Author : LAW MAN FAI

Problem statement: To build a CNN based model which can accurately detect melanoma. Melanoma is a type of cancer that can be deadly if not detected early. It accounts for 75% of skin cancer deaths. A solution which can evaluate images and alert the dermatologists about the presence of melanoma has the potential to reduce a lot of manual effort needed in diagnosis.

Importing Skin Cancer Data

To do: Take necessary actions to read the data

Importing all the important libraries

```
import pathlib
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import PIL
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

```
## If you are using the data by mounting the google drive, use the following :
from google.colab import drive
drive.mount('/content/gdrive')

##Ref:https://towardsdatascience.com/downloading-datasets-into-google-drive-via-google-colab-bcb1b30b0166
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
```

This assignment uses a dataset of about 2357 images of skin cancer types. The dataset contains 9 sub-directories in each train and test subdirectories. The 9 sub-directories contains the images of 9 skin cancer types respectively.

```
# Defining the path for train and test images
## Todo: Update the paths of the train and test dataset
data_dir_train = pathlib.Path("/content/gdrive/My Drive/CNN_assignment/Skin cancer ISIC The International Skin Imaging Collaboration/Train/"
data_dir_test = pathlib.Path("/content/gdrive/My Drive/CNN_assignment/Skin cancer ISIC The International Skin Imaging Collaboration/Test/")
```

```
image_count_train = len(list(data_dir_train.glob('*/*.jpg')))
print(image_count_train)
image_count_test = len(list(data_dir_test.glob('*/*.jpg')))
print(image_count_test)
```

```
2239
118
```

Load using keras.preprocessing

Let's load these images off disk using the helpful image_dataset_from_directory utility.

Create a dataset

Define some parameters for the loader:

```
batch_size = 32
img_height = 180
img_width = 180
```

Use 80% of the images for training, and 20% for validation.

```
## Write your train dataset here
## Note use seed=123 while creating your dataset using tf.keras.preprocessing.image_dataset_from_directory
```

```
## Note, make sure your resize your images to the size img_height*img_width, while writting the dataset
train_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir_train,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 2239 files belonging to 9 classes.
Using 1792 files for training.
```

```
## Write your validation dataset here
## Note use seed=123 while creating your dataset using tf.keras.preprocessing.image_dataset_from_directory
## Note, make sure your resize your images to the size img_height*img_width, while writting the dataset
val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir_train,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 2239 files belonging to 9 classes.
Using 447 files for validation.
```

```
# List out all the classes of skin cancer and store them in a list.
# You can find the class names in the class_names attribute on these datasets.
# These correspond to the directory names in alphabetical order.
class_names = train_ds.class_names
print(class_names)
```

```
['actinic keratosis', 'basal cell carcinoma', 'dermatofibroma', 'melanoma', 'nevus', 'pigmented benign keratosis', 'seborrheic keratosis', 'squa
```

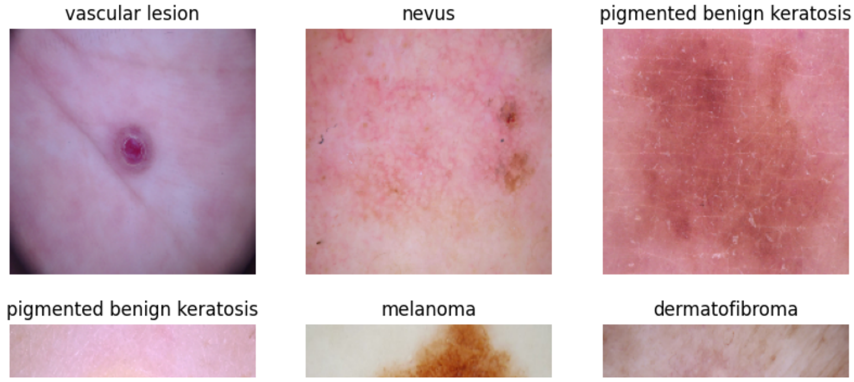
Visualize the data

Todo, create a code to visualize one instance of all the nine classes present in the dataset

```
import matplotlib.pyplot as plt

### your code goes here, you can use training or validation data to visualize

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```




```
for image_batch, labels_batch in train_ds:
    print(image_batch.shape)
    print(labels_batch.shape)
    break
```

```
(32, 180, 180, 3)
(32,)
```

The `image_batch` is a tensor of the shape `(32, 180, 180, 3)`. This is a batch of 32 images of shape `180x180x3` (the last dimension refers to color channels RGB). The `label_batch` is a tensor of the shape `(32,)`, these are corresponding labels to the 32 images.

`Dataset.cache()` keeps the images in memory after they're loaded off disk during the first epoch.

