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
 99% 321M/325M [00:11<00:00, 32.3MB/s]
100% 325M/325M [00:11<00:00, 29.0MB/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
```

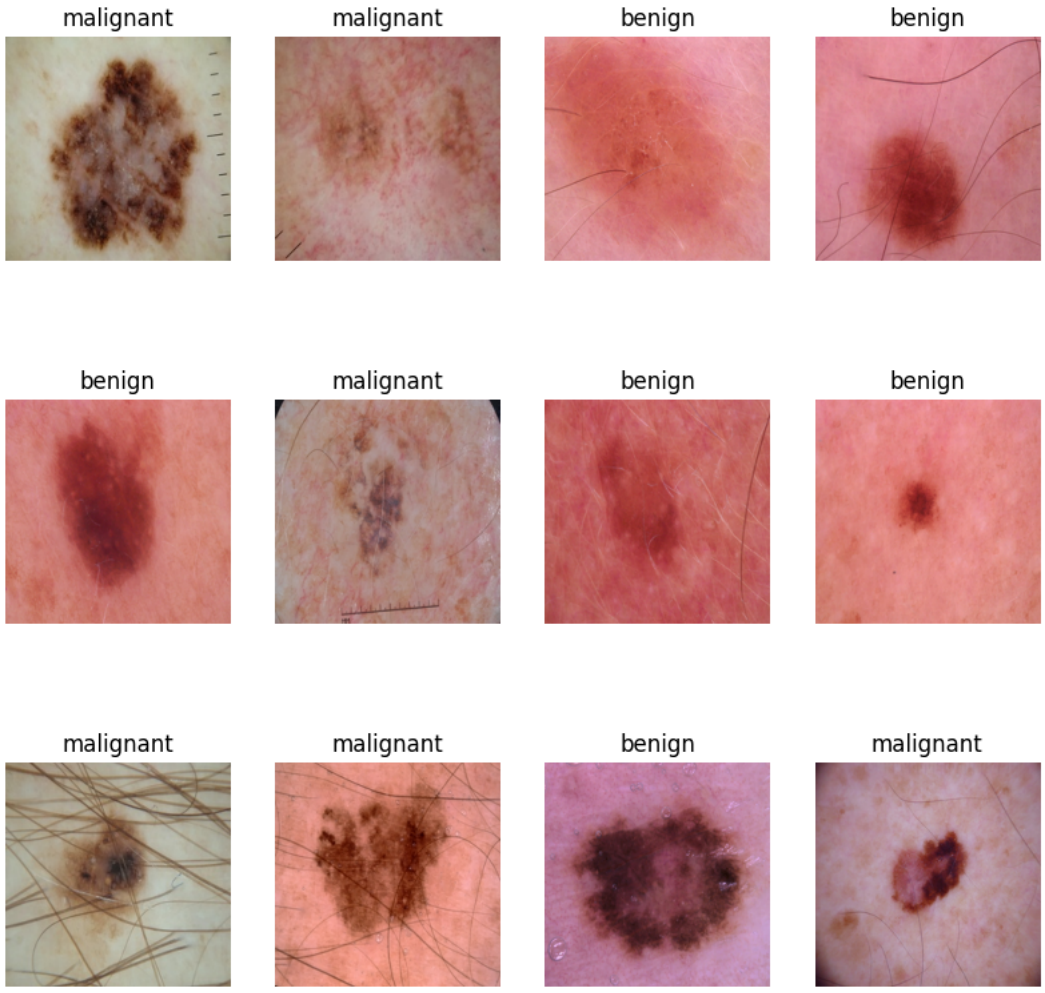
```
['benign', 'malignant']
```

