```
!pip3 install opencv-python
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dis
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dis
```

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

```
Mounted at /content/drive
```

```
#Import relevant libraries
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import warnings
from IPython.display import display
# filter warnings
warnings.filterwarnings('ignore')
```

```
#Import relevant libraries for building the models
from keras.applications.vgg19 import VGG19
from keras.utils import to_categorical
import cv2
import numpy as np
from keras.layers import Dense, Flatten
from glob import glob
```

```
X = np.load('/content/drive/MyDrive/breast_cancer/X.npy') # images
Y = np.load('/content/drive/MyDrive/breast_cancer/Y.npy') # labels associated to
```

Data Analysis

```
print(X)
```

```
  [[170 110 154]
   [163 115 156]
```

```
   [166  99 140]
   ...
   [149 107 149]
   [142 102 144]
   [148  90 142]]]


 [[[167  99 143]
   [155  93 138]
   [166 105 148]
   ...
   [181 148 182]
   [192 153 181]
   [115  78 122]]

  [[180 127 161]
   [163 106 147]
   [177 112 147]
   ...
   [142  99 146]
   [150 110 155]
   [189 151 182]]

  [[163  97 144]
   [175 113 149]
   [173 113 153]
   ...
   [150  92 141]
   [192 161 191]
   [161 115 150]]

  ...

  [[147  91 140]
   [144  95 143]
   [151 110 154]
   ...
   [177 136 173]
   [151 113 155]
   [192 154 184]]

  [[140  95 144]
   [132  69 118]
   [157  80 128]
   ...
   [202 172 197]
   [164 128 169]
   [194 154 187]]

  [[164 121 160]
```

```
        [137  74 124]
        [155  84 130]
        ...
        [172 133 173]
        [157 115 161]
        [188 150 185]]]]
```

```
print(Y) #Exploring the labels associated to images in this dataset (0 = no IDC,
```

```
[0 0 0 ... 1 1 1]
```

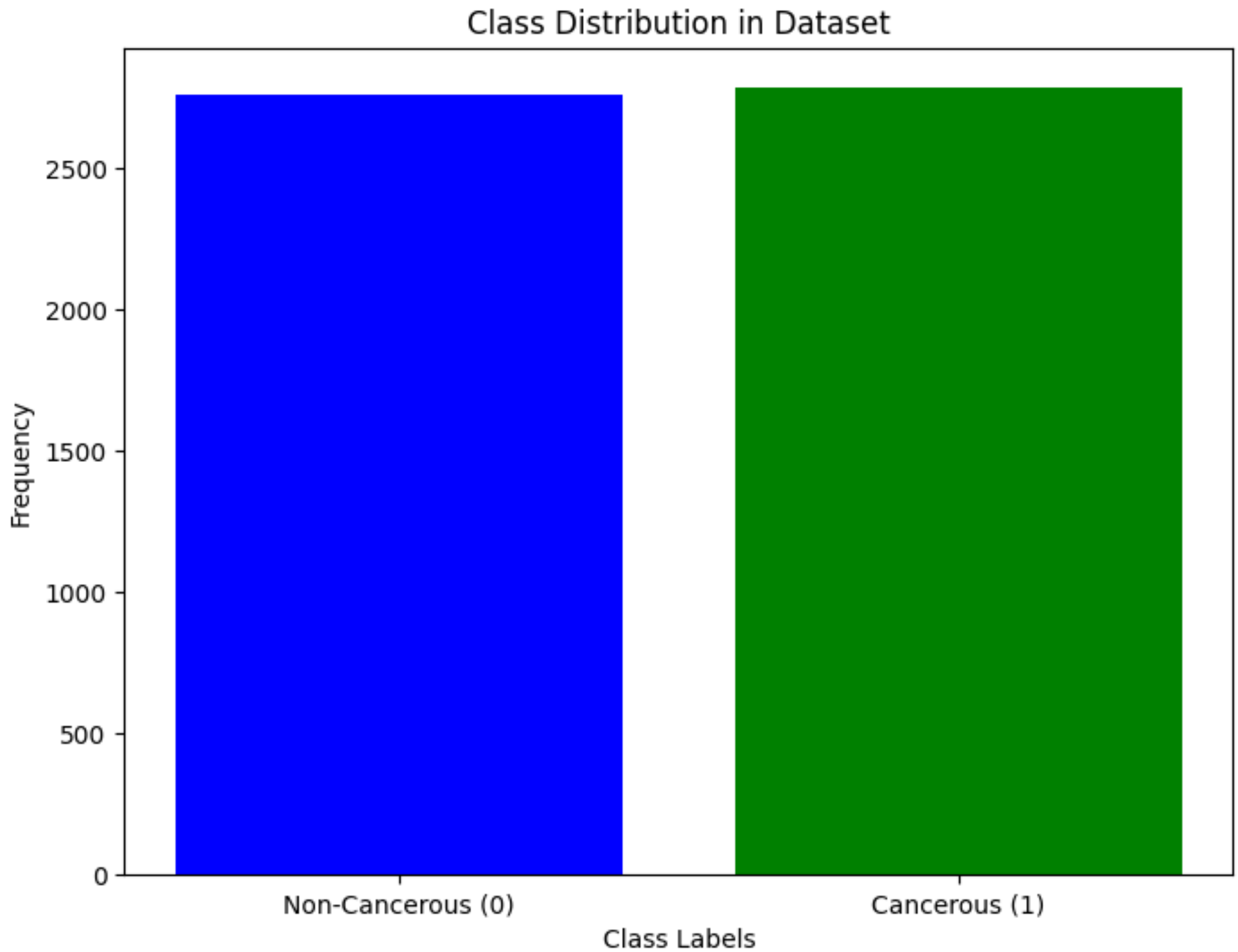
```
#Printing dimensions of dataset
print("X shape: ", X.shape)
print("Y shape: ", Y.shape)
```

```
X shape:  (5547, 50, 50, 3)
Y shape:  (5547,)
```

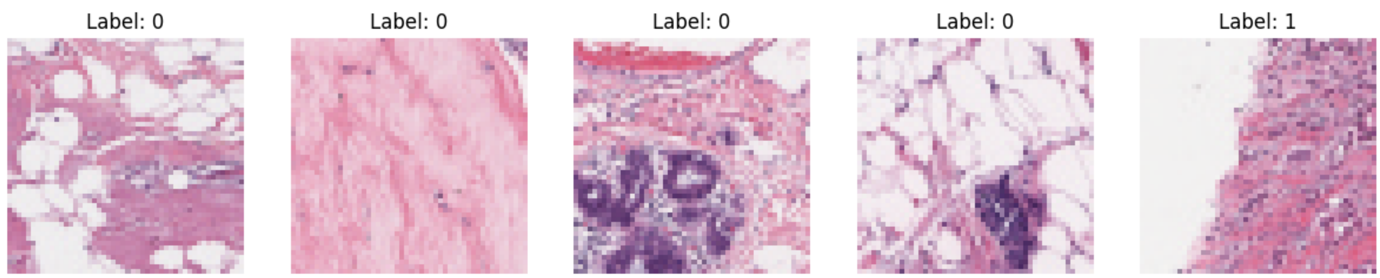
```
missing_values_in_X = np.isnan(X).any() #Check for missing values in the image da
missing_values_in_Y = np.isnan(Y).any() #Check for missing values in the labels Y

print(missing_values_in_X)
print(missing_values_in_Y)
```

```
False
False
```

```
#Check for imbalance in the Dataset
unique, counts = np.unique(Y, return_counts=True)
class_counts = dict(zip(unique, counts))
plt.figure(figsize=(8, 6))
plt.bar(class_counts.keys(), class_counts.values(), color=['blue', 'green'])
plt.xlabel('Class Labels')
plt.ylabel('Frequency')
plt.title('Class Distribution in Dataset')
plt.xticks(ticks=[0, 1], labels=['Non-Cancerous (0)', 'Cancerous (1)'])
plt.show()
```



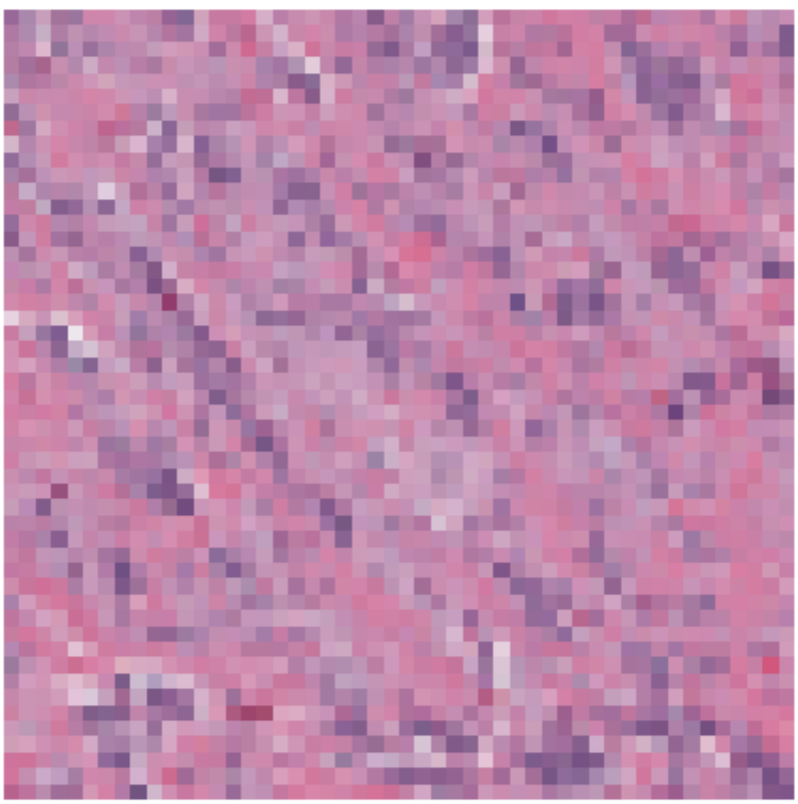
```
def plot_images(X, Y, num_images=5):
    fig, axes = plt.subplots(1, num_images, figsize=(15, 3))

    for i, ax in enumerate(axes):
        #Randomly pick an image and display it
        idx = np.random.randint(0, X.shape[0])
        ax.imshow(X[idx])
        ax.set_title('Label: ' + str(Y[idx]))
        ax.axis('off')

    plt.show()


plot_images(X, Y, num_images=5)
```



```
import numpy as np
import matplotlib.pyplot as plt
import ipywidgets as widgets
from IPython.display import display

def interactive_image_display(X, Y):
    def view_image(index):
        plt.imshow(X[index])
        plt.title('IDC Present: ' + ('Yes' if Y[index] == 1 else 'No'))
        plt.axis('off')
        plt.show()


    index_slider = widgets.IntSlider(value=0, min=0, max=len(X) - 1, step=1, desc
    widgets.interact(view_image, index=index_slider)

interactive_image_display(X, Y)
```

