Diagnosis of Covid-19 Pneumonia using Deep Learning and Transfer learning Techniques

MSc Research Project
Data Analytics

Paritosh Diwakar Mohite
Student ID: x19199554

School of Computing
National College of Ireland

Supervisor: Prof. Hicham Rifai

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Paritosh Diwakar Mohite
Student ID:	x19199554
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Prof. Hicham Rifai
Submission Due Date:	16/08/2021
Project Title:	Diagnosis of Covid-19 Pneumonia using Deep Learning and Transfer learning Techniques
Word Count:	XXX
Page Count:	13

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	21st September 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Diagnosis of Covid-19 Pneumonia using Deep Learning and Transfer learning Techniques

Paritosh Diwakar Mohite
x19199554

1 Software and Hardware Requirement

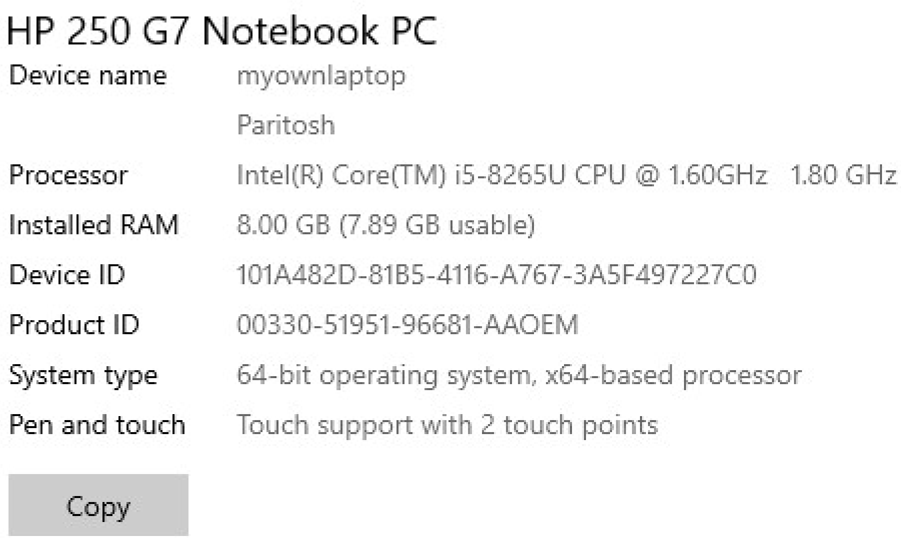

Figure 1: Hardware Requirement

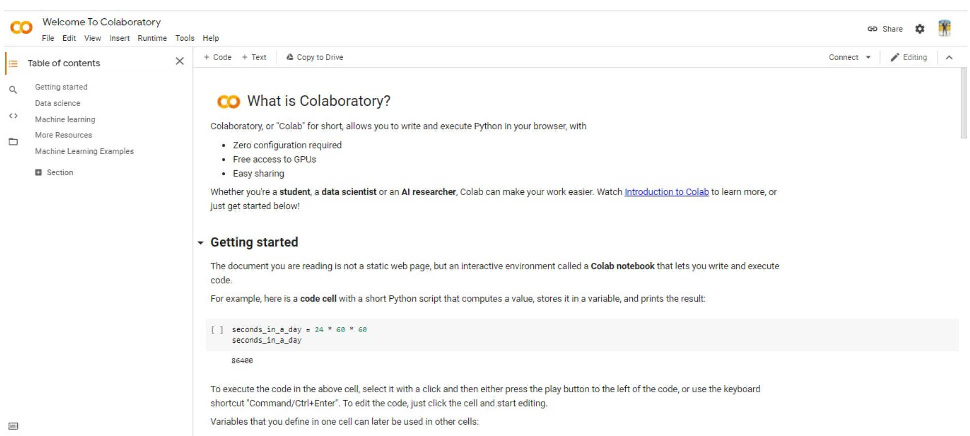

Figure 2: Software Requirement

The hardware configuration used in this research is shown in figure 1 The Software used in this research is google colab in which all the model are implemented and evaluate. It has its own many library so there is no need to install anaconda navigator is shown in figure 2

2 Google Colab Environment setup

- For setting up an environment in google coloab we need first create an account in colab
- Once the account is created we will be on the home page as shown in figure 2
- A new notebook needs to be created to start the code. for that we need to go in file and then select new notebook as shown in figure 3

