COVID-19 Diagnosis with Neural-Networks

Project Report By:

Kratica Rastogi

Ankita Chakraborty

CSS 581 A (Machine Learning)

University of Washington, Bothell

Autumn 2020

Abstract

First identified in 2019 in Wuhan, China, COVID-19 is an infectious disease caused by severe acute respiratory syndrome. The disease can cause a range of breathing problems, from mild to critical. While 80% of the infected are asymptomatic, older adults and people with pre-existing health conditions like diabetes, kidney, lung, or heart disease may have more serious symptoms. When the virus SARS-CoV-2, gets in our body, it meets the mucous linings of our eyes, mouth, and nose. A healthy cell is invaded by the virus and it keeps replicating. The new viruses infect nearby cells, and they keep proliferating. As it travels down the airways, the linings can become irritated and inflamed. If our respiratory tract is imagined to be an upside-down tree, the trunk is our trachea which splits in our lungs into smaller branches. Each branch ends in tiny air sacs called alveoli. The novel coronavirus can infect the upper or lower part of the respiratory tract and sometimes can reach all the way down into the alveoli (when people develop pneumonia). These signs of respiratory inflammation can be analyzed by doctors by performing a chest Xray or CT scan. For an infected individual, a chest CT, might seem like "ground-glass opacity" because it looks like frosted glass on a shower door.

1. Problem Statement & Proposed Solution

Chest X-Ray (CXR) is the globally accepted standard for performing preliminary investigations of pulmonary ailments in a noninvasive manner. [1] The infected patients have displayed discrete trend of visual features in CXR images in all the preceding studies. [2] Where current lab test (RT-PCR) results of nose and throat samples take nearly 24 hours, CXR can act as a speedy alternative screening mechanism for the detection of nCOVID-19 or to validate the related diagnosis. Development of an AI system for the development of Covid-19 in lungs could help in rapid screening of patients, aiding in speedy detection of the virus which could potentially protect several associated lives by enforcing an early isolation.

So, in this project as a preliminary step in the recognition and analysis of Human chest rays for COVID – 19 infections we have trained various benchmarked Machine learning models to detect whether an X-Ray image is COVID or non-COVID. Based on the accuracy and performance from the different models we aim at selecting the best model that we shall work on to accomplish our future goals.

There is no specific application developed to detect COVID disease from chest x-ray images. But few research and experiments were done. Some of them are as follows:

In October 2020, University of Minnesota trained a model using 100,000 X-rays of patients who did not have COVID-19 and 18,000 X-rays of patients who did. Their tool could detect COVID-19 in seconds and display the risk score right away

In November 2020, a locally developed AI model trained with 35,000 CXRs gained approval from Drap – the Drug Regulatory Authority of Pakistan

Again, in late November this year, another breakthrough by the Northwestern University – DeepCOVID-XR could perform better than a team of specialized thoracic radiologists by spotting COVID-19 in X-rays about 10 times faster and 1%-6% more accurately. Accuracy from the radiologists ranged from 76% to 81%, while the algorithm had 82% accuracy.

2. Background

In the recent months various research around the world have realized this alternative and worked on building a more robust and accurate Covid-19 Detection Model from X-Rays. After the careful reflection and speculation of the various researches of our predecessors, we found that the number of CXR images used in the existing studies were quite limited number, which may have led to "under-fitting of the data-hungry DL models" [3] For this reason there are not many accurate automated tools available to appropriately diagnose and predict the extent of abnormalities in the lungs. Salient information about the COVID-19 virus is provided by recent findings obtained using radiology imaging techniques. Application of Advanced artificial intelligence (AI) techniques along with radiological imaging can be utilized to make accurate detection of COVID disease.

Once the application for detecting COVID-19 using x-ray is developed then it can be useful in healthcare departments. Findings using COVID-19 detection application and x-ray images will help doctors to confidently diagnose COVID-19 disease in patient.

The dataset which we have used in our project has x-ray images of patients who have COVID-19, other respiratory diseases, and patients with no disease. We have taken this dataset from Kaggle website.

