Predicting COVID-19 from Chest X-Ray Images

Jacob Pennington and Selmon Gadzey
Auburn University
Auburn, AL
jtp006@auburn.edu and smg0092@auburn.edu

Abstract

COVID-19 was a worldwide pandemic in 2020 that caused a lot of panic and being able to get a test was a large concern for many people during the onset COVID. Our main goal was to see if we could use AI analysis to circumvent this issue and find a more cost-effective solution to virus detection. We employed various machine and deep learning algorithms and models in order to see if we could achieve that. Our machine learning techniques did not yield much in way of success, but we got some very promising results from our deep learning models. We believe that further research in this could allow a way for the fields of artificial intelligence and virus detection to intersect and benefit from one another.

1 Introduction

The COVID-19 viral pandemic in 2020 caused worldwide shutdown and panic. Millions of lives were lost and even more were affected by the virus. Being able to detect COVID early was paramount to keeping people safe and slowing the spread of the virus. During the onset of the virus it was very hard to get a test and even harder to get a vaccine. There were not many methods available for people due to how new everything was and researchers were scrambling to find solutions to the growing demand problem. Our aim was to see if there were any alternative ways to complete a similar task with AI-analysis. If proven successful it would greatly benefit with things like disease testing because it is more cost effective and simpler to apply trained computer models than advanced medical tech.

Our primary goal in this project was to determine if by applying machine learning and deep learning techniques we could correctly identify and detect COVID using X-ray lung images. We would take x-rays of normal lungs, mix them with COVID infected lungs and push them and test them through various algorithms. The database we used for the project was the COVID-19 Radiography Database sourced Kaggle.com. The data can be accessed at this link: https://www.kaggle.com/datasets/tawsifurrahman/covid19-radiography-database/data. Our project flow would be to take in and preprocess the data, fit them to the specifications of each algorithm used, test various parameters of those algorithms, and analyze the data taken from them. Our main metric to focus on was the recall score which lets us know how well our models identified all true COVID lungs. We prioritized it for this project because our task is related to a virus and it is much better for someone to think they have a disease while they do not, than it is for someone with that disease to not think they have it.

We used various well-known and widely used machine and deep learning algorithms. For the mahcine learning models we employed the logistic regression, a statistical model, and multilayer preceptron (MLP) models. Logistic Regression and MLP are generally used for classification task because they can identify patterns in and group together similar data points. Our main goal for the machine learning section was to get initial insight into our data and take a preliminary look at how well it would perform in AI-driven analysis. After getting our preliminary results with machine learning we moved on to the more computationally-demanding task of applying various deep learning algorithms to our data. We used various deep learning methods including a Convolutional Neural Network (CNN), a Recurrent Neural Network (RNN), and VGG, a widely used deep learning model.

Overall the deep learning techniques, particularly the CNN's outperformed the machine learning ones by a sizable margin when it came to training, precision, recall, and test accuracy. With those deep learning techniques we were able to properly classify these images and mostly succeed in our goal of applying to COVID detection. We also were able to test and confirm our hypotheses about the most appropriate models for this problem.

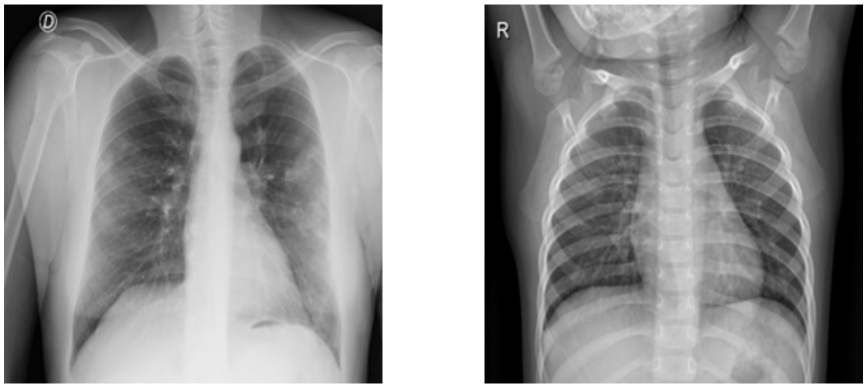


Figure 1: Example COVID (left) and Normal (right) Lung X-Ray Images

2 Methods

To tackle this image classification problem, we used Jupyter Notebooks to code in Python and train several different machine learning models.

2.1 Preprocessing

First, it was necessary that we import and preprocess the data to train the models. To do so, we downloaded the data from the Kaggle set and into our notebook. Each lung image in the Kaggle dataset also came with a corresponding "mask" that simply traced the outline of the lung of the image. We decided that we should only use the lung images with the mask applied on the image, as all the relevant information that we were hoping to train the models on came not from the noise around the lungs, but only within the actual lung X-Ray image.

So, to process the data, we loaded the square X-ray images and the masks and took a dot product of the two, so that all the pixel data values outside the mask would be 0's, and those inside the mask would be their original value. We resized the images appropriately, flattened them, and assigned the appropriate label based on the folder the image was taken from (0 if it was an image of a Normal lung, 1 if it was a COVID lung). We put all this data into X and y arrays for training, split them into training, validation, and test sets using the built-in sci-kit learn package, scaled them so that they would be better for training, and finally converted them to tensors so that our models could work with them.