`Dataset.prefetch()` overlaps data preprocessing and model execution while training.

```
AUTOTUNE = tf.data.experimental.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

Create the model

Todo: Create a CNN model, which can accurately detect 9 classes present in the dataset. Use `layers.experimental.preprocessing.Rescaling` to normalize pixel values between (0,1). The RGB channel values are in the `[0, 255]` range. This is not ideal for a neural network. Here, it is good to standardize values to be in the `[0, 1]`

```
num_classes = len(class_names)

model = Sequential([
    layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes,activation='softmax')
])
```

Compile the model

Choose an appropirate optimiser and loss function for model training

```
### Todo, choose an appropirate optimiser and loss function
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
# View the summary of all layers
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_3 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_9 (Conv2D)           (None, 180, 180, 16)      448

 max_pooling2d_9 (MaxPooling  (None, 90, 90, 16)       0
 2D)

 conv2d_10 (Conv2D)          (None, 90, 90, 32)        4640

 max_pooling2d_10 (MaxPoolin  (None, 45, 45, 32)       0
 g2D)

 conv2d_11 (Conv2D)          (None, 45, 45, 64)        18496

 max_pooling2d_11 (MaxPoolin  (None, 22, 22, 64)       0
 g2D)

 flatten_3 (Flatten)         (None, 30976)             0

 dense_6 (Dense)             (None, 128)               3965056

 dense_7 (Dense)             (None, 9)                 1161

=================================================================
Total params: 3,989,801
Trainable params: 3,989,801
Non-trainable params: 0
_________________________________________________________________
```

Train the model

```
epochs = 20
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)
```

```
Epoch 1/20
/usr/local/lib/python3.10/dist-packages/keras/backend.py:5612: UserWarning: "`sparse_categorical_crossentropy` received `from_logits=True`, but
  output, from_logits = _get_logits(
56/56 [==============================] - 27s 135ms/step - loss: 1.9872 - accuracy: 0.2846 - val_loss: 1.6928 - val_accuracy: 0.4251
Epoch 2/20
56/56 [==============================] - 1s 21ms/step - loss: 1.5269 - accuracy: 0.4632 - val_loss: 1.5486 - val_accuracy: 0.4720
Epoch 3/20
56/56 [==============================] - 1s 20ms/step - loss: 1.3584 - accuracy: 0.5391 - val_loss: 1.3728 - val_accuracy: 0.5391
Epoch 4/20
56/56 [==============================] - 1s 20ms/step - loss: 1.2748 - accuracy: 0.5463 - val_loss: 1.4094 - val_accuracy: 0.5190
Epoch 5/20
56/56 [==============================] - 1s 20ms/step - loss: 1.1863 - accuracy: 0.5792 - val_loss: 1.4251 - val_accuracy: 0.5078
Epoch 6/20
56/56 [==============================] - 1s 20ms/step - loss: 1.1387 - accuracy: 0.5837 - val_loss: 1.4940 - val_accuracy: 0.4922
Epoch 7/20
56/56 [==============================] - 1s 20ms/step - loss: 1.0748 - accuracy: 0.6144 - val_loss: 1.4424 - val_accuracy: 0.5235
Epoch 8/20
56/56 [==============================] - 1s 20ms/step - loss: 1.0140 - accuracy: 0.6334 - val_loss: 1.3746 - val_accuracy: 0.5056
Epoch 9/20
56/56 [==============================] - 1s 26ms/step - loss: 0.9096 - accuracy: 0.6780 - val_loss: 1.4496 - val_accuracy: 0.5101
Epoch 10/20
56/56 [==============================] - 1s 21ms/step - loss: 0.9106 - accuracy: 0.6685 - val_loss: 1.4259 - val_accuracy: 0.5168
Epoch 11/20
56/56 [==============================] - 1s 23ms/step - loss: 0.7603 - accuracy: 0.7204 - val_loss: 1.4644 - val_accuracy: 0.5280
Epoch 12/20
56/56 [==============================] - 1s 25ms/step - loss: 0.7166 - accuracy: 0.7427 - val_loss: 1.6434 - val_accuracy: 0.5459
Epoch 13/20
56/56 [==============================] - 1s 25ms/step - loss: 0.6593 - accuracy: 0.7545 - val_loss: 1.8866 - val_accuracy: 0.4855
Epoch 14/20
56/56 [==============================] - 1s 24ms/step - loss: 0.5553 - accuracy: 0.8036 - val_loss: 1.8735 - val_accuracy: 0.5213
Epoch 15/20
56/56 [==============================] - 1s 20ms/step - loss: 0.5139 - accuracy: 0.8136 - val_loss: 1.8492 - val_accuracy: 0.5324
Epoch 16/20
56/56 [==============================] - 1s 20ms/step - loss: 0.4457 - accuracy: 0.8304 - val_loss: 1.7530 - val_accuracy: 0.5638
Epoch 17/20
56/56 [==============================] - 1s 20ms/step - loss: 0.4174 - accuracy: 0.8454 - val_loss: 1.9927 - val_accuracy: 0.5459
Epoch 18/20
56/56 [==============================] - 1s 20ms/step - loss: 0.3648 - accuracy: 0.8700 - val_loss: 1.9508 - val_accuracy: 0.5280
Epoch 19/20
56/56 [==============================] - 1s 20ms/step - loss: 0.3611 - accuracy: 0.8644 - val_loss: 2.1271 - val_accuracy: 0.5391
Epoch 20/20
56/56 [==============================] - 1s 22ms/step - loss: 0.3193 - accuracy: 0.8789 - val_loss: 2.1023 - val_accuracy: 0.4989
```

Visualizing training results

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

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

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

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

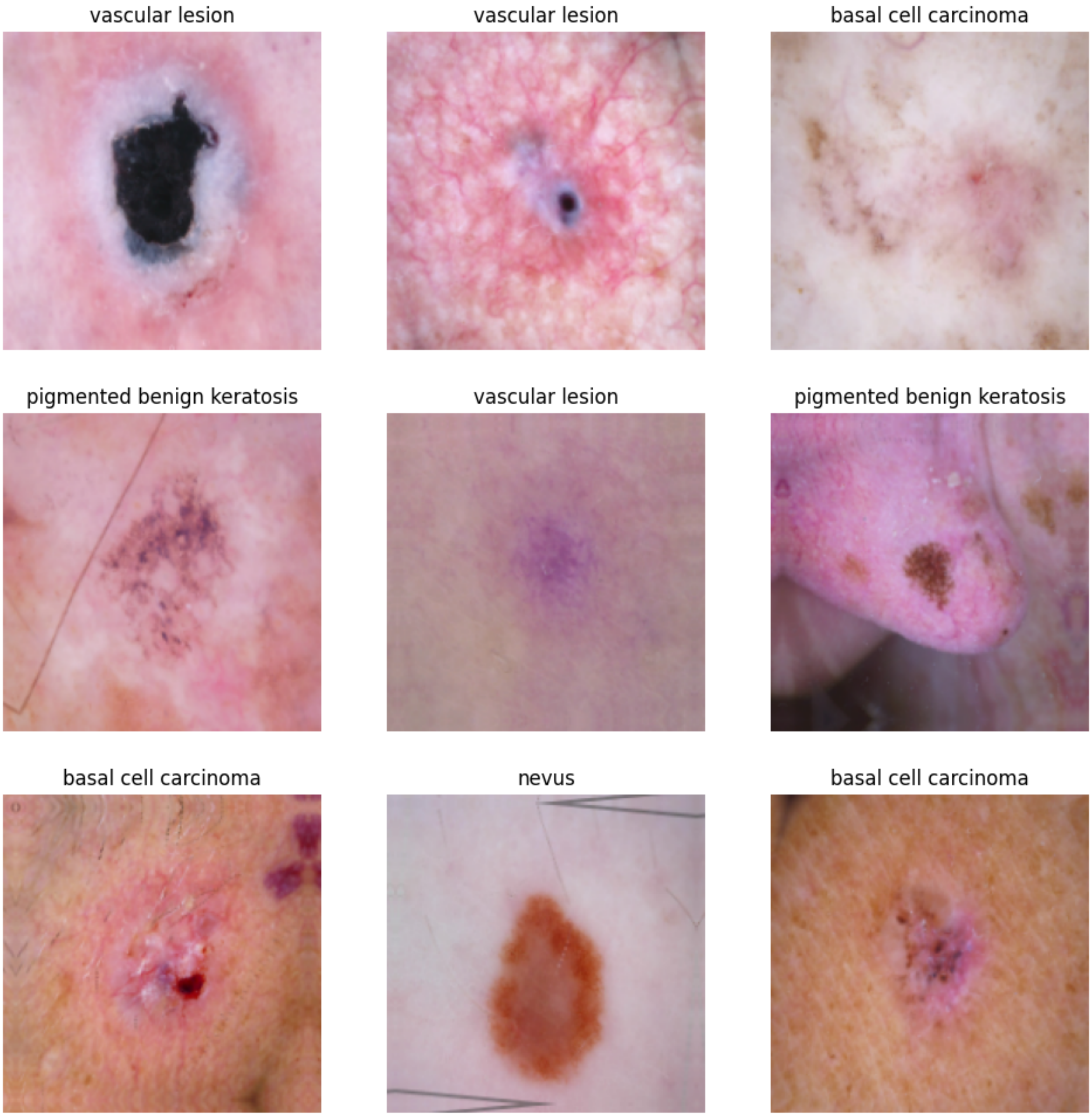
Todo: Write your findings after the model fit, see if there is an evidence of model overfit or underfit

Answer: The model has achieved only around 55% accuracy on the validation set but 95% accuracy, it is overfitting.

