Connect to Google Drive to access Dataset

```
from google.colab import drive
drive.mount('/content/drive')
%cd '/content/drive/My Drive/Covid19/BACP'
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
/content/drive/My Drive/Covid19/BACP
```

Import all Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from IPython.display import display
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, StandardScaler
from sklearn.metrics import confusion_matrix, roc_curve
from sklearn.preprocessing import LabelBinarizer
import pickle
import cv2
from glob import glob
from skimage.transform import resize
```

```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential, Model, load_model
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, AveragePooling2D, ZeroPadding2D, Dropout, Lambda
from keras.utils.np_utils import to_categorical
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import xception, vgg19, inception_v3
```

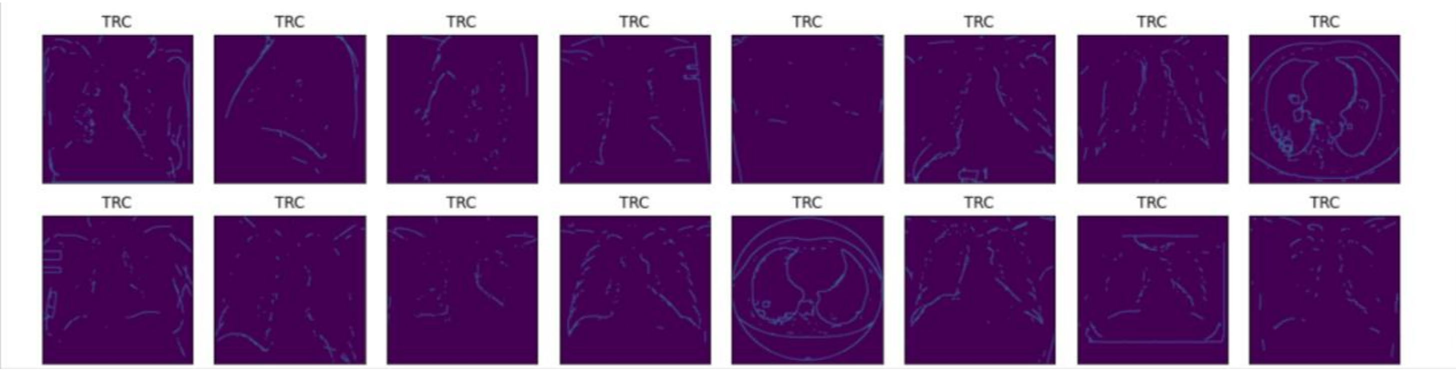
Get datasets

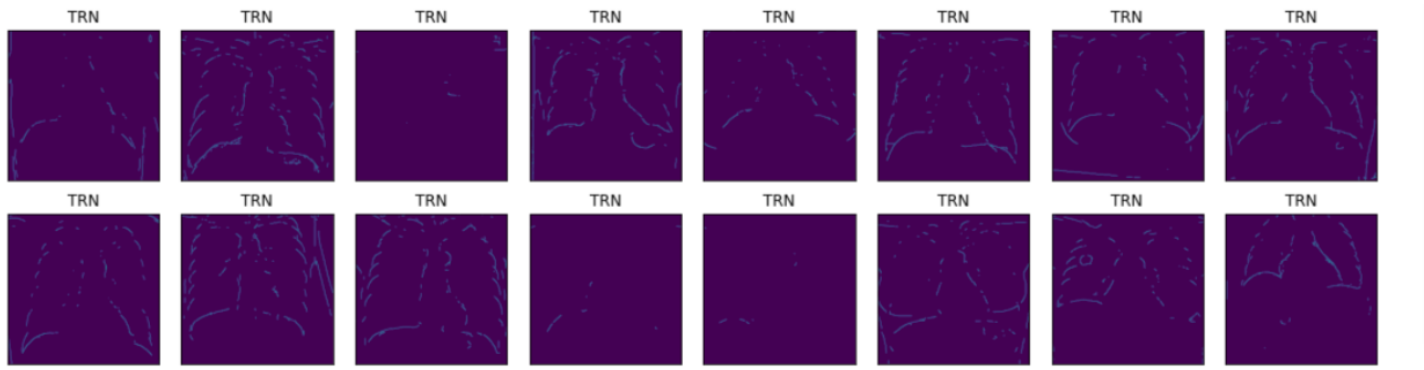
```
[ ] IMAGE_SIZE = [224, 224] # feel free to change depending on dataset

    #define paths
    train = 'CData/Train'
    valid = 'CData/Valid'
    test = 'CData/Test'

    # Use glob to grab images from path .jpg or jpeg
    trci = glob(train + '/Covid/*')
    trni = glob(train + '/Normal/*')
    vaci = glob(valid + '/Covid/*')
    vani = glob(valid + '/Normal/*')
    teci = glob(test + '/Covid/*')
    teni = glob(test + '/Normal/*')
```

Fetch Images and Class Labels from Files