Image Index 4018

IDC Present: Yes

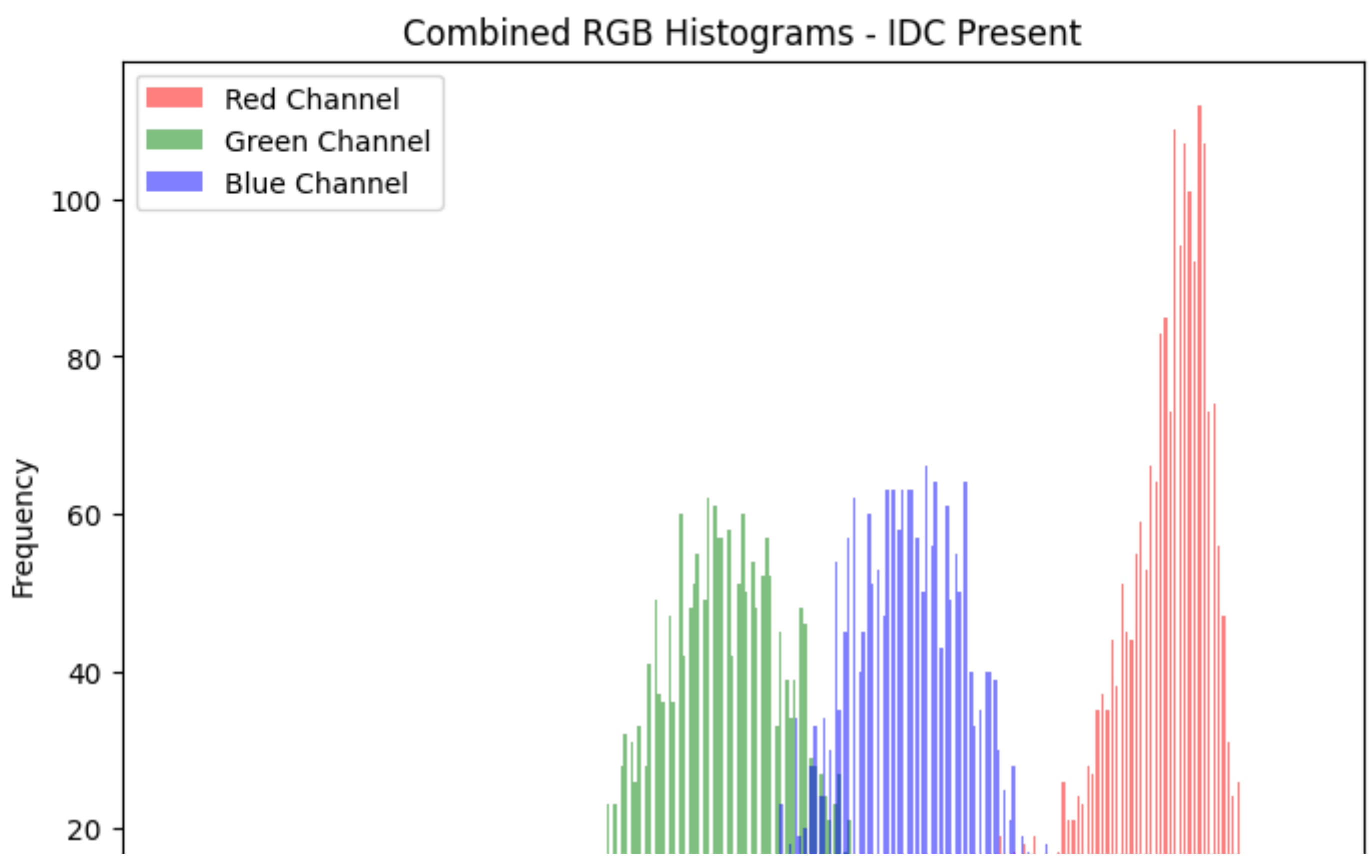


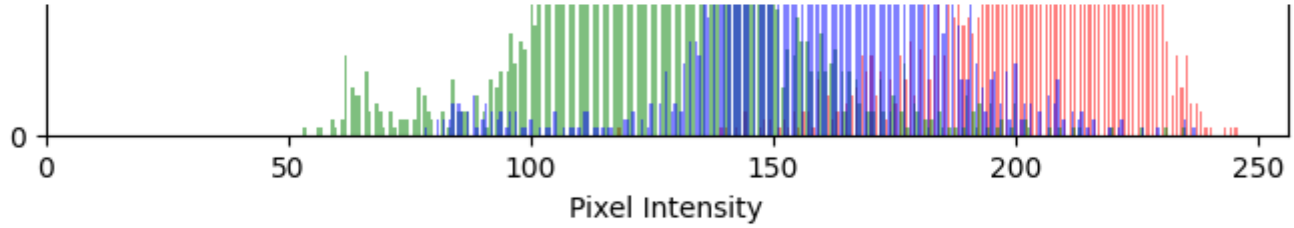
```
import numpy as np

# Assuming X is your image array with shape (5547, 50, 50, 3)

# Calculate the mean for each channel
red_mean = np.mean(X[:, :, :, 0])
green_mean = np.mean(X[:, :, :, 1])
blue_mean = np.mean(X[:, :, :, 2])

# Calculate the standard deviation for each channel
red_std = np.std(X[:, :, :, 0])
green_std = np.std(X[:, :, :, 1])
blue_std = np.std(X[:, :, :, 2])

# Calculate the minimum value for each channel
red_min = np.min(X[:, :, :, 0])
green_min = np.min(X[:, :, :, 1])
blue_min = np.min(X[:, :, :, 2])

# Calculate the maximum value for each channel
red_max = np.max(X[:, :, :, 0])
green_max = np.max(X[:, :, :, 1])
blue_max = np.max(X[:, :, :, 2])

# Print all results in a concise format
print(f"Mean values for each channel: R = {red_mean:.2f}, G = {green_mean:.2f}, B
print(f"Standard deviation for each channel: R = {red_std:.2f}, G = {green_std:.2
print(f"Minimum values for each channel: R = {red_min}, G = {green_min}, B = {blu
print(f"Maximum values for each channel: R = {red_max}, G = {green_max}, B = {blu
```

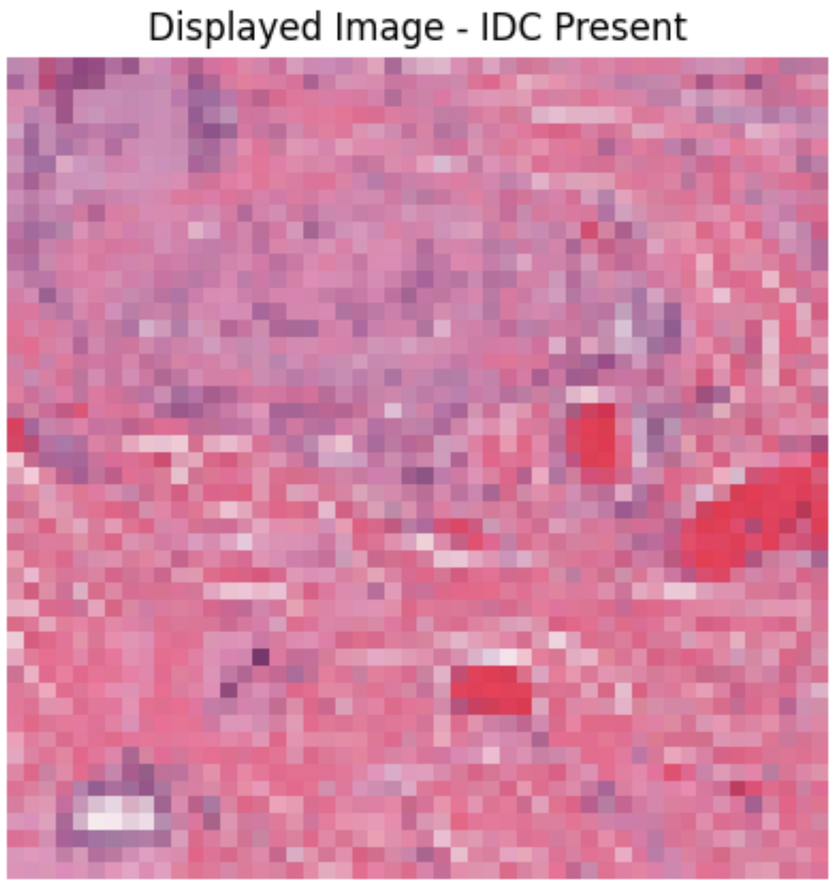
```
Mean values for each channel: R = 205.79, G = 161.87, B = 187.44
Standard deviation for each channel: R = 36.29, G = 53.94, B = 38.69
Minimum values for each channel: R = 4, G = 2, B = 5
Maximum values for each channel: R = 255, G = 255, B = 255
```

```
def plot_combined_rgb_histograms(X, Y):
    #Select a random image
    idx = np.random.randint(0, X.shape[0])
    image = X[idx]
    label = Y[idx]
    label_text = 'IDC Present' if label == 1 else 'No IDC'
    plt.figure(figsize=(8, 6))

    #Colors and labels for each channel
```

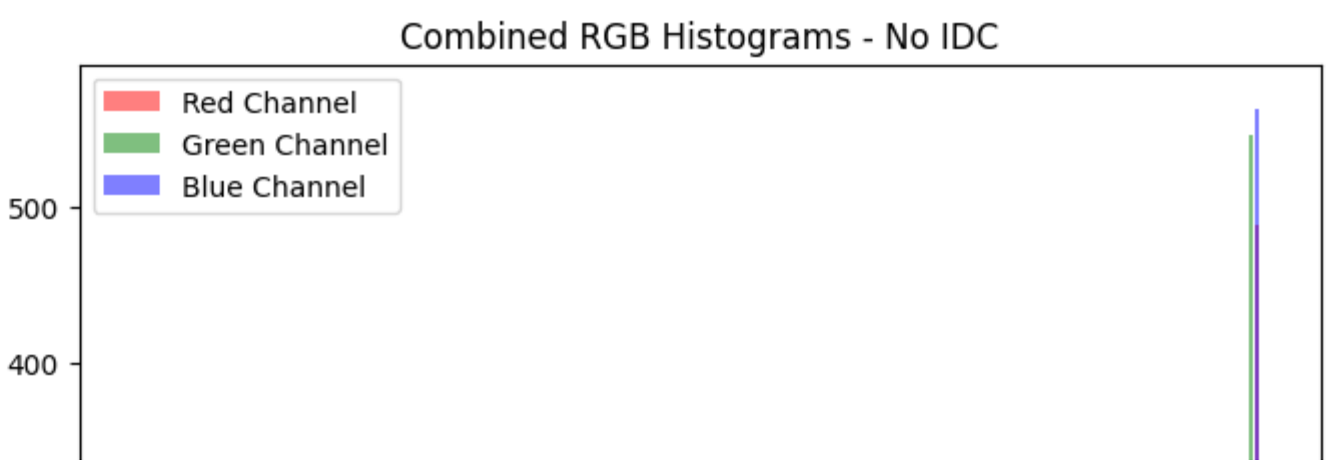
```
    colors = ['red', 'green', 'blue']
    channel_labels = ['Red Channel', 'Green Channel', 'Blue Channel']

    #Plot histograms for each channel
    for i, color in enumerate(colors):
        plt.hist(image[:, :, i].ravel(), bins=256, color=color, alpha=0.5, label=

    plt.title(f'Combined RGB Histograms - {label_text}')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')
    plt.xlim([0, 256])
    plt.legend()

    plt.show()

    #Display the selected image
    plt.figure(figsize=(5, 5))
    plt.imshow(image)
    plt.title(f'Displayed Image - {label_text}')
    plt.axis('off')
    plt.show()

plot_combined_rgb_histograms(X, Y)
```

