**DETECTION AND DIAGNOSIS OF INFECTED TISSUE BASED ON LUNG CT IMAGE IN COVID-19 PATIENTS USING CNN**

Report submitted to the SASTRA Deemed to be University

as the requirement for the course

BCSCCS708: **MINI-PROJECT**

Submitted by

**PRAVEEN KUMAR REDDY L**

(Reg.No.:-121003207,CSE)

**KURIVELLA JAYA VEERA SURENDRA GUPTA**

(Reg.No.:121003153,CSE)

**HIMAPRIYA MADITHATI**

(Reg.No.: 121003113,CSE)

**December 2020**

****

****

**Bonafide Certificate**

This is to certify that the report titled “**Detection and Diagnosis of Infected tissue based on lung CT image in COVID-19 patients using CNN**” submitted as a requirement for the course, **BCSCCS708: MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Miss.** **Himapriya Madithati (121003113,CSE), Mr. Kurivella Jaya Veera Surendra Gupta (121003153,CSE), Mr. Praveen Kumar Reddy L (121003207,CSE)** during the academic year 2020-21, in the School of Computing, under my supervision

**Signature of the Project Supervisor :**

**Name with Affiliation :**

**Date :**

Mini Project Viva voce held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Examiner 1 Examiner 2**

**ACKNOWLEDGEMENT**

I express my sincere thanks to **Dr.Vaidhyasubramaniam.S**, Vice Chancellor of SASTRA Deemed to be University for providing the opportunity to work on this project.

My sincere thanks to **Dr. Umamakeswari.A**, Dean, School of Computing, SASTRA Deemed University, for her moral support and facilities.

I wish to extend my sincere and grateful thanks to my guide **Dr. Sekar.KR**, School of Computing, SASTRA Deemed University, for his advice, continuous support, encouragement which are significant to my project.

I thank all our **teaching, non-teaching and technical staff** members, who helped me during our project work.

I take immense pride in thanking our **Parents** who were a constant source of encouragement and inspiration and provided their cooperation to enable us to complete this project with utmost satisfaction.

Finally, I thank one and all for their valuable assistance throughout the project.

**ABSTRACT**

The COVID-19 had abruptly changed the way we live and it has been showing a severe impact on the economy of several countries. The discovery of COVID-19 is presently a basic errand for the clinical professionals. In this, it is important to recognize the contaminated individuals with the goal that anticipation of spread can be taken. Currently, we are using Reverse Transcription-Polymerase Chain Reaction(RT-PCR) for the diagnosis of infected patients and it is expensive and time-consuming. Therefore, developing a feasible method is necessary.

With this work, we propose two methods for the detection and one method for diagnosis of COVID-19 patients with the help of X-Ray CT images of lung. We present two methods for the diagnosis of the disease namely, deep neural network (DNN) based on the fractal features of images and convolution neural network (CNN) in which images are fed directly as input. Results show that CNN outperforms DNN with better accuracy and sensitivity. For detection of infected tissue, we propose CNN based Image Segmentation method in which the infected regions are detected with high accuracy.

We present the whole work as a GUI in which user can upload folder or image as input the respective algorithm and can visualize the results in the GUI.

**KEYWORDS:**

Reverse Transcription -Polymerase Chain Reaction(RT-PCR),

Convolution neural network(CNN), Deep Neural Network(DNN),

Image Segmentation, Accuracy, Sensitivity, GUI

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **SlNo.** | **Name of the Figure** | **Page No** |
| 1.1 | The Proposed Three Methods | 02 |
| 1.2 | DNN architecture used on features extracted | 03 |
| 1.3 | CNN Architecture for Diagnosis | 04 |
| 1.4 | CNN architecture for segmentation of COVID-19images | 05 |
| 2.1 | Sample CT images from Dataset-1 | 06 |
| 2.2 | Sample Images from SemiSeg Dataset | 06 |
| 3.1.1 | MSE vs Epoch for DNN with fractal features | 31 |
| 3.1.2 | Confusion Matrix for DNN with fractal features | 31 |
| 3.2.1 | Accuracy and loss plots for CNN | 32 |
| 3.2.2 | Confusion Matrix for CNN | 32 |
| 3.3 | Comparision of Performance for DNN and CNN Models | 33 |
| 3.4.1 | Accuracy and loss plots for Image Segmentation | 33 |
| 3.4.2 | Detecting Infected tissue using Image Segmentation | 33 |
| 3.5.1 | GUI for uploading folder for diagnosis | 34 |
| 3.5.2 | GUI for uploading single image for diagnosis | 34 |
| 3.6.1 | GUI for uploading folder for Image Segmentation | 35 |
| 3.6.2 | GUI for uploading single image for Image Segmentation | 35 |

**List of Tables**

|  |  |  |
| --- | --- | --- |
| **SlNo.** | **Name of Table** | **Page No** |
| 01 | Formulae for Performance Evaluation | 07 |

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **SlNo** | **Title** | **Page No** |
| 01. | Bonafide Certificate | ii |
| 02. | Acknowledgements | iii |
| 03. | Abstract | iv |
| 04. | List of Figures | v |
| 05. | List of Tables | v |
| 06. | Summary of the base paper | 01 |
| 07. | Merits and Demerits of the base paper | 08 |
| 08. | Source Code | 10 |
| 09. | Snapshots | 31 |
| 10. | Conclusion | 35 |
| 11. | References | 35 |

**SUMMARY OF THE BASE PAPER**

**Title of Base Paper:** Diagnosis and detection of infected tissue of COVID-19 patients based on lung x-ray image using convolutional neural network approaches

**Journal Name:** Chaos, Solitons and Fractals

**Authors:** Shayan Hassantabar, Mohsen Ahmadi, Abbas Sharifi

**Publisher:**  PERGAMON-ELSEVIER SCIENCE LTD, THE BOULEVARD, LANGFORD LANE, KIDLINGTON,OXFORD,ENGLAND,OX5 1GB

**Year of Publication:** 2020

**Indexed in:** SCIE

**Introduction:**

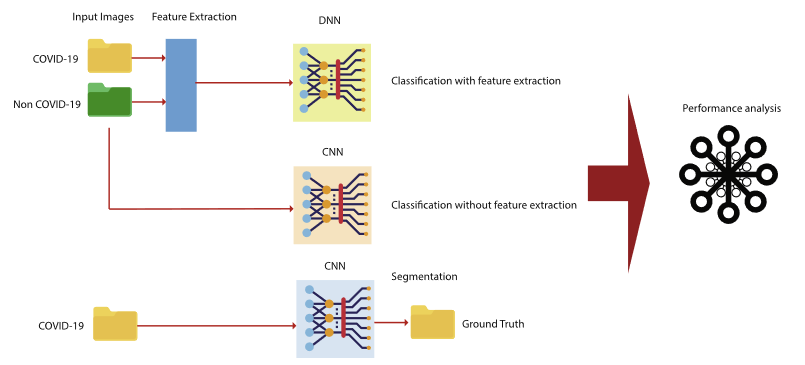
These days, "Coronavirus" is prefixed with "novel," as it is another strain of the virus. As shown by the WHO, Coronaviruses have a spot with a wide family run from the standard cold to risky diseases (World Health Organization, 2020). Novel coronavirus disease (known as COVID-19) emerged in Wuhan, China in December 2019 and is significantly impacting the world. So far, millions of individuals are influenced and it brought about the passing of thousands. Hence early and exact conclusion of COVID-19 is essential for controlling the spread of the illness and to diminish the death rate.

Currently, we are using Reverse Transcription-Polymerase Chain Reaction (RT-PCR) method which involves detection of viral nucleic acid from sputum. This method is not feasible as the test materials are not available everywhere and it is expensive and time-consuming. In addition, this method has a relatively low sensitivity (true positive rate). Hence, the trial of COVID-19 is at present a troublesome assignment due to the inaccessibility of the finding framework all over, which is causing alarm. By virtue of the limited openness of COVID19 testing units, we need to rely upon various discovering measures.

Deep Neural Networks is gradually changing the way we look at technologies and it has shown an impact in the fields of Image Recognition, Self driving cars, automatic machine translation, computer vision. In addition to this, DNN architecture is computationally effective and provides accurate predictions. Hence researchers opted DNN as AI detection algorithms for COVID-19. Since radiographic patterns on CT images assures accurate results like high sensitivity and specificity compared to RT-PCR, CT features are used for immediate diagnosis of COVID-19 patients and RT-PCR as a conformation tool. Because of the effective results several researchers have worked on collecting CT image datasets and made them publicly available.

In this work, we presented Deep Learning methods for diagnosis and detection of COVID-19 in patients. For diagnosis and classification we have used two AI based methods using MRI images. The first technique is used for classification between COVID and NON-COVID based on fractal methods and the second technique involves CNN (Convolutional Neural Networks).For the detection we have implemented CNN based segmentation methods that identifies the infected regions in MRI images.

**Proposed methodology:**



**(Fig 1.1 The Proposed Three Methods)**

1. ***Feature Extraction using the fractal method(DNN):***

Feature Extraction involves extracting features from the input data which are intended to be informative and non-redundant. Dealing with a large number of resources is expensive and requires a huge amount of memory and computational power. Fractal method is used to extract features and the fractal algorithm is applied with the aid of Covariance analysis to generate eigenvalues from the image and reduce the dimension. For this the input images should be of same size and they should be gray images with specific resolution. Image is converted into a column vector through reshaping and is loaded through a matrix of size M\*N where M is the number of images and N is the number of pixels in each image. Initially there are M images each image is of N\*N dimension.

(i)

(ii)

*2* (iii)

Average of each image is calculated to find its Standard Deviation from the original image. Then the Covariance is calculated.

(iv)

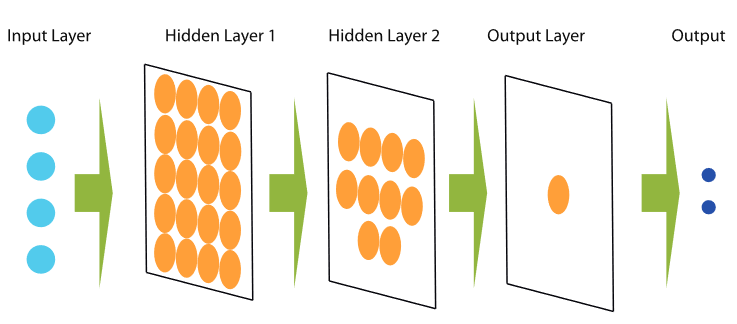
(v)

(vi)

Then the eigenvalues and eigenvectors are obtained using

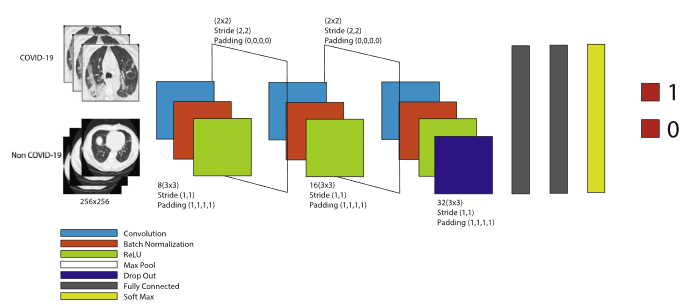
(vii)

The extracted features are given as input the following DNN architecture **(Fig 1.2 DNN architecture used on features extracted)**



1. ***Convolutional Neural Network(CNN):***

CNN has several layers that are taught effectively. In general CNN has three layers: The convolution layer, The pooling layer and The fully connected layer. It also has some other functional layers as follows.