```
[ ] fig, axes = plt.subplots(nrows=5, ncols=8, figsize=(15,10), subplot_kw={'xticks':[], 'yticks':[]})
    for i, ax in enumerate(axes.flat):
        img = cv2.imread(trci[i])
        img = cv2.resize(img, (224,224)) #resize images
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #convert the images to greyscale
        # img = cv2.addWeighted (img, 4, cv2.GaussianBlur(image, (0,0), 512/10), -4, 128) #apply Gaussian blur
        kernel = np.ones((5, 5), np.uint8)
        img = cv2.erode(img, kernel, iterations=3) #apply Erosion
        img = cv2.dilate(img, kernel, iterations=3) # apply Dilation
        img = cv2.Canny(img, 80, 100) #apply Canny edge detection
        ax.imshow(img)
        ax.set_title("TRC")
    fig.tight_layout()
    plt.show()

    fig, axes = plt.subplots(nrows=5, ncols=8, figsize=(15,10), subplot_kw={'xticks':[], 'yticks':[]})
    for i, ax in enumerate(axes.flat):
        img = cv2.imread(trni[i])
        img = cv2.resize(img, (224,224)) #resize images
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #convert the images to greyscale
        # img = cv2.addWeighted (img, 4, cv2.GaussianBlur(image, (0,0), 512/10), -4, 128) #apply Gaussian blur
        kernel = np.ones((5, 5), np.uint8)
        img = cv2.erode(img, kernel, iterations=3) #apply Erosion
        img = cv2.dilate(img, kernel, iterations=3) # apply Dilation
        img = cv2.Canny(img, 80, 100) #apply Canny edge detection
        ax.imshow(img)
        ax.set_title("TRN")
    fig.tight_layout()
    plt.show()
```

```python
fig, axes = plt.subplots(nrows=5, ncols=8, figsize=(15,10), subplot_kw={'xticks':[], 'yticks':[]})
for i, ax in enumerate(axes.flat):
    img = cv2.imread(vaci[i])
    img = cv2.resize(img, (224,224)) #resize images
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #convert the images to greyscale
    # img = cv2.addWeighted (img, 4, cv2.GaussianBlur(image, (0,0), 512/10), -4, 128) #apply Gaussian blur
    kernel = np.ones((5, 5), np.uint8)
    img = cv2.erode(img, kernel, iterations=3) #apply Erosion
    img = cv2.dilate(img, kernel, iterations=3) # apply Dilation
    img = cv2.Canny(img, 80, 100) #apply Canny edge detection
    ax.imshow(img)
    ax.set_title("VAC")
fig.tight_layout()
plt.show()

fig, axes = plt.subplots(nrows=5, ncols=8, figsize=(15,10), subplot_kw={'xticks':[], 'yticks':[]})
for i, ax in enumerate(axes.flat):
    img = cv2.imread(vani[i])
    img = cv2.resize(img, (224,224)) #resize images
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #convert the images to greyscale
    # img = cv2.addWeighted (img, 4, cv2.GaussianBlur(image, (0,0), 512/10), -4, 128) #apply Gaussian blur
    kernel = np.ones((5, 5), np.uint8)
    img = cv2.erode(img, kernel, iterations=3) #apply Erosion
    img = cv2.dilate(img, kernel, iterations=3) # apply Dilation
    img = cv2.Canny(img, 80, 100) #apply Canny edge detection
    ax.imshow(img)
    ax.set_title("VAN")
fig.tight_layout()
plt.show()

fig, axes = plt.subplots(nrows=5, ncols=8, figsize=(15,10), subplot_kw={'xticks':[], 'yticks':[]})
for i, ax in enumerate(axes.flat):
    img = cv2.imread(teci[i])
    img = cv2.resize(img, (224,224)) #resize images
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #convert the images to greyscale
    # img = cv2.addWeighted (img, 4, cv2.GaussianBlur(image, (0,0), 512/10), -4, 128) #apply Gaussian blur
    kernel = np.ones((5, 5), np.uint8)
    img = cv2.erode(img, kernel, iterations=3) #apply Erosion
    img = cv2.dilate(img, kernel, iterations=3) # apply Dilation
    img = cv2.Canny(img, 80, 100) #apply Canny edge detection
    ax.imshow(img)
    ax.set_title("TEC")
fig.tight_layout()
plt.show()

fig, axes = plt.subplots(nrows=5, ncols=8, figsize=(15,10), subplot_kw={'xticks':[], 'yticks':[]})
for i, ax in enumerate(axes.flat):
    img = cv2.imread(teni[i])
    img = cv2.resize(img, (224,224)) #resize images
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #convert the images to greyscale
    # img = cv2.addWeighted (img, 4, cv2.GaussianBlur(image, (0,0), 512/10), -4, 128) #apply Gaussian blur
    kernel = np.ones((5, 5), np.uint8)
    img = cv2.erode(img, kernel, iterations=3) #apply Erosion
    img = cv2.dilate(img, kernel, iterations=3) # apply Dilation
    img = cv2.Canny(img, 80, 100) #apply Canny edge detection
    ax.imshow(img)
    ax.set_title("TEN")
fig.tight_layout()
plt.show()
```





Normalizing the data to help with the training

