

```
#import 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
```

```
# mount google drive.
```

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

```
Mounted at /content/gdrive
```

```
# Unzip the dataset
```

```
!unzip "/content/gdrive/MyDrive/CNN_assignment.zip" > /dev/null
```

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
```

```
data_dir_train = pathlib.Path("/content/Skin cancer ISIC The  
International Skin Imaging Collaboration/Train/")
data_dir_test = pathlib.Path("/content/Skin cancer ISIC The  
International Skin Imaging Collaboration/Test/")
```

```
# Count the number of image in Train and Test directory
```

```
# Using the glob to retrieve files/pathnames matching a specified  
pattern.
```

```
#Train Image count
```

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

```
#Test Image count
```

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

```
2239
```

```
118
```

Create a dataset

Define some parameters for the loader:

```

batch_size = 32
img_height = 180
img_width = 180

#Train dataset
train_ds =
tf.keras.preprocessing.image_dataset_from_directory(data_dir_train,bat
ch_size=batch_size,image_size=(img_height,img_width),label_mode='categ
orical',

```

```

seed=123,subset="training",validation_split=0.2)

```

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

```

#Validation Dataset
val_ds
=tf.keras.preprocessing.image_dataset_from_directory(data_dir_train,ba
tch_size=batch_size,image_size=(img_height,img_width),label_mode='cate
gorical',

```

```

seed=123,subset="validation",validation_split=0.2)

```

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

```

#All the classes of skin cancer.
class_names = train_ds.class_names
print(class_names)

```

```

['actinic keratosis', 'basal cell carcinoma', 'dermatofibroma',
'melanoma', 'nevus', 'pigmented benign keratosis', 'seborrheic
keratosis', 'squamous cell carcinoma', 'vascular lesion']

```

Visualize the data

```

import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import load_img

```

```

#Dictionary to store the path of image as per the class
files_path_dict = {}

```

```

for c in class_names:
    files_path_dict[c] = list(map(lambda x:str(data_dir_train)
+'/' +c+'/' +x,os.listdir(str(data_dir_train)+'/' +c)))

```

```

#Visualize image
plt.figure(figsize=(15,15))
index = 0
for c in class_names:
    path_list = files_path_dict[c][:1]