2.2 MLP Models

Once we had loaded the data, we began work on training machine learning models to predict the label, COVID or Normal, of an X-Ray lung image. First, we trained several MLP models to try to obtain some baseline results. We trained these models using Pytorch packages.

The first MLP model we trained was a simple, two-layer model. The final flattened images had size 65536 (256x256), so this was the input layer. Then we added a hidden layer of 512 neurons, and an output layer of just 1 value. For all the MLP models, we used the rectified linear unit (ReLU) activation function, for computational efficiency, sparsity, and because this activation function is proven to work best for image classification problems. We then ran the final (single) value through the Sigmoid function so that it would output a probability of being a COVID-positive lung.

We trained this model, and also the rest of the MLP models, using Binary Cross-Entropy loss, which PyTorch has a built-in function for. We tested several runs and found that the hidden layer of around 500 runs worked best. We also found that a batch size of 128 gave better results than smaller and larger sizes. We recorded the average accuracy and loss at each epoch to plot training curves of the model.

It was also helpful to program a function, test_model, which allowed us to evaluate model performance. We created a function that used sci-kit learn built-in functions to return accuracy, precision, recall, and F1 given a model and a test set. We treated a value of > 0.5 as a predicted 1 and $<= 0.5$ as a predicted 0. We felt that these metrics would allow us to get a good picture of how the model was performing, as well as how it was predicting COVID-positive lungs. T-5 didn't make sense to report, as this was a binary classification problem, only 2 classes.

After the simple MLP model, we trained a more complex MLP model, consisting of three hidden layers of 1024, 512, and 100 neurons, and dropout probability of 0.5 for each layer. Once again, we tested to determine the best of these values, and to determine that a larger batch size of 256 was better for this model. We used dropout because we had noticed a slight overfitting problem with the first model. For all the MLP models, we used a learning rate of 0.001, as we found that this allowed the models to converge without jumping erratically. Once again, we plotted training curves and evaluated the model.

Due to the size of the dataset and the fact that each observation was an image, training these large models was a time-expensive process. We wanted to determine if we could train models more quickly and achieve decent results. So, we used sci-kit learn's Principal Component Analysis to decrease the dimension of the data to just 1000 data points. We fit PCA on the training data and resized the test data accordingly, then trained two models using this low-dimension data.

Once again, we trained a simple PCA model with 1 hidden layer (100 neurons), and batch size 128, and a complex PCA model with 3 hidden layers (500, 200, and 50 neurons), and batch size 32.

In all of our MLP models, we observed an overfitting problem, as the models were performing much better on the training data than on the test data. This is why we added validation sets and early stopping criteria. We fine-tuned patience values for the number of epochs the model could not improve before stopping training for each model, and plotted validation loss and accuracy on the same plots to visualize.

2.3 Logistic Regression and RNN

After we trained the MLP models, we wanted to train one Logistic Regression, just to determine if the statistical model could have a decent performance. We simply transformed the data to 1 data point and ran it through the Sigmoid function using PyTorch once again, and trained using BCE loss.

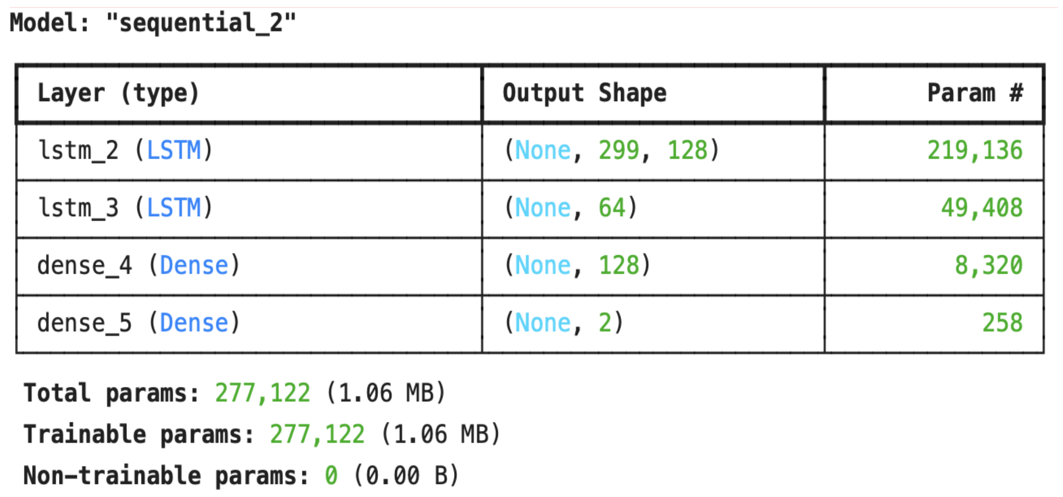
We also trained an RNN, which isn't the appropriate model for this problem. We wanted to test and confirm the result that this model would perform poorly on an image classification task. We used Keras's Sequential LSTM model and Sparse Categorical Cross-Entropy loss to train. Figure 2 shows the setup of this model.