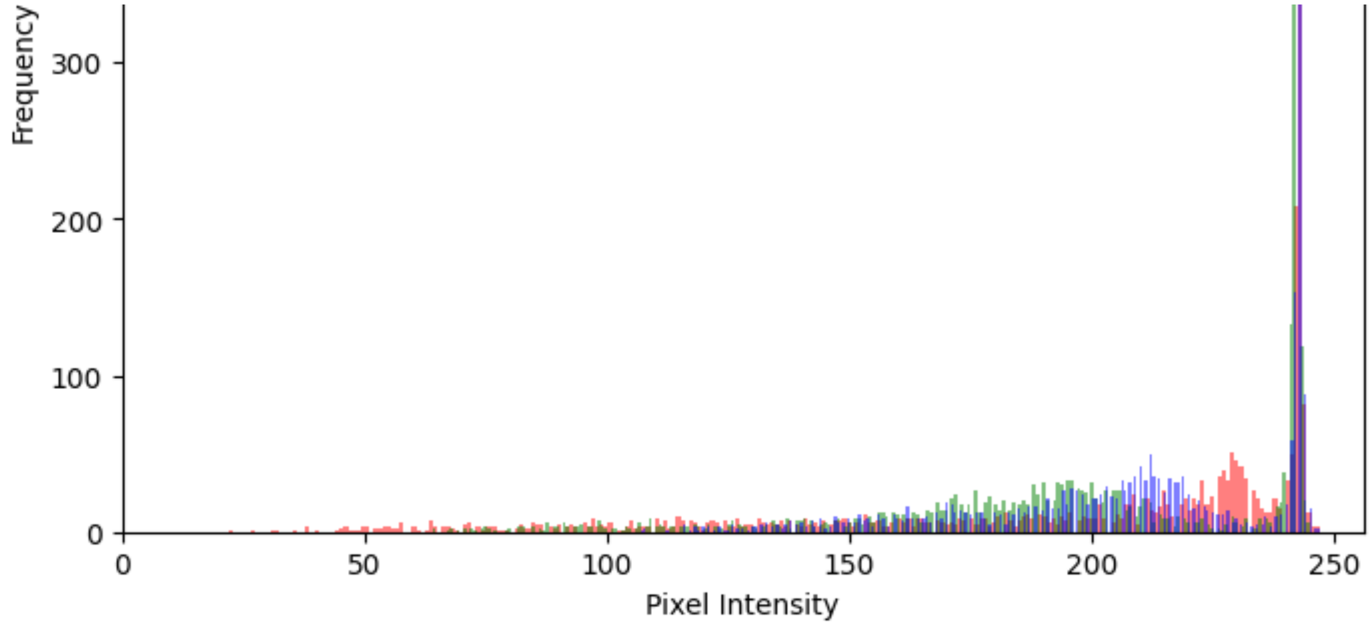




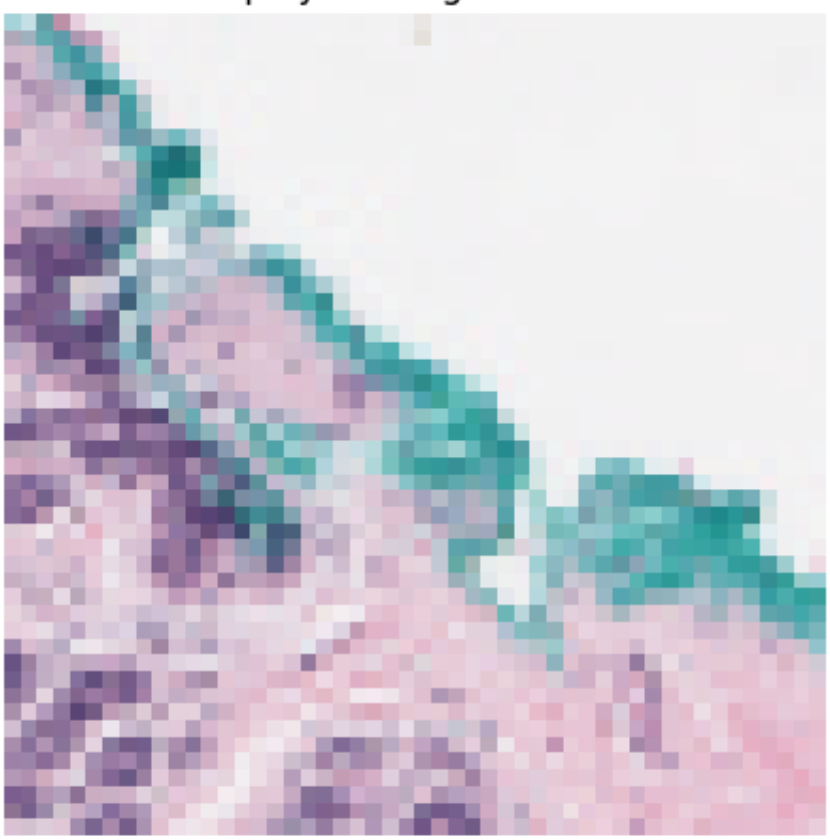



```
plot_combined_rgb_histograms(X, Y)
```





Displayed Image - No IDC



Data Splitting

```
from sklearn.model_selection import train_test_split
# Normalize the image data
X_normalized = X.astype('float32') / 255.0

# Split the data into training and testing sets
xtrain, xtest, ytrain, ytest = train_test_split(X_normalized, Y, test_size=0.3, r
```

```
numberoftrain = xtrain.shape[0] #Number of rows of data in numberoftrain
numberoftest = xtest.shape[0] #Number of rows of data in numberoftest
xtrain.shape #Dimensions of xtrain
```

```
(3882, 50, 50, 3)
```

```
#Reshape Xtrain & Xtest (only used for models other than Neural Networks)

xtrain = xtrain.reshape(numberoftrain,xtrain.shape[1]*xtrain.shape[2]*xtrain.shap
xtest = xtest.reshape(numberoftest,xtest.shape[1]*xtest.shape[2]*xtest.shape[3])
print("X Train: ",xtrain.shape)
print("X Test: ",xtest.shape)
```

```
X Train:  (3882, 7500)
X Test:  (1665, 7500)
```

```
len(ytest)
```

```
1665
```

```
len(ytrain)
```

```
3882
```

Basic Neural Network

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split


#The split done above will not be used for the Basic Neural Network
X = np.load('/content/drive/MyDrive/breast_cancer/X.npy') # images
Y = np.load('/content/drive/MyDrive/breast_cancer/Y.npy') # labels associated to

#Normalize the image data to be between 0 and 1
X_normalized = X.astype('float32') / 255.0

#Flatten the images to 1D since we're not using convolutional layers
X_flattened = X_normalized.reshape(X.shape[0], -1)
X_train, X_test, y_train, y_test = train_test_split(X_flattened, Y, test_size=0.3


model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),  # Input laye
    Dense(1, activation='sigmoid')  # Output layer with a single neuron and sigmo
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

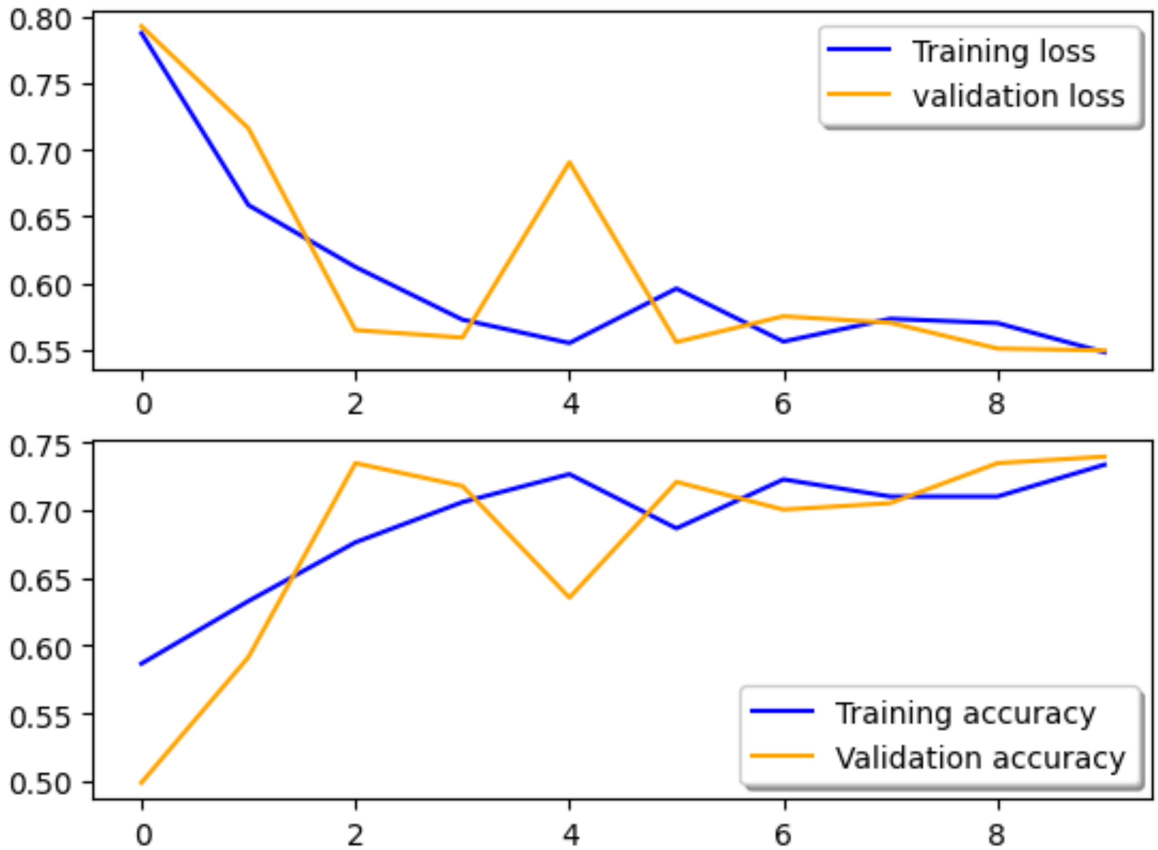
```
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test)
```

```
Epoch 1/10
122/122 [==============================] - 3s 23ms/step - loss: 0.7874 - accu
Epoch 2/10
122/122 [==============================] - 3s 24ms/step - loss: 0.6586 - accu
Epoch 3/10
122/122 [==============================] - 3s 28ms/step - loss: 0.6122 - accu
Epoch 4/10
122/122 [==============================] - 2s 19ms/step - loss: 0.5728 - accu
Epoch 5/10
122/122 [==============================] - 2s 16ms/step - loss: 0.5553 - accu
Epoch 6/10
122/122 [==============================] - 2s 16ms/step - loss: 0.5961 - accu
Epoch 7/10
122/122 [==============================] - 2s 18ms/step - loss: 0.5564 - accu
Epoch 8/10
122/122 [==============================] - 3s 24ms/step - loss: 0.5735 - accu
Epoch 9/10
122/122 [==============================] - 3s 25ms/step - loss: 0.5702 - accu
Epoch 10/10
122/122 [==============================] - 3s 24ms/step - loss: 0.5484 - accu
```

```
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test accuracy: {test_acc}, Test loss: {test_loss}")
```

```
53/53 [==============================] - 0s 9ms/step - loss: 0.5497 - accuracy
Test accuracy: 0.7393393516540527, Test loss: 0.5497421622276306
```

```
# Plot the loss and accuracy curves for training and validation
fig, ax = plt.subplots(2,1)
ax[0].plot(history.history['loss'], color='b', label="Training loss")
ax[0].plot(history.history['val_loss'], color='orange', label="validation loss",a
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['accuracy'], color='b', label="Training accuracy")
ax[1].plot(history.history['val_accuracy'], color='orange',label="Validation accu
legend = ax[1].legend(loc='best', shadow=True)
```



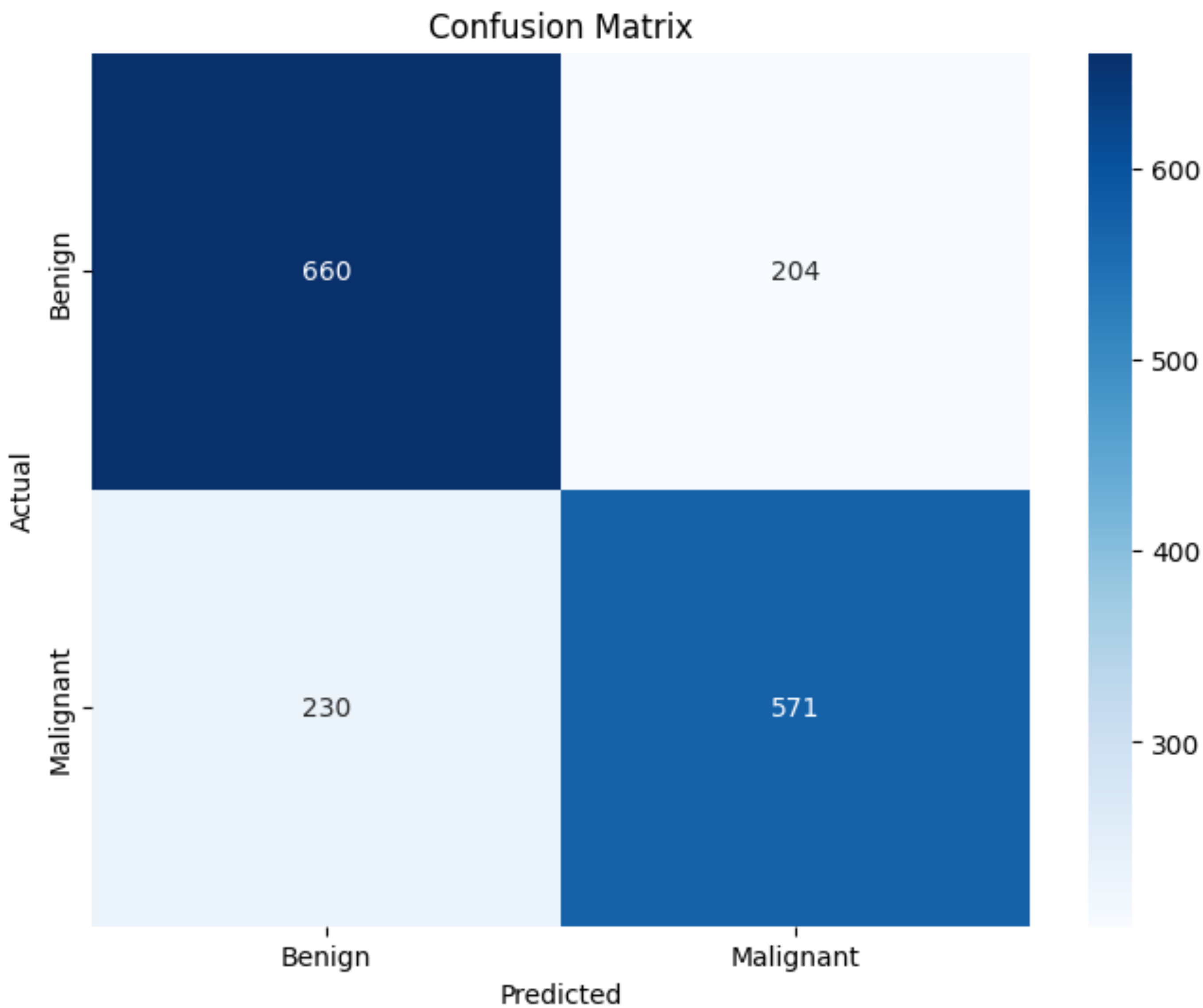

```
Y_pred = model.predict(X_test)
```

```
53/53 [==============================] - 0s 6ms/step
```

```
threshold = 0.5  # Threshold value to determine class membership
# Convert predicted probabilities into class labels
Y_pred_classes = np.where(Y_pred >= threshold, 1, 0)
print(Y_pred_classes)
```

```
[[0]
 [1]
 [0]
 ...
 [1]
 [0]
 [0]]
```