```
# Todo, after you have analysed the model fit history for presence of underfit or overfit, choose an appropriate data augmentatio
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal_and_vertical",
                                    input_shape=(img_height,
                                                        img_width,
                                                        3)),
        layers.RandomRotation(0.1, fill_mode='reflect'),
        layers.RandomZoom(0.1),
    ]
)
```

```
# Todo, visualize how your augmentation strategy works for one instance of training image.

plt.figure(figsize=(12, 12))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(data_augmentation(images)[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```



Todo:

Create the model, compile and train the model

```
model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.25),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.25),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dropout(0.25),
```

```
    layers.Dense(num_classes,activation='softmax')
])
```

Compiling the model

```
## Your code goes here
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_4 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_12 (Conv2D)          (None, 180, 180, 16)      448

 max_pooling2d_12 (MaxPoolin  (None, 90, 90, 16)       0
 g2D)

 dropout_6 (Dropout)         (None, 90, 90, 16)        0

 conv2d_13 (Conv2D)          (None, 90, 90, 32)        4640

 max_pooling2d_13 (MaxPoolin  (None, 45, 45, 32)       0
 g2D)

 conv2d_14 (Conv2D)          (None, 45, 45, 64)        18496

 max_pooling2d_14 (MaxPoolin  (None, 22, 22, 64)       0
 g2D)

 dropout_7 (Dropout)         (None, 22, 22, 64)        0

 flatten_4 (Flatten)         (None, 30976)             0

 dense_8 (Dense)             (None, 128)               3965056

 dropout_8 (Dropout)         (None, 128)               0

 dense_9 (Dense)             (None, 9)                 1161

=================================================================
Total params: 3,989,801
Trainable params: 3,989,801
Non-trainable params: 0
_________________________________________________________________
```