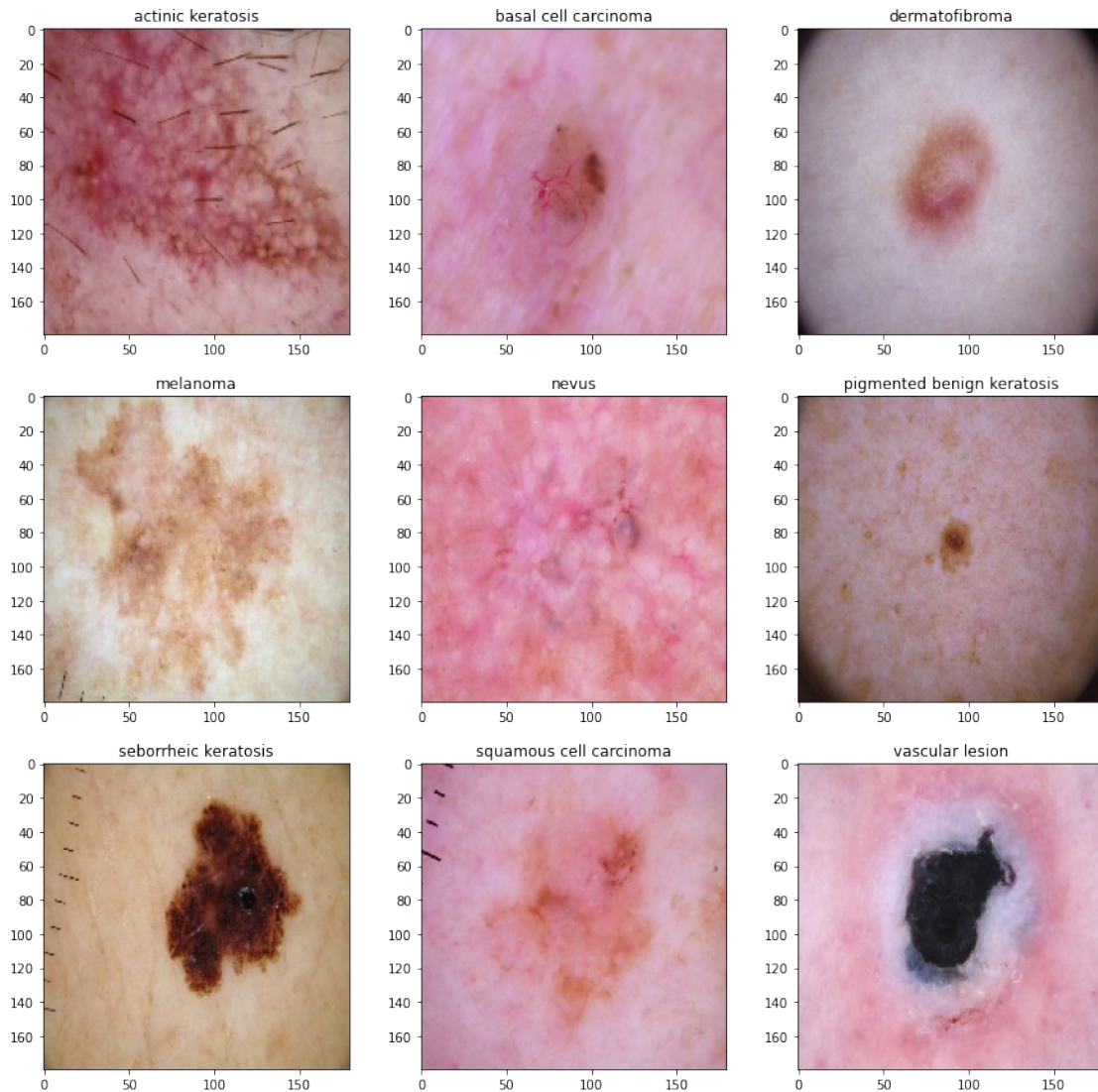
```

```

index += 1
plt.subplot(3,3,index)

plt.imshow(load_img(path_list[0],target_size=(img_height,img_width)))
plt.title(c)

```



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)
```

Model Creation

```
input_shape = (img_height,img_width,3)
```

```
model = Sequential()    #Sequential allows you to create models layer-  
by-layer
```

#First Convulation Layer

```
model.add(layers.experimental.preprocessing.Rescaling(1./255,input_shape=input_shape))
model.add(layers.Conv2D(32,kernel_size=(3,3),activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))
```

#Second Convulation Layer

```
model.add(layers.Conv2D(64,kernel_size=(3,3),activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))
```

#Third Convulation Layer

```
model.add(layers.Conv2D(128,kernel_size=(3,3),activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))
```

```
model.add(layers.Flatten())    #Keras.layers.flatten function flattens  
the multi-dimensional input tensors into a single dimension.
```

#Dense Layer

```
model.add(layers.Dense(512,activation='relu'))
```

#Dense Layer

```
model.add(layers.Dense(128,activation='relu'))
```

#Dense Layer with softmax activation function.

#Softmax is an activation function that scales numbers/logits into probabilities.

```
model.add(layers.Dense(len(class_names),activation='softmax'))
```

Compile the model

#Adam optimization: is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.

#categorical_crossentropy: Used as a loss function for multi-class classification model where there are two or more output labels.

```
model.compile(optimizer='Adam',
```

```
loss="categorical_crossentropy",
metrics=['accuracy'])
```

```
# summary of all layers
```

```
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 180, 180, 3)	0
conv2d (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_1 (Conv2D)	(None, 87, 87, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_2 (Conv2D)	(None, 41, 41, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 20, 20, 128)	0
flatten (Flatten)	(None, 51200)	0
dense (Dense)	(None, 512)	26214912
dense_1 (Dense)	(None, 128)	65664
dense_2 (Dense)	(None, 9)	1161
Total params: 26,374,985		
Trainable params: 26,374,985		
Non-trainable params: 0		

Train the model

```
epochs = 20
```

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

```
Epoch 1/20
```

```
56/56 [=====] - 51s 139ms/step - loss: 1.9974
- accuracy: 0.2695 - val_loss: 1.7946 - val_accuracy: 0.3132
```

