Capstone Thesis URL

http://ece.northsouth.edu/capstone-design/covid-19-classification-system-using-deep-convolutional-neural-networks-dcnn/

Scroll down to read the full report.

Department of Electrical and Computer Engineering
North South University

Thesis Report

Covid-19 Classification System using Deep Convolutional Neural Networks (DCNN)

Salman Zaman Chowdhury **ID # 1612210042**

Arefa Patwary **ID # 1620265042**

Faculty Advisor:

Syed Athar Bin Amir

Lecturer

ECE Department

Spring, 2020

DECLARATION

This is to certify that this Thesis Project is our original work. We haven't submitted our project anywhere else either partially or fully for the award of any other course or degree. Any material taken from elsewhere in this project has been properly acknowledged.

Students' name & Signature

1) **Arefa Patwary**

2) **Salman Zaman Chowdhury**

ACKNOWLEDGMENT

First of all, we would like to express our gratitude to the Almighty for giving us the strength and opportunity to perform our responsibilities and complete the report.

The project is a to bridge the gap between the theoretical knowledge and real-life experience. This report will give a practical experience through the theoretical understanding.

We would also like to thank our teacher and mentor Syed Bin Athar sir who has constantly helped throughout the semester by providing us the technical knowledge and moral support to complete the project with full understanding.

We thank our friends and family for their technical and moral support to finish this project.

APPROVAL

The capstone thesis project entitled “Covid-19 classification system using deep learning” by Arefa Patwary (ID#1620165042) and Salman Zaman Chowdhury (ID#1612210042) is approved in partial fulfillment of the requirement of the Degree of Bachelor of Science in Computer Science and Engineering on October and has been accepted as satisfactory.

Supervisor’s Signature

Syed Athar Bin Amir

Lecturer

Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh.

ABSTRACT

The effect of Covid-19 pandemic on the health and well-being of the global population was devastating and is still continuing. An important step to fight this COVID-19 is to implement a successful and less time-consuming screening process of contaminated patients, and one of the vital screening processes is radiological imaging using chest radiography. The main goal of this project was to automatically detect COVID-19 patients using digital chest x- ray images while maximizing the accuracy in detection using deep convolutional neural networks (DCNN). The dataset consists 8723 chest x-ray images. In this project, DCNN based model VGG-19, Resnet-50 and a custom CNN model with transfer learning have been proposed for the detection of coronavirus infected patients using chest X-ray radiographs and gives a classification accuracy of more than 97% training accuracy. All images were of the same size and stored in JPEG and PNG. The average sensitivity, specificity, and accuracy of the lung classification using the proposed models' has shown very good results. The results demonstrate that transfer learning showed robust performance and finally because of the smaller size of our custom model it is easily deployable for COVID-19 detection.

Table of Contents

CHAPTER: 01 10
OVERVIEW 10
1.1 Introduction 10
1.2 Project Details 12
1.3 Summary 48
CHAPTER: 02 49
MOTIVATION 49
2.1 Introduction 49
2.2 Motivation towards our project 49
2.3 Summary 49
CHAPTER: 03 50
RELATED WORKS 50
3.1 Introduction 50
3.2 Systems related to our project 50
3.3 Drawbacks 56
3.4 Proposed Solution 57
3.5 Summary 58
CHAPTER: 04 59
SKILLS 59
4.1 Introduction 59
4.2 Skills Obtained 59
4.3 Summary 62
CHAPTER: 05 63
DATASETS 63
5.1 Introduction 63
5.2 Collecting data from real world 63
5.3 Collecting data from the internet 63
5.4 Sample datasets 64
5.5 Summary 67
CHAPTER: 06 68
TENSORFLOW 68
6.1 Introduction 68

6.2 What is TensorFlow? 68
6.3 How TensorFlow works? 68
6.4 How will we use TensorFlow? 69
6.5 Summary 69
CHAPTER: 07 70
DEEP LEARNING 70
7.1 Introduction 70
7.2 What is Deep Learning 70
7.3 How does Deep Learning Work 70
7.4 Applications: 72
7.5 Summary 72
CHAPTER 08 73
KERAS 73
8.1 What is Keras? 73
8.2 How do we use it? 73
8.3 Summary 74
CHAPTER: 09 75
Deploying the model 75
9.1 Introduction 75
9.2 What is FLASK? 75
9.3 Deploying Covidnet-8: 75
9.4 Summary 79
CHAPTER: 10 80
Project code 80
10.1 Introduction 80
10.2 Project code 80
10.3 Summary 88
CHAPTER: 11 89
FUTURE WORK 89
CHAPTER 12 90
PRACTICALITY 90
CHAPTER 12 91
CONCLUSION 91

BIBLIOGRAPHY 92

CHAPTER: 01

OVERVIEW

1.1 Introduction

The 2019-nCov is a novel emerging coronavirus that was first discovered at a local wildlife market in December 2019 in Wuhan, China. In early 2020, it was titled as a highly contagious virus causing a global pandemic since it has infected millions worldwide. Although infected people exhibit noticeable symptoms of COVID-19, it is also likely that there can be asymptomatic cases. This is one of the very reasons why detection techniques of SARS-CoV-2 should be highly critical in the containment of disease transmission.

There are a lot of different methods to diagnose the virus and scientists are still working on finding new ways every day. Researchers have been utilizing Artificial Intelligence (AI) to follow the flare-up continuously. AI makes strong relationships between different variables that a human brain is not advanced enough of making. It can link datasets that would not be possible to bind by human beings. AI has been implemented in data analytics in order to determine the infection state of coronavirus by observing the symptoms of Covid-19, from images of CT scans of lungs, by monitoring changes in body temperature with the help of wearable sensors and so on. AI systems can also analyze X-rays to help the radiologists easily predict which patients are affected and most likely to need a ventilator.

Diagnosis of COVID-19 generally depends on both the symptoms of pneumonia and Chest X-ray tests. Chest X-ray is the first imaging technique that is used to diagnose COVID-19 disease. Most of COVID-19 cases have comparable highlights on radiographic pictures including reciprocal, multi-focal, ground-glass opacities with a fringe or back dissemination, primarily in the lower projections, in the early stage and pulmonary consolidation in the late stage. Although typical Chest X-Ray images are used for early screening of suspected cases, the pictures of different viral cases of pneumonia are very similar to the CXRs of Covid-19 and other infectious and inflammatory lung diseases so they tend to they overlap with each other.

Therefore, it is hard for radiologists to recognize COVID-19 from other viral pneumonia and respiratory diseases. The side effects of COVID-19 CXRs and symptoms being like viral pneumonia can sometimes lead to confusion and wrong determination, especially now because of the overwhelming number of patients, which leads to the medical clinics being over-burden and working nonstop. Therefore, wrong result can result in wrong treatment and wrong medicine with resulting costs, exertion and danger of presentation to positive COVID-19 patients. Deep learning techniques can reveal image features, which are not apparent in the original images. In particular, Convolutional Neural Network (CNN) has been demonstrated amazing results in extracting features and learning and therefore widely adopted by the research community.

On the other hand, especially from our country's perspective, the test of COVID-19 is currently a difficult task because of inaccessibility of diagnosis system everywhere, which is causing panic and because of the limited availability of COVID-19 testing kits, we have to depend on different determination measures. Since COVID-19 damages the epithelial cells that line our respiratory tract, we can utilize X-rays to investigate the strength of a patient's lungs. And almost in all hospitals have X-ray imaging machines, it could be possible to use X-ray's to test for COVID-19 without the dedicated test kits. But X-ray examination requires a radiologist and it is time consuming, which is valuable when people are sick. Therefore, developing an automated analysis system using AI is essential to save medical professionals' valuable time.

Several classical machine learning approaches have been previously used for automatic classification of digitized chest images. The proposed work is implemented with customized pretrained models VGG-19 and Resnet-50 and finally the project has been implemented on a custom CNN model that was trained to classify normal, pneumonia and COVID-19 Chest X-ray images and tested on and obtained more than 97% of classification accuracy. The proposed model is developed to provide accurate diagnostics for multi-class classification (COVID vs. Normal vs. Pneumonia. The custom CNN model was used in our study as a classifier. We implemented 6 convolutional layers and introduced different filtering on each layer. The model can be employed to assist radiologists in validating their initial screening.

1.2 Project Details

The Project is segmented into few different parts:

Dataset:

The dataset used in this project is a combination of multiple datasets and these datasets were uploaded on Kaggle and on various other websites as open source datasets. All of these data have been resized into fixed shape 256 x 256 pixels. There are about 8723 images in that dataset belonging to 3 different classes. These data then, have been augmented.

Sample data:

Covid-19:

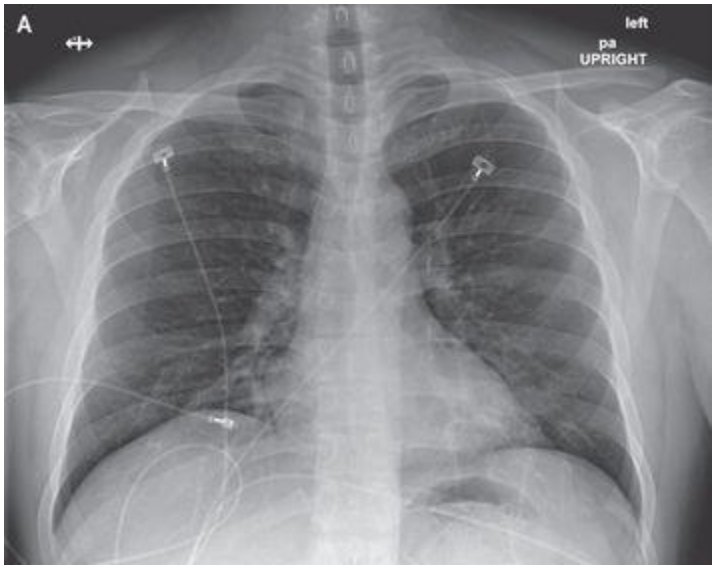


Fig: Covid-19 chest X-ray image

Normal:

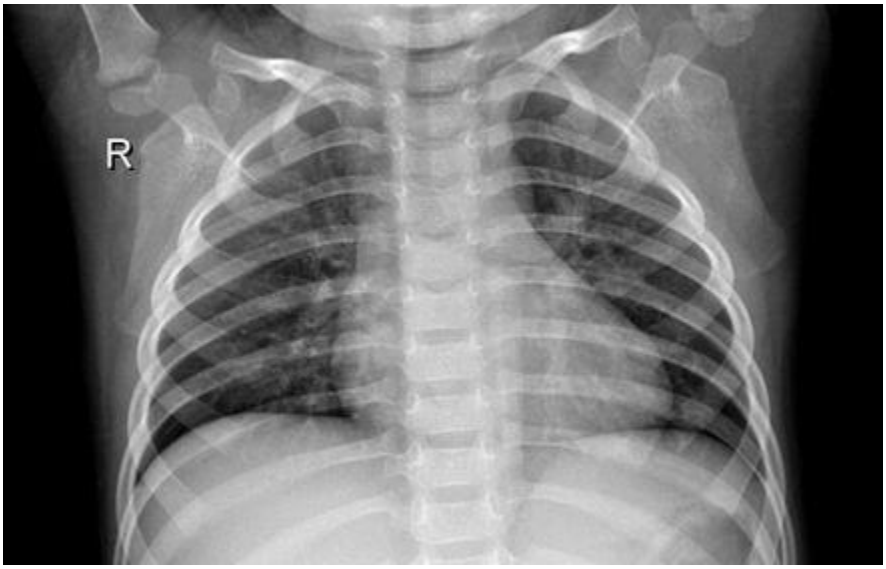


Fig: Normal chest X-ray image

Pneumonia:

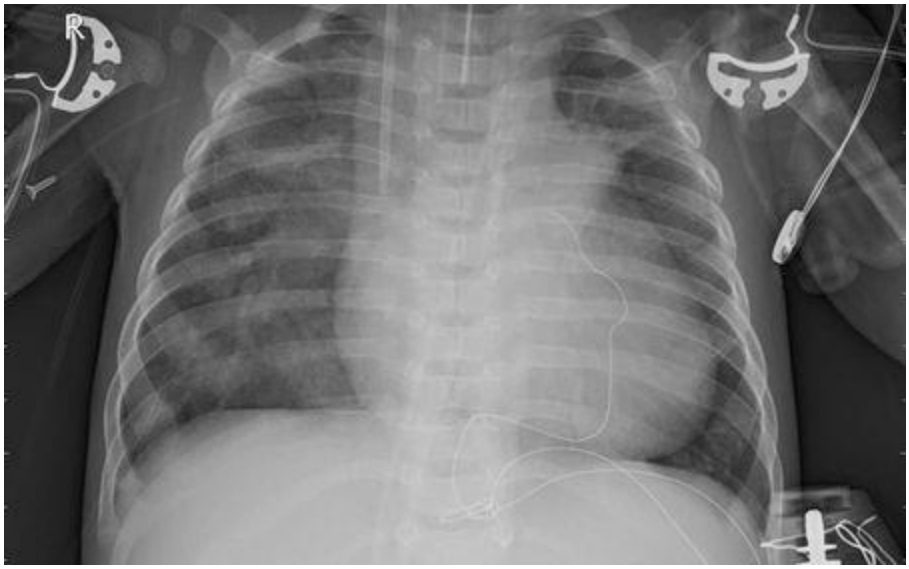


Fig: Pneumonia chest X-ray image

Building the customized CNN model Covidnet-8:

It is a Sequential model consisting of six 2D convolutional layers and two dense layers. The activation function used is Relu for all the 2D conv layers and dense layer except for the last dense layer where Softmax has been used as an activation function. The learning rate optimizer used in the model is ADAM with a learning rate of lr = 0.001. The loss used is categorical_crossentropy. Finally, batch normalization and a dropout of 0.3 has been applied on it.

Model Background:

Convolutional Neural Network (CNN) is a neural network which extracts or identifies a feature in a particular image forming one of the most fundamental operations in Machine Learning and is widely used as an independent model or as a base model in majority of Neural Networks like GoogleNet, VGG19 and others for various tasks such as Object Detection, Image Classification and others.

CNN has the following five basic components:

- **Convolution**: to detect features in an image.

  Convolution is the fundamental mathematical operation that is highly useful to detect features of an image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs: Image matrix and a filter.

  The convolution operation takes place as follows:

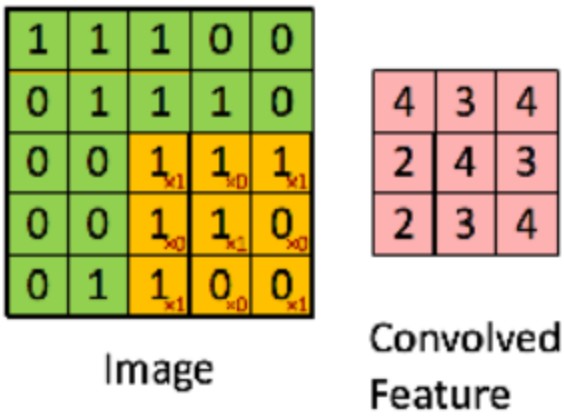


Fig: Convolutional operation using a 3x3 filter

- **ReLU**: to make the image smooth and make boundaries distinct

  An image, in general, is highly non-linear that is it has varied pixel values, especially if it has many features to be detected. In order to decrease the non-linearity in the convolution layer, the rectifier function is added, so that in case if there are negative pixel values, they will be replaced by zeroes. It helps greatly in feature detection because: 1) it helps in breaking the flow of the gradient in an image and 2) it brings a sharp or drastic change when there is a different feature.

- **Pooling**: to help fix distorted images.

  Pooling ensures that even if the features are a little distorted, or not very similar in different images, it still has the flexibility to identify the feature. In max pooling, a certain part of the feature map is taken, which is obtained in the previous step and report the highest or maximum value in that portion. In doing so, the feature of the image is preserved but at the same time get rid of more unimportant information.

  As a result of max pooling, the feature would have been preserved even if the picture were a little tilted because the largest number in a certain region of the feature is recorded. Moreover, the size is being reduced by a very significant amount which will make the computation much easier.

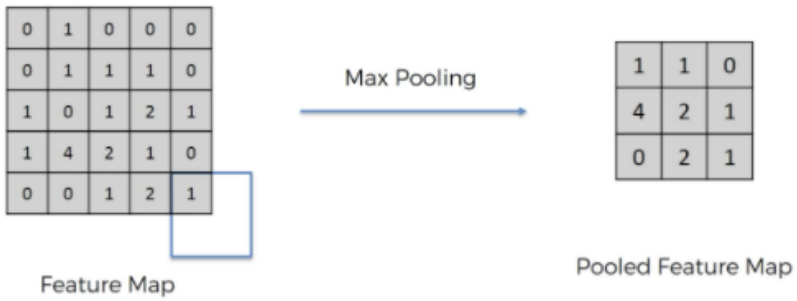


Fig: Pooled feature map

- **Flattening**: to turn the image into a suitable representation.

  In the flattening procedure, the elements in a pooled feature map is taken and put in a vector form. This becomes the input layer for the upcoming ANN.

- **Full connection**: to process the data in a neural network.

  Once the image is convolved, max pooled and flattened, the result is a vector. This vector acts as the input layer for an ANN which then is used to detect the image. It assigns random weights to each synapse; the input layer is weight adjusted and put into an activation function. The output of this is then compared to the true values and the error generated is back-propagated, i.e. the weights are re-adjusted and all the processes repeated. This is done until the error or cost function is minimized. Thus, the basic difference between a CNN and an ANN is only the preprocessing stage. After the flattening stage when the image is converted into a vector, all the subsequent steps resemble those of the ANNs. Unlike ANN where the input is a vector, input in a CNN is a multi-channeled image.

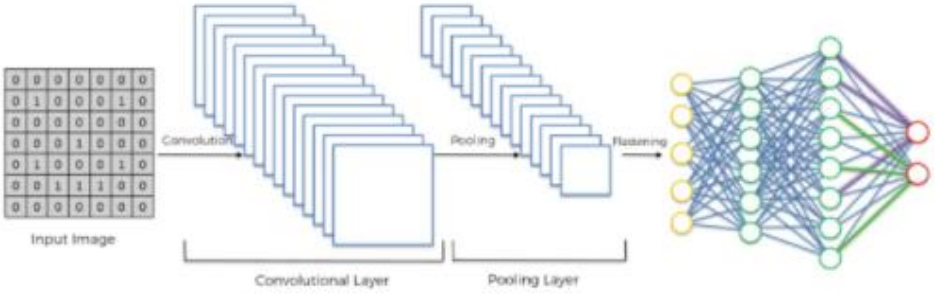


Fig: Fully connected layer

Covidnet-8 model structure:

```
model = Sequential()
model.add(Conv2D(64, (5, 5), input_shape=(256, 256,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D((3, 3)))

model.add(Conv2D(128, (4, 4)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(512, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(512, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(512, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(512, (2, 2)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(512, kernel_regularizer=regularizers.l2(0.02)))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(3))
model.add(Activation('softmax'))
```

The summary output of the model before training is:

```
[ ] Model: "sequential_1"
    _________________________________________________________________
    Layer (type)                 Output Shape              Param #
    =================================================================
    conv2d_1 (Conv2D)            (None, 252, 252, 64)      4864
    _________________________________________________________________
    activation_1 (Activation)    (None, 252, 252, 64)      0
    _________________________________________________________________
    max_pooling2d_1 (MaxPooling2 (None, 84, 84, 64)        0
    _________________________________________________________________
    conv2d_2 (Conv2D)            (None, 81, 81, 128)       131200
    _________________________________________________________________
    activation_2 (Activation)    (None, 81, 81, 128)       0
    _________________________________________________________________
    max_pooling2d_2 (MaxPooling2 (None, 40, 40, 128)       0
    _________________________________________________________________
    conv2d_3 (Conv2D)            (None, 38, 38, 512)       590336
    _________________________________________________________________
    activation_3 (Activation)    (None, 38, 38, 512)       0
    _________________________________________________________________
    max_pooling2d_3 (MaxPooling2 (None, 19, 19, 512)       0
    _________________________________________________________________
    conv2d_4 (Conv2D)            (None, 17, 17, 512)       2359808
    _________________________________________________________________
    activation_4 (Activation)    (None, 17, 17, 512)       0
    _________________________________________________________________
    max_pooling2d_4 (MaxPooling2 (None, 8, 8, 512)         0
    _________________________________________________________________
    conv2d_5 (Conv2D)            (None, 6, 6, 512)         2359808
    _________________________________________________________________
    activation_5 (Activation)    (None, 6, 6, 512)         0
    _________________________________________________________________
    max_pooling2d_5 (MaxPooling2 (None, 3, 3, 512)         0
    _________________________________________________________________
    conv2d_6 (Conv2D)            (None, 2, 2, 512)         1049088
    _________________________________________________________________
    activation_6 (Activation)    (None, 2, 2, 512)         0
    _________________________________________________________________
    max_pooling2d_6 (MaxPooling2 (None, 1, 1, 512)         0
    _________________________________________________________________
    flatten_1 (Flatten)          (None, 512)               0
    _________________________________________________________________
    dense_1 (Dense)              (None, 512)               262656
    _________________________________________________________________
    activation_7 (Activation)    (None, 512)               0
    _________________________________________________________________
    dropout_1 (Dropout)          (None, 512)               0
    _________________________________________________________________
    dense_2 (Dense)              (None, 3)                 1539
    _________________________________________________________________
    activation_8 (Activation)    (None, 3)                 0
    =================================================================
    Total params: 6,759,299
    Trainable params: 6,759,299
    Non-trainable params: 0
    _________________________________________________________________
```

Covidnet-8 model architecture:

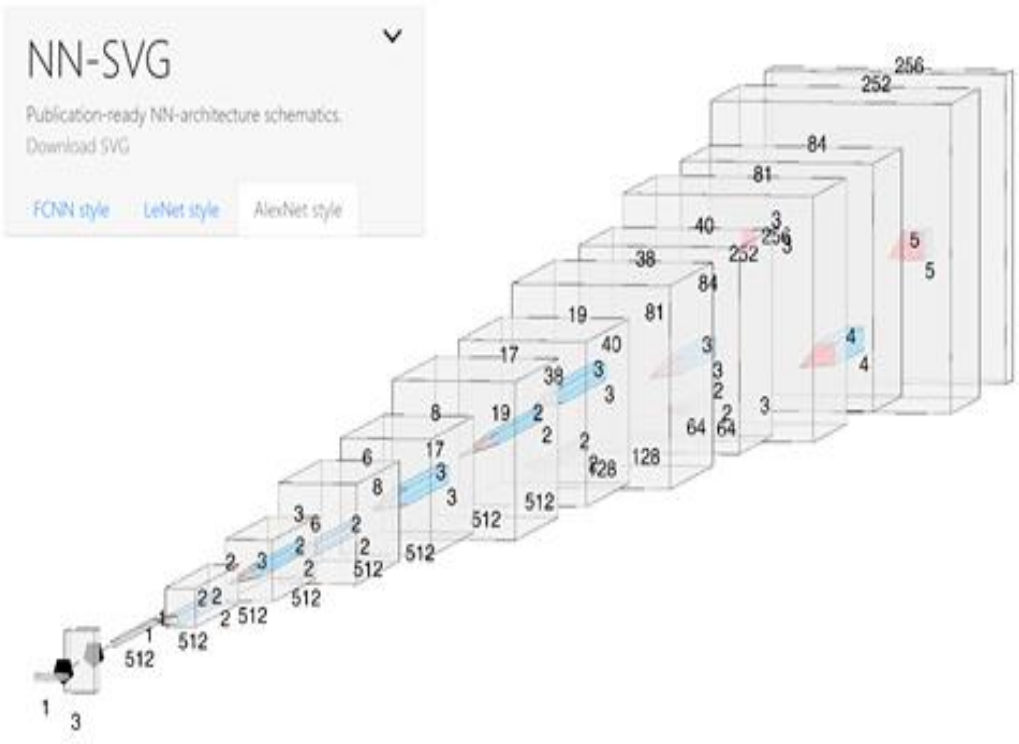


Fig: Covidnet-8 architecture

Loss Function: We have used the 'Categorical Crossentropy' loss function.

$$CCE(p,t) = -\sum_{c=1}^{C} t_{o,c} \log\left(p_{o,c}\right)$$

Preprocessing:

Preprocessing refers to all the transformations on the raw data before it is fed to the machine learning or deep learning algorithm. For instance, training a convolutional neural network on raw images will probably lead to bad classification performances. The preprocessing is also important to speed up training. The only preprocessing that was done is that they subtracted the mean RGB value from each pixel, computed over the whole training set. So, all the data in the dataset have been preprocessed before feeding it to the model for training.