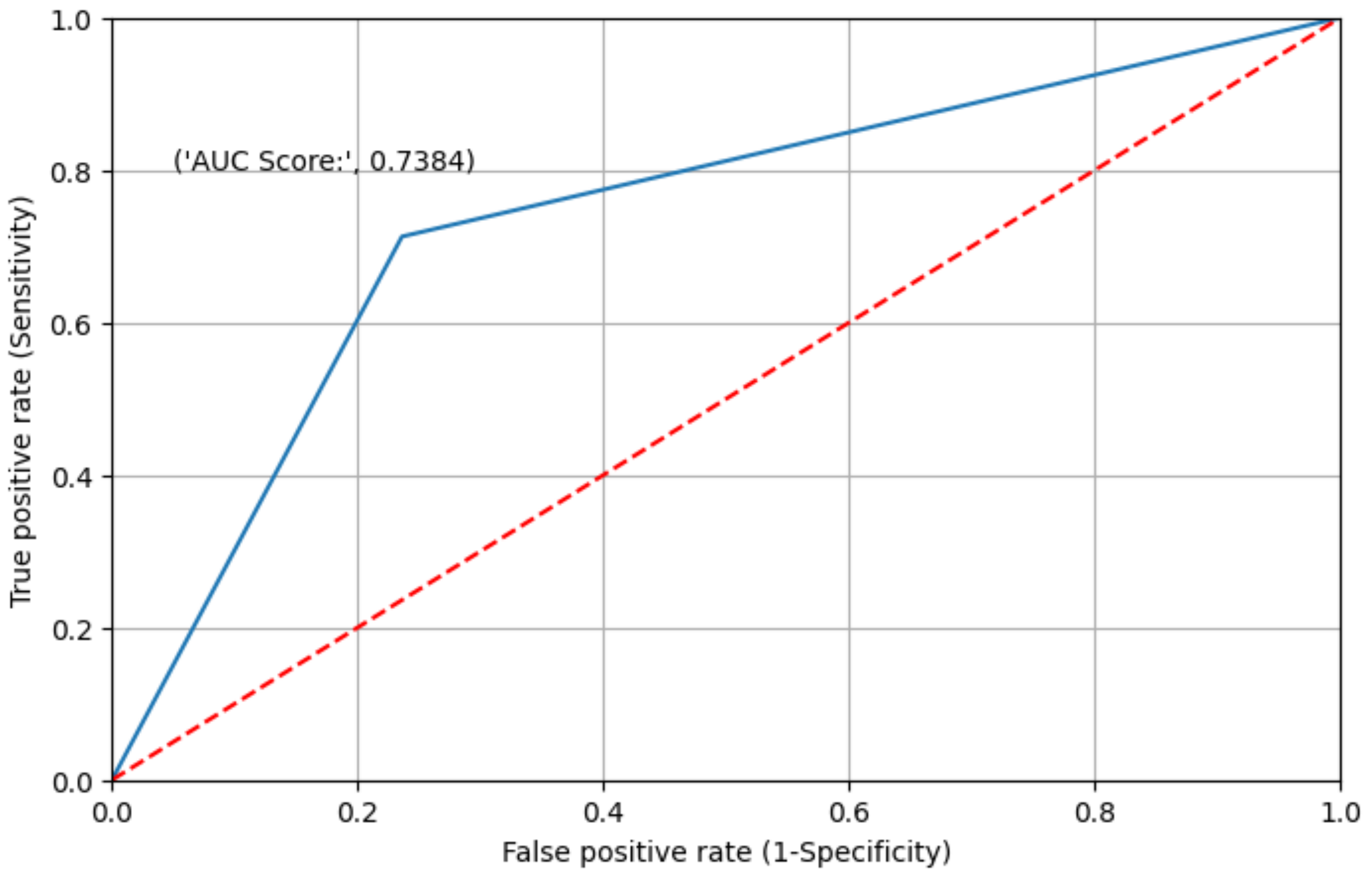
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
import numpy as np

conf_matrix = confusion_matrix(y_test, Y_pred_classes)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap='Blues', xticklabels=['Benign'
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
```



```
from sklearn.metrics import classification_report
# accuracy measures by classification_report()
result = classification_report(y_test,Y_pred_classes)

# print the result
print(result)
```

```
              precision    recall  f1-score   support

           0       0.74      0.76      0.75       864
           1       0.74      0.71      0.72       801

    accuracy                           0.74      1665
   macro avg       0.74      0.74      0.74      1665
weighted avg       0.74      0.74      0.74      1665
```

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

plt.rcParams['figure.figsize']=(8,5)

fpr, tpr, thresholds = roc_curve(ytest, Y_pred_classes)
plt.plot(fpr,tpr)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.plot([0, 1], [0, 1],'r--') # r-- : Red dashed line
plt.text(x = 0.05, y = 0.8, s =('AUC Score:', round(roc_auc_score(ytest, Y_pred_c
plt.xlabel('False positive rate (1-Specificity)')
plt.ylabel('True positive rate (Sensitivity)')

# plot the grid
plt.grid(True)
```



```
import pandas as pd
from sklearn import metrics

# Define columns
cols = ['Model', 'AUC Score', 'Precision Score', 'Recall Score','Accuracy Score',
result_tabulation = pd.DataFrame(columns=cols)
Neural_Network = pd.DataFrame({'Model': ["Basic Neural Network"],
                     'AUC Score' : [metrics.roc_auc_score(ytest, Y_pred_classes)]
                 'Precision Score': [metrics.precision_score(ytest, Y_pred_classe
                 'Recall Score': [metrics.recall_score(ytest, Y_pred_classes)],
                 'Accuracy Score': [metrics.accuracy_score(ytest, Y_pred_classes)
                  'f1-score': [metrics.f1_score(ytest, Y_pred_classes)]})

# appending our result table
result_tabulation = pd.concat([result_tabulation, Neural_Network], ignore_index=T
result_tabulation
```

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	f1-score
0	Basic Neural	0.738374	0.736774	0.712859	0.739339	0.724619

Convolutional Neural Network

```
X = np.load('/content/drive/MyDrive/breast_cancer/X.npy') # images
Y = np.load('/content/drive/MyDrive/breast_cancer/Y.npy') # labels associated to
```

```
from sklearn.model_selection import train_test_split
# Normalize the image data
X_normalized = X.astype('float32') / 255.0

# Split the data into training and testing sets
xtrain, xtest, ytrain, ytest = train_test_split(X_normalized, Y, test_size=0.3, r
```

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(50, 50, 3)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
#With Early Stopping
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weigh
history = model.fit(xtrain, ytrain, epochs=100, validation_data=(xtest, ytest), c
```

```
Epoch 1/100
122/122 [==============================] - 100s 817ms/step - loss: 0.5826 - a
Epoch 2/100
122/122 [==============================] - 69s 563ms/step - loss: 0.5648 - acc
Epoch 3/100
122/122 [==============================] - 109s 895ms/step - loss: 0.5686 - a
Epoch 4/100
122/122 [==============================] - 99s 811ms/step - loss: 0.5323 - acc
Epoch 5/100
122/122 [==============================] - 83s 682ms/step - loss: 0.5241 - acc
Epoch 6/100
122/122 [==============================] - 65s 535ms/step - loss: 0.5119 - acc
Epoch 7/100
122/122 [==============================] - 61s 502ms/step - loss: 0.5305 - acc
Epoch 8/100
122/122 [==============================] - 62s 505ms/step - loss: 0.5732 - acc
Epoch 9/100
122/122 [==============================] - 60s 495ms/step - loss: 0.5161 - acc
```

```
test_loss, test_acc = model.evaluate(xtest, ytest)
print(f"Test accuracy: {test_acc}, Test loss: {test_loss}")
```

```
53/53 [==============================] - 6s 105ms/step - loss: 0.5057 - accura
Test accuracy: 0.7621621489524841, Test loss: 0.5057265162467957
```

```
# Plot the loss and accuracy curves for training and validation
fig, ax = plt.subplots(2,1)
ax[0].plot(history.history['loss'], color='b', label="Training loss")
ax[0].plot(history.history['val_loss'], color='orange', label="validation loss",a
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['accuracy'], color='b', label="Training accuracy")
ax[1].plot(history.history['val_accuracy'], color='orange',label="Validation accu
legend = ax[1].legend(loc='best', shadow=True)
```

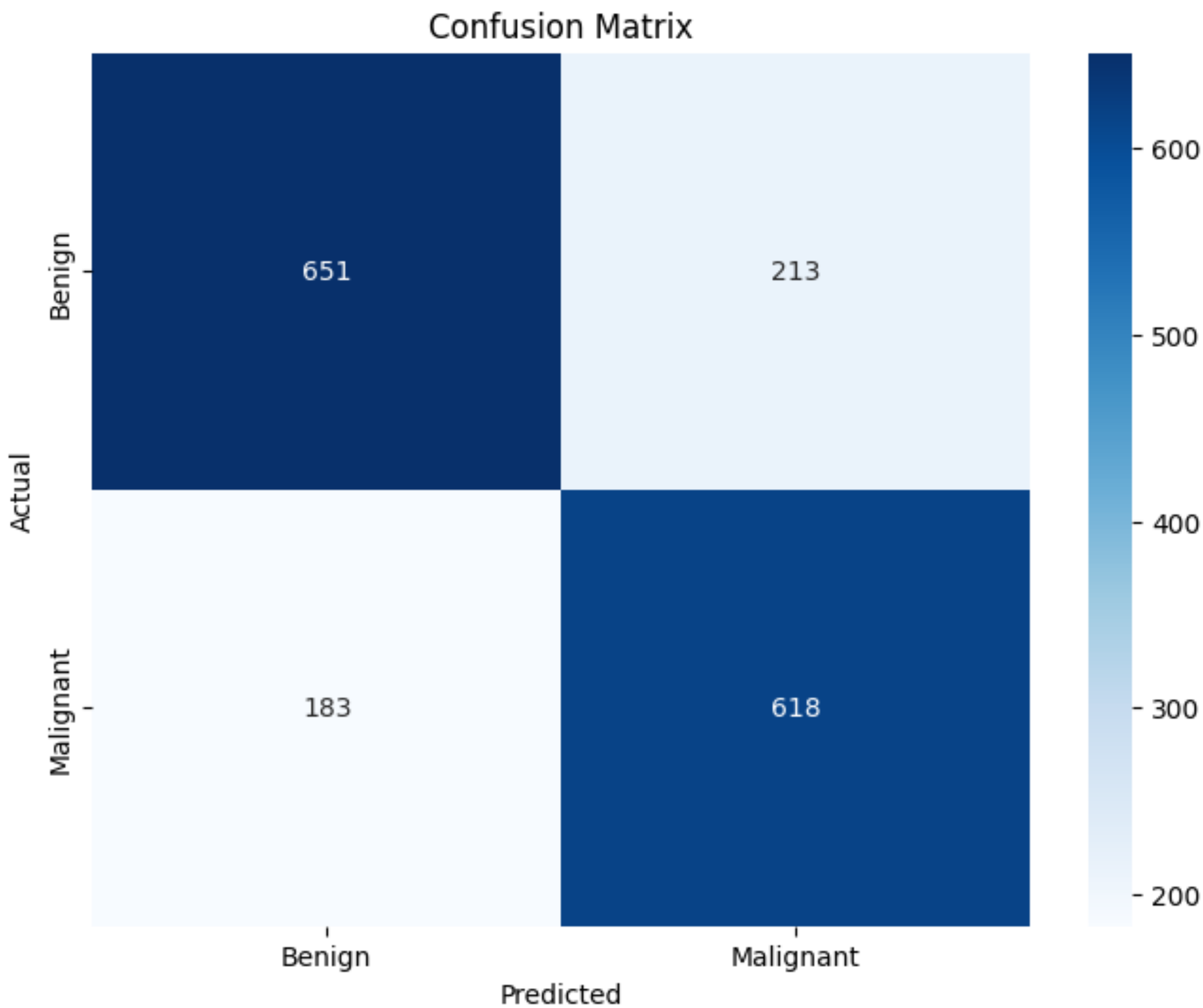

```
Y_pred = model.predict(xtest)
```

```
53/53 [==============================] - 5s 101ms/step
```

```
threshold = 0.5  # Threshold value to determine class membership
# Convert predicted probabilities into class labels
Y_pred_classes = np.where(Y_pred >= threshold, 1, 0)
print(Y_pred_classes)
```

