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

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

```
import tensorflow as tf
from tensorflow.python.ops.gen_math_ops import sub
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image, ImageOps
import glob
import os
import tqdm
from sklearn.model_selection import StratifiedKFold
import cv2
from tensorflow.keras.callbacks import EarlyStopping,ModelCheckpoint,ReduceLROnPlateau
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDi
import pandas as pd
import random
from keras.preprocessing.image import ImageDataGenerator, load_img
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.applications.resnet50 import ResNet50
from keras.utils.np_utils import to_categorical # convert labels to one-hot-encoding
from sklearn.metrics import accuracy_score
from keras import backend as K
```

Load Images

Get 300 benign images and 300 malignant images total from the 2020 ISIC Challenge for our train, validation, and test dataset. We will use 70% of our images for training, 15% for validation, and 15% for testing.

```
def load_and_crop(image_path, crop_size, normalized=False):
    # image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
    image = cv2.imread(image_path)
    # percent by which the image is resized
    scale_percent = 10
    # calculate the 10 percent of original dimensions
    width = int(image.shape[1] * scale_percent / 100) # dont use, raw images are not i
    height = int(image.shape[0] * scale_percent / 100) # dont use
    # dsize = (width, height)
    dsize = (256,256) # resize to 256 * 256, can use this code to resize the image aga
    image = cv2.resize(image, dsize)
    width, height, color_channel = image.shape # get dimensions
```

```
    if normalized:
        return np.array(image).astype(np.float32) / 255.0
    else:
        return np.array(image).astype(np.float32)

cell_types = ['benign','malignant']
cell_inds = np.arange(0, len(cell_types))
benign_data = []
malignant_data = []
x_data = []
y_data = []

# benign
benign_images = glob.glob(os.path.join('/content/drive/MyDrive/ML_Imaging_Final_Projec
benign_data += [load_and_crop(image_path, 128, normalized = True) for image_path in be
y_data += [0]*300 # make the total number of images to be 600 (300 benign + 300 mal)

# malignant
malignant_images = glob.glob(os.path.join('/content/drive/MyDrive/ML_Imaging_Final_Pro
malignant_data += [load_and_crop(image_path, 128, normalized = True) for image_path in
y_data += [1]*300

# combine benign and malignant
x_data = benign_data + malignant_data


print('Total number of images:', len(x_data))
print('Total number of labels:', len(y_data))
```

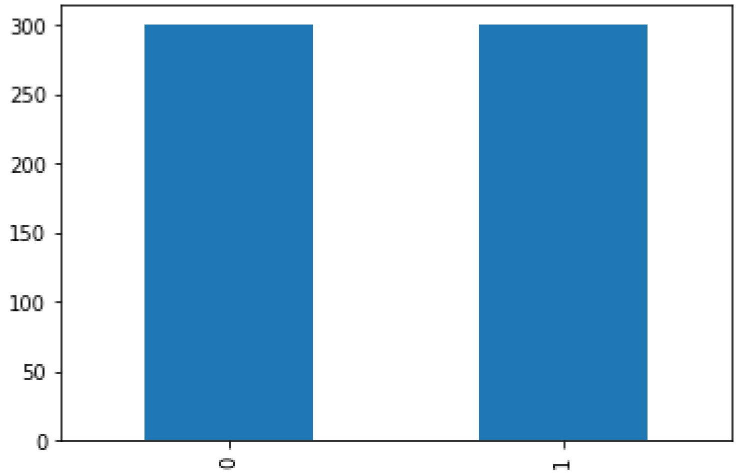
```
Total number of images: 600
Total number of labels: 600
```

```
# data frame with all the filenames and labels
df = pd.DataFrame({
    'filename': benign_images[0:300] + malignant_images[0:300],
    'category': y_data
})
df.head()
```

	filename	category
0	/content/drive/MyDrive/ML_Imaging_Final_Projec...	0
1	/content/drive/MyDrive/ML_Imaging_Final_Projec...	0
2	/content/drive/MyDrive/ML_Imaging_Final_Projec...	0
3	/content/drive/MyDrive/ML_Imaging_Final_Projec...	0
4	/content/drive/MyDrive/ML_Imaging_Final_Projec...	0

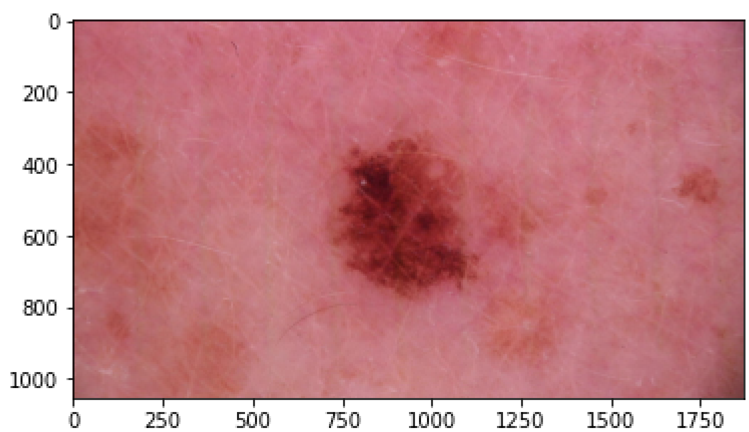
```
# visualize total count in data
df['category'].value_counts().plot.bar()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f623fd28f10>




```
# see sample image
sample = random.choice(df['filename'])
image = load_img(sample)
plt.imshow(image)
```

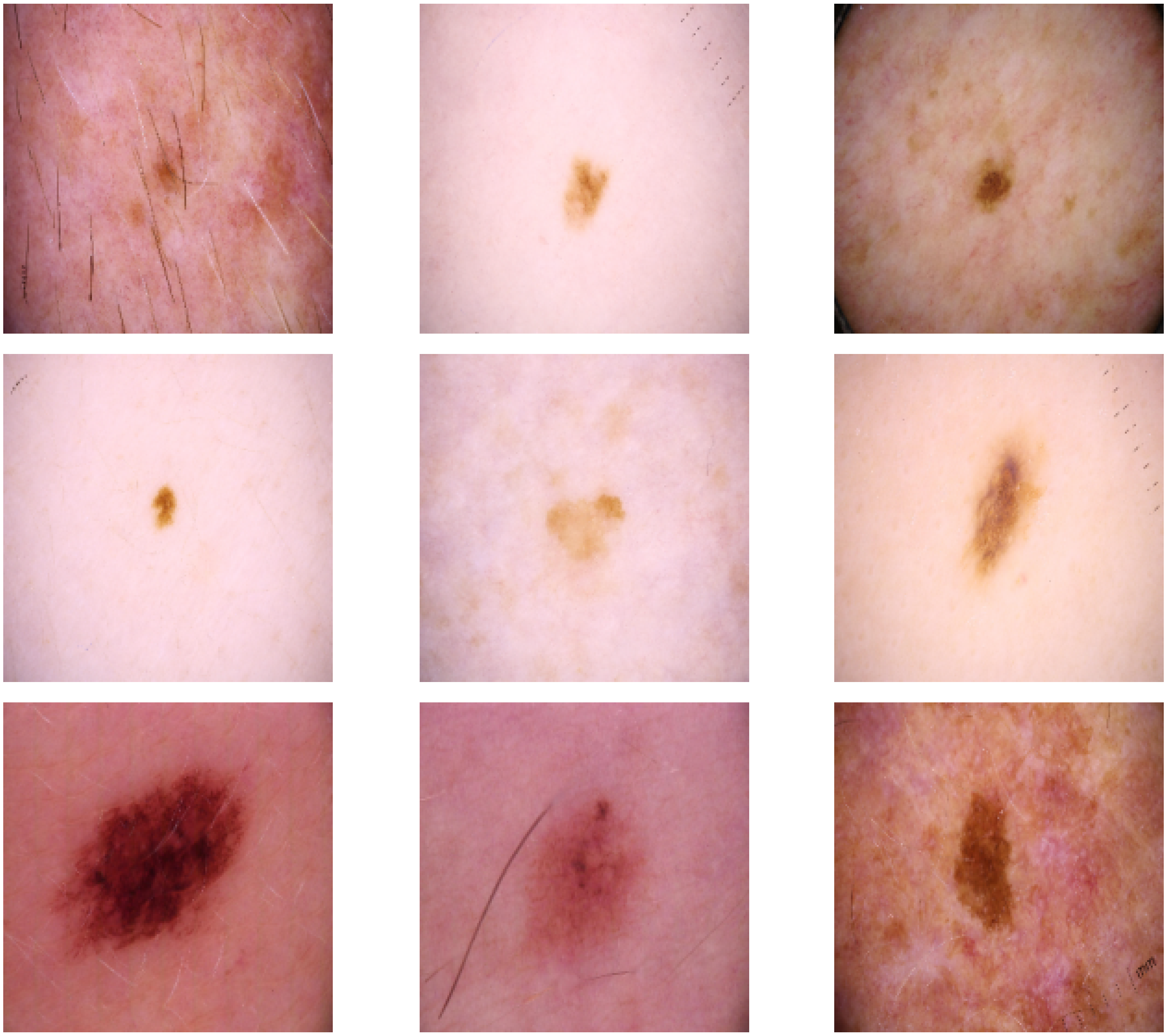
<matplotlib.image.AxesImage at 0x7f623f7072d0>




```
# visualize benign images

print('Display Benign Images')

# adjust the size of your images
plt.figure(figsize=(10,8))

# iterate and plot first 9 benign images
for i in range(9):
    plt.subplot(3, 3, i + 1)
    img = cv2.cvtColor(benign_data[i], cv2.COLOR_BGR2RGB)
    plt.imshow(img, cmap='gray')
    plt.axis('off')
```