2.4 CNN Models

Finally, we trained our most robust and appropriate models, the Convolutional Neural Networks.

First, we worked on really trying to hone and fine-tune our own CNN model. We used Keras and Tensorflow to define a sequential model that we felt would best learn the classification of the images, and went through multiple rounds of testing to tweak the number of layers, number of dense layers, the size of these layers, dropout, and more. We finally ended with a CNN that we were happy with, and Figure 3 shows the setup for this model. We trained it both for 10 epochs and for 60 epochs and compared its results.

After we trained the sequential CNN, we trained a VGG model that we felt would really give good performance on the test data. We used Keras to build and optimize this model, and tested to go with a

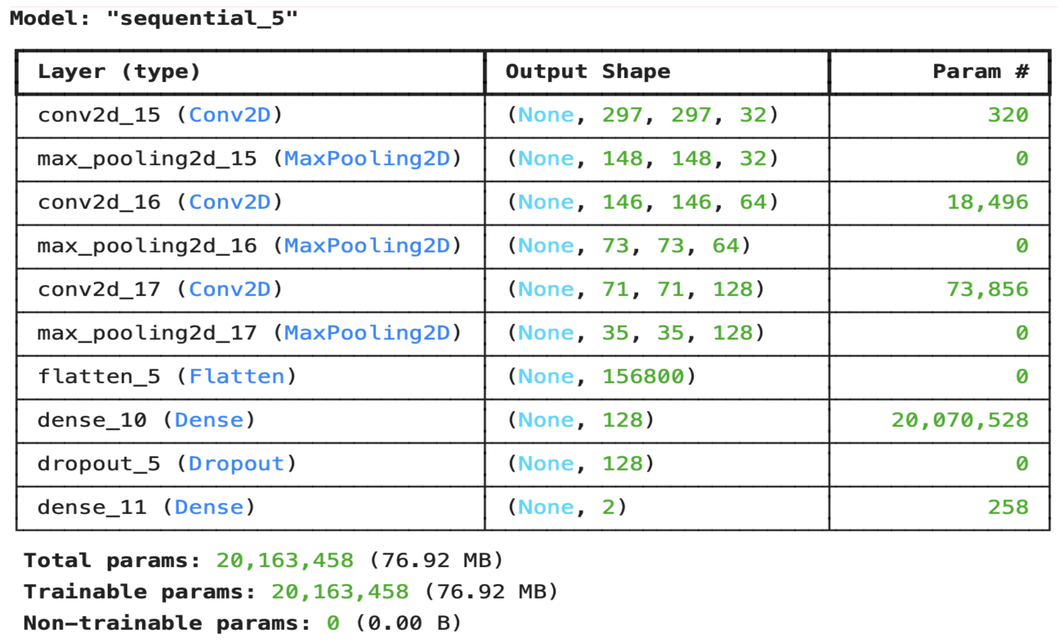


Model: "sequential_2"

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 299, 128)	219,136
lstm_3 (LSTM)	(None, 64)	49,408
dense_4 (Dense)	(None, 128)	8,320
dense_5 (Dense)	(None, 2)	258

Total params: 277,122 (1.06 MB)
Trainable params: 277,122 (1.06 MB)
Non-trainable params: 0 (0.00 B)

Figure 2: RNN Model Setup



Model: "sequential_5"

Layer (type)	Output Shape	Param #
conv2d_15 (Conv2D)	(None, 297, 297, 32)	320
max_pooling2d_15 (MaxPooling2D)	(None, 148, 148, 32)	0
conv2d_16 (Conv2D)	(None, 146, 146, 64)	18,496
max_pooling2d_16 (MaxPooling2D)	(None, 73, 73, 64)	0
conv2d_17 (Conv2D)	(None, 71, 71, 128)	73,856
max_pooling2d_17 (MaxPooling2D)	(None, 35, 35, 128)	0
flatten_5 (Flatten)	(None, 156800)	0
dense_10 (Dense)	(None, 128)	20,070,528
dropout_5 (Dropout)	(None, 128)	0
dense_11 (Dense)	(None, 2)	258

Total params: 20,163,458 (76.92 MB)
Trainable params: 20,163,458 (76.92 MB)
Non-trainable params: 0 (0.00 B)

Figure 3: CNN Model Setup

final batch size of 16. We used sparse categorical cross-entropy loss. We trained this model five times and evaluated each run.

3 Results

In total, we trained **8** different tuned models to tackle the image classification problem. We trained one logistic regression model, four MLP models (two using the original data, and two using PCA dimension-reduced data), two CNN's (one being a VGG model), and an RNN. Here, we will go into details describing the results from each model.

3.1 Logistic Regression

The first model we trained was a very simple logistic regression model. Figure 4 shows both the loss and accuracy of the training set and the validation set at each training epoch. As you can see, validation loss and accuracy didn't change significantly at all, while training set loss decreased and accuracy improved. This shows that the logistic regression model was overfitting to the training data and wasn't learning important features from it.