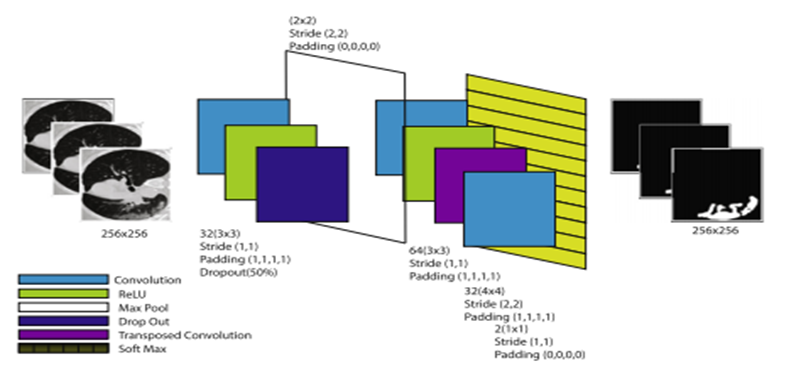
**(Fig 1.3 CNN Architecture for Diagnosis)**

Here the network has 16 layers with 3 convolutional layers and the images are labelled as 1 for COVID-19 and 0 for NON-COVID-19 patients. Padding is made in all Convolution Layers and Zero Padding is made in Pooling Layer. For Pooling, Max Pooling is used. In between layers have RELU as Activation function and Softmax is used in the last output layer.

Grayscale CT-images of lung are giving as input to the model with 256x256 dimension.

1. ***CNN based Image Segmentation****:*

Here the architecture has 11 layers with 3 convolutional layers and the input images are 256\*256 grayscale CT scan images and the output is the detected infected region of damaged tissues of input images.



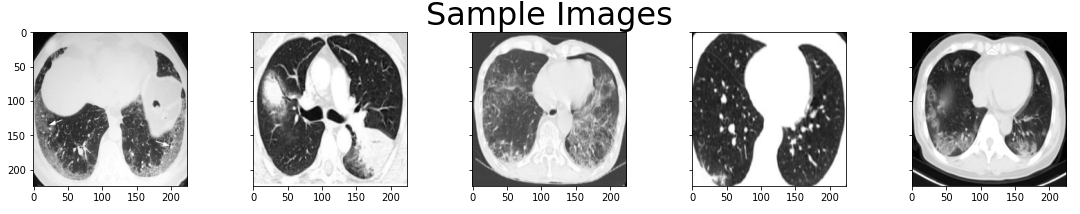
**(Fig 1.4 CNN architecture for segmentation of COVID-19images)**

Softmax function is used as the activation function in the final layer and the output image is of dimension 256x256x2

The ground truth image consisting of the infected regions is the benchmark image which should be similar to the image obtained through the output layer.

**Datasets:**

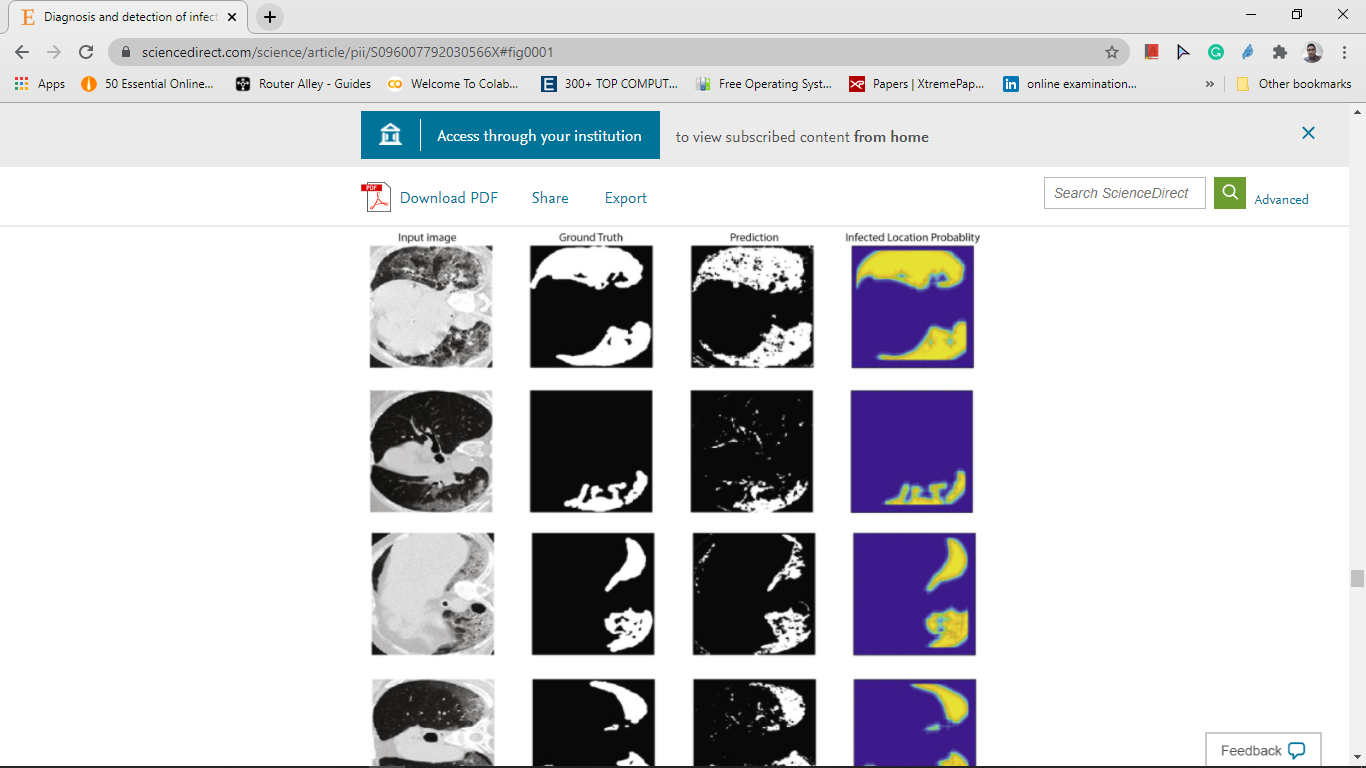
For the diagnosis of the disease, we have utilized a database of computed tomography images (CT images). A computerized tomography (CT) scan consolidates a series of X-ray pictures taken from various points around the body and uses Computer processing to make cross-sectional images (cuts) of the bones, veins and delicate tissues inside our body. CT scan images give more-nitty gritty information than plain X-rays do. The images were classified into two COVID-19 and non-COVID-19 images of patients and typical individuals. This dataset has 682 images and is used for the classification and diagnosis of COVID-19.



**(Fig 2.1 Sample CT images from Dataset-1)**

For the detection of the infected tissue in lung X-ray images we used COVID-Semiseg dataset. This dataset incorporates two sections consisting of input and ground truth images. The

ground truth images show the infected regions in the X-ray images of lungs. This dataset has 100 input images and 100 ground truth images and for the training of the model, we utilised 80% of the images and for testing we used 20% .



**(Fig 2.2 Sample Images from SemiSeg Dataset)**

**Performance Metrics:**

The following parameters are used to evaluate the efficiency of classification:

* True Positive (TP): The healthy image is correctly detected.
* False Positive (FP): The healthy image is misdiagnosed as Covid affected.
* True Negative (TN): The Covid affected lung image is correctly detected
* False Negative (FN): The Covid affected lung image is misdiagnosed as Normal image.

|  |  |  |
| --- | --- | --- |
| **SlNo** | **Metric Name** | **Formula** |
| 01 | Accuracy(Acc) |  |
| 02 | Specificity(S) |  |
| 03 | Sensitivity or Recall (R) |  |
| 04 | Fall-out |  |
| 05 | Miss rate |  |
| 06 | S-dice |  |
| 07 | Similarity of Jaccard |  |

**(Table1: Formulae for Performance Evaluation)**

**Merits and Demerits:**

In the beginning stages of coronavirus, the only method used for diagnosis of the disease is RT-PCR, at that time Korean J Radiol came up with False-Negative Results of RT-PCR for Severe Acute Respiratory Syndrome Coronavirus which created significant impact on the research and it created the importance to develop novel methods for diagnosis of the disease. In a paper named ‘Investigation of effective climatology parameters on COVID-19 outbreak in Iran’, the author concluded that humidity has reverse relationship with the virus outbreak speed, Considering the results of the existing research , Santi kumari behera developed the classification model, i.e. ResNet50 plus SVM, which performed better compared to other 12 classification models and in a paper named Covid-19: automatic detection from X-ray images utilizing transfer learning with CNN it suggested that Deep Learning with X-ray imaging may extract significant biomarkers related to Covid-19 disease, with the best accuracy, sensitivity, and specificity.

In addition to this, in a paper named AI4 VIVID-19: AI enabled preliminary diagnosis for COVID-19 from cough samples via an app, the author proved that AI for COVID-19 can distinguish among COVID-19 coughs and several types of non-COVID-19 coughs, which encouraged the researchers to develop the Model/Method which helps doctors to identify the disease in a effective manner. After these results finally an author came up with a fully automatic deep learning system for COVID-19 diagnostic and prognostic analysis, which proved that Deep learning provides a convenient tool for fast screening of COVID-19 and identifying potential high-risk patients, which may be helpful for medical resource optimisation and early prevention before patients show severe symptoms.