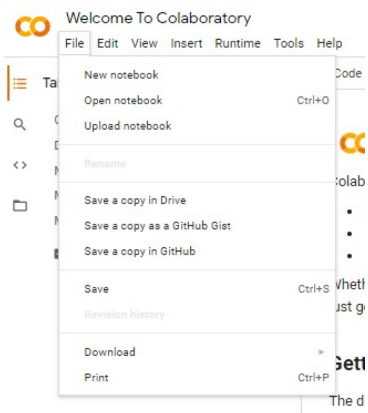

Figure 3: Notebook

- In order to run the code faster without any lagging the google colob ah an option to switch the runtime from local machine to GPU which is shown in figure 5

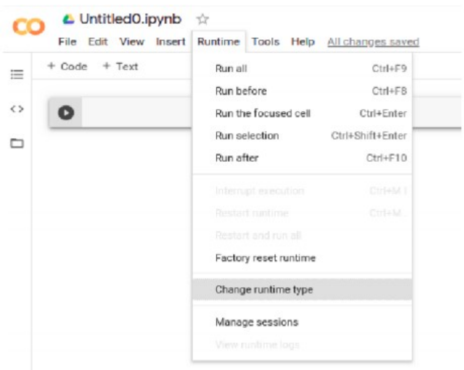

Figure 4: change run time

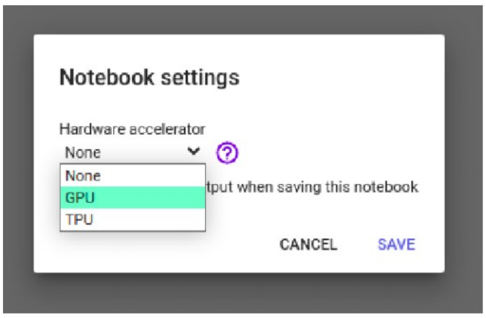

Figure 5: GPU allocation

3 Data Preparation

Certain libraries are required before starting the data preparation

```
import os
import zipfile
import datetime
from collections import Counter

import sklearn

import numpy as np
import pandas as pd
from sklearn.model_selection import StratifiedShuffleSplit
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow import keras
from tensorflow.keras import models,layers
from tensorflow.keras.utils import plot_model


import matplotlib.pyplot as plt
import PIL.Image
from IPython.display import Image
import matplotlib.pyplot as plt
```

Figure 6: Libraries before data preparation

- creating a image directory for each cases.

```
img_path = '/content/drive/My Drive/covid 19 dataset/'
covid_imgs = os.listdir(img_path+'/COVID')
normal_imgs = os.listdir(img_path+'/Normal')
pneumonia_imgs = os.listdir(img_path + '/Viral Pneumonia')
```

Figure 7: creating a image directory

- Image has been plot to check whether correct image directory is created for each cases.

```
import PIL.Image
from IPython.display import Image
import matplotlib.pyplot as plt
```

```
print("Normal chest x-ray")
im = PIL.Image.open(os.path.join(img_path,'Normal',normal_imgs[0]))
plt.imshow(im,cmap="gray")
plt.show()
print("Covid chest x-ray")
im = PIL.Image.open(os.path.join(img_path,'COVID',covid_imgs[0]))
plt.imshow(im,cmap="gray")
plt.show()
print("Viral Pneumonia chest x-ray")
im = PIL.Image.open(os.path.join(img_path,'Viral Pneumonia',pneumonia_imgs[0]))
plt.imshow(im,cmap="gray")
plt.show()
```

Figure 8: Plot Image

- After correct image directory is created now next step is to append all the images into a data frame. for that bellow figure show the steps

```python
import pandas as pd
import numpy as np

list_files = []
for dir in os.listdir(img_path):
  aux_dir = os.path.join(img_path,dir)

  if os.path.isdir(aux_dir):
    for img in os.listdir(aux_dir):
      list_files.append([os.path.join(dir,img),dir])
df = pd.DataFrame(list_files, columns=['id','label'])
print(df)
df.describe()
```

```python
from collections import Counter
def frequency_plot(df):
  freq_abs = Counter(df.label);
  freq_a = pd.DataFrame.from_dict(freq_abs, orient='index').reset_index()
  total = len(df.index)
  freq_r = freq_a[0]/total
  freq_a[1] = freq_r
  freq_a.columns = ['Label','No of images', 'Percentage']
  return freq_a
frequency_plot(df)
```