Data augmentation:

Data augmentation has been applied on the dataset in order to increase the data size.

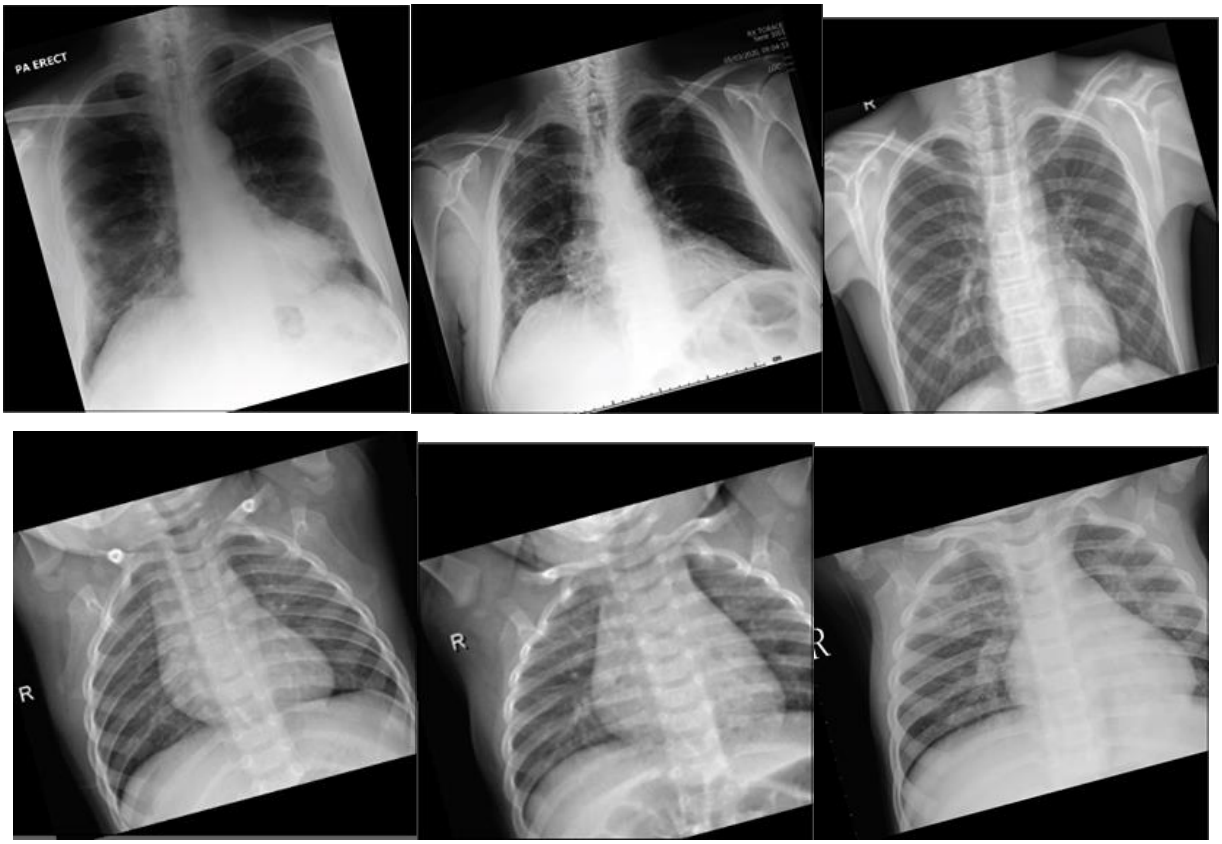


Fig:Augmented data

Training:

There are total 6900 training images in the dataset. A batch size of 32 was used and data augmentation, by rotating the images was used to further increase the number of samples.

Training a Neural Network Model means, making the model understand the relation between the input data over and over again so that it can successfully predict the outcome when a foreign data of the same kind is provided to it.

Now two things are essential for training any Neural Network model.

1) A large Dataset.
2) High Computational Power.

We have discussed about the chapter in an earlier chapter. After setting up the environment we have split the dataset into two categories. Train and Test. Usually 70 – 80 percent of the dataset is set for using in training the model, and 20 – 30 percent of the dataset is used for testing the model.

To train the model, we need to give it inputs from the dataset, and compare its outputs with the outputs from the data set. Since the model is still untrained, its outputs will be wrong. Once we go through the entire dataset, we can create a function that will show us how wrong the model's outputs were from the real outputs. This function is called the Cost Function. Our ultimate goal is to reduce this cost function. To minimize the cost function, we have to iterate through the dataset many times. This is why a large amount of computational power is needed. Once model is finished training, we will move on to the testing phase.

```
Epoch 1/20
 - 4051s - loss: 6.0192 - accuracy: 0.6751 - val_loss: 2.5296 - val_accuracy: 0.8117
Epoch 2/20
 - 261s - loss: 2.3061 - accuracy: 0.8162 - val_loss: 1.5620 - val_accuracy: 0.8660
Epoch 3/20
 - 260s - loss: 1.4900 - accuracy: 0.8883 - val_loss: 1.2278 - val_accuracy: 0.9154
Epoch 4/20
 - 262s - loss: 1.0907 - accuracy: 0.9151 - val_loss: 0.7868 - val_accuracy: 0.8834
Epoch 5/20
 - 261s - loss: 0.8682 - accuracy: 0.9159 - val_loss: 0.6185 - val_accuracy: 0.9428
Epoch 6/20
 - 259s - loss: 0.6899 - accuracy: 0.9283 - val_loss: 0.9413 - val_accuracy: 0.9311
Epoch 7/20
 - 264s - loss: 0.5361 - accuracy: 0.9483 - val_loss: 0.3405 - val_accuracy: 0.9580
Epoch 8/20
 - 260s - loss: 0.4631 - accuracy: 0.9443 - val_loss: 0.8231 - val_accuracy: 0.9456
Epoch 9/20
 - 260s - loss: 0.3918 - accuracy: 0.9462 - val_loss: 0.2074 - val_accuracy: 0.9669
Epoch 10/20
 - 259s - loss: 0.3482 - accuracy: 0.9457 - val_loss: 0.1837 - val_accuracy: 0.9652
Epoch 11/20
 - 259s - loss: 0.2738 - accuracy: 0.9568 - val_loss: 0.2168 - val_accuracy: 0.9725
Epoch 12/20
 - 259s - loss: 0.2286 - accuracy: 0.9583 - val_loss: 0.5136 - val_accuracy: 0.9787
Epoch 13/20
 - 261s - loss: 0.2077 - accuracy: 0.9581 - val_loss: 0.0874 - val_accuracy: 0.9748
Epoch 14/20
 - 259s - loss: 0.1782 - accuracy: 0.9639 - val_loss: 0.0568 - val_accuracy: 0.9815
Epoch 15/20
 - 258s - loss: 0.1724 - accuracy: 0.9604 - val_loss: 0.0472 - val_accuracy: 0.9815
Epoch 16/20
 - 258s - loss: 0.1634 - accuracy: 0.9639 - val_loss: 0.0492 - val_accuracy: 0.9753
Epoch 17/20
 - 256s - loss: 0.1344 - accuracy: 0.9686 - val_loss: 0.5410 - val_accuracy: 0.9798
Epoch 18/20
 - 257s - loss: 0.1346 - accuracy: 0.9665 - val_loss: 0.1890 - val_accuracy: 0.9703
Epoch 19/20
 - 257s - loss: 0.1244 - accuracy: 0.9696 - val_loss: 0.0615 - val_accuracy: 0.9742
Epoch 20/20
 - 256s - loss: 0.1135 - accuracy: 0.9716 - val_loss: 0.0216 - val_accuracy: 0.9793
```

Fig: Model training with 20 epochs

Validating:

There are total 1784 validation images in the dataset. The batch size for the validation set was set to 16.

Testing:

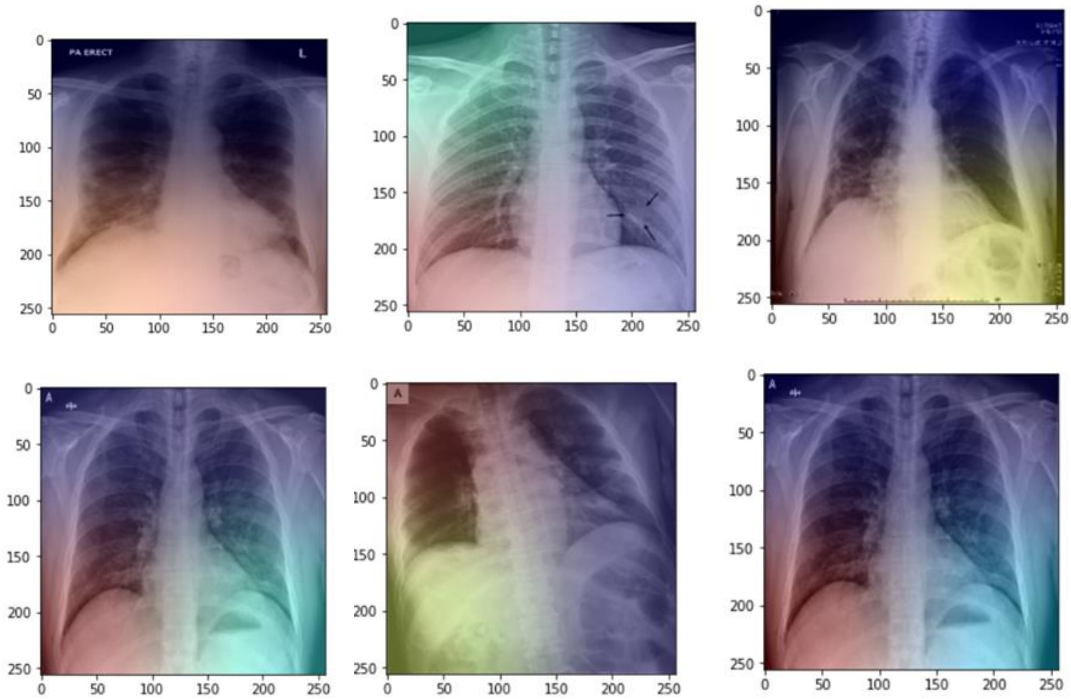
There are total 39 testing images in the dataset. The test set had a good balance of the three classes.

After training the model comes the testing phase. In the testing phase we will use the portion of the dataset that was preserved for testing and analyze the results.

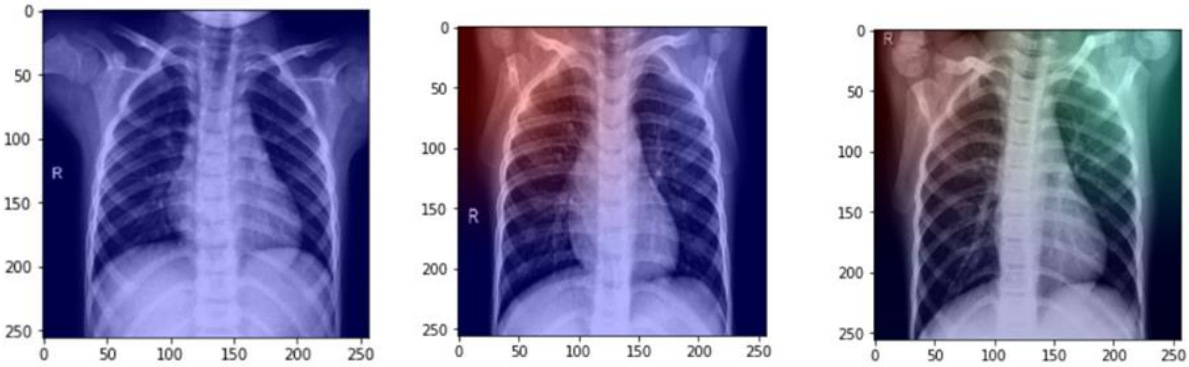
Through testing our model, we are able verify the accuracy of our model and calculate the loss function as well.

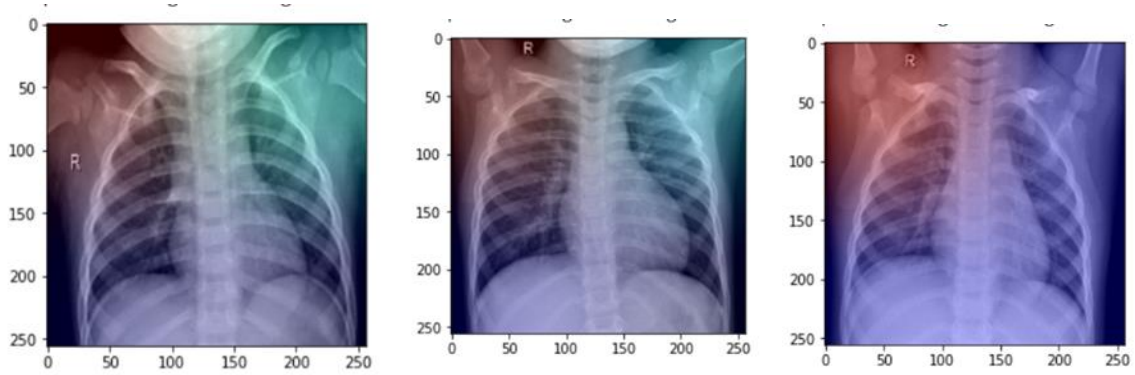
Heat maps:

1. Covid-19:

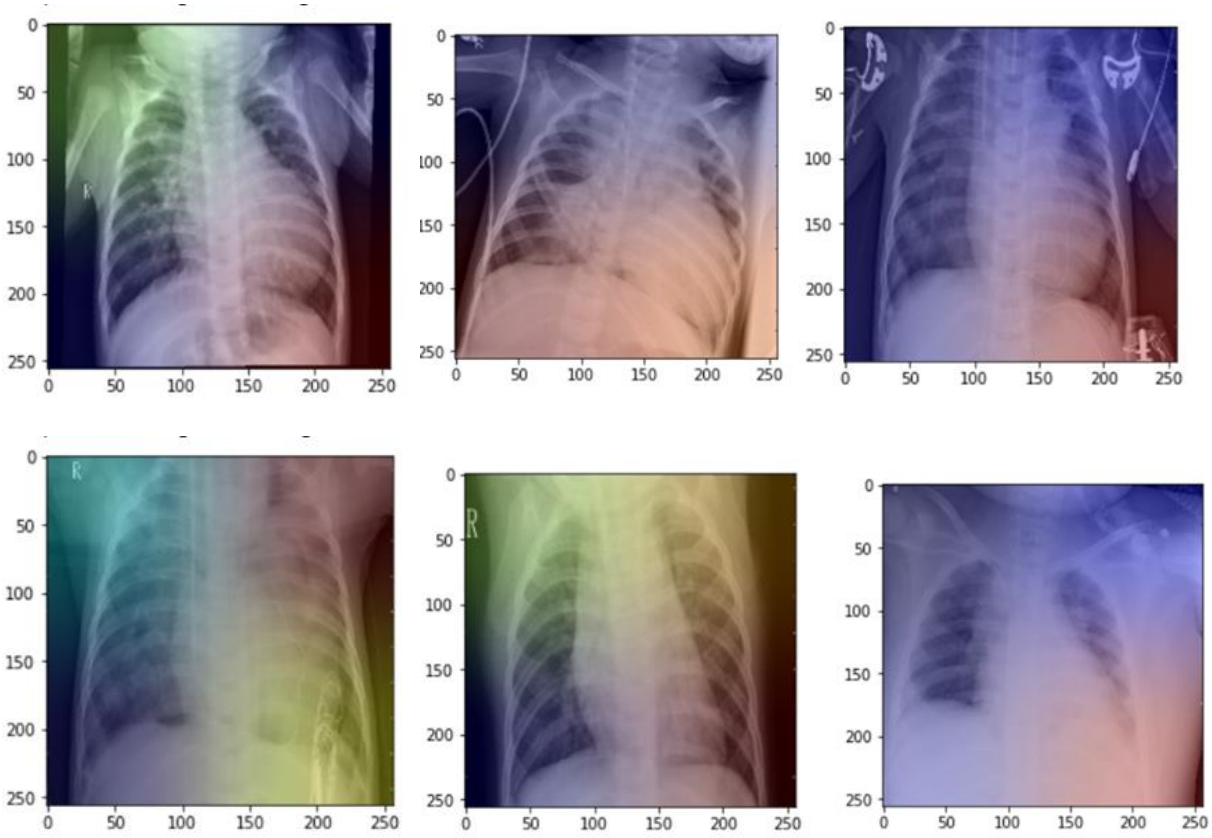


2. Normal:





3. Pneumonia:



Graphical outputs:

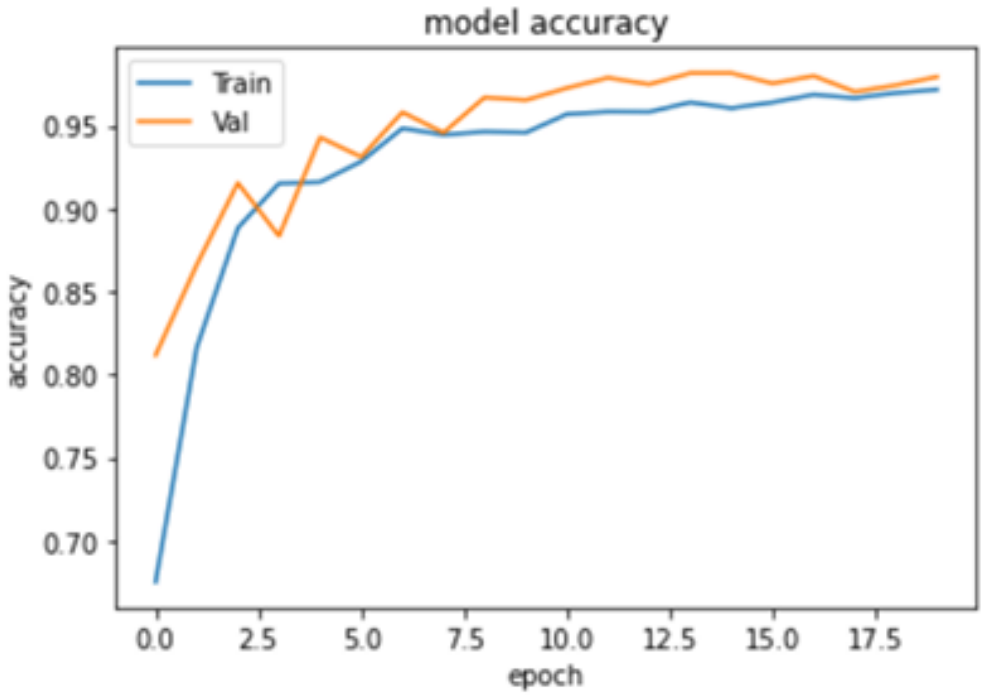


Fig: Model accuracy

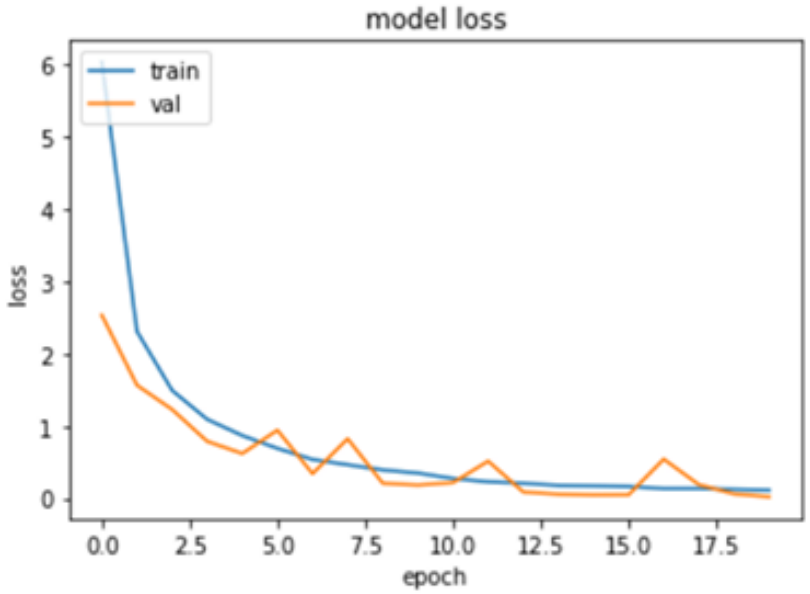


Fig: Model loss

Confusion matrix:

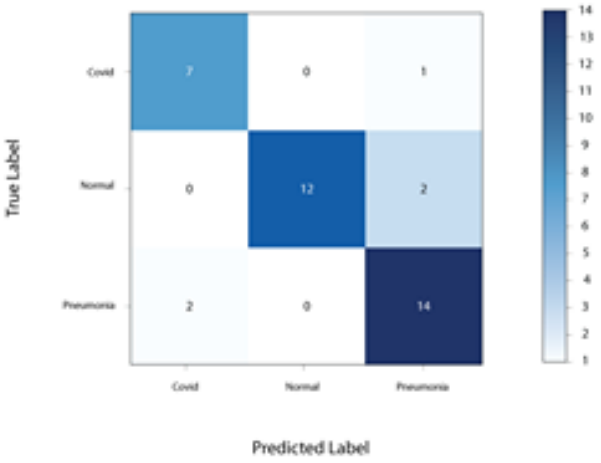


Fig: Confusion matrix of the 39 test samples

Sensitivity:

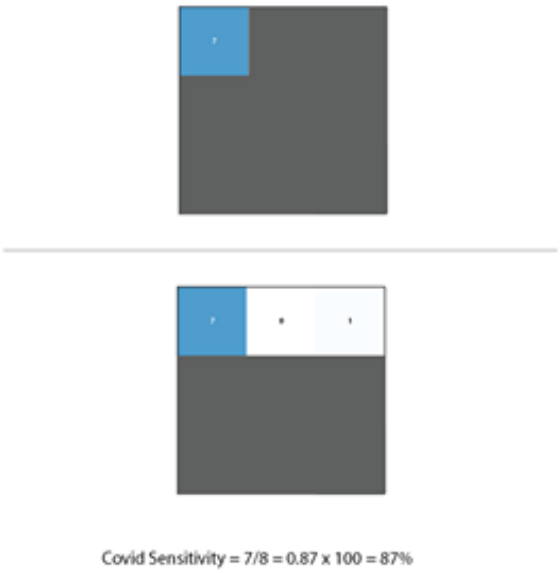


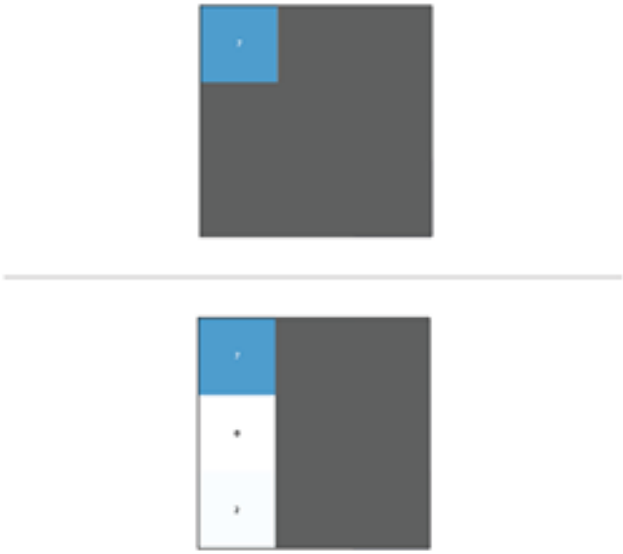
Fig: COVID-19 Sensitivity

Specificity:

Covid Specificity = 28/30 = 0.93 x 100 = 93%

Fig: COVID-19 specificity

Precision or positive predictive value PPV:



Positive Predictive Value = 7/9 = 0.77 x 100 = 87%

Fig: COVID-19 PPV

Negative predictive value NPV:

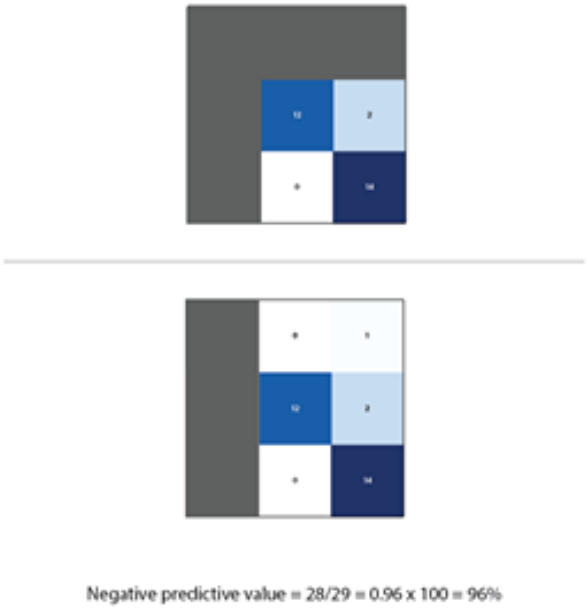


Fig: COVID-19 NPV

Accuracy: The custom CovidNet-8 model achieved training accuracy of 97.16% and a validation accuracy of 96.52%.