Currently, we are using Reverse Transcription Polymerase Chain Reaction(RT-PCR) method for the detection of the disease which is expensive and time-consuming and all the test materials for the process are not available everywhere and the method has a relatively low sensitivity, so we can’t assure that the results are accuarate. To address this, we presented two strategies which can distinguish the COVID-19 with high accuracy. In the first model we implemented CNN with CT-scan images as input gave us an accuracy of above 90% , sensitivity of about 90.08% and in the second model we have implemented the feature extraction techniques followed by DNN for the same CT-scan images that give an accuracy above 80% and sensitivity of around 79.08% but, with very high speed compared to the first model, because of dimensionality reduction done using fractal feature extraction. Since both the models are well-trained previously by several covid and non covid images, whenever a new CT-scan image is given as an input, the model can predict immediately which helps for early detection and prevention of the spread of the virus and also we have implemented a CNN based Image Segmentation model for detection of the infected tissue in the lung CT images which helps us to identify the infected region accurately. The methods which have been implemented assure best accuracy and they are fast in diagnosis of the disease. Hence, these methods can be effective in diagnosis and can help in reducing the spread of the disease which in turn results in reducing the death rate.

Yet, the significant worry here is the accuracy which isn't almost 100%, so there is a possibility for false positives and false negatives which can lead to chaos, so it is an open concern which needs to be improved further. In our case, false negatives are not entertained but false postives doesn’t have serious concern. There is also a chance for false negatives which makes the infected person go out freely and leads to further spread of the disease which also needs an improvement. The model requires a CT-scan image which includes the utilization of X-rays. This kind of radiation negatively affects organisms, particularly for their capacity to cause genetic mutations (more noticeable in cells that are in the period of fast increase). Despite the fact that the danger of creating anomalies is little, in humans. For instance, the utilization of CT scans isn't advisable for pregnant ladies and kids.

In veterinary use, it should be avoided during pregnancy. The radiation dose regulated in this sort of examination is viewed as high when contrasted with regular strategies, so suitable criteria should be used to request this system, remembering the cost-benefit ratio, since the danger might be satisfactory when contrasted with the quantity of data acquired and also it is time taking, so even though the model is able to predict very fast but for the input the person needs to wait to get the CT-scan image, which needs to be improved. So, developing models based on other inputs can significantly improve the time to get the result.

**Source Code:**

The Source code is implemented in Google Colab using python, as we need GPU.

### **Global Parameters**

INIT\_LR, EPOCHS = 3e-4, 500

BS, IMG\_SIZE = 16, 224

DATASET\_PATH = r'/content/DATASET'

CATEGORIES =['CT\_NonCOVID','CT\_COVID']

TEST\_SET\_SIZE = 0.2

classNames = ['Non-Covid','Covid']

lb = LabelBinarizer()

### **Function to Display Sample Image**

variable\_name,I = "",0

def displayImages(images,IMG\_SIZE):

  fig,ax=plt.subplots(1,len(images),figsize=(15,15),sharey=True)

  fig.subplots\_adjust(top=0.5)

  for img in images:

        img\_array = cv2.imread(img,cv2.IMREAD\_GRAYSCALE)

        new\_array = cv2.resize(img\_array, (IMG\_SIZE, IMG\_SIZE))

        ax[i].imshow(new\_array, cmap='gray')

  ax[2].set\_title('Sample Images',fontsize=32)

  fig.tight\_layout()

  plt.show()

### **Reading Images from the Dataset**

def readImages(DATASET\_PATH,CATEGORIES,IMG\_SIZE):

    training\_data = []

    imgpath = []

    for category in CATEGORIES:

        path = os.path.join(DATASET\_PATH,category)

        class\_label = category.split('\_')[1] # get the label

        for img in tqdm(os.listdir(path)):

            img\_ar=cv2.imread(os.path.join(path,img),cv2.IMREAD\_GRAYSCALE)

            img\_array = cv2.cvtColor(img\_ar, cv2.COLOR\_BGR2RGB)

            new\_array = cv2.resize(img\_array, (IMG\_SIZE, IMG\_SIZE))

            training\_data.append([new\_array, class\_label

              imgpath.append(os.path.join(path,img))

    displayImages(imgpath,IMG\_SIZE)

    return training\_data

### **Extracting Input from the dataset**

y\_dnn=""

X\_dnn=""

def preprocess():

  global y\_dnn

  global x\_dnn

  dataset = readImages(DATASET\_PATH,CATEGORIES,IMG\_SIZE)

  print("The size of dataset : ",len(dataset))

  random.shuffle(dataset)

  X = []

  y = []

  for features,label in dataset:

      X.append(features)

      y.append(label)

  X = np.array(X)

  x\_dnn = X.reshape(-1,IMG\_SIZE\*IMG\_SIZE\*3)

  X = X.reshape(-1, IMG\_SIZE, IMG\_SIZE, 3)

  X = X/255.0

  y = np.array(y)

  y = lb.fit\_transform(y)

  y\_dnn=y

  y = to\_categorical(y)

  return train\_test\_split(X,y,test\_size=0.2,stratify=y,random\_state=42)

(trainX, testX, trainY, testY) = preprocess()

### **Function definition for Accuracy and Loss Plots**

def plothistory(history):

    fig, (ax1, ax2) = plt.subplots(1, 2,figsize=(13,5))

    ax1.plot(history.history['accuracy'],'r-')

    ax1.plot(history.history['val\_accuracy'],'b-')

    ax1.set\_title('model accuracy')

    ax1.set\_xlabel('epoch')

    ax1.set\_ylabel('accuracy')

    ax1.legend(['Train Accuracy', 'Test Accuracy'], loc='upper left')

    ax2.plot(history.history['loss'])

    ax2.plot(history.history['val\_loss'])

    ax2.set\_title('model loss')

    ax2.set\_xlabel('epoch')

    ax2.set\_ylabel('loss')

    ax2.legend(['Train Loss', 'Test Loss'], loc='upper left')

    plt.tight\_layout()

### **Function to plot Confusion Matrix**

def plotconfusematrix(cm\_train,cm\_test,classNames):

    tick\_marks,total = np.arange(len(classNames)), np.sum(cm\_train)

    s = [['TN','FP'], ['FN', 'TP']]

    fig, ax = plt.subplots(1, 2,figsize=(15,5))

    ax[0].set\_title('Confusion Matrix - Train Data')

    ax[0].set\_xlabel('Predicted label')

    ax[0].set\_ylabel('True label')

    ax[0].set\_xticks(tick\_marks)

    ax[0].set\_yticks(tick\_marks)

    ax[0].set\_xticklabels(classNames)

    ax[0].set\_yticklabels(classNames)

    plt.setp(ax[0].get\_yticklabels(), rotation=90)

    ax[0].imshow(cm\_train, interpolation='nearest', cmap='Blues')

    for i in range(2):

        for j in range(2):

            ax[0].text(j,i, str(s[i][j])+" = "+str(cm\_train[i][j])+"\n"+"{0:.2%}".format(cm\_train[i][j]/total))

    total = np.sum(cm\_test)

    ax[1].set\_title('Confusion Matrix - Test Data')

    ax[1].set\_xlabel('Predicted label')

    ax[1].set\_ylabel('True label')

    ax[1].set\_xticks(tick\_marks)

    ax[1].set\_yticks(tick\_marks)

    ax[1].set\_xticklabels(classNames)

    ax[1].set\_yticklabels(classNames)

    plt.setp(ax[1].get\_yticklabels(), rotation=90)

    ax[1].imshow(cm\_test, interpolation='nearest', cmap='Blues')

    for i in range(2):

        for j in range(2):

            ax[1].text(j,i, str(s[i][j])+" = "+str(cm\_test[i][j])+"\n"+"{0:.2%}".format(cm\_test[i][j]/total))

    plt.show()

def confusematrix(model,trainX,trainY,testX,testY,classNames):

    yhat =np.argmax(model.predict(trainX, batch\_size=BS),axis=1)    cm\_train = confusion\_matrix(trainY.argmax(axis=1), yhat)

    pred = np.argmax(model.predict(testX, batch\_size=BS),axis=1)

    cm\_test = confusion\_matrix(testY.argmax(axis=1), pred)

    print(' \tTraining Accuracy : ',accuracy\_score(trainY.argmax(axis=1),yhat),'\t\t\tTesting Accuracy : ',accuracy\_score(testY.argmax(axis=1),pred))

    plotconfusematrix(cm\_train,cm\_test,classNames)

    return testY.argmax(axis=1),pred,cm\_test

### **Function for Performance Metrics**

from prettytable import PrettyTable

def performance\_metrics(y\_test,yhat,cm,title):

    TP = cm[1][1]

    TN = cm[0][0]

    FP = cm[0][1]

    FN = cm[1][0]

    t = PrettyTable(['Preformance\_Metrics',title])

    t.add\_row(['Precision ',precision\_score(y\_test, yhat)])

    t.add\_row(['Miss-Rate ', FN / (FN + TP)])

    t.add\_row(['Fall-out ', FP / (FN + TP)])

    t.add\_row(['Sensitivity or recall ',recall\_score(y\_test, yhat)])

    t.add\_row(['Accuracy ',accuracy\_score(y\_test,yhat)])

    t.add\_row(['Specificity ', TN/(TN + FP)])

    t.add\_row(['F1 Score ',f1\_score(y\_test, yhat)])

    t.add\_row(['Similarity of Jaccard',TP/(TP + FP + FN)])

    t.add\_row(['S-dice ',(2\*TP)/(2\*TP + FP + FN)])

    print(t)

### **Extracting Features**

from sklearn import decomposition, datasets

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

print("Initial features: "+str(x\_dnn.shape[1]))

x\_std = sc.fit\_transform(x\_dnn)

pca = decomposition.PCA(n\_components=512)

x\_new = pca.fit\_transform(x\_std)

print("Extracted features: "+str(x\_new.shape[1]))

### **Splitting Data**

variable\_name = ""

X\_train, X\_test\_dnn, y\_train, y\_test\_dnn = train\_test\_split(x\_new, y\_dnn, test\_size=0.30, random\_state=42)

input\_shape = X\_train.shape[1:]

print("samples for training: "+str(X\_train.shape[0]))

print("samples for testing: "+str(X\_test\_dnn.shape[0]))

### **Creating DNN Model**

import tensorflow

from tensorflow.keras.layers import Dropout

def create\_dnn\_model():

  dnn\_model=Sequential()

  dnn\_model.add(Dense(256, activation='relu', input\_shape=input\_shape))

  dnn\_model.add(Dense(256, activation='relu'))

  dnn\_model.add(Dense(128, activation='relu'))

  dnn\_model.add(Dense(128, activation='relu'))

  dnn\_model.add(Dense(64, activation='relu'))

  dnn\_model.add(Dense(64, activation='relu'))

  dnn\_model.add(Dense(32, activation='relu'))

  dnn\_model.add(Dropout(0.5))

  dnn\_model.add(Dense(1, activation='sigmoid'))

  dnn\_model.compile(loss='binary\_crossentropy',optimizer='adam',metrics=['accuracy','mse'])

  dnn\_model.summary()

return dnn\_model

dnn\_model=create\_dnn\_model()