	Label	No of images	Percentage
0	COVID	3686	0.239709
1	Normal	10346	0.672823
2	Viral Pneumonia	1345	0.087468

Figure 9: Data frame creation

3.1 Data Pre-processing

In this secyion steps involved in pre-procesing data are shown,

- As that data set is imbalanced the first step is to balance the data. Imbalanced data is shown in figure 9
- once the data is balanced next step is to append those balanced image to the previous data frame.

```python
def balance_dataset(focus_class, df):
  classes = df.label.unique()
  classes_df_list = []
  focus_class_df = df[df.label == focus_class]
  classes_df_list.append(focus_class_df)
  count_focus_class = len(focus_class_df)
  ##Divide by class
  for classe in classes:
    class_df = df[df.label == classe]
    count_class = len(class_df)
    if classe == focus_class:
      continue
    if count_class > count_focus_class+1000:
      class_df = class_df.sample(count_focus_class+1000)
    classes_df_list.append(class_df)
  df_under = pd.concat(classes_df_list, axis=0)
  return df_under


df = balance_dataset('COVID',df)

frequency_plot(df)
```

	Label	No of images	Percentage
0	COVID	3686	0.379335
1	Normal	4686	0.482248
2	Viral Pneumonia	1345	0.138417

Figure 10: Data Balance

- after creating new data frame with balanced next step is to split the data into train, validation and test.

```
def split_dataset(df,test_size,random_state = 101):
  sss = StratifiedShuffleSplit(n_splits=1, test_size=test_size, random_state=random_state)
  df = df.reset_index()
  for train_index, test_index in sss.split(df['id'], df['label']):
    X_train, X_test = df['id'][train_index], df['id'][test_index]
    y_train, y_test = df['label'][train_index], df['label'][test_index]
  traindf = pd.concat([X_train, y_train], axis=1)
  testdf = pd.concat([X_test, y_test], axis=1)
  return traindf,testdf

#divide train/val/test in 80/10/10
traindf,testdf = split_dataset(df,test_size = 0.25)
valdf,testdf= split_dataset(testdf,test_size = 0.50)
# traindf = balance_dataset('COVID',df)


traindf.id = img_path + traindf.id
valdf.id = img_path + valdf.id
testdf.id = img_path + testdf.id
```

Figure 11: Data set Splitting into train, validation and testing

- Now the image augmentation and Normalization steps are performed shown in bellow figure

```
BATCH_SIZE=32
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)
train_datagen = ImageDataGenerator(
    rotation_range = 40,
    width_shift_range = 0.2,
  height_shift_range=0.2,
  shear_range = 0.2,
  zoom_range = 0.1,
  fill_mode = 'nearest')
print("Creating train generator...")
train_generator = train_datagen.flow_from_dataframe(
    dataframe=traindf,
    directory="COVID-19_Radiography_Dataset",
    x_col="id",
    y_col="label",
    batch_size=BATCH_SIZE,
    color_mode="rgb",
    seed=5,
    shuffle=True,
    class_mode="categorical",
    target_size=(224,224)
```

```
    target_size=(224,224)
)
test_generator = test_datagen.flow_from_dataframe(
    dataframe=testdf,
    directory="COVID-19_Radiography_Dataset",
    x_col="id",
    y_col="label",
    color_mode="rgb",
    seed=5,
    shuffle = False,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    target_size=(224,224)
)

print("\nCreating val generator...")
val_generator = val_datagen.flow_from_dataframe(
    dataframe=valdf,
    directory="COVID-19_Radiography_Dataset",
    x_col="id",
    y_col="label",
    batch_size=BATCH_SIZE,
    color_mode="rgb",
    seed=5,
    shuffle=True,
    class_mode="categorical",
    target_size=(224,224)
)
```

Figure 12: Data Augmentation and Normalization

4 Model Implementation

In this section steps involved in implementing a model are addressed. Before starting the model building there are some libraries that needs to be installed. Those libraries are shown in figure 29

```
import keras,os
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D , Flatten
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten, AveragePooling2D, Dropout
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.applications import Xception
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.vgg16 import VGG16
```