```
[[0]
 [1]
 [0]
 ...
 [0]
 [0]
 [0]]
```

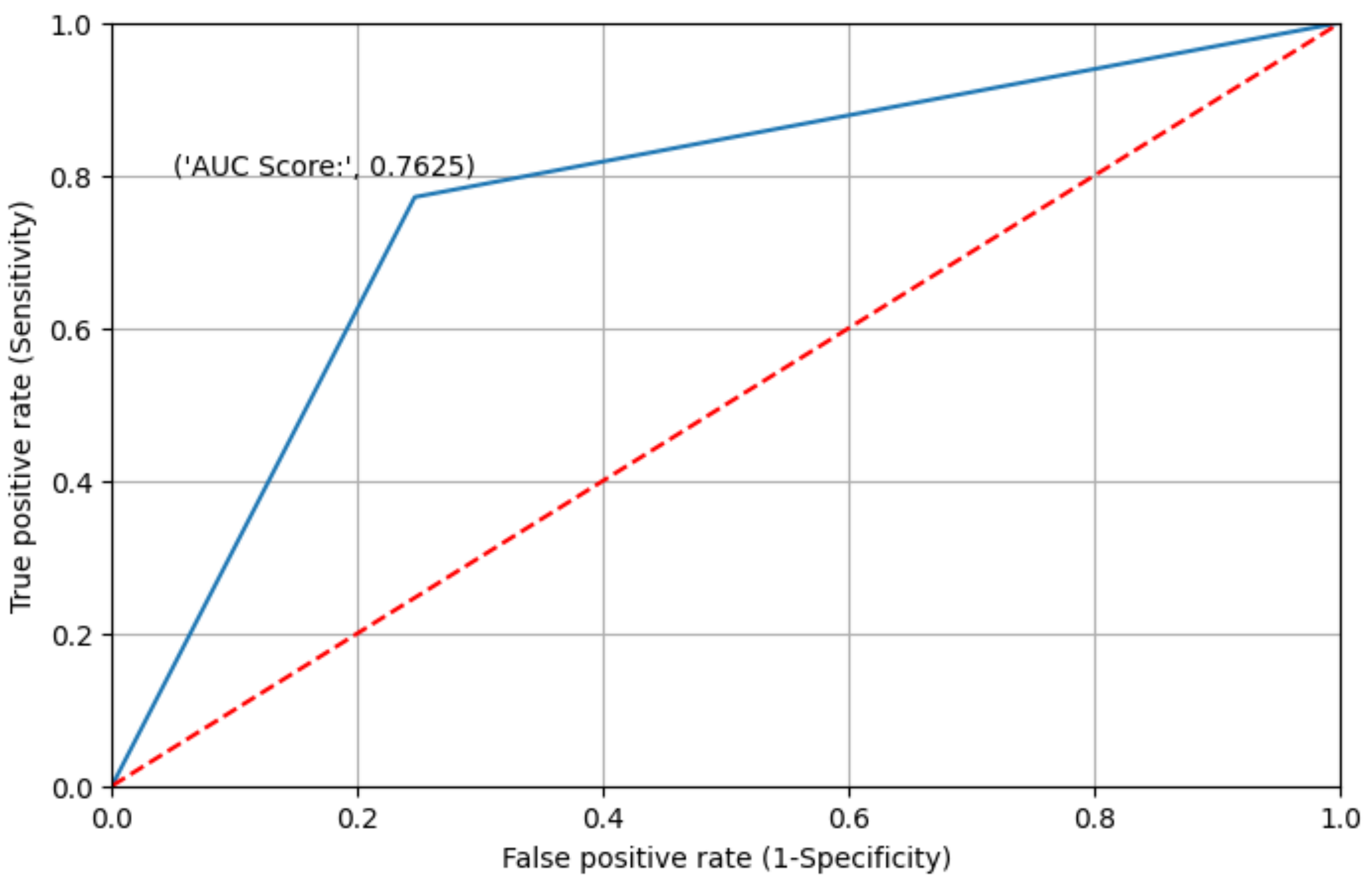
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
import numpy as np

conf_matrix = confusion_matrix(y_test, Y_pred_classes)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap='Blues', xticklabels=['Benign'
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
```



```
from sklearn.metrics import classification_report
# accuracy measures by classification_report()
result = classification_report(y_test,Y_pred_classes)

# print the result
print(result)
```

```
              precision    recall  f1-score   support

           0       0.78      0.75      0.77       864
           1       0.74      0.77      0.76       801

    accuracy                           0.76      1665
   macro avg       0.76      0.76      0.76      1665
weighted avg       0.76      0.76      0.76      1665
```

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

plt.rcParams['figure.figsize']=(8,5)

fpr, tpr, thresholds = roc_curve(ytest, Y_pred_classes)
plt.plot(fpr,tpr)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.plot([0, 1], [0, 1],'r--') # r-- : Red dashed line
plt.text(x = 0.05, y = 0.8, s =('AUC Score:', round(roc_auc_score(ytest, Y_pred_c
plt.xlabel('False positive rate (1-Specificity)')
plt.ylabel('True positive rate (Sensitivity)')

# plot the grid
plt.grid(True)
```



```
CNN = pd.DataFrame({'Model': ["Convolutional Neural Network"],
                     'AUC Score' : [metrics.roc_auc_score(ytest, Y_pred_classes)]
                 'Precision Score': [metrics.precision_score(ytest, Y_pred_classe
                 'Recall Score': [metrics.recall_score(ytest, Y_pred_classes)],
                 'Accuracy Score': [metrics.accuracy_score(ytest, Y_pred_classes)
                  'f1-score': [metrics.f1_score(ytest, Y_pred_classes)]})

# appending our result table
result_tabulation = pd.concat([result_tabulation, CNN], ignore_index=True)
result_tabulation
```

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	f1-score
0	Basic Neural Network	0.738374	0.736774	0.712859	0.739339	0.724619
1	Convolutional Neural	0.762504	0.743682	0.771536	0.762162	0.757353

Next steps: View recommended plots

Transfer Learning

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split


base_model = VGG16(weights='imagenet', include_top=False, input_shape=(50, 50, 3)
base_model.trainable = False  # Freeze the convolutional base to prevent its weig


x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(1, activation='sigmoid')(x)  # Sigmoid for binary classificat
model = Model(inputs=base_model.input, outputs=predictions)
```

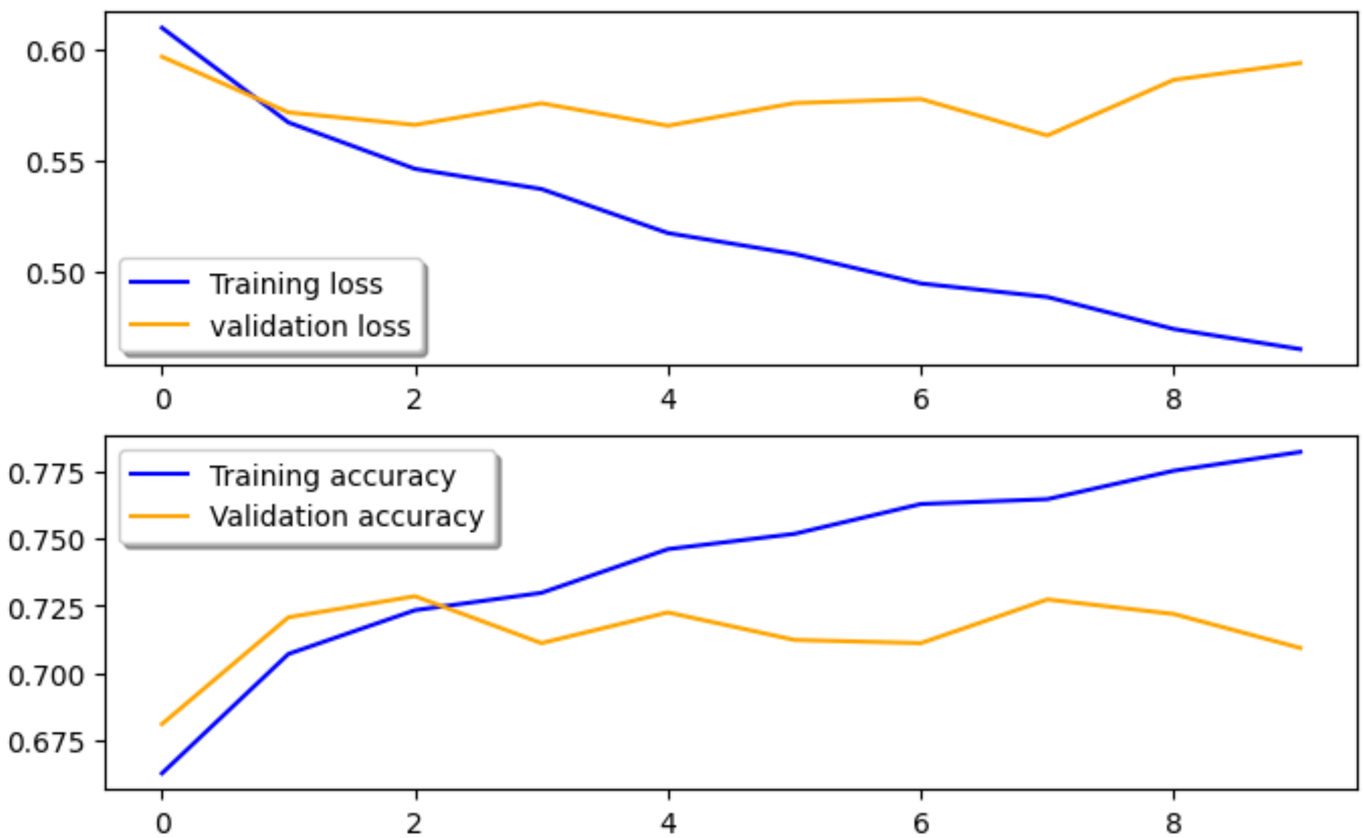
```
model.compile(optimizer=Adam(lr=0.0001), loss='binary_crossentropy', metrics=['ac
history = model.fit(xtrain, ytrain, epochs=10, validation_data=(xtest, ytest))
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate`
Epoch 1/10
122/122 [==============================] - 213s 2s/step - loss: 0.6100 - accu
Epoch 2/10
122/122 [==============================] - 198s 2s/step - loss: 0.5673 - accu
Epoch 3/10
122/122 [==============================] - 196s 2s/step - loss: 0.5464 - accu
Epoch 4/10
122/122 [==============================] - 163s 1s/step - loss: 0.5373 - accu
Epoch 5/10
122/122 [==============================] - 197s 2s/step - loss: 0.5174 - accu
Epoch 6/10
122/122 [==============================] - 197s 2s/step - loss: 0.5080 - accu
Epoch 7/10
122/122 [==============================] - 198s 2s/step - loss: 0.4947 - accu
Epoch 8/10
122/122 [==============================] - 198s 2s/step - loss: 0.4886 - accu
Epoch 9/10
122/122 [==============================] - 198s 2s/step - loss: 0.4741 - accu
Epoch 10/10
122/122 [==============================] - 199s 2s/step - loss: 0.4650 - accu
```

```
test_loss, test_acc = model.evaluate(xtest, ytest)
print(f"Test accuracy: {test_acc}, Test loss: {test_loss}")
```

```
53/53 [==============================] - 50s 939ms/step - loss: 0.5941 - accu
Test accuracy: 0.7093092799186707, Test loss: 0.5941338539123535
```

```
# Plot the loss and accuracy curves for training and validation
fig, ax = plt.subplots(2,1)
ax[0].plot(history.history['loss'], color='b', label="Training loss")
ax[0].plot(history.history['val_loss'], color='orange', label="validation loss",a
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['accuracy'], color='b', label="Training accuracy")
ax[1].plot(history.history['val_accuracy'], color='orange',label="Validation accu
legend = ax[1].legend(loc='best', shadow=True)
```