```
Epoch 2/20
```

```
56/56 [=====] - 4s 78ms/step - loss: 1.6104 -
accuracy: 0.4090 - val_loss: 1.5644 - val_accuracy: 0.4183
```

Epoch 3/20
56/56 [=====] - 4s 78ms/step - loss: 1.5775 - accuracy: 0.4392 - val_loss: 1.5345 - val_accuracy: 0.4452
Epoch 4/20
56/56 [=====] - 4s 78ms/step - loss: 1.4652 - accuracy: 0.4838 - val_loss: 1.4956 - val_accuracy: 0.5078
Epoch 5/20
56/56 [=====] - 4s 78ms/step - loss: 1.3683 - accuracy: 0.5140 - val_loss: 1.3912 - val_accuracy: 0.5324
Epoch 6/20
56/56 [=====] - 4s 77ms/step - loss: 1.2163 - accuracy: 0.5709 - val_loss: 1.4011 - val_accuracy: 0.5168
Epoch 7/20
56/56 [=====] - 4s 77ms/step - loss: 1.2053 - accuracy: 0.5703 - val_loss: 1.4251 - val_accuracy: 0.4989
Epoch 8/20
56/56 [=====] - 4s 77ms/step - loss: 1.1963 - accuracy: 0.5765 - val_loss: 1.3598 - val_accuracy: 0.5123
Epoch 9/20
56/56 [=====] - 4s 77ms/step - loss: 1.0492 - accuracy: 0.6194 - val_loss: 1.4452 - val_accuracy: 0.5145
Epoch 10/20
56/56 [=====] - 4s 77ms/step - loss: 0.9761 - accuracy: 0.6585 - val_loss: 1.5185 - val_accuracy: 0.5213
Epoch 11/20
56/56 [=====] - 4s 78ms/step - loss: 0.9290 - accuracy: 0.6669 - val_loss: 1.4921 - val_accuracy: 0.5280
Epoch 12/20
56/56 [=====] - 4s 78ms/step - loss: 0.8361 - accuracy: 0.6931 - val_loss: 1.4196 - val_accuracy: 0.5414
Epoch 13/20
56/56 [=====] - 4s 77ms/step - loss: 0.7434 - accuracy: 0.7344 - val_loss: 1.6979 - val_accuracy: 0.5481
Epoch 14/20
56/56 [=====] - 4s 77ms/step - loss: 0.6123 - accuracy: 0.7656 - val_loss: 1.7575 - val_accuracy: 0.5391
Epoch 15/20
56/56 [=====] - 4s 77ms/step - loss: 0.5529 - accuracy: 0.7913 - val_loss: 1.7751 - val_accuracy: 0.5123
Epoch 16/20
56/56 [=====] - 4s 77ms/step - loss: 0.5492 - accuracy: 0.7974 - val_loss: 2.0880 - val_accuracy: 0.5123
Epoch 17/20
56/56 [=====] - 4s 77ms/step - loss: 0.4801 - accuracy: 0.8198 - val_loss: 2.1833 - val_accuracy: 0.5190
Epoch 18/20
56/56 [=====] - 4s 77ms/step - loss: 0.4094 - accuracy: 0.8482 - val_loss: 2.2062 - val_accuracy: 0.5459
Epoch 19/20
56/56 [=====] - 4s 77ms/step - loss: 0.3341 -

```
accuracy: 0.8638 - val_loss: 2.3358 - val_accuracy: 0.5213
Epoch 20/20
56/56 [=====] - 4s 77ms/step - loss: 0.2847 -
accuracy: 0.8884 - val_loss: 2.5026 - val_accuracy: 0.5145
```

Visualizing training results

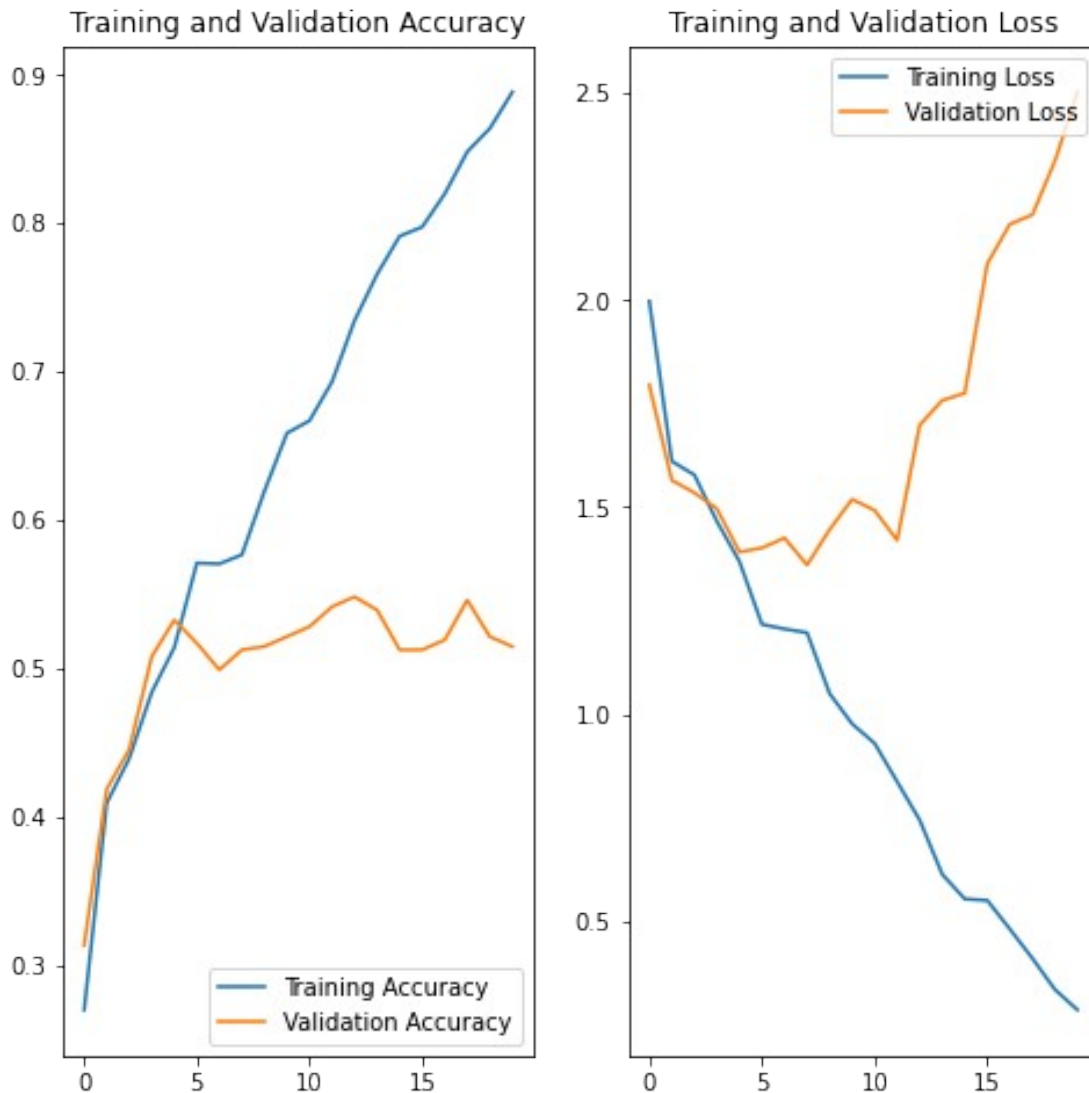
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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()
```