```
[ ] trci = np.array(trci).astype('float32') / 255
    trnci = np.array(trnci).astype('float32') / 255
    vaci = np.array(vaci).astype('float32') / 255
    vanci = np.array(vanci).astype('float32') / 255
    teci = np.array(teci).astype('float32') / 255
    tenci = np.array(tenci).astype('float32') / 255
```

Train Test Split

```
[ ] x_train = np.concatenate((trci, trnci), axis=0)
    y_train = np.concatenate((trcl, trncl), axis=0)
    x_val = np.concatenate((vaci, vanci), axis=0)
    y_val = np.concatenate((vacl, vancl), axis=0)
    x_test = np.concatenate((teci, tenci), axis=0)
    y_test = np.concatenate((tecl, tencl), axis=0)
```

```
[ ] #Print the data type of x_train, y_train, x_val, y_val, x_test, y_test
    print(type(x_train),'\t',type(y_train),'\t',type(x_val),'\t',type(y_val),'\t',type(x_test),'\t',type(y_test))
```

```
<class 'numpy.ndarray'>    <class 'numpy.ndarray'>    <class 'numpy.ndarray'>    <class 'numpy.ndarray'>    <class 'numpy.ndarray'>    <class 'numpy.ndarray'>
```

```
[ ] #Get the shape of x_train, y_train, x_val, y_val x_train, y_train
    print('x_train shape:', x_train.shape)
    print('y_train shape:', y_train.shape)
    print('x_val shape:', x_val.shape)
    print('y_val shape:', y_val.shape)
    print('x_test shape:', x_test.shape)
    print('y_test shape:', y_test.shape)
```

```
x_train shape: (600, 224, 224, 3)
y_train shape: (600,)
x_val shape: (120, 224, 224, 3)
y_val shape: (120,)
x_test shape: (160, 224, 224, 3)
y_test shape: (160,)
```

Building the input vector from the 224x224 pixels

```
[ ] x_test.shape[0], x_val.shape[0], x_train.shape[0]
```

```
(160, 120, 600)
```

```
[ ] x_train = x_train.reshape(x_train.shape[0], 224, 224, 3)
    x_val = x_val.reshape(x_val.shape[0], 224, 224, 3)
    x_test = x_test.reshape(x_test.shape[0], 224, 224, 3)
```

y_train and y_test contain class lables 0 and 1

```
[ ] # make labels into categories - either 0 or 1
    y_train = to_categorical(y_train)
    y_val = to_categorical(y_val)
    y_test = to_categorical(y_test)
    print("Shape: ", y_train.shape,'\t',y_val.shape,'\t',y_test.shape)
```

```
Shape:  (600, 2)        (120, 2)        (160, 2)
```

Build VGG19-Model

```
[ ] vggModel = vgg19.VGG19(weights="imagenet", include_top=False, input_shape=(224, 224, 3))

    outputs = vggModel.output
    outputs = Flatten(name="flatten")(outputs)
    outputs = Dropout(0.5)(outputs)
    outputs = Dense(2, activation="softmax")(outputs)

    model1 = Model(inputs=vggModel.input, outputs=outputs)

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

    #Image Augmentation
    train_aug = ImageDataGenerator(rotation_range=20, width_shift_range=0.2, height_shift_range=0.2, horizontal_flip=True)
```

```
[ ] model1.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_1 (InputLayer)        [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)       (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160
_________________________________________________________________
```

Compliling Model

```
[ ] model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Train the Model

```
[ ] hist = model1.fit(train_aug.flow(x_train, y_train, batch_size=64), validation_data=(x_val, y_val),epochs=100)
```

```
Epoch 1/100
10/10 [==============================] - 41s 2s/step - loss: 1.8308 - accuracy: 0.4516 - val_loss: 0.9683 - val_accuracy: 0.5167
Epoch 2/100
10/10 [==============================] - 7s 655ms/step - loss: 0.9457 - accuracy: 0.5592 - val_loss: 0.7607 - val_accuracy: 0.6583
Epoch 3/100
10/10 [==============================] - 7s 659ms/step - loss: 0.7805 - accuracy: 0.6508 - val_loss: 0.7217 - val_accuracy: 0.7000
Epoch 4/100
10/10 [==============================] - 7s 662ms/step - loss: 0.6486 - accuracy: 0.6973 - val_loss: 0.5142 - val_accuracy: 0.7667
Epoch 5/100
10/10 [==============================] - 7s 661ms/step - loss: 0.6082 - accuracy: 0.7133 - val_loss: 0.4227 - val_accuracy: 0.8083
Epoch 6/100
10/10 [==============================] - 7s 652ms/step - loss: 0.5486 - accuracy: 0.7379 - val_loss: 0.5485 - val_accuracy: 0.7333
Epoch 7/100
10/10 [==============================] - 7s 653ms/step - loss: 0.6311 - accuracy: 0.7016 - val_loss: 0.4276 - val_accuracy: 0.8000
Epoch 8/100
10/10 [==============================] - 7s 670ms/step - loss: 0.4756 - accuracy: 0.7857 - val_loss: 0.4242 - val_accuracy: 0.7833
Epoch 9/100
10/10 [==============================] - 7s 661ms/step - loss: 0.4893 - accuracy: 0.7401 - val_loss: 0.3530 - val_accuracy: 0.8667
Epoch 10/100
10/10 [==============================] - 7s 663ms/step - loss: 0.4669 - accuracy: 0.7661 - val_loss: 0.3997 - val_accuracy: 0.8333
Epoch 11/100
10/10 [==============================] - 7s 665ms/step - loss: 0.5170 - accuracy: 0.7463 - val_loss: 0.3170 - val_accuracy: 0.8833
```