### **Training the DNN Model**

class myCallback(tf.keras.callbacks.Callback):

def on\_epoch\_end(self, epoch, logs={}):

    if(logs.get('val\_accuracy')>0.85):

      print("\nReached 85% accuracy so cancelling training!")

      self.model.stop\_training = True

checkpoint = ModelCheckpoint(r'Covid\_result\_DNN.h5',monitor='val\_accuracy',mode='max',save\_best\_only=True,verbose=1)

dnn\_history = dnn\_model.fit(

x = X\_train,

y = y\_train,

batch\_size=BS,

                    epochs = EPOCHS,

callbacks = [myCallback(),checkpoint],

                    validation\_data=(X\_test\_dnn, y\_test\_dnn)

)

### **Accuracy Calculation**

dnn\_model = tf.keras.models.load\_model('/content/Covid\_result\_DNN.h5')

y\_predicted=dnn\_model.predict(X\_test\_dnn)

pred=[]

for val in y\_predicted.T[0]:

    pred.append(round(val))

score = dnn\_model.evaluate(X\_test\_dnn, y\_test\_dnn, verbose=0)

print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')

score = dnn\_model.evaluate(X\_train, y\_train, verbose=0)

print(f'Train loss: {score[0]} / Train accuracy: {score[1]}')

print('Accuracy : ',accuracy\_score(y\_test\_dnn,pred))

### **Graphs**

def plot\_his(history):

    fig, (ax1) = plt.subplots(1, 1,figsize=(13,8))

    ax1.plot(history.history['mse'],'b-')

    ax1.set\_title('MSE vs Epochs')

    ax1.set\_xlabel('epoch')

    ax1.set\_ylabel('Mean Squared Error (MSE)')

    ax1.set\_yticks([-0.6,-0.4,-0.2,0,0.05,0.1,0.15,0.2,0.4])

    ax1.legend(['Train'], loc='upper right')

plot\_his(dnn\_history)

plothistory(dnn\_history)

### **Plot Confusion Matrix**

from sklearn.metrics import confusion\_matrix

y\_predicted=dnn\_model.predict(X\_train)

pred=[]

for val in y\_predicted.T[0]:

    pred.append(round(val))

pred=np.array(pred)

cm\_train=confusion\_matrix(y\_train,pred)

y\_predicted=dnn\_model.predict(X\_test\_dnn)

pred=[]

for val in y\_predicted.T[0]:

    pred.append(round(val))

pred=np.array(pred)

cm\_dnn=confusion\_matrix(y\_test\_dnn,pred)

plotconfusematrix(cm\_train,cm\_dnn,classNames)

performance\_metrics(y\_test\_dnn,pred,cm\_dnn,'DNN')

### **Proposed CNN MODEL**

def createmodel():

  model = tf.keras.models.Sequential([

    tf.keras.layers.Conv2D(filters = 8,kernel\_size = (3,3),strides = (1,1),input\_shape = (224, 224, 3),padding = 'same'),

    tf.keras.layers.BatchNormalization(),

    tf.keras.layers.Activation('relu'),

    tf.keras.layers.MaxPool2D(pool\_size=(2, 2), strides=(2,2), padding='valid'),

    tf.keras.layers.Conv2D(filters = 16,kernel\_size = (3,3), strides = (1,1),padding = 'same'),

    tf.keras.layers.BatchNormalization(),

    tf.keras.layers.Activation('relu'),

    tf.keras.layers.MaxPool2D(pool\_size = (2, 2), strides = (2,2), padding='valid'),

    tf.keras.layers.Conv2D(filters = 3,kernel\_size = (3,3), strides = (1,1),padding = 'same'),

    tf.keras.layers.BatchNormalization(),

    tf.keras.layers.Activation('relu'),

    tf.keras.layers.Dropout(rate=0.4),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(128, activation='relu'),

    tf.keras.layers.Dense(2),

    tf.keras.layers.Activation('softmax')

  ])

  opt = Adam(lr=INIT\_LR, decay=INIT\_LR / EPOCHS)

  model.compile(loss="binary\_crossentropy", optimizer=opt,metrics=["accuracy"])

  return model

pmodel = createmodel()

pmodel.summary()

### **Training The Proposed Mode**

class myCallback(tf.keras.callbacks.Callback):

  def on\_epoch\_end(self, epoch, logs={}):

    if(logs.get('val\_accuracy')>0.90):

      print("\nReached 91% accuracy so cancelling training!")

      self.model.stop\_training = True

checkpoint = ModelCheckpoint(r'Covid\_result\_Normal.h5',monitor='val\_accuracy',mode='max',save\_best\_only=True,verbose=1)

reduce\_lr = ReduceLROnPlateau(monitor='val\_accuracy',factor=0.2,patience=5,verbose=2,min\_delta=0.0001)

history\_normal = pmodel.fit(x = trainX, y = trainY, batch\_size=BS,

epochs = 100, validation\_data=(testX, testY),

                    callbacks = [myCallback(),checkpoint,reduce\_lr],

                    steps\_per\_epoch=len(trainX) // BS,

                    validation\_steps=len(testX) // BS)

print("[INFO] saving COVID-19 detector model...")

plothistory(history\_normal)

new\_model = tf.keras.models.load\_model('/content/drive/MyDrive/MiniProject/Covid\_result\_Normal.h5')

y\_test\_cnn,yhat\_cnn,cm\_cnn = confusematrix(new\_model,trainX,trainY,testX,testY,classNames)

print(classification\_report(y\_test\_cnn,yhat\_cnn,target\_names=lb.classes\_))

performance\_metrics(y\_test\_cnn,yhat\_cnn,cm\_cnn,'Proposed CNN Model')

### **Training the Proposed-model using Image Augumentation**

### **Creating Training and Testing Directories**

COVID\_SOURCE\_DIR = '/content/DATASET/CT\_COVID'

TRAINING\_DIR = '/content/covid-vs-noncovid/training'

TESTING\_DIR =  '/content/covid-vs-noncovid/testing'

TRAINING\_COVID\_DIR = '/content/covid-vs-noncovid/training/covid'

TESTING\_COVID\_DIR = '/content/covid-vs-noncovid/testing/covid'

NONCOVID\_SOURCE\_DIR = '/content/DATASET/CT\_NonCOVID'

TRAINING\_NONCOVID\_DIR = '/content/covid-vs-noncovid/training/non-covid'

TESTING\_NONCOVID\_DIR = '/content/covid-vs-noncovid/testing/non-covid'

to\_create = ['/content/covid-vs-noncovid',TRAINING\_DIR,TESTING\_DIR,

TRAINING\_COVID\_DIR,TESTING\_COVID\_DIR,TRAINING\_NONCOVID\_DIR,TESTING\_NONCOVID\_DIR]

for directory in to\_create:

    try:

        os.mkdir(directory)

    except:

        print(directory, 'failed')

def split\_data(SOURCE, TRAINING, TESTING, SPLIT\_SIZE):

    all\_files = []

    for file\_name in os.listdir(SOURCE):

        file\_path = os.path.join(SOURCE,file\_name)

        if os.path.getsize(file\_path):

            all\_files.append(file\_name)

        else:

            print('{} is zero length, so ignoring'.format(file\_name))

    n\_files = len(all\_files)

    split\_point = int(n\_files \* SPLIT\_SIZE)

    shuffled = random.sample(all\_files, n\_files)

    train\_set = shuffled[:split\_point]

    test\_set = shuffled[split\_point:]

    for file\_name in train\_set:

        copyfile(file\_path, os.path.join(TRAINING , file\_name))

    for file\_name in test\_set:

        copyfile(file\_path, os.path.join(TESTING , file\_name))

split\_size = .9

split\_data(COVID\_SOURCE\_DIR, TRAINING\_COVID\_DIR, TESTING\_COVID\_DIR, split\_size)

split\_data(NONCOVID\_SOURCE\_DIR, TRAINING\_NONCOVID\_DIR, TESTING\_NONCOVID\_DIR, split\_size)

### **Creating Test and Train Image Data Generators**

train\_datagen = ImageDataGenerator(

                            rescale=1 / 255,

                            rotation\_range=40,

                            width\_shift\_range=.2,

                            height\_shift\_range=.2,

                            shear\_range=.2,

                            zoom\_range=.2,

                            horizontal\_flip=True,

                            fill\_mode='nearest'

                            )

train\_generator = train\_datagen.flow\_from\_directory(TRAINING\_DIR,batch\_size=16,class\_mode='binary',target\_size=(224,224))

testing\_datagen = ImageDataGenerator(

                          rescale=1 / 255,

                          )

test\_generator = testing\_datagen.flow\_from\_directory(TESTING\_DIR,batch\_size=16,class\_mode='binary',target\_size=(224,224))

### **Training the model with generators**

augmodel = createmodel()

history\_augm = augmodel.fit(train\_generator,epochs=50,verbose=1,validation\_data=test\_generator)

plothistory(history\_augm)

### **Construct Confusion Matrix and Performance Metrics**

def confusematrix\_gen(model,train\_generator,test\_generator,batch\_size,classNames):

    yhat = np.argmax(model.predict(train\_generator, batch\_size+1, verbose=1, workers=0),axis=1)

    cm\_train=confusion\_matrix(train\_generator.classes,yhat)

    pred = np.argmax(model.predict(test\_generator, batch\_size+1, verbose=1, workers=0),axis=1)

    cm\_test=confusion\_matrix(test\_generator.classes,pred)

    print('\tTraining Accuracy : ',accuracy\_score(train\_generator.classes,yhat),'\t\t\tTesting Accuracy : ',accuracy\_score(test\_generator.classes,pred))

    plotconfusematrix(cm\_train,cm\_test,classNames)

    performance\_metrics(test\_generator.classes,pred,cm\_test,'Proposed CNN with Image Augumentation')

confusematrix\_gen(augmodel,train\_generator,test\_generator,16,classNames)

### **Build Pre-trained Model**

def pretrainedmodel():

  ## build network

  baseModel = VGG16(weights="imagenet", include\_top=False,input\_tensor=Input(shape=(224, 224, 3)))

  # construct the head of the model that will be placed on top of the

  # the base model

  headModel = baseModel.output

  headModel = BatchNormalization()(headModel)

  headModel = Activation('relu')(headModel)

  headModel = Dropout(0.4)(headModel)

  headModel = Flatten(name="flatten")(headModel)

  headModel = Dense(128, activation="relu")(headModel)

  headModel = Dense(2, activation="softmax")(headModel)

