

**DEEP LEARNING BASED MODEL FOR THE
AUTOMATIC IDENTIFICATION OF COVID-19
DISEASE USING CT SCAN IMAGE**

A Project report submitted in partial fulfilment of the requirements for the degree of B. Tech in
Electrical Engineering By

Shashwata Roy(EE2017/016)

Soumyajit Roy(EE2017/018)

Somtirtha pal(EE2017/023)

Amit Mondal(EE2017/033)

Under the supervision of

Professor (Dr.) Alok kole

Department of Electrical Engineering, RCCIIT



Department of Electrical Engineering

RCC INSTITUTE OF INFORMATION TECHNOLOGY

CANAL SOUTH ROAD, BELIAGHATA, KOLKATA - 700015, WEST BENGAL

Maulana Abul Kalam Azad University of Technology (MAKAUT)

© 2021

PREFACE

This report comprises the summary of work we namely, Shashwata Roy, Soumyajit Roy, Somtirtha pal and Amit Mondal have achieved during our final year project. The task has been chosen to carry out during this year is *DEEP LEARNING BASED MODEL FOR THE AUTOMATIC IDENTIFICATION OF COVID-19 DISEASE USING CT IMAGE HISTORY*. The effort is conducted under the supervision of Prof. ALOK KOLE, Department of Electrical Engineering, RCC Institute of Information Technology.

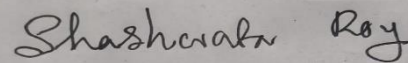
Here a deep learning-based CNN model is used for a very good performance with a low computational cost. The performance result will be based on the collected data set of CT images of minimum hundreds normal and COVID-19 patients and the performance analysis will be done in terms of estimated class probabilities

ACKNOWLEDGEMENT

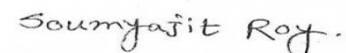
It is my great fortune that I have got the opportunity to carry out the project work under the supervision of Prof. Alok Kole in the department of Electrical Engineering, RCC Institute of Information Technology (RCCIIT) Canal South Road, Beliaghata, Kolkata-700015, affiliated to Maulana Abul Kalam Azad University of Technology (MAKAUT), West Bengal, India. I express my sincere thanks and deepest sense of gratitude to my guide for his constant support, unparalleled guidance and limitless encouragement.

We wish to convey my gratitude to Prof. (Dr.) Debasish Mondal, HOD, Department of Electrical Engineering, RCCIIT and to the authority of RCCIIT for providing all kind of infrastructural facility towards the research work.


We would also like to convey my gratitude to all the faculty members and the staffs of the Department of Electrical Engineering, RCCIIT for their wholehearted cooperation to make this work turn into reality. Thanks to fellow members of our group for working as a team.



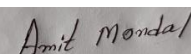
Shashwata Roy



Soumyajit Roy



Somtirtha pal



Amit Mondal

Name and Signature Of the Student

Place: Kolkata

Date: 06/07/2021



Department of Electrical Engineering
RCC INSTITUTE OF INFORMATION TECHNOLOGY
GROUND FLOOR, NEW BUILDING,
CANAL SOUTH ROAD, BELIAGHATA, KOLKATA - 700015, WEST BENGAL

CERTIFICATE

To whom it may concern

This is to certify that the project work entitled *DEEP LEARNING BASED MODEL FOR THE AUTOMATIC IDENTIFICATION OF COVID-19 DISEASE USING CT IMAGE HISTORY* is the bona fide work carried out by **Shashwata Roy(11701617037)**, **Soumyajit Roy(11701617033)**, **Somtirtha Pal (11701618036)**, **Amit Mondal (11701618075)** a student of B. Tech in the Dept. of Electrical Engineering, RCC Institute of Information Technology (RCCIIT), Canal South Road, Beliaghata, Kolkata-700015, affiliated to Maulana Abul Kalam Azad University of Technology (MAKAUT), West Bengal, India, during the academic year 2020- 21, in partial fulfilment of the requirements for the degree of Bachelor of Technology in Electrical Engineering and that this project has not been submitted previously for the award of any other degree, diploma or fellowship.

Signature of the Guide

Name: Professor (Dr.) Alok Kole

Designation: Professor, Dept. of EE

HoD, Dept. of EE
RCC Institute of Information Technology
Kolkata-700015

Signature of HOD

Name: Dr. Debasish Mondal

Designation: HOD, Dept. of

EE

INDEX PAGE

Page no.

• Abstract	6
• Introduction	7-9
• Literature Review	10-11
• Theory	12-18
• Implementation	19
• Methodology	20-23
• Model Architecture	24
• Software Code	25-28
• Performance Analysis	29-33
• Discussion	34
• Future Scope	35
• Reference	36

ABSTRACT

The detection of coronavirus (COVID-19) is now a critical task for the medical practitioner. The coronavirus spread so quickly between people and approaches more than 194,420,580 people worldwide. Therefore, it is very much essential to implement an automatic detection system as a quick alternative diagnosis option to prevent COVID-19 spreading among people. In this proposed project, a robust deep learning based methodology is proposed to be developed for the automatic identification of coronavirus infected patient and also to distinguish COVID-19 from community acquired pneumonia (CAP) using chest CT images. A Convolution Neural Network (CNN) model architecture and support vector machine (SVM) classifies the corona affected CT images from others using the deep feature. In this project, a deep learning based CNN and SVM model architecture which is trained on ImageNet database are proposed to be developed to extract the visual features from volumetric chest CT image for the identification of COVID-19. This methodology is beneficial for the medical practitioner for diagnosis of coronavirus infected patient.

INTRODUCTION

The virus called the severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) had been discovered in late 2019. The virus which originated in China became a cause of a disease known as Corona Virus Disease 2019 or COVID-19. The World Health Organization (WHO) declared the disease as a pandemic in March 2020 . According to the reports issued and updated by global healthcare authorities and state governments, the pandemic affected millions of people globally. The most serious illness caused by COVID-19 is related to the lungs such as pneumonia. The symptoms of the disease can vary and include dyspnea, high fever, runny nose, and cough. These cases can most commonly be diagnosed using chest X-ray imaging analysis for the abnormalities .

X-radiation or X-ray is an electromagnetic form of penetrating radiation. These radiations are passed through the desired human body parts to create images of internal details of the body part. The X-ray image is a representation of the internal body parts in black and white shades. X-ray is one of the oldest and commonly used medical diagnosis tests. Chest X-ray is used to diagnose the chest-related diseases like pneumonia and other lung diseases , as it provides the image of the thoracic cavity, consisting of the chest and spine bones along with the soft organs including the lungs, blood vessels, and airways. The X-ray imaging technique provides numerous advantages as an alternative diagnosis procedure for COVID-19 over other testing procedures. These benefits include its low cost, the vast availability of X-ray facilities, non-invasiveness, less time consumption, and device affordability. Thus, X-ray imaging may be considered a better candidate for the mass, easy, and quick diagnosis procedure for a pandemic like COVID-19 considering the current global healthcare crisis.

