Import all the Dependencies

```
import numpy as np
import cv2

import PIL.Image as Image

import shutil
import os
from IPython.display import HTML

import matplotlib.pylab as plt

import tensorflow as tf
import tensorflow_hub as hub

from tensorflow import keras
from tensorflow.keras import models, layers
from tensorflow.keras.models import Sequential

import time
```

Downloading Data from link

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

```
!kaggle datasets download -d fanconic/skin-cancer-malignant-vs-benign
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Downloading skin-cancer-malignant-vs-benign.zip to /content
 94% 304M/325M [00:04<00:00, 30.4MB/s]
100% 325M/325M [00:04<00:00, 73.6MB/s]
```

```
import zipfile
zip_ref = zipfile.ZipFile('/content/skin-cancer-malignant-vs-benign.zip', 'r')
zip_ref.extractall('/content')
zip_ref.close()
```

```
curr_dir = os.getcwd()
curr_dir
```

```
'/content'
```

Setting up directories

```
# creating folders

datasetFolder = curr_dir + "/DATASET"
os.makedirs(datasetFolder)
os.makedirs(datasetFolder + "/benign")
os.makedirs(datasetFolder + "/malignant")
```

```
# defining source and destination folders paths

src1 = curr_dir + "/test/benign"
src2 = curr_dir + "/train/benign"

src3 = curr_dir + "/test/malignant"
src4 = curr_dir + "/train/malignant"
```

```
benign_src = [src1, src2]
malignant_src = [src3, src4]
```

```
benign_dest = curr_dir + "/DATASET/benign"
malignant_dest = curr_dir + "/DATASET/malignant"
```

```
## copying files

for src in benign_src:
  for dirs, subdirs, files in os.walk(src):
    print(" Total benign files : ", len(files))
    for file in files:
      if file.endswith('.jpg'):
        filename = os.path.join(src, dirs, file)
        if os.path.exists(filename):
          # print(filename)
          shutil.copy(filename, benign_dest)

for src in malignant_src:
  for dirs, subdirs, files in os.walk(src):
    print(" Total malignant files : ", len(files))
    for file in files:
      if file.endswith('.jpg'):
        filename = os.path.join(src, dirs, file)
        if os.path.exists(filename):
          # print(filename)
          shutil.copy(filename, malignant_dest)
```

```
 Total benign files :  360
 Total benign files :  1440
 Total malignant files :  300
 Total malignant files :  1197
```

```
## deleting old folders

shutil.rmtree(curr_dir + "/data")
shutil.rmtree(curr_dir + "/test")
shutil.rmtree(curr_dir + "/train")
```

```
Total_images = 0

for dirs, subdirs, files in (os.walk(benign_dest)):
  print(f'Benign : {len(files)}')
  Total_images = Total_images + len(files)

for dirs, subdirs, files in (os.walk(malignant_dest)):
  print(f'Malignant : {len(files)}')
  Total_images = Total_images + len(files)

print(f'\nTotal images : {Total_images}')
```

```
Benign : 1800
Malignant : 1497

Total images : 3297
```

Setting Constants

```
BATCH_SIZE = 32
IMAGE_SIZE = 224
CHANNELS = 3
EPOCHS = 50
```

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "./DATASET",
    seed = 123,
    shuffle = True,
    image_size = (IMAGE_SIZE,IMAGE_SIZE),
    batch_size = BATCH_SIZE
)
```

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

Data Visualization