Training the model

```
## Your code goes here, note: train your model for 20 epochs
epochs = 20
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)
```

```
Epoch 1/20
56/56 [==============================] - 4s 31ms/step - loss: 2.4452 - accuracy: 0.1875 - val_loss: 2.0686 - val_accuracy: 0.1924
Epoch 2/20
56/56 [==============================] - 2s 28ms/step - loss: 2.0565 - accuracy: 0.1830 - val_loss: 2.0558 - val_accuracy: 0.2103
Epoch 3/20
56/56 [==============================] - 2s 27ms/step - loss: 1.9952 - accuracy: 0.2606 - val_loss: 1.9393 - val_accuracy: 0.3512
Epoch 4/20
56/56 [==============================] - 2s 28ms/step - loss: 1.8933 - accuracy: 0.2958 - val_loss: 1.8854 - val_accuracy: 0.3289
Epoch 5/20
56/56 [==============================] - 2s 29ms/step - loss: 1.7834 - accuracy: 0.3532 - val_loss: 1.7000 - val_accuracy: 0.3870
Epoch 6/20
56/56 [==============================] - 2s 30ms/step - loss: 1.6527 - accuracy: 0.4141 - val_loss: 1.6702 - val_accuracy: 0.4251
Epoch 7/20
56/56 [==============================] - 2s 28ms/step - loss: 1.6461 - accuracy: 0.4163 - val_loss: 1.6655 - val_accuracy: 0.4251
Epoch 8/20
56/56 [==============================] - 2s 28ms/step - loss: 1.5691 - accuracy: 0.4431 - val_loss: 1.5425 - val_accuracy: 0.5056
Epoch 9/20
56/56 [==============================] - 2s 27ms/step - loss: 1.4854 - accuracy: 0.4821 - val_loss: 1.4911 - val_accuracy: 0.5056
Epoch 10/20
56/56 [==============================] - 2s 28ms/step - loss: 1.4331 - accuracy: 0.4978 - val_loss: 1.4488 - val_accuracy: 0.5101
Epoch 11/20
```

```
56/56 [==============================] - 2s 27ms/step - loss: 1.3733 - accuracy: 0.5262 - val_loss: 1.3950 - val_accuracy: 0.5369
Epoch 12/20
56/56 [==============================] - 2s 29ms/step - loss: 1.3630 - accuracy: 0.5363 - val_loss: 1.4166 - val_accuracy: 0.5302
Epoch 13/20
56/56 [==============================] - 2s 28ms/step - loss: 1.3291 - accuracy: 0.5424 - val_loss: 1.4306 - val_accuracy: 0.4966
Epoch 14/20
56/56 [==============================] - 2s 28ms/step - loss: 1.2986 - accuracy: 0.5502 - val_loss: 1.5042 - val_accuracy: 0.4877
Epoch 15/20
56/56 [==============================] - 2s 28ms/step - loss: 1.2619 - accuracy: 0.5569 - val_loss: 1.3870 - val_accuracy: 0.5257
Epoch 16/20
56/56 [==============================] - 2s 28ms/step - loss: 1.2303 - accuracy: 0.5725 - val_loss: 1.4934 - val_accuracy: 0.4899
Epoch 17/20
56/56 [==============================] - 2s 28ms/step - loss: 1.1704 - accuracy: 0.5731 - val_loss: 1.3588 - val_accuracy: 0.5347
Epoch 18/20
56/56 [==============================] - 2s 30ms/step - loss: 1.1869 - accuracy: 0.5837 - val_loss: 1.4270 - val_accuracy: 0.5280
Epoch 19/20
56/56 [==============================] - 2s 29ms/step - loss: 1.1461 - accuracy: 0.6004 - val_loss: 1.4237 - val_accuracy: 0.5011
Epoch 20/20
56/56 [==============================] - 2s 28ms/step - loss: 1.1064 - accuracy: 0.5910 - val_loss: 1.3429 - val_accuracy: 0.5391
```

Visualizing the results

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

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

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

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

Todo: Write your findings after the model fit, see if there is an evidence of model overfit or underfit. Do you think there is some improvement now as compared to the previous model run?

Answer: ovewfit is still exist but the loss is few.

Todo: Find the distribution of classes in the training dataset.

Context: Many times real life datasets can have class imbalance, one class can have proportionately higher number of samples compared to the others. Class imbalance can have a detrimental effect on the final model quality. Hence as a sanity check it becomes important to check what is the distribution of classes in the data.

```
from glob import glob
path_list = [x for x in glob(os.path.join(data_dir_train, '*', '*.jpg'))]
lesion_list = [os.path.basename(os.path.dirname(y)) for y in glob(os.path.join(data_dir_train, '*', '*.jpg'))]

path_list += [x for x in glob(os.path.join(data_dir_test, '*', '*.jpg'))]
lesion_list += [os.path.basename(os.path.dirname(y)) for y in glob(os.path.join(data_dir_test, '*', '*.jpg'))]


df_dict_org = dict(zip(path_list, lesion_list))
org_df = pd.DataFrame(list(df_dict_org.items()),columns = ['Path','Label'])
org_df
```