```
# adjust subplot parameters to give specified padding
plt.tight_layout()
```

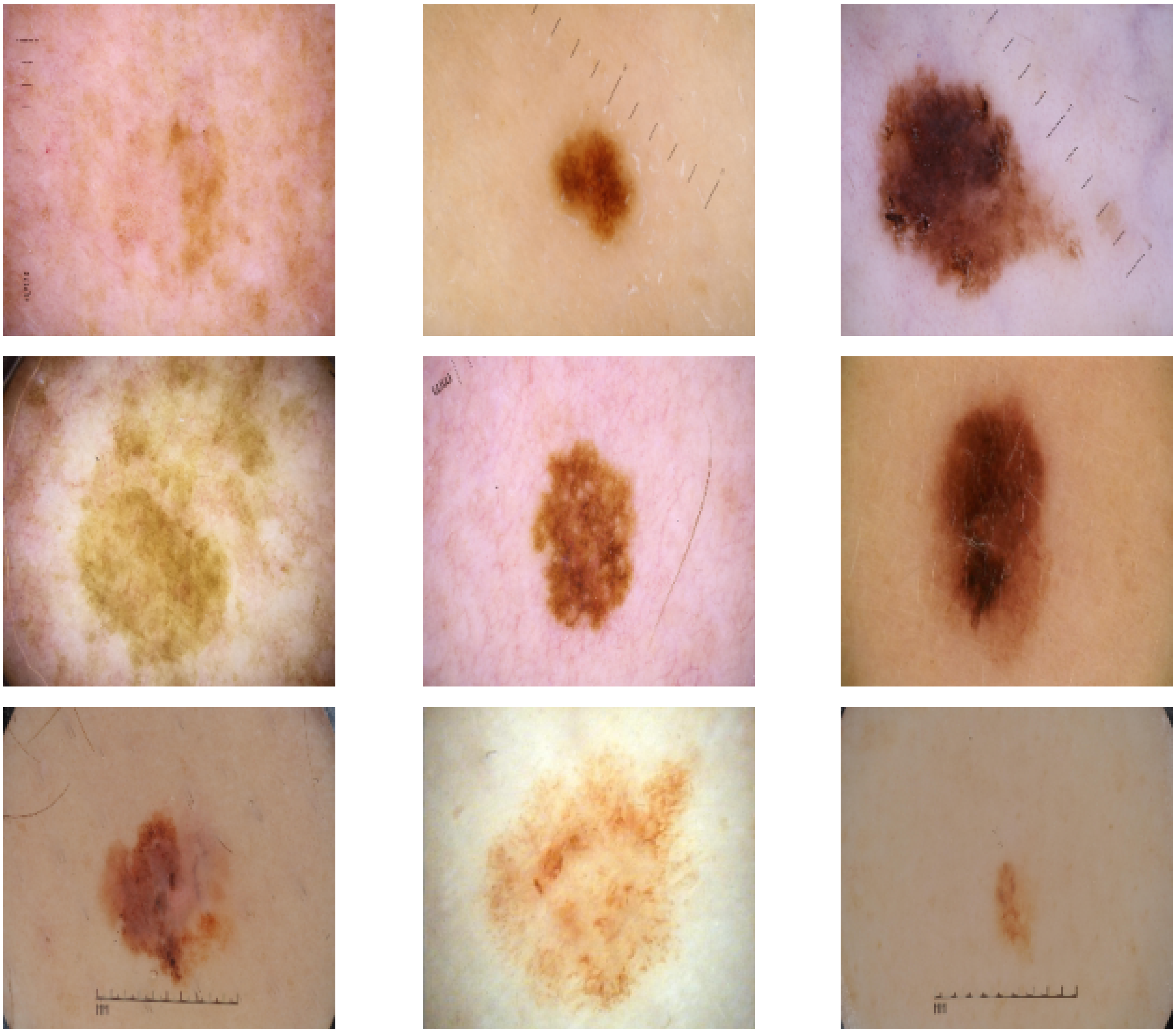
```
Display Benign Images
```



```
# visualize malignant images
print('Display Malignant Images')

# adjust the size of your images
plt.figure(figsize=(10,8))

# iterate and plot first 9 malignant images
for i in range(9):
    plt.subplot(3, 3, i + 1)
    img = cv2.cvtColor(malignant_data[i], cv2.COLOR_BGR2RGB)
    plt.imshow(img, cmap='gray')
    plt.axis('off')
```

```
# adjust subplot parameters to give specified padding
plt.tight_layout()
```

Display Malignant Images



```
# generating train/val/test splits
# convert 1 to malignant and 0 to benign since we are using class mode = categorical
# df["category"] = df["category"].replace({0: 'benign', 1: 'malignant'})
train_df, validate_df, test_df = np.split(df.sample(frac=1, random_state=42), [int(.6
train_df = train_df.reset_index(drop=True)
validate_df = validate_df.reset_index(drop=True)
test_df = test_df.reset_index(drop=True)


# train: 360 imgs, shape: 256x256x3/img
x_train = np.array([load_and_crop(x, 128, normalized = True) for x in train_df['filena
y_train = np.array(train_df['category'])

# valid: 120 imgs, shape: 256x256x3/img
x_val = np.array([load_and_crop(x, 128, normalized = True) for x in validate_df['filen
y_val = np.array(validate_df['category'])

# test: 120 imgs, shape: 256x256x3/img
```

```
x_test = np.array([load_and_crop(x, 128, normalized = True) for x in test_df['filename
y_test = np.array(test_df['category'])
```

```
y_train= to_categorical(y_train, num_classes= 2)
y_val = to_categorical(y_val, num_classes= 2)
y_test = to_categorical(y_test, num_classes= 2)
```

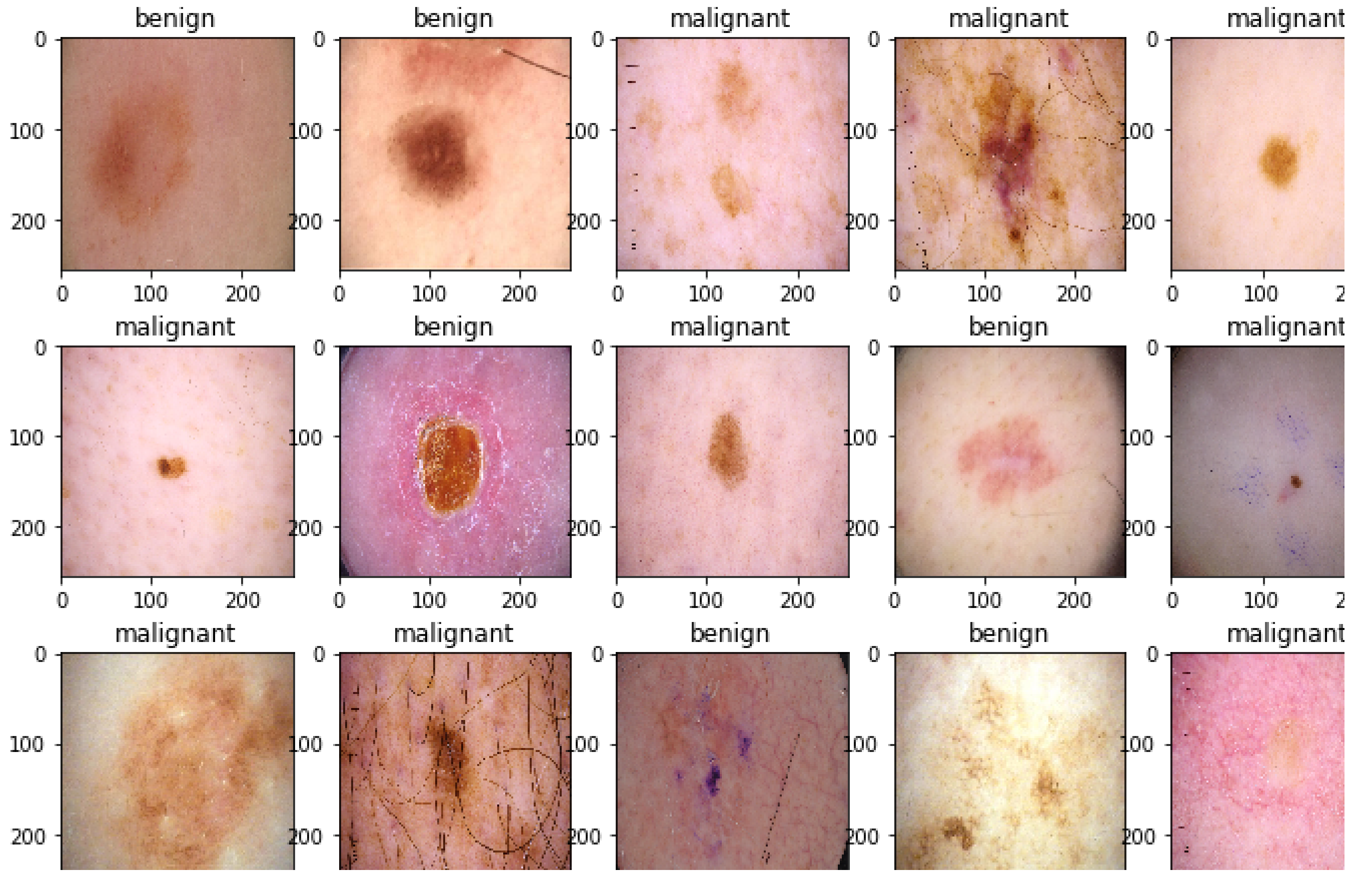
```
print(x_train.shape, y_train.shape)
print(x_val.shape, y_val.shape)
print(x_test.shape, y_test.shape)
```

```
(360, 256, 256, 3) (360, 2)
(120, 256, 256, 3) (120, 2)
(120, 256, 256, 3) (120, 2)
```

```
# distribution of labels
print('{} benign images in training'.format(len(train_df.loc[train_df['category'] == (
print('{} malignant images in training'.format(len(train_df.loc[train_df['category'] =
print('{} benign images in validation'.format(len(validate_df.loc[train_df['category']
print('{} malignant images in validation'.format(len(validate_df.loc[train_df['categor
print('{} benign images in testing'.format(len(test_df.loc[train_df['category']== 0]))
print('{} malignant images in testing'.format(len(test_df.loc[train_df['category']== 1
```

```
180 benign images in training
180 malignant images in training
62 benign images in validation
58 malignant images in validation
62 benign images in testing
58 malignant images in testing
```

```
# display first 15 images of train, and how they are classified
w=40
h=30
fig=plt.figure(figsize=(12, 8))
columns = 5
rows = 3

for i in range(1, columns*rows +1):
    ax = fig.add_subplot(rows, columns, i)
    if y_train[i][0] == 0:
        ax.title.set_text('benign')
    else:
        ax.title.set_text('malignant')
    img = cv2.cvtColor(x_train[i],cv2.COLOR_RGB2BGR)
    plt.imshow(img, interpolation='nearest')
```



Evaluation Metrics

```
def roc_plot(y_trues, y_preds):
  # Evaluation of the custom_model
  # ROC curve & AUC score
  tpr, fpr, threshold = roc_curve(y_trues, y_preds)
  auc = roc_auc_score(y_trues, y_preds)
  plt.plot(fpr, tpr, label='AUC score=%.3f' % auc, marker='o', markersize=1)
  plt.xlabel('False Positive Rate'); plt.xlim((0, 1))
  plt.ylabel('True Positive Rate'); plt.ylim((0, 1))
  plt.title('ROC Curve')
  plt.legend()

def confusionmatrix(y_trues, y_preds):
  # Confusion matrix
  matrix = confusion_matrix(y_trues, y_preds)
  fig = ConfusionMatrixDisplay(matrix)
  fig.plot()
  plt.title('Confusion Matrix')
  plt.show()

  # Precision, Recall, and F1 score
  tn, fp, fn, tp = matrix[0, 0], matrix[0, 1], matrix[1, 0], matrix[1, 1]
  precision = tp / (tp + fp)
  recall = tp / (tp + fn)
```

```
    f1_score = 2 * (precision * recall) / (precision + recall)
    eval_tab = pd.DataFrame({'Precision': [precision], 'Recall': [recall], 'F1 Score': [
    print(eval_tab)
    print(classification_report(y_trues, y_preds, target_names = ['benign (Class 0)','ma
```

Models

Baseline Models

Simple CNN Model