Figure 13: Libraries for model implementation

4.1 CNN model Implementation

```
model = Sequential()
#model.add(base_convnet)
model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Flatten())
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=3, activation="softmax"))
#compiling
model.compile(loss = 'categorical_crossentropy',
                 optimizer = keras.optimizers.Adam(0.0001),
                 metrics = ['acc'])
model.summary()
```

```
from keras.callbacks import ReduceLROnPlateau
from keras.callbacks import EarlyStopping
#Model Parameters
epochs = 20
BATCH_SIZE=32
#Callbacks
from keras.callbacks import ReduceLROnPlateau
learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss', patience=10, verbose=1, factor=0.5, min_lr=0.00001)
early_stopping_monitor = EarlyStopping(patience=100,monitor='val_loss', mode = 'min',verbose=1)

callbacks_list = [learning_rate_reduction,early_stopping_monitor]
```

```
#Verbose set to 0 to avoid Notebook visual pollution
cnn_history = model.fit(train_generator, steps_per_epoch=len(train_generator) // BATCH_SIZE,
                          validation_steps=len(test_generator) // BATCH_SIZE,
                          validation_data=val_generator, epochs=epochs,callbacks=[callbacks_list],
                          verbose=1)
```

Figure 14: Implementation of CNN model

In this section implemention steps for CNN model are shown. In figure shown bellow first all the steps involved to create the CNN model are initialized and then the steps required to compile the model added. after creating model next step shown is to initialze the early stopping rate and learning reduction rate. Then the model.fit() function used to start the model initialization with the following paramters. At last the optimizer are initialized to train the model.

4.2 VGG-16 model Implementation

Similar steps of CNN model are performed for VGG-16 as well.

```
base_model = VGG16(input_shape = (224, 224, 3), # Shape of our images
include_top = False, # Leave out the last fully connected layer
weights = 'imagenet')
for layer in base_model.layers:
    layer.trainable = False
```

```
# Flatten the output layer to 1 dimension
x = layers.Flatten()(base_model.output)

# Add a fully connected layer with 512 hidden units and ReLU activation
x = layers.Dense(512, activation='relu')(x)

# Add a dropout rate of 0.5
x = layers.Dropout(0.5)(x)

# Add a final sigmoid layer for classification
x = layers.Dense(3, activation='sigmoid')(x)

vggmodel = tf.keras.models.Model(base_model.input, x)

vggmodel.compile(optimizer = tf.keras.optimizers.RMSprop(lr=0.0001), loss = 'categorical_crossentropy',metrics = ['acc'])
```

```
vggmodel.summary()
```

```
#Callbacks
learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss', patience=10, verbose=1, factor=0.5, min_lr=0.00001)
early_stopping_monitor = EarlyStopping(patience=100,monitor='val_loss', mode = 'min',verbose=1)

callbacks_list = [learning_rate_reduction,early_stopping_monitor]


vgghist = vggmodel.fit(train_generator, steps_per_epoch=len(train_generator) // BATCH_SIZE,
                       validation_steps=len(test_generator) // BATCH_SIZE,
                       validation_data=val_generator, epochs=epochs,callbacks=[callbacks_list],
                       verbose=1)
```

Figure 15: Implementation of CNN model

4.3 InceptionV3 model Implementation

Similar steps of CNN model are performed for InceptionV3 as well.

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
base_incpmodel = InceptionV3(input_shape = (224, 224, 3), include_top = False, weights = 'imagenet')
```

```
for layer in base_incpmodel.layers:
    layer.trainable = False
```

```
from tensorflow.keras.optimizers import RMSprop

x = layers.Flatten()(base_incpmodel.output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.2)(x)
x = layers.Dense(3, activation='sigmoid')(x)

incpmodel = tf.keras.models.Model(base_incpmodel.input, x)

incpmodel.compile(optimizer = RMSprop(lr=0.0001), loss = 'categorical_crossentropy', metrics = ['acc'])
```

```
incpmodel.summary()
```

```
#Model Parameters
epochs = 20
BATCH_SIZE=32
#Callbacks
from keras.callbacks import ReduceLROnPlateau
learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss', patience=10, verbose=1, factor=0.5, min_lr=0.00001)
early_stopping_monitor = EarlyStopping(patience=100,monitor='val_loss', mode = 'min',verbose=1)

callbacks_list = [learning_rate_reduction,early_stopping_monitor]
```

```
#Callbacks
learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss', patience=10, verbose=1, factor=0.5, min_lr=0.00001)
early_stopping_monitor = EarlyStopping(patience=100,monitor='val_loss', mode = 'min',verbose=1)
callbacks_list = [learning_rate_reduction,early_stopping_monitor]
incphist = incpmodel.fit(train_generator, steps_per_epoch=len(train_generator) // BATCH_SIZE,
                         validation_steps=len(test_generator) // BATCH_SIZE,
                         validation_data=val_generator, epochs=20,callbacks=[callbacks_list],
                         verbose=1)
```

Figure 16: Implementation of InceptionV3 model

4.4 Xception model Implementation

Similar steps of CNN model are performed for Xception as well.