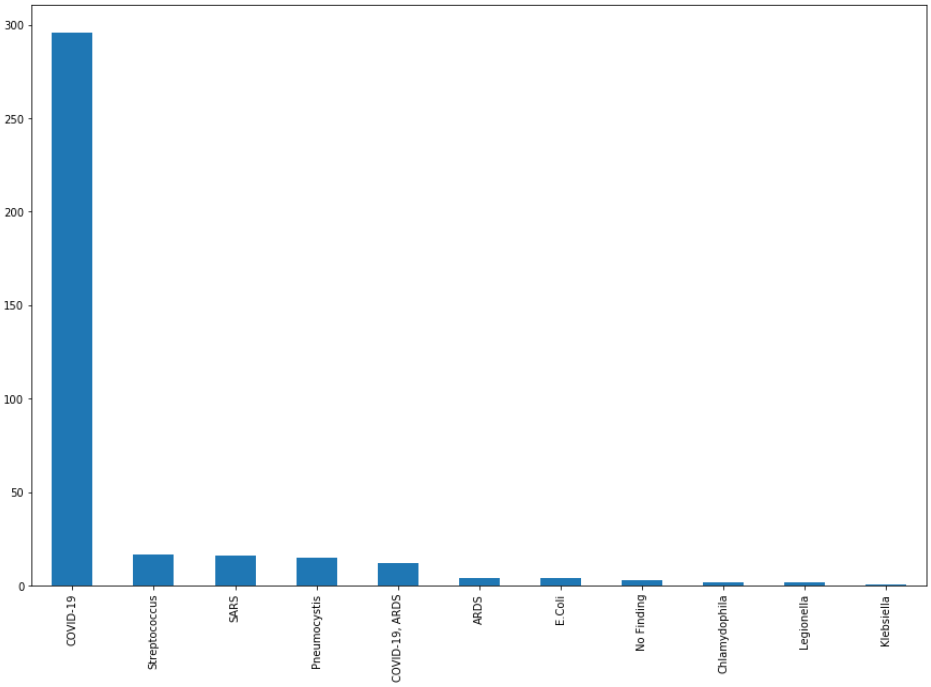
COVID-19 chest xray | Kaggle

COVID-19 Radiography Database | Kaggle

3. Data

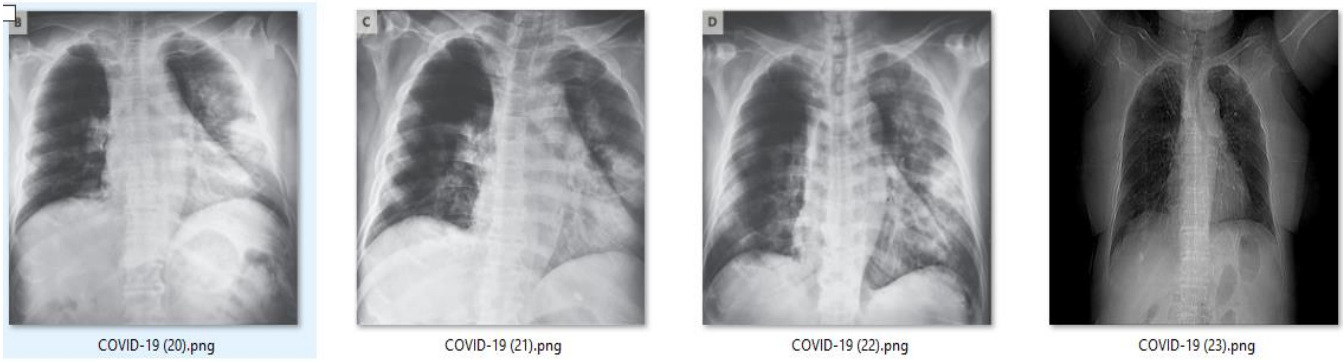
A. Data Preparation

A machine learning model was trained on chest X-Ray images. One of the chest X-Ray image datasets was: https://www.kaggle.com/bachrr/covid-chest-xray. It was a 240 Mb dataset which is a database of Covid-19 cases paired with other respiratory diseases such as MERS, SARS and ARDS.
The challenges we faced with this database was that it had only X-Ray images of infected lungs. We thought that to train our model with greater accuracy, the training dataset set had to be larger and had to cover a greater variety of data. Also, this dataset has 296 covid images and approximately 64 other respiratory diseases. In that case while training the model, the model will be more bias towards covid images as compared to other type of respiratory diseases.



Above figure shows the number of patients infected with various respiratory diseases.

We also used the dataset from the below link:
https://www.kaggle.com/tawsifurrahman/covid19-radiography-database
This database contains 219 Covid-19 Images and 1341 Normal X-Ray Images. We reduced the normal images to those of number of Covid images. So that the model should be trained on equal amount of dataset for both normal and Covid x-ray images. Going with this dataset lent our models enhanced accuracy. The below figure :1 offer a depiction of the various kinds of images available in the 1 GB dataset.



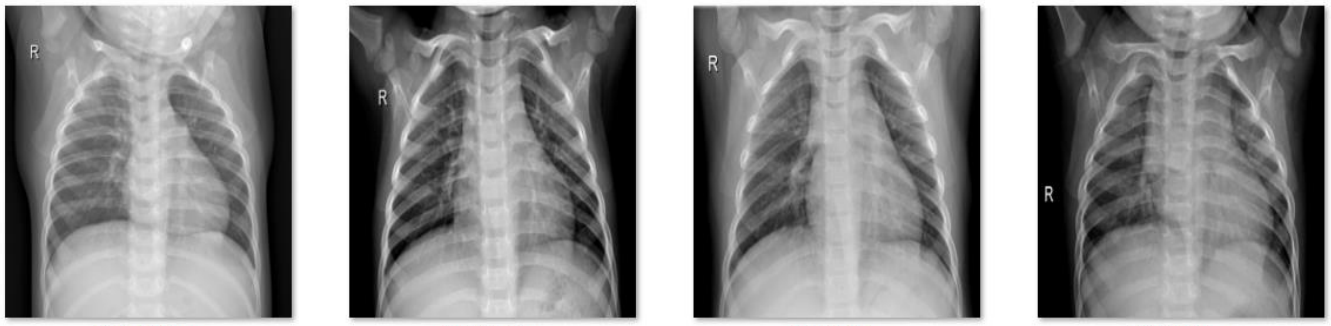


Figure: 1

As a subsequent step, the images had to be picked from the physical drive location in which they had been stored and saved in our train/test datasets. We read these images by using the Open CV library.
In this context let us understand how computer vision and Open CV works in a nutshell. Forming the foundation of Artificial Intelligence, Computer vision is the process of comprehending images and videos and how to store, manipulate or retrieve data from them. Image processing is analyzing and manipulating of a digitized image, with the specific aim of improving its quality. Machines perceive everything including images and videos as combinations of numbers. Since it treats colors as RGB triples, the values it picks are based on the various intensities of Red, Green or Blue between 0 and 255. [4]
In the setting of our project, we had to find an application that would read our X Rays in the best way possible. We came across various software like Amazon Rekognition, Google Cloud Vision API, SimpleCV, Deepdream, Clarifai, DeepPy etc. [5]
But we zeroed in on OpenCV because this software-toolkit works best when coupled with Convoluted Neural Networks and Deep Neural Networks and is by far the best open-source tool available. OpenCV was used in our code in the following way

```
import cv2
```

```
    for filename in os.listdir(os.path.join("C:/Users/Kratica Rastogi/PycharmProjects/FinalProject/COVID-19 Radiography Database", "COVI
-19")):
        img = cv2.imread(os.path.join("C:/Users/Kratica Rastogi/PycharmProjects/FinalProject/COVID-19 Radiography Database", "COVID-19",
filename), cv2.IMREAD_GRAYSCALE)
        if img is not None:
            images.append(img)
            labels.append("covid")
```