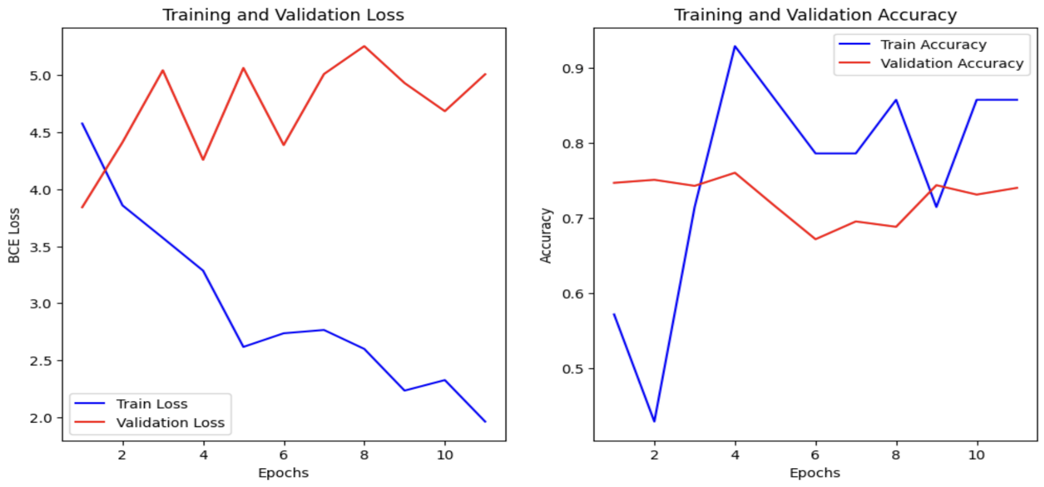
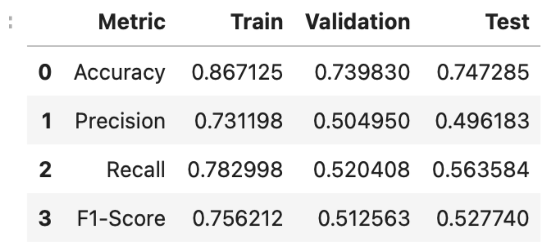


Figure 4: Logistic Regression Training Curves

Figure 5 shows the evaluation metrics we used for this image classification problem that the trained Logistic Regression model gave. We recorded Accuracy, Precision, Recall, and F1-Score for the Training Set, Validation Set, and the Test Set. We treat a COVID label as a positive label and a Normal lung label as a negative label.



	Metric	Train	Validation	Test
0	Accuracy	0.867125	0.739830	0.747285
1	Precision	0.731198	0.504950	0.496183
2	Recall	0.782998	0.520408	0.563584
3	F1-Score	0.756212	0.512563	0.527740

Figure 5: Logistic Regression Evaluation Metrics

The logistic regression model had an accuracy of 86.71% on the training data, and only an accuracy of 74.73% on the test data. Our data consists of 10912 observations of Normal lungs and 3616 observations of lungs with COVID, meaning that 73.8% of the data is labeled normal. So, a test accuracy of 74.7% is hardly an improvement over just labeling all lungs as Normal lungs.

The recall and precision are similarly bad. The test recall is 0.563, meaning that only 56% of lungs with COVID were correctly identified. The test precision is 0.496, meaning that just under half of the model's predicted COVID lungs are actually COVID positive. The Test F1 score is 0.528. The trainig metrics being much better than these in all categories shows that the model has overfit very much and doesn't learn hardly any really important information.

However, comparing a statistical Logistic Regression model to a neural network is an unfair comparison, as they are entirely different things. This simply confirms that the logistic regression model won't be a good choice for this image classification problem, which we knew going in.

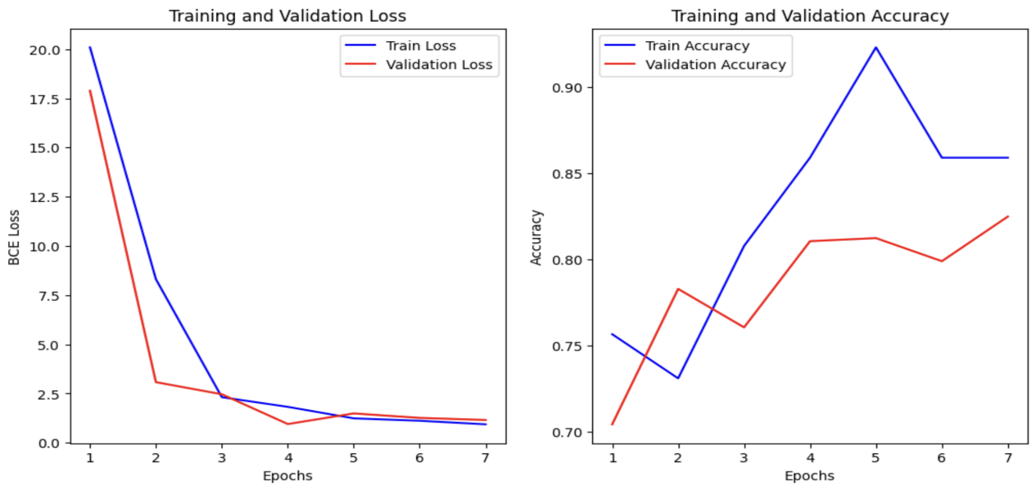
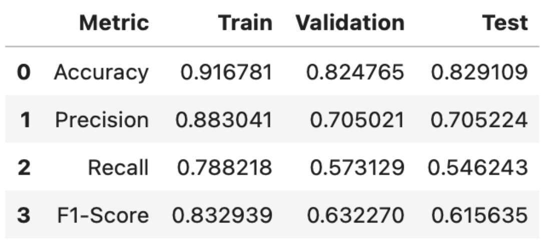