Model Evaluate

```
[ ]  score = model1.evaluate(x_test, y_test, verbose=0)
     print('Test loss:', score[0])
     print('Test accuracy:', score[1])
```

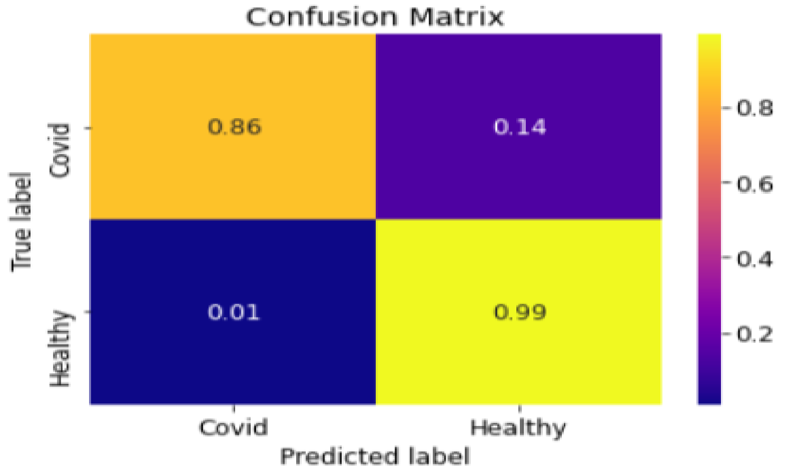
```
Test loss: 0.2084309309720993
Test accuracy: 0.893750011920929
```

```
[ ]  y_pred_train1 = model1.predict(x_train, batch_size=64)

     y_pred_test1 = model1.predict(x_test, batch_size=64)
```

Plot Confusion Matrix

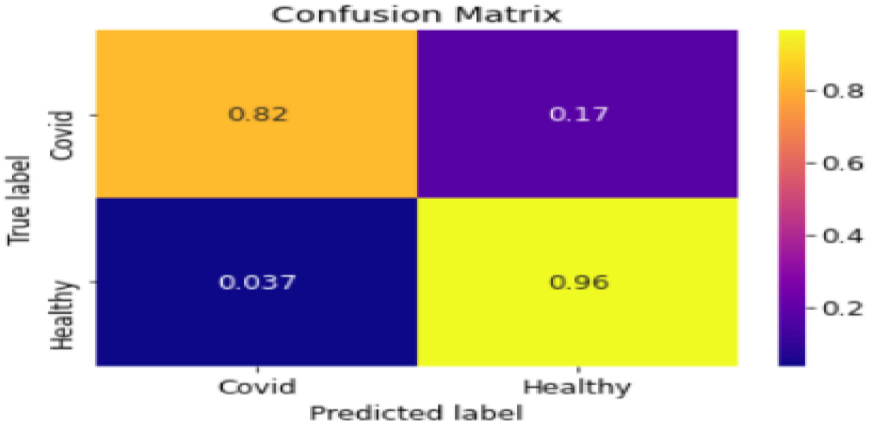
```
[ ]  def plot_confusion_matrix(normalize):
       classes = ['Covid','Healthy']
       tick_marks = [0.5,1.5]
       cn = confusion_matrix(y_train_bin11, y_pred_bin11, normalize=normalize)
       sns.heatmap(cn,cmap='plasma',annot=True)
       plt.xticks(tick_marks, classes)
       plt.yticks(tick_marks, classes)
       plt.title('Confusion Matrix')
       plt.ylabel('True label')
       plt.xlabel('Predicted label')
       plt.show()

     print('Confusion Matrix with Normalized Values')
     plot_confusion_matrix(normalize='true')
```

```
Confusion Matrix with Normalized Values
```



```
def plot_confusion_matrix(normalize):
  classes = ['Covid','Healthy']
  tick_marks = [0.5,1.5]
  cn = confusion_matrix(y_test_bin21, y_pred_bin21,normalize=normalize)
  sns.heatmap(cn,cmap='plasma',annot=True)
  plt.xticks(tick_marks, classes)
  plt.yticks(tick_marks, classes)
  plt.title('Confusion Matrix')
  plt.ylabel('True label')
  plt.xlabel('Predicted label')
  plt.show()

print('Confusion Matrix with Normalized Values')
plot_confusion_matrix(normalize='true')
```

```
Confusion Matrix with Normalized Values
```