```python
from keras.callbacks import ReduceLROnPlateau
from keras.callbacks import EarlyStopping
#Model Parameters
epochs = 20
BATCH_SIZE=32
#Callbacks
from keras.callbacks import ReduceLROnPlateau
learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss', patience=10, verbose=1, factor=0.5, min_lr=0.00001)
early_stopping_monitor = EarlyStopping(patience=100,monitor='val_loss', mode = 'min',verbose=1)

callbacks_list = [learning_rate_reduction,early_stopping_monitor]
```

```python
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten, AveragePooling2D, Dropout
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.applications import Xception
xception = Xception(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))
outputs = xception.output
outputs = Flatten(name="flatten")(outputs)
outputs = Dropout(0.5)(outputs)
outputs = Dense(3, activation="softmax")(outputs)

xcep_model = Model(inputs=xception.input, outputs=outputs)

for layer in xception.layers:
    layer.trainable = False

xcep_model.compile(
        loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy']
)
```

```python
xcep_model.summary()
```

```python
#Verbose set to 0 to avoid Notebook visual pollution
xcep_history = xcep_model.fit(train_generator, steps_per_epoch=len(train_generator) // BATCH_SIZE,
                               validation_steps=len(test_generator) // BATCH_SIZE,
                               validation_data=val_generator, epochs=epochs,callbacks=[callbacks_list],
                               verbose=1)
```

Figure 17: Implementation of Xception model

5 Evaluation of Implemented models

In this section, evaluation steps that are required are addressed.

5.1 Evaluation of CNN model

```python
fig , ax = plt.subplots(1,2)
fig.set_size_inches(20, 8)

cnn_train_acc = cnn_history.history['acc']
cnn_train_loss = cnn_history.history['loss']
cnn_val_acc = cnn_history.history['val_acc']
cnn_val_loss = cnn_history.history['val_loss']

epochs = range(1, len(cnn_train_acc) + 1)

ax[0].plot(epochs , cnn_train_acc , 'go-' , label = 'Training Accuracy')
ax[0].plot(epochs , cnn_val_acc , 'yo-' , label = 'Validation Accuracy')
ax[0].set_title('CNN Model Training & Validation Accuracy')
ax[0].legend()
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Accuracy")

ax[1].plot(epochs , cnn_train_loss , 'go-' , label = 'Training Loss')
ax[1].plot(epochs , cnn_val_loss , 'yo-' , label = 'Validation Loss')
ax[1].set_title('CNN Model Training & Validation Loss')
ax[1].legend()
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Loss")

plt.show()
```

Figure 18: Steps for accuracy and loss graph

```
#defining target for testing
y_test = test_generator.classes
```

```
#prediction Making
y_pred = model.predict(test_generator)
y_pred = np.argmax(y_pred,axis=1)
```

Figure 19: making predictions

```
#getting confusion matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

#plotting the confusion matrix
plot_confusion_matrix(cm, classes=['COVID', 'Normal', 'Viral Pneumonia'], normalize = False)
```

Figure 20: Steps for Confusion matrix and Classification report

Firstly, accuracy and loss plot is taken into consideration as performance metric. The figure shows the steps involved Second steps to create y prediction i.e to predict the predicted value with actual value Third step is to plot confusion matrix and classification report.

5.2 Evaluation of VGG-16 model

Firstly, accuracy and loss plot is taken into consideration as performance metric. The bellow figure shows the steps involved Second steps to create y prediction i.e to predict the predicted value with actual value. Third step is to plot confusion matrix and classification report.