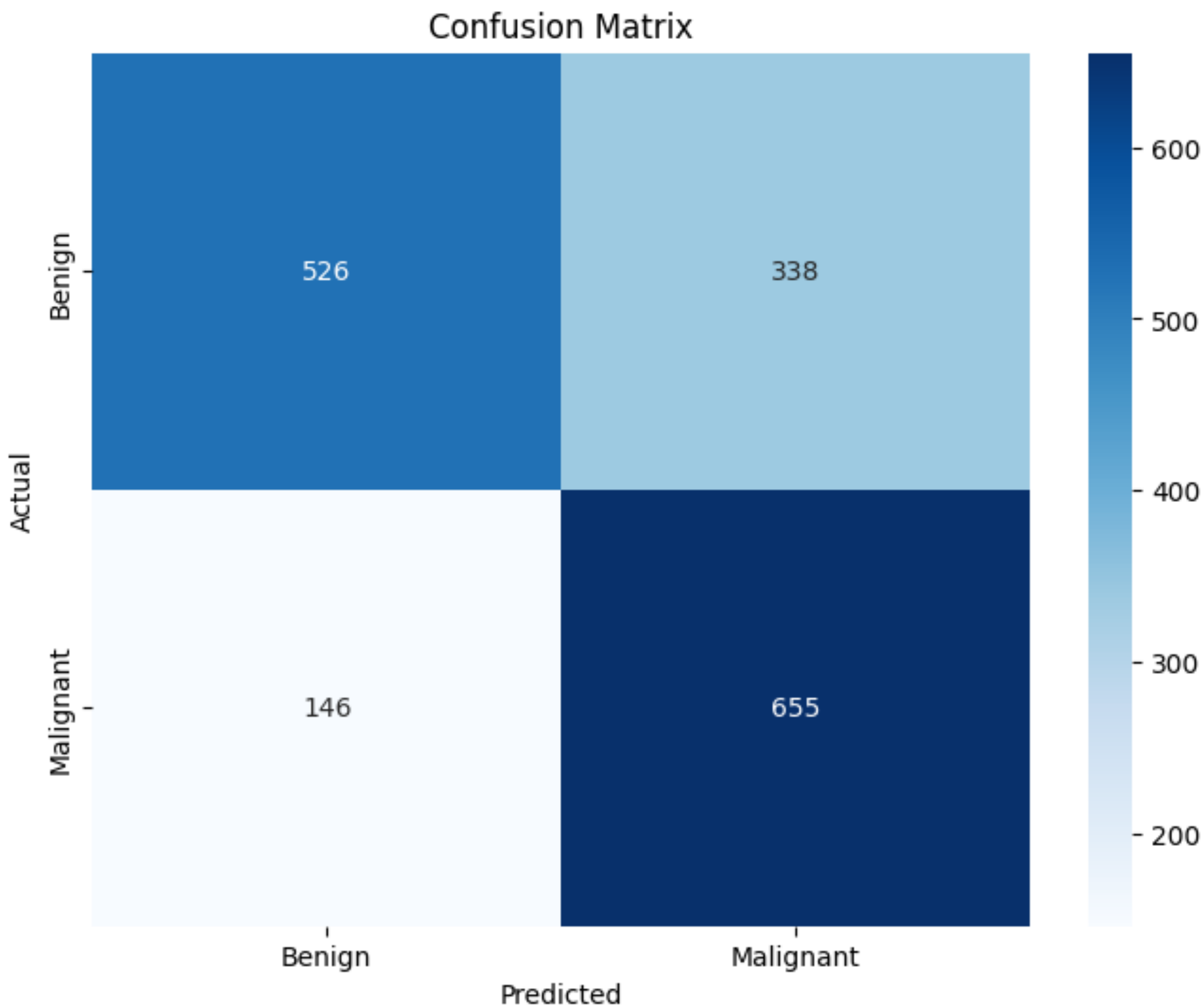

```
Y_pred = model.predict(xtest)
```

```
53/53 [==============================] - 51s 944ms/step
```

```
threshold = 0.5  # Threshold value to determine class membership
# Convert predicted probabilities into class labels
Y_pred_classes = np.where(Y_pred >= threshold, 1, 0)
print(Y_pred_classes)
```

```
[[1]
 [1]
 [1]
 ...
 [0]
 [0]
 [0]]
```

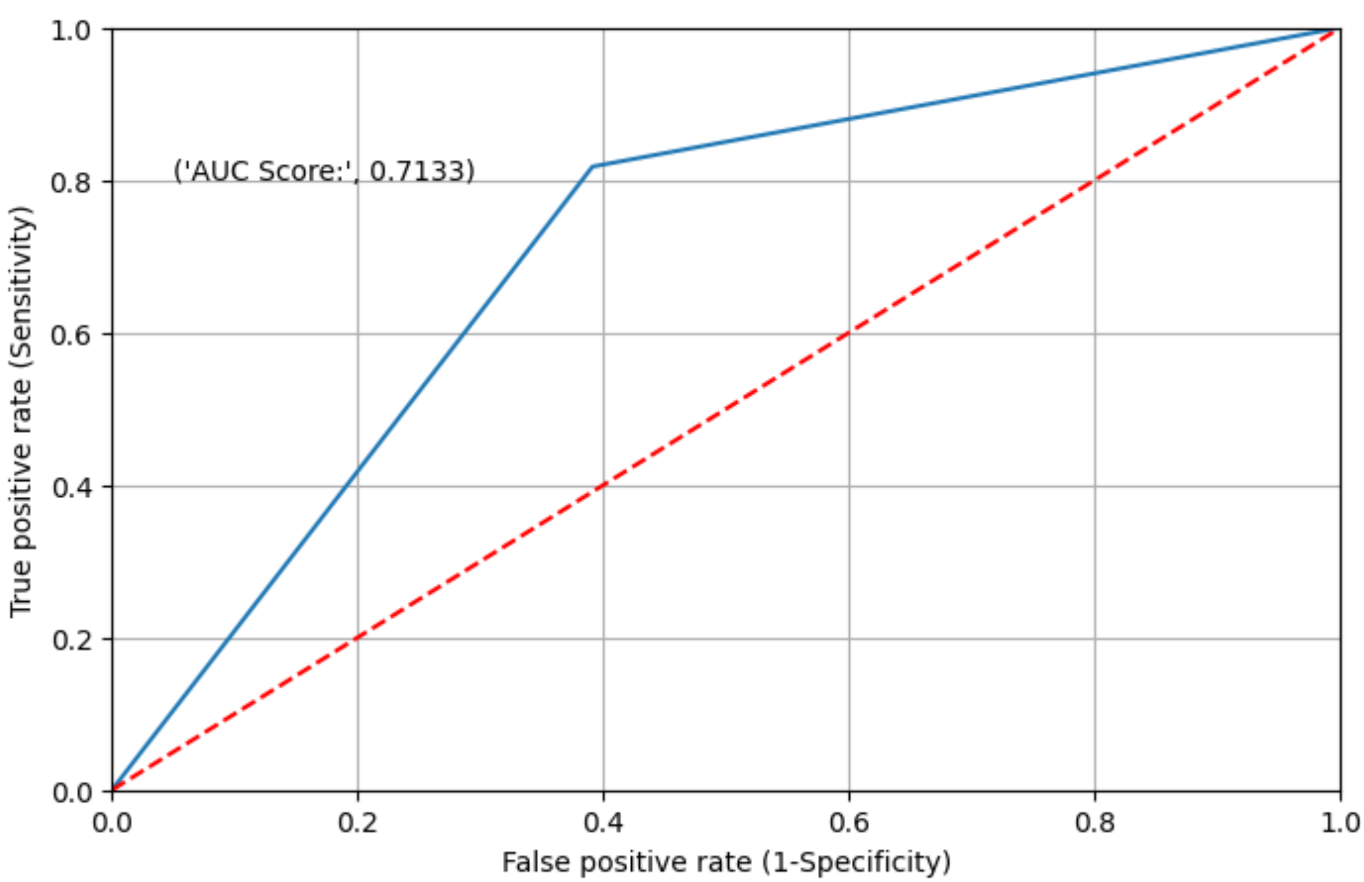
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
import numpy as np

conf_matrix = confusion_matrix(y_test, Y_pred_classes)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap='Blues', xticklabels=['Benign'
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
```



```
from sklearn.metrics import classification_report
# accuracy measures by classification_report()
result = classification_report(y_test,Y_pred_classes)
print(result)
```

```
              precision    recall  f1-score   support

           0       0.78      0.61      0.68       864
           1       0.66      0.82      0.73       801

    accuracy                           0.71      1665
   macro avg       0.72      0.71      0.71      1665
weighted avg       0.72      0.71      0.71      1665
```

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

plt.rcParams['figure.figsize']=(8,5)

fpr, tpr, thresholds = roc_curve(ytest, Y_pred_classes)
plt.plot(fpr,tpr)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.plot([0, 1], [0, 1],'r--') # r-- : Red dashed line
plt.text(x = 0.05, y = 0.8, s =('AUC Score:', round(roc_auc_score(ytest, Y_pred_c
plt.xlabel('False positive rate (1-Specificity)')
plt.ylabel('True positive rate (Sensitivity)')

# plot the grid
plt.grid(True)
```



```
VGG = pd.DataFrame({'Model': ["VGG-16: Transfer Learning"],
                     'AUC Score' : [metrics.roc_auc_score(ytest, Y_pred_classes)]
                 'Precision Score': [metrics.precision_score(ytest, Y_pred_classe
                 'Recall Score': [metrics.recall_score(ytest, Y_pred_classes)],
                 'Accuracy Score': [metrics.accuracy_score(ytest, Y_pred_classes)
                  'f1-score': [metrics.f1_score(ytest, Y_pred_classes)]})

# appending our result table
result_tabulation = pd.concat([result_tabulation, VGG], ignore_index=True)
result_tabulation
```

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	f1-score
0	Basic Neural Network	0.738374	0.736774	0.712859	0.739339	0.724619
1	Convolutional Neural Network	0.762504	0.743682	0.771536	0.762162	0.757353

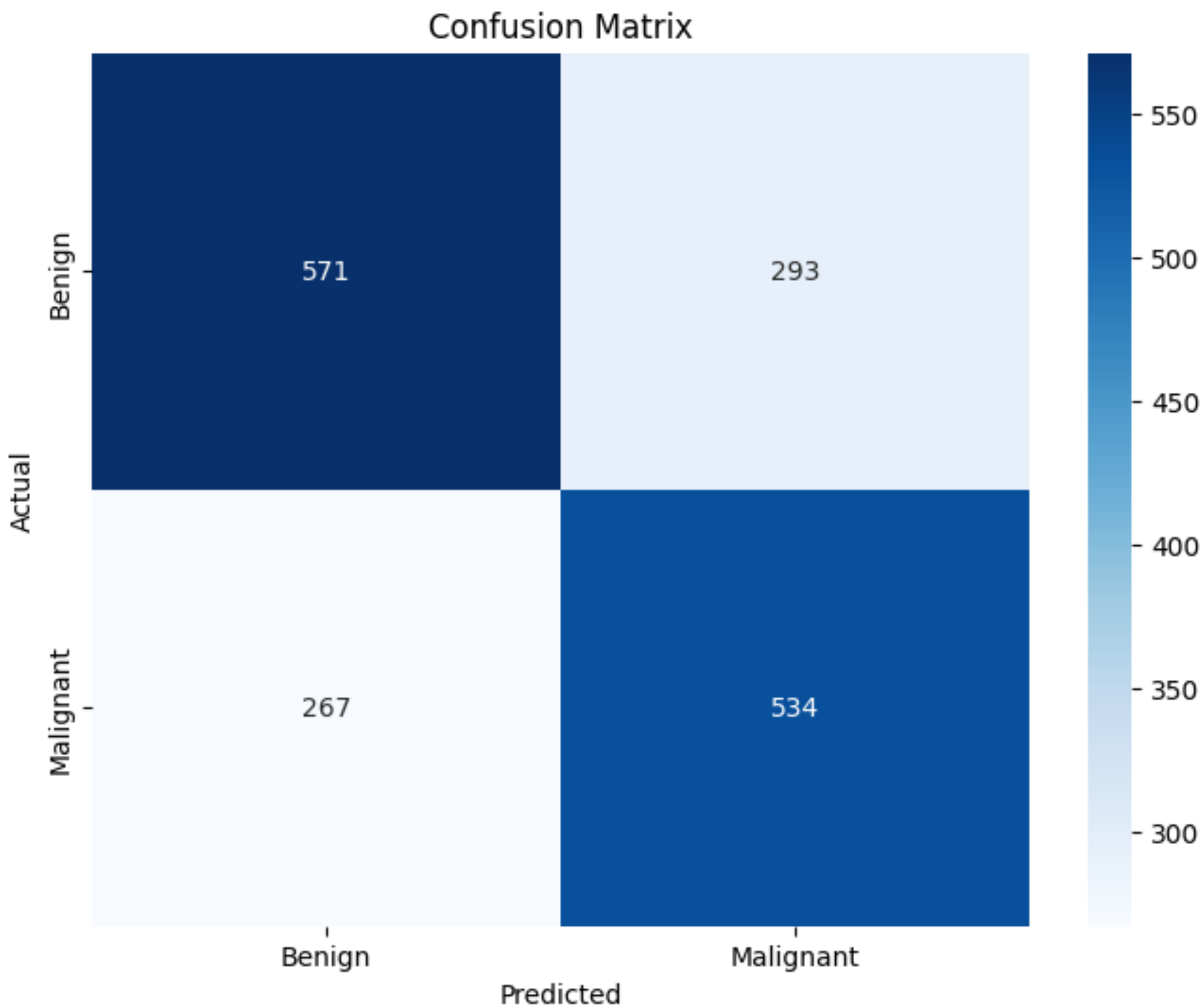
Next steps: View recommended plots

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
DTC = DecisionTreeClassifier() #initialization
decision_tree= DTC.fit(xtrain,ytrain) #fits on training data