  # place the head FC model on top of the base model (this will become

  # the actual model we will train)

  model = Model(inputs=baseModel.input, outputs=headModel)

  for layer in baseModel.layers:

    layer.trainable = False

  opt = Adam(lr=INIT\_LR, decay=INIT\_LR / EPOCHS)

  model.compile(loss="binary\_crossentropy", optimizer=opt,metrics=["accuracy"])

  model.summary()

  return model

model = pretrainedmodel()

### **Training The Pre-trained Model**

class myCallback(tf.keras.callbacks.Callback):

  def on\_epoch\_end(self, epoch, logs={}):

    if(logs.get('val\_accuracy')>0.93):

      print("\nReached 91% accuracy so cancelling training!")

      self.model.stop\_training = True

checkpoint = ModelCheckpoint(r'Covid\_result\_Pretrained.h5',monitor='val\_accuracy',mode='max',save\_best\_only=True,verbose=1)

earlystop = EarlyStopping(monitor='val\_accuracy',min\_delta=0,patience=5,verbose=2,restore\_best\_weights=True)

reduce\_lr = ReduceLROnPlateau(monitor='val\_accuracy',factor=0.2,patience=5,verbose=2,min\_delta=0.0001)

history = model.fit(x = trainX,y = trainY,batch\_size=BS,epochs = EPOCHS,

                    callbacks = [myCallback(),checkpoint],

                    steps\_per\_epoch=len(trainX) // BS,

                    validation\_data=(testX, testY),

                    validation\_steps=len(testX) // BS,)

plothistory(history)

new\_model = tf.keras.models.load\_model('/content/Covid\_result\_Pretrained.h5')

y\_test,yhat,cm = confusematrix(new\_model,trainX,trainY,testX,testY,classNames)

### **Comparision of Performance Metrics**

from prettytable import PrettyTable

def performance\_metrics(y\_test\_cnn,yhat\_cnn,cm\_cnn,y\_test\_dnn,yhat\_dnn,cm\_dnn):

    TP\_cnn = cm\_cnn[1][1]

    TN\_cnn = cm\_cnn[0][0]

    FP\_cnn = cm\_cnn[0][1]

    FN\_cnn = cm\_cnn[1][0]

    TP\_dnn = cm\_dnn[1][1]

    TN\_dnn = cm\_dnn[0][0]

    FP\_dnn = cm\_dnn[0][1]

    FN\_dnn = cm\_dnn[1][0]

    t = PrettyTable(['Preformance\_Metrics','DNN','CNN'])

    t.add\_row(['Precision ',precision\_score(y\_test\_dnn, yhat\_dnn, average="macro"),precision\_score(y\_test\_cnn, yhat\_cnn, average="macro")])

    t.add\_row(['Miss-Rate ', FN\_dnn / (FN\_dnn + TP\_dnn), FN\_cnn / (FN\_cnn + TP\_cnn)])

    t.add\_row(['Fall-out ', FP\_dnn / (FN\_dnn + TP\_dnn), FP\_cnn / (FN\_cnn + TP\_cnn)])

    t.add\_row(['Sensitivity or recall ',recall\_score(y\_test\_dnn, yhat\_dnn , average="macro"),recall\_score(y\_test\_cnn, yhat\_cnn , average="macro")])

    t.add\_row(['Accuracy ',accuracy\_score(y\_test\_dnn,yhat\_dnn),accuracy\_score(y\_test\_cnn,yhat\_cnn)])

    t.add\_row(['Specificity ', TN\_dnn/(TN\_dnn + FP\_dnn), TN\_cnn/(TN\_cnn + FP\_cnn)])

    t.add\_row(['F1 Score ',f1\_score(y\_test\_dnn, yhat\_dnn , average="macro"),f1\_score(y\_test\_cnn, yhat\_cnn , average="macro")])

    t.add\_row(['Similarity of Jaccard',jaccard\_similarity\_score(y\_test\_dnn, yhat\_dnn),jaccard\_similarity\_score(y\_test\_cnn, yhat\_cnn)])

    t.add\_row(['S-dice ',(2\*TP\_dnn)/(2\*TP\_dnn + FP\_dnn + FN\_dnn),(2\*TP\_cnn)/(2\*TP\_cnn + FP\_cnn + FN\_cnn)])

    print(t)

performance\_metrics(y\_test,yhat,cm,y\_test\_dnn,pred,cm\_dnn)

### **Image Segmentation**

### Data Preprocessing

def convertnii2png(inputfile,outputfile,name):

    print('Input file is ', inputfile)

    print('Output folder is ', outputfile)

    image\_array = nibabel.load(inputfile).get\_data()

if not os.path.exists(outputfile):

            os.makedirs(outputfile)

    if len(image\_array.shape) == 4:

        nx, ny, nz, nw = image\_array.shape

        total\_volumes = image\_array.shape[3]

        total\_slices = image\_array.shape[2]

        for current\_volume in range(0, total\_volumes):

            slice\_counter = 0

            for current\_slice in range(0, total\_slices):

                if (slice\_counter % 1) == 0:

                    data = image\_array[:,:, current\_slice, current\_volume]

                    print('Saving image...')

                    image\_name = inputfile[:-4] + "\_t" + "{:0>3}".format(str(current\_volume+1)) + "\_z" + "{:0>3}".format(str(current\_slice+1))+ ".png"

                    imageio.imwrite(image\_name, data)

                    print('Saved.\n','Moving files...')

                    src = image\_name

                    shutil.move(src, outputfile)

                    slice\_counter += 1

                    print('Moved.')

    elif len(image\_array.shape) == 3:

        nx, ny, nz = image\_array.shape

        total\_slices = image\_array.shape[2]

       slice\_counter = 0

           for current\_slice in range(0, total\_slices):

            if (slice\_counter % 1) == 0:

                    data = image\_array[:, :, current\_slice]

                if (slice\_counter % 1) == 0:

                    print('Saving image...')

                    image\_name = name + "{:0>3}".format(str(current\_slice+1))+ ".png"

                    imageio.imwrite(image\_name, data)

                    print('Moving image...')

                    src = image\_name

                    shutil.move(src, outputfile)

                    slice\_counter += 1

### **Converting images from .nii format to .png**

inputfile = r'C:\Users\SurendraGupta\Downloads\tr\_im.nii\tr\_im.nii'

outputfile = r'E:\sem7\SemiSegData\images'

convertnii2png(inputfile,outputfile,'img')

DATASET\_PATH = r'/content/SemiSegData'

CATEGORIES =['images','mask']

Mas,imge = [],[]

for filename in os.listdir(os.path.join(DATASET\_PATH,CATEGORIES[0]) ):

    if filename.endswith('.png'):

        imge.append(filename)

for filename in os.listdir(os.path.join(DATASET\_PATH,CATEGORIES[1]) ):

    if filename.endswith('.png'):

        mas.append(filename)

mas.sort()

imge.sort()

X=[]

y=[]

for filename in imge:

    img=cv2.imread(os.path.join(os.path.join(DATASET\_PATH,CATEGORIES[0]), filename),cv2.IMREAD\_REDUCED\_COLOR\_2)

    X.append(img)

for filename in mas:

    img=cv2.imread(os.path.join(os.path.join(DATASET\_PATH,CATEGORIES[1]), filename),cv2.IMREAD\_REDUCED\_GRAYSCALE\_2)

    y.append(img)

X=np.array(X)

y=np.array(y)

X,y = X/255,y/255

y=np.expand\_dims(y,3)

print('The shape of input : ',X.shape)

print('The shape of mask : ',y.shape)

### **Displaying Input Image and True Mask Image**

def display(display\_list):

    plt.figure(figsize=(15, 15))

    title = ['Input Image', 'True Mask',]

    for i in range(len(display\_list)):

        plt.subplot(1, len(display\_list), i+1)

        plt.title(title[i])

        plt.imshow(tf.keras.preprocessing.image.array\_to\_img(display\_list[i]))

    plt.axis('off')

    plt.show()

display([X[0],y[0]])

# **Creating the Model**

def create\_model(INPUT\_SHAPE):

    model = Sequential()

    model.add(Conv2D(filters = 32, kernel\_size=(3, 3),strides=(1,1),padding="same", input\_shape=INPUT\_SHAPE))

    model.add(Activation('relu'))

    model.add(Dropout(0.5))

    model.add(MaxPooling2D(pool\_size=(2, 2),strides=(2,2),padding="valid"))

    model.add(Conv2D(filters = 64, kernel\_size=(3, 3),strides=(1,1),padding="same"))

    model.add(Activation('relu'))

    model.add(Conv2DTranspose(filters = 32, kernel\_size=(4, 4),strides=(2,2),padding="same"))

    model.add(Conv2D(filters = 2, kernel\_size=(1, 1),strides=(1,1),padding="valid"))

    model.add(Activation('softmax'))

    model.compile(loss='sparse\_categorical\_crossentropy',optimizer='adam',metrics=['accuracy'])

    model.summary()

    return model

model = create\_model([256,256,3])

# **Training the model**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

def displayImage(img):

    size = len(img)

    fig, ax = plt.subplots(1, size,figsize=(15,5))

    title = ['Training Image','Ground image','Predicted image']

    for i in range(size):

        ax[i].set\_title(title[i])

        if (len(img[i].shape) >= 3 and img[i].shape[2] == 2):

            img[i] = img[i][:,:,1]

        if (len(img[i].shape) == 2 ):

            ax[i].imshow(img[i])

        else:

            ax[i].imshow(tf.keras.preprocessing.image.array\_to\_img(img[i]))

        fig.tight\_layout()

    plt.show()

def show\_predictions(X\_test,y\_test):

      pred\_mask = model.predict(X\_test)

      displayImage([X\_test[3], y\_test[3], pred\_mask[3]])

from IPython.display import clear\_output

class DisplayCallback(tf.keras.callbacks.Callback):

  def on\_epoch\_end(self, epoch, logs=None):

    clear\_output(wait=True)

    show\_predictions(X\_test,y\_test)

    print ('\nSample Prediction after epoch {}\n'.format(epoch+1))

checkpoint = tf.keras.callbacks.ModelCheckpoint(r'Image\_Segmetation.h5',monitor='val\_accuracy',mode='max',save\_best\_only=True,verbose=1)

EPOCHS = 15

BS = 16

VAL\_SUBSPLITS = 5

STEPS\_PER\_EPOCH = (len(X\_train))// BS

VALIDATION\_STEPS = (len(X\_test))//BS

history = model.fit(x = X\_train,

                   y = y\_train,

                   epochs=EPOCHS,

                   batch\_size = BS,

                   steps\_per\_epoch=STEPS\_PER\_EPOCH,

                   validation\_steps=VALIDATION\_STEPS,

                   validation\_data=(X\_test,y\_test),

                   callbacks=[DisplayCallback(),checkpoint])