```
class_names = dataset.class_names
class_names
```

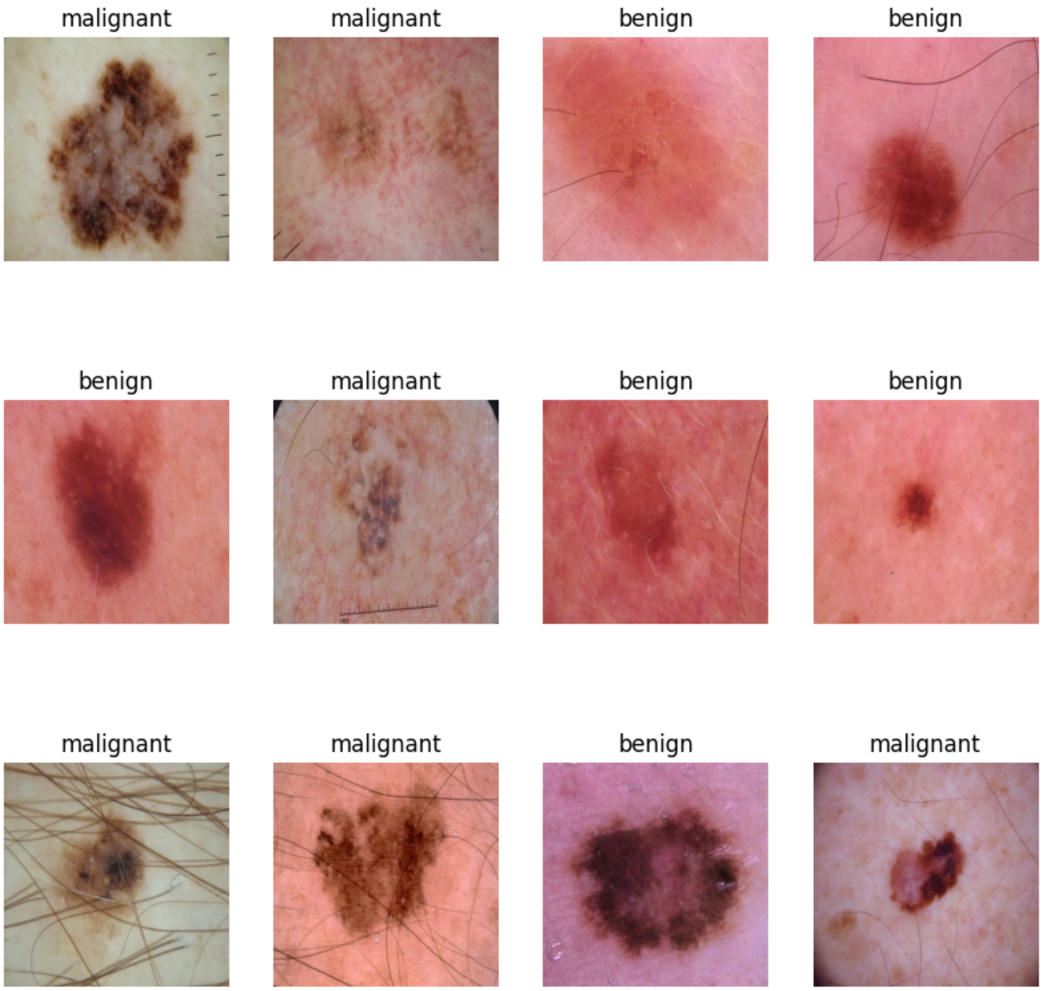
```
['benign', 'malignant']
```

```
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]
```

```
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")
```



```
len(dataset)
```

```
104
```

```
train_size = 0.8
len(dataset)*train_size
```

```
83.2
```

```
train_ds = dataset.take(54)
len(train_ds)
```

```
     54
```

```
test_ds = dataset.skip(54)
len(test_ds)
```

```
     50
```

```
val_size=0.1
len(dataset)*val_size
```

```
     10.4
```

```
val_size=0.1
len(dataset)*val_size
```

```
     10.4
```

```
test_ds = test_ds.skip(6)
len(test_ds)
```

```
     44
```

```
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=10000):
    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
```

```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

```
len(train_ds)
```

```
     83
```

```
len(val_ds)
```

```
     10
```

```
len(test_ds)
```

```
     11
```

```
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)
```

Catching, Prefetching and setting resize rescale layers

```
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)
```