Figure: 2

Since these high-resolution X-Ray images were taking up a lot of memory space when we read them by doing just cv2.imread, we were not able to load the entire dataset. So, we decided to convert the images to Grayscale with cv2.IMREAD_GRAYSCALE.

All these images were then stored in a 2-D array images[] and their respective annotations was saved in the array named labels[]

```
    if img is not None and count_covid_images > 0:
        images.append(img)
        labels.append("normal")
```

Figure: 3

Next the dataset was randomly divided into train and test data with 75% of the total data dedicated towards training and 25% for validation.

B. Data Preprocessing

The image data now a part of our 2-D array images [] must be reshaped to the desired dimension without the changing the inherent data. This is done to build a matrix representation of the RGB triplet.

```
def preprocess_data():
    reshaped_train_images = train_images.reshape(len(train_images), 1024 * 1024)
    reshaped_test_images = test_images.reshape(len(test_images), 1024 * 1024)
    le = preprocessing.LabelEncoder()

    # Normalize pixel values to be between 0 and 1
    return reshaped_train_images / 255.0, reshaped_test_images / 255.0, le.fit_transform(
        train_labels), le.fit_transform(test_labels)
```

Figure: 4

C. Feature extraction and selection

This step is achieved by doing a Principal Component Analysis, or PCA, which is a dimensionality-reduction method. We are going to condense the dimensionality of our data sets which is quite voluminous, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

Basically, the idea behind PCA is to preserve as much information as possible by reduction of number of variables in it for the sake of simplicity.[6] Here as observed in Figure 5 we have chosen all important pixels from the image. The image size in 1GB dataset was 1024x1024x3. Finally, we applied PCA by taking n_components as 196 (14x14x1) as PCA = min (n_samples, n_features)

```python
def preprocess_data():
    """
    This method will normalize the data by dividing by 255.
    The normalized values will lie between 0 and 1
    Number of features before applying PCA 1048576 (1024*1024)
    We are trying to reduce to n_components

    :return: train_images, test_images, train_labels, test_labels
    """
    le = preprocessing.LabelEncoder()

    # applying pca and reducing to 14*14
    pca = PCA(n_components=196)

    train_images_reshaped = train_images.reshape(len(train_images), 1024 * 1024)/255.
    test_images_reshaped = test_images.reshape(len(test_images), 1024 * 1024)/255.

    pca_train_images = pca.fit_transform(train_images_reshaped)
    pca_test_images = pca.transform(test_images_reshaped)

    train_images_reshaped = pca_train_images.reshape(len(train_images), 14, 14, 1)
    test_images_reshaped = pca_test_images.reshape(len(test_images), 14, 14, 1)

    # Normalize pixel values to be between 0 and 1
    return train_images_reshaped, test_images_reshaped, le.fit_transform(train_labels), le.fit_transform(test_labels)
```

Figure: 5

D. Performance evaluation metrics

The performance of our models was evaluated using the four performance metrics, Accuracy, Recall, Precision and F1-Score, “where, the number of infected and normal CXR images correctly predicted by the proposed system was denoted by true positive (TP) and true negative (TN), respectively; the false positive (FP) and false-negative (FN) denoted the misclassification of normal and infected images, respectively; P = TP + FN and N = TN + FP.” [2]

$$accuracy = \frac{\sum true\ positive + \sum true\ negative}{\sum total\ population}$$

Figure: 6 [7]

$$recall = \frac{true\ positives}{true\ positives\ + false\ negatives} \qquad precision = \frac{true\ positives}{true\ positives + false\ positives}$$

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Fig: 7 [8]

Finally, we make the use of the confusion matrices with and without normalization in order to visually interpret how the labels are predicted. [9] This is especially useful because our datasets are unbalanced. The number of Covid-19 CXRs are smaller than the Normal and Other Respiratory diseases combined.

	PREDICTED: FRAUD	PREDICTED: NOT FRAUD
ACTUAL: FRAUD	TRUE POSITIVE	FALSE NEGATIVE
ACTUAL: NOT FRAUD	FALSE POSITIVE	TRUE NEGATIVE

Figure: 8 [9]

Figure 8 above, shows what the confusion matrix depicts. The final goal is to maximize the number of true positives and true negatives (the right predictions) and minimize the number of false negatives and false positives(the erroneous predictions). We have also normalized the pixel value of images to 0 or 1 and performed using label encoding techniques to normalize the labels.

4. Experimental results and discussions

A. KNN

MachineLearningProject/knn.ipynb at main · KraticaRastogi/MachineLearningProject (github.com)

The first algorithm we tried to use was K-Nearest Neighbors or KNN which is a very simple algorithm working on the principle of "Birds of a feather flock together". This algorithm assumes that similar things exist in proximity. We chose KNN to begin our analysis because of few of the advantages it brings to the table. It comes without any training period which means that it stores the training data and learns

in the real time. This enhances its speed and also aids in seamless addition of data without any loss of accuracy. [10] Further the implementation of this algorithm is also quite simple like below.

```
def create_and_train_model():
    knn = KNeighborsClassifier(n_neighbors=2)
    return knn.fit(train_images, train_labels)
```

Figure: 8

But before the KNN model is trained the data must be preprocessed.