Deep learning and ANNs have endorsed an exponential research focus over the last decade. The deep ANNs have outperformed other conventional models on many essential

benchmarks. Thus, ANNs have generally proved to be the state-of-the-art technology across a wide range of application areas, including NLP, speech recognition, image processing, biological sciences, and other commercial as well as academic areas. The advancement of ANNs has massive potential in healthcare applications, specifically in medical data analysis, diagnosis through medical image processing, and analysis. As seen in recent times, various parts of the world face the healthcare crisis both in terms of the needed number of healthcare professionals and testing equipment. Considering the present pandemic situation, there is an appurtenant relationship between the detection of COVID-19 cases and chest X-ray image analysis and classification. In this work, an automatic diagnostic system has been developed using CNN which uses chest X-ray analysis results to diagnose whether a person is COVID-19-affected or normal. Preliminary analysis of this study has shown promising results in terms of its accuracy and other performance parameters to diagnose the disease in a cost-effective and time-efficient manner. This study used CNN with extra layers to improve the COVID-19 X-ray image classification accuracy. In neural networks, the CNN structure is specially designed to process the two-dimensional image tasks although it can also be used in one- and three-dimensional data. CNN is a type of DNN, inspired by the visual system of the human brain, and is most commonly used in the analysis of visual imagery. To train the CNN model, first, the dataset has been obtained from GitHub . Since the dataset obtained for training the model was very small in size and imbalanced, to solve the problem of having very-limited-sized X-ray image dataset, it has been extended using data augmentation techniques to increase its size and also to make the model training feature rich. Image flipping and rotation at different angles have been used to generate more data. For dataset balancing in terms of proportion of images with different class labels, the dataset has been further extended with some more image instances of the minority class. After data augmentation and dataset balancing, the CNN model has been trained using a total of 800 images (400 COVID-19 and 400

normal) and then the model has been tested by using a test set. The CNN model performance evaluation has then been done using different performance metrics. These metrics include accuracy, precision, sensitivity, specificity, ROC and F_1 score. Later, the proposed CNN model has also been tested using an independent dataset obtained from the IEEE data port for independent validation of the proposed CNN model. Various machine learning models have also been used for the comparative performance analysis in comparison with the proposed CNN model to show its significance over these models.

The following are some of the key findings of this study:(i)CNN with extra convolutional layers (e.g., six layers have been used in the CNN proposed in this study) performs best in COVID-19 diagnosis(ii)CNN models require a sufficient amount of images for efficient and more accurate image classification(iii)Data augmentation techniques are very effective to improve the CNN model performance remarkably by generating more data from an existing limited-size dataset(iv)Data augmentation is also effective in image classification as it gives the ability of invariance to CNNs(v)The proposed CNN model performance has been proved statistically significant in the performance of other ML models(vi)CNN-based diagnosis using X-ray imaging can be very effective for medical sector to handle the mass testing situations in pandemics like COVID-19.

Literature Review

Deep learning and ANNs have endorsed an exponential research focus over the last decade. The deep ANNs have outperformed other conventional models on many essential benchmarks. Thus, ANNs have generally proved to be the state-of-the-art technology across a wide range of application areas, including NLP, speech recognition, image processing, biological sciences, and other commercial as well as academic areas. The advancement of ANNs has massive potential in healthcare applications, specifically in medical data analysis, diagnosis through medical image processing, and analysis. As seen in recent times, various parts of the world face the healthcare crisis both in terms of the needed number of healthcare professionals and testing equipment. Considering the present pandemic situation, there is an appurtenant relationship between the detection of COVID-19 cases and chest X-ray image analysis and classification. In this work, an automatic diagnostic system has been developed using CNN which uses chest X-ray analysis results to diagnose whether a person is COVID-19-affected or normal. Preliminary analysis of this study has shown promising results in terms of its accuracy and other performance parameters to diagnose the disease in a cost-effective and time-efficient manner. This study used CNN with extra layers to improve the COVID-19 X-ray image classification accuracy. In neural networks, the CNN structure is specially designed to process the two-dimensional image tasks although it can also be used in one- and three-dimensional data. CNN is a type of DNN, inspired by the

visual system of the human brain, and is most commonly used in the analysis of visual imagery. To train the CNN model, first, the dataset has been obtained from GitHub . Since the dataset obtained for training the model was very small in size and imbalanced, to solve the problem of having very-limited-sized X-ray image dataset, it has been extended using data augmentation techniques to increase its size and also to make the model training feature rich. Image flipping and rotation at different angles have been used to generate more data. For dataset balancing in terms of proportion of images with different class labels, the dataset has been further extended with some more image instances of the minority class. After data augmentation and dataset balancing, the CNN model has been trained using a total of 800 images (400 COVID-19 and 400 normal) and then the model has been tested by using a test set. The CNN model performance evaluation has then been done using different performance metrics. These metrics include accuracy, precision, sensitivity, specificity, ROC AUC, and F_1 score. Later, the proposed CNN model has also been tested using an independent dataset obtained from the IEEE data port for independent validation of the proposed CNN model. Various machine learning models have also been used for the comparative performance analysis in comparison with the proposed CNN model to show its significance over these models.

THEORY

Input Image

In the figure, we have an RGB image which has been separated by its three color planes — Red, Green, and Blue. There are a number of such color spaces in which images exist — Grayscale, RGB, HSV, CMYK, etc.

You can imagine how computationally intensive things would get once the images reach dimensions, say 8K (7680×4320). The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets.

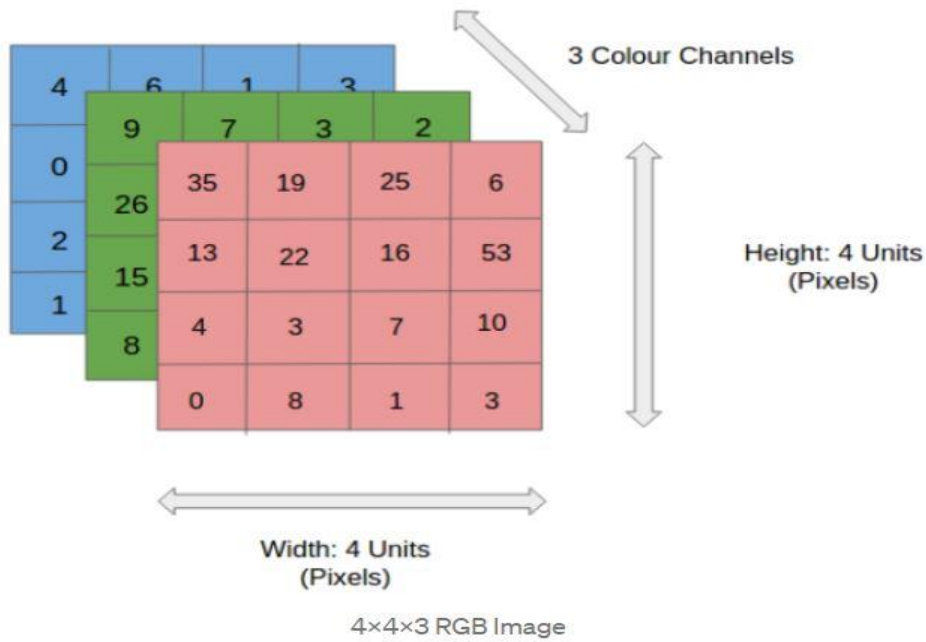
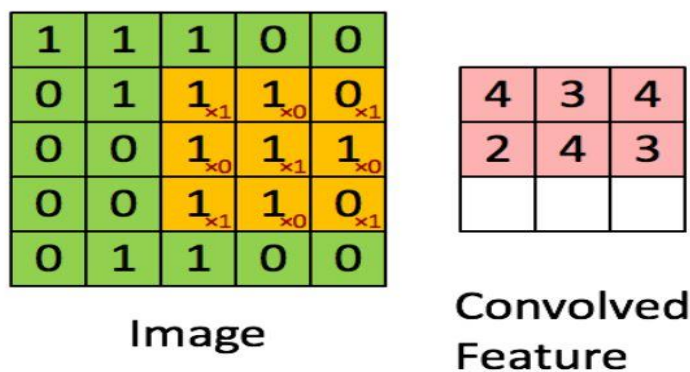


Fig 1: Input image Matrix

Convolution Layer — The Kernel

Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB)



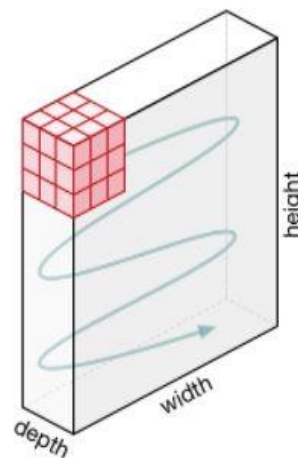
Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

Fig 2: Convolution Operation

In the above demonstration, the green section resembles our **5x5x1 input image, I**. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the **Kernel/Filter, K**, represented in the color yellow. We have selected **K as a 3x3x1 matrix**.