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

Data Augmentation

```
# Data Augmentation
# Data Augmentation is needed when we have less data, this boosts the accuracy of our model by augmenting the data.

# 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 Building

```
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_shape),
    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)
```

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_2 (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
_________________________________________________________________
```

```
import time
t0 = time.time()


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


history = model.fit(
    train_ds,
    batch_size = BATCH_SIZE,
    validation_data = val_ds,
    verbose = 1,
    epochs = EPOCHS,
)
```

```
    83/83 [==============================] - 4s 46ms/step - loss: 0.0606 - accuracy: 0.9800 - val_loss: 0.2816 - val_accuracy: 0.9625
    Epoch 39/50
    83/83 [==============================] - 4s 46ms/step - loss: 0.0594 - accuracy: 0.9774 - val_loss: 0.2707 - val_accuracy: 0.9406
    Epoch 40/50
    83/83 [==============================] - 4s 45ms/step - loss: 0.0959 - accuracy: 0.9605 - val_loss: 0.3883 - val_accuracy: 0.9438
    Epoch 41/50
    83/83 [==============================] - 4s 44ms/step - loss: 0.0763 - accuracy: 0.9688 - val_loss: 0.1718 - val_accuracy: 0.9688
    Epoch 42/50
    83/83 [==============================] - 4s 44ms/step - loss: 0.0575 - accuracy: 0.9808 - val_loss: 0.2479 - val_accuracy: 0.9563
    Epoch 43/50
    83/83 [==============================] - 4s 46ms/step - loss: 0.0221 - accuracy: 0.9917 - val_loss: 0.2163 - val_accuracy: 0.9688
    Epoch 44/50
    83/83 [==============================] - 4s 44ms/step - loss: 0.0602 - accuracy: 0.9782 - val_loss: 0.2967 - val_accuracy: 0.9594
    Epoch 45/50
    83/83 [==============================] - 4s 45ms/step - loss: 0.0211 - accuracy: 0.9940 - val_loss: 0.2518 - val_accuracy: 0.9625
    Epoch 46/50
    83/83 [==============================] - 4s 44ms/step - loss: 0.0062 - accuracy: 0.9985 - val_loss: 0.3219 - val_accuracy: 0.9688
    Epoch 47/50
    83/83 [==============================] - 4s 44ms/step - loss: 0.0231 - accuracy: 0.9936 - val_loss: 0.3485 - val_accuracy: 0.9531
    Epoch 48/50
    83/83 [==============================] - 4s 47ms/step - loss: 0.0495 - accuracy: 0.9831 - val_loss: 0.3193 - val_accuracy: 0.9500
    Epoch 49/50
    83/83 [==============================] - 4s 45ms/step - loss: 0.0311 - accuracy: 0.9906 - val_loss: 0.2742 - val_accuracy: 0.9563
    Epoch 50/50
    83/83 [==============================] - 4s 44ms/step - loss: 0.0286 - accuracy: 0.9902 - val_loss: 0.3232 - val_accuracy: 0.9500
```

Model Analysis

```
t1 = time.time()
```

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

```
    CNN Model Training time:   3.990845835208893 minutes
```

```
scores = model.evaluate(test_ds)
```

```
    11/11 [==============================] - 8s 26ms/step - loss: 0.3968 - accuracy: 0.9432
```

```
scores
```

```
    [0.3968188166618347, 0.9431818127632141]
```

```
predicted = model.predict(test_ds)
```

```
    11/11 [==============================] - 1s 20ms/step
```

```
import numpy as np

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