Figure 6: Simple MLP Training Curves

3.2 Simple MLP

The next model we trained was the very simple MLP model with 2 layers. Figure 6 shows the training curves for this model, and Figure 7 shows the metrics.



	Metric	Train	Validation	Test
0	Accuracy	0.916781	0.824765	0.829109
1	Precision	0.883041	0.705021	0.705224
2	Recall	0.788218	0.573129	0.546243
3	F1-Score	0.832939	0.632270	0.615635

Figure 7: Simple MLP Evaluation Metrics

The training curves show a very good decrease in both test and train loss as well as an increase in both test and training accuracy. As mentioned in the methods, patience was tuned, but it looks like in this training the model could have continued to improve by being allowed to run for a more epochs.

This model decent training results - 0.917 accuracy, 0.883 precision, and 0.788 recall. These could have improved if the model had trained for more epochs, at the tradeoff of potential overfitting.

The model also preformed decently on the test data, but certainly worse than on the training data. It gave a test accuracy of 82.9%, which is an improvement over labeling all lungs as negative. It had a test precision of 70.5%, showing that most of the model's positive predictions were actually positive. However, the test recall of 54.6% is very poor, especially for this problem. The model is only identifying half of the lungs with COVID, which would result in bad consequences if it were used in practice.

3.3 Complex MLP with Dropout

We also trained a more robust MLP model that consisted of four layers and employed dropout to reduce overfitting. Figures 8 and 9 show the training curves and metrics.

The training curves clearly show that the early stopping was more effective for this model. Both validation loss and accuracy tapered off, and the training stopped after a couple more epochs accordingly.

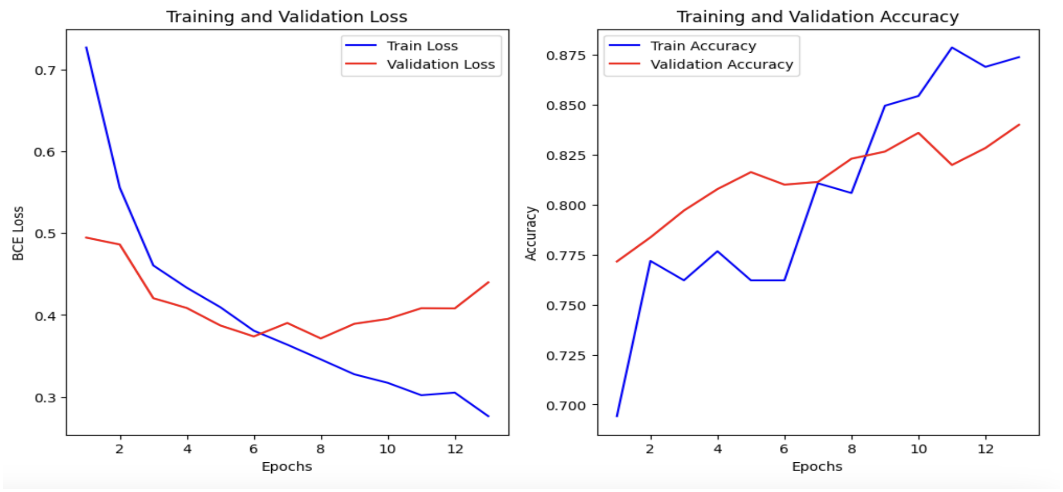
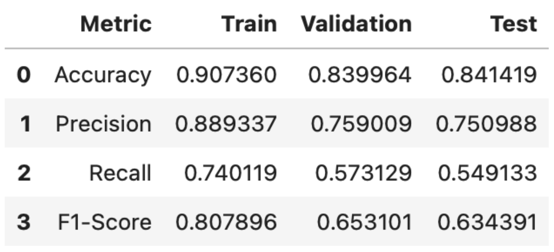


Figure 8: Complex MLP Training Curves



	Metric	Train	Validation	Test
0	Accuracy	0.907360	0.839964	0.841419
1	Precision	0.889337	0.759009	0.750988
2	Recall	0.740119	0.573129	0.549133
3	F1-Score	0.807896	0.653101	0.634391

Figure 9: Complex MLP Evaluation Metrics

This model performed better than the simple MLP model for all the metrics. Its test accuracy of 84.1% and test precision of 75.1% were improvements, and they show that this model is better at identifying *correct* COVID lungs. However, the test recall of 54.6% is hardly an improvement, and still very poor. The F1-score of 0.6156 is not great due to this recall value.