Kernel/Filter,	K	=	1	0	1
0			1		0
1	0	1			

The Kernel shifts 9 times because of **Stride Length = 1 (Non-Strided)**, every time performing a **matrix multiplication operation between K and the portion P of the image** over which the kernel is hovering.



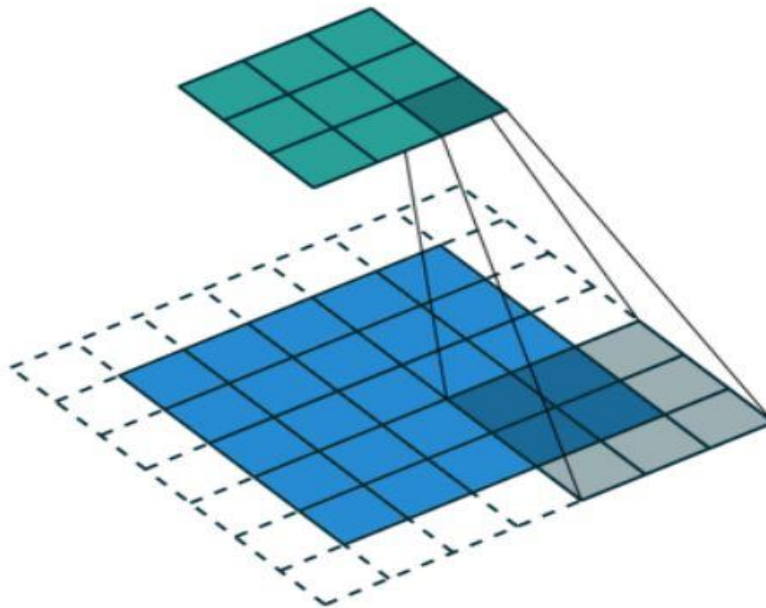
Movement of the Kernel

Fig 3: Movement Of Kernal

The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.

The objective of the Convolution Operation is to **extract the high-level features** such as edges, from the input image.

ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.



Convolution Operation with Stride Length = 2

Fig 4: Convolution Opeartion

Pooling Layer

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to **decrease the computational power required to process the data** through dimensionality reduction. Furthermore, it is useful for **extracting dominant features** which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

There are two types of Pooling: Max Pooling and Average Pooling. **Max Pooling** returns the **maximum value** from the

portion of the image covered by the Kernel. On the other hand, **Average Pooling** returns the **average of all the values** from the portion of the image covered by the Kernel.

Max Pooling also performs as a **Noise Suppressant**. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that **Max Pooling performs a lot better than Average Pooling**.

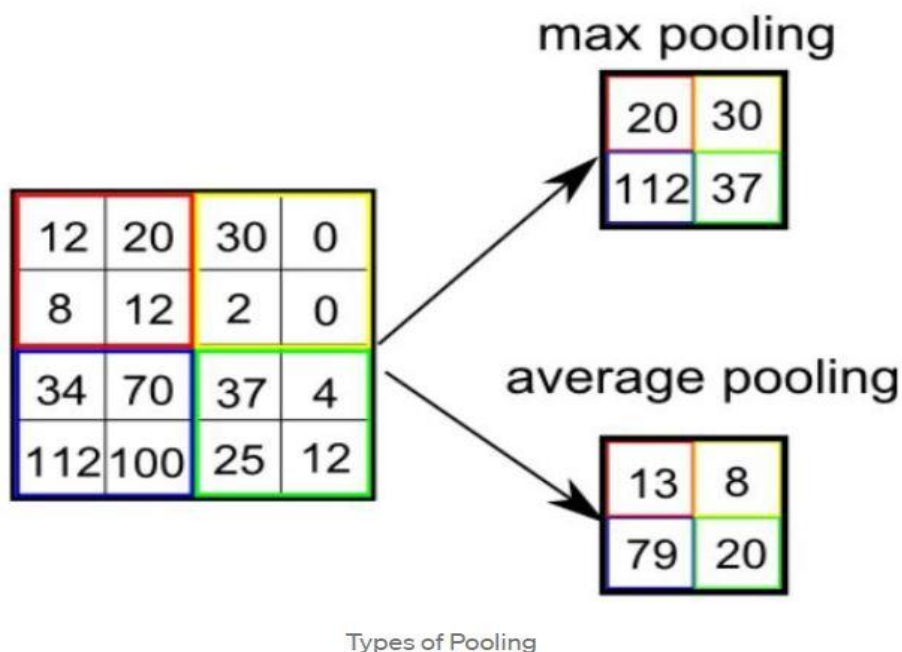


Fig 5:Maxpooling Opeartion

The Convolutional Layer and the Pooling Layer, together form the i-th layer of a Convolutional Neural Network. Depending on the complexities in the images, the number of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power.

After going through the above process, we have successfully enabled the model to understand the features. Moving on, we are going to flatten the final output and feed it to a regular Neural Network for classification purposes.

Classification — Fully Connected Layer (FC Layer):