# predicting values
y_pred  = decision_tree.predict(xtest)


y_pred
```

```
array([0, 1, 1, ..., 0, 0, 0])
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
import numpy as np

conf_matrix = confusion_matrix(ytest, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap='Blues', xticklabels=['Benign'
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
```



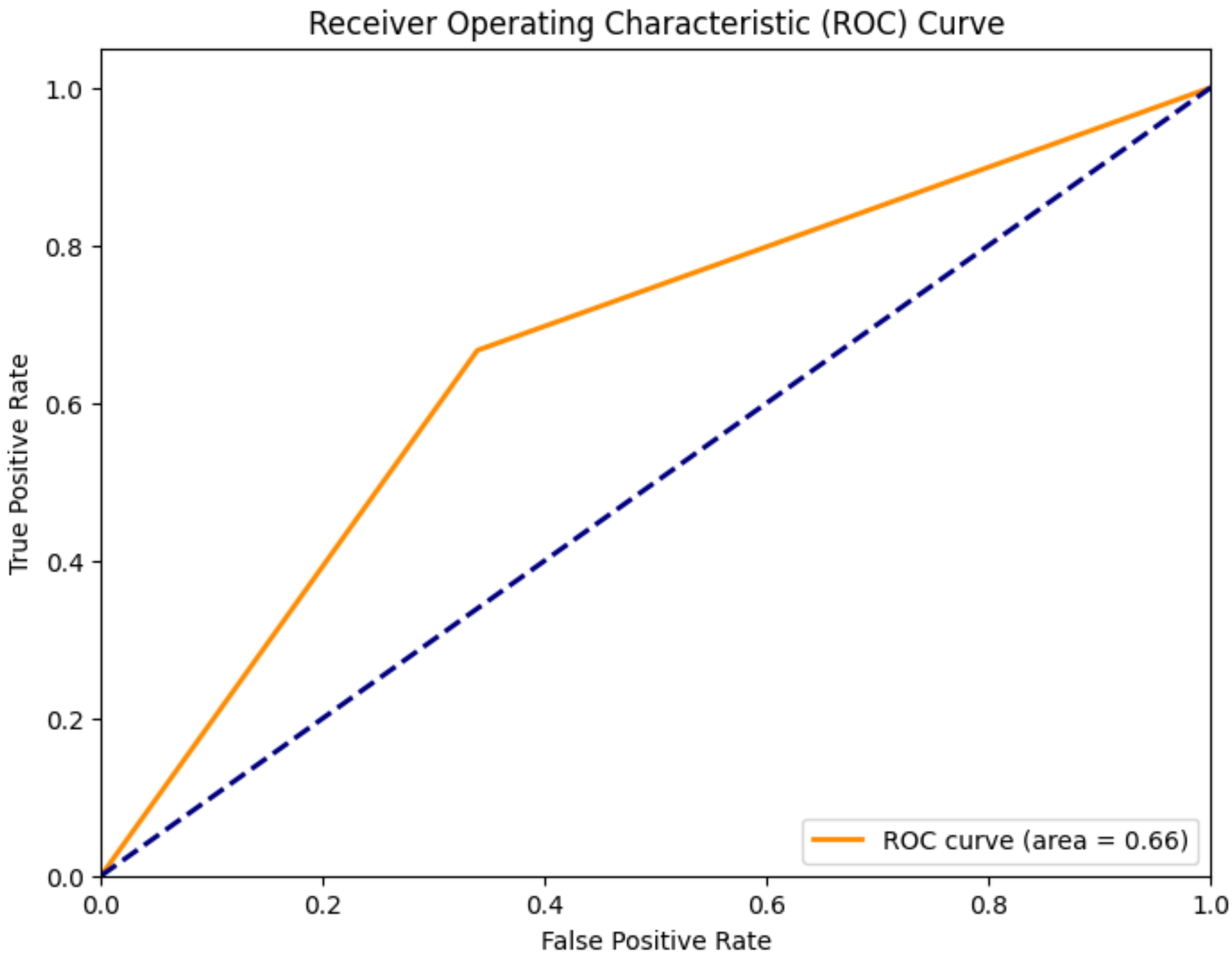
```
from sklearn.metrics import classification_report
# accuracy measures by classification_report()
result = classification_report(y_test,Y_pred_classes)

# print the result
print(result)
```

```
              precision    recall  f1-score   support

           0       0.78      0.61      0.68       864
           1       0.66      0.82      0.73       801

    accuracy                           0.71      1665
   macro avg       0.72      0.71      0.71      1665
weighted avg       0.72      0.71      0.71      1665
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
accuracy = accuracy_score(ytest, y_pred)
precision = precision_score(ytest, y_pred)
recall = recall_score(ytest, y_pred)  # Sensitivity
f1 = f1_score(ytest, y_pred)
roc_auc = roc_auc_score(ytest, y_pred)
# Confusion Matrix and Specificity
cm = confusion_matrix(ytest, y_pred)
tn, fp, fn, tp = cm.ravel()
specificity = tn / (tn + fp)
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall (Sensitivity): {recall}")
print(f"F1 Score: {f1}")
print(f"ROC AUC Score: {roc_auc}")
print(f"Specificity: {specificity}")
```

```
Accuracy: 0.6636636636636637
Precision: 0.6457073760580411
Recall (Sensitivity): 0.6666666666666666
F1 Score: 0.656019656019656
ROC AUC Score: 0.6637731481481481
Specificity: 0.6608796296296297
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(ytest, y_pred)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % r
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.66)

```
Decision_Tree = pd.DataFrame({'Model': ["Decision Tree"],
                     'AUC Score' : [metrics.roc_auc_score(ytest, y_pred)],
                 'Precision Score': [metrics.precision_score(ytest, y_pred)],
                 'Recall Score': [metrics.recall_score(ytest, y_pred)],
                 'Accuracy Score': [metrics.accuracy_score(ytest, y_pred)],
                  'f1-score': [metrics.f1_score(ytest, y_pred)]})

# appending our result table
result_tabulation = pd.concat([result_tabulation, Decision_Tree], ignore_index=Tr
```

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	f1-score
0	Basic Neural Network	0.738374	0.736774	0.712859	0.739339	0.724619
1	Convolutional Neural Network	0.762504	0.743682	0.771536	0.762162	0.757353
2	VGG-16: Transfer Learning	0.713262	0.659617	0.817728	0.709309	0.730212
3	Decision Tree	0.713262	0.659617	0.817728	0.709309	0.730212

Next steps: View recommended plots

```
result_tabulation = result_tabulation.drop(4)
result_tabulation
```

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	f1-score
0	Basic Neural Network	0.738374	0.736774	0.712859	0.739339	0.724619
1	Convolutional Neural Network	0.762504	0.743682	0.771536	0.762162	0.757353
2	VGG-16: Transfer Learning	0.713262	0.659617	0.817728	0.709309	0.730212

Next steps: View recommended plots

Gradient Boosting

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-package
```

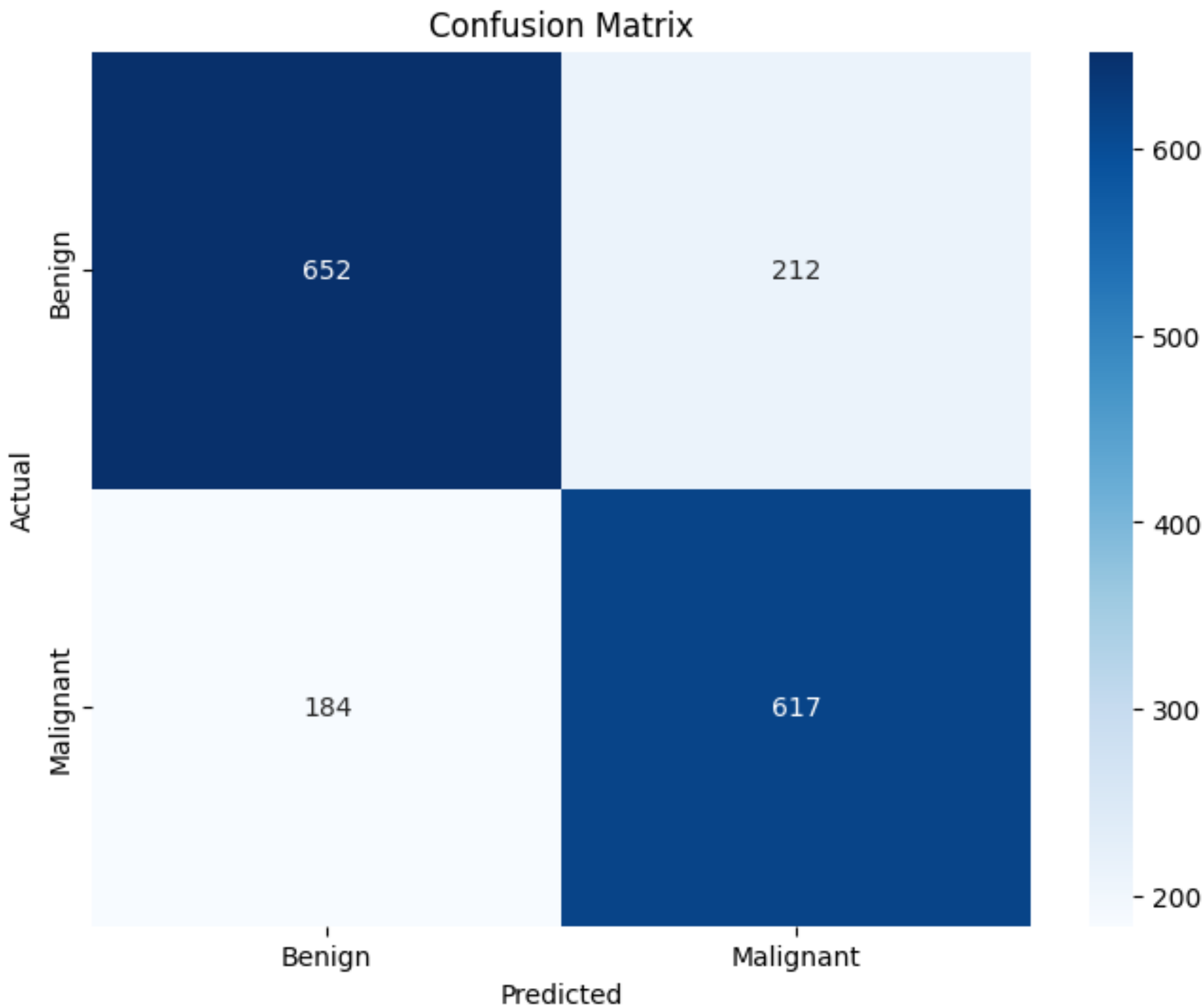
```
from xgboost import XGBClassifier
xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')  #Initializa
xgb_model = xgb.fit(xtrain, ytrain) #Fitting
```

```
y_pred = xgb_model.predict(xtest)
```

```
y_pred
```

```
array([0, 1, 0, ..., 0, 0, 0])
```

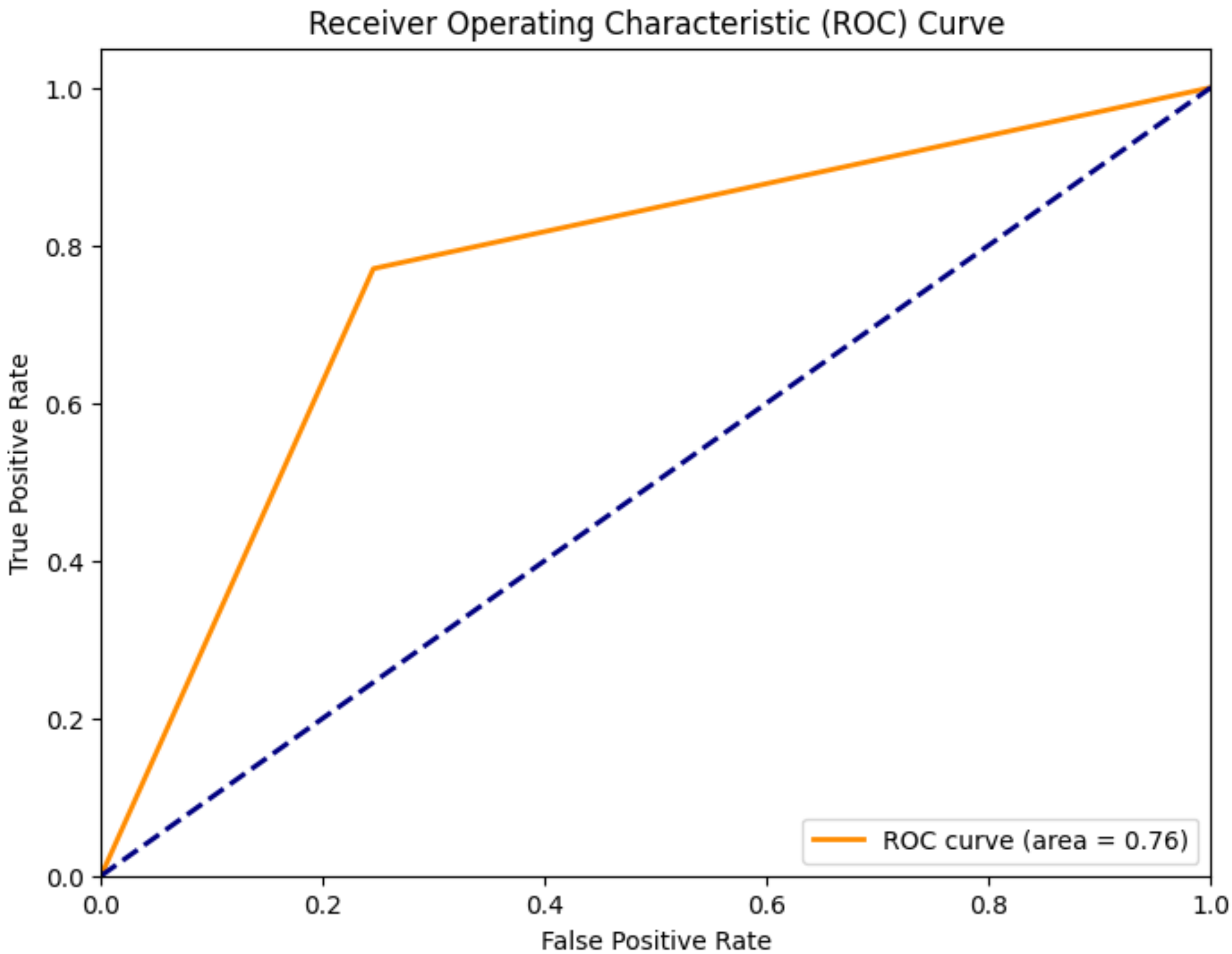
```
conf_matrix = confusion_matrix(ytest, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap='Blues', xticklabels=['Benign'
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
```