### **Displaying predicted images**

def displayImage(img):

    size = len(img)

    fig, ax = plt.subplots(1, size,figsize=(15,5))

    title = ['Training Image','Ground image','Predicted image']

    for i in range(size):

        ax[i].set\_title(title[i])

        if (len(img[i].shape) >= 3 and img[i].shape[2] == 2):

            img[i] = img[i][:,:,1]

        if (len(img[i].shape) == 2 ):

            ax[i].imshow(img[i])

        else:

            ax[i].imshow(tf.keras.preprocessing.image.array\_to\_img(img[i]))

    plt.show()

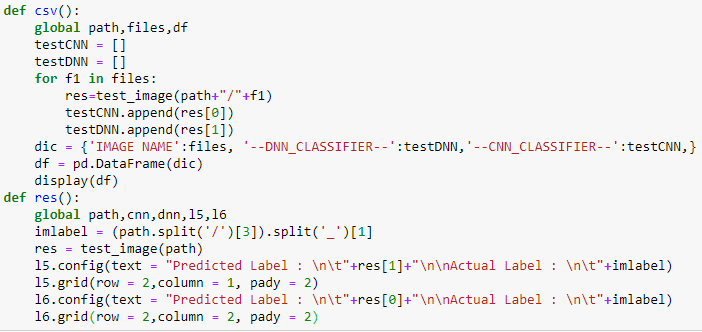
yhat = model.predict(X\_test)

li=[1,2,3,4,5,8]

for i in li:

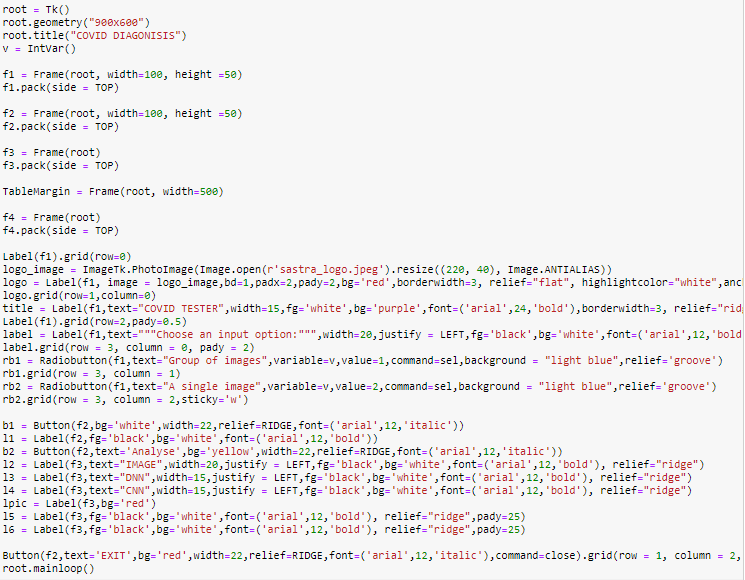
    displayImage([X\_test[i],y\_test[i],yhat[i]])

### **GUI for DNN and CNN:**





****

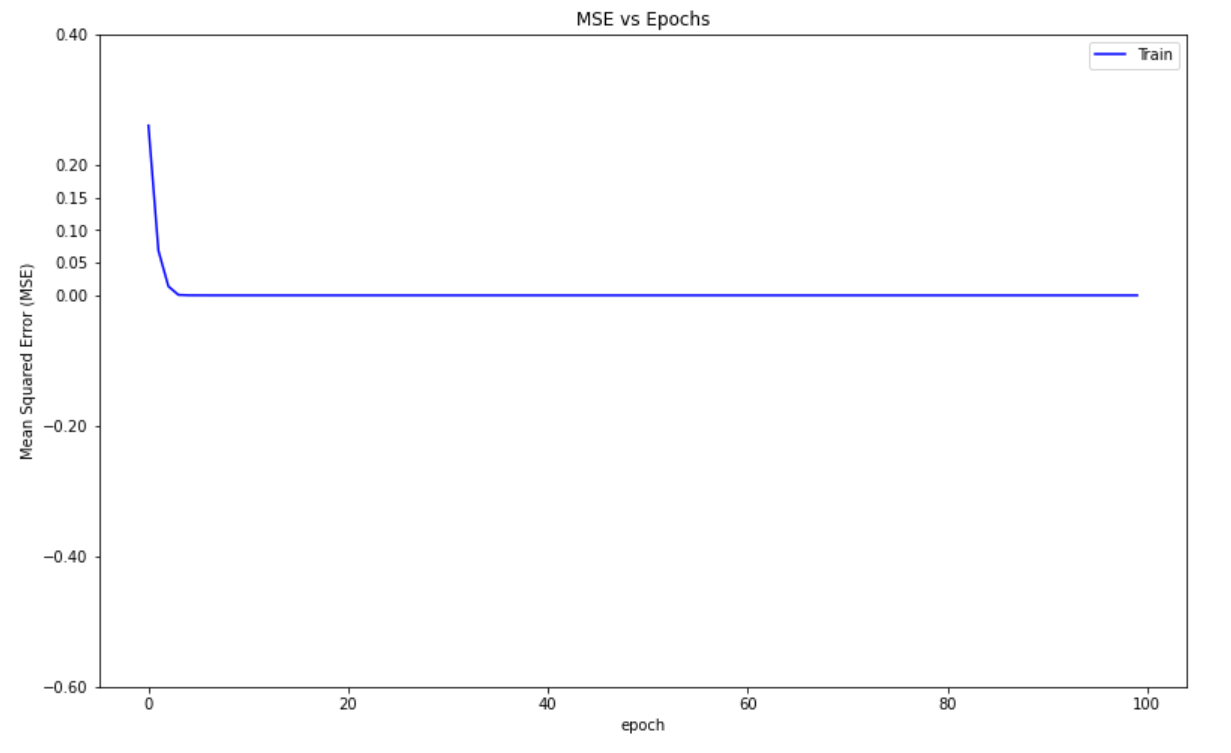
****

### **GUI for Image Segmentation:**

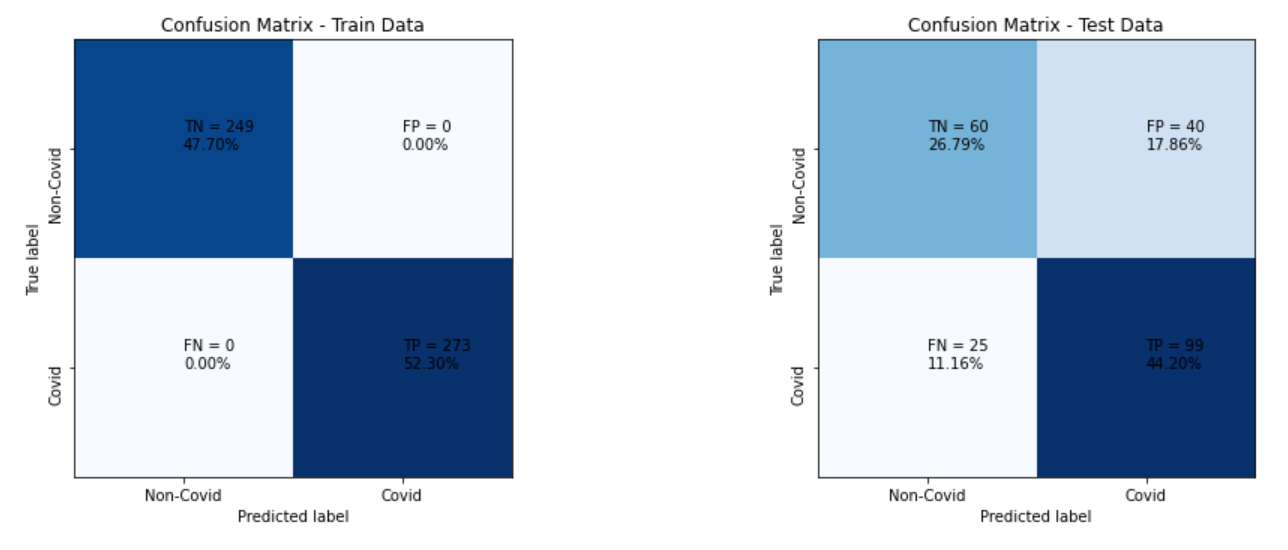
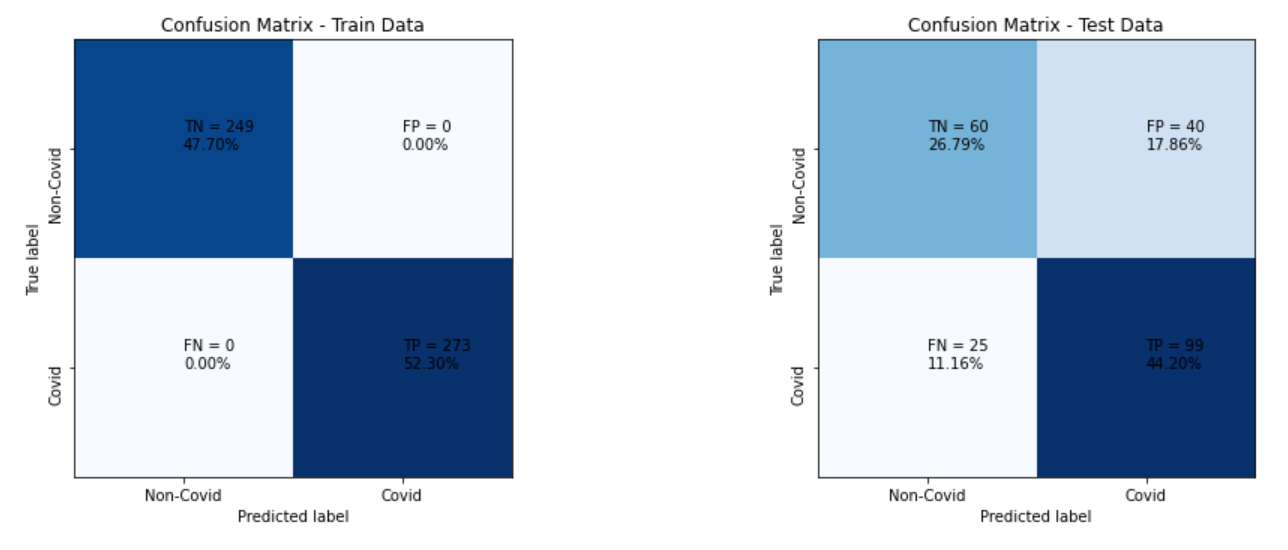
****