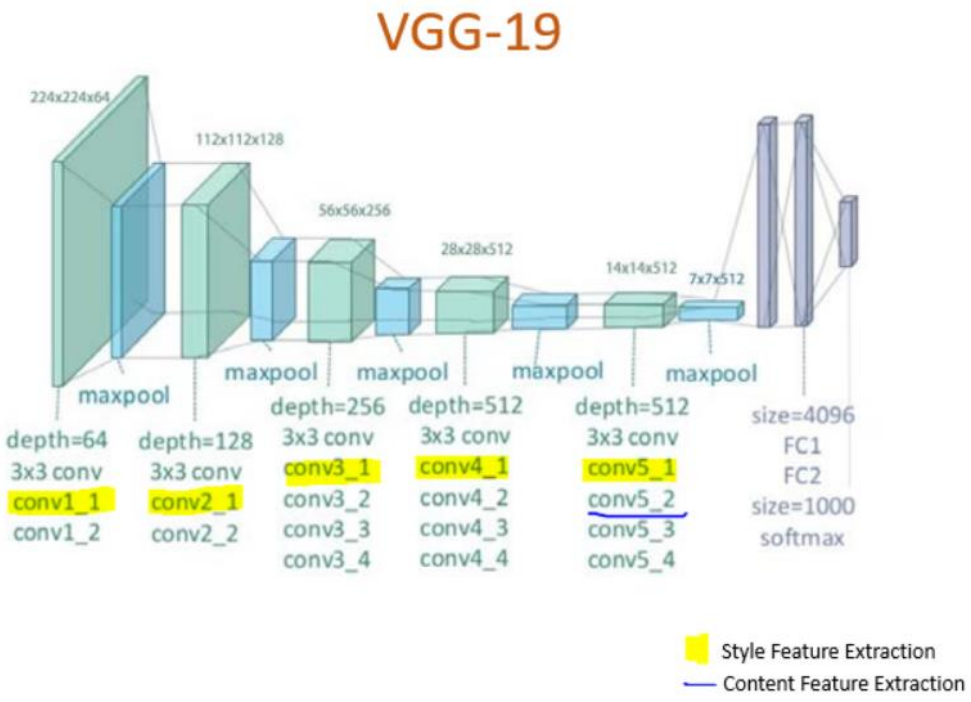
<u>Using the Pretrained model VGG-19 using transfer learning:</u>

For this segment the pretrained VGG-19 model is used on the same Covid-19 dataset for Covid-19 classification. The dataset consisted of chest x-ray images where 3 different types classes were present including Covid-19, pneumonia and normal ones.

Background:

VGG is a successor of the AlexNet. VGG-19 is a Pre-trained model that is built using transfer learning techniques, where the model uses knowledge to solve a problem (i.e., recognizing a boat) and applies it to a related but similar problem (recognizing a ship).

The VGG-19 model can recognize low-level features using shallow (earlier) layers and high-level features using deeper layers. The images below show the layer structure of the VGG-19 Network:



```
[13] Model: "vgg19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
```

Fig: VGG-19 architecture

Uses of the VGG Neural Network

- Used just as a good classification architecture for many other datasets and as the authors made the models available to the public they can be used as is or with modification for other similar tasks also.
- Transfer learning: can be used for facial recognition tasks also.
- weights are easily available with other frameworks like keras so they can be tinkered with and used for as one wants.
- Content and style loss using VGG-19 network

Customized VGG-19 Model Architecture:

Sample input:

Covid-19:

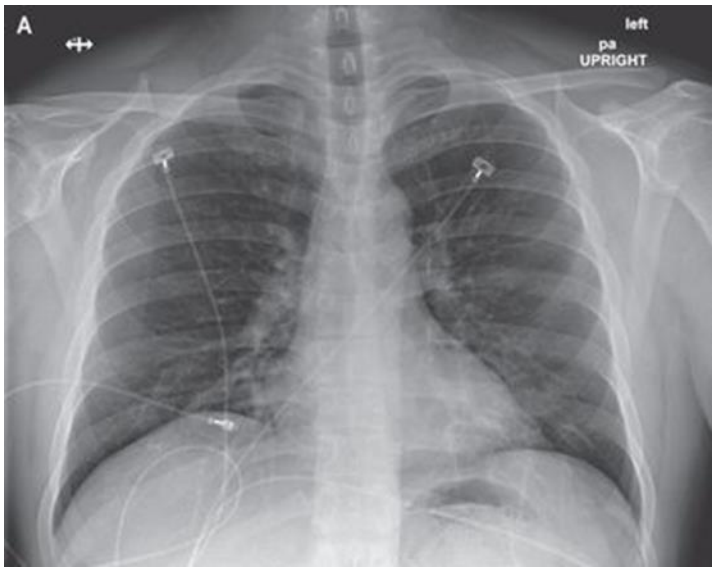


Fig: COVID-19 CXR

Normal:

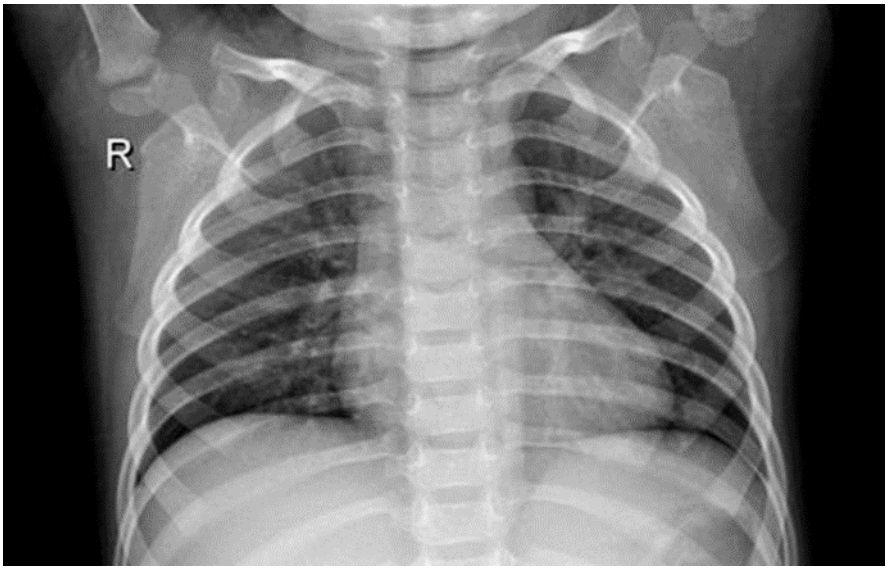


Fig: Normal CXR

Pneumonia:

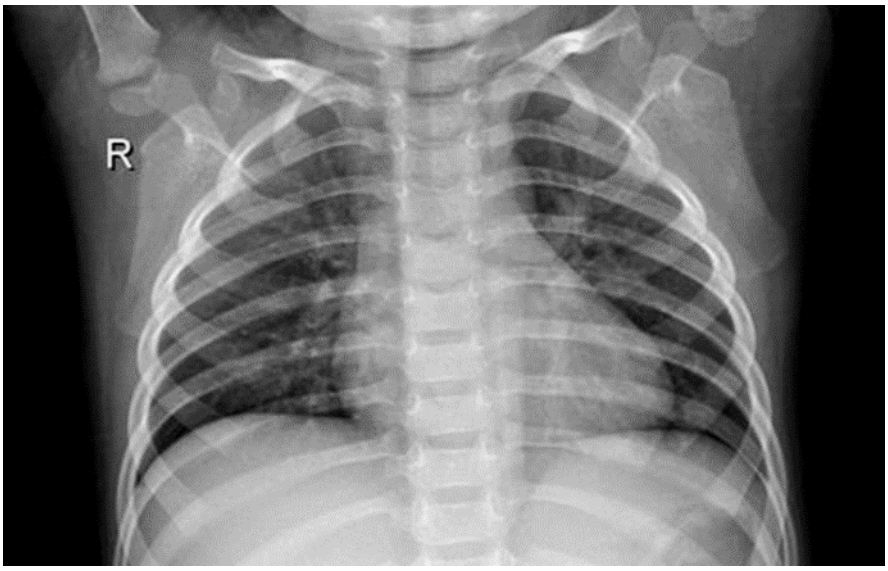


Fig: Pneumonia CXR

Architecture:

```
[17] Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
average_pooling2d_1 (Average (None, 1, 1, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                32832
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
output_layer (Dense)         (None, 3)                 195
=================================================================
Total params: 20,057,411
```

Fig: Customized VGG-19 model architecture

Training: The training set had 6900 samples with a batch size of 32.

```
Epoch 1/25
 - 1411s - loss: 3.8087 - accuracy: 0.5129 - val_loss: 1.3387 - val_accuracy: 0.6643
Epoch 2/25
 - 1203s - loss: 2.4957 - accuracy: 0.5848 - val_loss: 1.1341 - val_accuracy: 0.6911
Epoch 3/25
 - 1129s - loss: 1.5516 - accuracy: 0.6648 - val_loss: 0.2764 - val_accuracy: 0.7268
Epoch 4/25
 - 79s - loss: 1.2251 - accuracy: 0.6687 - val_loss: 0.8783 - val_accuracy: 0.7337
Epoch 5/25
 - 74s - loss: 1.0668 - accuracy: 0.6926 - val_loss: 1.3008 - val_accuracy: 0.7393
Epoch 6/25
 - 76s - loss: 0.9491 - accuracy: 0.6906 - val_loss: 0.4097 - val_accuracy: 0.7429
Epoch 7/25
 - 77s - loss: 0.8094 - accuracy: 0.7158 - val_loss: 0.3439 - val_accuracy: 0.7826
Epoch 8/25
 - 75s - loss: 0.7613 - accuracy: 0.7276 - val_loss: 0.4572 - val_accuracy: 0.7643
Epoch 9/25
 - 76s - loss: 0.7235 - accuracy: 0.7269 - val_loss: 0.5654 - val_accuracy: 0.7679
Epoch 10/25
 - 76s - loss: 0.6387 - accuracy: 0.7577 - val_loss: 0.6745 - val_accuracy: 0.7717
Epoch 11/25
 - 76s - loss: 0.6098 - accuracy: 0.7500 - val_loss: 0.4811 - val_accuracy: 0.8125
Epoch 12/25
 - 77s - loss: 0.6080 - accuracy: 0.7646 - val_loss: 0.2989 - val_accuracy: 0.7768
Epoch 13/25
 - 76s - loss: 0.5428 - accuracy: 0.7849 - val_loss: 0.4335 - val_accuracy: 0.8243
Epoch 14/25
 - 75s - loss: 0.5150 - accuracy: 0.8074 - val_loss: 0.4542 - val_accuracy: 0.7911
Epoch 15/25
 - 77s - loss: 0.5335 - accuracy: 0.7962 - val_loss: 0.1917 - val_accuracy: 0.8375
Epoch 16/25
 - 75s - loss: 0.5430 - accuracy: 0.7872 - val_loss: 0.5871 - val_accuracy: 0.8170
Epoch 17/25
 - 76s - loss: 0.4824 - accuracy: 0.8111 - val_loss: 0.2694 - val_accuracy: 0.8196
Epoch 18/25
 - 76s - loss: 0.4852 - accuracy: 0.8091 - val_loss: 0.1942 - val_accuracy: 0.8518
Epoch 19/25
 - 76s - loss: 0.4896 - accuracy: 0.8074 - val_loss: 0.3269 - val_accuracy: 0.8125
Epoch 20/25
 - 78s - loss: 0.4785 - accuracy: 0.8146 - val_loss: 0.2424 - val_accuracy: 0.8478
Epoch 21/25
 - 74s - loss: 0.4455 - accuracy: 0.8270 - val_loss: 0.2102 - val_accuracy: 0.8429
Epoch 22/25
 - 77s - loss: 0.4651 - accuracy: 0.8219 - val_loss: 0.5354 - val_accuracy: 0.8554
Epoch 23/25
 - 77s - loss: 0.4436 - accuracy: 0.8249 - val_loss: 0.3149 - val_accuracy: 0.8351
Epoch 24/25
 - 74s - loss: 0.4745 - accuracy: 0.8157 - val_loss: 0.4660 - val_accuracy: 0.8518
Epoch 25/25
 - 76s - loss: 0.4505 - accuracy: 0.8369 - val_loss: 0.2622 - val_accuracy: 0.8500
```

Fig: VGG-19 Model training for 25 epochs

Validating: The validation set had 1732 images with a batch size of 16.

Graphs:

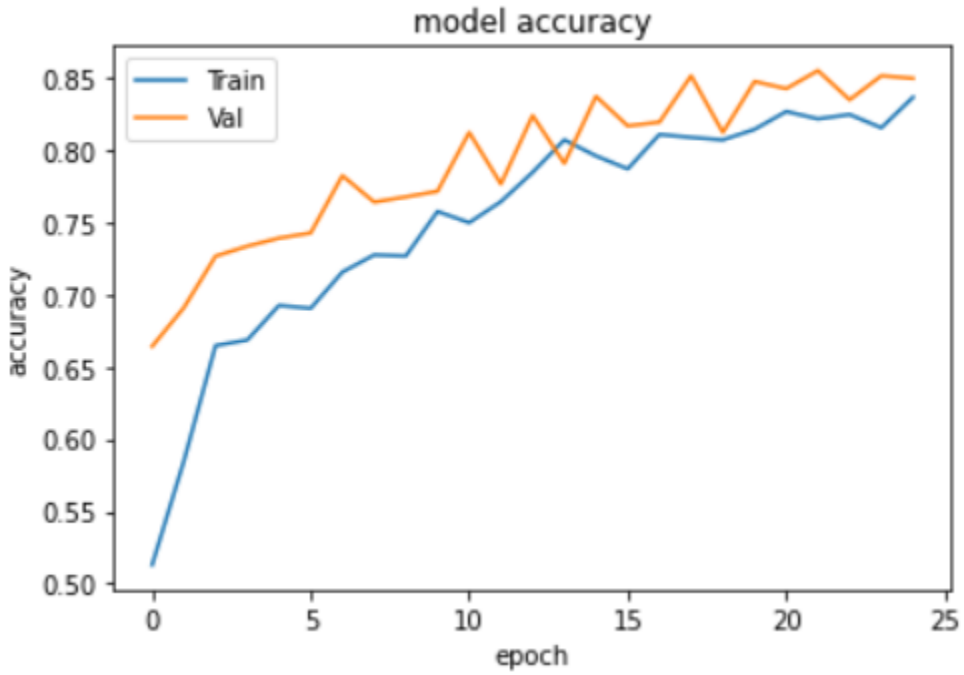


Fig: Model accuracy

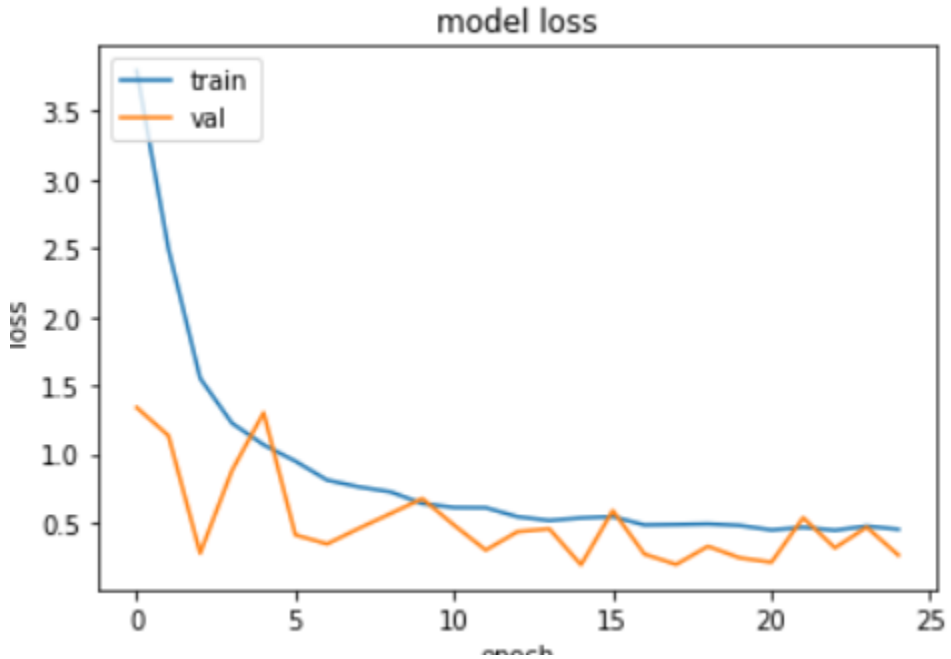


Fig: Model loss

Accuracy: The training accuracy was 83.69 and validation accuracy was 85.00.

Using the pre-trained Resnet-50 model:

For this segment the pretrained Resnet-50 model is used on the same Covid-19 dataset for Covid-19 classification. The dataset consisted of chest x-ray images where 3 different types classes were present including Covid-19, pneumonia and normal ones.

Background:

In a deep convolutional neural network, several layers are stacked and are trained to the task at hand. The network learns several low/mid/high level features at the end of its layers. In residual learning, instead of trying to learn some features, we try to learn some residual. Residual can be simply understood as subtraction of feature learned from input of that layer. ResNet does this using shortcut connections (directly connecting input of nth layer to some (n+x) th layer.

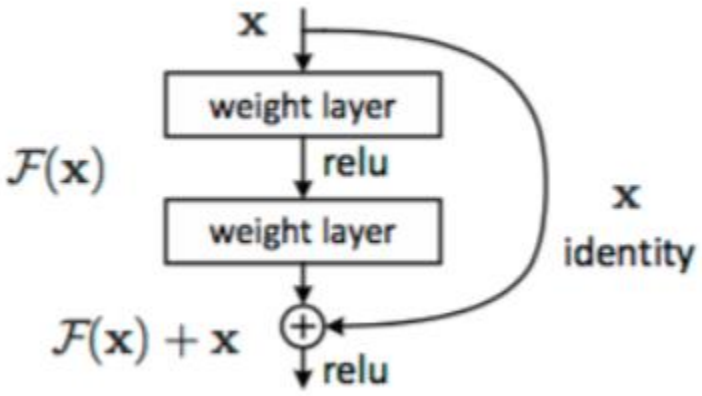


Fig: Residual learning: A building block

ResNet-50 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 50 layers deep and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224 x 224.