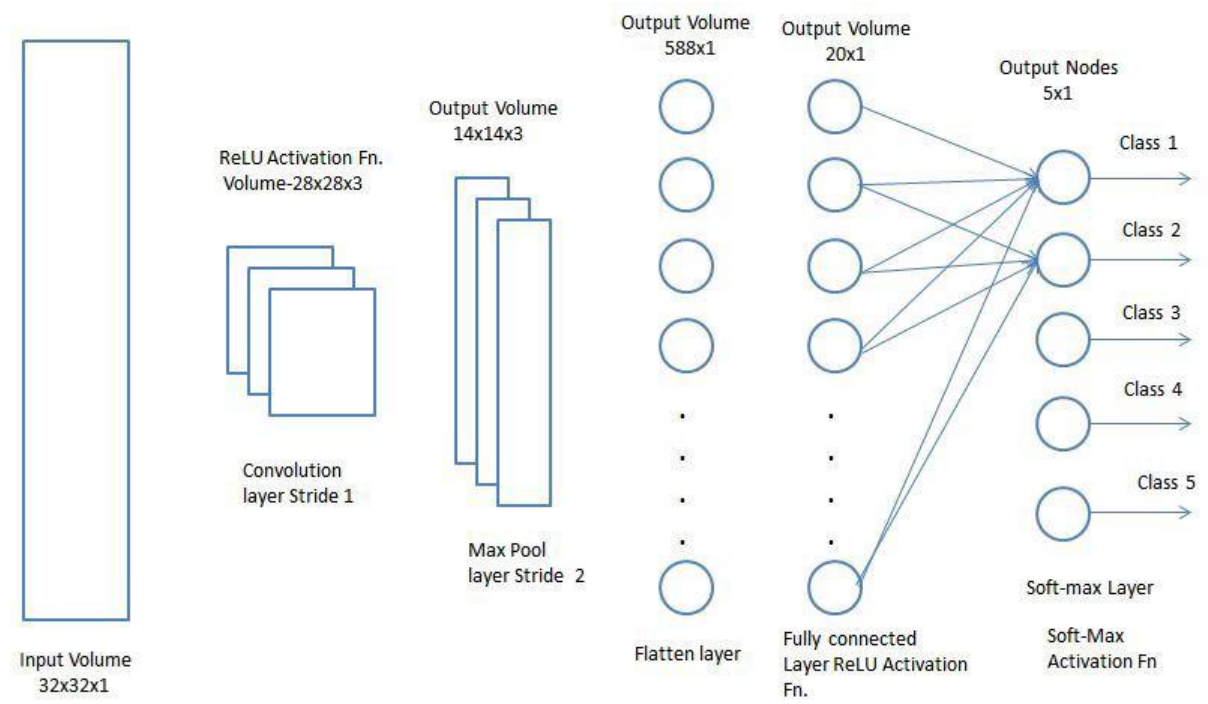


Fig 6:Fully Connected Layer

Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

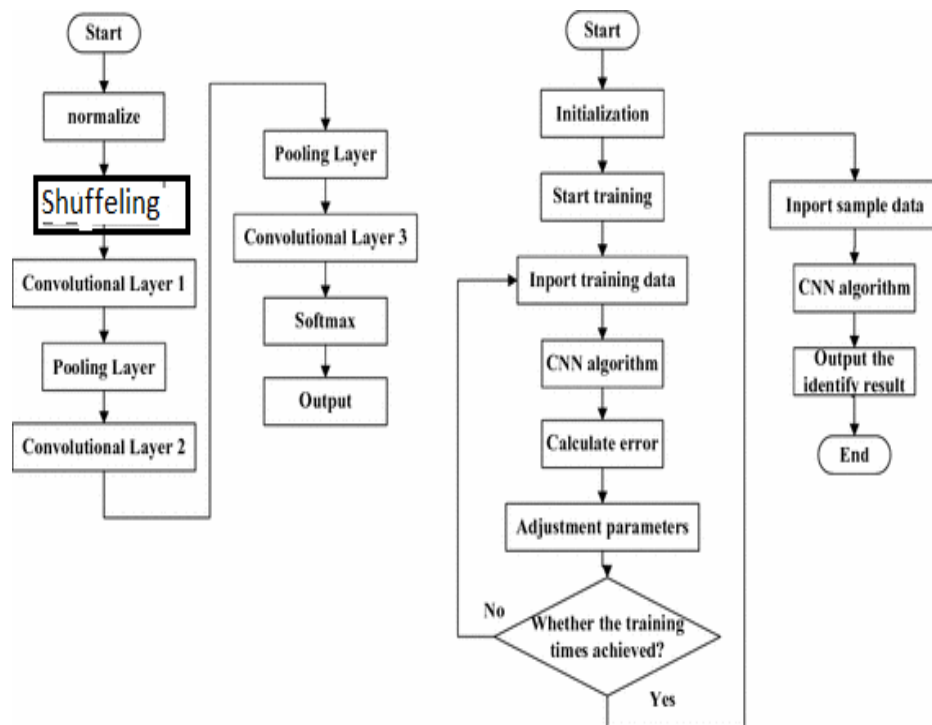
Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a

column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the **Softmax Classification** technique.

There are various architectures of CNNs available which have been key in building algorithms which power and shall power AI as a whole in the foreseeable future

IMPLEMENTATION

Flow Chart:



METHODOLOGY

BACKPROPAGATION :