```
# Baseline model without physical layer
image_size = (256, 256, 3)
custom_model = tf.keras.models.Sequential([tf.keras.layers.Input(image_size),
                                           tf.keras.layers.Conv2D(16, 3, padding="same
                                           tf.keras.layers.Conv2D(16, 3, padding="same
                                          #  tf.keras.layers.GaussianNoise(0.2),
                                          #  tf.keras.layers.BatchNormalization(),
                                           tf.keras.layers.MaxPool2D((2, 2), strides=2
                                           tf.keras.layers.Conv2D(16, 3, padding="same
                                           tf.keras.layers.Conv2D(16, 3, padding="same
                                          #  tf.keras.layers.BatchNormalization(),
                                           tf.keras.layers.MaxPool2D((2, 2), strides=2
                                           tf.keras.layers.Flatten(),
                                           tf.keras.layers.Dense(128, activation="relu
                                           tf.keras.layers.GaussianDropout(0.3),
                                           tf.keras.layers.Dense(2, activation="softma

custom_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 16)      448

 conv2d_1 (Conv2D)           (None, 256, 256, 16)      2320

 max_pooling2d (MaxPooling2D  (None, 128, 128, 16)     0
 )

 conv2d_2 (Conv2D)           (None, 128, 128, 16)      2320

 conv2d_3 (Conv2D)           (None, 128, 128, 16)      2320

 max_pooling2d_1 (MaxPooling  (None, 64, 64, 16)       0
 2D)
```

```
 flatten (Flatten)           (None, 65536)             0

 dense (Dense)               (None, 128)               8388736

 gaussian_dropout (GaussianD  (None, 128)              0
 ropout)

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 8,396,402
Trainable params: 8,396,402
Non-trainable params: 0
_________________________________________________________________
```

```
lr = 1e-5
epochs = 150
batch_size = 15
opt = tf.optimizers.Adam(learning_rate=lr)
# to prevent overfitting, stop the learning after 10 epochs and when val_loss value do
earlystop = EarlyStopping(patience=10)

# reduce learning rate when accuracy doesn't increase for 2 steps
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=5,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=1e-7)

callbacks = [earlystop, learning_rate_reduction]

custom_model.compile(optimizer=opt , loss='categorical_crossentropy', metrics=['accura


history = custom_model.fit(x_train, y_train, epochs=epochs, verbose = 1, validation_da
```

```
Epoch 1/150
12/12 [==============================] - 14s 308ms/step - loss: 0.6765 - accu
Epoch 2/150
12/12 [==============================] - 2s 203ms/step - loss: 0.6578 - accur
Epoch 3/150
12/12 [==============================] - 2s 204ms/step - loss: 0.6351 - accur
Epoch 4/150
12/12 [==============================] - 2s 204ms/step - loss: 0.6270 - accur
Epoch 5/150
12/12 [==============================] - 2s 206ms/step - loss: 0.6276 - accur
Epoch 6/150
12/12 [==============================] - 2s 205ms/step - loss: 0.6179 - accur
Epoch 7/150
12/12 [==============================] - 2s 207ms/step - loss: 0.6184 - accur
Epoch 8/150
12/12 [==============================] - 2s 209ms/step - loss: 0.6086 - accur
Epoch 9/150
12/12 [==============================] - 2s 205ms/step - loss: 0.6044 - accur
```

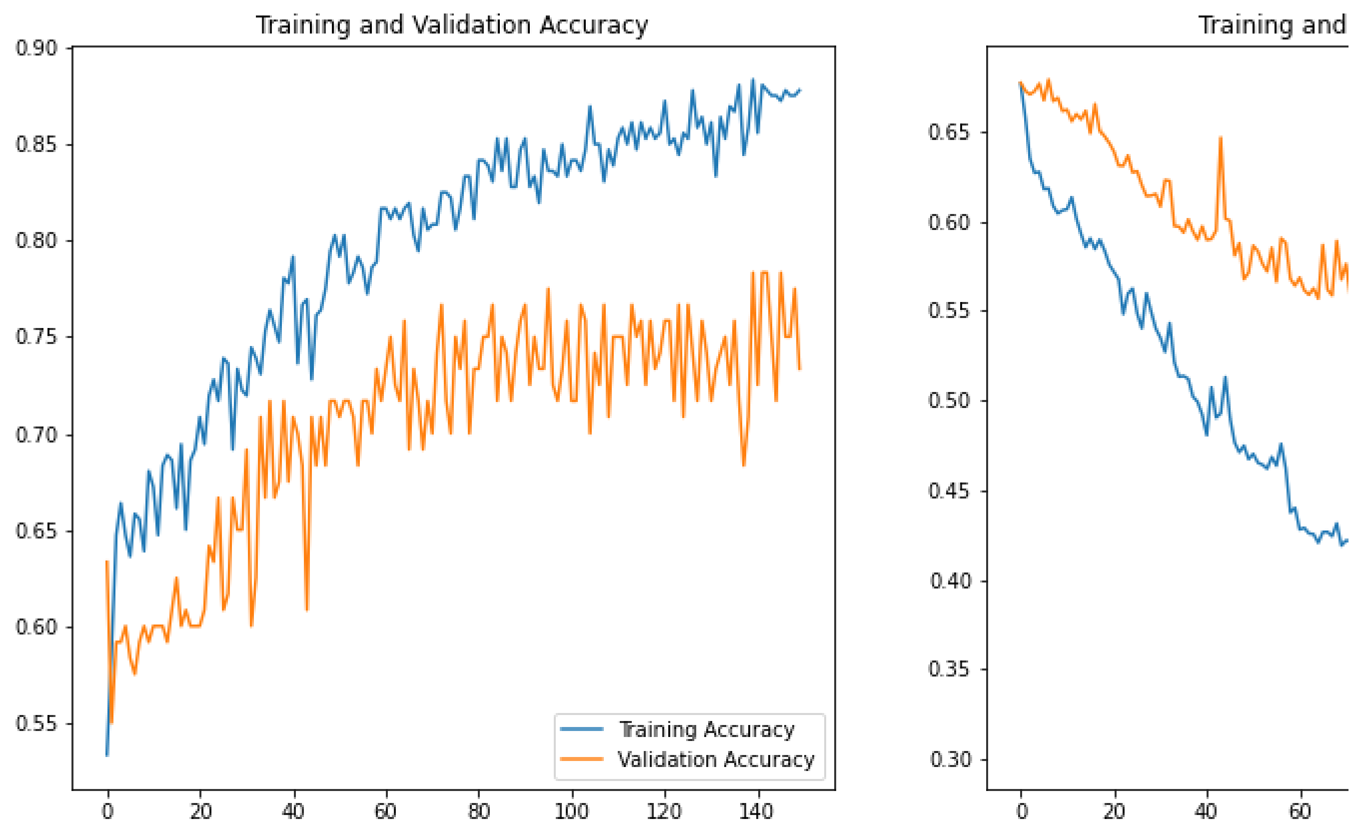
```
Epoch 10/150
12/12 [==============================] - 2s 206ms/step - loss: 0.6061 - accur
Epoch 11/150
12/12 [==============================] - 2s 205ms/step - loss: 0.6067 - accur
Epoch 12/150
12/12 [==============================] - 2s 204ms/step - loss: 0.6134 - accur
Epoch 13/150
12/12 [==============================] - 2s 204ms/step - loss: 0.6016 - accur
Epoch 14/150
12/12 [==============================] - 2s 206ms/step - loss: 0.5930 - accur
Epoch 15/150
12/12 [==============================] - 2s 208ms/step - loss: 0.5855 - accur
Epoch 16/150
12/12 [==============================] - 2s 207ms/step - loss: 0.5905 - accur
Epoch 17/150
12/12 [==============================] - 2s 206ms/step - loss: 0.5845 - accur
Epoch 18/150
12/12 [==============================] - 2s 206ms/step - loss: 0.5897 - accur
Epoch 19/150
12/12 [==============================] - 2s 206ms/step - loss: 0.5835 - accur
Epoch 20/150
12/12 [==============================] - 2s 205ms/step - loss: 0.5757 - accur
Epoch 21/150
12/12 [==============================] - 2s 206ms/step - loss: 0.5717 - accur
Epoch 22/150
12/12 [==============================] - 2s 206ms/step - loss: 0.5676 - accur
Epoch 23/150
12/12 [==============================] - 2s 205ms/step - loss: 0.5482 - accur
Epoch 24/150
12/12 [==============================] - 2s 209ms/step - loss: 0.5596 - accur
Epoch 25/150
12/12 [==============================] - 2s 207ms/step - loss: 0.5625 - accur
Epoch 26/150
12/12 [==============================] - 2s 206ms/step - loss: 0.5485 - accur
Epoch 27/150
12/12 [==============================] - 2s 208ms/step - loss: 0.5402 - accur
Epoch 28/150
12/12 [==============================] - 2s 207ms/step - loss: 0.5598 - accur
```

```
# save baseline simple cnn model
baseline_model_json = custom_model.to_json()

with open("baseline_model.json", "w") as json_file:
    json_file.write(baseline_model_json)

# serialize weights to HDF5
custom_model.save_weights("baseline_model.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

```
# plot train and validation loss
```

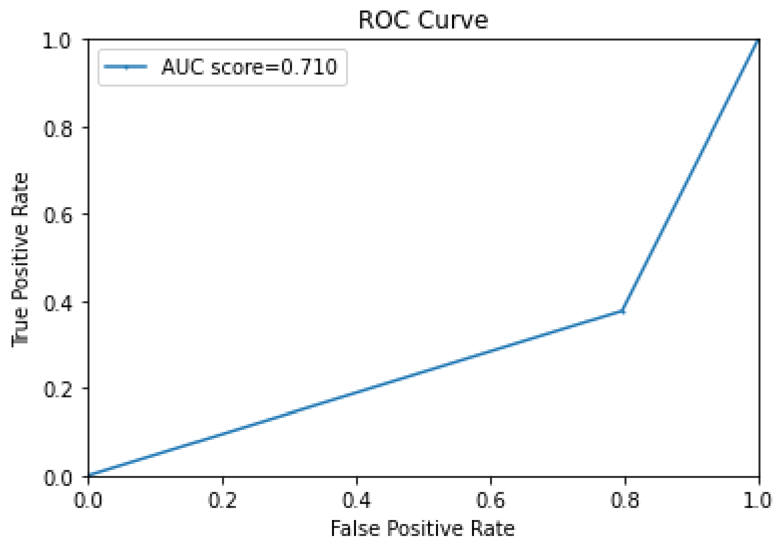
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