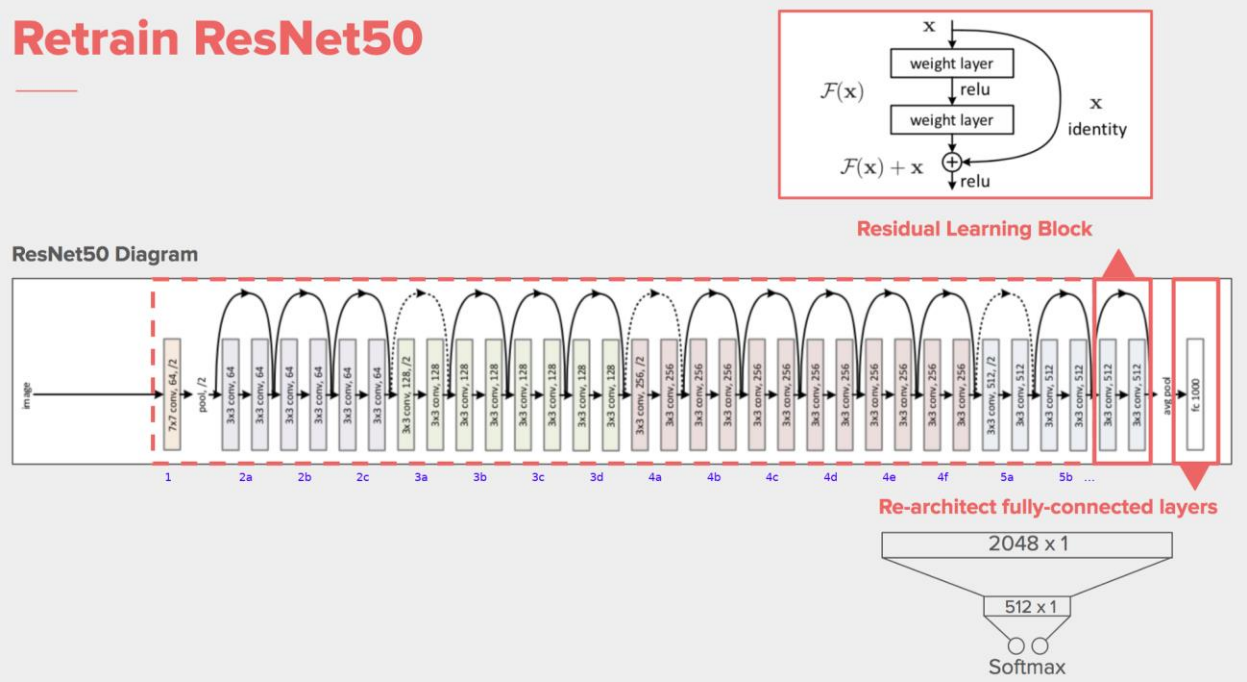


Fig: Resnet-50 architecture

Sample input:

Covid-19:

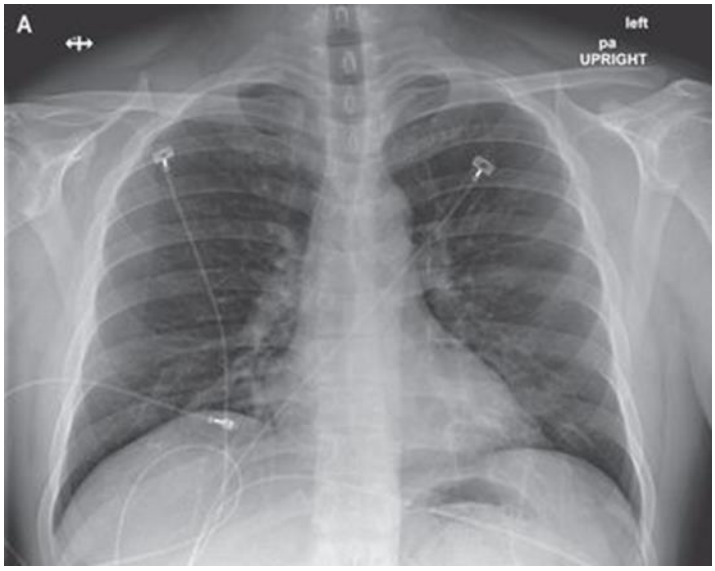


Fig: COVID-19 CXR

Normal:

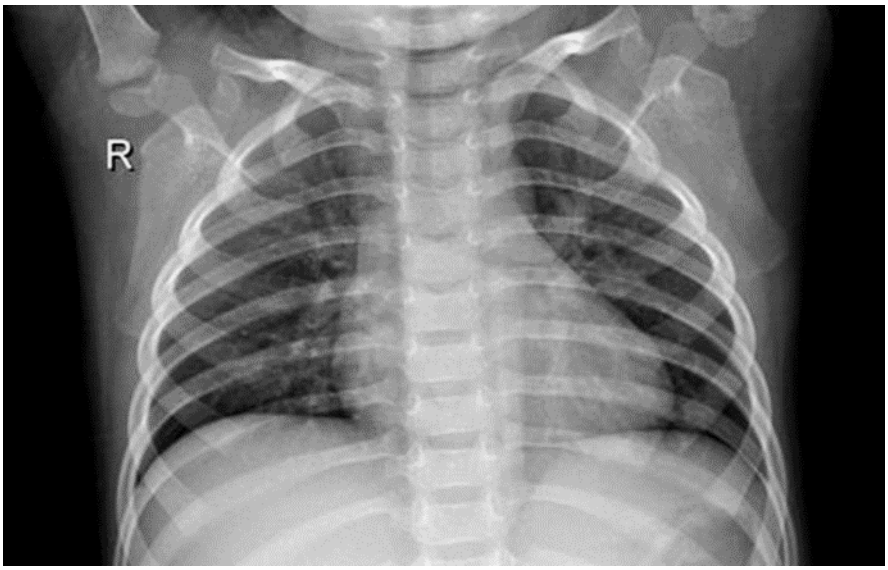


Fig: Normal CXR

Pneumonia:

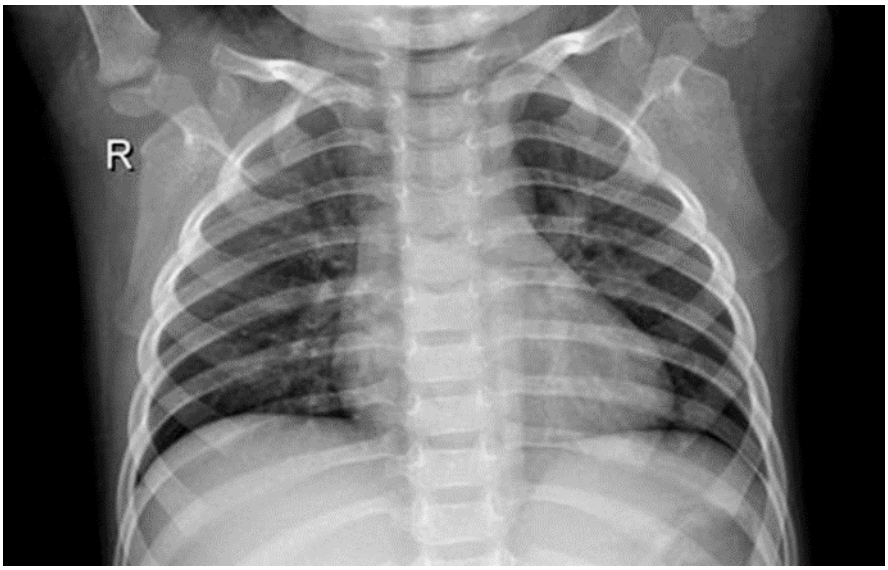


Fig: Pneumonia CXR

Architecture:

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 224, 224, 3)	0	
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	input_3[0][0]
conv1_conv (Conv2D)	(None, 112, 112, 64)	9472	conv1_pad[0][0]
conv1_bn (BatchNormalization)	(None, 112, 112, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 112, 112, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 56, 56, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 56, 56, 64)	4160	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormali	(None, 56, 56, 64)	256	conv2_block1_1_conv[0][0]
conv2_block1_1_relu (Activation	(None, 56, 56, 64)	0	conv2_block1_1_bn[0][0]
conv2_block1_2_conv (Conv2D)	(None, 56, 56, 64)	36928	conv2_block1_1_relu[0][0]
conv2_block1_2_bn (BatchNormali	(None, 56, 56, 64)	256	conv2_block1_2_conv[0][0]
conv2_block1_2_relu (Activation	(None, 56, 56, 64)	0	conv2_block1_2_bn[0][0]
conv2_block1_0_conv (Conv2D)	(None, 56, 56, 256)	16640	pool1_pool[0][0]
conv2_block1_3_conv (Conv2D)	(None, 56, 56, 256)	16640	conv2_block1_2_relu[0][0]
conv2_block1_0_bn (BatchNormali	(None, 56, 56, 256)	1024	conv2_block1_0_conv[0][0]
conv2_block1_3_bn (BatchNormali	(None, 56, 56, 256)	1024	conv2_block1_3_conv[0][0]
conv2_block1_add (Add)	(None, 56, 56, 256)	0	conv2_block1_0_bn[0][0] conv2_block1_3_bn[0][0]
conv2_block1_out (Activation)	(None, 56, 56, 256)	0	conv2_block1_add[0][0]
conv2_block2_1_conv (Conv2D)	(None, 56, 56, 64)	16448	conv2_block1_out[0][0]
conv2_block2_1_bn (BatchNormali	(None, 56, 56, 64)	256	conv2_block2_1_conv[0][0]
conv2_block2_1_relu (Activation	(None, 56, 56, 64)	0	conv2_block2_1_bn[0][0]
conv2_block2_2_conv (Conv2D)	(None, 56, 56, 64)	36928	conv2_block2_1_relu[0][0]
conv2_block2_2_bn (BatchNormali	(None, 56, 56, 64)	256	conv2_block2_2_conv[0][0]
conv2_block2_2_relu (Activation	(None, 56, 56, 64)	0	conv2_block2_2_bn[0][0]
conv2_block2_3_conv (Conv2D)	(None, 56, 56, 256)	16640	conv2_block2_2_relu[0][0]
conv2_block2_3_bn (BatchNormali	(None, 56, 56, 256)	1024	conv2_block2_3_conv[0][0]

conv2_block2_add (Add)	(None, 56, 56, 256)	0	conv2_block1_out[0][0] conv2_block2_3_bn[0][0]
conv2_block2_out (Activation)	(None, 56, 56, 256)	0	conv2_block2_add[0][0]
conv2_block3_1_conv (Conv2D)	(None, 56, 56, 64)	16448	conv2_block2_out[0][0]
conv2_block3_1_bn (BatchNormali	(None, 56, 56, 64)	256	conv2_block3_1_conv[0][0]
conv2_block3_1_relu (Activation	(None, 56, 56, 64)	0	conv2_block3_1_bn[0][0]
conv2_block3_2_conv (Conv2D)	(None, 56, 56, 64)	36928	conv2_block3_1_relu[0][0]
conv2_block3_2_bn (BatchNormali	(None, 56, 56, 64)	256	conv2_block3_2_conv[0][0]
conv2_block3_2_relu (Activation	(None, 56, 56, 64)	0	conv2_block3_2_bn[0][0]
conv2_block3_3_conv (Conv2D)	(None, 56, 56, 256)	16640	conv2_block3_2_relu[0][0]
conv2_block3_3_bn (BatchNormali	(None, 56, 56, 256)	1024	conv2_block3_3_conv[0][0]
conv2_block3_add (Add)	(None, 56, 56, 256)	0	conv2_block2_out[0][0] conv2_block3_3_bn[0][0]
conv2_block3_out (Activation)	(None, 56, 56, 256)	0	conv2_block3_add[0][0]
conv3_block1_1_conv (Conv2D)	(None, 28, 28, 128)	32896	conv2_block3_out[0][0]
conv3_block1_1_bn (BatchNormali	(None, 28, 28, 128)	512	conv3_block1_1_conv[0][0]
conv3_block1_1_relu (Activation	(None, 28, 28, 128)	0	conv3_block1_1_bn[0][0]
conv3_block1_2_conv (Conv2D)	(None, 28, 28, 128)	147584	conv3_block1_1_relu[0][0]
conv3_block1_2_bn (BatchNormali	(None, 28, 28, 128)	512	conv3_block1_2_conv[0][0]
conv3_block1_2_relu (Activation	(None, 28, 28, 128)	0	conv3_block1_2_bn[0][0]
conv3_block1_0_conv (Conv2D)	(None, 28, 28, 512)	131584	conv2_block3_out[0][0]
conv3_block1_3_conv (Conv2D)	(None, 28, 28, 512)	66048	conv3_block1_2_relu[0][0]
conv3_block1_0_bn (BatchNormali	(None, 28, 28, 512)	2048	conv3_block1_0_conv[0][0]
conv3_block1_3_bn (BatchNormali	(None, 28, 28, 512)	2048	conv3_block1_3_conv[0][0]
conv3_block1_add (Add)	(None, 28, 28, 512)	0	conv3_block1_0_bn[0][0] conv3_block1_3_bn[0][0]
conv3_block1_out (Activation)	(None, 28, 28, 512)	0	conv3_block1_add[0][0]
conv3_block2_1_conv (Conv2D)	(None, 28, 28, 128)	65664	conv3_block1_out[0][0]
conv3_block2_1_bn (BatchNormali	(None, 28, 28, 128)	512	conv3_block2_1_conv[0][0]
conv3_block2_1_relu (Activation	(None, 28, 28, 128)	0	conv3_block2_1_bn[0][0]
conv3_block2_2_conv (Conv2D)	(None, 28, 28, 128)	147584	conv3_block2_1_relu[0][0]

conv3_block2_1_relu (Activation	(None, 28, 28, 128)	0	conv3_block2_1_bn[0][0]
conv3_block2_2_conv (Conv2D)	(None, 28, 28, 128)	147584	conv3_block2_1_relu[0][0]
conv3_block2_2_bn (BatchNormali	(None, 28, 28, 128)	512	conv3_block2_2_conv[0][0]
conv3_block2_2_relu (Activation	(None, 28, 28, 128)	0	conv3_block2_2_bn[0][0]
conv3_block2_3_conv (Conv2D)	(None, 28, 28, 512)	66048	conv3_block2_2_relu[0][0]
conv3_block2_3_bn (BatchNormali	(None, 28, 28, 512)	2048	conv3_block2_3_conv[0][0]
conv3_block2_add (Add)	(None, 28, 28, 512)	0	conv3_block1_out[0][0] conv3_block2_3_bn[0][0]
conv3_block2_out (Activation)	(None, 28, 28, 512)	0	conv3_block2_add[0][0]
conv3_block3_1_conv (Conv2D)	(None, 28, 28, 128)	65664	conv3_block2_out[0][0]
conv3_block3_1_bn (BatchNormali	(None, 28, 28, 128)	512	conv3_block3_1_conv[0][0]
conv3_block3_1_relu (Activation	(None, 28, 28, 128)	0	conv3_block3_1_bn[0][0]
conv3_block3_2_conv (Conv2D)	(None, 28, 28, 128)	147584	conv3_block3_1_relu[0][0]
conv3_block3_2_bn (BatchNormali	(None, 28, 28, 128)	512	conv3_block3_2_conv[0][0]
conv3_block3_2_relu (Activation	(None, 28, 28, 128)	0	conv3_block3_2_bn[0][0]
conv3_block3_3_conv (Conv2D)	(None, 28, 28, 512)	66048	conv3_block3_2_relu[0][0]
conv3_block3_3_bn (BatchNormali	(None, 28, 28, 512)	2048	conv3_block3_3_conv[0][0]
conv3_block3_add (Add)	(None, 28, 28, 512)	0	conv3_block2_out[0][0] conv3_block3_3_bn[0][0]
conv3_block3_out (Activation)	(None, 28, 28, 512)	0	conv3_block3_add[0][0]
conv3_block4_1_conv (Conv2D)	(None, 28, 28, 128)	65664	conv3_block3_out[0][0]
conv3_block4_1_bn (BatchNormali	(None, 28, 28, 128)	512	conv3_block4_1_conv[0][0]
conv3_block4_1_relu (Activation	(None, 28, 28, 128)	0	conv3_block4_1_bn[0][0]
conv3_block4_2_conv (Conv2D)	(None, 28, 28, 128)	147584	conv3_block4_1_relu[0][0]
conv3_block4_2_bn (BatchNormali	(None, 28, 28, 128)	512	conv3_block4_2_conv[0][0]
conv3_block4_2_relu (Activation	(None, 28, 28, 128)	0	conv3_block4_2_bn[0][0]
conv3_block4_3_conv (Conv2D)	(None, 28, 28, 512)	66048	conv3_block4_2_relu[0][0]
conv3_block4_3_bn (BatchNormali	(None, 28, 28, 512)	2048	conv3_block4_3_conv[0][0]
conv3_block4_add (Add)	(None, 28, 28, 512)	0	conv3_block3_out[0][0] conv3_block4_3_bn[0][0]
conv3_block4_out (Activation)	(None, 28, 28, 512)	0	conv3_block4_add[0][0]

```
conv3_block4_out (Activation)   (None, 28, 28, 512)  0           conv3_block4_add[0][0]
conv4_block1_1_conv (Conv2D)    (None, 14, 14, 256)  131328      conv3_block4_out[0][0]
conv4_block1_1_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block1_1_conv[0][0]
conv4_block1_1_relu (Activation (None, 14, 14, 256)  0           conv4_block1_1_bn[0][0]
conv4_block1_2_conv (Conv2D)    (None, 14, 14, 256)  590080      conv4_block1_1_relu[0][0]
conv4_block1_2_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block1_2_conv[0][0]
conv4_block1_2_relu (Activation (None, 14, 14, 256)  0           conv4_block1_2_bn[0][0]
conv4_block1_0_conv (Conv2D)    (None, 14, 14, 1024) 525312      conv3_block4_out[0][0]
conv4_block1_3_conv (Conv2D)    (None, 14, 14, 1024) 263168      conv4_block1_2_relu[0][0]
conv4_block1_0_bn (BatchNormali (None, 14, 14, 1024) 4096        conv4_block1_0_conv[0][0]
conv4_block1_3_bn (BatchNormali (None, 14, 14, 1024) 4096        conv4_block1_3_conv[0][0]
conv4_block1_add (Add)          (None, 14, 14, 1024) 0           conv4_block1_0_bn[0][0]
                                                                 conv4_block1_3_bn[0][0]
conv4_block1_out (Activation)   (None, 14, 14, 1024) 0           conv4_block1_add[0][0]
conv4_block2_1_conv (Conv2D)    (None, 14, 14, 256)  262400      conv4_block1_out[0][0]
conv4_block2_1_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block2_1_conv[0][0]
conv4_block2_1_relu (Activation (None, 14, 14, 256)  0           conv4_block2_1_bn[0][0]
conv4_block2_2_conv (Conv2D)    (None, 14, 14, 256)  590080      conv4_block2_1_relu[0][0]
conv4_block2_2_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block2_2_conv[0][0]
conv4_block2_2_relu (Activation (None, 14, 14, 256)  0           conv4_block2_2_bn[0][0]
conv4_block2_3_conv (Conv2D)    (None, 14, 14, 1024) 263168      conv4_block2_2_relu[0][0]
conv4_block2_3_bn (BatchNormali (None, 14, 14, 1024) 4096        conv4_block2_3_conv[0][0]
conv4_block2_add (Add)          (None, 14, 14, 1024) 0           conv4_block1_out[0][0]
                                                                 conv4_block2_3_bn[0][0]
conv4_block2_out (Activation)   (None, 14, 14, 1024) 0           conv4_block2_add[0][0]
conv4_block3_1_conv (Conv2D)    (None, 14, 14, 256)  262400      conv4_block2_out[0][0]
conv4_block3_1_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block3_1_conv[0][0]
conv4_block3_1_relu (Activation (None, 14, 14, 256)  0           conv4_block3_1_bn[0][0]
conv4_block3_2_conv (Conv2D)    (None, 14, 14, 256)  590080      conv4_block3_1_relu[0][0]
conv4_block3_2_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block3_2_conv[0][0]
conv4_block3_2_relu (Activation (None, 14, 14, 256)  0           conv4_block3_2_bn[0][0]
```

```
]
conv4_block3_3_conv (Conv2D)    (None, 14, 14, 1024) 263168      conv4_block3_2_relu[0][0]
__________________________________________________________________________________________________
conv4_block3_3_bn (BatchNormali (None, 14, 14, 1024) 4096        conv4_block3_3_conv[0][0]
__________________________________________________________________________________________________
conv4_block3_add (Add)          (None, 14, 14, 1024) 0           conv4_block2_out[0][0]
                                                                 conv4_block3_3_bn[0][0]
__________________________________________________________________________________________________
conv4_block3_out (Activation)   (None, 14, 14, 1024) 0           conv4_block3_add[0][0]
__________________________________________________________________________________________________
conv4_block4_1_conv (Conv2D)    (None, 14, 14, 256)  262400      conv4_block3_out[0][0]
__________________________________________________________________________________________________
conv4_block4_1_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block4_1_conv[0][0]
__________________________________________________________________________________________________
conv4_block4_1_relu (Activation (None, 14, 14, 256)  0           conv4_block4_1_bn[0][0]
__________________________________________________________________________________________________
conv4_block4_2_conv (Conv2D)    (None, 14, 14, 256)  590080      conv4_block4_1_relu[0][0]
__________________________________________________________________________________________________
conv4_block4_2_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block4_2_conv[0][0]
__________________________________________________________________________________________________
conv4_block4_2_relu (Activation (None, 14, 14, 256)  0           conv4_block4_2_bn[0][0]
__________________________________________________________________________________________________
conv4_block4_3_conv (Conv2D)    (None, 14, 14, 1024) 263168      conv4_block4_2_relu[0][0]
__________________________________________________________________________________________________
conv4_block4_3_bn (BatchNormali (None, 14, 14, 1024) 4096        conv4_block4_3_conv[0][0]
__________________________________________________________________________________________________
conv4_block4_add (Add)          (None, 14, 14, 1024) 0           conv4_block3_out[0][0]
                                                                 conv4_block4_3_bn[0][0]
__________________________________________________________________________________________________
conv4_block4_out (Activation)   (None, 14, 14, 1024) 0           conv4_block4_add[0][0]
__________________________________________________________________________________________________
conv4_block5_1_conv (Conv2D)    (None, 14, 14, 256)  262400      conv4_block4_out[0][0]
__________________________________________________________________________________________________
conv4_block5_1_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block5_1_conv[0][0]
__________________________________________________________________________________________________
conv4_block5_1_relu (Activation (None, 14, 14, 256)  0           conv4_block5_1_bn[0][0]
__________________________________________________________________________________________________
conv4_block5_2_conv (Conv2D)    (None, 14, 14, 256)  590080      conv4_block5_1_relu[0][0]
__________________________________________________________________________________________________
conv4_block5_2_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block5_2_conv[0][0]
__________________________________________________________________________________________________
conv4_block5_2_relu (Activation (None, 14, 14, 256)  0           conv4_block5_2_bn[0][0]
__________________________________________________________________________________________________
conv4_block5_3_conv (Conv2D)    (None, 14, 14, 1024) 263168      conv4_block5_2_relu[0][0]
__________________________________________________________________________________________________
conv4_block5_3_bn (BatchNormali (None, 14, 14, 1024) 4096        conv4_block5_3_conv[0][0]
__________________________________________________________________________________________________
conv4_block5_add (Add)          (None, 14, 14, 1024) 0           conv4_block4_out[0][0]
                                                                 conv4_block5_3_bn[0][0]
__________________________________________________________________________________________________
conv4_block5_out (Activation)   (None, 14, 14, 1024) 0           conv4_block5_add[0][0]
__________________________________________________________________________________________________
conv4_block6_1_conv (Conv2D)    (None, 14, 14, 256)  262400      conv4_block5_out[0][0]
__________________________________________________________________________________________________
conv4_block6_1_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block6_1_conv[0][0]
__________________________________________________________________________________________________
conv4_block6_1_relu (Activation (None, 14, 14, 256)  0           conv4_block6_1_bn[0][0]
__________________________________________________________________________________________________
conv4_block6_2_conv (Conv2D)    (None, 14, 14, 256)  590080      conv4_block6_1_relu[0][0]
```

```
conv4_block6_1_relu (Activation (None, 14, 14, 256)  0           conv4_block6_1_bn[0][0]
__________________________________________________________________________________________________
conv4_block6_2_conv (Conv2D)    (None, 14, 14, 256)  590080      conv4_block6_1_relu[0][0]
__________________________________________________________________________________________________
conv4_block6_2_bn (BatchNormali (None, 14, 14, 256)  1024        conv4_block6_2_conv[0][0]
__________________________________________________________________________________________________
conv4_block6_2_relu (Activation (None, 14, 14, 256)  0           conv4_block6_2_bn[0][0]
__________________________________________________________________________________________________
conv4_block6_3_conv (Conv2D)    (None, 14, 14, 1024) 263168      conv4_block6_2_relu[0][0]
__________________________________________________________________________________________________
conv4_block6_3_bn (BatchNormali (None, 14, 14, 1024) 4096        conv4_block6_3_conv[0][0]
__________________________________________________________________________________________________
conv4_block6_add (Add)          (None, 14, 14, 1024) 0           conv4_block5_out[0][0]
                                                                 conv4_block6_3_bn[0][0]
__________________________________________________________________________________________________
conv4_block6_out (Activation)   (None, 14, 14, 1024) 0           conv4_block6_add[0][0]
__________________________________________________________________________________________________
conv5_block1_1_conv (Conv2D)    (None, 7, 7, 512)    524800      conv4_block6_out[0][0]
__________________________________________________________________________________________________
conv5_block1_1_bn (BatchNormali (None, 7, 7, 512)    2048        conv5_block1_1_conv[0][0]
__________________________________________________________________________________________________
conv5_block1_1_relu (Activation (None, 7, 7, 512)    0           conv5_block1_1_bn[0][0]
__________________________________________________________________________________________________
conv5_block1_2_conv (Conv2D)    (None, 7, 7, 512)    2359808     conv5_block1_1_relu[0][0]
__________________________________________________________________________________________________
conv5_block1_2_bn (BatchNormali (None, 7, 7, 512)    2048        conv5_block1_2_conv[0][0]
__________________________________________________________________________________________________
conv5_block1_2_relu (Activation (None, 7, 7, 512)    0           conv5_block1_2_bn[0][0]
__________________________________________________________________________________________________
conv5_block1_0_conv (Conv2D)    (None, 7, 7, 2048)   2099200     conv4_block6_out[0][0]
__________________________________________________________________________________________________
conv5_block1_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624     conv5_block1_2_relu[0][0]
__________________________________________________________________________________________________
conv5_block1_0_bn (BatchNormali (None, 7, 7, 2048)   8192        conv5_block1_0_conv[0][0]
__________________________________________________________________________________________________
conv5_block1_3_bn (BatchNormali (None, 7, 7, 2048)   8192        conv5_block1_3_conv[0][0]
__________________________________________________________________________________________________
conv5_block1_add (Add)          (None, 7, 7, 2048)   0           conv5_block1_0_bn[0][0]
                                                                 conv5_block1_3_bn[0][0]
__________________________________________________________________________________________________
conv5_block1_out (Activation)   (None, 7, 7, 2048)   0           conv5_block1_add[0][0]
__________________________________________________________________________________________________
conv5_block2_1_conv (Conv2D)    (None, 7, 7, 512)    1049088     conv5_block1_out[0][0]
__________________________________________________________________________________________________
conv5_block2_1_bn (BatchNormali (None, 7, 7, 512)    2048        conv5_block2_1_conv[0][0]
__________________________________________________________________________________________________
conv5_block2_1_relu (Activation (None, 7, 7, 512)    0           conv5_block2_1_bn[0][0]
__________________________________________________________________________________________________
conv5_block2_2_conv (Conv2D)    (None, 7, 7, 512)    2359808     conv5_block2_1_relu[0][0]
__________________________________________________________________________________________________
conv5_block2_2_bn (BatchNormali (None, 7, 7, 512)    2048        conv5_block2_2_conv[0][0]
__________________________________________________________________________________________________
conv5_block2_2_relu (Activation (None, 7, 7, 512)    0           conv5_block2_2_bn[0][0]
__________________________________________________________________________________________________
conv5_block2_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624     conv5_block2_2_relu[0][0]
__________________________________________________________________________________________________
conv5_block2_3_bn (BatchNormali (None, 7, 7, 2048)   8192        conv5_block2_3_conv[0][0]
__________________________________________________________________________________________________
```