Classification Report

```
from sklearn.metrics import classification_report
print(classification_report(y_train_bin11, y_pred_bin11))
```

```
              precision    recall  f1-score   support

           0       0.99      0.86      0.92       300
           1       0.88      0.99      0.93       300

    accuracy                           0.93       600
   macro avg       0.93      0.93      0.92       600
weighted avg       0.93      0.93      0.92       600
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test_bin21, y_pred_bin21))
```

```
              precision    recall  f1-score   support

           0       0.96      0.82      0.89        80
           1       0.85      0.96      0.90        80

    accuracy                           0.89       160
   macro avg       0.90      0.89      0.89       160
weighted avg       0.90      0.89      0.89       160
```

Build Inceptionv3-Model

```
[ ]  inception = inception_v3.InceptionV3(weights="imagenet", include_top=False, input_shape=(224, 224, 3))

     outputs = inception.output
     outputs = Flatten(name="flatten")(outputs)
     outputs = Dropout(0.5)(outputs)
     outputs = Dense(2, activation="softmax")(outputs)

     model2 = Model(inputs=inception.input, outputs=outputs)

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

     #Image Augmentation
     train_aug = ImageDataGenerator(rotation_range=20, width_shift_range=0.2, height_shift_range=0.2, horizontal_flip=True)
```

```
[ ]  model2.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 111, 111, 32) 864         input_2[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 111, 111, 32) 96          conv2d[0][0]
__________________________________________________________________________________________________
activation (Activation)         (None, 111, 111, 32) 0           batch_normalization[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 109, 109, 32) 9216        activation[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 109, 109, 32) 96          conv2d_1[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, 109, 109, 32) 0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 109, 109, 64) 18432       activation_1[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 109, 109, 64) 192         conv2d_2[0][0]
__________________________________________________________________________________________________
activation_2 (Activation)       (None, 109, 109, 64) 0           batch_normalization_2[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 54, 54, 64)   0           activation_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 54, 54, 80)   5120        max_pooling2d[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 54, 54, 80)   240         conv2d_3[0][0]
__________________________________________________________________________________________________
activation_3 (Activation)       (None, 54, 54, 80)   0           batch_normalization_3[0][0]
__________________________________________________________________________________________________
```

Compiling Model

```
[ ] model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Train the Model

```
[ ] histo = model2.fit(train_aug.flow(x_train, y_train, batch_size=64), validation_data=(x_val, y_val),epochs=100)
```

```
Epoch 1/100
10/10 [==============================] - 31s 1s/step - loss: 5.9453 - accuracy: 0.5187 - val_loss: 2.5141 - val_accuracy: 0.7333
Epoch 2/100
10/10 [==============================] - 6s 567ms/step - loss: 1.9035 - accuracy: 0.7658 - val_loss: 1.5069 - val_accuracy: 0.7917
Epoch 3/100
10/10 [==============================] - 6s 562ms/step - loss: 0.8319 - accuracy: 0.8517 - val_loss: 1.5002 - val_accuracy: 0.7667
Epoch 4/100
10/10 [==============================] - 6s 564ms/step - loss: 0.8768 - accuracy: 0.8338 - val_loss: 0.8213 - val_accuracy: 0.7583
Epoch 5/100
10/10 [==============================] - 6s 627ms/step - loss: 0.5823 - accuracy: 0.8560 - val_loss: 0.9832 - val_accuracy: 0.8083
Epoch 6/100
10/10 [==============================] - 6s 561ms/step - loss: 0.5957 - accuracy: 0.8686 - val_loss: 0.6948 - val_accuracy: 0.7833
Epoch 7/100
10/10 [==============================] - 6s 564ms/step - loss: 0.6268 - accuracy: 0.8396 - val_loss: 1.4014 - val_accuracy: 0.7500
Epoch 8/100
10/10 [==============================] - 6s 563ms/step - loss: 0.5617 - accuracy: 0.8648 - val_loss: 1.4977 - val_accuracy: 0.7667
Epoch 9/100
10/10 [==============================] - 6s 555ms/step - loss: 0.6422 - accuracy: 0.8655 - val_loss: 0.8874 - val_accuracy: 0.7667
Epoch 10/100
10/10 [==============================] - 6s 575ms/step - loss: 0.7117 - accuracy: 0.8401 - val_loss: 1.7827 - val_accuracy: 0.7667
Epoch 11/100
10/10 [==============================] - 6s 569ms/step - loss: 0.8423 - accuracy: 0.8529 - val_loss: 0.7491 - val_accuracy: 0.7583
Epoch 12/100
10/10 [==============================] - 6s 563ms/step - loss: 0.6374 - accuracy: 0.8719 - val_loss: 0.8756 - val_accuracy: 0.8083
```

Model Evaluate