At first, we were first trying to execute this model with K = 3 but the results were not so great. So, we changed its value to 2 and below are the various metrics obtained after executing this model

KNN results are:

```
accuracy score: 0.9636363636363636
precision score: 0.9642975206611569
recall score: 0.9636363636363636
f score: 0.9636483886483884
Confusion Matrix: [[52  1]
 [ 3 54]]
Confusion matrix, without normalization
[[52  1]
 [ 3 54]]

Process finished with exit code 0
```

Figure: 9

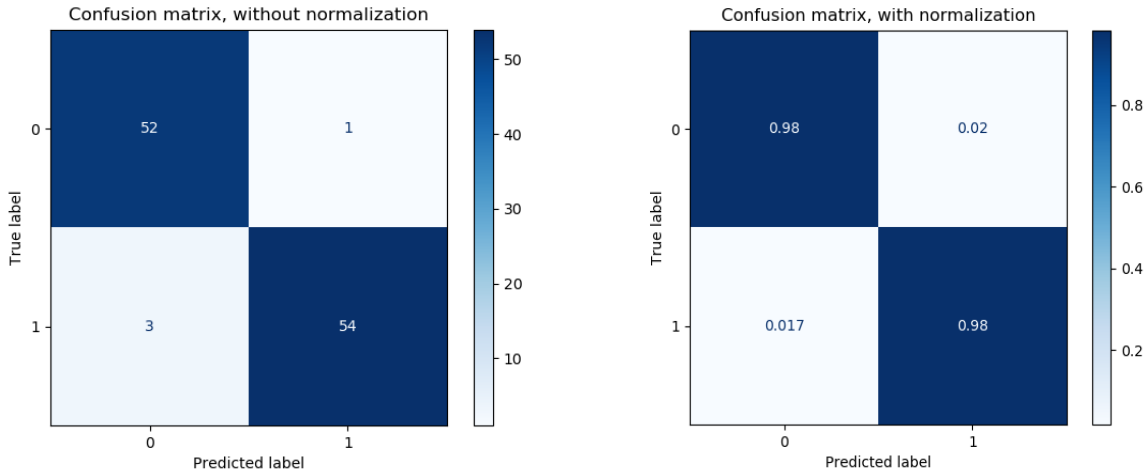


Figure: 10

Though we achieved a reasonable accuracy of about 96% for this model, we decided not to go with it for the following reasons:

i. This algorithm experiences a degraded performance with larger dataset
ii. It requires feature scaling in terms of standardization and normalization
iii. It has high sensitivity to noises, missing values and outliers and they must be manually removed.

Since these tasks were not so easily achievable with such a high-quality radiology image dataset we decided to move on.

B. SVM

MachineLearningProject/svm.ipynb at main · KraticaRastogi/MachineLearningProject (github.com)

Next, we tried to train our model using SVMs (also known as support-vector networks). To give a background, these are supervised learning models which analyze data used for classification and regression analysis. It constructs a hyperplane or its set in a high- or infinite-dimensional space, which is useful for classification, regression, or outlier detection. It helps to achieve a good separation by the hyperplane which has the highest distance to the nearest training-data point of any class, since in general with a large margin, the generalization error of the classifier gets lowered.[11]

We chose SVM as it works comparatively better when the margin of separation between classes is definite and also because it is memory efficient.

The below piece of code was used to train the SVM classifier with training images and their relevant labels.

```
svm_classifier = svm.SVC()
return svm_classifier.fit(train_images, train_labels)
```

Figure: 11

SVM results are:

```
accuracy score: 0.9818181818181818
precision score: 0.9818181818181818
recall score: 0.9818181818181818
f score: 0.9818181818181818
Confusion Matrix: [[54  1]
 [ 1 54]]
Confusion matrix, without normalization
[[54  1]
 [ 1 54]]
```

Figure: 12

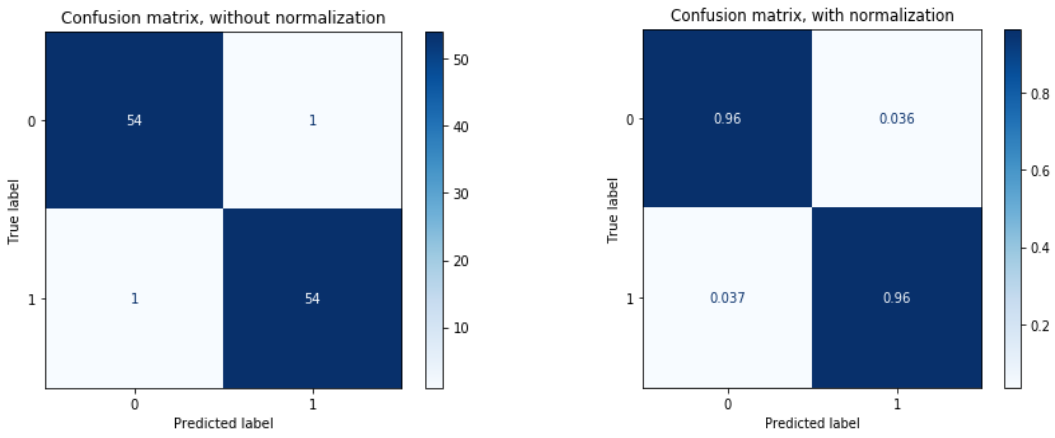


Figure: 13

There was not much improvement in the results of KNN and for this reason we trained model on other algorithms as well.