conv5_block2_out (Activation)	(None, 7, 7, 2048)	0	conv5_block2_add[0][0]
conv5_block3_1_conv (Conv2D)	(None, 7, 7, 512)	1049088	conv5_block2_out[0][0]
conv5_block3_1_bn (BatchNormali	(None, 7, 7, 512)	2048	conv5_block3_1_conv[0][0]
conv5_block3_1_relu (Activation	(None, 7, 7, 512)	0	conv5_block3_1_bn[0][0]
conv5_block3_2_conv (Conv2D)	(None, 7, 7, 512)	2359808	conv5_block3_1_relu[0][0]
conv5_block3_2_bn (BatchNormali	(None, 7, 7, 512)	2048	conv5_block3_2_conv[0][0]
conv5_block3_2_relu (Activation	(None, 7, 7, 512)	0	conv5_block3_2_bn[0][0]
conv5_block3_3_conv (Conv2D)	(None, 7, 7, 2048)	1050624	conv5_block3_2_relu[0][0]
conv5_block3_3_bn (BatchNormali	(None, 7, 7, 2048)	8192	conv5_block3_3_conv[0][0]
conv5_block3_add (Add)	(None, 7, 7, 2048)	0	conv5_block2_out[0][0] conv5_block3_3_bn[0][0]
conv5_block3_out (Activation)	(None, 7, 7, 2048)	0	conv5_block3_add[0][0]
average_pooling2d_2 (AveragePoo	(None, 1, 1, 2048)	0	conv5_block3_out[0][0]
flatten (Flatten)	(None, 2048)	0	average_pooling2d_2[0][0]
dense_2 (Dense)	(None, 64)	131136	flatten[0][0]
dropout_2 (Dropout)	(None, 64)	0	dense_2[0][0]
output_layer (Dense)	(None, 3)	195	dropout_2[0][0]

Total params: 23,719,043
Trainable params: 23,665,923
Non-trainable params: 53,120

Fig: Custom Resnet-50 model architecture

Training: The training set had 6900 samples with a batch size of 32.

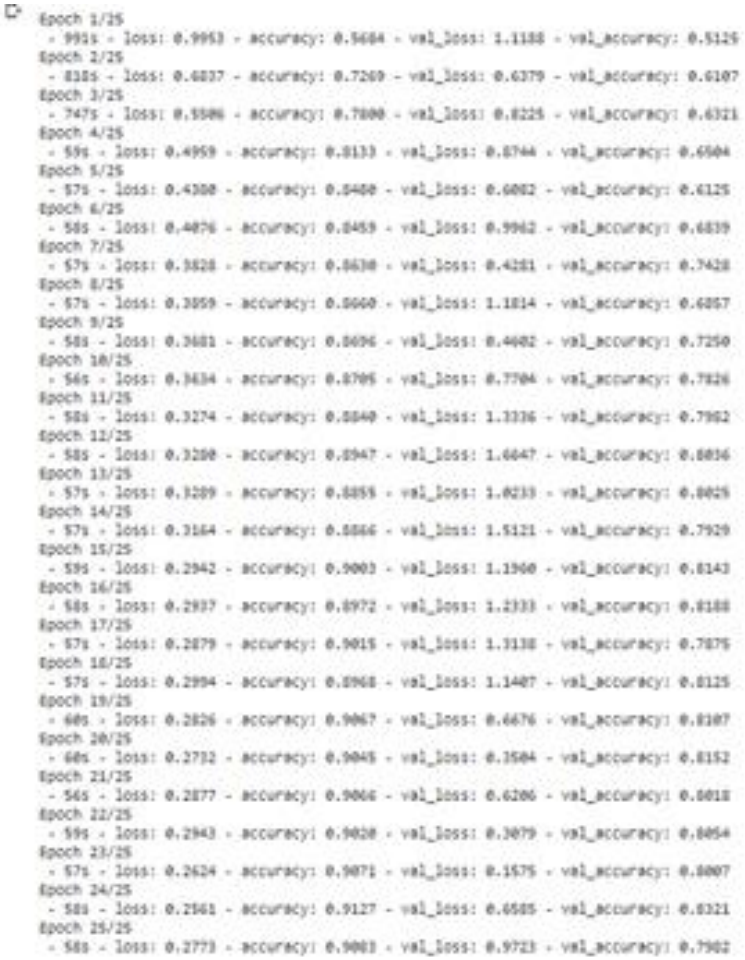


Fig: Model Training with 25 epochs

Validating: The training set had 1732 samples with a batch size of 16.

Graphs:

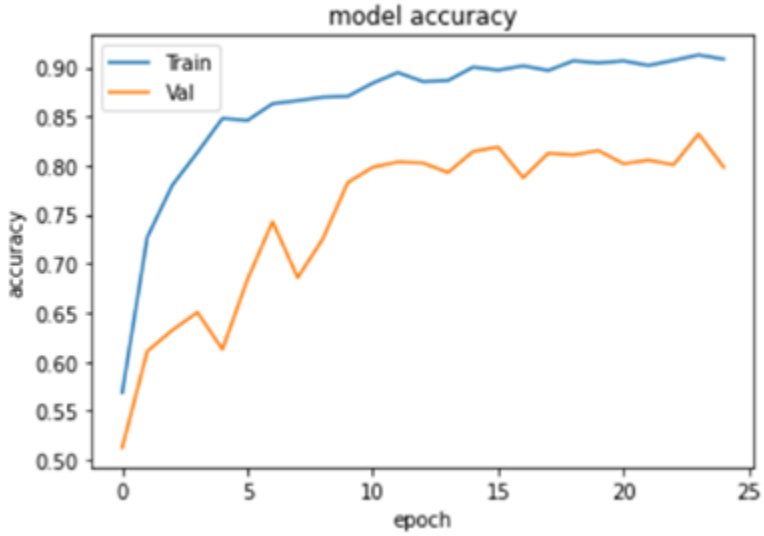


Fig: Model accuracy

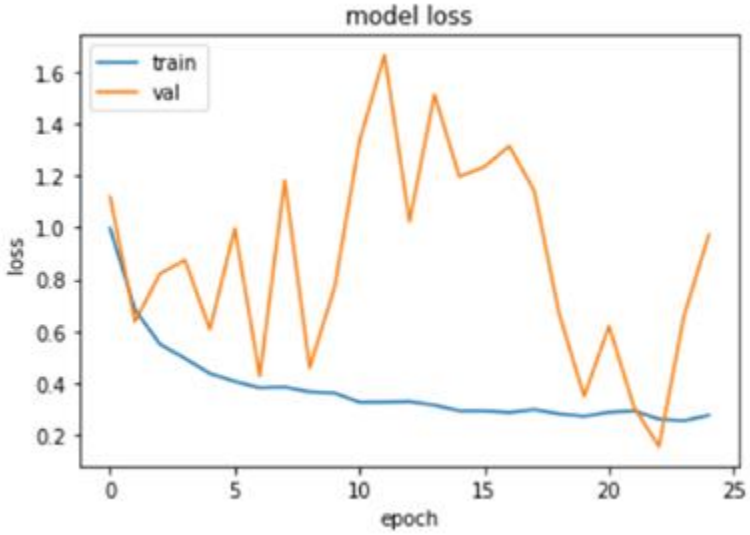


Fig: Model loss

Accuracy: The training accuracy was 90.93% and the validation accuracy was 79.92%.

1.3 Summary

Overall, all of the model worked pretty well on the given dataset but particularly the customized CNN model worked really well and gave a better accuracy. The aim of this project was to successfully predict Covid-19 from the chest X-ray images.

CHAPTER: 02

MOTIVATION

2.1 Introduction

In this chapter we will discuss our motivation due to which we thought of working and implementing this system. We will also discuss in this chapter as to why we have chosen the medical field apart from all other fields to work it.

2.2 Motivation towards our project

This year has been extremely stressful with the corona virus outbreak and the rising number of the patients. This whole situation is a crisis not only for the government and general people but for doctors as well who's working relentlessly day and night to treat those patients. Incorporating Artificial Intelligence to fight this pandemic was one of the vital move scientists have taken and one of them is implementing in the screening process. The main goal of this project is to build an AI based classification system that will help to increase the screening accuracy compared to the existing system and also make it easier to use, more efficient and reliable. This system will help understand the concept of AI based Covid-19 detection system and ways to detect the cancer. It will classify different types of respiratory diseases, mostly Covid-19, based on the dataset provided both from internet and real-life data and then several models will be implemented using that dataset to compare their accuracies. This system will be easily available for the radiologists to use. The medical field was chosen because not only it is very interesting topic but also the contribution will be helpful in future for a good cause.

2.3 Summary

This chapter provided the idea about the motivation towards our project which aims to detect and classify Covid-19 from CXR images without any other extra diagnosis.

CHAPTER: 03

RELATED WORKS

3.1 Introduction

This chapter discusses the different types of covid-19 detection methods that currently exist. We also focus on the drawbacks that the current system entails and how we are planning to overcome those drawbacks and improve our system.

3.2 Systems related to our project

Few common lists of systems have been discussed below:

1. Viral test or diagnostic tests:

A viral test checks samples to find out if the patient is currently infected with COVID-19. The time it takes to process these tests can vary. Currently there are two types of diagnostic tests which detect the virus – **molecular** tests, such as RT-PCR tests, that detect the virus's genetic material, and **antigen** tests that detect specific proteins on the surface of the virus.

It is tested with a swab that is placed inside the nose or throat. The swab is then tested to see if the virus is present or not.

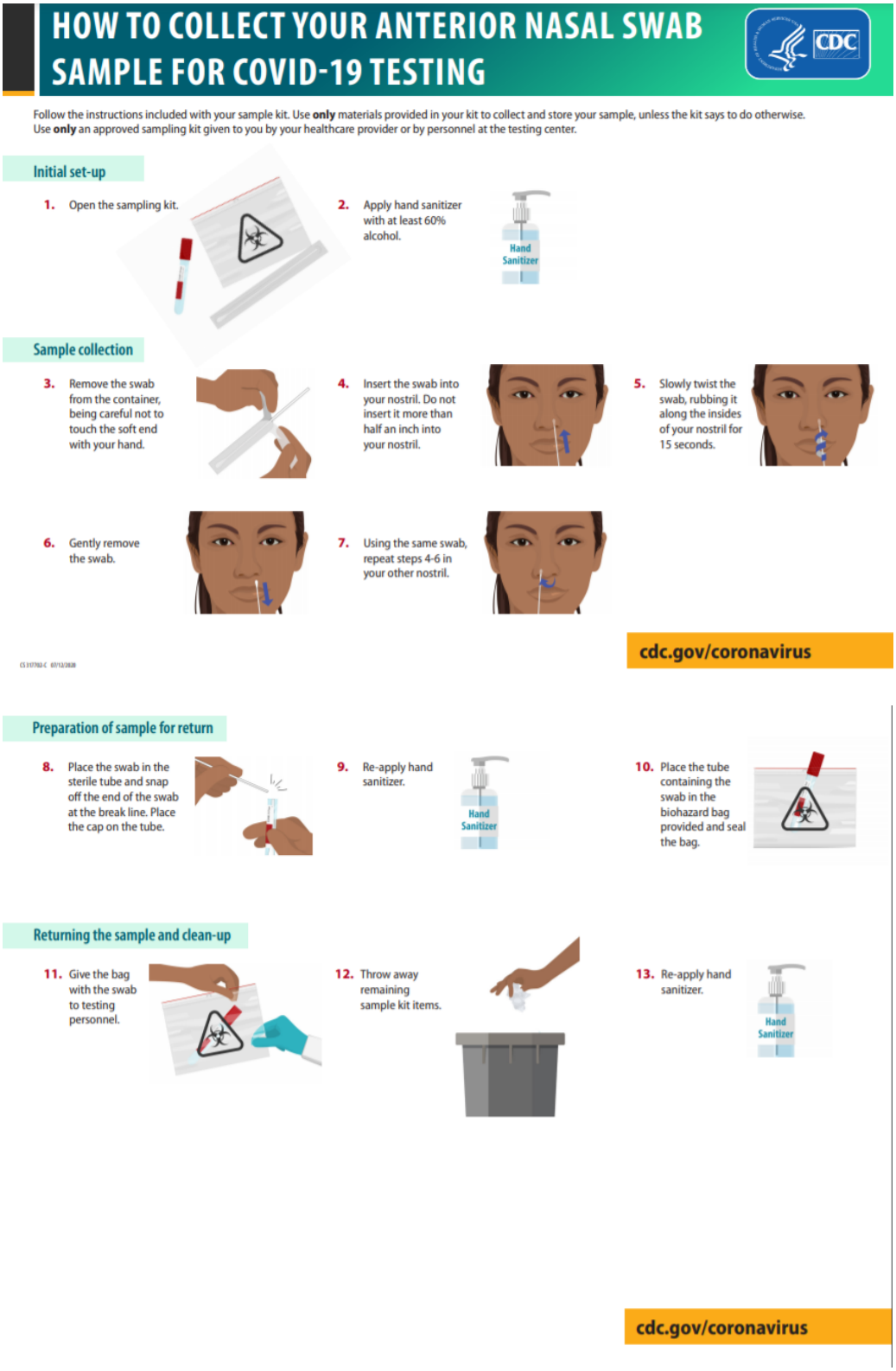


Fig: Nasal test swab procedure

Antibody test:

Antibody tests check the blood by looking for antibodies, which may tell the patient if they had a past infection with the virus that causes COVID-19. Antibodies are proteins that help fight off infections and can provide protection against getting that disease again (immunity). Antibody tests should not be used to diagnose a current COVID-19 infection, except if the viral testing is delayed. An antibody test may not show if the patient has a current COVID-19 infection because it can take 1–3 weeks after infection for the body to make antibodies. Antibody tests for COVID-19 are available through healthcare providers and laboratories.

	Molecular Test	Antigen Test	Antibody Test
Also known as...	Diagnostic test, viral test, molecular test, nucleic acid amplification test (NAAT), RT-PCR test, LAMP test	Rapid diagnostic test (Some molecular tests are also rapid tests.)	Serological test, serology, blood test, serology test
How the sample is taken...	Nasal or throat swab (most tests) Saliva (a few tests)	Nasal or throat swab	Finger stick or blood draw
How long it takes to get results...	Same day (some locations) or up to a week	One hour or less	Same day (many locations) or 1-3 days
Is another test needed...	This test is typically highly accurate and usually does not need to be repeated.	Positive results are usually highly accurate but negative results may need to be confirmed with a molecular test.	Sometimes a second antibody test is needed for accurate results.
What it shows...	Diagnoses active coronavirus infection	Diagnoses active coronavirus infection	Shows if you've been infected by coronavirus in the past
What it can't do...	Show if you ever had COVID-19 or were infected with the coronavirus in the past	Definitively rule out active coronavirus infection. Antigen tests are more likely to miss an active coronavirus infection compared to molecular tests. Your health care provider may order a molecular test if your antigen test shows a negative result but you have symptoms of COVID-19.	Diagnose active coronavirus infection at the time of the test or show that you do not have COVID-19

Fig: Difference between Viral and antibody test

2. Nucleic acid test:

Nucleic acid testing is the primary method of diagnosing COVID-19. A number of reverse transcription polymerase chain reaction (RT-PCR) kits have been designed to detect SARS-CoV-2 genetically. RT-PCR involves the reverse transcription of SARS-CoV-2 RNA into complementary DNA (cDNA) strands, followed by amplification of specific regions of the cDNA. The design process generally involves two main steps: (1) sequence alignment and primer design, and (2) assay optimization and testing.

3. Point-of-Care Testing

Point-of-care tests are used to diagnose patients without sending samples to centralized facilities, thereby enabling communities without laboratory infrastructure to detect infected patients. Lateral flow antigen detection for SARS-CoV-2 is one point-of-care approach under development for diagnosing COVID-19.

4. Smartphone Surveillance of Infectious Diseases

Controlling epidemics requires extensive surveillance, sharing of epidemiological data, and patient monitoring. Healthcare entities, from local hospitals to the WHO, require tools that can improve the speed and ease of communication to manage the spread of diseases. Smartphones can be leveraged for this purpose as they possess the connectivity, computational power, and hardware to facilitate electronic reporting, epidemiological databasing, and point-of-care testing.

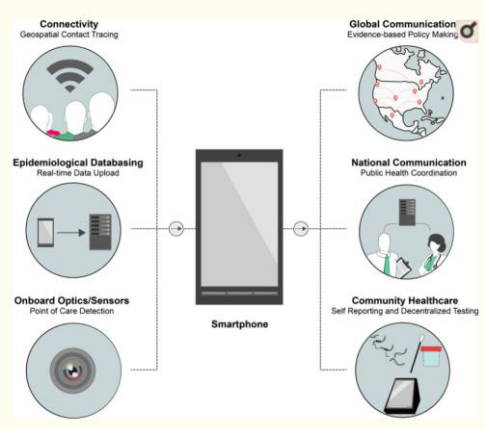


Fig: Role of smartphones in diagnostics

5. Next generation sequencing (NGS)

The, next-generation sequencing (NGS) is also called as high-throughput sequencing (HTS). This method can be used to determine the genomic sequence, even more than 1 million base pairs in a single experiment and can diagnose the inheritable diseases, cancer, and infectious diseases.

6. Loop-mediated isothermal amplification (LAMP)

LAMP is a novel technique which in process of approval for COVID-19 diagnosis is molecular amplification technique that can amplify any genomic material with high efficiency and in shorter time. It is a user-friendly technique which can provides reliable, sensitive and specific results in lesser time as compared to other conventional techniques. This technique has the advantage of requiring only single constant temperature, and thus eliminating the need of thermocycler and so as power consumption.

7. Rapid test

Rapid tests involve non-automated and are used for in vitro diagnostics (IVDs) for COVID-19 diagnosis. These tests can provide results within 10–30 min, so their results are considered as instant. Additionally, these tests are user friendly, thus won't require any extensive training or expertise to operate and can be used either in hospital environment, in the laboratories or at patient bedside without any difficulty.

8. Biosensor

Biosensor is a self-contained integrated analytical device consisting of the bioreceptor, transducer and a signal detector.

Nano-biosensors using aptamers are one of such potent analytical tools for rapid diagnosis of diseases with high sensitivity and specificity in a cost effective and user-friendly manner compared to conventional methods. These nano-sensor will have great potential for detection of SARS-CoV-2 even in person without any symptoms with high sensitivity, specificity and selectivity only for COVID-19.

9. Paper based detection

An alternative paper-based technology using waste water as samples has been suggested by Kang Mao et al. This technology can act as an alternative detection tool for rapid tracing for the source or presence of causative agents like COVID-19 in any pandemic area. Feces and urine from disease carriers in the community, entering in the sewer system could contain many biomarkers of the virus, and same has been confirmed in a recent study which showed that these infectious agents can remain active for several days even after has been disseminated from the patients, if found suitable environment. There is strong potential in this paper-based device to trace the COVID-19 transmission in community wastewater by analyzing SARS-CoV-2 in feces, urine and other excreted output of human

10. Computed Tomography

Due to the shortage of kits and false negative rate of RT-PCR, doctors use CT scans as a clinical diagnosis for COVID-19. Chest CT scans are non-invasive and involve taking many X-ray measurements at different angles across a patient's chest to produce cross-sectional images. The images are analyzed by radiologists to look for abnormal features that can lead to a diagnosis. The imaging features of COVID-19 are diverse and depend on the stage of infection after the onset of symptoms. The most common hallmark features of COVID-19 include bilateral and peripheral ground-glass opacities (areas of hazy opacity) and consolidations of the lungs (fluid or solid material in compressible lung tissue). Based on these imaging features, several retrospective studies have shown that CT scans have a higher sensitivity (86–98%) and improved false negative rates compared to RT-PCR. The main caveat of using CT for COVID-19 is that the specificity is low (25%) because the imaging features overlap with other viral pneumonia.

11. A.I based classifications:

Machine learning techniques on chest X- Rays are getting popularity as they can be easily utilized with low-cost imaging techniques and there is an abundance of data available for training different machine-learning models. Concept of transfer learning in deep learning framework was utilized by Vikash et al. for the recognition of pneumonia utilizing pre-trained ImageNet models and their ensembles.