Ultimately, this model performed the best on the test data out of any of the MLP models that we trained. The more complex setup of this model allowed it to learn more from the training data. However, its performance was still not close to the results we were looking for.

3.4 Simple MLP using PCA-Reduced Data

We also trained models using dimension-reduced training data, reduced using Principal Component Analysis. The first was a very simple two layer model. Figures 10 and 11 show the training curves and metrics.

The training curves for this model were very disturbing. The training loss decreased and training accuracy improved, but the validation loss only increased and validation accuracy remained constant.

On the training data, this reduced-dimension model had extremely high numbers; accuracy, precision, recall, and F1 were all about 98%. But these numbers in addition to the training curves indicate that the model may be overfitting quite a bit.

Surprisingly, the model still performed decently on the test data. It gave an accuracy of 81.7% which was similar to the other MLP models, though its test precision of 64.5% indicates that it didn't predict COVID-positive lungs with very much correctness. Its test recall of 0.604 indicates that the model was able to locate COVID lungs better than any of the models so far, and its F1 of 0.623 was also better than the complex high-dimension MLP model.

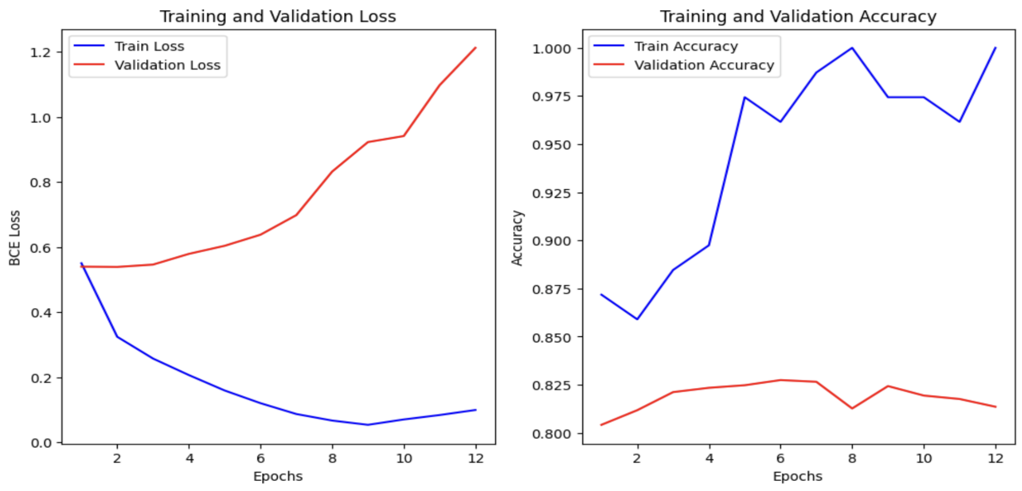
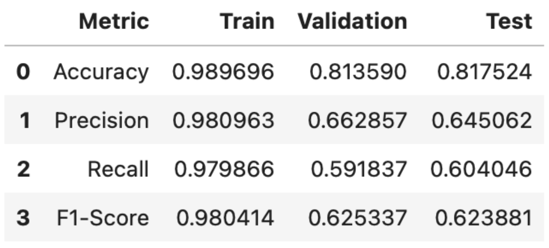


Figure 10: Low-Dimension MLP Training Curves



	Metric	Train	Validation	Test
0	Accuracy	0.989696	0.813590	0.817524
1	Precision	0.980963	0.662857	0.645062
2	Recall	0.979866	0.591837	0.604046
3	F1-Score	0.980414	0.625337	0.623881

Figure 11: Low-Dimension MLP Evaluation Metrics

The overfitting problem that this model had wasn't fixed by simplifying it; it could be that the model simply wasn't able to learn important features from the PCA-reduced training data.

3.5 Complex MLP using PCA-Reduced Data

The last MLP model that we trained was a more complex, four-layer model using the PCA-reduced training data. Figures 12 and 13 show the training curves and metrics for these models.

The training curves for this model look very similar to those of the first PCA model, with the training loss decreasing and accuracy increasing, while validation loss skyrockets and accuracy remains constant. Again, this shows that the model overfit using the dimension-reduced data.

The training evaluation metrics weren't as good for this model as the first PCA model, indicating that it may not have overfit as much as the first. The test results also back up this claim. On the test data, the model gave 83.2% accuracy, 68.6% precision, 60.7% Recall, and an F1 of 0.644.

This model gave better accuracy, precision, and recall than the other low-dimension model, and a very similar recall. If choosing between all the MLP models, even though the complex high-dimension model gave better accuracy and precision, this model may be preferable due to its higher recall. This is significant in this scenario because of the importance of flagging as many COVID-positive lungs as possible.