	Path	Label
0	/content/gdrive/My Drive/CNN_assignment/Skin c...	actinic keratosis
1	/content/gdrive/My Drive/CNN_assignment/Skin c...	actinic keratosis
2	/content/gdrive/My Drive/CNN_assignment/Skin c...	actinic keratosis
3	/content/gdrive/My Drive/CNN_assignment/Skin c...	actinic keratosis
4	/content/gdrive/My Drive/CNN_assignment/Skin c...	actinic keratosis
...	...	...
2352	/content/gdrive/My Drive/CNN_assignment/Skin c...	squamous cell carcinoma
2353	/content/gdrive/My Drive/CNN_assignment/Skin c...	squamous cell carcinoma
2354	/content/gdrive/My Drive/CNN_assignment/Skin c...	vascular lesion
2355	/content/gdrive/My Drive/CNN_assignment/Skin c...	vascular lesion
2356	/content/gdrive/My Drive/CNN_assignment/Skin c...	vascular lesion

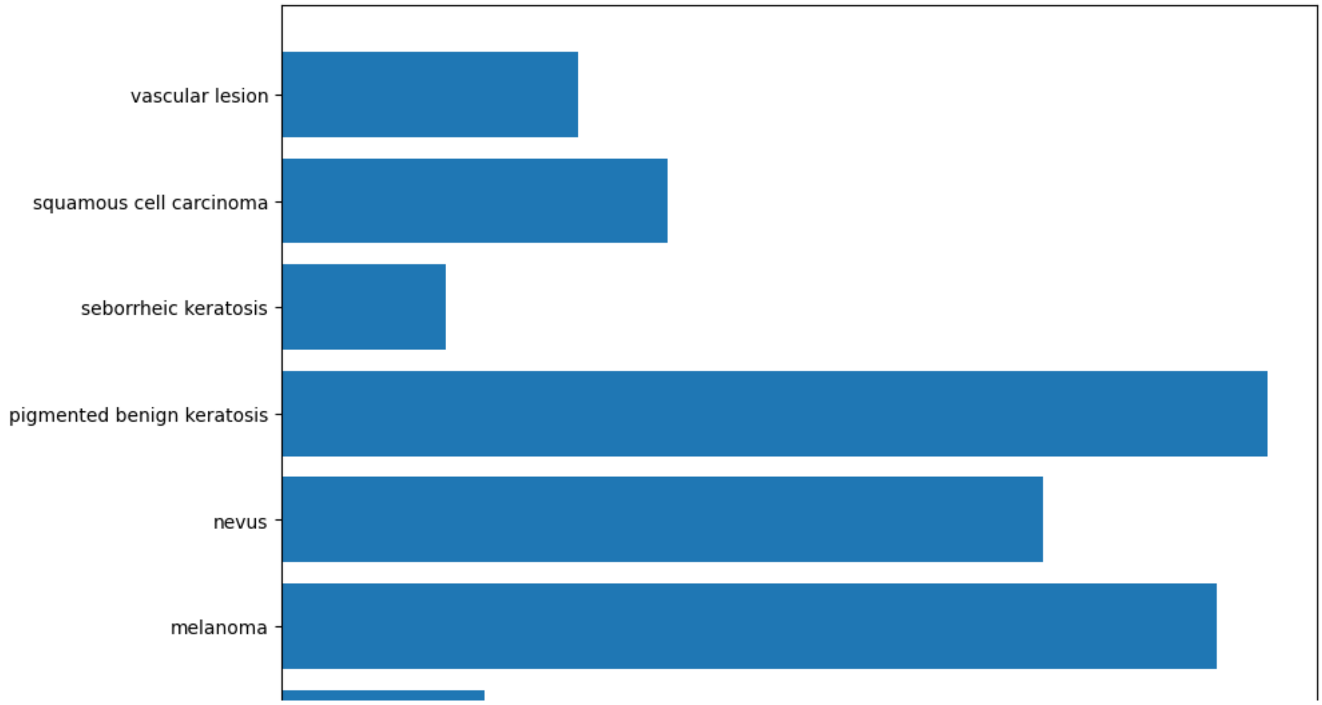
2357 rows × 2 columns

```
len(df_dict_org)
```

```
2357
```

```
count=[]
for i in class_names:
    count.append(len(list(data_dir_train.glob(i+'/*.jpg'))))
plt.figure(figsize=(10,10))
plt.barh(class_names,count)
```

```
<BarContainer object of 9 artists>
```



Todo: Write your findings here:

- Which class has the least number of samples?

- Which classes dominate the data in terms proportionate number of samples?

Answer-1 :- seborrtheic keratosis has least number of samples

Answer-2:- pigmented benign keratosis and melanoma have proprtionate number of classes

Todo: Rectify the class imbalance

Context: You can use a python package known as `Augmentor` (https://augmentor.readthedocs.io/en/master/) to add more samples across all classes so that none of the classes have very few samples.

```
!pip install Augmentor
```

```
Requirement already satisfied: Augmentor in /usr/local/lib/python3.10/dist-packages (0.2.12)
Requirement already satisfied: Pillow>=5.2.0 in /usr/local/lib/python3.10/dist-packages (from Augmentor) (9.4.0)
Requirement already satisfied: tqdm>=4.9.0 in /usr/local/lib/python3.10/dist-packages (from Augmentor) (4.65.0)
Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from Augmentor) (1.22.4)
```

To use `Augmentor`, the following general procedure is followed:

1. Instantiate a `Pipeline` object pointing to a directory containing your initial image data set.
2. Define a number of operations to perform on this data set using your `Pipeline` object.
3. Execute these operations by calling the `Pipeline's sample()` method.

```
path_to_training_dataset="/content/gdrive/My Drive/CNN_assignment/Skin cancer ISIC The International Skin Imaging Collaboration/Train/"
import Augmentor
for i in class_names:
    #p = Augmentor.Pipeline(path_to_training_dataset + i)
    p = Augmentor.Pipeline(path_to_training_dataset + i, output_directory='/content/gdrive/My Drive/CNN_assignment/Augmentor/'+i+'/')
    p.rotate(probability=0.7, max_left_rotation=10, max_right_rotation=10)
    p.sample(500) ## We are adding 500 samples per class to make sure that none of the classes are sparse.
```

```
Initialised with 114 image(s) found.
Output directory set to /content/gdrive/My Drive/CNN_assignment/Augmentor/actinic keratosis/.Processing <PIL.Image.Image image mode=RGB size=600
Initialised with 376 image(s) found.
Output directory set to /content/gdrive/My Drive/CNN_assignment/Augmentor/basal cell carcinoma/.Processing <PIL.Image.Image image mode=RGB size=
Initialised with 95 image(s) found.
Output directory set to /content/gdrive/My Drive/CNN_assignment/Augmentor/dermatofibroma/.Processing <PIL.Image.Image image mode=RGB size=600x45
Initialised with 438 image(s) found.
Output directory set to /content/gdrive/My Drive/CNN_assignment/Augmentor/melanoma/.Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB
Initialised with 357 image(s) found.
Output directory set to /content/gdrive/My Drive/CNN_assignment/Augmentor/nevus/.Processing <PIL.Image.Image image mode=RGB size=824x719 at 0x7D
Initialised with 462 image(s) found.
```

```
Output directory set to /content/gdrive/My Drive/CNN_assignment/Augmentor/pigmented benign keratosis/.Processing <PIL.Image.Image image mode=RGB
Initialised with 77 image(s) found.
Output directory set to /content/gdrive/My Drive/CNN_assignment/Augmentor/seborrheic keratosis/.Processing <PIL.Image.Image image mode=RGB size=
Initialised with 181 image(s) found.
Output directory set to /content/gdrive/My Drive/CNN_assignment/Augmentor/squamous cell carcinoma/.Processing <PIL.Image.Image image mode=RGB si
Initialised with 139 image(s) found.
Output directory set to /content/gdrive/My Drive/CNN_assignment/Augmentor/vascular lesion/.Processing <PIL.JpegImagePlugin.JpegImageFile image m
```

Augmentor has stored the augmented images in the output sub-directory of each of the sub-directories of skin cancer types.. Lets take a look at total count of augmented images.

按兩下 (或按 Enter 鍵) 即可編輯

```
data_dir_train_aug = pathlib.Path('/content/gdrive/My Drive/CNN_assignment/Augmentor/')
```

```
image_count_train = len(list(data_dir_train_aug.glob('*/*.jpg')))
print(image_count_train)
```

```
9000
```

```
%cd '/content/gdrive/My Drive/CNN_assignment/Org_With_Aug_Data'
```

```
/content/gdrive/My Drive/CNN_assignment/Org_With_Aug_Data
```

```
# copy orginal train and augmented data into another folder

%cd '/content/gdrive/My Drive/CNN_assignment/Augmentor/'
%cp -av * '/content/gdrive/My Drive/CNN_assignment/Org_With_Aug_Data/' >> /dev/null
```

```
/content/gdrive/My Drive/CNN_assignment/Augmentor
```

```
%cd '/content/gdrive/My Drive/CNN_assignment/Skin cancer ISIC The International Skin Imaging Collaboration/Train'
%cp -av * '/content/gdrive/My Drive/CNN_assignment/Org_With_Aug_Data' >> /dev/null
```

```
/content/gdrive/My Drive/CNN_assignment/Skin cancer ISIC The International Skin Imaging Collaboration/Train
```

```
%cd '/content/gdrive/My Drive/CNN_assignment/Org_With_Aug_Data'
%ls
```

```
/content/gdrive/My Drive/CNN_assignment/Org_With_Aug_Data
'actinic keratosis'/     'pigmented benign keratosis'/
'basal cell carcinoma'/  'seborrheic keratosis'/
 dermatofibroma/         'squamous cell carcinoma'/
 melanoma/               'vascular lesion'/
 nevus/
```

Lets see the distribution of augmented data after adding new images to the original training data.

```
data_dir_train_aug='/content/gdrive/My Drive/CNN_assignment/Org_With_Aug_Data/'
path_list_new = [x for x in glob(os.path.join(data_dir_train_aug, '*', '*.jpg'))]
len(path_list_new)
```

```
11239
```

```
lesion_list_new = [os.path.basename(os.path.dirname(y)) for y in glob(os.path.join(data_dir_train_aug, '*', '*.jpg'))]
len(lesion_list_new)
```

```
11239
```

```
dataframe_dict_new = dict(zip(path_list_new, lesion_list_new))
```

```
df2 = pd.DataFrame(list(dataframe_dict_new.items()),columns = ['Path','Label'])
#new_df = org_df.append(df2)
new_df=pd.concat([org_df, df2], ignore_index=True)
```