A customized VGG16 model was used by Xianghong et al.for lung regions identification and various sorts of pneumonia characterization. Wang et al. used a large dataset and Ronneburger et al. used image augmentation along with CNN to show signs of improvement results via preparing on little arrangement of pictures. Rajpurkar et al. announced a 121-layer CNN on chest X- rays to identify 14 distinct pathologies, including pneumonia utilizing an ensemble of different networks. A pre-trained DenseNet-121 and feature extraction techniques were used in the accurate identification of 14 thoracic diseases in. Sundaram et al. used AlexNet and GoogLeNet with image augmentation to obtain an Area Under the Curve (AUC) of 0.95 in pneumonia discovery. Shuai et al. used deep learning techniques on CT images to screen COVID-19 patients with an accuracy, specificity and sensitivity of 89.5%, 88% and 87% respectively. Linda et al. presented a DCNN, called COVID-Net for the detection of COVID-19 cases from the chest X-ray images with an accuracy of 83.5%. Ayrton used a small dataset of 339 images for training and testing using ResNet50 based deep transfer learning technique and reported the validation accuracy of 96.2%. In this study, we have developed an automatic detection of COVID-19 using a DCNN based Inception V3 model and Chest X-ray images. AI-based algorithms can readily identify CT scans with COVID-19 associated pneumonia, as well as distinguish non-COVID related pneumonias with high specificity in diverse patient populations.

3.3 Drawbacks

Each technique has its own drawbacks. The issues with RT-PCR are: 1) the availability of PCR reagent kits has not kept up with demand. 2) community hospitals outside of urban cities lack the PCR infrastructure to accommodate high sample throughput and 3) RT-PCR relies on the presence of detectable SARS-CoV-2 in the sample collected. If an asymptomatic patient was infected with SARS-CoV-2 but has since recovered, PCR would not identify this prior infection, and control measures would not be implemented on that patient. Meanwhile, CT systems are expensive, require technical expertise, and cannot specifically diagnose COVID-19 because of its overlapping with other respiratory diseases. Other technologies need to be adapted to SARS-CoV-2 to address these deficiencies. The huge cost of the equipment and

chemicals required for NGS restricts its utilization in routine laboratory diagnosis of the diseases.

The manual system could never reach its pinnacle in terms of its efficiency. X-rays can be quite difficult to interpret. Moreover, based on the level of experience of the radiologist their recommendations can vary too. Furthermore; whenever particular information is needed to be retrieved it becomes really difficult to find out at a given point of time. Again, accuracy comparisons between two detection devices can be very difficult; also because of the inherent variability of technologies like chest X-rays. Thus, different radiologists can interpret the same X-ray differently. Furthermore, little differences while placing the chest during the different testing procedures can alter the resulting images and how those images are interpreted.

Chest X-rays are one of the best Covid-19 diagnosis methods but CXRs also have their limits such as, their accuracy level isn't 100% in showing if a person is COVID positive or not due to its overlapping nature with other diseases. CXRs can give a false-positive or false-negative result where false-negative CXR looks normal even though the patient is COVID positive and false-positive CXR looks abnormal even though the patient is not Covid positive, this is most common when a radiologist manually checks those mammograms without using any A.I based classification system or other diagnostic test results.

Even AI based classification systems are not always 100% accurate especially the earlier versions. There are several different types of classification systems using image processing or deep learning to COVID-19 classification system. But as it's already mentioned earlier that these classification systems themselves are not always 100% reliable but it has the potential to decrease the false-negative rate.

3.4 Proposed Solution

Recently with the rapid advancement in Deep and machine learning, it is possible to increase the accuracy level of these DCNN based classification systems for each specific disease. Thus, the proposed solution is to try to improve the existing classification systems using deep

learning and neural networks mostly. Initially we tried to run the same dataset on various models including VGG-16, VGG-19, Resnet-30, Resnet-50, a custom DCNN which we later named Covidnet-8 and compared the accuracies between the models. In this way our project will give a broader prospective on which model works best for what type of dataset. We also tried to tweak the models and tried to increase the accuracies of our models and try to achieve maximum accuracy with minimum loss.

3.5 Summary

Considering all aspects, our proposed solution is an optimal one since it will help to compare and detect the current flaws in these A.I based DCNN classification systems and it will also be easier to try to improve them. The DCNN based COVID-19 classification system will allow the radiologists classify the presence and the type of breast cancer and reduce false-negative results. It will also be very easy to navigate as we will integrate software for better user experience and can be maintained by non-IT people as well. Therefore, this chapter gives the idea about the current systems that are available in the market for COVID-19 diagnosis and about the motivation towards developing this dynamic system.

CHAPTER: 04

SKILLS

4.1 Introduction

In this section, we will discuss about the skills that we have obtained so far in order to develop the COVID-19 classification system using DCNN.

4.2 Skills Obtained

Throughout the making of this project, the following skills have been developed.

- **Python**

  Python is a high-level programming language. There are a lot of factors that puts Python higher up in the programming language chain, and they are. Readable and maintainable code, multiple programming paradigms, compatible with almost all major platforms and systems. Robust standard library containing thousands of functions. Many Opens source framework and tools, simplifies complex software development and many more. When it comes to deep learning, python has libraries like NumPy, SciPy, Pandas, Matplotlib; frameworks like Theano, TensorFlow, Keras etc. These libraries help in building efficient deep learning models.

Fig: Python logo

- **Jupyter Notebooks**

  The Jupyter Notebook is an open-source web application, which can be used to create and share documents that contain live code, equations, visualizations and narrative text. Jupyter Notebook can be used for data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, building effective deep learning models and more. For our project we used Google Colab, which is a free Jupyter Notebook environment. Using our .edu email, we were give 5TB of drive space, a dedicated GPU and 12 GB of ram. But one caveat of using Google Colab is that, it can run only for 12 hours at a stretch.[7]

Fig: Jupyter Notebook Logo

- **Google Colab**

  Colaboratory is a **Google** research project created to help disseminate machine learning education and research. It's a Jupyter notebook environment that requires no setup to use and runs entirely in the cloud.

Fig: Google colab

- **Neural Networks**

  A Computer neural network is a series of algorithms that attempts to recognize the basic relationships in a set of data through a process that imitates the way a normal human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adjust to change in input, so the neural network can generate the best possible result without needing the output criteria to be redesigned.

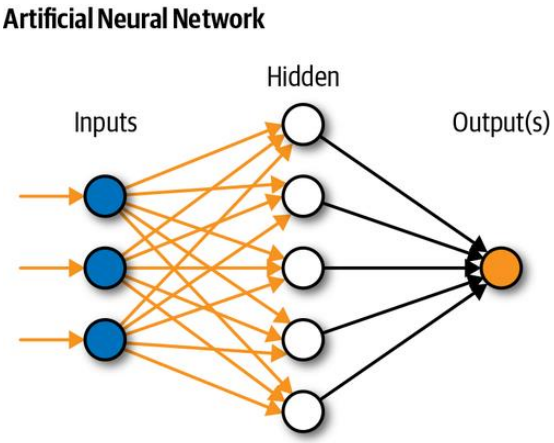


Fig: An artificial neural network

- **Basis knowledge of Chest X-rays Pictures**

  Since we are working on detectingCovid-19, and we are using datasets of CXR images, we needed to learn a bit about Corona virus in general and also the procedures of screening CXRs. We learned about the orientations of a CXR, difference between top view, side view. How to detect the regions of interest and most importantly the visual differences between a COVID-19 positive CXR, Normal CXR and a CXR of a patient with Pneumonia.

Fig: Covid-19 positive CXR

4.3 Summary

This chapter we discussed about all the skills that have been obtained throughout the process of developing our DCNN based COVID-19 classification System.

CHAPTER: 05

DATASETS

5.1 Introduction

This section will discuss about the datasets that have been gathered so far, for training the models.

5.2 Collecting data from real world

This is the main goal of this project, to see how the proposed models work with real life collected datasets of COVID-19 and other respiratory disease patients. This is still an ongoing search and several hospitals have been approached for this and finally one of them responded, we are still on a working on the collection of that dataset and possibly come up with a better solution by using them. The proposed work does not include any real-life dataset for now.

5.3 Collecting data from the internet

The dataset used in the proposed models are a combination of several datasets collected from different sources from Internet. Kaggle is one of the vital source of these datasets as most of the open source datasets are available there in the public domain and can be used for projects with given credits.

The sources of the combined dataset are:

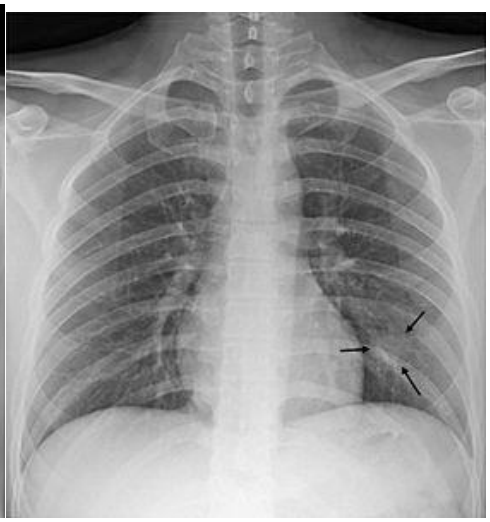
1. https://l.facebook.com/l.php?u=https%3A%2F%2Fwww.pyimagesearch.com%2F2020%2F03%2F16%2Fdetecting-covid-19-in-x-ray-images-with-keras-tensorflow-and-deep-learning%2F%3Ffbclid%3DIwAR3NxfOA_w2kiLX7g-E5-A_jnuC6vISnseuEu7lLUsiILI7kNc1C56Imots&h=AT15RUW1MY7t7a1dErsC23CuabiTygsFZoTDBhXu9JiLE7g4mgbo8DHRzeh1thsixy-AKS4gTFLsOon-lMbIFzT8IomZYss9aFVp2XMlycnieBaYwxeAWpmBFnKBThOOGUoPNw
2. https://l.facebook.com/l.php?u=https%3A%2F%2Fwww.kaggle.com%2Fdarshan1504%2Fcovid19-detection-xray-dataset%3Ffbclid%3DIwAR0-etAjaNvLUDbyBKkW-uecKcsYS6Pve5ehmrI2qiziN213ZlBGoGHKMJE&h=AT15RUW1MY7t7a1dErsC23CuabiTygsFZoTDBhXu9JiLE7g4mgbo8DHRzeh1thsixy-AKS4gTFLsOon-lMbIFzT8IomZYss9aFVp2XMlycnieBaYwxeAWpmBFnKBThOOGUoPNw

3. https://l.facebook.com/l.php?u=https%3A%2F%2Fwww.kaggle.com%2Fpranavraikokte%2Fcovid19-image-dataset%3Ffbclid%3DIwAR1lp9A6rV7rhPeKmifGGkRaO9t_G5U-l03K-5ryB3Qe50BzvtiI4T7ACzE&h=AT15RUW1MY7t7a1dErsC23CuabiTygsFZoTDBhXu9JiLE7g4mgbo8DHRzeh1thsixy-AKS4gTFLsOon-lMbIFzT8IomZYss9aFVp2XMlycnieBaYwxeAWpmBFnKBThOOGUoPNw
4. https://l.facebook.com/l.php?u=https%3A%2F%2Fwww.kaggle.com%2Fmuhammadmasdar%2Fpneumonia-virus-vs-covid19%3Ffbclid%3DIwAR1HWFff2FTN5qKMjoOwTSBFRDbwysCgv0uC60VzICtH1lXyD9QpwZQJnLg&h=AT15RUW1MY7t7a1dErsC23CuabiTygsFZoTDBhXu9JiLE7g4mgbo8DHRzeh1thsixy-AKS4gTFLsOon-lMbIFzT8IomZYss9aFVp2XMlycnieBaYwxeAWpmBFnKBThOOGUoPNw
5. https://l.facebook.com/l.php?u=https%3A%2F%2Fwww.kaggle.com%2Fusmantahirkiani%2Fcovid19-vs-healthy-xray%3Ffbclid%3DIwAR0gUMh_2BqozqmzAWK7qYeoLk54mWmRkdqniaLAPolqzLmr3m6BekWA8QA&h=AT15RUW1MY7t7a1dErsC23CuabiTygsFZoTDBhXu9JiLE7g4mgbo8DHRzeh1thsixy-AKS4gTFLsOon-lMbIFzT8IomZYss9aFVp2XMlycnieBaYwxeAWpmBFnKBThOOGUoPNw
6. https://l.facebook.com/l.php?u=https%3A%2F%2Fwww.kaggle.com%2Fuddiptadas%2Fcovid19-xray-images5%3Ffbclid%3DIwAR2JqNj0CSkfJzDYjDJhBQbdgvDzwgrE3wF7VzaDJ10PpzVeDtTxwC9dltg&h=AT15RUW1MY7t7a1dErsC23CuabiTygsFZoTDBhXu9JiLE7g4mgbo8DHRzeh1thsixy-AKS4gTFLsOon-lMbIFzT8IomZYss9aFVp2XMlycnieBaYwxeAWpmBFnKBThOOGUoPNw
7. https://l.facebook.com/l.php?u=https%3A%2F%2Fwww.kaggle.com%2Fakkinasrikar%2Fcovid19-xray-images-using-cnn%3Ffbclid%3DIwAR3OGTgXDFFtAqwNppj_XVVbLu48omNRgdejpuIlycP2wg-YU5Oge0N3c0I&h=AT15RUW1MY7t7a1dErsC23CuabiTygsFZoTDBhXu9JiLE7g4mgbo8DHRzeh1thsixy-AKS4gTFLsOon-lMbIFzT8IomZYss9aFVp2XMlycnieBaYwxeAWpmBFnKBThOOGUoPNw

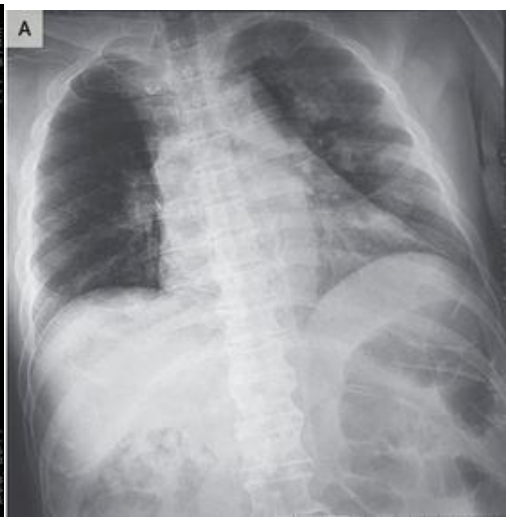
5.4 Sample datasets

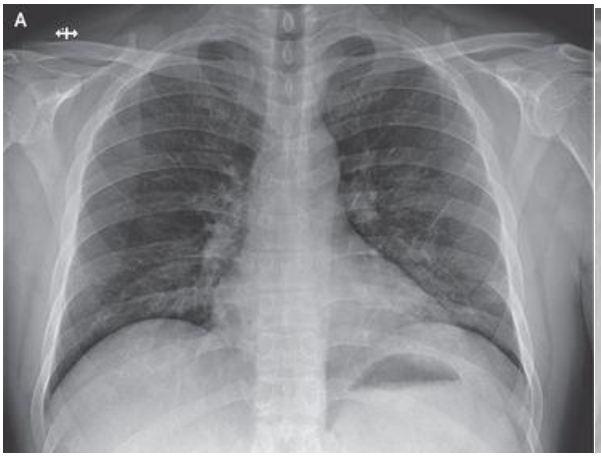
1. COVID-19:


PA ERECT
L

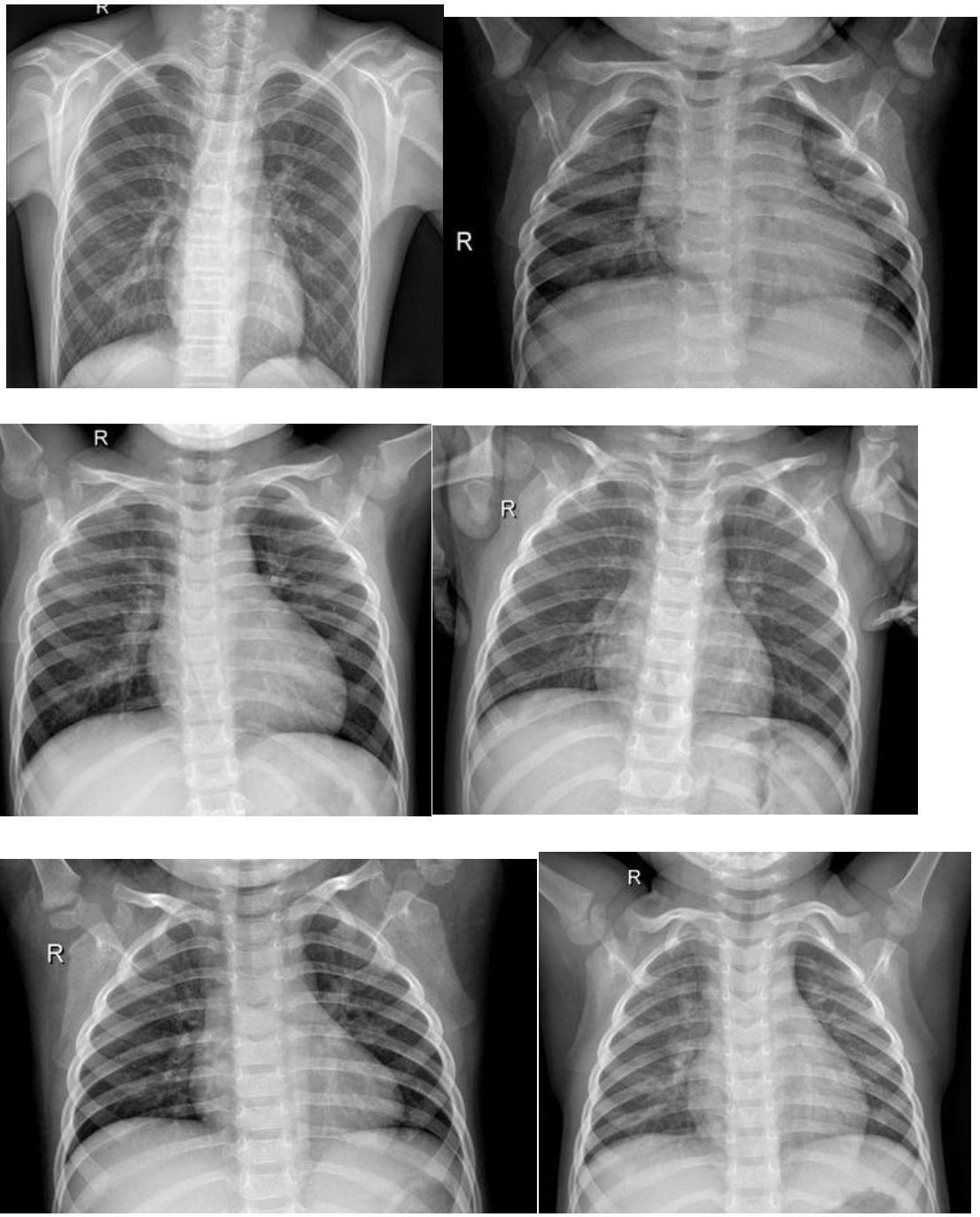



RX THORACE
Serie 1001
05/03/2020, 09:04:13

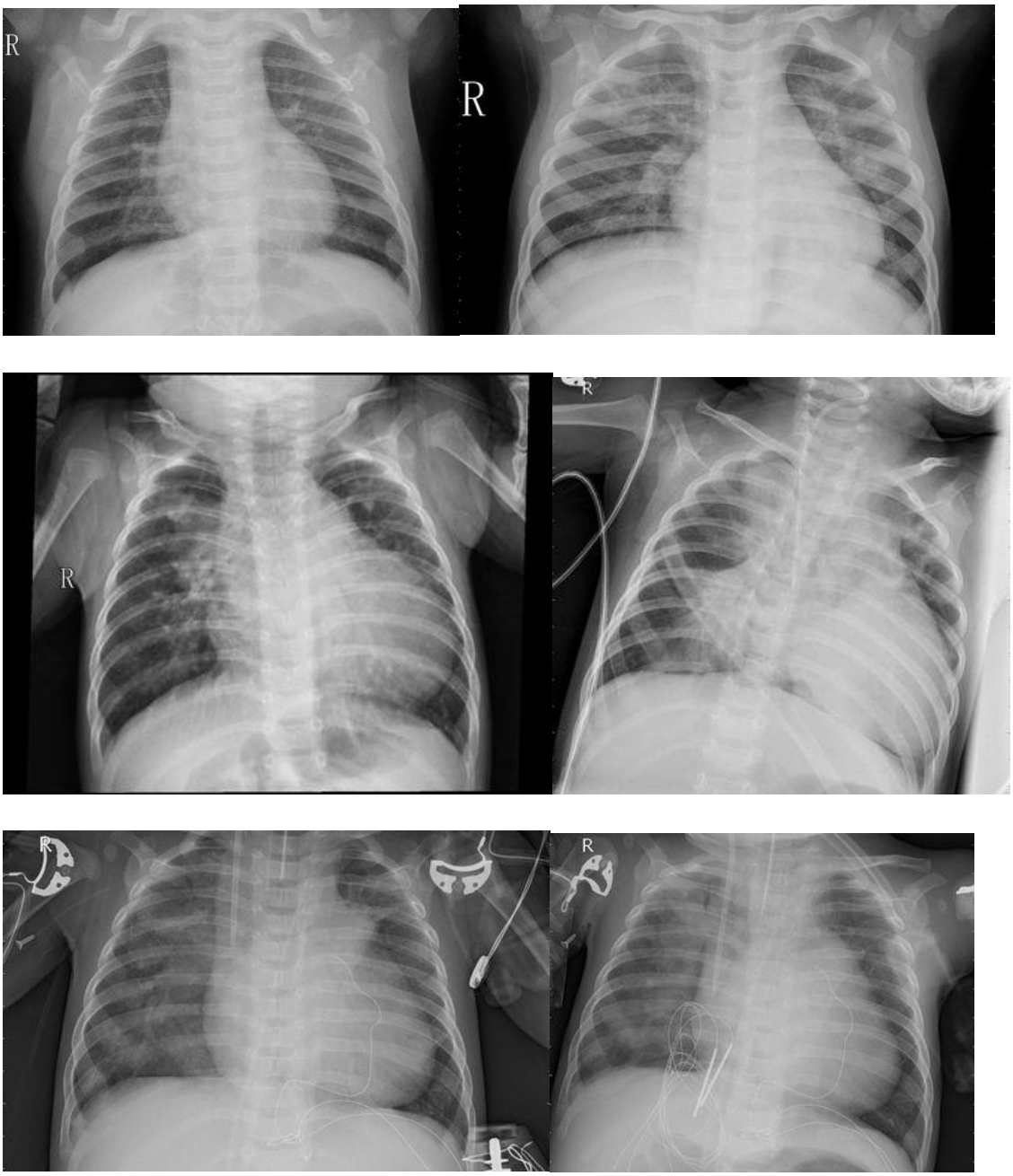

A


A

2. Normal:



3. Pneumonia:



5.5 Summary

Both these databases are extremely important as they will help us to train our model After the training process, it will be able to successfully classify breast cancer.

CHAPTER: 06

TENSORFLOW

6.1 Introduction

TensorFlow is one of the main elements of making deep learning models. We will use the TensorFlow library in building our own model.