3.6 RNN

After we trained the MLP models, we trained an RNN model. We know that this isn't the correct kind of model for a question like this, and our testing confirmed this result. Figure 14 shows the metrics evaluated on the test data for the RNN model.

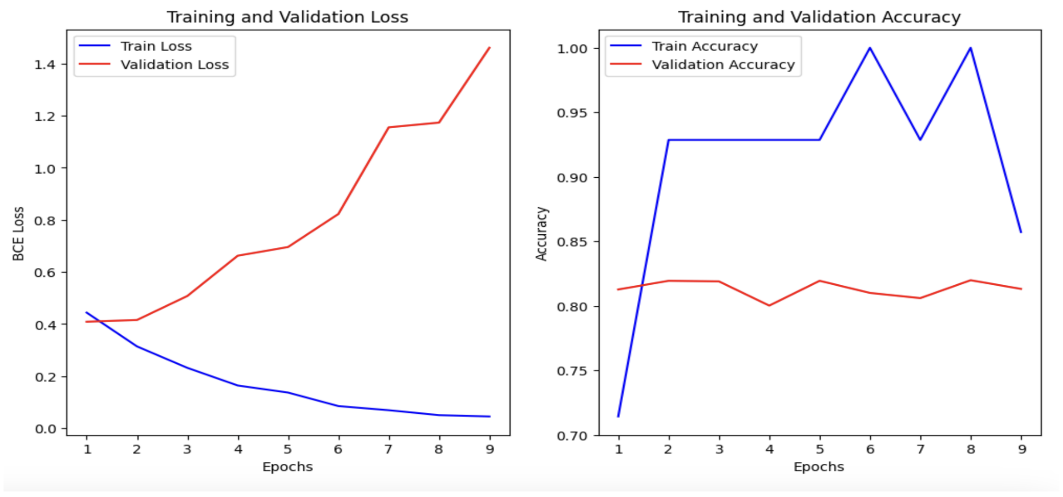
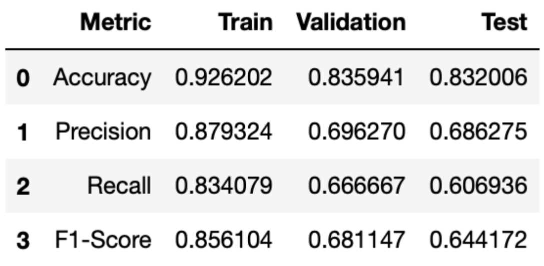
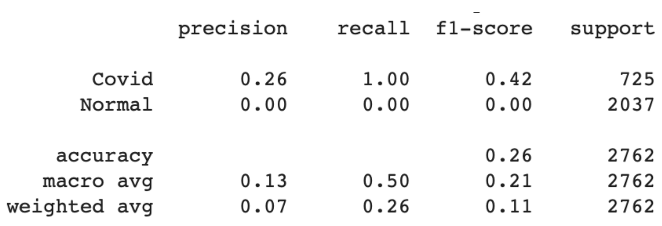


Figure 12: Complex Low-Dimension MLP Training Curves



	Metric	Train	Validation	Test
0	Accuracy	0.926202	0.835941	0.832006
1	Precision	0.879324	0.696270	0.686275
2	Recall	0.834079	0.666667	0.606936
3	F1-Score	0.856104	0.681147	0.644172

Figure 13: Complex Low-Dimension MLP Evaluation Metrics



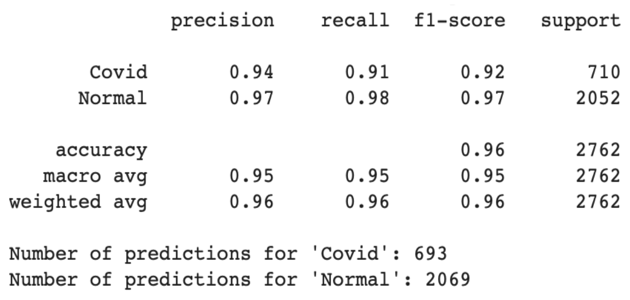
```
              precision    recall  f1-score   support

       Covid       0.26      1.00      0.42       725
      Normal       0.00      0.00      0.00      2037

    accuracy                           0.26      2762
   macro avg       0.13      0.50      0.21      2762
weighted avg       0.07      0.26      0.11      2762
```

Figure 14: RNN Test Metrics

Clearly, this model preformed horribly on the test data. Its accuracy of 26% and recall of 100% indicate that it only learned to predict that every observation was a COVID-positive observation. This coincides with our understanding that RNN is not the correct model for this problem.