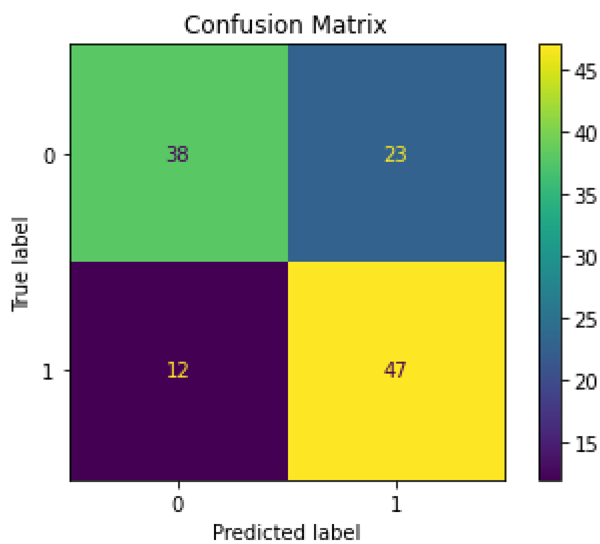



```
# evaluate model
y_pred = np.argmax(custom_model.predict(x_test), axis=1)
acc = custom_model.evaluate(x_test, y_test)
```

```
4/4 [==============================] - 0s 72ms/step - loss: 0.6295 - accuracy: 0
```

```
# ROC curve and confusion matrix
y_labels = np.argmax(y_test.astype(int),axis=1)
roc_plot(y_labels, y_pred)
confusionmatrix(y_labels, y_pred)
```






```
   Precision    Recall  F1 Score
0   0.671429  0.79661  0.728682
                     precision    recall  f1-score   support

   benign (Class 0)       0.76      0.62      0.68        61
malignant (Class 1)       0.67      0.80      0.73        59

           accuracy                           0.71       120
          macro avg       0.72      0.71      0.71       120
       weighted avg       0.72      0.71      0.71       120
```

```
# Clear memory, because of memory overload
del custom_model
K.clear_session()
```

ResNet50 model

Try to see if ResNet50 model with data augmentation improves performance. The model ends up being overfit so we will not use this model.

```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        zca_epsilon=1e-06,
        rotation_range=0,
        width_shift_range=0.1,
        height_shift_range=0.1,
        shear_range=0.,
        zoom_range=0.,
        channel_shift_range=0.,
        fill_mode='nearest',
        cval=0.,
        horizontal_flip=True,
        vertical_flip=False,
        rescale=None,
        preprocessing_function=None,
        data_format=None,
        validation_split=0.0)

Train_Datagen = ImageDataGenerator(preprocessing_function=tf.keras.applications.resnet
#Train_Datagen = ImageDataGenerator(dtype = 'float32', preprocessing_function=tf.keras
Val_Datagen = ImageDataGenerator(dtype = 'float32', preprocessing_function=tf.keras.ap
Test_Datagen = ImageDataGenerator(dtype = 'float32', preprocessing_function=tf.keras.a


# resnet model

resnet_model = ResNet50(include_top=True,
                  weights= None,
                  input_tensor=None,
                  input_shape=image_size,
                  pooling='avg',
                  classes=2)

resnet_model.compile(optimizer = opt ,loss = "categorical_crossentropy", metrics=["acc


resnet_model.summary()
```

```
Model: "resnet50"
_________________________________________________________________________________
 Layer (type)                    Output Shape          Param #     Connected to
```

```
==================================================================================
 input_1 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 262, 262, 3)  0           ['input_1[0]

 conv1_conv (Conv2D)            (None, 128, 128, 64  9472        ['conv1_pad[
                                )

 conv1_bn (BatchNormalization)  (None, 128, 128, 64  256         ['conv1_conv
                                )

 conv1_relu (Activation)        (None, 128, 128, 64  0           ['conv1_bn[0
                                )

 pool1_pad (ZeroPadding2D)      (None, 130, 130, 64  0           ['conv1_relu
                                )

 pool1_pool (MaxPooling2D)      (None, 64, 64, 64)   0           ['pool1_pad[

 conv2_block1_1_conv (Conv2D)   (None, 64, 64, 64)   4160        ['pool1_pool

 conv2_block1_1_bn (BatchNormal  (None, 64, 64, 64)  256         ['conv2_bloc
 ization)

 conv2_block1_1_relu (Activatio  (None, 64, 64, 64)  0           ['conv2_bloc
 n)

 conv2_block1_2_conv (Conv2D)   (None, 64, 64, 64)   36928       ['conv2_bloc

 conv2_block1_2_bn (BatchNormal  (None, 64, 64, 64)  256         ['conv2_bloc
 ization)

 conv2_block1_2_relu (Activatio  (None, 64, 64, 64)  0           ['conv2_bloc
 n)

 conv2_block1_0_conv (Conv2D)   (None, 64, 64, 256)  16640       ['pool1_pool

 conv2_block1_3_conv (Conv2D)   (None, 64, 64, 256)  16640       ['conv2_bloc

 conv2_block1_0_bn (BatchNormal  (None, 64, 64, 256)  1024       ['conv2_bloc
 ization)

 conv2_block1_3_bn (BatchNormal  (None, 64, 64, 256)  1024       ['conv2_bloc
 ization)

 conv2_block1_add (Add)         (None, 64, 64, 256)  0           ['conv2_bloc
                                                                  'conv2_bloc

 conv2_block1_out (Activation)  (None, 64, 64, 256)  0           ['conv2_bloc

 conv2_block2_1_conv (Conv2D)   (None, 64, 64, 64)   16448       ['conv2_bloc

 conv2_block2_1_bn (BatchNormal  (None, 64, 64, 64)  256         ['conv2_bloc
```

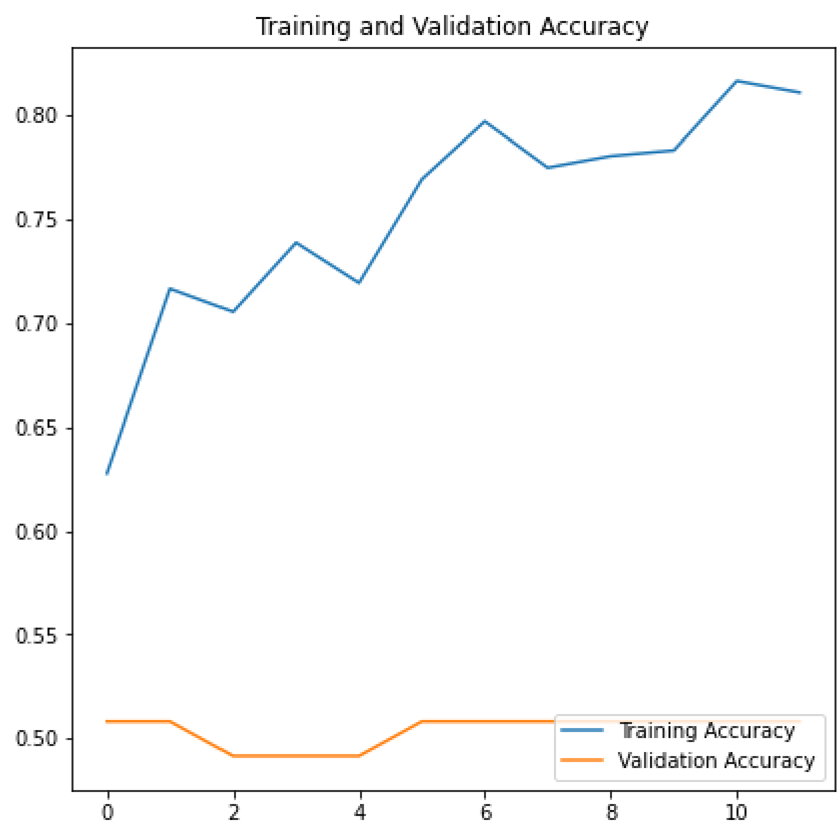
```
#datagen.fit(x_train)
epochs=75
resnet_history = resnet_model.fit(Train_Datagen.flow(x_train, y_train, batch_size=batc
                         validation_data=(x_val, y_val),
                         epochs=epochs, verbose=1, workers=4, callbacks = callbacks)


# resnet_history = resnet_model.fit(x_train, y_train, validation_data = (x_val, y_val)
#                          epochs=epochs, verbose=1, workers=4, callbacks = callbacks)
```

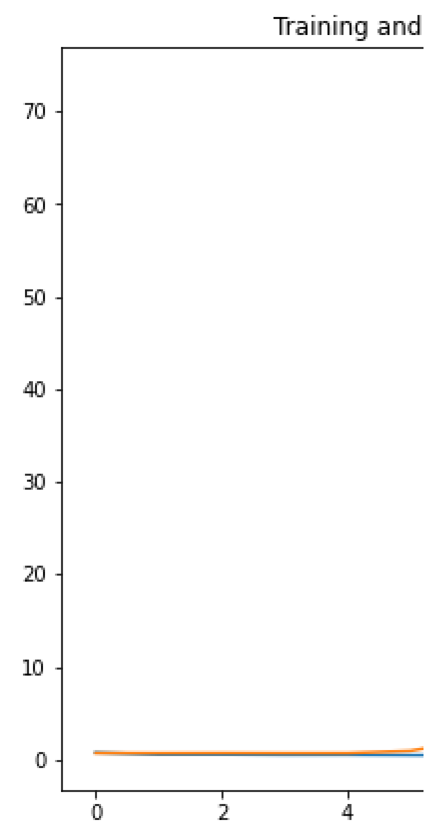
```
Epoch 1/75
24/24 [==============================] - 20s 534ms/step - loss: 0.7588 - accuracy
Epoch 2/75
24/24 [==============================] - 12s 485ms/step - loss: 0.5531 - accuracy
Epoch 3/75
24/24 [==============================] - 12s 485ms/step - loss: 0.5503 - accuracy
Epoch 4/75
24/24 [==============================] - 12s 485ms/step - loss: 0.5124 - accuracy
Epoch 5/75
24/24 [==============================] - 12s 486ms/step - loss: 0.5178 - accuracy
Epoch 6/75
24/24 [==============================] - ETA: 0s - loss: 0.4853 - accuracy: 0.76
Epoch 6: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-06.
24/24 [==============================] - 12s 485ms/step - loss: 0.4853 - accuracy
Epoch 7/75
24/24 [==============================] - 12s 484ms/step - loss: 0.4608 - accuracy
Epoch 8/75
24/24 [==============================] - 12s 484ms/step - loss: 0.4364 - accuracy
Epoch 9/75
24/24 [==============================] - 12s 485ms/step - loss: 0.4644 - accuracy
Epoch 10/75
24/24 [==============================] - 12s 485ms/step - loss: 0.4378 - accuracy
Epoch 11/75
24/24 [==============================] - ETA: 0s - loss: 0.3951 - accuracy: 0.81
Epoch 11: ReduceLROnPlateau reducing learning rate to 1.249999968422344e-06.
24/24 [==============================] - 12s 484ms/step - loss: 0.3951 - accuracy
Epoch 12/75
24/24 [==============================] - 12s 490ms/step - loss: 0.4024 - accuracy
```

```
# save model
# serialize model to JSON
resnet50_json = resnet_model.to_json()

with open("resnet50.json", "w") as json_file:
    json_file.write(resnet50_json)

# serialize weights to HDF5
resnet_model.save_weights("resnet50.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