6.2 What is TensorFlow?

TensorFlow is an open source python friendly library for numerical computation and large-scale machine learning. TensorFlow can train and run deep neural networks for different aspects like handwritten digit and letter classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation) based simulations. TensorFlow also supports production prediction at scale, with the same models used for training.

6.3 How TensorFlow works?

TensorFlow allows developers to create dataflow graphs structures that describe how data is related to each other and moves through a graph, or a series of processing nodes. The graphs can be visualized using the built-in tensor board. Each node in the graph represents a mathematical operation, and each connection or edge between nodes is a multidimensional data array, or tensor. TensorFlow gives the best performance and can iterate quickly through different models, also can train models faster and run more experiments.

6.4 How will we use TensorFlow?

In our experiment we will import the TensorFlow library, we will also import all the necessary libraries that is needed to run TensorFlow. We will split our dataset for training and validation. Then we will feed the dataset to our model to initiate the training process.

Fig: TensorFlow logo

6.5 Summary

TensorFlow will help us build our Neural Network Model which will be used to classify the mammogram pictures.

CHAPTER: 07

DEEP LEARNING

7.1 Introduction

We are going to use concepts of Deep Learning to build the model which will be used to classify Covid-19.

7.2 What is Deep Learning

Deep learning is a subset of machine learning in artificial intelligence that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network. IT utilizes a hierarchical level of artificial neural networks to carry out the process of machine learning. The artificial neural networks are built like the human brain, with neuron nodes connected together like a web. While traditional programs build analysis with data in a linear way, the hierarchical function of deep learning systems enables machines to process data with a nonlinear approach.

7.3 How does Deep Learning Work

The term “Neural Networks” comes from the network of neurons that are present in the human brain. In a neural network model, there are three layers.

1) Input Layer
2) Hidden Layer(s)
3) Output Layer(s)

The input layer receives input data. The input data could be of any form. The input layer passes the inputs to the first hidden layer. The hidden layers perform mathematical computations on the input layer. One of the challenges that is faced while creating a neural utilizes a hierarchical level of artificial neural networks to carry out the process of machine learning. The artificial neural networks are built like the human brain, with neuron nodes connected together like a web. While traditional programs build analysis with data in a linear way, the hierarchical function of deep

learning systems enables machines to process data with a nonlinear approach. Networks is deciding the number of hidden layers, as well as the number of neurons for each layer. The "Deep" in Deep Learning is a reference to having more than one hidden layer. And finally, there is the output layer, which shows the output of the model.

Each connection between the neurons are associated with a certain weight. This weight controls the importance of the input value. The initial weights are set randomly on their own. Each of the neurons have an Activation Function. Once a set of input data has passed through all the layers of the neural network, it returns the output data through the output layer. This goes on for quite a while.

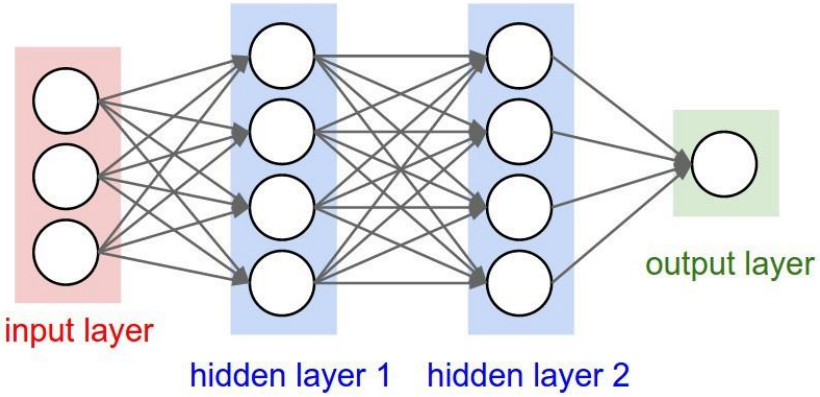


Figure: Neural Network Example

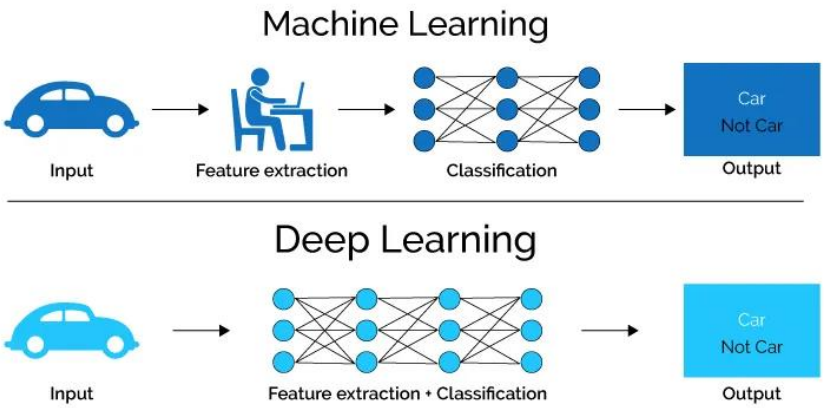


Fig: Basic difference between ML and DCNN

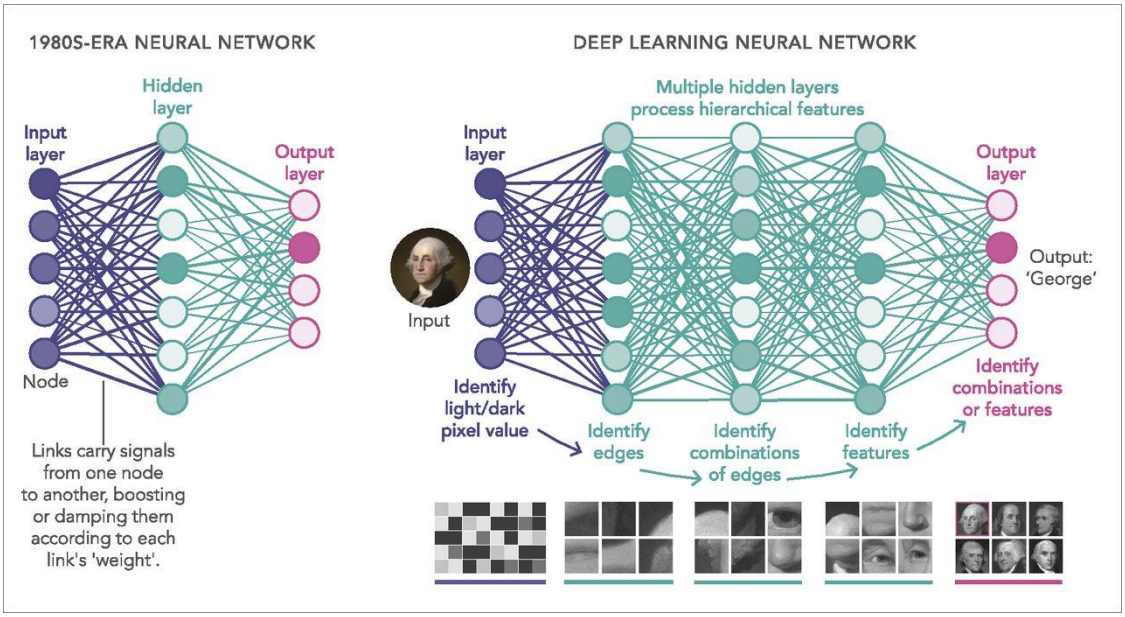


Fig: Regular Neural network vs. DCNN

7.4 Applications:

- Electromyography (EMG) recognition
- Image recognition
- Visual art processing
- Natural language processing
- Drug discovery and toxicology
- Customer relationship management
- Recommendation systems
- Bioinformatics
- Medical Image Analysis
- Mobile advertising
- Image restoration
- Financial fraud detection
- Military

7.5 Summary

Deep learning will be an essential part of our project as it will help us to develop the core foundation of our neural network model.

CHAPTER 08

KERAS

8.1 What is Keras?

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Keras is the high-level API of TensorFlow 2.0: an approachable, highly-productive interface for solving machine learning problems, with a focus on modern deep learning. It provides essential abstractions and building blocks for developing and shipping machine learning solutions with high iteration velocity.

8.2 How do we use it?

The core data structures of Keras are layers and models. The simplest type of model is the Sequential model, a linear stack of layers. For more complex architectures, you should use the Keras functional API, which allows to build arbitrary graphs of layers, or write models entirely from scratch via subclassing.

Here is the Sequential model:

```
from tensorflow.keras.models import Sequential

model = Sequential()
```

Stacking layers is as easy as .add():

```
from tensorflow.keras.layers import Dense

model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=10, activation='softmax'))
```

Once your model looks good, configure its learning process with .compile():

```
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

You can now iterate on your training data in batches:

```
# x_train and y_train are Numpy arrays --just like in the Scikit-Learn API.
model.fit(x_train, y_train, epochs=5, batch_size=32)
```

Evaluate your test loss and metrics in one line:

```
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
```

Or generate predictions on new data:

```
classes = model.predict(x_test, batch_size=128)
```

8.3 Summary

Keras is also a highly-flexible framework suitable to iterate on state-of-the-art research ideas. Keras follows the principle of progressive disclosure of complexity: it makes it easy to get started, yet it makes it possible to handle arbitrarily advanced use cases, only requiring incremental learning at each step.

CHAPTER: 09

Deploying the model

9.1 Introduction

In order for us to use a neural network model to classify COVID-19 CXRs, we first have to train the model. To deploy Covidnet-8, we used a web application framework called FLASK.

9.2 What is FLASK?

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks.

Flask offers suggestions, but doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use. There are many extensions provided by the community that make adding new functionality easy.

9.3 Deploying Covidnet-8:

Frontend of FLASK application:

UI:

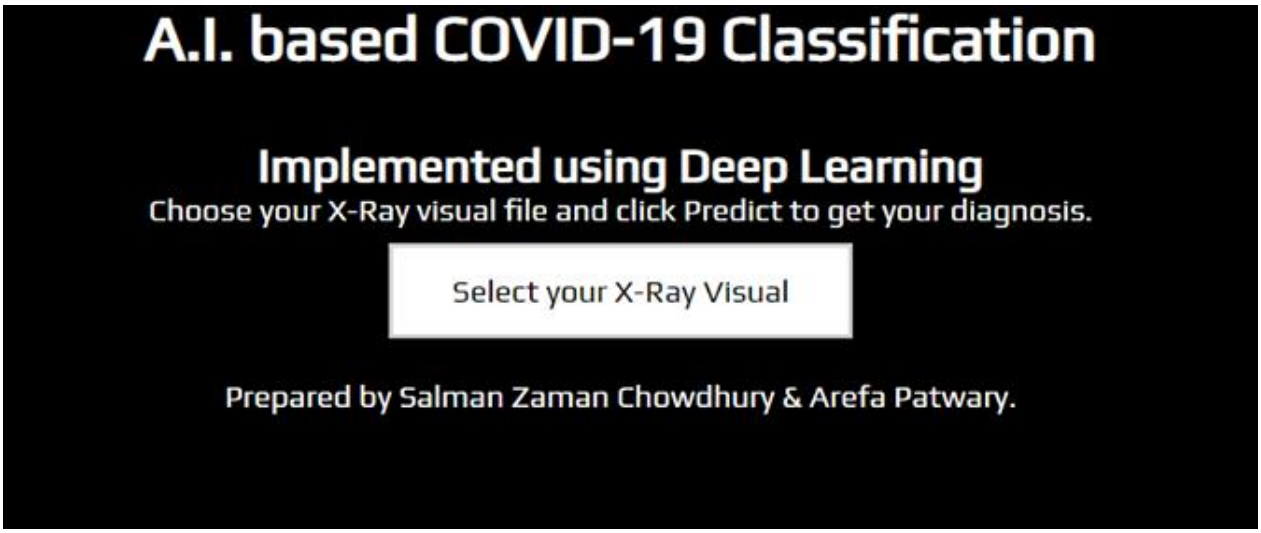


Fig: UI of the FLASK frontend

Selecting an Image for classification:

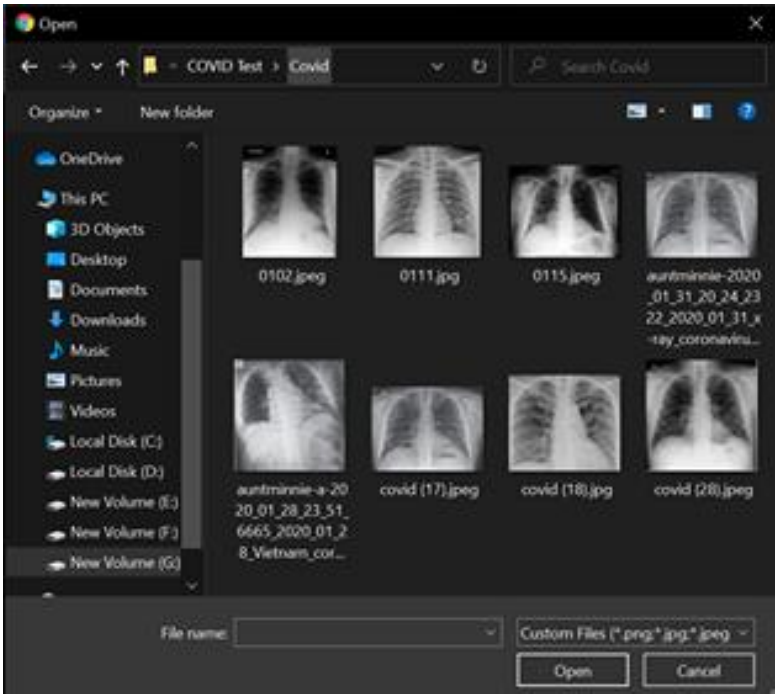


Fig: Selecting an image from test dataset for output

After selecting, click on the predict button to predict:

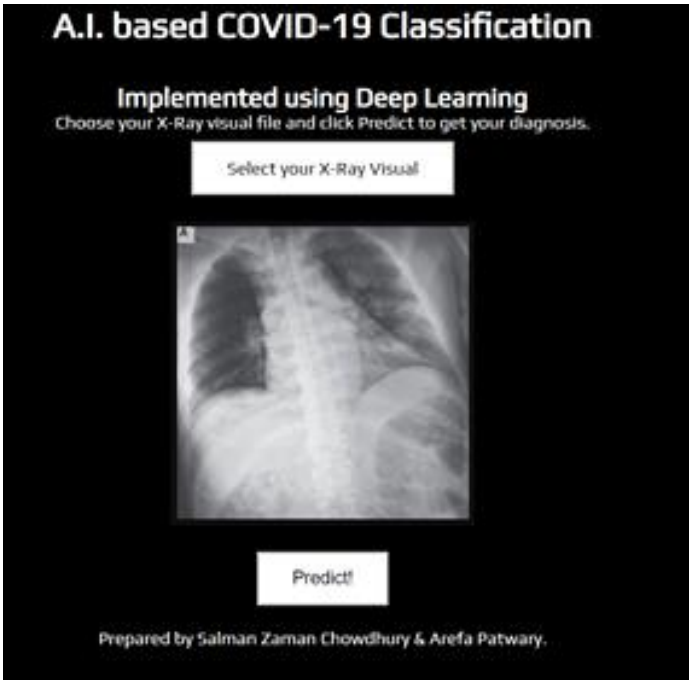


Fig: Predict output

The model will classify the given Image and show the diagnosis of the lung's status:

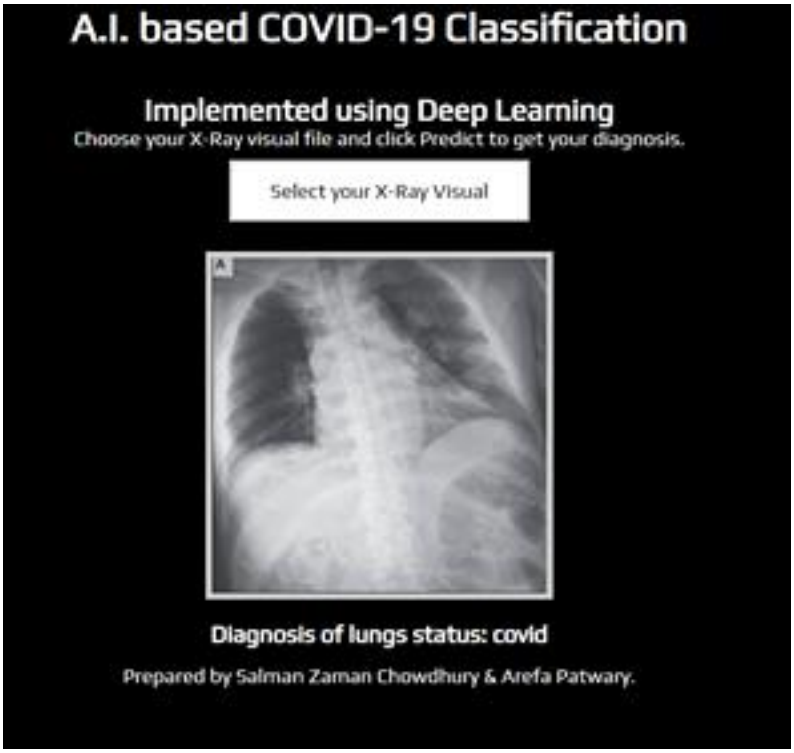


Fig: Output

Backend of FLASK application:

```
from flask import Flask, redirect, url_for, request, render_template
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer

# Import Keras dependencies
from tensorflow.keras.models import model_from_json
from tensorflow.python.framework import ops
ops.reset_default_graph()
from keras.preprocessing import image

# Import other dependecies
import numpy as np
import h5py
from PIL import Image
import PIL
import os

#::: Flask App Engine :::
# Define a Flask app
app = Flask(__name__)

# ::: Prepare Keras Model :::
# Model files
MODEL_ARCHITECTURE = '2ndownmodel.json'
MODEL_WEIGHTS = '2ndownmodel.h5'

# Load the model from external files
json_file = open(MODEL_ARCHITECTURE)
loaded_model_json = json_file.read()
json_file.close()
model = model_from_json(loaded_model_json)

# Get weights into the model
model.load_weights(MODEL_WEIGHTS)
print('Model loaded')


# ::: MODEL FUNCTIONS :::
def model_predict(img_path, model):
        '''
```

```
        Args:
                -- img_path : an URL path where a given image is stored.
                -- model : a given Keras CNN model.
        '''

        IMG = image.load_img(img_path)
        print(type(IMG))

        # Pre-processing the image
        IMG= IMG.resize((256, 256))
        print(type(IMG))
        IMG = np.array(IMG)
        print(IMG.shape)
        IMG= np.true_divide(IMG, 3)
        IMG = IMG.reshape(1,256,256,3)
        print(type(IMG), IMG.shape)

        print(model)

        model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='rmsprop')
        prediction = model.predict_classes(IMG)

        return prediction


# ::: FLASK ROUTES
@app.route('/', methods=['GET'])
def index():
        # Main Page
        return render_template('index.html')

@app.route('/predict', methods=['GET', 'POST'])
def upload():

        # Constants:
        classes = {'TRAIN': ['Covid','Normal','Pneumonia'],
                   'VALIDATION': ['Covid','Normal','Pneumonia'],
                   'TEST': ['Covid','Normal','Pneumonia']}


              if request.method == 'POST':

                      # Get the file from post request
                      f = request.files['file']

                      # Save the file to ./uploads
                      basepath = os.path.dirname(__file__)
                      file_path = os.path.join(
                              basepath, 'uploads', secure_filename(f.filename))
                      f.save(file_path)

                      # Make a prediction
                      prediction = model_predict(file_path, model)

                      predicted_class = classes['TRAIN'][prediction[0]]
                      print('We think that is {}.'.format(predicted_class.lower()))

                      return str(predicted_class).lower()

      if __name__ == '__main__':
              app.run(debug = True)
```

Fig: Backend of FLASK

9.4 Summary

Since our model is not very large, FLASK is a good tool to start with for deployment. But as the model gets bigger it is better to switch to other frameworks such as Django.

CHAPTER: 10

Project code

10.1 Introduction

The DCNN model named Covidnet-8 is a custom model with 8 layers in total. It has 6 Conv 2D layers and 2 dense layers. The activation function used in every layer is Relu except in the last dense layer where SoftMax is used.

Other methods such as lr=0.001, Adam optimizer, L2 Regulariser, dropout of 0.3, data augmentation has been applied on it in order to minimize loss, avoid overfitting and increase accuracy.

10.2 Project code

Import libraries:

```
import numpy as np
import keras
from keras import backend as K
from keras.models import Sequential
from keras import regularizers
from keras.layers import Activation, Dropout
import tensorflow as tf
from keras.preprocessing import image
from keras.layers.core import Dense, Flatten
from keras.layers.convolutional import MaxPooling2D
from keras.layers import Convolution2D as Conv2D
from keras.optimizers import Adam
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import *
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix
from keras.utils.vis_utils import plot_model
import itertools
import matplotlib.pyplot as plt
%matplotlib inline
```

Fig: Import libraries

Mount dataset from Google drive:

```
[ ]  from google.colab import drive
     drive.mount('/content/gdrive')

     Mounted at /content/gdrive
```

Fig: Mount data

```
[4]  train_path = '/content/gdrive/My Drive/Colab Notebooks/Covid-19 Kaggle-Copy-499-dataset/Covid-19 Kaggle - Copy/Covid-19 Kaggle/COVID Train'
     # test_path = '/content/drive/My Drive/Colab Notebooks/Covid-19 Kaggle-Copy-499-dataset/Covid-19 Kaggle - Copy/Covid-19 Kaggle/COVID Test'
     valid_path = '/content/gdrive/My Drive/Colab Notebooks/Covid-19 Kaggle-Copy-499-dataset/Covid-19 Kaggle - Copy/Covid-19 Kaggle/COVID Valid'
```

```
[ ]  train_batches = ImageDataGenerator(rotation_range=15).flow_from_directory(train_path,target_size=(256, 256),  classes=['Covid','Normal','Pneumonia'],batch_size=32)
     valid_batches = ImageDataGenerator().flow_from_directory(valid_path,target_size=(256, 256), classes=['Covid','Normal','Pneumonia'],batch_size=16)

