Import all the Dependencies

In [50]:
```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from IPython.display import HTML
```

Set all the Constants

In [51]:
```
BATCH_SIZE = 32
IMAGE_SIZE = 224
CHANNELS = 3
EPOCHS = 20
```

Import data into tensorflow dataset object

In [52]:
```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "../Dataset/CancerDetection",
    seed = 123,
    shuffle = True,
    image_size = (IMAGE_SIZE,IMAGE_SIZE),
    batch_size = BATCH_SIZE
)
```

```
Found 3297 files belonging to 2 classes.
```

In [53]:
```
class_names = dataset.class_names
class_names
```

Out[53]:
```
['benign', 'malignant']
```

In [54]:
```
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())
```

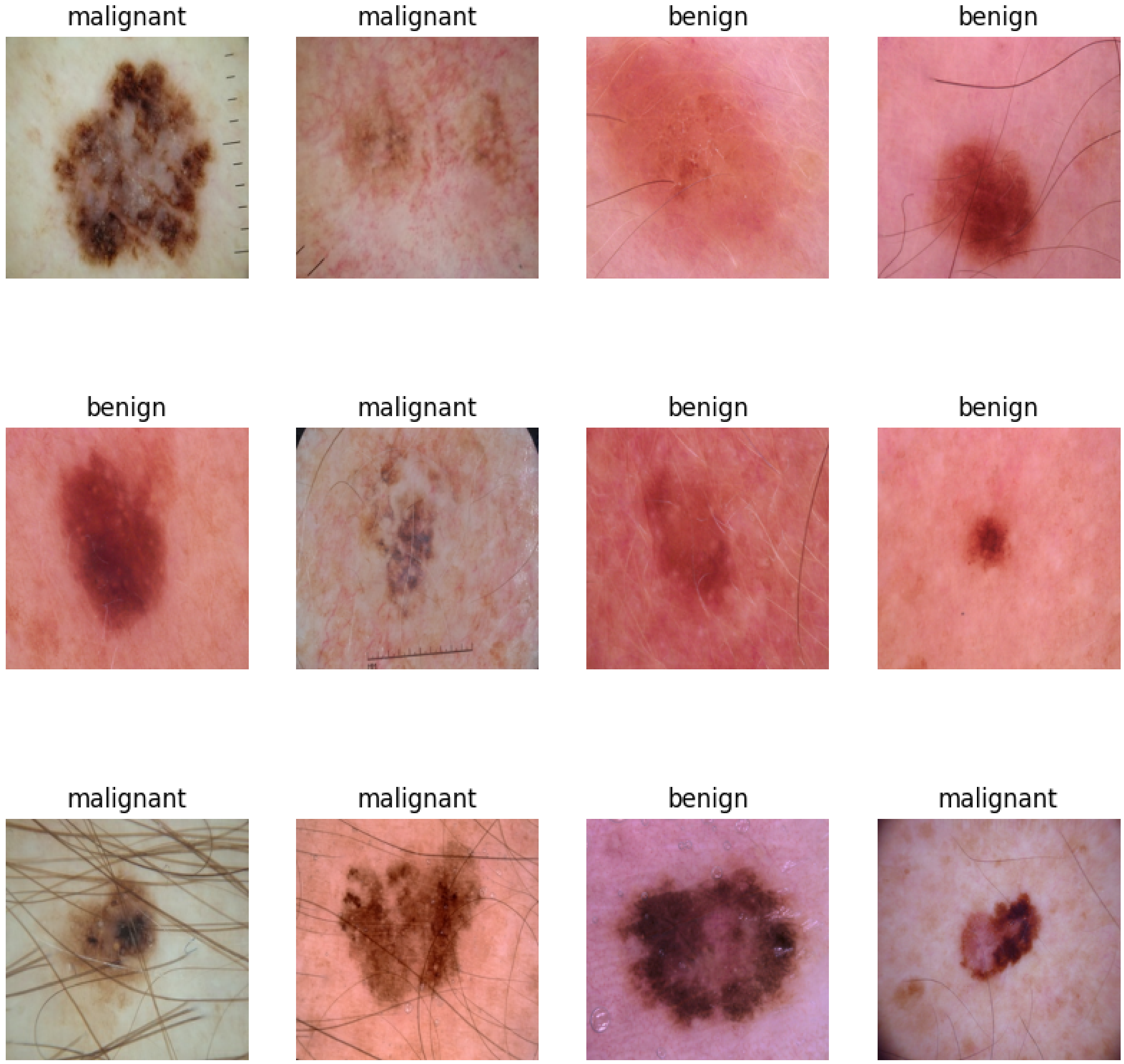
```
(32, 224, 224, 3)
[1 0 1 0 1 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 1 1 1 0 1 0 1 0 0 1 1 0]
```