```
# plot train and validation loss
acc = resnet_history.history['accuracy']
val_acc = resnet_history.history['val_accuracy']
loss = resnet_history.history['loss']
val_loss = resnet_history.history['val_loss']

plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```






```
# Test set accuracy
y_pred = np.argmax(resnet_model.predict(x_test), axis=1)
acc = resnet_model.evaluate(x_test, y_test)
```

```
4/4 [==============================] - 1s 236ms/step - loss: 75.7775 - accuracy:
```

```
# Model Evaluation
y_labels = np.argmax(y_test.astype(int),axis=1)
roc_plot(y_labels, y_pred)
confusionmatrix(y_labels, y_pred)
```


```
del resnet_model
K.clear_session()
```

BGR Physical Layer Model

```
# B, G, R filters
class BGR_Filter(tf.keras.layers.Layer):
    def __init__(self, is_train=False):
        # code here
        super(BGR_Filter, self).__init__()
        self.is_train = is_train

    def build(self, input_shape):
        # initialize BGR weight with 0 mean 0.05 std
        weight_init = tf.random_normal_initializer(0,0.05)
        # code here
        self.bgrfilter = tf.keras.layers.Conv2D(filters = 1,kernel_size = (1,1), use_b
        #tf.Variable(initial_value=weight_init(shape=(input_shape[-1], )), dtype='floa

    def call(self, inputs):
        # code here
        #inputs = inputs* self.weight
        output = self.bgrfilter(inputs)
        #output = tf.reshape(tf.reduce_sum(inputs, axis = -1), [64,64,1])
        return output
```

Precision Recall F1 Score

```
bgr_model = tf.keras.models.Sequential([tf.keras.layers.Input(image_size),
                                        # code here
                                        BGR_Filter(is_train = True),
                                        tf.keras.layers.Conv2D(16, 3, padding="same
                                        tf.keras.layers.Conv2D(16, 3, 2, padding="s
                                       #  tf.keras.layers.GaussianNoise(0.2),
                                       #  tf.keras.layers.BatchNormalization(),
                                        tf.keras.layers.MaxPool2D((2, 2), strides=2
                                        tf.keras.layers.Conv2D(16, 3, padding="same
                                        tf.keras.layers.Conv2D(16, 3, 2, padding="s
                                       #  tf.keras.layers.BatchNormalization(),
                                        tf.keras.layers.MaxPool2D((2, 2), strides=2
                                        tf.keras.layers.Flatten(),
                                        tf.keras.layers.Dense(128, activation="relu
                                        tf.keras.layers.GaussianDropout(0.2),
                                        tf.keras.layers.Dense(2, activation="softma
```

```
bgr_model.summary()
```

```
Model: "sequential"
```

```
 Layer (type)                Output Shape              Param #   
=================================================================
 bgr__filter (BGR_Filter)    (None, 256, 256, 1)       3         
                                                                 
 conv2d (Conv2D)             (None, 256, 256, 16)      160       
                                                                 
 conv2d_1 (Conv2D)           (None, 128, 128, 16)      2320      
                                                                 
 max_pooling2d (MaxPooling2D  (None, 64, 64, 16)       0         
 )                                                               
                                                                 
 conv2d_2 (Conv2D)           (None, 64, 64, 16)        2320      
                                                                 
 conv2d_3 (Conv2D)           (None, 32, 32, 16)        2320      
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 16, 16, 16)       0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 4096)              0         
                                                                 
 dense (Dense)               (None, 128)               524416    
                                                                 
 gaussian_dropout (GaussianD  (None, 128)              0         
 ropout)                                                         
                                                                 
 dense_1 (Dense)             (None, 2)                 258       
                                                                 
=================================================================
Total params: 531,797
Trainable params: 531,797
Non-trainable params: 0
_________________________________________________________________
```

```
lr = 5e-5
opt = tf.optimizers.Adam(learning_rate=lr)
bgr_model.compile(optimizer=opt , loss='categorical_crossentropy', metrics=['accuracy'
```

```
epochs=150
bgr_history = bgr_model.fit(x_train,y_train, epochs=epochs, verbose = 1, validation_da
```

```
Epoch 1/150
12/12 [==============================] - 4s 201ms/step - loss: 0.6916 - accuracy
Epoch 2/150
12/12 [==============================] - 2s 127ms/step - loss: 0.6896 - accuracy
Epoch 3/150
12/12 [==============================] - 2s 126ms/step - loss: 0.6876 - accuracy
Epoch 4/150
12/12 [==============================] - 2s 127ms/step - loss: 0.6854 - accuracy
Epoch 5/150
12/12 [==============================] - 1s 126ms/step - loss: 0.6819 - accuracy
Epoch 6/150
12/12 [==============================] - 2s 126ms/step - loss: 0.6772 - accuracy
Epoch 7/150
```

```
12/12 [==============================] - 2s 127ms/step - loss: 0.6697 - accuracy
Epoch 8/150
12/12 [==============================] - 2s 126ms/step - loss: 0.6586 - accuracy
Epoch 9/150
12/12 [==============================] - 1s 125ms/step - loss: 0.6466 - accuracy
Epoch 10/150
12/12 [==============================] - 2s 127ms/step - loss: 0.6281 - accuracy
Epoch 11/150
12/12 [==============================] - 2s 126ms/step - loss: 0.6153 - accuracy
Epoch 12/150
12/12 [==============================] - 1s 125ms/step - loss: 0.6023 - accuracy
Epoch 13/150
12/12 [==============================] - 2s 126ms/step - loss: 0.5972 - accuracy
Epoch 14/150
12/12 [==============================] - 2s 126ms/step - loss: 0.5958 - accuracy
Epoch 15/150
11/12 [==========================>...] - ETA: 0s - loss: 0.5917 - accuracy: 0.67
Epoch 15: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-05.
12/12 [==============================] - 2s 127ms/step - loss: 0.5894 - accuracy
Epoch 16/150
12/12 [==============================] - 1s 125ms/step - loss: 0.5853 - accuracy
Epoch 17/150
12/12 [==============================] - 1s 125ms/step - loss: 0.5834 - accuracy
Epoch 18/150
12/12 [==============================] - 1s 125ms/step - loss: 0.5830 - accuracy
Epoch 19/150
12/12 [==============================] - 1s 126ms/step - loss: 0.5787 - accuracy
Epoch 20/150
11/12 [==========================>...] - ETA: 0s - loss: 0.5736 - accuracy: 0.68
Epoch 20: ReduceLROnPlateau reducing learning rate to 1.249999968422344e-05.
12/12 [==============================] - 2s 127ms/step - loss: 0.5772 - accuracy
Epoch 21/150
12/12 [==============================] - 1s 125ms/step - loss: 0.5794 - accuracy
Epoch 22/150
12/12 [==============================] - 2s 127ms/step - loss: 0.5804 - accuracy
```

```
# print BGR weights after training
bgr_model.get_weights()[0]
```

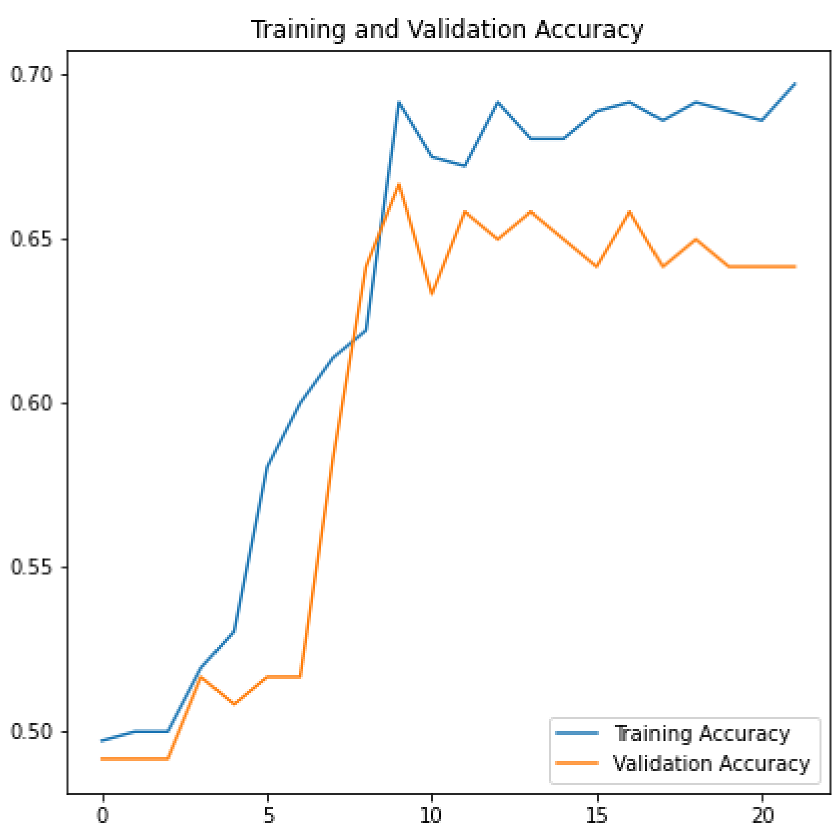
```
array([[[[-0.10497732],
         [-0.02994923],
         [ 0.03670643]]]], dtype=float32)
```

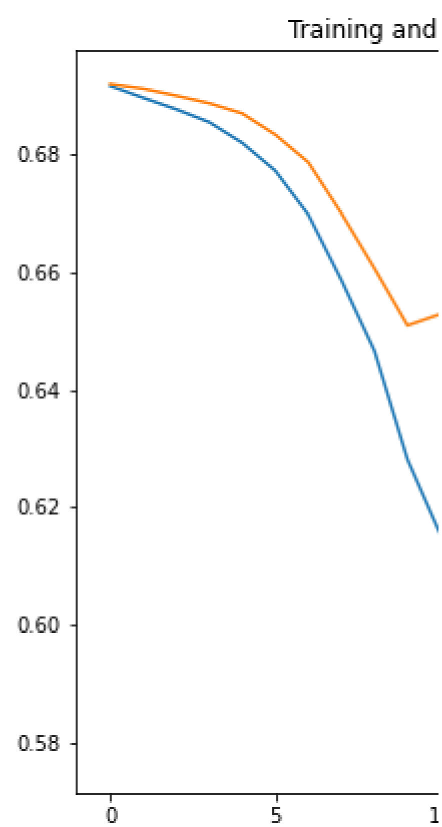
```
# save model
# serialize model to JSON
#bgr_json = bgr_model.to_json()

#with open("bgr.json", "w") as json_file:
#    json_file.write(bgr_json)

# serialize weights to HDF5
```

```
bgr_model.save_weights("bgr.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