     Found 6900 images belonging to 3 classes.
     Found 1784 images belonging to 3 classes.
```

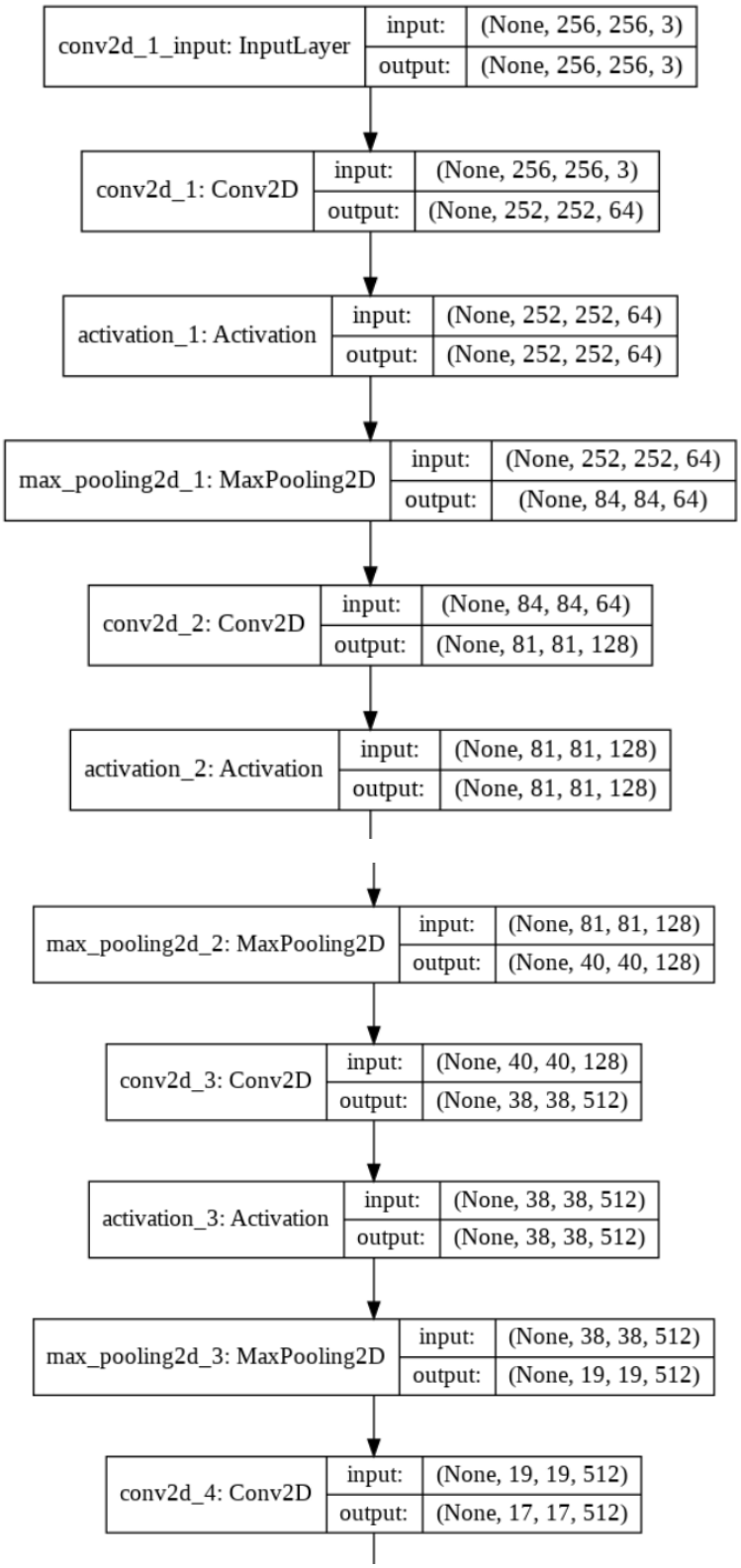
```
def plots(ims, figsize=(12,6), rows=1, interp=False, titles=None):
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims).astype(np.uint8)
        if (ims.shape[-1] != 3):
            ims = ims.transpose((0,2,3,1))
    f = plt.figure(figsize=figsize)
    cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
    for i in range(len(ims)):
        sp = f.add_subplot(rows, cols, i+1)
        sp.axis('Off')
        if titles is not None:
            sp.set_title(titles[i], fontsize=16)
        plt.imshow(ims[i], interpolation=None if interp else 'none')
```

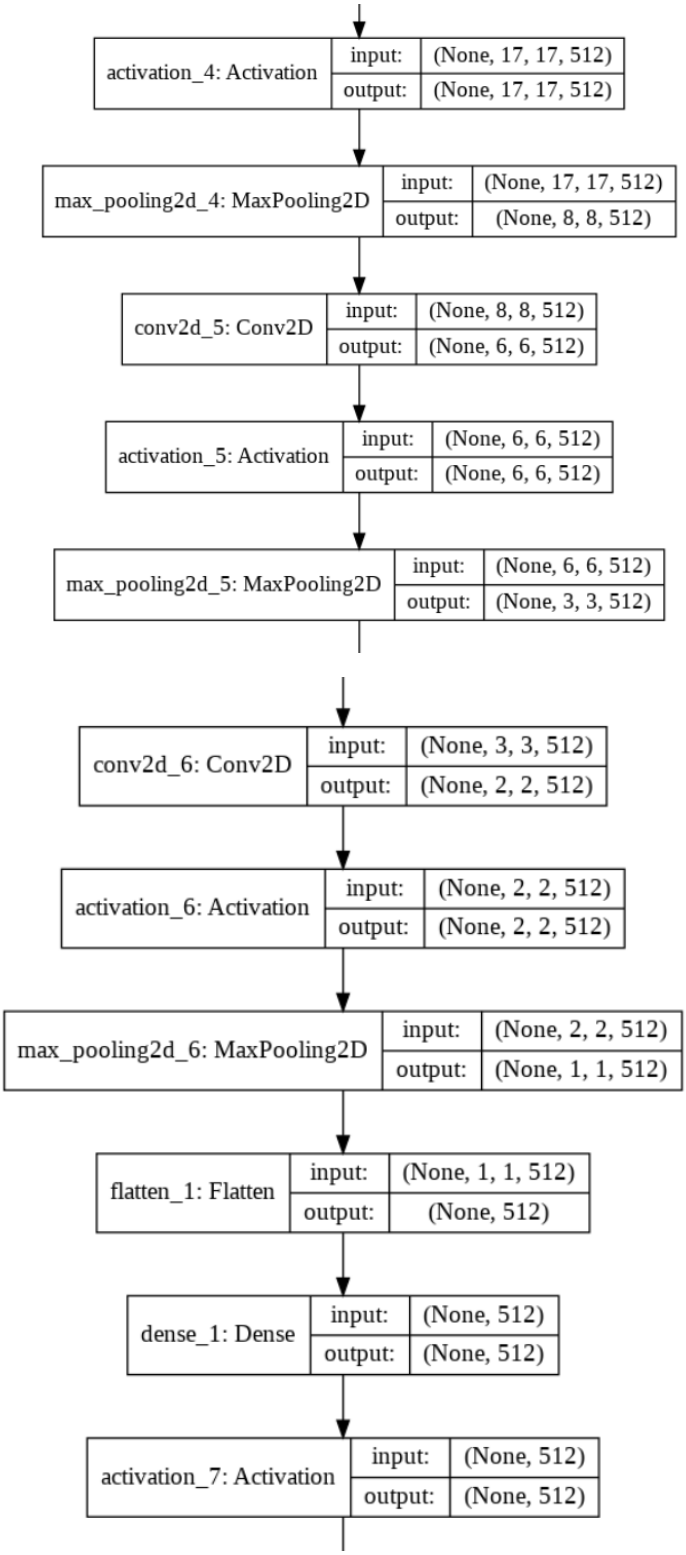
```
[ ]  imgs, labels = next(train_batches)
```

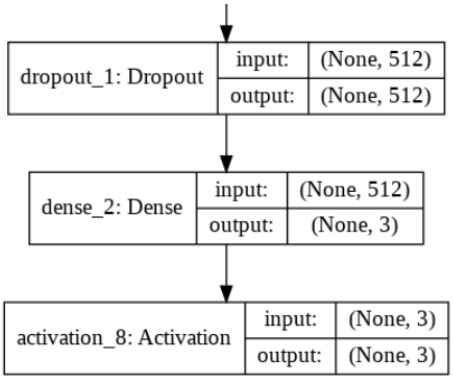
```
[ ]  plots(imgs, titles=labels)

     /usr/local/lib/python3.6/dist-packages/matplotlib/text.py:1165: FutureWarning: elementwise compari
       if s != self._text:
     [0.[0.[1.]0.]0.]1.]0.]1.]0.]0.]1.]1.]0.]1.]0.]1.]0.]1.]0.]1.]0.]1.]0.]1.]0.]1.]0.]1.]0.]1.]0.]1.]0.]1.]
```

```
model = Sequential()
model.add(Conv2D(64, (5, 5), input_shape=(256, 256,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D((3, 3)))

model.add(Conv2D(128, (4, 4)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(512, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(512, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(512, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(512, (2, 2)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(512, kernel_regularizer=regularizers.l2(0.02)))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(3))
model.add(Activation('softmax'))
```

```
plot_model(model, show_shapes=True, show_layer_names=True)
```


conv2d_1_input: InputLayer
input: (None, 256, 256, 3)
output: (None, 256, 256, 3)
conv2d_1: Conv2D
input: (None, 256, 256, 3)
output: (None, 252, 252, 64)
activation_1: Activation
input: (None, 252, 252, 64)
output: (None, 252, 252, 64)
max_pooling2d_1: MaxPooling2D
input: (None, 252, 252, 64)
output: (None, 84, 84, 64)
conv2d_2: Conv2D
input: (None, 84, 84, 64)
output: (None, 81, 81, 128)
activation_2: Activation
input: (None, 81, 81, 128)
output: (None, 81, 81, 128)
max_pooling2d_2: MaxPooling2D
input: (None, 81, 81, 128)
output: (None, 40, 40, 128)
conv2d_3: Conv2D
input: (None, 40, 40, 128)
output: (None, 38, 38, 512)
activation_3: Activation
input: (None, 38, 38, 512)
output: (None, 38, 38, 512)
max_pooling2d_3: MaxPooling2D
input: (None, 38, 38, 512)
output: (None, 19, 19, 512)
conv2d_4: Conv2D
input: (None, 19, 19, 512)
output: (None, 17, 17, 512)


activation_4: Activation
input: (None, 17, 17, 512)
output: (None, 17, 17, 512)
max_pooling2d_4: MaxPooling2D
input: (None, 17, 17, 512)
output: (None, 8, 8, 512)
conv2d_5: Conv2D
input: (None, 8, 8, 512)
output: (None, 6, 6, 512)
activation_5: Activation
input: (None, 6, 6, 512)
output: (None, 6, 6, 512)
max_pooling2d_5: MaxPooling2D
input: (None, 6, 6, 512)
output: (None, 3, 3, 512)
conv2d_6: Conv2D
input: (None, 3, 3, 512)
output: (None, 2, 2, 512)
activation_6: Activation
input: (None, 2, 2, 512)
output: (None, 2, 2, 512)
max_pooling2d_6: MaxPooling2D
input: (None, 2, 2, 512)
output: (None, 1, 1, 512)
flatten_1: Flatten
input: (None, 1, 1, 512)
output: (None, 512)
dense_1: Dense
input: (None, 512)
output: (None, 512)
activation_7: Activation
input: (None, 512)
output: (None, 512)



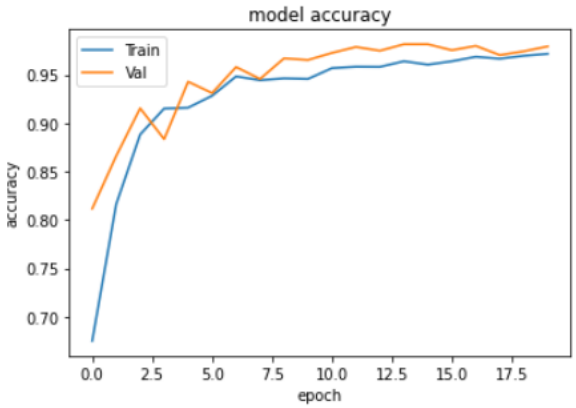

```
regularizer = tf.keras.regularizers.l2(0.01)

for layer in model.layers:
    for attr in ['kernel_regularizer']:
        if hasattr(layer, attr):
          setattr(layer, attr, regularizer)
```

```
model.compile(Adam(lr=0.001),loss='categorical_crossentropy',metrics=['accuracy'])
```

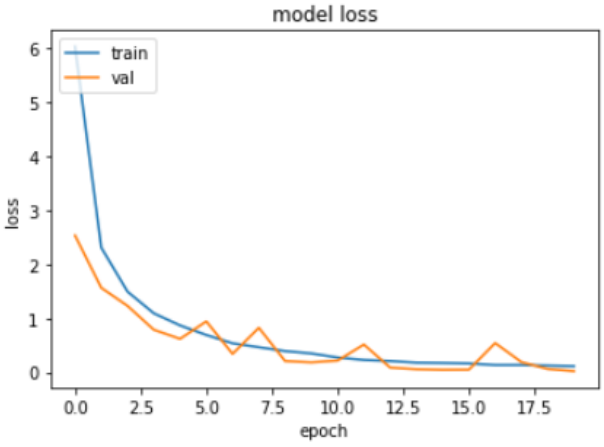
```
history = model.fit_generator(train_batches,
                    validation_data=valid_batches, epochs=20, verbose=2)
```

```
Epoch 1/20
 - 4051s - loss: 6.0192 - accuracy: 0.6751 - val_loss: 2.5296 - val_accuracy: 0.8117
Epoch 2/20
 - 261s - loss: 2.3061 - accuracy: 0.8162 - val_loss: 1.5620 - val_accuracy: 0.8660
Epoch 3/20
 - 260s - loss: 1.4900 - accuracy: 0.8883 - val_loss: 1.2278 - val_accuracy: 0.9154
Epoch 4/20
 - 262s - loss: 1.0907 - accuracy: 0.9151 - val_loss: 0.7868 - val_accuracy: 0.8834
Epoch 5/20
 - 261s - loss: 0.8682 - accuracy: 0.9159 - val_loss: 0.6185 - val_accuracy: 0.9428
Epoch 6/20
 - 259s - loss: 0.6899 - accuracy: 0.9283 - val_loss: 0.9413 - val_accuracy: 0.9311
Epoch 7/20
 - 264s - loss: 0.5361 - accuracy: 0.9483 - val_loss: 0.3405 - val_accuracy: 0.9580
Epoch 8/20
 - 268s - loss: 0.4631 - accuracy: 0.9443 - val_loss: 0.8231 - val_accuracy: 0.9456
Epoch 9/20
 - 268s - loss: 0.3918 - accuracy: 0.9462 - val_loss: 0.2074 - val_accuracy: 0.9669
Epoch 10/20
 - 259s - loss: 0.3482 - accuracy: 0.9457 - val_loss: 0.1837 - val_accuracy: 0.9652
Epoch 11/20
 - 259s - loss: 0.2738 - accuracy: 0.9568 - val_loss: 0.2168 - val_accuracy: 0.9725
Epoch 12/20
 - 259s - loss: 0.2286 - accuracy: 0.9583 - val_loss: 0.5136 - val_accuracy: 0.9787
Epoch 13/20
 - 261s - loss: 0.2077 - accuracy: 0.9581 - val_loss: 0.0874 - val_accuracy: 0.9748
Epoch 14/20
 - 259s - loss: 0.1782 - accuracy: 0.9639 - val_loss: 0.0568 - val_accuracy: 0.9815
Epoch 15/20
 - 258s - loss: 0.1724 - accuracy: 0.9604 - val_loss: 0.0472 - val_accuracy: 0.9815
Epoch 16/20
 - 258s - loss: 0.1634 - accuracy: 0.9639 - val_loss: 0.0492 - val_accuracy: 0.9753
Epoch 17/20
 - 256s - loss: 0.1344 - accuracy: 0.9686 - val_loss: 0.5410 - val_accuracy: 0.9798
Epoch 18/20
 - 257s - loss: 0.1346 - accuracy: 0.9665 - val_loss: 0.1890 - val_accuracy: 0.9703
Epoch 19/20
 - 257s - loss: 0.1244 - accuracy: 0.9696 - val_loss: 0.0615 - val_accuracy: 0.9742
Epoch 20/20
 - 256s - loss: 0.1135 - accuracy: 0.9716 - val_loss: 0.0216 - val_accuracy: 0.9793
```

```
import keras
from matplotlib import pyplot as plt
#history = model1.fit(train_x, train_y,validation_split = 0.1, epochs=50, batch_size=4)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```




```
[ ]  plt.plot(history.history['loss'])
     plt.plot(history.history['val_loss'])
     plt.title('model loss')
     plt.ylabel('loss')
     plt.xlabel('epoch')
     plt.legend(['train', 'val'], loc='upper left')
     plt.show()
```



```
[ ]  model.save('/content/gdrive/My Drive/Colab Notebooks/2ndownmodel.h5')
```

```
[ ]  # Save the Model to JSON
     model_json = model.to_json()
     with open('/content/gdrive/My Drive/Colab Notebooks/2ndownmodel.json', 'w') as json_file:
         json_file.write(model_json)

     print('Model saved to the disk.')
```

```
Model saved to the disk.
```

```
[10]
     # ----------------------------------------------------------------------
     # Load saved model and its weights
     '''
     >> Model weights are saved to HDF5 format.
     >> The model structure can be described and saved using two different formats: JSON and YAML.
     '''

     # Import dependencies
     from keras.optimizers import Adam
     from tensorflow.keras.models import model_from_json
     from tensorflow.python.framework import ops
     ops.reset_default_graph()
     import h5py
     from PIL import Image
     import PIL
     # from vb100_utils import *
```

```
[11] # Get the architecture of CNN
     json_file = open('/content/gdrive/My Drive/Colab Notebooks/2ndownmodel.json')
     loaded_model_json = json_file.read()
     json_file.close()
     loaded_model = model_from_json(loaded_model_json)

     # Get weights into the model
     loaded_model.load_weights('/content/gdrive/My Drive/Colab Notebooks/2ndownmodel.h5')
```

```
[14] # Define optimizer and run
     opt = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0)
     loaded_model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='rmsprop')



     '''
     Important Note! For this block optimizer is entered manualy as Tensorflow object.
     For future, need to change it for include it as variable with full set of
     parameters as Tensorflow variable.

     '''

     IMG = Image.open('/content/gdrive/My Drive/Colab Notebooks/Covid-19 Kaggle-Copy-499-dataset/Covid-19 Kaggle
     print(type(IMG))
     x = image.img_to_array(IMG)
     print(x.shape)


     IMG = IMG.resize((256, 256))
     IMG = np.array(IMG)
     print('po array = {}'.format(IMG.shape))
     IMG = np.true_divide(IMG,3)
     IMG = IMG.reshape(1, 256, 256, 3)
     print(type(IMG), IMG.shape)

     predictions = loaded_model.predict(IMG)

     print(loaded_model)
     predictions_c = loaded_model.predict_classes(IMG)
```

```
<class 'PIL.JpegImagePlugin.JpegImageFile'>
(448, 425, 3)
po array = (256, 256, 3)
<class 'numpy.ndarray'> (1, 256, 256, 3)
<tensorflow.python.keras.engine.sequential.Sequential object at 0x7ffac18328d0>
```

```
[15] classes = {'TRAIN': ['Covid','Normal','Pneumonia'],
                'VALIDATION': ['Covid','Normal','Pneumonia'],
                'TEST': ['Covid','Normal','Pneumonia']}

     predicted_class = classes['TRAIN'][predictions_c[0]]
     print('We think that is {}.'.format(predicted_class.lower()))
     print(predictions * 100)
```

```
We think that is covid.
[[99.06876     0.33384025  0.59738815]]
```

10.3 Summary

We are working further to improve out model and make it more efficient.

CHAPTER: 11

FUTURE WORK

In future we plan on training our model with single channel images, to support multichannel image classification.

We will also apply network compression methods such as pruning, so that we can deploy it in smaller devices such as phone and tabs where radiologists can easily access it from anywhere.

We also have plans for collecting and working with locally collected real-life data. We will compare COVID-19 with various other respiratory diseases using chest x-ray.

Our ultimate goal is to publish a paper on this project.

CHAPTER 12

PRACTICALITY

The project has a lot of practical benefits. It is very accurate and effective thus consumes less time to screen a CXR by a radiologist.

Since it is a web application with a very minimal hardware, it can be accessed from anywhere.

Given enough time to further on it, it can also be deployed in phone or tablets, so that doctors, radiologists as well as the patients themselves can access it and quick check their reports.

Most effective in third- world countries where medical facilities, and quality doctors are limited.

CHAPTER 12

CONCLUSION

The goal of this work is to detect the COVID-19 regions from a CXR and to whether it is COVID positive or Normal or Pneumonia. So far, we have used a custom DCNN model that has been optimized to classify three types of CXRs. We have been able to successfully run the model and deploy it locally to predict the output. Now our plan is to use all the knowledge that we have acquired and improve our model even more and make it efficient, practical and easily accessible. We will continuously work on the model to improve its accuracy as much as possible and compare our models with pre-existing models.

BIBLIOGRAPHY

1. https://tbsnews.net/feature/panorama/artificial-intelligence-weapon-fight-coronavirus-98632
2. https://arxiv.org/abs/2003.09871
3. https://towardsdatascience.com/using-deep-learning-to-detect-ncov-19-from-x-ray-images-1a89701d1acd
4. https://www.medrxiv.org/content/medrxiv/early/2020/02/18/2020.02.14.20023028.full.pdf
5. https://link.springer.com/article/10.1007/s10096-020-03901-z?fbclid=IwAR3JxHplpoBdlsTKyl5WfsBCs2aw8Lq-IdiqsX6U8y1QV-gqRJLRLG3ifOI
6. https://journals.plos.org/plosone/article?id=10.1371%2Fjournal.pone.0235187&fbclid=IwAR33uqegiptlodgOVAZIXplpG396_kTJfQgRzwtp8FuAJu4rJQVjxh88_z0
7. https://stats.stackexchange.com/questions/91044/how-to-calculate-precision-and-recall-in-a-3-x-3-confusion-matrix?fbclid=IwAR00LCuwaOgQxoQP93QmROOVaajwTaRkiyvfVmJ_wDpnCRxp3O31wqyL0j8
8. https://link.springer.com/article/10.1007/s13337-020-00599-7
9. https://www.cdc.gov/coronavirus/2019-ncov/symptoms-testing/testing.html
10. https://www.cdc.gov/coronavirus/2019-ncov/testing/diagnostic-testing.html
11. https://www.cdc.gov/coronavirus/2019-ncov/testing/serology-overview.html
12. https://www.fda.gov/consumers/consumer-updates/coronavirus-testing-basics
13. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7144809/
14. https://apsjournals.apsnet.org/doi/10.1094/PHYTO-02-18-0067-R
15. https://link.springer.com/article/10.1007/s00216-020-02889-x
16. https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge
17. https://keras.io/about/
18. https://www.investopedia.com/terms/d/deep-learning.asp#:~:text=Deep%20learning%20is%20a%20subset,learning%20or%20deep%20neural%20network.
19. https://palletsprojects.com/p/flask/
20. https://www.medrxiv.org/content/medrxiv/early/2020/02/18/2020.02.14.20023028.full.pdf
21. https://tbsnews.net/feature/panorama/artificial-intelligence-weapon-fight-coronavirus-98632
22. file:///D:/499%20report%20samples/2020.03.30.20047456v3.full.pdf

23. https://www.nature.com/articles/s41467-020-17971-2
24. https://www.researchgate.net/publication/340472082_Automatic_X-ray_COVID-19_Lung_Image_Classification_System_based_on_Multi-Level_Thresholding_and_Support_Vector_Machine?fbclid=IwAR1iC1GDqxXWuKnKN_JJJ7HtpDQow3uets42qcz89XuiofXc61CQE7IAbJ0
25. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7187882/?fbclid=IwAR0-k5oAhWQy48DU01mCA7n3LAUo29uWuMxjfa8NJoqmFMHhoDdZMAGNR6A
26. https://iq.opengenus.org/convolutional-neural-networks/
27. https://iq.opengenus.org/vgg19-architecture/
28. https://www.researchgate.net/figure/llustration-of-the-network-architecture-of-VGG-19-model-conv-means-convolution-FC-means_fig2_325137356
29. https://iq.opengenus.org/resnet/
30. https://www.freecodecamp.org/news/https-medium-com-hadrienj-preprocessing-for-deep-learning-9e2b9c75165c/