```
fig , ax = plt.subplots(1,2)
fig.set_size_inches(20, 8)

vgg_train_acc = vgghist.history['acc']
vgg_train_loss = vgghist.history['loss']
vgg_val_acc = vgghist.history['val_acc']
vgg_val_loss = vgghist.history['val_loss']

epochs = range(1, len(vgg_train_acc) + 1)

ax[0].plot(epochs , vgg_train_acc , 'go-' , label = 'Training Accuracy')
ax[0].plot(epochs , vgg_val_acc , 'yo-' , label = 'Validation Accuracy')
ax[0].set_title('VGG16 Model Training & Validation Accuracy')
ax[0].legend()
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Accuracy")

ax[1].plot(epochs , vgg_train_loss , 'go-' , label = 'Training Loss')
ax[1].plot(epochs , vgg_val_loss , 'yo-' , label = 'Validation Loss')
ax[1].set_title('VGG16 Model Training & Validation Loss')
ax[1].legend()
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Loss")

plt.show()
```

Figure 21: Steps for accuracy and loss graph

```
#defining target for testing
y_test = test_generator.classes
```

```
#prediction Making
y_pred = vggmodel.predict(test_generator)
y_pred = np.argmax(y_pred,axis=1)
```

Figure 22: making prediction

```
#classification report of the model
from sklearn.metrics import classification_report
print('Model: VGG16', '\n', classification_report(y_test, y_pred, target_names = ['COVID', 'Normal', 'Viral Pneumonia']))
```

```
Model: VGG16
                 precision    recall  f1-score   support

          COVID       0.87      0.93      0.90       461
         Normal       0.93      0.90      0.91       586
Viral Pneumonia       0.97      0.89      0.93       168

       accuracy                           0.91      1215
      macro avg       0.92      0.91      0.91      1215
   weighted avg       0.91      0.91      0.91      1215
```

```
classes=["COVID", "Normal", "Viral Pneumonia"]
from sklearn.metrics import confusion_matrix
import seaborn as sns
#Build Confusion Matrix
CMatrix = pd.DataFrame(confusion_matrix(y_true, y_pred), columns=classes, index =classes)

plt.figure(figsize=(12, 6))
ax = sns.heatmap(CMatrix, annot = True, fmt = 'g' ,vmin = 0, vmax = 250,cmap = 'Blues')
ax.set_xlabel('Predicted',fontsize = 14,weight = 'bold')
ax.set_xticklabels(ax.get_xticklabels(),rotation =0);

ax.set_ylabel('Actual',fontsize = 14,weight = 'bold')
ax.set_yticklabels(ax.get_yticklabels(),rotation =0);
ax.set_title('Confusion Matrix - Test Set',fontsize = 16,weight = 'bold',pad=20);
```

Figure 23: Steps for Confusion matrix and Classification report

5.3 Evaluation of InceptionV3 model

Firstly, accuracy and loss plot is taken into consideration as performance metric. The bellow figure shows the steps involved

```
fig , ax = plt.subplots(1,2)
fig.set_size_inches(20, 8)

incp_train_acc = incphist.history['acc']
incp_train_loss = incphist.history['loss']
incp_val_acc = incphist.history['val_acc']
incp_val_loss = incphist.history['val_loss']

epochs = range(1, len(incp_train_acc) + 1)

ax[0].plot(epochs , incp_train_acc , 'go-' , label = 'Training Accuracy')
ax[0].plot(epochs , incp_val_acc , 'yo-' , label = 'Validation Accuracy')
ax[0].set_title('Inception Model Training & Validation Accuracy')
ax[0].legend()
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Accuracy")

ax[1].plot(epochs , incp_train_loss , 'go-' , label = 'Training Loss')
ax[1].plot(epochs , incp_val_loss , 'yo-' , label = 'Validation Loss')
ax[1].set_title('Inception Model Training & Validation Loss')
ax[1].legend()
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Loss")
```

Figure 24: Steps for accuracy and loss graph

Second steps to create y prediction i.e to predict the predicted value with actual value Third step is to plot confusion matrix and classification report.