```
# plot train and validation loss
acc = bgr_history.history['accuracy']
val_acc = bgr_history.history['val_accuracy']
loss = bgr_history.history['loss']
val_loss = bgr_history.history['val_loss']

plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

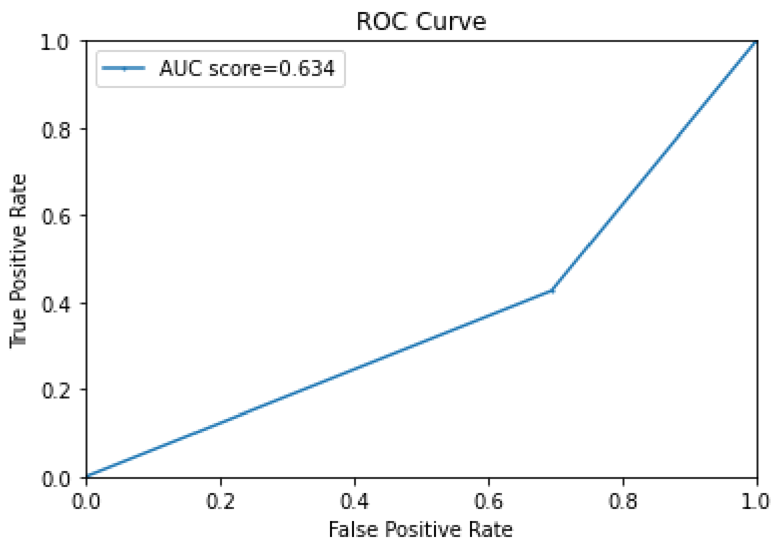


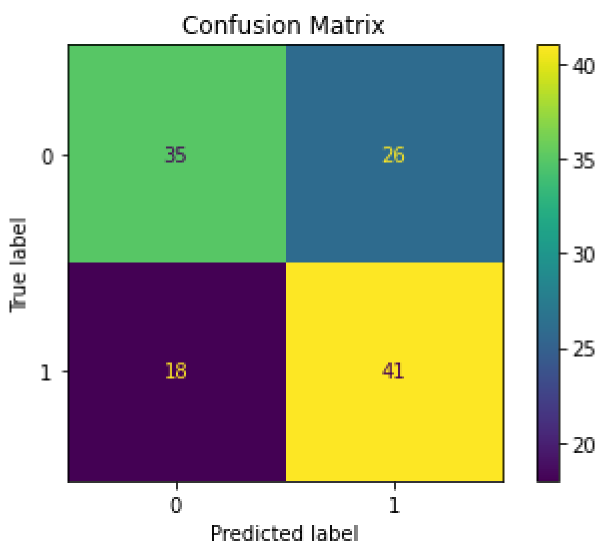



```
# Test set accuracy
y_pred = np.argmax(bgr_model.predict(x_test), axis=1)
acc = bgr_model.evaluate(x_test, y_test)
```

```
4/4 [==============================] - 0s 39ms/step - loss: 0.6852 - accuracy: 0
```

```
# Model Evaluation
y_labels = np.argmax(y_test.astype(int),axis=1)
roc_plot(y_labels, y_pred)
confusionmatrix(y_labels, y_pred)
```






```
   Precision    Recall  F1 Score
0    0.61194  0.694915  0.650794
                     precision    recall  f1-score   support

   benign (Class 0)       0.66      0.57      0.61        61
malignant (Class 1)       0.61      0.69      0.65        59

           accuracy                           0.63       120
          macro avg       0.64      0.63      0.63       120
       weighted avg       0.64      0.63      0.63       120
```

```
del bgr_model
```

```
K.clear_session()
```

Illumination Physical Layer

```
# BGR to gray
x_val_gray = []
x_train_gray = []
x_test_gray = []

for image in x_val:
  x_val_gray += [cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)]

print(len(x_val_gray), x_val_gray[0].shape)

for image in x_train:
  x_train_gray += [cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)]

print(len(x_train_gray), x_train_gray[0].shape)

for image in x_test:
  x_test_gray += [cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)]

print(len(x_test_gray), x_test_gray[0].shape)
```

```
120 (256, 256)
360 (256, 256)
120 (256, 256)
```

```
wavelength = .5e-3 # units are mm; assuming green light
delta_x = 0.5*wavelength # let's sample at nyquist rate
num_samples = 256

# in real world, microscope samples are 3D and have thickness, which introduce a phase
# For simplicity, let's further assume the sample thickness and amplitude are inversel
# the more light it absorb.
def convert_images(sample_amplitude):
    sample_phase = 1 - sample_amplitude
    optical_thickness = 20 * wavelength
    return sample_amplitude * np.exp(1j * sample_phase*optical_thickness/wavelength)

x_train_digital = convert_images(np.array(x_train_gray))
x_val_digital = convert_images(np.array(x_val_gray))
x_test_digital = convert_images(np.array(x_test_gray))
```

```
def crop_center(sample, cropx, cropy):
    y,x = sample.shape
    startx = x//2-(cropx//2)
    starty = y//2-(cropy//2)
```

```
        return sample[starty:starty+cropy,startx:startx+cropx]

    # Define the spatial coordinates of the sample
    # code here
    num_samples = 256
    starting_coordinate = (-num_samples/2) * delta_x
    ending_coordinate = (num_samples/2 - 1) * delta_x

    # make linspace, meshgrid as needed
    # code here
    x = np.linspace(starting_coordinate, ending_coordinate, num=num_samples)
    y = np.linspace(starting_coordinate, ending_coordinate, num=num_samples)
    xx, yy = np.meshgrid(x, y)

    # define total range of spatial frequency axis, 1/mm
    # code here
    f_range = int(1/delta_x)
    num_samples =  256
    delta_fx = f_range/num_samples

    # make linspace, meshgrid as needed
    # code here
    starting_coordinate = (-num_samples/2) * delta_fx
    ending_coordinate = (num_samples/2 - 1) * delta_fx
    xf = np.linspace(starting_coordinate, ending_coordinate, num=num_samples)
    yf = np.linspace(starting_coordinate, ending_coordinate, num=num_samples)
    xxf, yyf = np.meshgrid(xf, yf)

    # Define lens numerical aperture as percentage of total width of spatial freqeuncy dom
    # Let's make the lens transfer function diameter 1/4th the total spatial frequency axi
    # code here
    p = 0.25
    d =int((ending_coordinate - starting_coordinate+1) * p)
    r = d/2

    # Define lens transfer function as matrix with 1's within desired radius, 0's outside
    # code here
    trans = np.zeros((num_samples, num_samples))
    dist = np.sqrt((xxf)**2+(yyf)**2)
    trans[np.where(dist<r)]=1
    plane_wave_angle_xy = np.array([[0,0], [10,0], [10,10], [0,10], [-10,10], [-10,0], [-1
    def return_illumination_data(sample):
        illumination_data = np.zeros((64, 64, 9))
        # code here
        for i, plane_wave_angle in enumerate(plane_wave_angle_xy):
            # Define plane waves
            # code here
            plane_wave_angle_x = plane_wave_angle[0]
            plane_wave_angle_y = plane_wave_angle[1]
            illumination_plane_wave = np.exp(1j*2*np.pi/wavelength * (np.sin(plane_wave_ar
```

```
        # Define field emerging from sample
        # code here
        emerging_field = np.multiply(illumination_plane_wave, sample)
        #plt.imshow(np.abs(emerging_field))

        # Take 2D fourier transform of sample
        # code here
        fourier_field = np.fft.fftshift(np.fft.fft2(emerging_field))

        # Create filtered sample spectrum with center crop (64 x 64)
        # trans: only within desired radius is 1
        # so we can crop the outer part
        # code here

        filtered_sample = np.multiply(fourier_field, trans)
        centered_filtered_sample = crop_center(filtered_sample, 64, 64)

        # Propagate filtered sample spectrum to image plane
        # code here
        inverse_fourier_field = np.fft.ifft2(np.fft.ifftshift(centered_filtered_sample

        # save the intensity of inverse_fourier_field
        # code here
        detected_field = np.square(np.abs(inverse_fourier_field))
        illumination_data[:,:,i] = detected_field

    return illumination_data


x_train_illumination = np.zeros((len(x_train),64,64,9))
x_val_illumination = np.zeros((len(x_val),64,64,9))
x_test_illumination = np.zeros((len(x_val),64,64,9))
for i in range(len(x_train_illumination)):
    x_train_illumination[i] = return_illumination_data(x_train_digital[i])
for i in range(len(x_val_illumination)):
    x_val_illumination[i] = return_illumination_data(x_val_digital[i])
for i in range(len(x_test_illumination)):
    x_test_illumination[i] = return_illumination_data(x_test_digital[i])


class Illumination(tf.keras.layers.Layer):
    def __init__(self, is_train=False):
        # code here
        super(Illumination, self).__init__()
        self.is_train = is_train

    def build(self, input_shape):
        # initialize illumination weight with 0 mean 0.05 std
        weight_init = tf.random_normal_initializer(0,0.05)
        # code here
        self.illumination = tf.keras.layers.Conv2D(filters = 1,kernel_size = (1,1), us
```

```
    def call(self, inputs):
        # code here
        output = self.illumination(inputs)
        return output


image_size =(64, 64, 9)
illum_model = tf.keras.models.Sequential([tf.keras.layers.Input(image_size),
                                           Illumination(is_train = True),
                                           tf.keras.layers.Conv2D(16, 3, padding="same
                                           tf.keras.layers.Conv2D(16, 3, 2, padding="s
                                          #  tf.keras.layers.GaussianNoise(0.2),
                                           tf.keras.layers.BatchNormalization(),
                                           tf.keras.layers.MaxPool2D((2, 2), strides=2
                                           tf.keras.layers.Conv2D(16, 3, padding="same
                                           tf.keras.layers.Conv2D(16, 3, 2, padding="s
                                           tf.keras.layers.BatchNormalization(),
                                           tf.keras.layers.MaxPool2D((2, 2), strides=2
                                           tf.keras.layers.Flatten(),
                                           tf.keras.layers.Dense(128, activation="relu
                                           tf.keras.layers.GaussianDropout(0.2),
                                           tf.keras.layers.Dense(2, activation="softma

illum_model.summary()
```

```
    Model: "sequential_1"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #   
    =================================================================
     illumination_1 (Illuminatio  (None, 64, 64, 1)        9         
     n)                                                              
                                                                     
     conv2d_4 (Conv2D)           (None, 64, 64, 16)        160       
                                                                     
     conv2d_5 (Conv2D)           (None, 32, 32, 16)        2320      
                                                                     
     batch_normalization_2 (Batc  (None, 32, 32, 16)       64        
     hNormalization)                                                 
                                                                     
     max_pooling2d_2 (MaxPooling  (None, 16, 16, 16)       0         
     2D)                                                             
                                                                     
     conv2d_6 (Conv2D)           (None, 16, 16, 16)        2320      
                                                                     
     conv2d_7 (Conv2D)           (None, 8, 8, 16)          2320      
                                                                     
     batch_normalization_3 (Batc  (None, 8, 8, 16)         64        
     hNormalization)                                                 
                                                                     
     max_pooling2d_3 (MaxPooling  (None, 4, 4, 16)         0         
     2D)                                                             
                                                                     
     flatten_1 (Flatten)         (None, 256)               0         
```

```
 dense_2 (Dense)             (None, 128)               32896

 gaussian_dropout_1 (Gaussia  (None, 128)              0
 nDropout)

 dense_3 (Dense)             (None, 2)                 258

=================================================================
Total params: 40,411
Trainable params: 40,347
Non-trainable params: 64
_________________________________________________________________
```

```
lr = 1e-5
opt = tf.optimizers.Adam(learning_rate=lr)
illum_model.compile(optimizer=opt , loss='categorical_crossentropy', metrics=['accurac
```