```
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)
```

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_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 224, 224, 3)         0

 conv2d (Conv2D)             (32, 222, 222, 32)        896

 max_pooling2d (MaxPooling2D  (32, 111, 111, 32)       0
 )

 conv2d_1 (Conv2D)           (32, 109, 109, 64)        18496

 max_pooling2d_1 (MaxPooling  (32, 54, 54, 64)         0
 2D)

 conv2d_2 (Conv2D)           (32, 52, 52, 64)          36928

 max_pooling2d_2 (MaxPooling  (32, 26, 26, 64)         0
 2D)
```

```
 conv2d_3 (Conv2D)           (32, 24, 24, 64)          36928

 max_pooling2d_3 (MaxPooling  (32, 12, 12, 64)         0
 2D)

 conv2d_4 (Conv2D)           (32, 10, 10, 64)          36928

 max_pooling2d_4 (MaxPooling  (32, 5, 5, 64)           0
 2D)

 conv2d_5 (Conv2D)           (32, 3, 3, 64)            36928

 max_pooling2d_5 (MaxPooling  (32, 1, 1, 64)           0
 2D)

 flatten (Flatten)           (32, 64)                  0

 dense (Dense)               (32, 64)                  4160

 dense_1 (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,
)
```

```
 Epoch 8/50
 83/83 [==============================] - 4s 44ms/step - loss: 0.3773 - accuracy: 0.8230 - val_loss: 0.4124 - val_accuracy: 0.7656
 Epoch 9/50
 83/83 [==============================] - 4s 43ms/step - loss: 0.3514 - accuracy: 0.8362 - val_loss: 0.3530 - val_accuracy: 0.8250
 Epoch 10/50
 83/83 [==============================] - 4s 46ms/step - loss: 0.3418 - accuracy: 0.8415 - val_loss: 0.3068 - val_accuracy: 0.8500
 Epoch 11/50
 83/83 [==============================] - 4s 43ms/step - loss: 0.3316 - accuracy: 0.8460 - val_loss: 0.3519 - val_accuracy: 0.8250
 Epoch 12/50
 83/83 [==============================] - 4s 43ms/step - loss: 0.3220 - accuracy: 0.8539 - val_loss: 0.3811 - val_accuracy: 0.8406
 Epoch 13/50
 83/83 [==============================] - 4s 46ms/step - loss: 0.3149 - accuracy: 0.8550 - val_loss: 0.2779 - val_accuracy: 0.8750
 Epoch 14/50
 83/83 [==============================] - 4s 43ms/step - loss: 0.2805 - accuracy: 0.8758 - val_loss: 0.3197 - val_accuracy: 0.8500
 Epoch 15/50
 83/83 [==============================] - 4s 43ms/step - loss: 0.3041 - accuracy: 0.8596 - val_loss: 0.2961 - val_accuracy: 0.8687
 Epoch 16/50
 83/83 [==============================] - 4s 45ms/step - loss: 0.2691 - accuracy: 0.8840 - val_loss: 0.2783 - val_accuracy: 0.8750
 Epoch 17/50
 83/83 [==============================] - 4s 44ms/step - loss: 0.2422 - accuracy: 0.8953 - val_loss: 0.2423 - val_accuracy: 0.8875
 Epoch 18/50
 83/83 [==============================] - 4s 44ms/step - loss: 0.2359 - accuracy: 0.9002 - val_loss: 0.2521 - val_accuracy: 0.8813
```

```
Epoch 26/50
83/83 [==============================] - 4s 43ms/step - loss: 0.1292 - accuracy: 0.9526 - val_loss: 0.2560 - val_accuracy: 0.9000
Epoch 27/50
83/83 [==============================] - 4s 44ms/step - loss: 0.1010 - accuracy: 0.9559 - val_loss: 0.2193 - val_accuracy: 0.9438
Epoch 28/50
83/83 [==============================] - 4s 46ms/step - loss: 0.0838 - accuracy: 0.9676 - val_loss: 0.2342 - val_accuracy: 0.9375
Epoch 29/50
83/83 [==============================] - 4s 44ms/step - loss: 0.0694 - accuracy: 0.9763 - val_loss: 0.1824 - val_accuracy: 0.9656
Epoch 30/50
83/83 [==============================] - 4s 44ms/step - loss: 0.0823 - accuracy: 0.9665 - val_loss: 0.1886 - val_accuracy: 0.9531
Epoch 31/50
83/83 [==============================] - 4s 45ms/step - loss: 0.1281 - accuracy: 0.9537 - val_loss: 0.1999 - val_accuracy: 0.9438
Epoch 32/50
83/83 [==============================] - 4s 44ms/step - loss: 0.0532 - accuracy: 0.9812 - val_loss: 0.2141 - val_accuracy: 0.9625
Epoch 33/50
83/83 [==============================] - 4s 44ms/step - loss: 0.0556 - accuracy: 0.9804 - val_loss: 0.3103 - val_accuracy: 0.9500
Epoch 34/50
83/83 [==============================] - 4s 43ms/step - loss: 0.0662 - accuracy: 0.9752 - val_loss: 0.2456 - val_accuracy: 0.9531
Epoch 35/50
83/83 [==============================] - 4s 46ms/step - loss: 0.0300 - accuracy: 0.9898 - val_loss: 0.3837 - val_accuracy: 0.9688
Epoch 36/50
```