```
# predicted

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

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

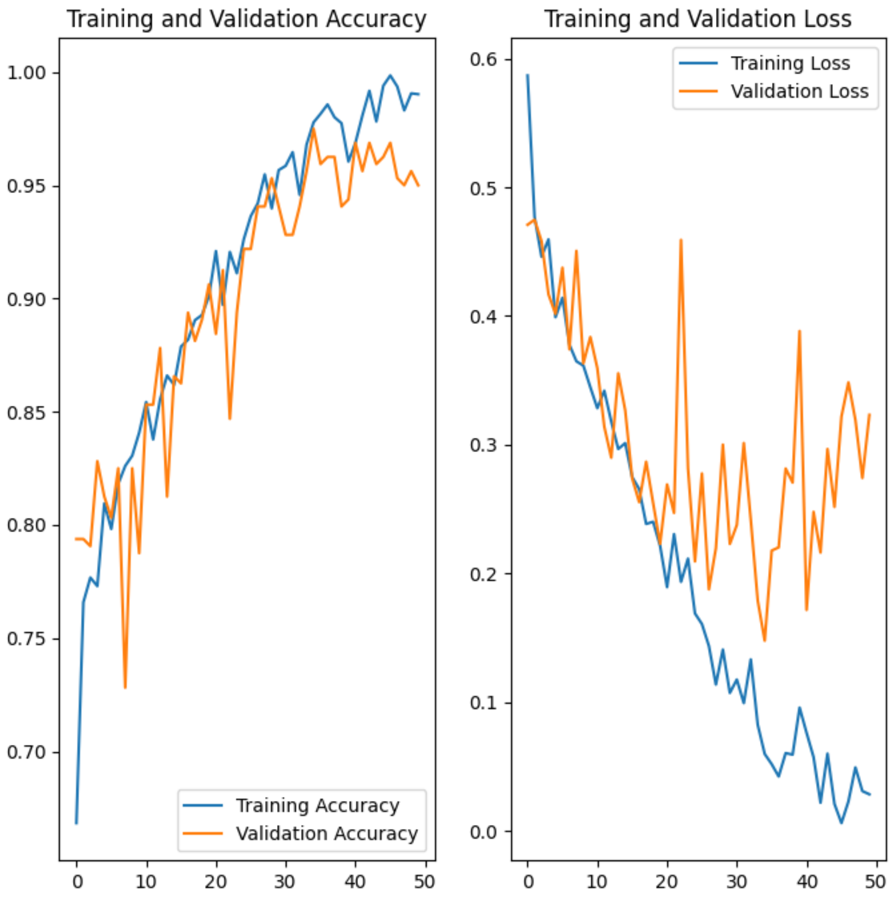
```
    352
    11
    352
```

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

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

```
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()
```

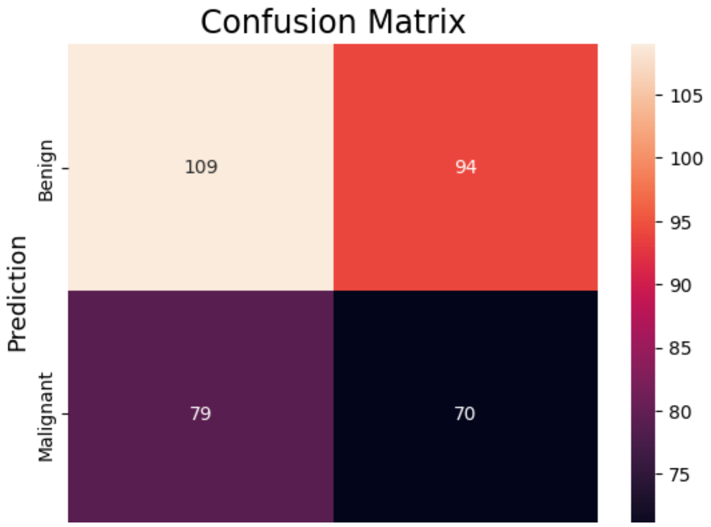


```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt


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()
```



```
from sklearn.metrics import classification_report
print(classification_report(actual_label_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.58      0.54      0.56       203
           1       0.43      0.47      0.45       149

    accuracy                           0.51       352
   macro avg       0.50      0.50      0.50       352
weighted avg       0.52      0.51      0.51       352
```

Saving Model

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

✓ 0s completed at 10:40 AM