C. Random Forest

MachineLearningProject/random_forest.ipynb at main · KraticaRastogi/MachineLearningProject (github.com)

We further built a model with Random forest which is an ensemble learning method that constructs several decision trees at training time and outputs a class which is the mode of the classes (classification) of the individual trees. [12] The below code snippet was used to train our Random_Forest model:

```
def create_and_train_model():
    """
    This method will create and fit Random_Forest model and return the same.

    :return: knn
    """
    random_forest_classifier = RandomForestClassifier()
    return random_forest_classifier.fit(train_images, train_labels)
```

Figure: 14

Below is the snapshot of the results obtained for Random Forest model:

```
accuracy score: 0.9545454545454546
precision score: 0.9546660236315408
recall score: 0.9545454545454546
f score: 0.9545266327875024
Confusion Matrix: [[55  2]
 [ 3 50]]
Confusion matrix, with normalization
[[0.96491228 0.03508772]
 [0.05660377 0.94339623]]
```

Figure: 15

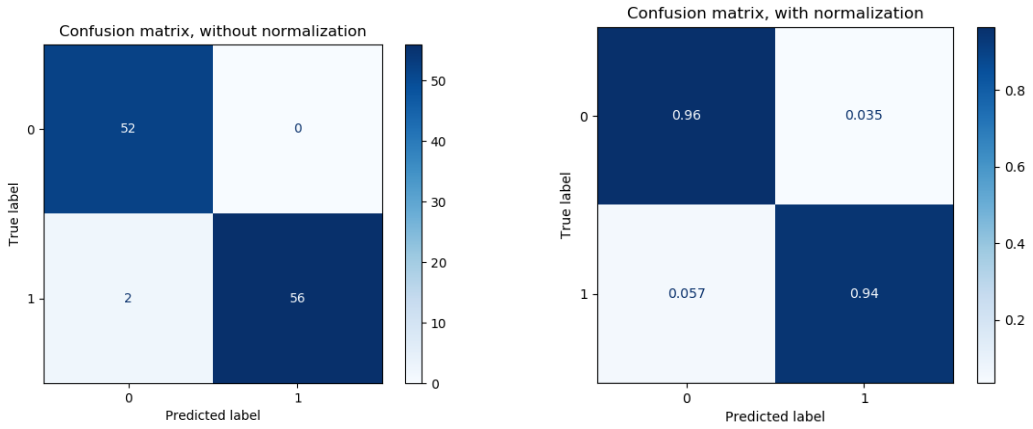
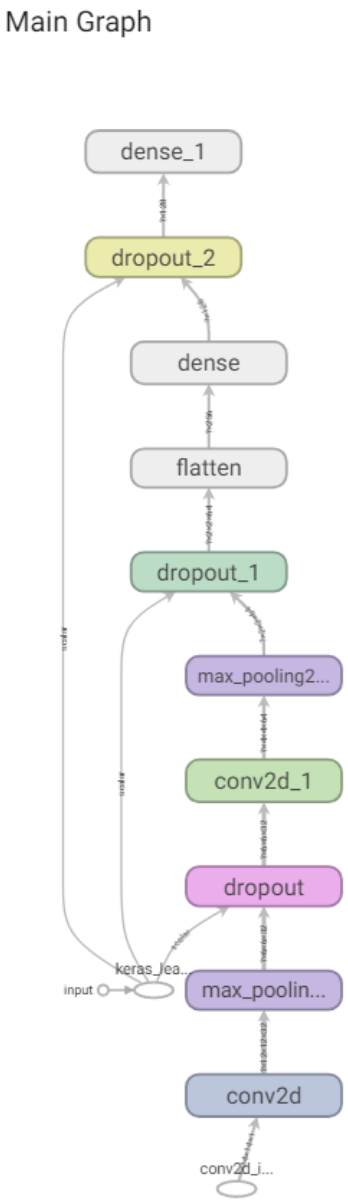


Figure: 16

D. Convoluted Neural Network(CNN)

Below are the layers main graph performed for CNN model. This graph we have taken from tensorboard.



We trained the 240 MB dataset first without applying PCA and then after applying PCA.

CNN without applying PCA – 240MB dataset

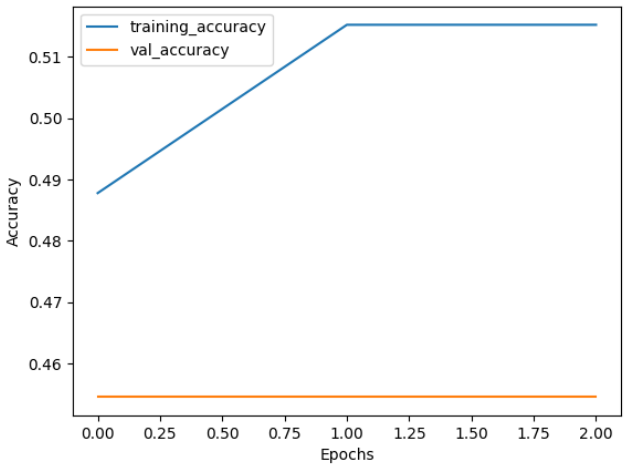


Figure: 17

```
Accuracy: 0.4545454680919647
Loss: 0.8587162494659424
```

Figure: 18

Results (Fig:21) with PCA on 240MB dataset and one conv2d layer

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(14, 14, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))
```

Figure: 19

```
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(2, activation='softmax'))
```

Figure: 20