Model Analysis

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

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

```
CNN Model Training time:   3.7909056107203165 minutes
```

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

```
11/11 [==============================] - 5s 23ms/step - loss: 0.3990 - accuracy: 0.9631
```

```
scores
```

```
[0.3989540934562683, 0.9630681872367859]
```

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

```
11/11 [==============================] - 0s 16ms/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
```

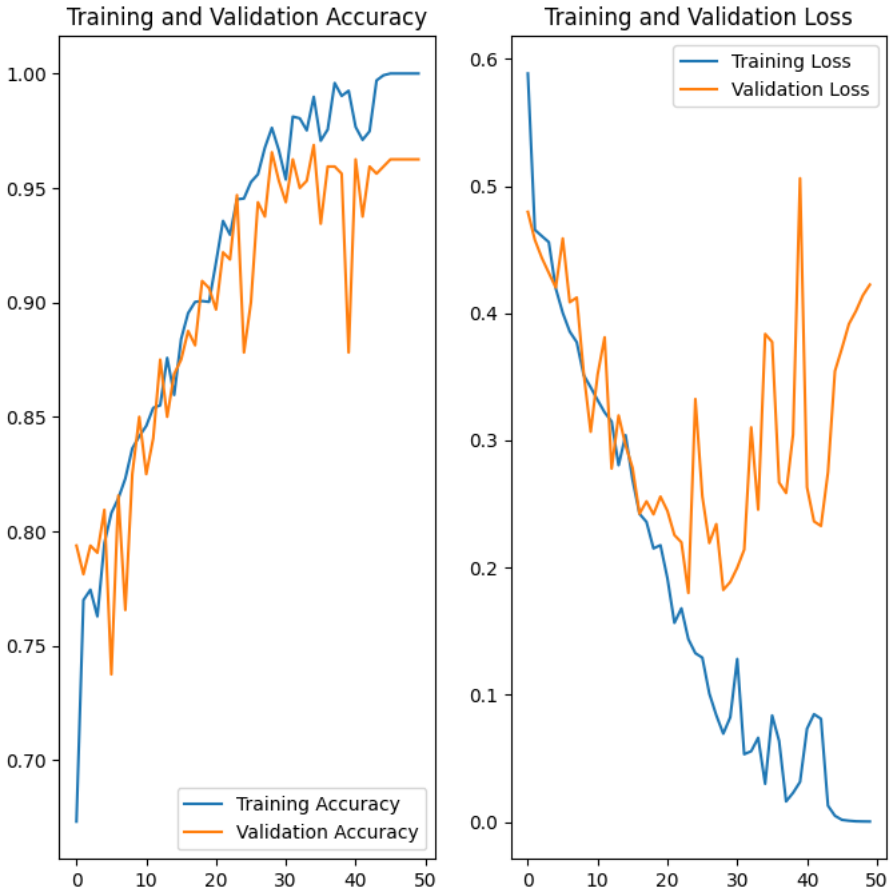
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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 classification_report
# print(classification_report(actual_label_test, predictions))
```

Saving Model

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