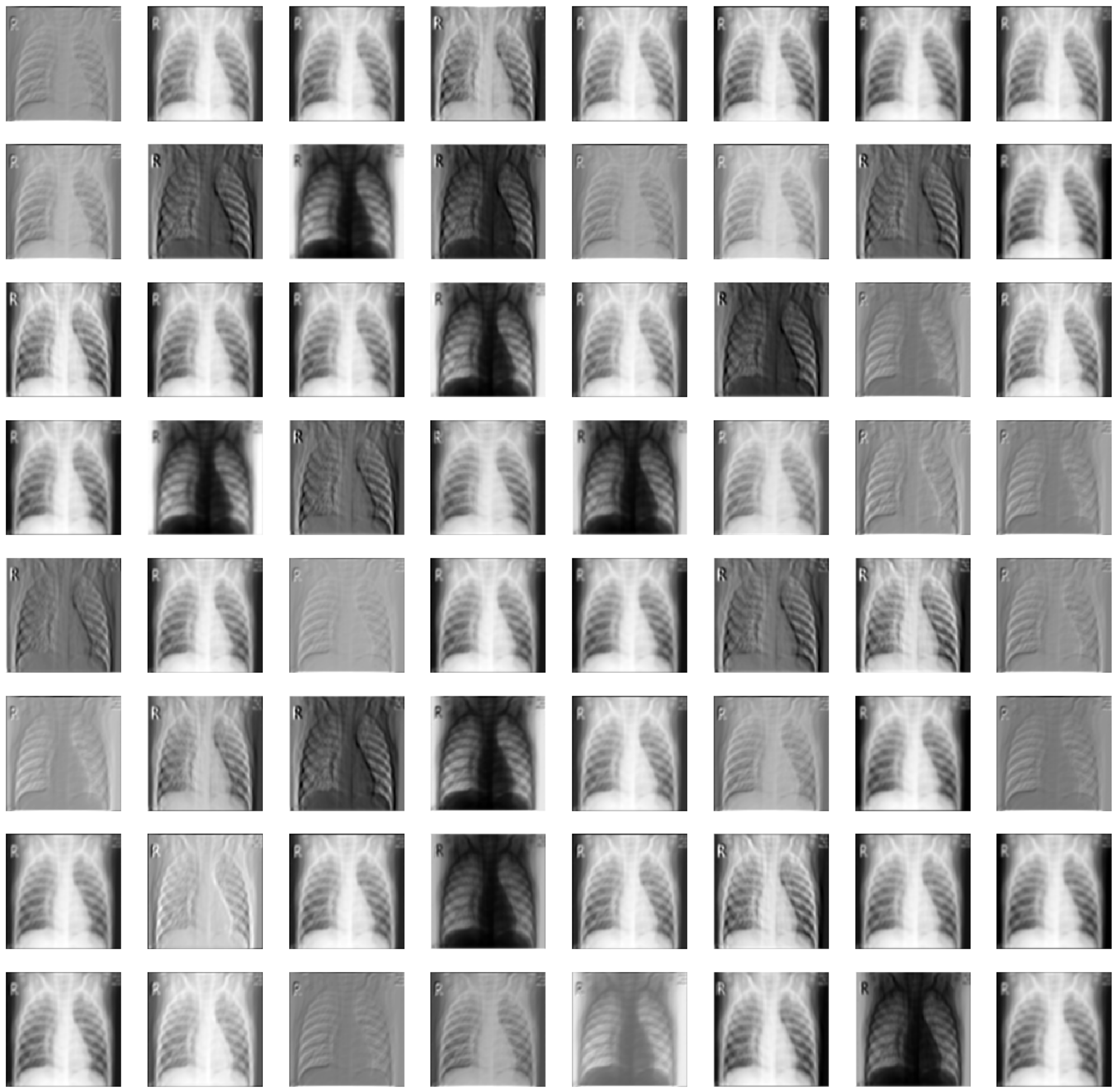


Figure 19: Feature Maps of Layer 4, Second Layer Model

5. Data Augmentation proof of benefit to our model

The following graphs show the results of model training learning, with data augmentation process implemented.