```
[ ] score = model2.evaluate(x_test, y_test, verbose=0)
    print('Test loss:', score[0])
    print('Test accuracy:', score[1])
```

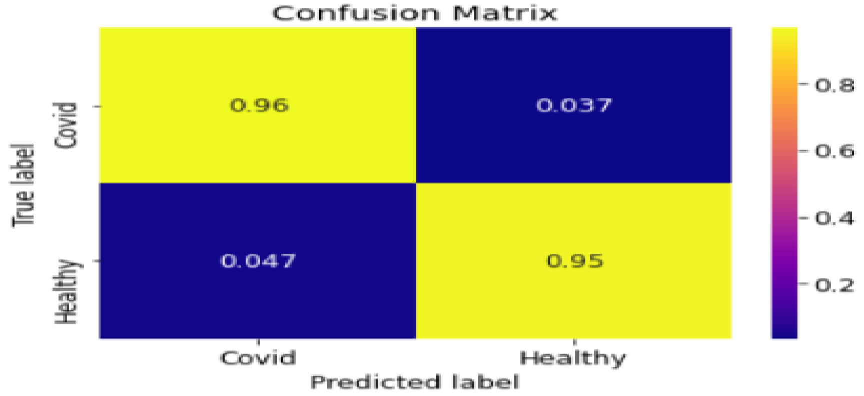
```
Test loss: 1.3081718683242798
Test accuracy: 0.8374999761581421
```

```
[ ] y_pred_train2 = model2.predict(x_train, batch_size=64)

    y_pred_test2 = model2.predict(x_test, batch_size=64)
```

Plot Confusion Matrix

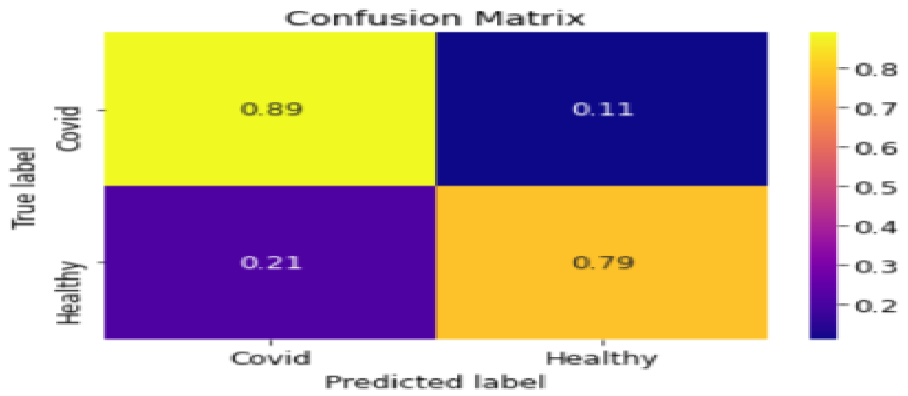
```
[ ] def plot_confusion_matrix(normalize):
      classes = ['Covid','Healthy']
      tick_marks = [0.5,1.5]
      cn = confusion_matrix(y_train_bin12, y_pred_bin12, normalize=normalize)
      sns.heatmap(cn,cmap='plasma',annot=True)
      plt.xticks(tick_marks, classes)
      plt.yticks(tick_marks, classes)
      plt.title('Confusion Matrix')
      plt.ylabel('True label')
      plt.xlabel('Predicted label')
      plt.show()

    print('Confusion Matrix with Normalized Values')
    plot_confusion_matrix(normalize='true')
```

```
Confusion Matrix with Normalized Values
```




```
[ ] def plot_confusion_matrix(normalize):
      classes = ['Covid','Healthy']
      tick_marks = [0.5,1.5]
      cn = confusion_matrix(y_test_bin22, y_pred_bin22,normalize=normalize)
      sns.heatmap(cn,cmap='plasma',annot=True)
      plt.xticks(tick_marks, classes)
      plt.yticks(tick_marks, classes)
      plt.title('Confusion Matrix')
      plt.ylabel('True label')
      plt.xlabel('Predicted label')
      plt.show()

    print('Confusion Matrix with Normalized Values')
    plot_confusion_matrix(normalize='true')
```

```
Confusion Matrix with Normalized Values
```



Classification Report

```
[ ] from sklearn.metrics import classification_report
    print(classification_report(y_train_bin12, y_pred_bin12))
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.96       300
           1       0.96      0.95      0.96       300

    accuracy                           0.96       600
   macro avg       0.96      0.96      0.96       600
weighted avg       0.96      0.96      0.96       600
```

```
[ ] from sklearn.metrics import classification_report
    print(classification_report(y_test_bin22, y_pred_bin22))
```

```
              precision    recall  f1-score   support

           0       0.81      0.89      0.85        80
           1       0.88      0.79      0.83        80

    accuracy                           0.84       160
   macro avg       0.84      0.84      0.84       160
weighted avg       0.84      0.84      0.84       160
```

Build Xception-Model