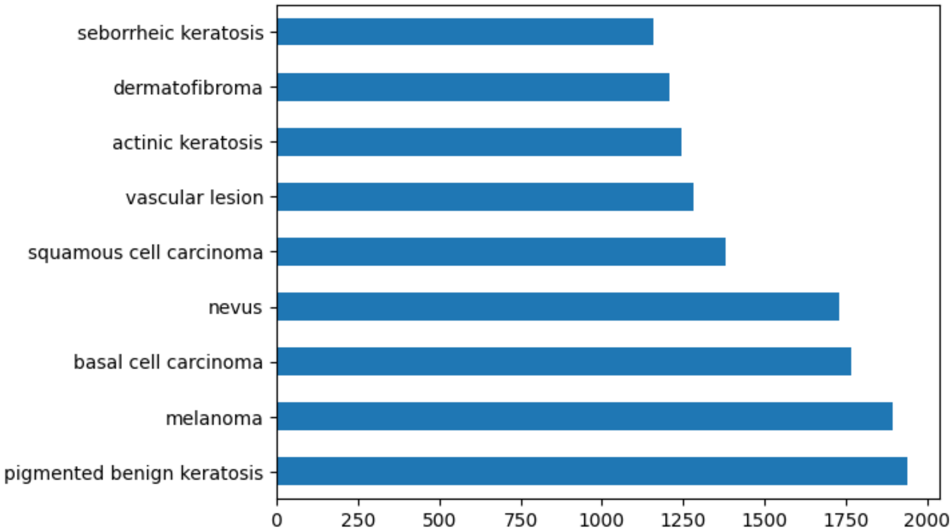
```
new_df['Label'].value_counts()
```

```
pigmented benign keratosis    1940
melanoma                      1892
basal cell carcinoma          1768
nevus                         1730
squamous cell carcinoma       1378
vascular lesion               1281
actinic keratosis             1244
dermatofibroma                1206
seborrheic keratosis          1157
Name: Label, dtype: int64
```

```
CountStatus = new_df.value_counts(new_df['Label'].values, sort=True)
```

```
CountStatus.plot.barh()
```

```
<Axes: >
```



So, now we have added 500 images to all the classes to maintain some class balance. We can add more images as we want to improve training process.

Todo: Train the model on the data created using Augmentor

```
batch_size = 32
img_height = 180
img_width = 180
```

Todo: Create a training dataset

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir_train_aug,
    seed=123,
    validation_split = 0.2,
    subset ='training',
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 11239 files belonging to 9 classes.
Using 8992 files for training.
```

Todo: Create a validation dataset

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir_train_aug,
    seed=123,
    validation_split = 0.2,
    subset = 'validation',
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 11239 files belonging to 9 classes.
Using 2247 files for validation.
```

```
AUTOTUNE = tf.data.experimental.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

Todo: Create your model (make sure to include normalization)

```
model = Sequential([
    layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.25),
    layers.Dense(num_classes,activation='softmax')
])
```

Todo: Compile your model (Choose optimizer and loss function appropriately)

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

Todo: Train your model

```
epochs = 50
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)
```

```
Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/backend.py:5612: UserWarning: "`sparse_categorical_crossentropy` received `from_logits=True`, b
  output, from_logits = _get_logits(
281/281 [==============================] - 84s 104ms/step - loss: 1.8689 - accuracy: 0.2815 - val_loss: 1.6156 - val_accuracy: 0.4108
Epoch 2/50
281/281 [==============================] - 8s 28ms/step - loss: 1.3824 - accuracy: 0.4685 - val_loss: 1.2658 - val_accuracy: 0.5389
Epoch 3/50
281/281 [==============================] - 8s 28ms/step - loss: 1.1722 - accuracy: 0.5582 - val_loss: 1.0824 - val_accuracy: 0.6088
Epoch 4/50
281/281 [==============================] - 9s 33ms/step - loss: 1.0239 - accuracy: 0.6219 - val_loss: 1.0819 - val_accuracy: 0.5964
Epoch 5/50
281/281 [==============================] - 10s 36ms/step - loss: 0.8683 - accuracy: 0.6849 - val_loss: 0.7853 - val_accuracy: 0.7365
Epoch 6/50
281/281 [==============================] - 8s 29ms/step - loss: 0.6960 - accuracy: 0.7450 - val_loss: 0.7591 - val_accuracy: 0.7339
Epoch 7/50
281/281 [==============================] - 9s 31ms/step - loss: 0.5883 - accuracy: 0.7812 - val_loss: 0.5864 - val_accuracy: 0.7824
Epoch 8/50
281/281 [==============================] - 8s 27ms/step - loss: 0.5246 - accuracy: 0.8102 - val_loss: 0.5043 - val_accuracy: 0.8242
Epoch 9/50
281/281 [==============================] - 8s 28ms/step - loss: 0.4625 - accuracy: 0.8292 - val_loss: 0.4716 - val_accuracy: 0.8385
Epoch 10/50
281/281 [==============================] - 8s 27ms/step - loss: 0.4051 - accuracy: 0.8534 - val_loss: 0.4791 - val_accuracy: 0.8505
Epoch 11/50
281/281 [==============================] - 8s 28ms/step - loss: 0.3823 - accuracy: 0.8614 - val_loss: 0.5677 - val_accuracy: 0.7988
Epoch 12/50
281/281 [==============================] - 8s 28ms/step - loss: 0.3655 - accuracy: 0.8643 - val_loss: 1.1829 - val_accuracy: 0.6538
Epoch 13/50
281/281 [==============================] - 8s 27ms/step - loss: 0.3475 - accuracy: 0.8727 - val_loss: 0.4529 - val_accuracy: 0.8669
Epoch 14/50
281/281 [==============================] - 8s 28ms/step - loss: 0.2956 - accuracy: 0.8896 - val_loss: 0.4458 - val_accuracy: 0.8549
Epoch 15/50
281/281 [==============================] - 8s 27ms/step - loss: 0.2705 - accuracy: 0.8976 - val_loss: 0.4187 - val_accuracy: 0.8674
Epoch 16/50
281/281 [==============================] - 8s 28ms/step - loss: 0.2657 - accuracy: 0.9025 - val_loss: 0.4421 - val_accuracy: 0.8687
Epoch 17/50
281/281 [==============================] - 8s 28ms/step - loss: 0.2797 - accuracy: 0.8959 - val_loss: 0.4444 - val_accuracy: 0.8647
Epoch 18/50
281/281 [==============================] - 8s 29ms/step - loss: 0.2531 - accuracy: 0.9062 - val_loss: 0.3707 - val_accuracy: 0.8941
```