```
#classification report of the model
from sklearn.metrics import classification_report
print('Model: Inception', '\n', classification_report(y_test_incp, y_pred_incp, target_names = ['COVID', 'Normal', 'Viral Pneumonia']))

Model: Inception
                 precision    recall  f1-score   support

          COVID       0.96      0.64      0.77       461
         Normal       0.75      0.97      0.85       586
Viral Pneumonia       0.97      0.86      0.91       168

       accuracy                           0.83      1215
      macro avg       0.89      0.83      0.84      1215
   weighted avg       0.86      0.83      0.83      1215

y_true_incp=test_generator.classes

classes=["COVID", "Normal", "Viral Pneumonia"]
from sklearn.metrics import confusion_matrix
import seaborn as sns
#Build Confusion Matrix
CMatrix = pd.DataFrame(confusion_matrix(y_true_incp, y_pred_incp), columns=classes, index =classes)

plt.figure(figsize=(12, 6))
ax = sns.heatmap(CMatrix, annot = True, fmt = 'g' ,vmin = 0, vmax = 250,cmap = 'Blues')
ax.set_xlabel('Predicted',fontsize = 14,weight = 'bold')
ax.set_xticklabels(ax.get_xticklabels(),rotation =0);

ax.set_ylabel('Actual',fontsize = 14,weight = 'bold')
ax.set_yticklabels(ax.get_yticklabels(),rotation =0);
ax.set_title('Confusion Matrix - Test Set',fontsize = 16,weight = 'bold',pad=20);
```

Figure 25: Steps for Confusion matrix and Classification report

```
#defining target for testing
y_test_incp = test_generator.classes
```

```
#prediction Making
y_pred_incp = incpmodel.predict(test_generator)
y_pred_incp = np.argmax(y_pred_incp,axis=1)
```

Figure 26: making predictions

5.4 Evaluation of Xception model

Firstly, accuracy and loss plot is taken into consideration as performance metric. The bellow figure shows the steps involved Second steps to create y prediction i.e to predict the

```
fig , ax = plt.subplots(1,2)
fig.set_size_inches(20, 8)

xcep_train_acc = xcep_history.history['accuracy']
xcep_train_loss = xcep_history.history['loss']
xcep_val_acc = xcep_history.history['val_accuracy']
xcep_val_loss = xcep_history.history['val_loss']

epochs = range(1, len(xcep_train_acc) + 1)

ax[0].plot(epochs , xcep_train_acc , 'go-' , label = 'Training Accuracy')
ax[0].plot(epochs , xcep_val_acc , 'yo-' , label = 'Validation Accuracy')
ax[0].set_title('Xception Model Training & Validation Accuracy')
ax[0].legend()
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Accuracy")

ax[1].plot(epochs , xcep_train_loss , 'go-' , label = 'Training Loss')
ax[1].plot(epochs , xcep_val_loss , 'yo-' , label = 'Validation Loss')
ax[1].set_title('Xception Model Training & Validation Loss')
ax[1].legend()
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Loss")

plt.show()
```

Figure 27: Steps for accuracy and loss graph

predicted value with actual value Third step is to plot confusion matrix and classification report.

```
#defining target for testing
y_test_xc = test_generator.classes
```

```
#prediction Making
y_pred_xc = xcep_model.predict(test_generator)
y_pred_xc = np.argmax(y_pred_xc,axis=1)
```

Figure 28: making predictions

```
#classification report of the model
from sklearn.metrics import classification_report
print('Model: Xception', '\n', classification_report(y_test_xc, y_pred_xc, target_names = ['COVID', 'Normal', 'Viral Pneumonia']))
```

```
Model: Xception
                 precision    recall  f1-score   support

          COVID       0.87      0.89      0.88       461
         Normal       0.90      0.88      0.89       586
Viral Pneumonia       0.94      0.94      0.94       168

       accuracy                           0.89      1215
      macro avg       0.90      0.91      0.90      1215
   weighted avg       0.90      0.89      0.89      1215
```

```
classes=["COVID", "Normal", "Viral Pneumonia"]
from sklearn.metrics import confusion_matrix
import seaborn as sns
#Build Confusion Matrix
CMatrix = pd.DataFrame(confusion_matrix(y_test_xc, y_pred_xc), columns=classes, index =classes)

plt.figure(figsize=(12, 6))
ax = sns.heatmap(CMatrix, annot = True, fmt = 'g' ,vmin = 0, vmax = 250,cmap = 'Blues')
ax.set_xlabel('Predicted',fontsize = 14,weight = 'bold')
ax.set_xticklabels(ax.get_xticklabels(),rotation =0);

ax.set_ylabel('Actual',fontsize = 14,weight = 'bold')
ax.set_yticklabels(ax.get_yticklabels(),rotation =0);
ax.set_title('Confusion Matrix - Test Set',fontsize = 16,weight = 'bold',pad=20);
```

Figure 29: Steps for Confusion matrix and Classification report