Visualize some of the images from our dataset

In [55]:
```
plt.figure(figsize=(10, 10))

for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
```

```
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```



Function to Split Dataset

Dataset should be bifurcated into 3 subsets, namely:

1. Training: Dataset to be used while training
2. Validation: Dataset to be tested against while training
3. Test: Dataset to be tested against after we trained a model

In [56]:
```
len(dataset)
```

Out[56]: 104

In [57]:
```
train_size = 0.8
len(dataset)*train_size
```
Out[57]: 83.2

In [58]:
```
train_ds = dataset.take(54)
len(train_ds)
```
Out[58]: 54

In [59]:
```
test_ds = dataset.skip(54)
len(test_ds)
```
Out[59]: 50

In [60]:
```
val_size=0.1
len(dataset)*val_size
```
Out[60]: 10.4

In [61]:
```
val_ds = test_ds.take(6)
len(val_ds)
```
Out[61]: 6

In [62]:
```
test_ds = test_ds.skip(6)
len(test_ds)
```
Out[62]: 44

In [63]:
```
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, s
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
```
In [64]:
```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```
In [65]:
```
len(train_ds)
```
Out[65]: 83

In [66]:
```
len(val_ds)
```

Out[66]: 10

In [67]:
```
len(test_ds)
```

Out[67]: 11

In [68]:
```
actual_label_test = []

for image_batch, labels_batch in test_ds:
    temp = labels_batch.numpy()
    for j in temp:
        actual_label_test.append(j)

# print(len(actual_label_test))
# print(actual_label_test)
```

Cache, Shuffle, and Prefetch the Dataset

In [69]:
```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Building the Model

Creating a Layer for Resizing and Normalization

Before we feed our images to network, we should be resizing it to the desired size. Moreover, to improve model performance, we should normalize the image pixel value (keeping them in range 0 and 1 by dividing by 256). This should happen while training as well as inference. Hence we can add that as a layer in our Sequential Model.

You might be thinking why do we need to resize (256,256) image to again (256,256). You are right we don't need to but this will be useful when we are done with the training and start using the model for predictions. At that time somone can supply an image that is not (256,256) and this layer will resize it

In [70]:
```
resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.experimental.preprocessing.Rescaling(1./255),
])
```

In [71]:
```
# Data Augmentation
# Data Augmentation is needed when we have less data, this boosts the accuracy of o
```

```
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])

# Applying Data Augmentation to Train Dataset
train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Model Architecture

We use a CNN coupled with a Softmax activation in the output layer. We also add the initial layers for resizing, normalization and Data Augmentation.

In [72]:

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 2

model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_sha
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])

model.build(input_shape=input_shape)
```

In [73]:

```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 sequential_3 (Sequential)   (32, 224, 224, 3)         0         
                                                                 
 conv2d_6 (Conv2D)           (32, 222, 222, 32)        896       
                                                                 
 max_pooling2d_6 (MaxPooling  (32, 111, 111, 32)       0         
 2D)                                                             
                                                                 
 conv2d_7 (Conv2D)           (32, 109, 109, 64)        18496     
                                                                 
 max_pooling2d_7 (MaxPooling  (32, 54, 54, 64)         0         
 2D)                                                             
                                                                 
 conv2d_8 (Conv2D)           (32, 52, 52, 64)          36928     
                                                                 
 max_pooling2d_8 (MaxPooling  (32, 26, 26, 64)         0         
 2D)                                                             
                                                                 
 conv2d_9 (Conv2D)           (32, 24, 24, 64)          36928     
                                                                 
 max_pooling2d_9 (MaxPooling  (32, 12, 12, 64)         0         
 2D)                                                             
                                                                 
 conv2d_10 (Conv2D)          (32, 10, 10, 64)          36928     
                                                                 
 max_pooling2d_10 (MaxPoolin  (32, 5, 5, 64)           0         
 g2D)                                                            
                                                                 
 conv2d_11 (Conv2D)          (32, 3, 3, 64)            36928     
                                                                 
 max_pooling2d_11 (MaxPoolin  (32, 1, 1, 64)           0         
 g2D)                                                            
                                                                 
 flatten_1 (Flatten)         (32, 64)                  0         
                                                                 
 dense_2 (Dense)             (32, 64)                  4160      
                                                                 
 dense_3 (Dense)             (32, 2)                   130       
                                                                 
=================================================================
Total params: 171,394
Trainable params: 171,394
Non-trainable params: 0
_________________________________________________________________
```

Compiling the Model

We use adam Optimizer, SparseCategoricalCrossentropy for losses, accuracy as a metric

In [74]:
```
import time
t0 = time.time()
```

In [75]:
```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