```
Epoch 10/10
11/11 [==============================] - 0s 5ms/step - loss: 0.3363 - accuracy: 0.9787 - val_loss: 0.3748 - val_accuracy: 0.9273
4/4 - 0s - loss: 0.3748 - accuracy: 0.9273
Accuracy: 0.9272727370262146
Loss: 0.37482044100761414
```

Figure: 21

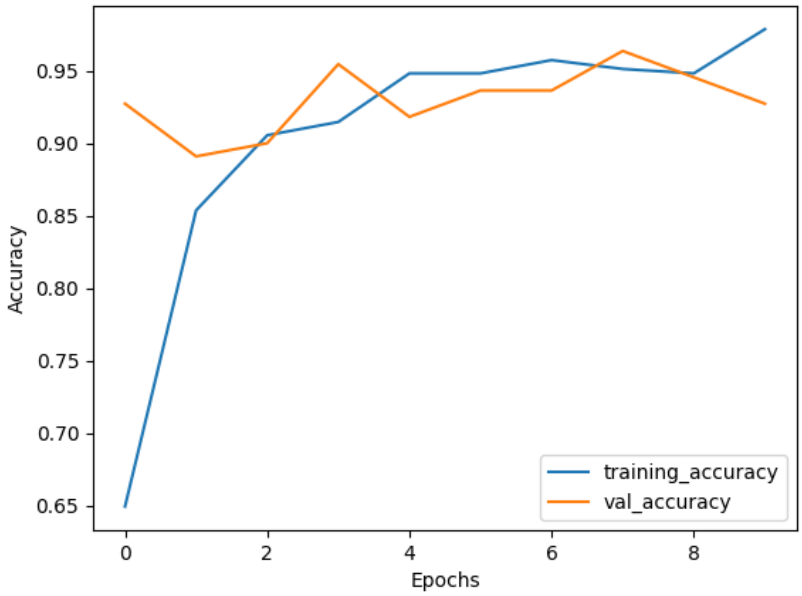


Figure: 22

We tried increasing Conv2d layers with epochs = 10

```python
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(14, 14, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(2, activation='softmax'))
```

Figure: 23

```
Accuracy: 0.8545454740524292
Loss: 0.45253053307533264
```

Figure: 24

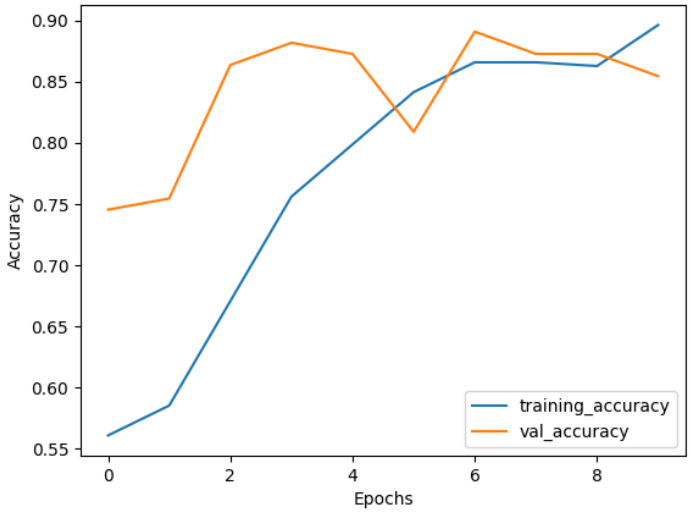


Figure: 25

Next, we tried increasing Epochs from 10 to 100 and accuracy was still approximately the same

```
Test Accuracy: 0.77272725
Test Loss: 1.7703138806603171
```

Figure: 26

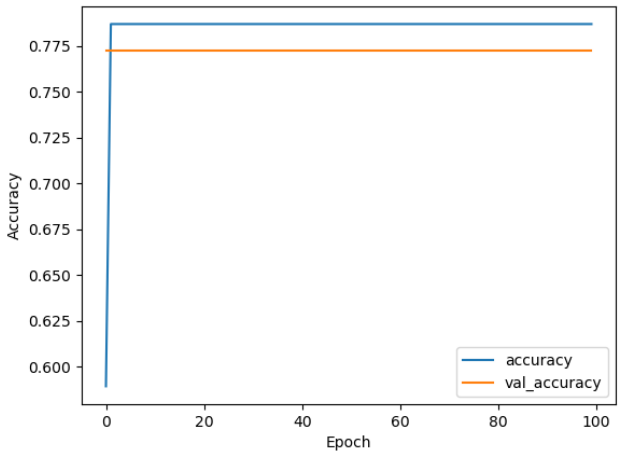


Figure: 27

Next, we tried with the 240 MB data to train a CNN model with PCA activation function from relu to sigmoid and got accuracy as ~77%

CNN model without applying PCA – 1GB data

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(1024,
1024, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))
```

Figure: 28

Resultant graph for CNN model without PCA:

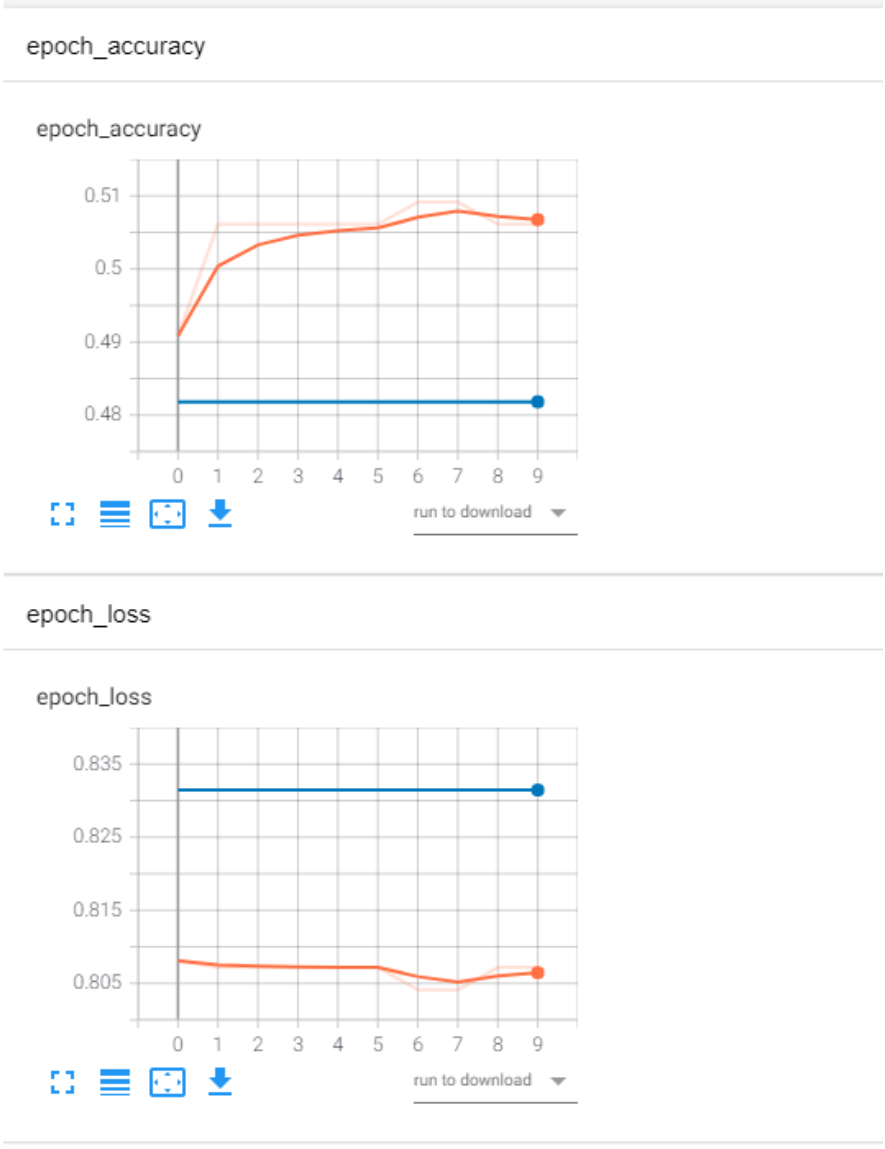


Figure: 29

CNN model - After applying PCA – 1GB dataset

```
max_pooling2d (MaxPooling2D) (None, 6, 6, 32)          0
_________________________________________________________________
dropout (Dropout)            (None, 6, 6, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dense (Dense)                (None, 128)               147584
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 148,162
Trainable params: 148,162
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
11/11 [==============================] - 0s 19ms/step - loss: 0.7098 - accuracy: 0.5945 - val_loss: 0.3901 - val_accuracy: 0.9182
Epoch 2/10
11/11 [==============================] - 0s 5ms/step - loss: 0.5006 - accuracy: 0.8140 - val_loss: 0.3421 - val_accuracy: 0.9727
Epoch 3/10
11/11 [==============================] - 0s 5ms/step - loss: 0.4472 - accuracy: 0.8567 - val_loss: 0.3517 - val_accuracy: 0.9636
Epoch 4/10
11/11 [==============================] - 0s 5ms/step - loss: 0.4198 - accuracy: 0.8841 - val_loss: 0.3325 - val_accuracy: 0.9909
Epoch 5/10
11/11 [==============================] - 0s 5ms/step - loss: 0.3812 - accuracy: 0.9299 - val_loss: 0.3301 - val_accuracy: 0.9909
Epoch 6/10
11/11 [==============================] - 0s 5ms/step - loss: 0.3926 - accuracy: 0.9207 - val_loss: 0.3300 - val_accuracy: 0.9818
Epoch 7/10
11/11 [==============================] - 0s 5ms/step - loss: 0.3644 - accuracy: 0.9451 - val_loss: 0.3582 - val_accuracy: 0.9455
Epoch 8/10
11/11 [==============================] - 0s 5ms/step - loss: 0.3768 - accuracy: 0.9329 - val_loss: 0.3290 - val_accuracy: 0.9909
Epoch 9/10
11/11 [==============================] - 0s 5ms/step - loss: 0.3962 - accuracy: 0.9116 - val_loss: 0.3382 - val_accuracy: 0.9727
Epoch 10/10
11/11 [==============================] - 0s 5ms/step - loss: 0.3783 - accuracy: 0.9329 - val_loss: 0.3294 - val_accuracy: 0.9818
4/4 - 0s - loss: 0.3294 - accuracy: 0.9818
Test Loss: 0.32944852113723755
Test Accuracy: 0.9818181991577148
```

Figure: 30

Resultant graph for CNN model with PCA:

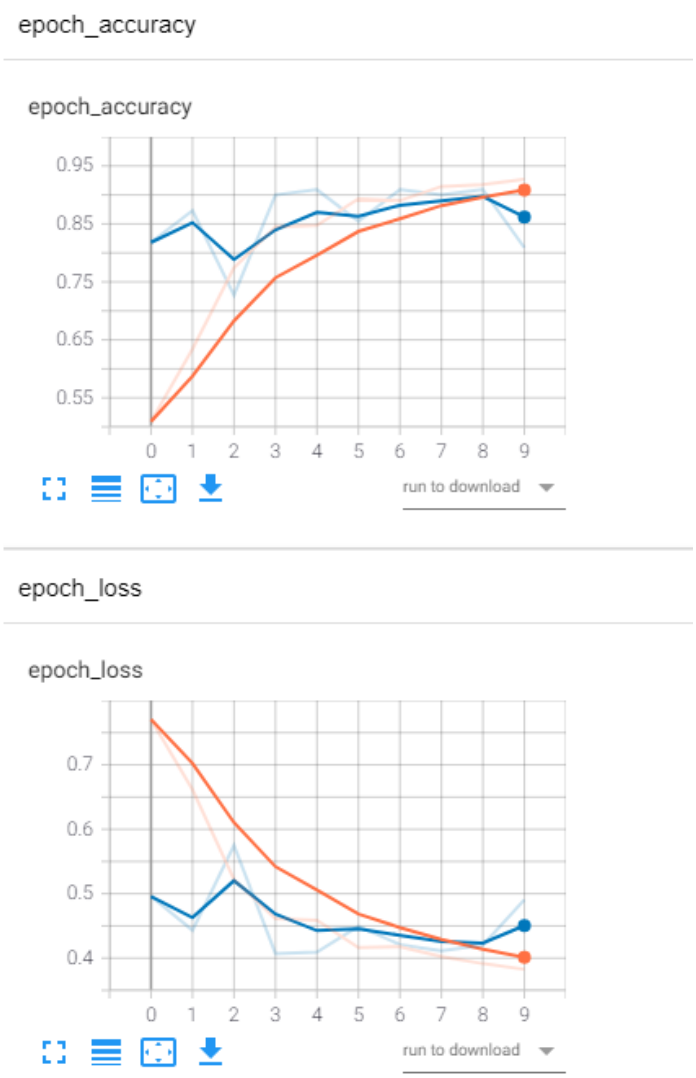


Figure: 31

Comparison of all algorithm's results for 1GB dataset:

Model	Accuracy(%)	Precision(%)	Recall(%)	F1 Score(%)
KNN	96.36	96.42	96.36	96.36
SVM	98.18	98.18	98.18	98.18
Random Forest	95.45	95.47	95.45	95.45
CNN(after PCA)	98.18	98.76	98.19	98.16

5. Conclusion and Future Work

The main conclusion of our work is that we got high accuracy for both SVM and CNN model i.e approximately 98%

Since we all know that CNN model requires high computation on each layer. So, if large dataset is used for training the model using CNN, then the size of CPU and GPU should also be high.

Therefore, according to our preference (from fastest to slowest):-

SVM > Random Forest > KNN > CNN

Below is the future work which we have planned so far:

- Adding more features to our model that would predict the risk factor of the infected patient. It means how badly the patient is infected with COVID.
- We have also envisioned the addition of an application, wherein uploading the Xray. This would simply classify the image and predict the result.

References

[1] T. B. Chandra and K. Verma, "Pneumonia Detection on Chest X-Ray Using Machine Learning Paradigm," in *Proceedings of 3rd International Conference on Computer Vision and Image Processing*, Singapore, 2020, pp. 21–33, doi: 10.1007/978-981-32-9088-4_3.

[2] T. B. Chandra, K. Verma, B. K. Singh, D. Jain, and S. S. Netam, "Coronavirus disease (COVID-19) detection in Chest X-Ray images using majority voting based classifier ensemble," *Expert Syst. Appl.*, vol. 165, p. 113909, Mar. 2021, doi: 10.1016/j.eswa.2020.113909.

[3] L. Wang and A. Wong, "COVID-Net: A Tailored Deep Convolutional Neural Network Design for Detection of COVID-19 Cases from Chest X-Ray Images," *ArXiv200309871 Cs Eess*, May 2020, Accessed: Dec. 16, 2020. [Online]. Available: http://arxiv.org/abs/2003.09871.

[4] "OpenCV Functions | OpenCV For Computer Vision," *Analytics Vidhya*, Mar. 25, 2019. https://www.analyticsvidhya.com/blog/2019/03/opencv-functions-computer-vision-python/ (accessed Dec. 16, 2020).

[5] "OpenCV Alternatives & Competitors," *G2*. https://www.g2.com/products/opencv/competitors/alternatives (accessed Dec. 16, 2020).

[6] "A Step-by-Step Explanation of Principal Component Analysis," *Built In*. https://builtin.com/data-science/step-step-explanation-principal-component-analysis (accessed Dec. 16, 2020).

[7] P. V. V, "Part-2: Error Analysis — The Wild West. Algorithms to Improve #NeuralNetwork Accuracy.," *Medium*, Aug. 22, 2016. https://medium.com/autonomous-agents/part-2-error-analysis-the-wild-west-algorithms-to-improve-neuralnetwork-accuracy-6121569e66a5 (accessed Dec. 16, 2020).

[8] W. Koehrsen, "Beyond Accuracy: Precision and Recall," *Medium*, Mar. 10, 2018. https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c (accessed Dec. 16, 2020).

[9] O. Media, "The normalized confusion matrix - Machine Learning with scikit-learn Quick Start Guide." https://learning.oreilly.com/library/view/machine-learning-with/9781789343700/23007d97-55d3-4427-b5a4-88a0322320e4.xhtml (accessed Dec. 16, 2020).

[10] N. Kumar, “The Professionals Point: Advantages and Disadvantages of KNN Algorithm in Machine Learning,” *The Professionals Point*, Feb. 23, 2019. http://theprofessionalspoint.blogspot.com/2019/02/advantages-and-disadvantages-of-knn.html (accessed Dec. 16, 2020).
[11] “Support Vector Machine — Introduction to Machine Learning Algorithms | by Rohith Gandhi | Towards Data Science.” https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (accessed Dec. 16, 2020).
[12] “Random forest,” *Wikipedia*. Dec. 15, 2020, Accessed: Dec. 17, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Random_forest&oldid=994316096.