```
from sklearn.metrics import classification_report
# accuracy measures by classification_report()
result = classification_report(y_test,Y_pred_classes)

# print the result
print(result)
```

```
              precision    recall  f1-score   support

           0       0.78      0.61      0.68       864
           1       0.66      0.82      0.73       801

    accuracy                           0.71      1665
   macro avg       0.72      0.71      0.71      1665
weighted avg       0.72      0.71      0.71      1665
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(ytest, y_pred)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % r
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```




```
Gradientboosting = pd.DataFrame({'Model': ["Gradient Boosting"],
                     'AUC Score' : [metrics.roc_auc_score(ytest, y_pred)],
                 'Precision Score': [metrics.precision_score(ytest, y_pred)],
                 'Recall Score': [metrics.recall_score(ytest, y_pred)],
                 'Accuracy Score': [metrics.accuracy_score(ytest, y_pred)],
                  'f1-score': [metrics.f1_score(ytest, y_pred)]})

# appending our result table
result_tabulation = pd.concat([result_tabulation, Gradientboosting], ignore_index
```

```
result_tabulation
```

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	f1-score
0	Basic Neural Network	0.738374	0.736774	0.712859	0.739339	0.724619
1	Convolutional Neural Network	0.762504	0.743682	0.771536	0.762162	0.757353
2	VGG-16: Transfer Learning	0.713262	0.659617	0.817728	0.709309	0.730212
3	Decision Tree	0.713262	0.659617	0.817728	0.709309	0.730212

Next steps: View recommended plots

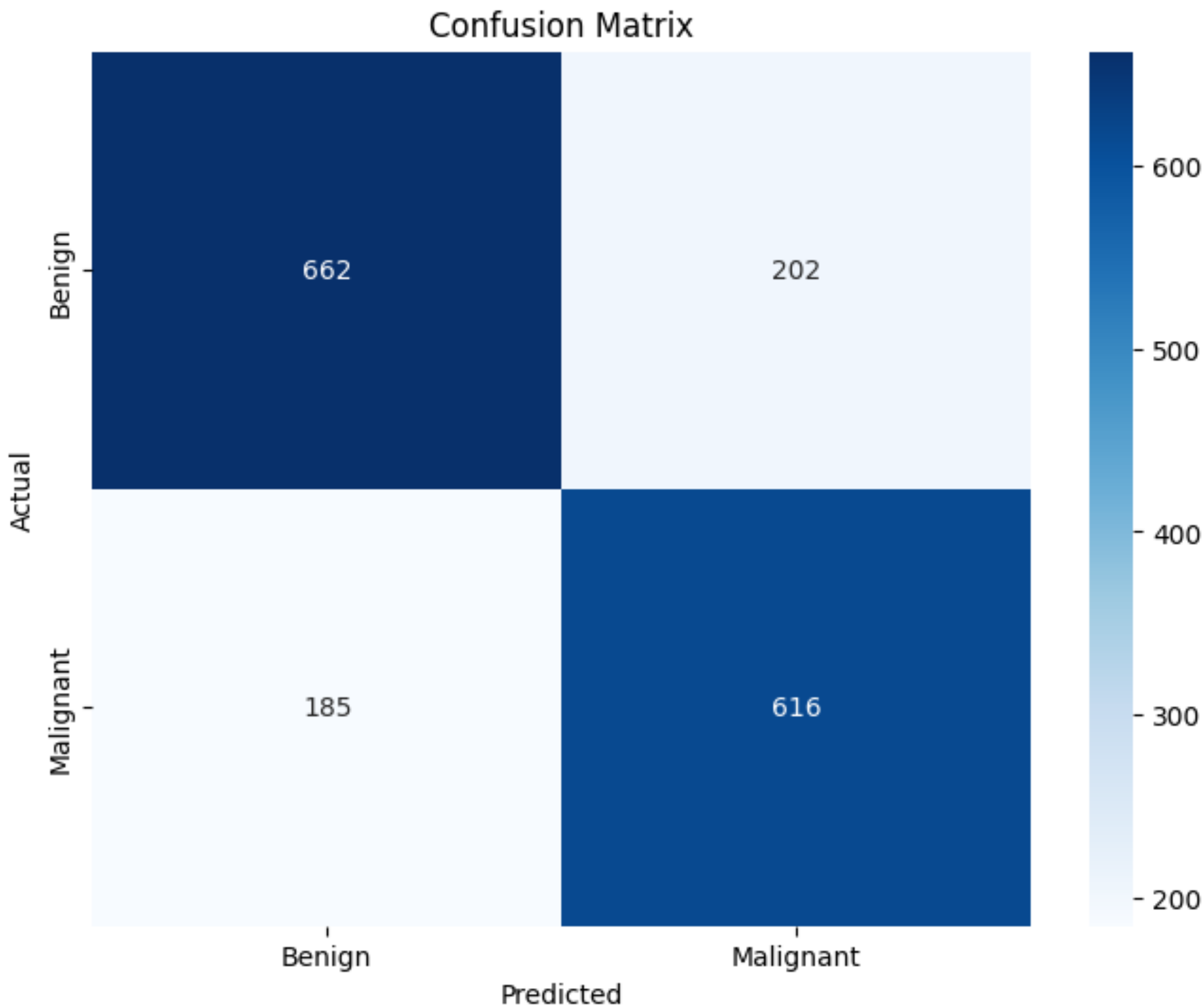
Support Vector Machine

```
from sklearn.svm import SVC
SVM = SVC(kernel= 'rbf', random_state=42) #SVC is the support vector machine clas
svm = SVM.fit(xtrain,ytrain)  #Fit on training data


# predicting values
y_pred  = svm.predict(xtest)
```

```
y_pred
```

```
array([0, 1, 0, ..., 0, 0, 0])
```

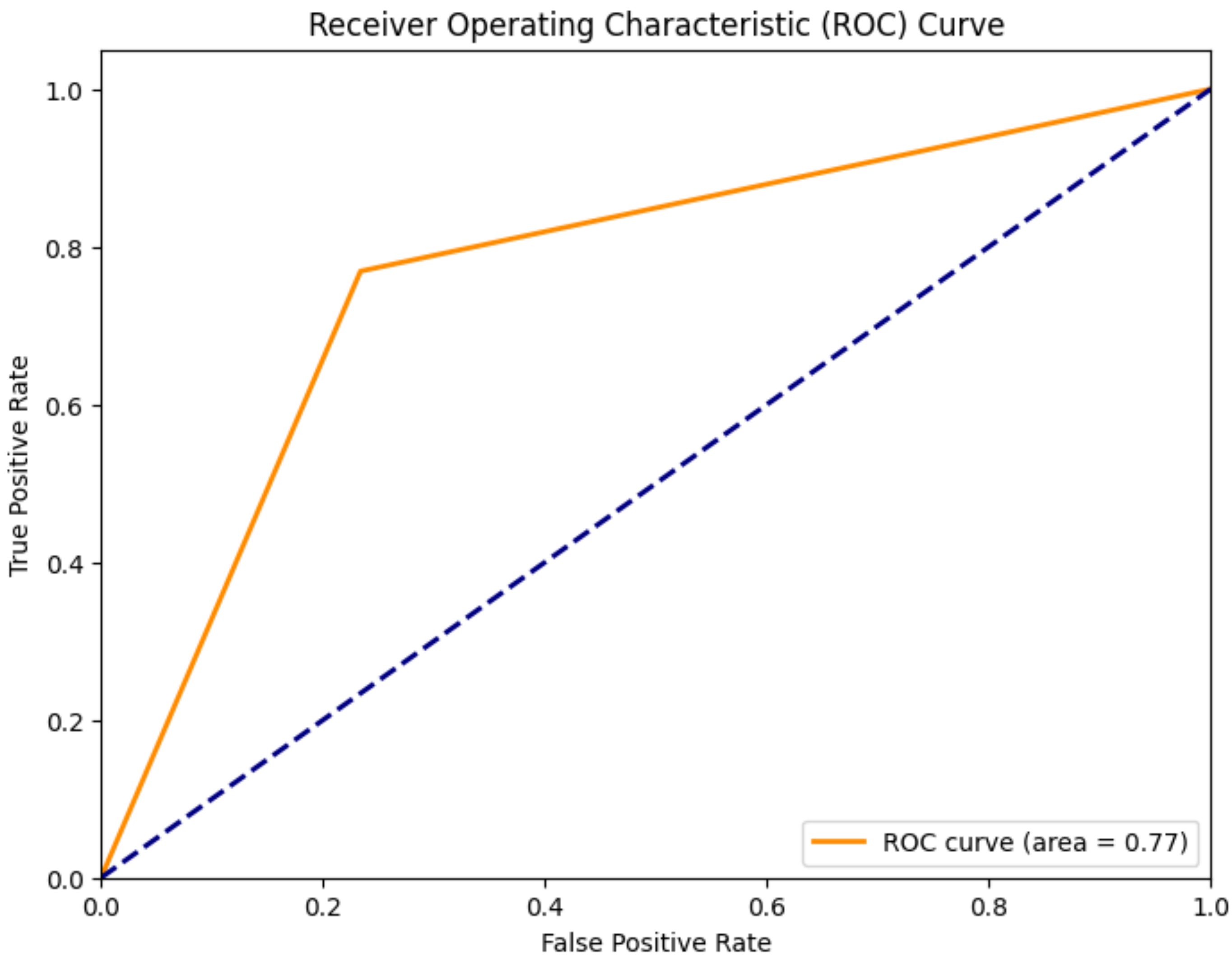
```
conf_matrix = confusion_matrix(ytest, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap='Blues', xticklabels=['Benign'
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
```



```
from sklearn.metrics import classification_report
# accuracy measures by classification_report()
result = classification_report(y_test,Y_pred_classes)

# print the result
print(result)
```

```
              precision    recall  f1-score   support

           0       0.78      0.61      0.68       864
           1       0.66      0.82      0.73       801

    accuracy                           0.71      1665
   macro avg       0.72      0.71      0.71      1665
weighted avg       0.72      0.71      0.71      1665
```

```
fpr, tpr, thresholds = roc_curve(ytest, y_pred)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % r
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



```
SVM = pd.DataFrame({'Model': ["Support Vector Machine"],
                    'AUC Score' : [metrics.roc_auc_score(ytest, y_pred)],
                'Precision Score': [metrics.precision_score(ytest, y_pred)],
                'Recall Score': [metrics.recall_score(ytest, y_pred)],
                'Accuracy Score': [metrics.accuracy_score(ytest, y_pred)],
                 'f1-score': [metrics.f1_score(ytest, y_pred)]})

# appending our result table
result_tabulation = pd.concat([result_tabulation, SVM], ignore_index=True)
result_tabulation
```

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	f1-score
0	Basic Neural Network	0.738374	0.736774	0.712859	0.739339	0.724619
1	Convolutional Neural Network	0.762504	0.743682	0.771536	0.762162	0.757353
2	VGG-16: Transfer Learning	0.713262	0.659617	0.817728	0.709309	0.730212
3	Decision Tree	0.713262	0.659617	0.817728	0.709309	0.730212
4	Gradient Boosting	0.762458	0.744270	0.770287	0.762162	0.757055

Next steps: View recommended plots