```
[ ] xception = xception.Xception(weights="imagenet", include_top=False, input_shape=(224, 224, 3))

    outputs = xception.output
    outputs = Flatten(name="flatten")(outputs)
    outputs = Dropout(0.5)(outputs)
    outputs = Dense(2, activation="softmax")(outputs)

    model3 = Model(inputs=xception.input, outputs=outputs)

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

    #Image Augmentation
    train_aug = ImageDataGenerator(rotation_range=20, width_shift_range=0.2, height_shift_range=0.2, horizontal_flip=True)
```

```
[ ] model3.summary()
```

```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
block1_conv1 (Conv2D)           (None, 111, 111, 32) 864         input_3[0][0]
__________________________________________________________________________________________________
block1_conv1_bn (BatchNormaliza (None, 111, 111, 32) 128         block1_conv1[0][0]
__________________________________________________________________________________________________
block1_conv1_act (Activation)   (None, 111, 111, 32) 0           block1_conv1_bn[0][0]
__________________________________________________________________________________________________
block1_conv2 (Conv2D)           (None, 109, 109, 64) 18432       block1_conv1_act[0][0]
__________________________________________________________________________________________________
block1_conv2_bn (BatchNormaliza (None, 109, 109, 64) 256         block1_conv2[0][0]
__________________________________________________________________________________________________
block1_conv2_act (Activation)   (None, 109, 109, 64) 0           block1_conv2_bn[0][0]
__________________________________________________________________________________________________
block2_sepconv1 (SeparableConv2 (None, 109, 109, 128 8768        block1_conv2_act[0][0]
__________________________________________________________________________________________________
block2_sepconv1_bn (BatchNormal (None, 109, 109, 128 512         block2_sepconv1[0][0]
__________________________________________________________________________________________________
block2_sepconv2_act (Activation (None, 109, 109, 128 0           block2_sepconv1_bn[0][0]
__________________________________________________________________________________________________
block2_sepconv2 (SeparableConv2 (None, 109, 109, 128 17536       block2_sepconv2_act[0][0]
__________________________________________________________________________________________________
block2_sepconv2_bn (BatchNormal (None, 109, 109, 128 512         block2_sepconv2[0][0]
__________________________________________________________________________________________________
conv2d_94 (Conv2D)              (None, 55, 55, 128)  8192        block1_conv2_act[0][0]
__________________________________________________________________________________________________
block2_pool (MaxPooling2D)      (None, 55, 55, 128)  0           block2_sepconv2_bn[0][0]
__________________________________________________________________________________________________
batch_normalization_94 (BatchNo (None, 55, 55, 128)  512         conv2d_94[0][0]
```

Compiling Model

```
[ ] model3.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Train the Model

```
[ ] hi = model3.fit(train_aug.flow(x_train, y_train, batch_size=64), validation_data=(x_val, y_val),epochs=100)
```

```
Epoch 1/100
10/10 [==============================] - 19s 1s/step - loss: 4.1605 - accuracy: 0.5784 - val_loss: 4.5045 - val_accuracy: 0.6417
Epoch 2/100
10/10 [==============================] - 7s 639ms/step - loss: 1.9294 - accuracy: 0.7559 - val_loss: 4.8418 - val_accuracy: 0.6667
Epoch 3/100
10/10 [==============================] - 7s 646ms/step - loss: 1.9514 - accuracy: 0.8100 - val_loss: 1.2673 - val_accuracy: 0.7833
Epoch 4/100
10/10 [==============================] - 7s 655ms/step - loss: 1.2073 - accuracy: 0.8422 - val_loss: 1.2271 - val_accuracy: 0.8083
Epoch 5/100
10/10 [==============================] - 7s 657ms/step - loss: 0.8273 - accuracy: 0.8505 - val_loss: 1.0143 - val_accuracy: 0.8250
Epoch 6/100
10/10 [==============================] - 7s 663ms/step - loss: 0.6258 - accuracy: 0.8762 - val_loss: 0.9149 - val_accuracy: 0.8250
Epoch 7/100
10/10 [==============================] - 7s 699ms/step - loss: 0.5334 - accuracy: 0.8897 - val_loss: 1.1478 - val_accuracy: 0.7750
Epoch 8/100
10/10 [==============================] - 7s 636ms/step - loss: 0.4330 - accuracy: 0.8864 - val_loss: 0.9800 - val_accuracy: 0.8417
Epoch 9/100
10/10 [==============================] - 7s 693ms/step - loss: 0.2835 - accuracy: 0.9289 - val_loss: 0.7969 - val_accuracy: 0.8167
Epoch 10/100
10/10 [==============================] - 7s 645ms/step - loss: 0.4675 - accuracy: 0.8876 - val_loss: 0.4846 - val_accuracy: 0.8500
Epoch 11/100
```