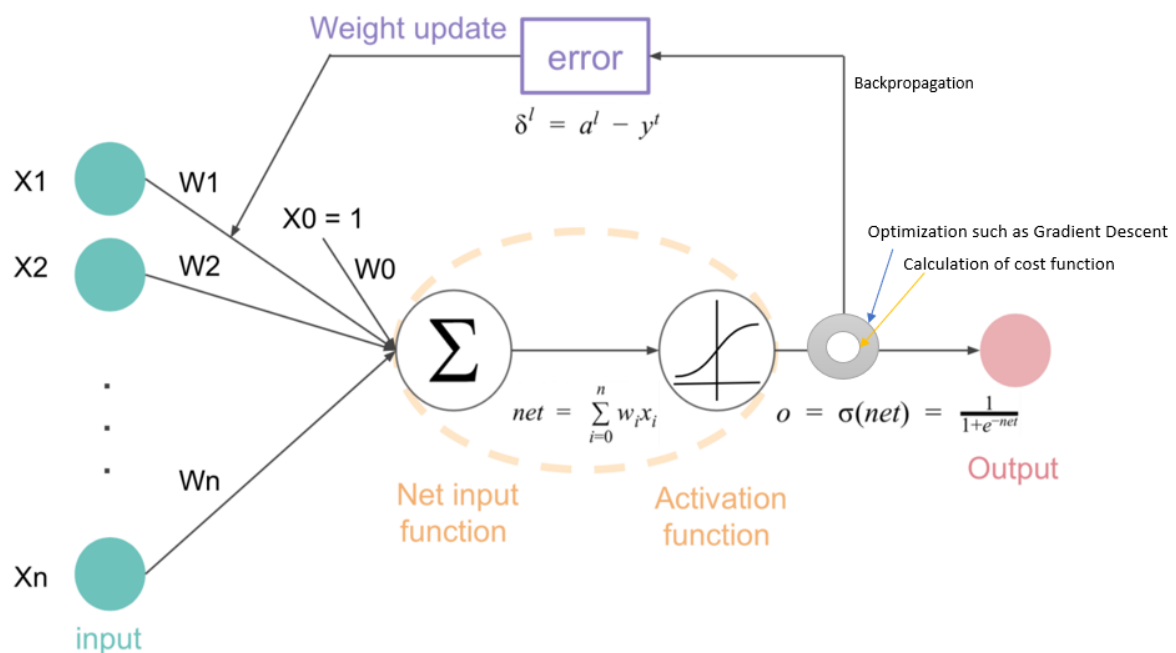


Fig 7:Backpropagation

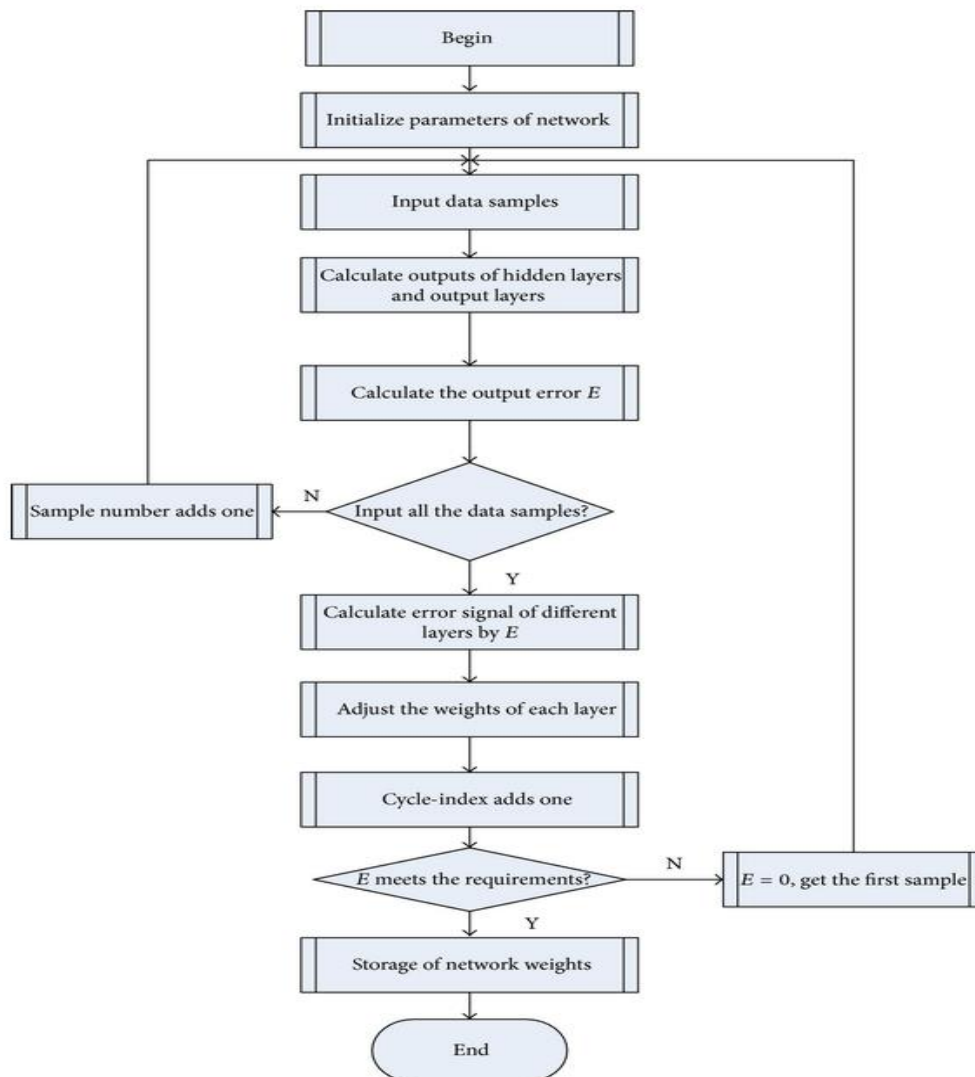
The Back propagation algorithm in neural network computes the gradient of the loss function for a single weight by the chain rule. It efficiently computes one layer at a time, unlike a native direct computation. It computes the gradient, but it does not define how the gradient is used. It generalizes the computation in the delta rule.

- Inputs X , arrive through the preconnected path
- Input is modeled using real weights W . The weights are usually randomly selected.
- Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
- Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.

- Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.

Keep repeating the process until the desired output is achieved

Backpropagation Flow Chart



Adam Optimizer:

Adaptive Moment Estimation is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problem involving a lot of data or parameters. It requires less memory and is

efficient. Intuitively, it is a combination of the 'gradient descent with momentum' algorithm and the 'RMSP' algorithm

- **Momentum:**

This algorithm is used to accelerate the gradient descent algorithm by taking into consideration the 'exponentially weighted average' of the gradients. Using averages makes the algorithm converge towards the minima in a faster pace.

where,

$$v_t = \beta v_{t-1} + (1 - \beta) * \left[\frac{\delta L}{\delta w_t} \right]^2$$

w_t = weights at time t

w_{t+1} = weights at time t+1

α_t = learning rate at time t

∂L = derivative of Loss Function

∂w_t = derivative of weights at time t

v_t = sum of square of past gradients. [i.e sum($\partial L / \partial w_{t-1}$)] (initially, $v_t = 0$)

β = Moving average parameter (const, 0.9)

ϵ = A small positive constant (10^{-8})

- **Root Mean Square Propagation (RMSP):**

Root mean square prop or RMSprop is an adaptive learning algorithm that tries to improve AdaGrad. Instead of taking the cumulative sum of squared gradients like in AdaGrad, it takes the 'exponential moving average'.

where,

$$v_t = \beta v_{t-1} + (1 - \beta) * \left[\frac{\delta L}{\delta w_t} \right]^2$$

W_t = weights at time t

W_{t+1} = weights at time t+1

α_t = learning rate at time t

∂L = derivative of Loss Function

∂W_t = derivative of weights at time t

V_t = sum of square of past gradients. [i.e sum($\partial L / \partial W_{t-1}$)] (initially, $V_t = 0$)

β = Moving average parameter (const, 0.9)

ϵ = A small positive constant (10^{-8})

Adam Optimizer inherits the strengths or the positive attributes of the above two methods and builds upon them to give a more optimized gradient descent.

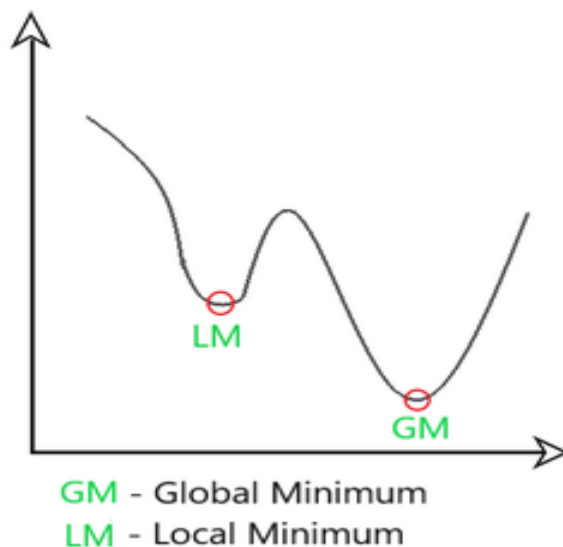


Fig 8:Error vs epochs

Here, we control the rate of gradient descent in such a way that there is minimum oscillation when it reaches the global minimum while taking big enough steps (step-size) so as to pass the local minima hurdles along the way. Hence, combining the features of the above methods to reach the global minimum efficiently.