In [76]:
```
history = model.fit(
    train_ds,
    batch_size = BATCH_SIZE,
    validation_data = val_ds,
    verbose = 1,
    epochs = EPOCHS,
)
```

```
Epoch 1/20
83/83 [==============================] - 72s 837ms/step - loss: 0.5791 - accuracy:
0.6875 - val_loss: 0.4907 - val_accuracy: 0.7875
Epoch 2/20
83/83 [==============================] - 68s 821ms/step - loss: 0.4904 - accuracy:
0.7572 - val_loss: 0.5020 - val_accuracy: 0.7812
Epoch 3/20
83/83 [==============================] - 70s 845ms/step - loss: 0.4919 - accuracy:
0.7575 - val_loss: 0.4661 - val_accuracy: 0.7969
Epoch 4/20
83/83 [==============================] - 68s 822ms/step - loss: 0.4589 - accuracy:
0.7700 - val_loss: 0.4556 - val_accuracy: 0.7969
Epoch 5/20
83/83 [==============================] - 69s 825ms/step - loss: 0.4257 - accuracy:
0.7782 - val_loss: 0.4136 - val_accuracy: 0.8125
Epoch 6/20
83/83 [==============================] - 77s 922ms/step - loss: 0.4047 - accuracy:
0.7910 - val_loss: 0.4414 - val_accuracy: 0.7688
Epoch 7/20
83/83 [==============================] - 68s 815ms/step - loss: 0.4029 - accuracy:
0.7933 - val_loss: 0.4321 - val_accuracy: 0.8062
Epoch 8/20
83/83 [==============================] - 67s 810ms/step - loss: 0.4524 - accuracy:
0.7779 - val_loss: 0.4128 - val_accuracy: 0.8094
Epoch 9/20
83/83 [==============================] - 68s 815ms/step - loss: 0.3944 - accuracy:
0.7944 - val_loss: 0.4291 - val_accuracy: 0.7875
Epoch 10/20
83/83 [==============================] - 78s 939ms/step - loss: 0.3892 - accuracy:
0.8035 - val_loss: 0.4431 - val_accuracy: 0.7937
Epoch 11/20
83/83 [==============================] - 68s 814ms/step - loss: 0.3803 - accuracy:
0.8185 - val_loss: 0.4018 - val_accuracy: 0.8156
Epoch 12/20
83/83 [==============================] - 67s 809ms/step - loss: 0.3686 - accuracy:
0.8215 - val_loss: 0.3576 - val_accuracy: 0.8438
Epoch 13/20
83/83 [==============================] - 67s 811ms/step - loss: 0.3769 - accuracy:
0.8268 - val_loss: 0.4518 - val_accuracy: 0.7937
Epoch 14/20
83/83 [==============================] - 70s 846ms/step - loss: 0.3753 - accuracy:
0.8174 - val_loss: 0.3973 - val_accuracy: 0.7969
Epoch 15/20
83/83 [==============================] - 81s 973ms/step - loss: 0.3584 - accuracy:
0.8298 - val_loss: 0.3608 - val_accuracy: 0.8406
Epoch 16/20
83/83 [==============================] - 68s 819ms/step - loss: 0.3642 - accuracy:
0.8163 - val_loss: 0.3929 - val_accuracy: 0.8313
Epoch 17/20
83/83 [==============================] - 70s 847ms/step - loss: 0.3629 - accuracy:
0.8234 - val_loss: 0.3591 - val_accuracy: 0.8188
Epoch 18/20
83/83 [==============================] - 70s 843ms/step - loss: 0.3606 - accuracy:
0.8257 - val_loss: 0.3499 - val_accuracy: 0.8344
Epoch 19/20
83/83 [==============================] - 70s 845ms/step - loss: 0.3425 - accuracy:
```

```
0.8336 - val_loss: 0.3654 - val_accuracy: 0.8344
Epoch 20/20
83/83 [==============================] - 71s 853ms/step - loss: 0.3471 - accuracy:
0.8370 - val_loss: 0.3699 - val_accuracy: 0.8219
```

In [77]:
```
t1 = time.time()
```

Training Speed

In [78]:
```
print("CNN Model Training time:  ", (t1-t0)/60 , "minutes")
```

```
CNN Model Training time:   23.492891856034596 minutes
```