Model Evaluate

```
[ ]  score = model3.evaluate(x_test, y_test, verbose=0)
     print('Test loss:', score[0])
     print('Test accuracy:', score[1])
```

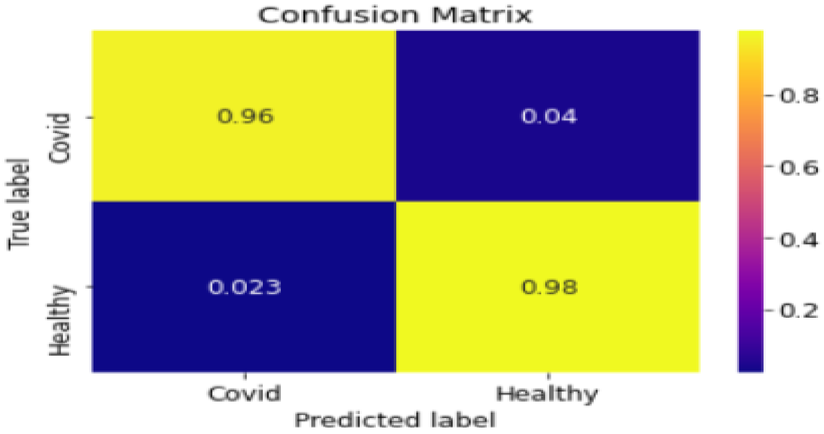
```
Test loss: 1.0250107049942017
Test accuracy: 0.84375
```

```
[ ]  y_pred_train3 = model3.predict(x_train, batch_size=64)

     y_pred_test3 = model3.predict(x_test, batch_size=64)
```

Plot Confusion Matrix

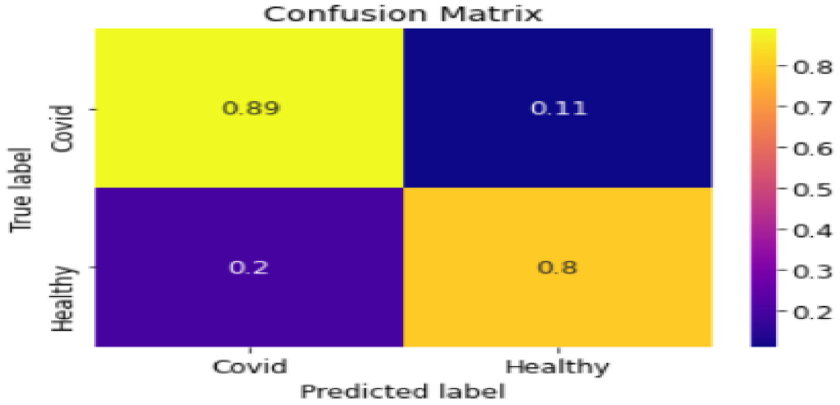
```
[ ] def plot_confusion_matrix(normalize):
      classes = ['Covid','Healthy']
      tick_marks = [0.5,1.5]
      cn = confusion_matrix(y_train_bin13, y_pred_bin13, normalize=normalize)
      sns.heatmap(cn,cmap='plasma',annot=True)
      plt.xticks(tick_marks, classes)
      plt.yticks(tick_marks, classes)
      plt.title('Confusion Matrix')
      plt.ylabel('True label')
      plt.xlabel('Predicted label')
      plt.show()

    print('Confusion Matrix with Normalized Values')
    plot_confusion_matrix(normalize='true')
```

```
Confusion Matrix with Normalized Values
```



```
[ ] def plot_confusion_matrix(normalize):
      classes = ['Covid','Healthy']
      tick_marks = [0.5,1.5]
      cn = confusion_matrix(y_test_bin23, y_pred_bin23,normalize=normalize)
      sns.heatmap(cn,cmap='plasma',annot=True)
      plt.xticks(tick_marks, classes)
      plt.yticks(tick_marks, classes)
      plt.title('Confusion Matrix')
      plt.ylabel('True label')
      plt.xlabel('Predicted label')
      plt.show()

    print('Confusion Matrix with Normalized Values')
    plot_confusion_matrix(normalize='true')
```

```
Confusion Matrix with Normalized Values
```



Classification Report

```
[ ] from sklearn.metrics import classification_report
    print(classification_report(y_train_bin13, y_pred_bin13))
```

```
              precision    recall  f1-score   support

           0       0.98      0.96      0.97       300
           1       0.96      0.98      0.97       300

    accuracy                           0.97       600
   macro avg       0.97      0.97      0.97       600
weighted avg       0.97      0.97      0.97       600
```

```
[ ] from sklearn.metrics import classification_report
    print(classification_report(y_test_bin23, y_pred_bin23))
```

```
              precision    recall  f1-score   support

           0       0.82      0.89      0.85        80
           1       0.88      0.80      0.84        80

    accuracy                           0.84       160
   macro avg       0.85      0.84      0.84       160
weighted avg       0.85      0.84      0.84       160
```

Save the Model

```
[ ] model1.save('Cmodel.h5')
```