Model Architecture

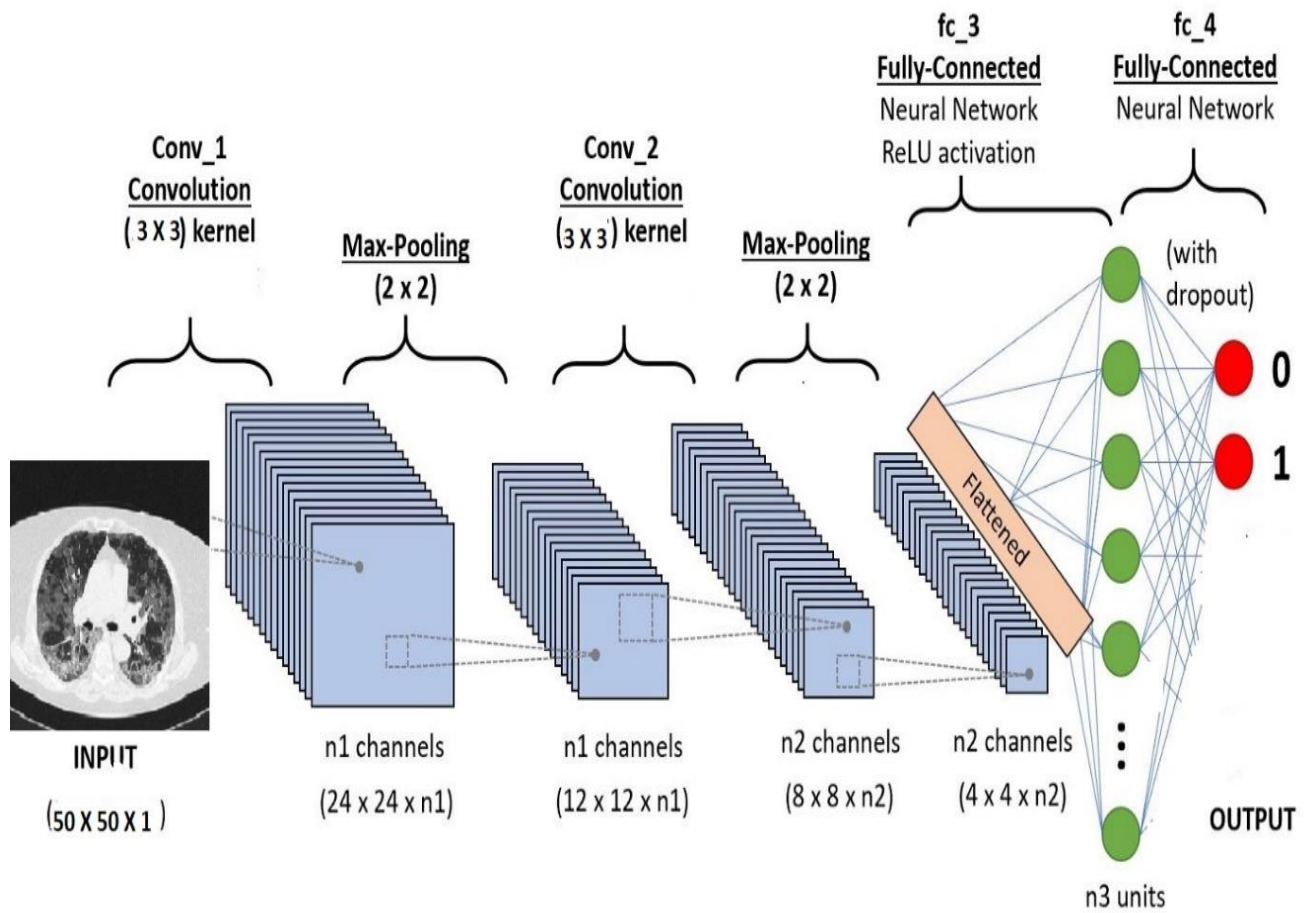


Fig 9:Model Architecture

Software Program:

```

import numpy as np
import matplotlib.pyplot as plt
import cv2
import os

size=50

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

train_data=[]
catagory=["covid","noncovid"]
for cata in catagory:
    class_name=catagory.index(cata)
    path=os.path.join("/content/gdrive/My Drive/project",cata)
    for img in os.listdir(path):

        img=cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)
        img=img/255
        new=cv2.resize(img,(size,size))
        train_data.append([new,class_name])
import random
random.shuffle(train_data)
x=[]
y=[]
for feature,label in train_data:
    x.append(feature)
    y.append(label)
x=np.array(x).reshape(-1,size,size,1)
import tensorflow as tf
from sklearn.metrics import recall_score,precision_score

from keras import backend as K
def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall
def precision_m(y_true, y_pred):

```

```

true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
precision = true_positives / (predicted_positives + K.epsilon())
return precision
def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))

from sklearn.metrics.classification import accuracy_score
from sklearn.model_selection import train_test_split

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Dense,Flatten,Dropout,Activation
X_train, X_test, Y_train, Y_test = train_test_split(x,np.array(y), test_size=0.2, random_state=0)

model=Sequential()

model.add(Conv2D(64,(3,3),input_shape=x.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['acc',f1_m,recall_m,precision_m])
history=model.fit(X_train,Y_train,validation_split=0.5,epochs=20,batch_size=100)
model.summary()
plt.plot(history.history['f1_m'])
plt.ylabel('f1_score')
plt.xlabel('epochs')
plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.show()
img=cv2.imread("/content/gdrive/My Drive/project/covid/0.jpg",cv2.IMREAD_GRAYSCALE)
new=cv2.resize(img,(50,50))
new=np.array(new).reshape(-1,size,size,1)

```

```
Y_pred=model.predict(X_test)
```

Loss vs epochs and f 1 score vs epochs plots:

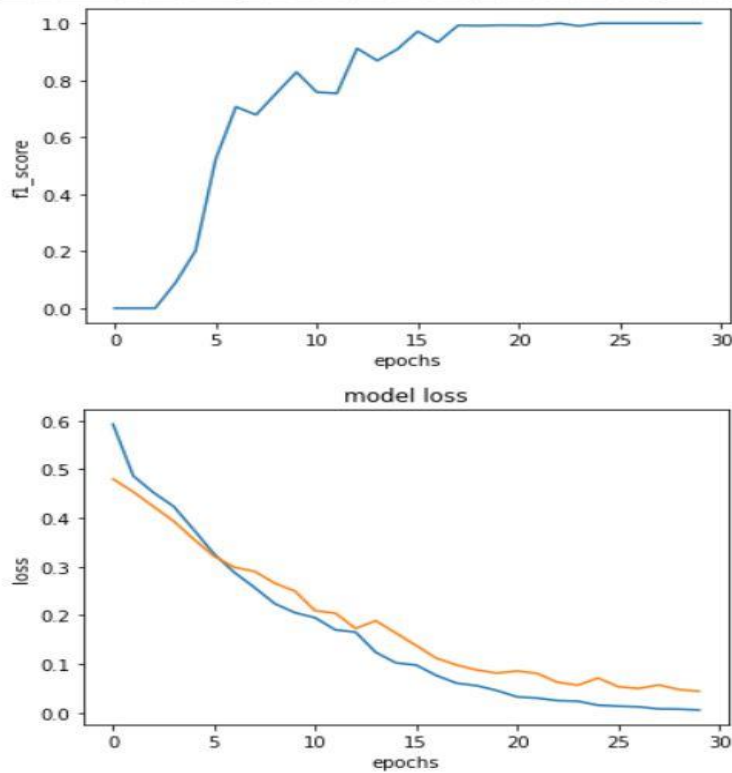


Fig 10: Loss vs Epochs and f1 score vs epochs

CNN Model training phase:

```
warnings.warn(message, FutureWarning)
Epoch 1/20
68/68 [=====] - 4s 35ms/step - loss: 0.4766 - acc: 0.8222 - f1_m: 0.8343 - val_loss: 0.3666 - val_acc: 0.8259 - val_f1_m: 0.8196
Epoch 2/20
68/68 [=====] - 2s 31ms/step - loss: 0.3142 - acc: 0.8815 - f1_m: 0.2245 - val_loss: 0.2342 - val_acc: 0.9111 - val_f1_m: 0.3059
Epoch 3/20
68/68 [=====] - 2s 31ms/step - loss: 0.1613 - acc: 0.9481 - f1_m: 0.4608 - val_loss: 0.1182 - val_acc: 0.9667 - val_f1_m: 0.4704
Epoch 4/20
68/68 [=====] - 2s 32ms/step - loss: 0.0555 - acc: 0.9852 - f1_m: 0.5951 - val_loss: 0.0586 - val_acc: 0.9667 - val_f1_m: 0.4704
Epoch 5/20
68/68 [=====] - 2s 31ms/step - loss: 0.0328 - acc: 0.9926 - f1_m: 0.5559 - val_loss: 0.0445 - val_acc: 0.9852 - val_f1_m: 0.5216
Epoch 6/20
68/68 [=====] - 2s 32ms/step - loss: 0.0051 - acc: 1.0000 - f1_m: 0.5441 - val_loss: 0.0292 - val_acc: 0.9741 - val_f1_m: 0.5172
Epoch 7/20
68/68 [=====] - 2s 32ms/step - loss: 0.0021 - acc: 1.0000 - f1_m: 0.5882 - val_loss: 0.0299 - val_acc: 0.9852 - val_f1_m: 0.5216
Epoch 8/20
68/68 [=====] - 2s 31ms/step - loss: 0.0015 - acc: 1.0000 - f1_m: 0.5588 - val_loss: 0.0305 - val_acc: 0.9704 - val_f1_m: 0.5142
Epoch 9/20
68/68 [=====] - 2s 32ms/step - loss: 0.0010 - acc: 1.0000 - f1_m: 0.5588 - val_loss: 0.0230 - val_acc: 0.9852 - val_f1_m: 0.5515
Epoch 10/20
68/68 [=====] - 2s 32ms/step - loss: 6.4702e-04 - acc: 1.0000 - f1_m: 0.5441 - val_loss: 0.0264 - val_acc: 0.9852 - val_f1_m: 0.5515
Epoch 11/20
68/68 [=====] - 2s 32ms/step - loss: 5.6014e-04 - acc: 1.0000 - f1_m: 0.5294 - val_loss: 0.0193 - val_acc: 0.9852 - val_f1_m: 0.5216
Epoch 12/20
68/68 [=====] - 2s 32ms/step - loss: 5.0003e-04 - acc: 1.0000 - f1_m: 0.5735 - val_loss: 0.0188 - val_acc: 0.9852 - val_f1_m: 0.5216
Epoch 13/20
68/68 [=====] - 2s 31ms/step - loss: 5.2515e-04 - acc: 1.0000 - f1_m: 0.5294 - val_loss: 0.0159 - val_acc: 1.0000 - val_f1_m: 0.5588
Epoch 14/20
68/68 [=====] - 2s 32ms/step - loss: 3.0459e-04 - acc: 1.0000 - f1_m: 0.5588 - val_loss: 0.0165 - val_acc: 1.0000 - val_f1_m: 0.5588
Epoch 15/20
68/68 [=====] - 2s 32ms/step - loss: 2.5465e-04 - acc: 1.0000 - f1_m: 0.6029 - val_loss: 0.0167 - val_acc: 1.0000 - val_f1_m: 0.5588
Epoch 16/20
68/68 [=====] - 2s 32ms/step - loss: 2.3266e-04 - acc: 1.0000 - f1_m: 0.5441 - val_loss: 0.0168 - val_acc: 0.9852 - val_f1_m: 0.5515
Epoch 17/20
68/68 [=====] - 2s 32ms/step - loss: 1.8922e-04 - acc: 1.0000 - f1_m: 0.6029 - val_loss: 0.0154 - val_acc: 1.0000 - val_f1_m: 0.5588
Epoch 18/20
68/68 [=====] - 2s 32ms/step - loss: 1.8013e-04 - acc: 1.0000 - f1_m: 0.5882 - val_loss: 0.0135 - val_acc: 1.0000 - val_f1_m: 0.5588
Epoch 19/20
68/68 [=====] - 2s 31ms/step - loss: 1.4897e-04 - acc: 1.0000 - f1_m: 0.5588 - val_loss: 0.0137 - val_acc: 1.0000 - val_f1_m: 0.5588
```

Fig 11: CNN Model Training

Performance Analysis

Model Prediction

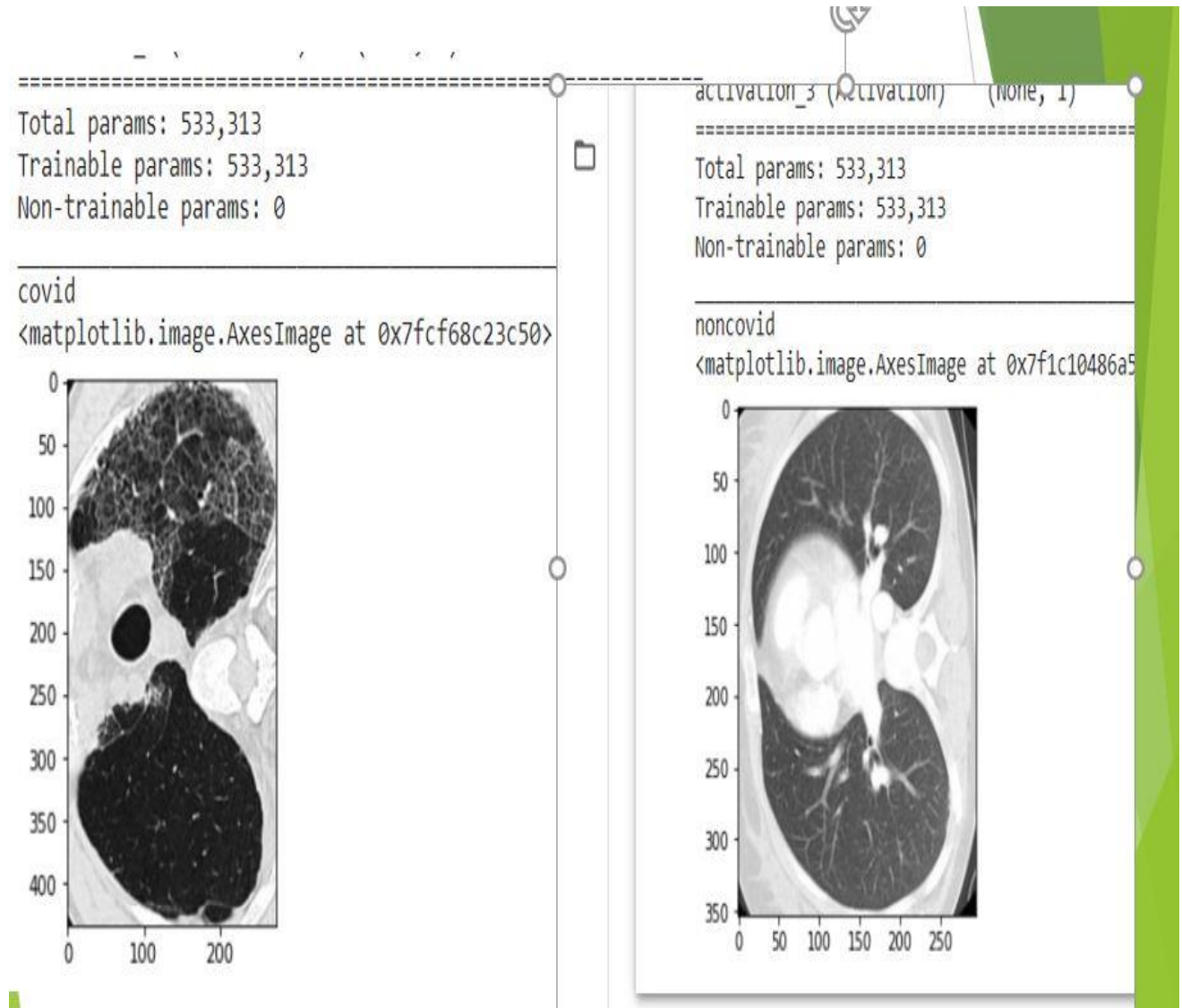


Fig 12:Model Prediction of CNN Model

DEEP BELIEF NETWORK MODEL

ACCURACY:

Deep belief networks are pretrained by using algorithm called Greedy algorithm. This algorithm uses layer-by-layer approach for

learning all the top-down approach and most important generative weights. These associated weights determine how all variables in one layer depend on the other variables in the above layer [14]. In DBN, we execute several steps of Gibbs sampling on the top two hidden layers. This stage is basically drawing a sample from the RBM by the two hidden layers at top.