Evaluation

In [79]:
```
scores = model.evaluate(test_ds)
```

```
11/11 [==============================] - 3s 159ms/step - loss: 0.3616 - accuracy:
0.8210
```

In [80]:
```
scores
```

Out[80]:
```
[0.36160534620285034, 0.8210227489471436]
```

Predictions

In [81]:
```
predicted = model.predict(test_ds)
```

```
11/11 [==============================] - 2s 147ms/step
```

In [82]:
```
import numpy as np

confidence = np.max(predicted, axis=1)
predictions = np.argmax(predicted, axis=1)
```

In [83]:
```
# predicted

# print(predicted)
print(len(predicted))
print(len(test_ds))

# print(predictions)
print(len(predictions))
```

```
352
11
352
```

In [ ]:

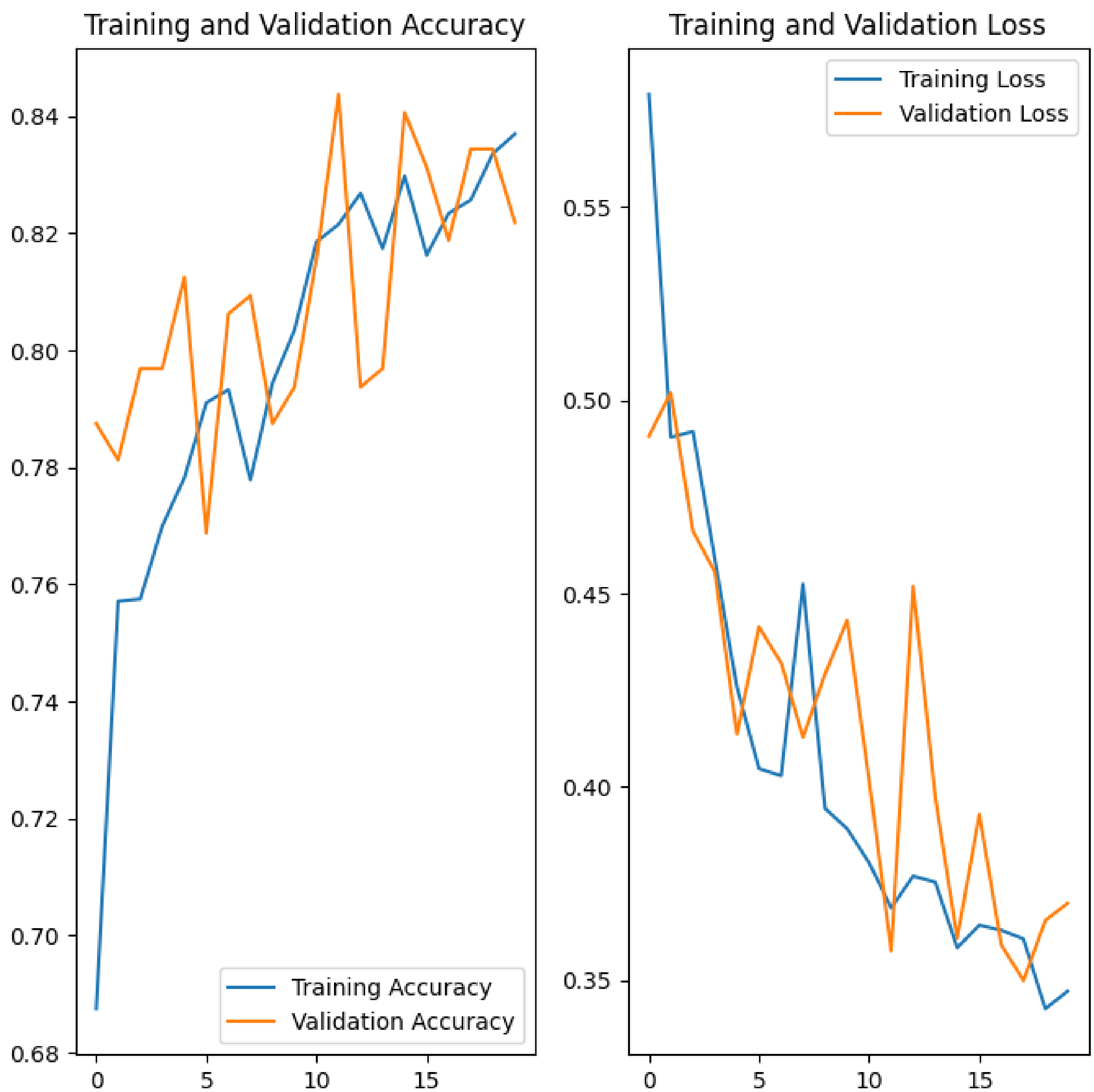
Plotting History

In [84]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
```

In [85]:
```
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss, label='Training Loss')
plt.plot(range(EPOCHS), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