```
Epoch 19/50
281/281 [==============================] - 8s 29ms/step - loss: 0.2268 - accuracy: 0.9157 - val_loss: 0.4549 - val_accuracy: 0.8718
Epoch 20/50
281/281 [==============================] - 8s 28ms/step - loss: 0.2370 - accuracy: 0.9117 - val_loss: 0.4682 - val_accuracy: 0.8603
Epoch 21/50
281/281 [==============================] - 8s 28ms/step - loss: 0.2247 - accuracy: 0.9155 - val_loss: 0.4029 - val_accuracy: 0.8901
Epoch 22/50
281/281 [==============================] - 8s 28ms/step - loss: 0.2154 - accuracy: 0.9135 - val_loss: 0.5238 - val_accuracy: 0.8540
Epoch 23/50
281/281 [==============================] - 8s 27ms/step - loss: 0.2289 - accuracy: 0.9114 - val_loss: 0.4271 - val_accuracy: 0.8785
Epoch 24/50
281/281 [==============================] - 8s 28ms/step - loss: 0.1924 - accuracy: 0.9253 - val_loss: 0.4257 - val_accuracy: 0.8999
Epoch 25/50
281/281 [==============================] - 8s 27ms/step - loss: 0.1943 - accuracy: 0.9295 - val_loss: 0.4275 - val_accuracy: 0.8723
Epoch 26/50
281/281 [==============================] - 8s 28ms/step - loss: 0.2081 - accuracy: 0.9220 - val_loss: 0.4858 - val_accuracy: 0.8652
Epoch 27/50
281/281 [==============================] - 8s 27ms/step - loss: 0.1856 - accuracy: 0.9306 - val_loss: 0.4432 - val_accuracy: 0.8781
Epoch 28/50
```

Todo: Visualize the model results

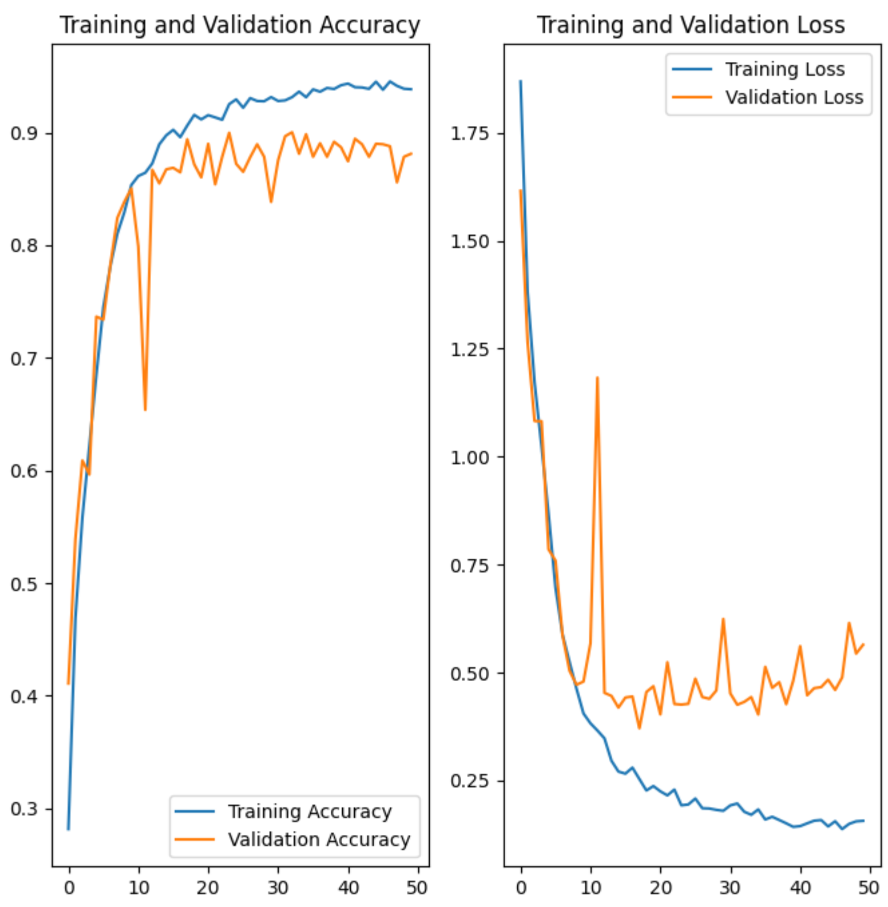
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

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



Todo: Analyze your results here. Did you get rid of underfitting/overfitting? Did class rebalance help?

Accuracy on training data is increased to around 85% by using Augmentor library.

The class rebalance is helped for overfitting issue.

For better accuracy, it can be solved by add more layer,neurons or adding dropout layers, and tuning the hyperparameter.

✓ 0 秒 完成時間：下午3:05