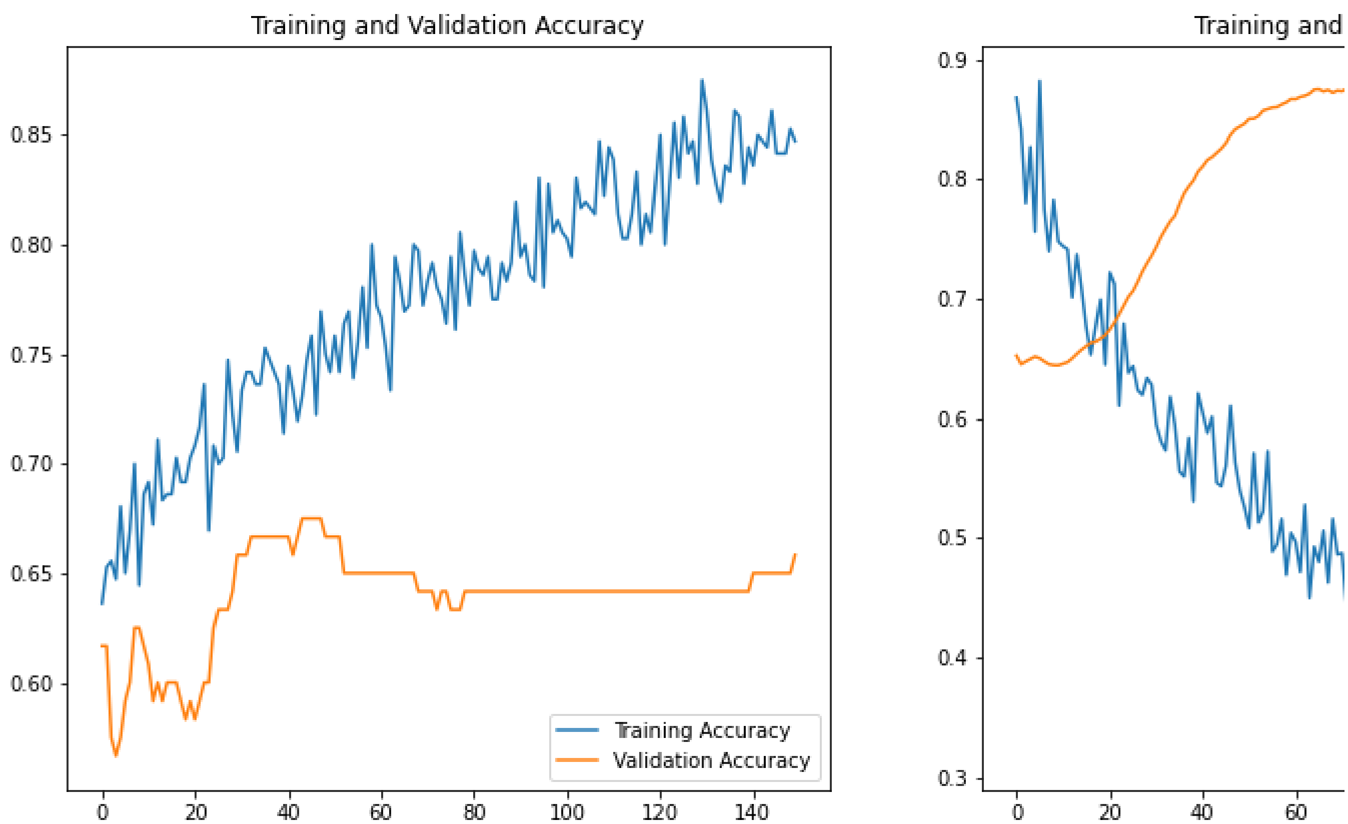
```
illum_history = illum_model.fit(x_train_illumination,y_train, epochs=150, verbose = 1,
                    validation_data=(x_val_illumination, y_val))
```

```
Epoch 1/150
12/12 [==============================] - 1s 42ms/step - loss: 0.8679 - accura
Epoch 2/150
12/12 [==============================] - 0s 21ms/step - loss: 0.8414 - accura
Epoch 3/150
12/12 [==============================] - 0s 18ms/step - loss: 0.7799 - accura
Epoch 4/150
12/12 [==============================] - 0s 18ms/step - loss: 0.8269 - accura
Epoch 5/150
12/12 [==============================] - 0s 18ms/step - loss: 0.7561 - accura
Epoch 6/150
12/12 [==============================] - 0s 21ms/step - loss: 0.8820 - accura
Epoch 7/150
12/12 [==============================] - 0s 21ms/step - loss: 0.7748 - accura
Epoch 8/150
12/12 [==============================] - 0s 18ms/step - loss: 0.7398 - accura
Epoch 9/150
12/12 [==============================] - 0s 18ms/step - loss: 0.7828 - accura
Epoch 10/150
12/12 [==============================] - 0s 17ms/step - loss: 0.7479 - accura
Epoch 11/150
12/12 [==============================] - 0s 19ms/step - loss: 0.7441 - accura
Epoch 12/150
12/12 [==============================] - 0s 18ms/step - loss: 0.7415 - accura
Epoch 13/150
12/12 [==============================] - 0s 18ms/step - loss: 0.7011 - accura
Epoch 14/150
12/12 [==============================] - 0s 21ms/step - loss: 0.7371 - accura
Epoch 15/150
12/12 [==============================] - 0s 17ms/step - loss: 0.7091 - accura
Epoch 16/150
12/12 [==============================] - 0s 21ms/step - loss: 0.6768 - accura
Epoch 17/150
12/12 [==============================] - 0s 21ms/step - loss: 0.6532 - accura
```

```
Epoch 18/150
12/12 [==============================] - 0s 17ms/step - loss: 0.6760 - accura
Epoch 19/150
12/12 [==============================] - 0s 20ms/step - loss: 0.6993 - accura
Epoch 20/150
12/12 [==============================] - 0s 17ms/step - loss: 0.6448 - accura
Epoch 21/150
12/12 [==============================] - 0s 21ms/step - loss: 0.7220 - accura
Epoch 22/150
12/12 [==============================] - 0s 18ms/step - loss: 0.7123 - accura
Epoch 23/150
12/12 [==============================] - 0s 18ms/step - loss: 0.6105 - accura
Epoch 24/150
12/12 [==============================] - 0s 18ms/step - loss: 0.6792 - accura
Epoch 25/150
12/12 [==============================] - 0s 21ms/step - loss: 0.6380 - accura
Epoch 26/150
12/12 [==============================] - 0s 19ms/step - loss: 0.6437 - accura
Epoch 27/150
12/12 [==============================] - 0s 17ms/step - loss: 0.6235 - accura
Epoch 28/150
12/12 [==============================] - 0s 18ms/step - loss: 0.6196 - accura
```

```
# save model
# serialize model to JSON
#illum_json = illum_model.to_json()

#with open("illum.json", "w") as json_file:
#    json_file.write(illum_json)

# serialize weights to HDF5
illum_model.save_weights("illum.h5")
print("Saved model to disk")


# plot train and validation loss
acc = illum_history.history['accuracy']
val_acc = illum_history.history['val_accuracy']
loss = illum_history.history['loss']
val_loss = illum_history.history['val_loss']

plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
```

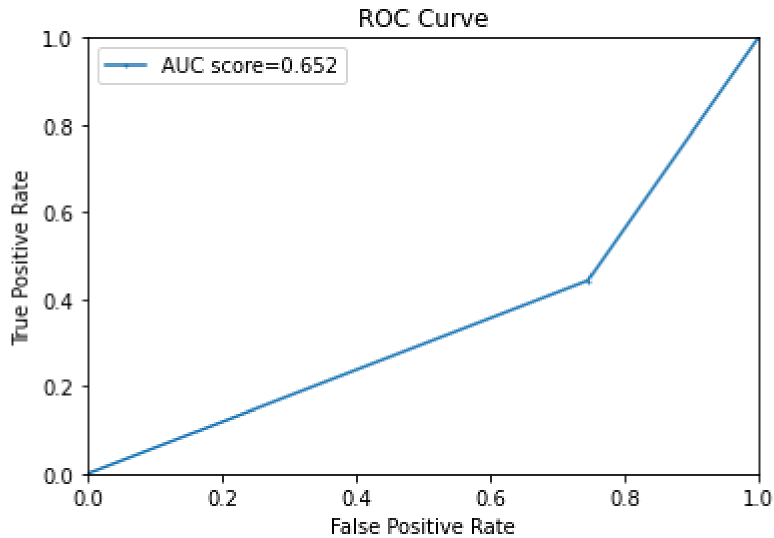
```
plt.title('Training and Validation Loss')
plt.show()
```

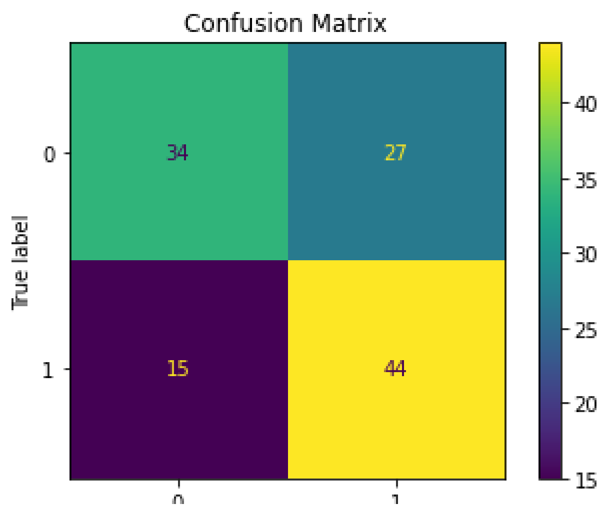



```
# Test set accuracy
y_pred = np.argmax(illum_model.predict(x_test_illumination), axis=1)
acc = illum_model.evaluate(x_test_illumination, y_test)
```

```
4/4 [==============================] - 0s 8ms/step - loss: 0.7780 - accuracy: 0.
```

```
y_labels = np.argmax(y_test.astype(int),axis=1)
roc_plot(y_labels, y_pred)
confusionmatrix(y_labels, y_pred)
```






```
del illum_model
K.clear_session()
```

```
                          precision    recall  f1-score   support
```

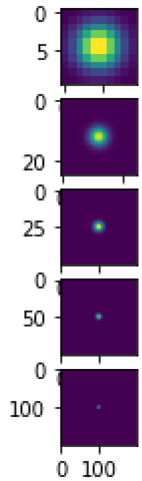
Gaussian Kernel Physical Layer

```
    malignant (class 1)       0.62      0.75      0.68        59
```

```
# create gaussian kernel
def gkern(l=7, sig=2):
  ax = np.linspace(-(l - 1) / 2., (l - 1) / 2., l)
  gauss = np.exp(-0.5 * np.square(ax) / np.square(sig))
  kernel = np.outer(gauss, gauss)
  return kernel / np.sum(kernel)


kernels = [gkern(l=k_size, sig = 2)*k_size/2 for k_size in [10, 25, 50, 100, 200]]


plt.figure()

#subplot(r,c) provide the no. of rows and columns
f, axarr = plt.subplots(5,1)
```

```
# use the created array to output your multiple images. In this case I have stacked 4
for i in range(len(kernels)):
  axarr[i].imshow(kernels[i])
```