Confusion Matrix

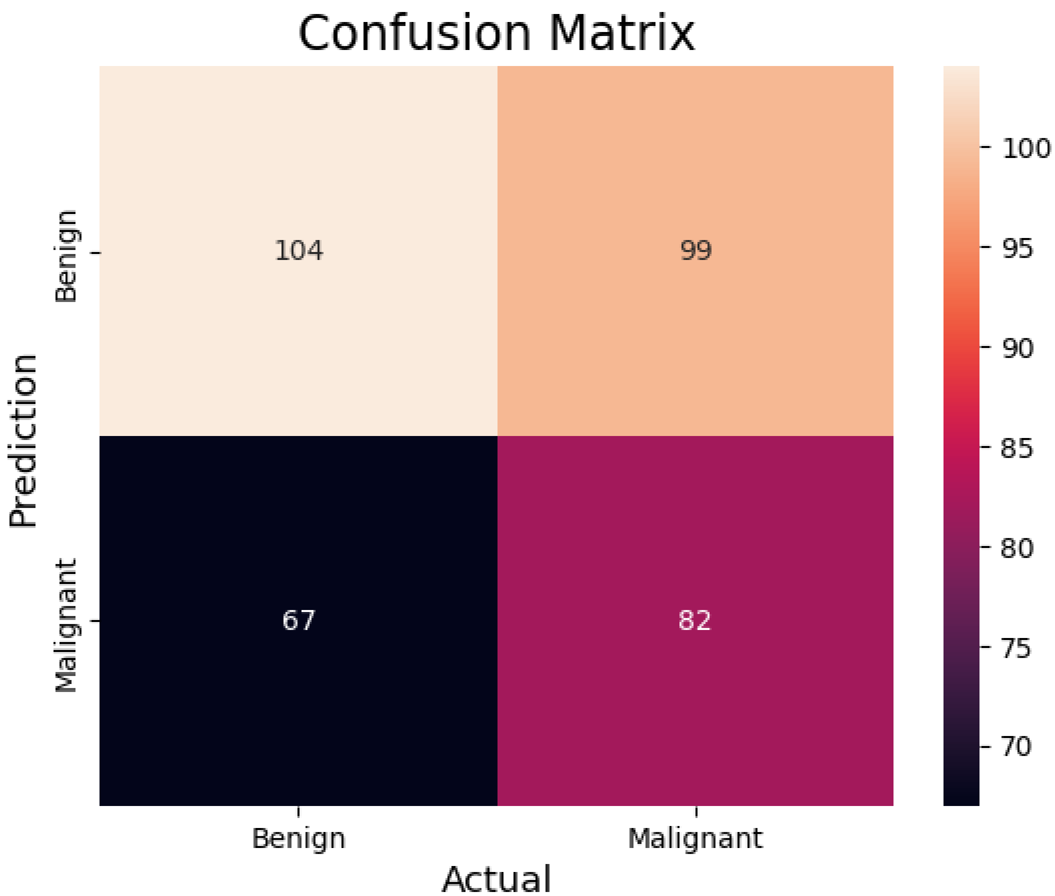
In [86]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

In [87]:
```
cm = confusion_matrix(actual_label_test, predictions)


sns.heatmap(
    cm,
    annot=True,
    fmt='g',
    xticklabels=['Benign','Malignant'],
    yticklabels=['Benign','Malignant']
)

plt.ylabel('Prediction',fontsize=13)
```

```
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()
```



In [88]:
```
from sklearn.metrics import classification_report
print(classification_report(actual_label_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.61      0.51      0.56       203
           1       0.45      0.55      0.50       149

    accuracy                           0.53       352
   macro avg       0.53      0.53      0.53       352
weighted avg       0.54      0.53      0.53       352
```

Saving the Model

We append the model to the list of models as a new version

In [89]:
```
import os
model_version=max([int(i) for i in os.listdir("../savedmodels") + [0]])+1
model.save(f"../savedmodels/{model_version}")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[89], line 2
      1 import os
----> 2 model_version=max([int(i) for i in os.listdir("../savedmodels") + [0]])+1
      3 model.save(f"../savedmodels/{model_version}")

FileNotFoundError: [WinError 3] The system cannot find the path specified: '../sav
edmodels'
```