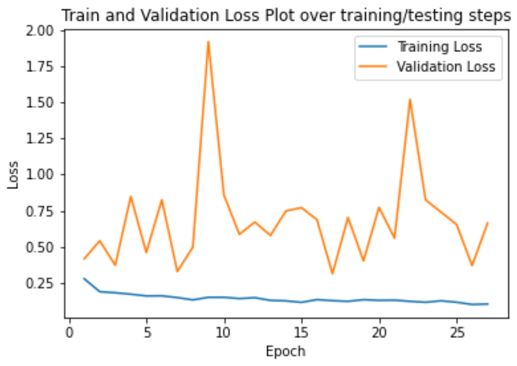


Figure 20:
First Layer Model Learning Curve

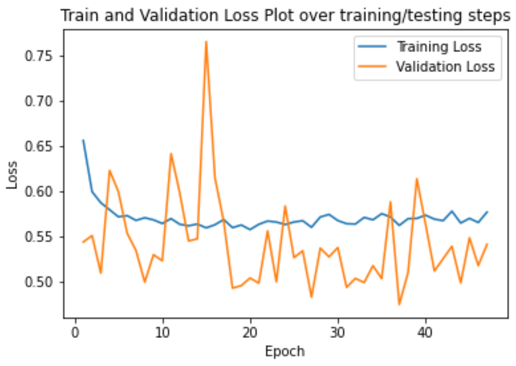


Figure 21:
Second Layer Model Learning Curve

As seen in the graphs, in comparison to the model trained without data augmentation, we can see some differences. Firstly, the models with data augmentation overfit later, shown by the higher epoch count (which means that our model continued training and achieved lower validation loss, to avoid early stopping). Secondly, we see a more stable pattern in both the training and validation learning curves, as over epochs the graphs do not go out of control. Even though the fluctuations in the graphs look to be very severe and similar to that of the vanilla model, the scale of the y-axis in the augmented graphs is much smaller than that of the vanilla graphs. As such, the degree of fluctuations in the validation loss for the augmented models is much smaller than that of the vanilla model.

References

[1] A. Ng, "Neural Networks - Normalizing inputs", 2017.

[2] H. Marius, "Overview: State-of-the-Art Machine Learning Algorithms per Discipline & per Task", Medium, 2020. [Online]. Available: https://towardsdatascience.com/overview-state-of-the-art-machine-learning-algorithms-per-discipline-per-task-c1a16a66b8bb. [Accessed: 20- Mar- 2021].

[3] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv:1409.1556 [cs], Apr. 2015, Accessed: Mar. 20, 2021. [Online]. Available: http://arxiv.org/abs/1409.1556.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," arXiv:1512.03385 [cs], Dec. 2015, Accessed: Mar. 20, 2021. [Online]. Available: http://arxiv.org/abs/1512.03385.

[5] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical Evaluation of Rectified Activations in Convolutional Network," arXiv:1505.00853 [cs, stat], Nov. 2015, Accessed: Mar. 20, 2021. [Online]. Available: http://arxiv.org/abs/1505.00853.

[6] G. Chen, P. Chen, Y. Shi, C.-Y. Hsieh, B. Liao, and S. Zhang, "Rethinking the Usage of Batch Normalization and Dropout in the Training of Deep Neural Networks," arXiv:1905.05928 [cs, stat], May 2019, Accessed: Mar. 20, 2021. [Online]. Available: http://arxiv.org/abs/1905.05928.

[7] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," arXiv:1412.6980 [cs], Jan. 2017, Accessed: Mar. 20, 2021. [Online]. Available: http://arxiv.org/abs/1412.6980.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," arXiv:1502.01852 [cs], Feb. 2015, Accessed: Mar. 20, 2021. [Online]. Available: http://arxiv.org/abs/1502.01852.

[9] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," arXiv:1907.10902 [cs, stat], Jul. 2019, Accessed: Mar. 20, 2021. [Online]. Available: http://arxiv.org/abs/1907.10902.

[10] C. Loomis, "Using Optuna to Optimize PyTorch Hyperparameters," Medium, Feb. 16, 2021. https://medium.com/pytorch/using-optuna-to-optimize-pytorch-hyperparameters-990607385e36 (accessed Mar. 20, 2021).

[11] M. Gour and S. Jain, "Stacked Convolutional Neural Network for Diagnosis of COVID-19 Disease from X-ray Images," arXiv:2006.13817 [cs, eess], Jun. 2020, Accessed: Mar. 20, 2021. [Online]. Available: http://arxiv.org/abs/2006.13817.

[12] “The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset,” ICT Express, vol. 6, no. 4, pp. 312–315, Dec. 2020, doi: 10.1016/j.icte.2020.04.010.

[13] “AdamW and Super-convergence is now the fastest way to train neural nets · fast.ai.” https://www.fast.ai/2018/07/02/adam-weight-decay/ (accessed Mar. 20, 2021).

[14] I. Loshchilov and F. Hutter, “Decoupled Weight Decay Regularization,” arXiv:1711.05101 [cs, math], Jan. 2019, Accessed: Mar. 20, 2021. [Online]. Available: http://arxiv.org/abs/1711.05101.

[15] “Radiologists find chest X-rays highly predictive of COVID-19,” ScienceDaily. https://www.sciencedaily.com/releases/2020/09/200903123254.htm (accessed Mar. 20, 2021).