****

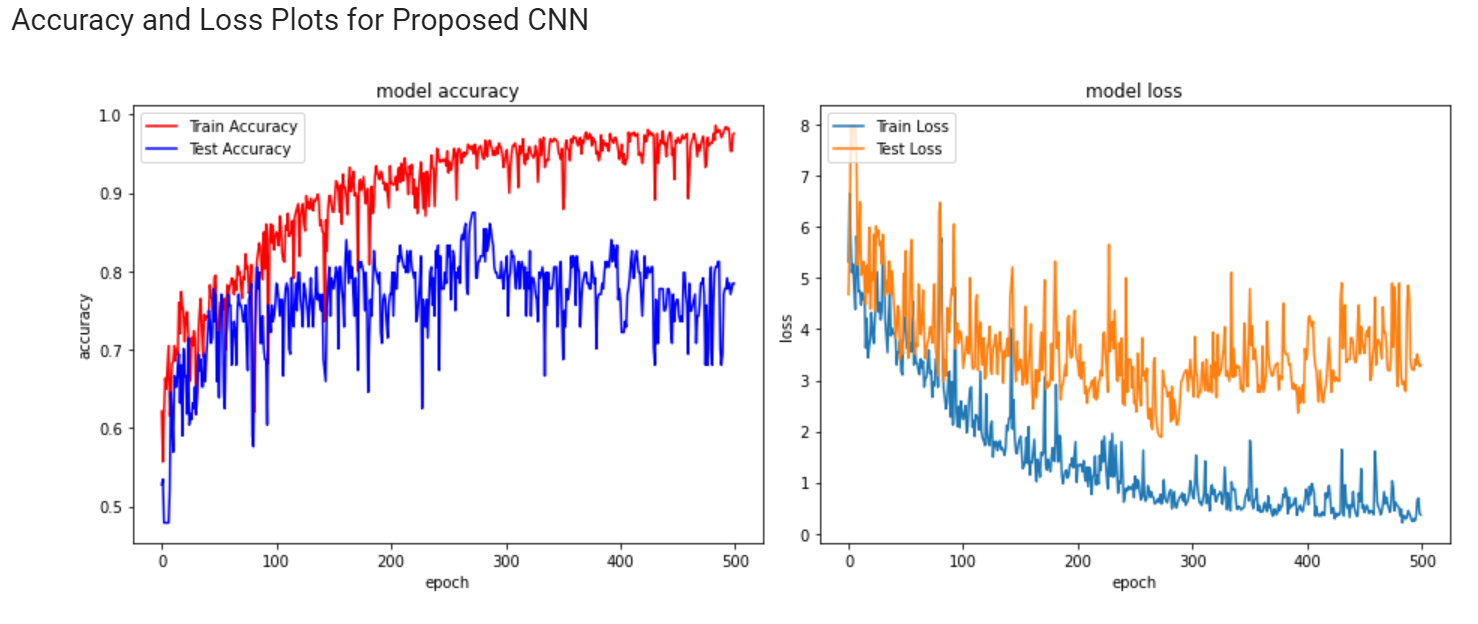
**Snapshots (Results):**



**(Fig 3.1.1 MSE vs Epoch for DNN with fractal features)**



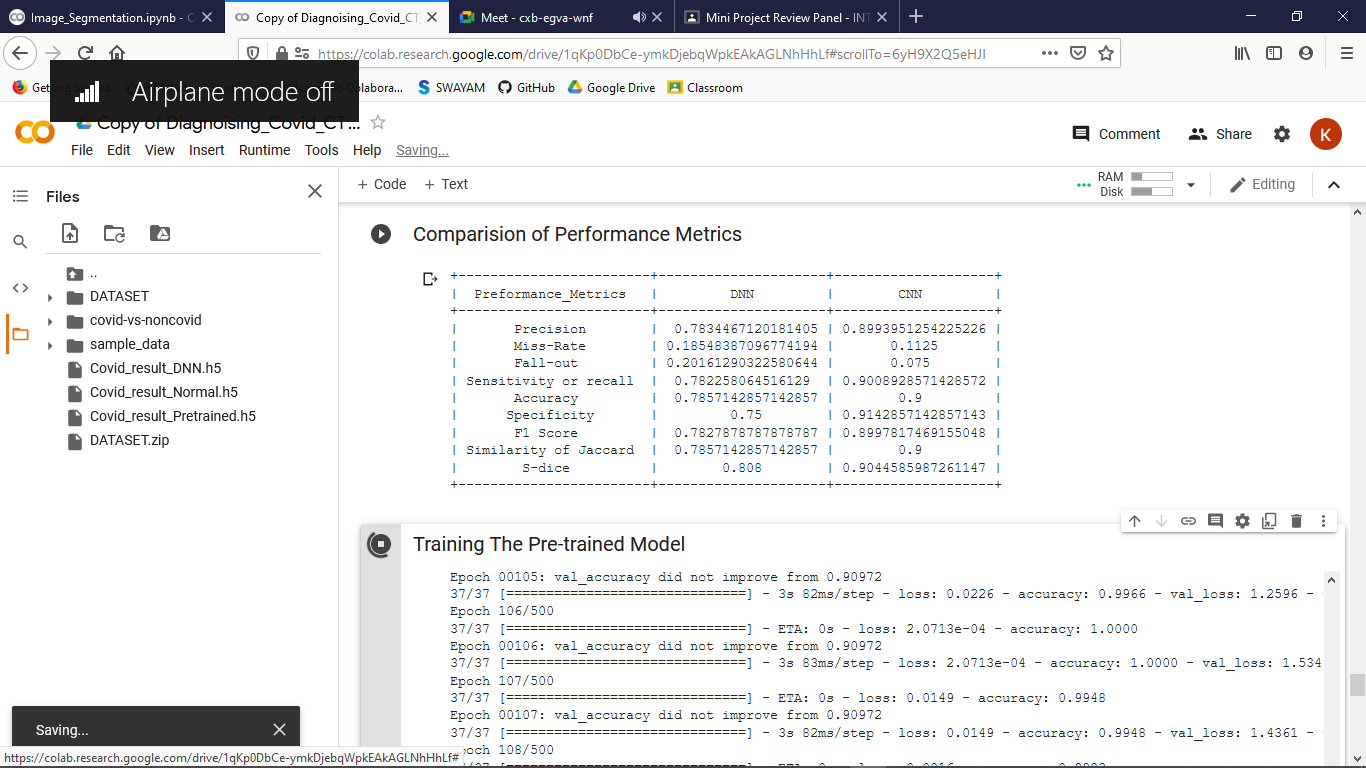
**(Fig 3.1.2 Confusion Matrix for DNN with fractal features)**

****

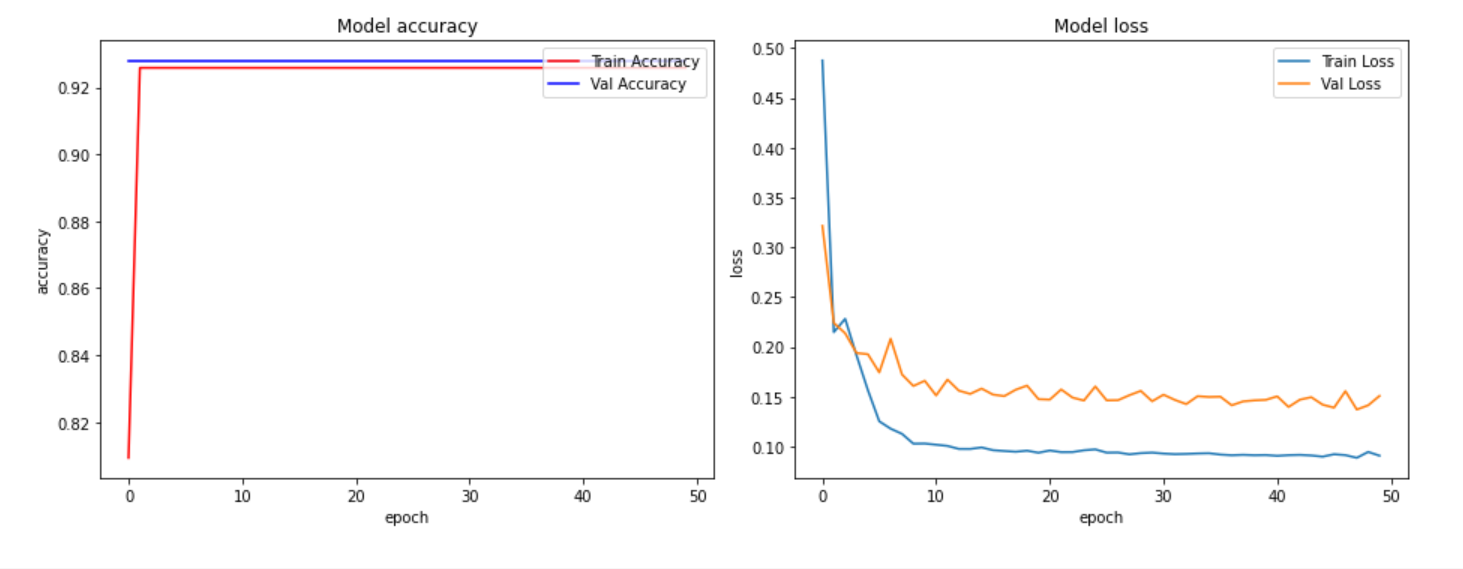
**(Fig 3.2.1 Accuracy and loss plots for CNN)**