3.7 CNN

We trained and fine-tuned a complex Convolutional Neural Network that we hoped would provide very good results on the test data for our problem. Figure 15 shows the metrics on the test data that this model gave.



```
              precision    recall  f1-score   support

       Covid       0.94      0.91      0.92       710
      Normal       0.97      0.98      0.97      2052

    accuracy                           0.96      2762
   macro avg       0.95      0.95      0.95      2762
weighted avg       0.96      0.96      0.96      2762

Number of predictions for 'Covid': 693
Number of predictions for 'Normal': 2069
```

Figure 15: CNN Test Metrics

As we predicted and as we hoped, this model had the best performance out of any model we saw so far. On the test data, it had an accuracy of 96%, much higher than any of the MLP accuracies. It had a precision of 94% (when treating COVID cases as positives), indicating that the model was able to correctly predict COVID cases quite well, and a recall of 91%, indicating that it located them well also, although still not as well as we hoped. The F1 of 0.92 indicates that precision and recall are very good, as the highest MLP F1 we got was 0.64. Obviously, the precision, recall, and F1 when treating Normal lungs as positives was greater, since there were more of them.

Ultimately, we had great results when using our tuned CNN. Its ability to detect features in the lung X-Ray images was much better than the MLP models' because of the way that it processes each image. However, the recall of 0.91 was still not as great as we hoped, as false negatives in this classification problem are important not to miss.

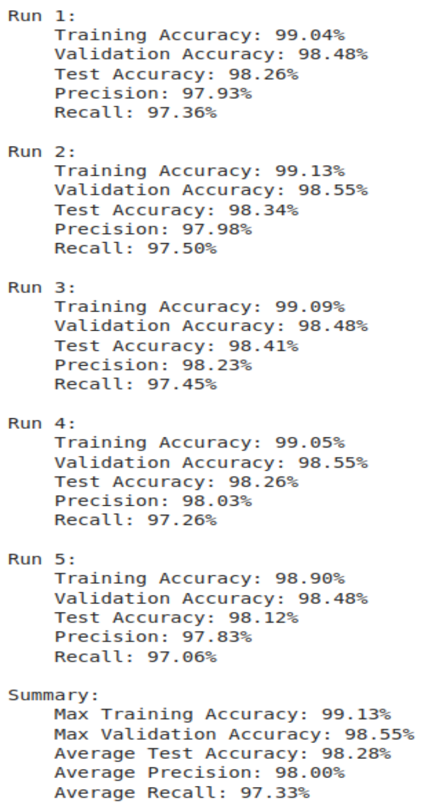
3.8 VGG

The last model that we trained was a VGG model. This robust and more complex Keras CNN gave us the best results out of any of the models we trained. We ran the model 5 times, and Figure 16 shows the results of the runs. The nature of this model is different from the others. The model only takes 3D dimensional data so our preprocessing involved adding another color channel for to the x-ray images. The VGG model preformed exceedingly well - test accuracy was consistently 98%, Precision was 98%, and Recall was about 97%. This means that the model was able to discern key features in the data much better than any model so far. The high recall tells us that the model is able to identify almost all of the COVID lungs, and the high precision tells us it identifies them with high correctness as well. The model was able to obtain great training accuracy without overfitting.

4 Conclusions

In conclusion, we were able to use machine learning techniques to accurately predict whether a lung is COVID-positive or normal given an X-ray image.

Through our testing, we found that, as expected, RNN's and Logistic Regression were not appropriate for the task we set out to accomplish. Our MLP models, though they gave an improvement in predicting COVID given an image over just guessing, did not provide results that we were satisfied with. However, the CNN's, which are well suited to image classification problems, were able to predict with high accuracy, precision, and recall whether a lung image was COVID positive or not.



```
Run 1:
    Training Accuracy: 99.04%
    Validation Accuracy: 98.48%
    Test Accuracy: 98.26%
    Precision: 97.93%
    Recall: 97.36%

Run 2:
    Training Accuracy: 99.13%
    Validation Accuracy: 98.55%
    Test Accuracy: 98.34%
    Precision: 97.98%
    Recall: 97.50%

Run 3:
    Training Accuracy: 99.09%
    Validation Accuracy: 98.48%
    Test Accuracy: 98.41%
    Precision: 98.23%
    Recall: 97.45%

Run 4:
    Training Accuracy: 99.05%
    Validation Accuracy: 98.55%
    Test Accuracy: 98.26%
    Precision: 98.03%
    Recall: 97.26%

Run 5:
    Training Accuracy: 98.90%
    Validation Accuracy: 98.48%
    Test Accuracy: 98.12%
    Precision: 97.83%
    Recall: 97.06%

Summary:
    Max Training Accuracy: 99.13%
    Max Validation Accuracy: 98.55%
    Average Test Accuracy: 98.28%
    Average Precision: 98.00%
    Average Recall: 97.33%
```

Figure 16: VGG Results

Ultimately, the VGG model that we trained was able to predict the binary diagnosis of a lung with 98.28% accuracy, 98.00% precision, and 97.33% recall. This is an important result, as it shows that there is enough information in the lung images to classify health, and it demonstrates that lung X-ray images could be used to detect whether a patient has COVID or not in the future. This also demonstrates that CNN's are powerful tools in recognizing patterns in images, as it is difficult for even a human to recognize the difference between COVID-positive and Normal lungs.

Both Selom and Jacob worked hard on this project, and each acknowledges the other's work. Jacob worked on the MLP and Logistic Regression Models, and Selom worked on the CNN, RNN, and VGG models.