Single pass of ancestral sampling is used through the rest of the model to draw a sample from the entire visible units. Learning the values of the all latent variables in each layer can be implicit by a single, bottom-up pass. Greedy pretraining begins with an observed data vector only in the bottom layer. It then affords the generative weights in the reverse direction using fine-tuning.

DEEP BELIEF NETWORK CODE:

```

from dbn import SupervisedDBNClassification
import numpy as np

np.random.seed(1337) # for reproducibility

from sklearn.model_selection import train_test_split
from sklearn.metrics import recall_score, precision_score
from sklearn.metrics.classification import accuracy_score
import matplotlib.pyplot as plt
from keras import backend as K

import cv2
import os
size=10
def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))

return precision

```

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

train_data=[]
catagory=["covid","noncovid"]
for cata in catagory:
    class_name=catagory.index(cata)
    path=os.path.join("/content/gdrive/My Drive/project",cata)
    for img in os.listdir(path):

        img=cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)
        img=img/255
        new=cv2.resize(img,(size,size))
        train_data.append([new,class_name])
import random
random.shuffle(train_data)
x=[]
y=[]
for feature,lable in train_data:
    x.append(feature.flatten())
    y.append(lable)
x=np.array(x)

X_train, X_test, Y_train, Y_test = train_test_split(x,y, test_size=0.2,
    random_state=0)

classifier = SupervisedDBNClassification(hidden_layers_structure=[100,2
50,250,2],

learning_rate_rbm=0.05,
learning_rate=0.1,
n_epochs_rbm=10,
n_iter_backprop=20,
batch_size=70,
activation_function='sigmoid',
dropout_p=0.2)

history=classifier.fit(X_train, Y_train)

classifier.save('model.pkl')

classifier = SupervisedDBNClassification.load('model.pkl')

Y_pred = classifier.predict(X_test)
print('Done.\nAccuracy: %f' % accuracy_m(Y_test, Y_pred))
```


DBN ACCURACY:

```
[END] Pre-training step
[START] Fine tuning step:
>> Epoch 1 finished      ANN training loss 2.565210
>> Epoch 2 finished      ANN training loss 1.468213
>> Epoch 3 finished      ANN training loss 1.364668
>> Epoch 4 finished      ANN training loss 1.241093
>> Epoch 5 finished      ANN training loss 1.208504
>> Epoch 6 finished      ANN training loss 1.164811
>> Epoch 7 finished      ANN training loss 1.137092
>> Epoch 8 finished      ANN training loss 1.168842
>> Epoch 9 finished      ANN training loss 1.083264
>> Epoch 10 finished     ANN training loss 1.077895
>> Epoch 11 finished     ANN training loss 1.095841
>> Epoch 12 finished     ANN training loss 1.042639
>> Epoch 13 finished     ANN training loss 1.019890
>> Epoch 14 finished     ANN training loss 1.012947
>> Epoch 15 finished     ANN training loss 1.013486
>> Epoch 16 finished     ANN training loss 1.004234
>> Epoch 17 finished     ANN training loss 1.000446
>> Epoch 18 finished     ANN training loss 1.005691
>> Epoch 19 finished     ANN training loss 1.003164
>> Epoch 20 finished     ANN training loss 1.020688
[END] Fine tuning step
Done.
Accuracy: 0.823529
```

DBN VS CNN MODEL COMPARATIVE ANALYSIS

Metric	CNN MODEL	DBN MODEL
Train Loss	0.0693	0.180891
Test Loss	0.0923	0.9934
Overall Accuracy	0.9926	0.757
Precision	0.9815	0.681
Recall	0.9117	0.553
F1_Score	0.9526	0.610

DISCUSSION

Artificial intelligence (AI) techniques in general and convolutional neural networks (CNNs) in particular have attained successful results in medical image analysis and classification. A deep CNN architecture has been proposed in this paper for the diagnosis of COVID-19 based on the chest X-ray image classification. Due to the nonavailability of sufficient-size and good-quality chest CT scan image dataset, an effective and accurate CNN classification was a challenge. To deal with these complexities such as the availability of a very-small-sized and imbalanced dataset with image-quality issues, the dataset has been pre-processed in different phases using different techniques to achieve an effective training dataset for the proposed CNN model to attain its best performance. The pre-processing stages of the datasets performed in this study include dataset balancing, medical experts' image analysis, and data augmentation. The experimental results have shown the overall accuracy as high as 99.5% which demonstrates the good capability of the proposed CNN model in the current application domain. The CNN model has been tested in two scenarios. In the first scenario, the model has been tested using the 100 CT scan images of the original processed dataset which achieved an accuracy of 100%. In the second scenario, the model has been tested using an independent dataset of COVID-19 CT Scan images. The performance in this test scenario was as high as 99.5%. To further prove that the proposed model outperforms other models, a comparative analysis has been done with some of the machine learning algorithms. The proposed model has outperformed all the models generally and specifically when the model testing was done using an independent testing set.

Future Scope/Extension of the work:

- Build an app-based user interface which allows doctors to easily determine the impact of tumor and suggest treatment accordingly
- A much higher accuracy can be achieved by gaining a better dataset with high-resolution images taken directly from the CT scanner.
- Improve testing accuracy by using classifier boosting techniques like sing
- more number images, fine-tuning hyper parameters, using U-Net for complex dataset, noise removal etc.

REFERENCES

- D. Cucinotta and M. Vanelli, "WHO declares COVID-19 a pandemic," *Acta Biomedica: Atenei Parmensis*, vol. 91, pp. 157–160, 2020.
- F. Rustam, A. A. Reshi, A. Mehmood et al., "COVID-19 future forecasting using supervised machine learning models," *IEEE Access*, 2020.
- D. J. Cennimo, "Coronavirus disease 2019 (COVID-19) clinical presentation," vol. 8, pp. 101489–101499, 2020, <https://emedicine.medscape.com/article/2500114-clinical#b2>, 2020. Online.
- J. P. Cohen, "Github Covid19 X-ray dataset," 2020, <https://github.com/ieee8023/covid-chestxray-dataset>, 2020. Online.
- Z. H. Chen, "Mask-RCNN detection of COVID-19 pneumonia symptoms by employing stacked autoencoders in deep unsupervised learning on low-dose high resolution CT," *IEEE Dataport*, 2020.
- A. S. Lundervold and A. Lundervold, "An overview of deep learning in medical imaging focusing on MRI," *Zeitschrift für Medizinische Physik*, vol. 29, no. 2, pp. 102–127, 2019.
- M. Ahmad, "Ground truth labeling and samples selection for hyperspectral image classification," *Optik*, vol. 230, Article ID 166267, 2021.
- M. Umer, S. Sadiq, M. Ahmad, S. Ullah, G. S. Choi, and A. Mehmood, "A novel stacked CNN for malarial parasite detection in Thin blood smear images," *IEEE Access*, vol. 8, pp. 93782–93792, 2020.