<Figure size 432x288 with 0 Axes>




```
# this operation will be done of each one of the five kernels
from scipy.signal import convolve2d
x_train_blurred = [convolve2d(p1_gray, kernels[0], boundary='symm', mode = 'same') for
x_train_blurred = np.array(x_train_blurred)
x_val_blurred = [convolve2d(p1_gray, kernels[0], boundary='symm', mode = 'same') for p
x_val_blurred = np.array(x_val_blurred)
## for other kernels, simply change kernels[0] to kernels[i] where i is from 1 to 4, a
```

```
image_size = (256, 256, 1)
blur_model = tf.keras.models.Sequential([tf.keras.layers.Input(image_size),
                                          tf.keras.layers.Conv2D(16, 3, padding="same
                                          tf.keras.layers.Conv2D(16, 3, 2, padding="s
                                        #  tf.keras.layers.GaussianNoise(0.2),
                                          tf.keras.layers.BatchNormalization(),
                                          tf.keras.layers.MaxPool2D((2, 2), strides=2
                                          tf.keras.layers.Conv2D(16, 3, padding="same
                                          tf.keras.layers.Conv2D(16, 3, 2, padding="s
                                          tf.keras.layers.BatchNormalization(),
                                          tf.keras.layers.MaxPool2D((2, 2), strides=2
                                          tf.keras.layers.Flatten(),
                                          tf.keras.layers.Dense(128, activation="relu
                                          tf.keras.layers.GaussianDropout(0.2),
                                          tf.keras.layers.Dense(2, activation="softma
```

```
blur_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 conv2d (Conv2D)             (None, 256, 256, 16)      160

 conv2d_1 (Conv2D)           (None, 128, 128, 16)      2320

 batch_normalization (BatchN  (None, 128, 128, 16)     64
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 64, 64, 16)       0
 )

 conv2d_2 (Conv2D)           (None, 64, 64, 16)        2320

 conv2d_3 (Conv2D)           (None, 32, 32, 16)        2320

 batch_normalization_1 (Batc  (None, 32, 32, 16)       64
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 16)       0
 2D)

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 128)               524416

 gaussian_dropout (GaussianD  (None, 128)              0
 ropout)

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 531,922
Trainable params: 531,858
Non-trainable params: 64
_________________________________________________________________
```

```
x_train_blurred= np.expand_dims(x_train_blurred, axis=-1)
```

```
x_val_blurred = np.expand_dims(x_val_blurred, axis=-1)
```

```
blur_model.compile(optimizer=opt , loss='categorical_crossentropy', metrics=['accuracy
```

```
blur_history = blur_model.fit(x_train_blurred,y_train, epochs=75, verbose = 1, validat
```

```
Epoch 1/75
12/12 [==============================] - 3s 159ms/step - loss: 0.9462 - accur
Epoch 2/75
12/12 [==============================] - 1s 108ms/step - loss: 0.6872 - accur
Epoch 3/75
12/12 [==============================] - 1s 107ms/step - loss: 0.5732 - accur
Epoch 4/75
12/12 [==============================] - 1s 106ms/step - loss: 0.5062 - accur
Epoch 5/75
12/12 [==============================] - 1s 107ms/step - loss: 0.4577 - accur
```

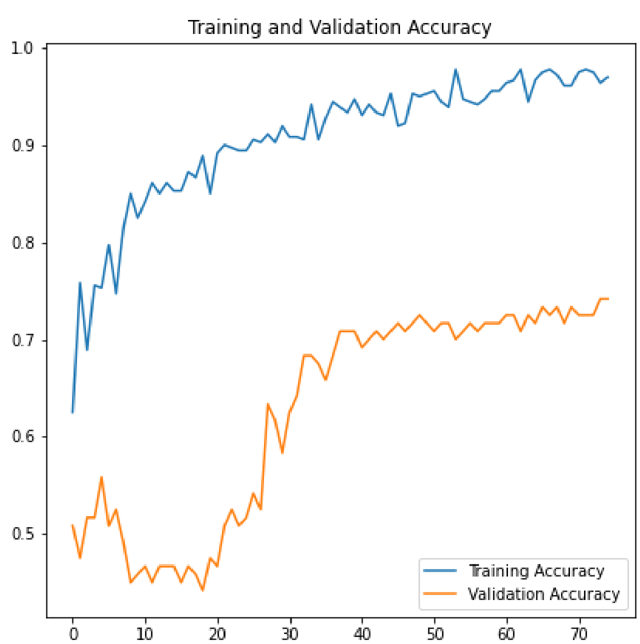
```
Epoch 6/75
12/12 [==============================] - 1s 108ms/step - loss: 0.3759 - accur
Epoch 7/75
12/12 [==============================] - 1s 108ms/step - loss: 0.4558 - accur
Epoch 8/75
12/12 [==============================] - 1s 107ms/step - loss: 0.3939 - accur
Epoch 9/75
12/12 [==============================] - 1s 108ms/step - loss: 0.3553 - accur
Epoch 10/75
12/12 [==============================] - 1s 107ms/step - loss: 0.3510 - accur
Epoch 11/75
12/12 [==============================] - 1s 106ms/step - loss: 0.3444 - accur
Epoch 12/75
12/12 [==============================] - 1s 108ms/step - loss: 0.3146 - accur
Epoch 13/75
12/12 [==============================] - 1s 108ms/step - loss: 0.3123 - accur
Epoch 14/75
12/12 [==============================] - 1s 108ms/step - loss: 0.3152 - accur
Epoch 15/75
12/12 [==============================] - 1s 108ms/step - loss: 0.3093 - accur
Epoch 16/75
12/12 [==============================] - 1s 120ms/step - loss: 0.3293 - accur
Epoch 17/75
12/12 [==============================] - 2s 127ms/step - loss: 0.2877 - accur
Epoch 18/75
12/12 [==============================] - 1s 121ms/step - loss: 0.2983 - accur
Epoch 19/75
12/12 [==============================] - 1s 125ms/step - loss: 0.2743 - accur
Epoch 20/75
12/12 [==============================] - 1s 116ms/step - loss: 0.2952 - accur
Epoch 21/75
12/12 [==============================] - 1s 107ms/step - loss: 0.2553 - accur
Epoch 22/75
12/12 [==============================] - 1s 108ms/step - loss: 0.2556 - accur
Epoch 23/75
12/12 [==============================] - 1s 107ms/step - loss: 0.2576 - accur
Epoch 24/75
12/12 [==============================] - 1s 108ms/step - loss: 0.2348 - accur
Epoch 25/75
12/12 [==============================] - 1s 108ms/step - loss: 0.2450 - accur
Epoch 26/75
12/12 [==============================] - 1s 109ms/step - loss: 0.2520 - accur
Epoch 27/75
12/12 [==============================] - 1s 108ms/step - loss: 0.2378 - accur
Epoch 28/75
12/12 [==============================] - 1s 107ms/step - loss: 0.2260 - accur
```

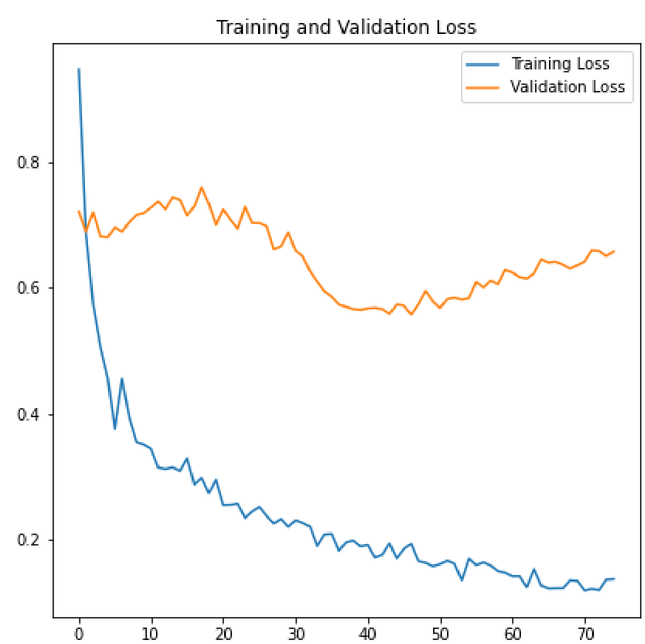
```
# save model
# serialize model to JSON
blur_json = blur_model.to_json()

with open("blur.json", "w") as json_file:
    json_file.write(blur_json)
```

```
# serialize weights to HDF5
blur_model.save_weights("blur.h5")
print("Saved model to disk")
```

```
    Saved model to disk
```

```
# plot train and validation loss
acc = blur_history.history['accuracy']
val_acc = blur_history.history['val_accuracy']
loss = blur_history.history['loss']
val_loss = blur_history.history['val_loss']

plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

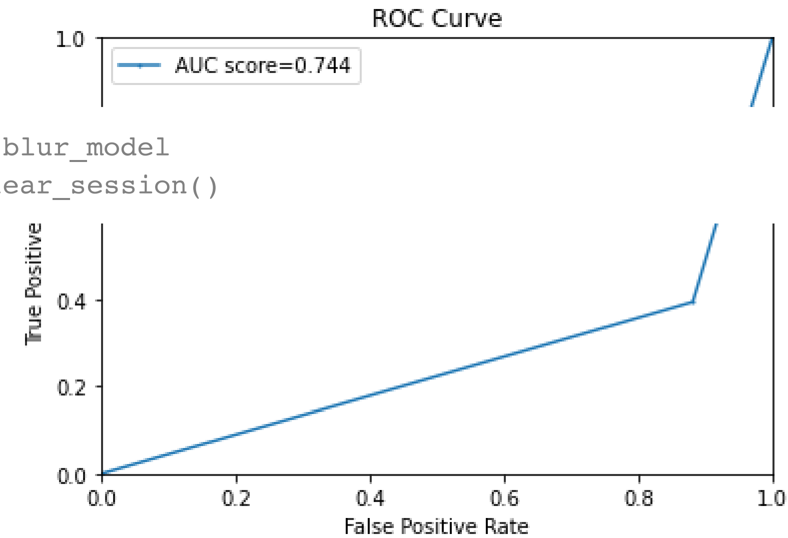




```
x_test_blurred = [convolve2d(p1_gray, kernels[0], boundary='symm', mode = 'same') for
x_test_blurred = np.array(x_test_blurred)
x_test_blurred = np.expand_dims(x_test_blurred, axis=-1)


y_pred = np.argmax(blur_model.predict(x_test_blurred), axis=1)
acc = blur_model.evaluate(x_test_blurred, y_test)
```

```
4/4 [==============================] - 0s 32ms/step - loss: 0.7507 - accuracy: 0
```

```
roc_plot(y_labels, y_pred)
confusionmatrix(y_labels, y_pred)
```




```
del blur_model
K.clear_session()
```