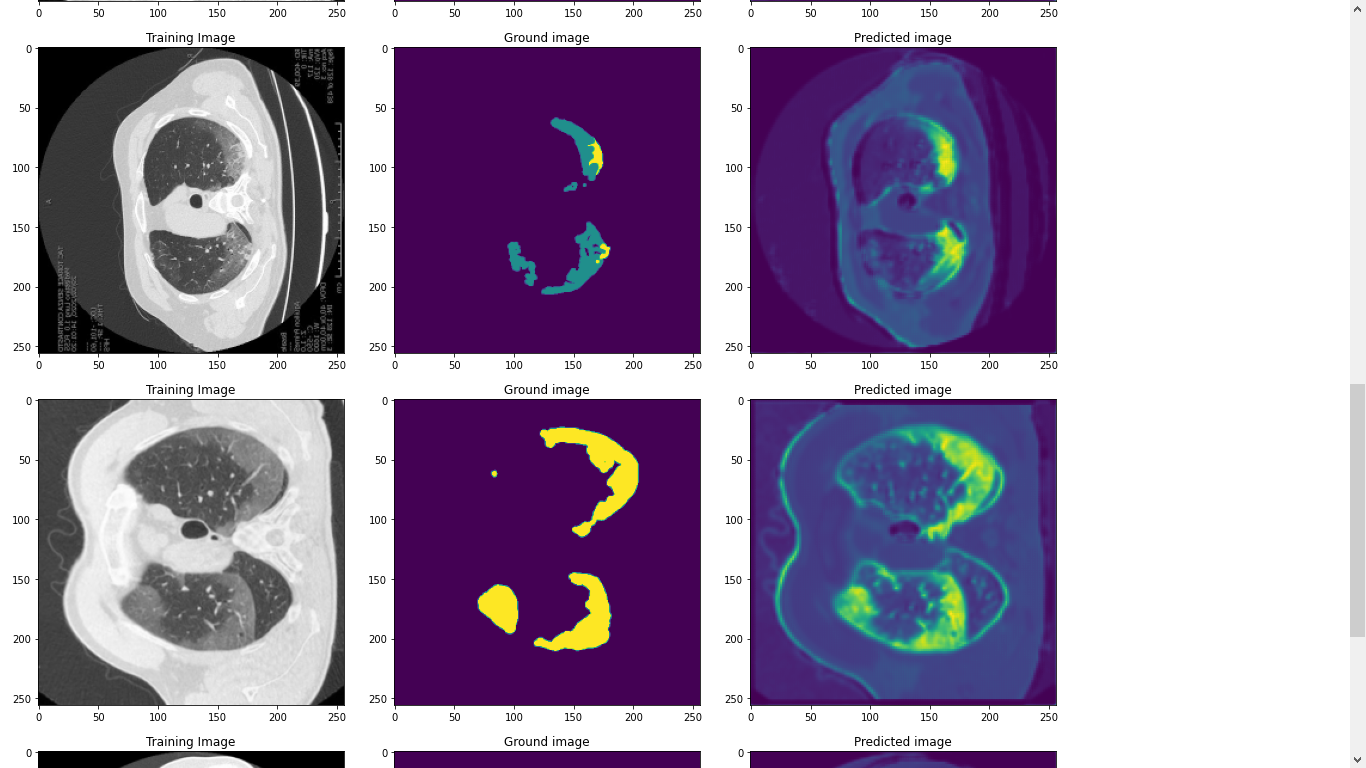
**(Fig 3.2.2 Confusion matrix for CNN)**

****

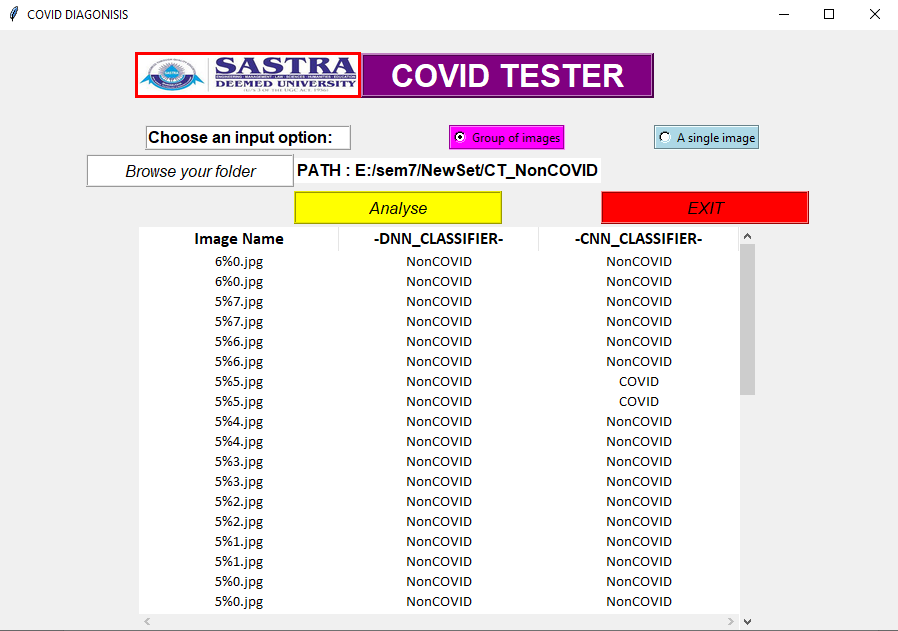
**(Fig 3.3 Comparision of Performance Metrics for DNN and CNN Models)**

****

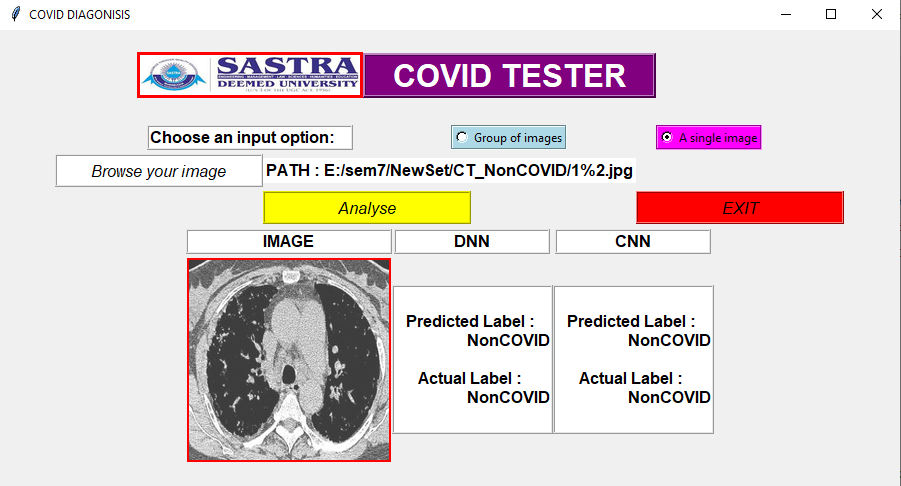
**(Fig 3.4.1 Accuracy and loss plots for Image Segmentation)**

****

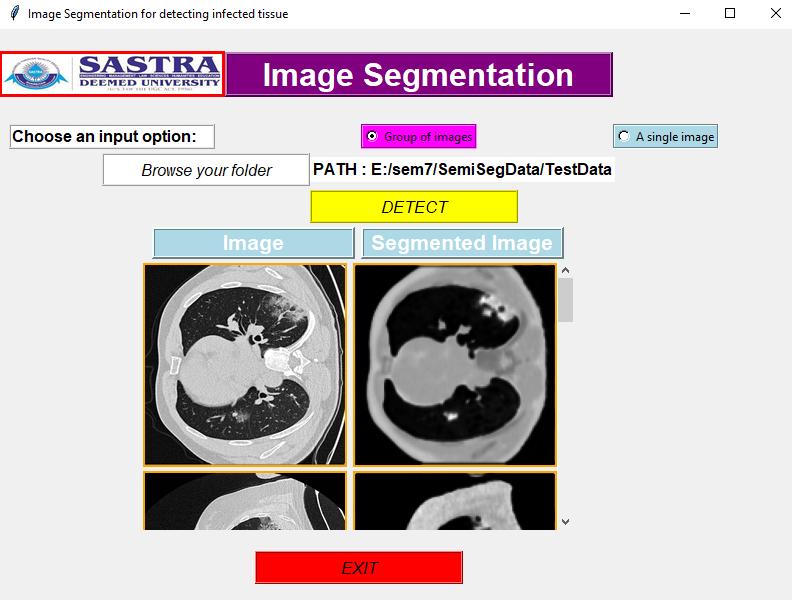
**(Fig 3.4.2 Detecting Infected tissue using Image Segmentation)**

****

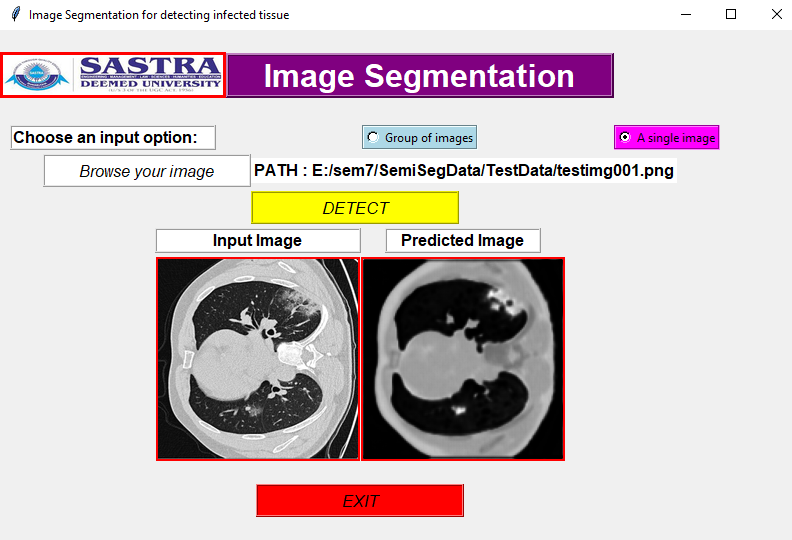
**(Fig 3.5.1 GUI for uploading folder for diagnosis)**

****

**(Fig 3.5.2 GUI for uploading single image for diagnosis)**

****

**(Fig 3.6.1 GUI for uploading folder for Image Segmentation)**

****

**(Fig 3.6.2 GUI for uploading single image for Image Segmentation)**

**Conclusion and Future plans:**

Here, we presented three sorts of deep learning strategies for the characterization and segmentation of CT X-Ray images of patient’s lungs contaminated by the COVID-19 infection. For the diagnosis, Deep Neural Network which is based on the fractal features of input images and Convolutional Neural Neural Network (CNN) with direct utilization of CT scan images have been implemented and for detection of infected regions in lung X-ray images, Image Segmentation based On CNN has been proposed.

Currently we are using Reverse Transcription Polymerase Chain Reaction (RT-PCR) which is time-consuming and expensive and the resources for this process are scarce. Hence this method will not be affordable for everyone. Results of classification shows that the presented CNN architecture with higher accuracy (90%) and sensitivity (90.08%) is outperforming than the DNN method with an accuracy of 78.22% and sensitivity of 78.22%.

Hence CNN can be used for early diagnosis of COVID-19 as a substitute for the RT-PCR method. In future better techniques with the combination of Artificial Intelligence and Deep Learning can be implemented to get results with better accuracy and specificity.

**References:**

1. False-Negative Results of Real-Time Reverse-Transcriptase Polymerase Chain Reaction for Severe Acute Respiratory Syndrome Coronavirus 2: Role of Deep-Learning-Based CT Diagnosis and Insights from Two Cases
2. Investigation of effective climatology parameters on COVID-19 outbreak in Iran
3. Detection of Coronavirus Disease (COVID-19) based on Deep Features and Support Vector Machine
4. A fully automatic deep learning system for COVID-19 diagnostic and prognostic analysis
5. AI4COVID-19: AI enabled preliminary diagnosis for COVID-19 from cough samples via an app
6. Covid-19: automatic detection from X-ray images utilizing transfer learning with convolutional neural networks