Model is overfitting. From the above Training vs Validation accuracy graph we can see that as the epoch increases the difference between Training accuracy and validation accuracy increases.

#Data augmentation strategy.

```
rescale = tf.keras.Sequential([
    #To rescale an input in the [0, 255] range to be in the [0, 1] range
    layers.experimental.preprocessing.Rescaling(1./255)
])

data_augmentation = tf.keras.Sequential([
    #Randomly flip each image horizontally and vertically.
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"
```



```

),

#Randomly rotate each image.
layers.experimental.preprocessing.RandomRotation(0.2),

#Randomly zoom each image during training.
layers.experimental.preprocessing.RandomZoom(0.2),

#Randomly translate each image during training.
layers.experimental.preprocessing.RandomTranslation(0.1, 0.1)
])

#Visualize the augmentation image
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")

```



Model 2 Creation

#Dropout layer: randomly sets input units to 0 with a frequency of rate at each step during training time, #which helps prevent overfitting. Inputs not set to 0 are scaled up by $1/(1 - \text{rate})$ such that the sum over all inputs is unchanged.

Your code goes here

*model2 = Sequential()
create models layer-by-layer*

#Sequential allows you to

*model2.add(data_augmentation)
model2.add(rescale)*

*#Augmentation layer
#Rescaling layer*

#First Convulation Layer

model2.add(layers.Conv2D(32, kernel_size=(3,3), activation='relu'))

```

model2.add(layers.MaxPool2D(pool_size=(2,2)))

#Dropout layer with 25% Fraction of the input units to drop.
model2.add(layers.Dropout(0.25))

#Second Convulation Layer
model2.add(layers.Conv2D(64,kernel_size=(3,3),activation='relu'))
model2.add(layers.MaxPool2D(pool_size=(2,2)))

#Dropout layer with 25% Fraction of the input units to drop.
model2.add(layers.Dropout(0.25))

#Third Convulation Layer
model2.add(layers.Conv2D(128,kernel_size=(3,3),activation='relu'))
model2.add(layers.MaxPool2D(pool_size=(2,2)))

#Keras.layers.flatten function flattens the multi-dimensional input tensors into a single dimension.
model2.add(layers.Flatten())

#Dense Layer
model2.add(layers.Dense(512,activation='relu'))

#Dense Layer
model2.add(layers.Dense(128,activation='relu'))

#Dropout layer with 50% Fraction of the input units to drop.
model2.add(layers.Dropout(0.50))

#Dense Layer with softmax activation function.
#Softmax is an activation function that scales numbers/logits into probabilities.
model2.add(layers.Dense(len(class_names),activation='softmax'))

```

Compiling the model

```

model2.compile(optimizer='Adam',
               loss="categorical_crossentropy",
               metrics=['accuracy'])

```

Training the model

```

epochs =20
history =
model2.fit(train_ds,epochs=epochs,validation_data=val_ds,verbose=1)

```

```

Epoch 1/20
56/56 [=====] - 8s 97ms/step - loss: 2.2754 -

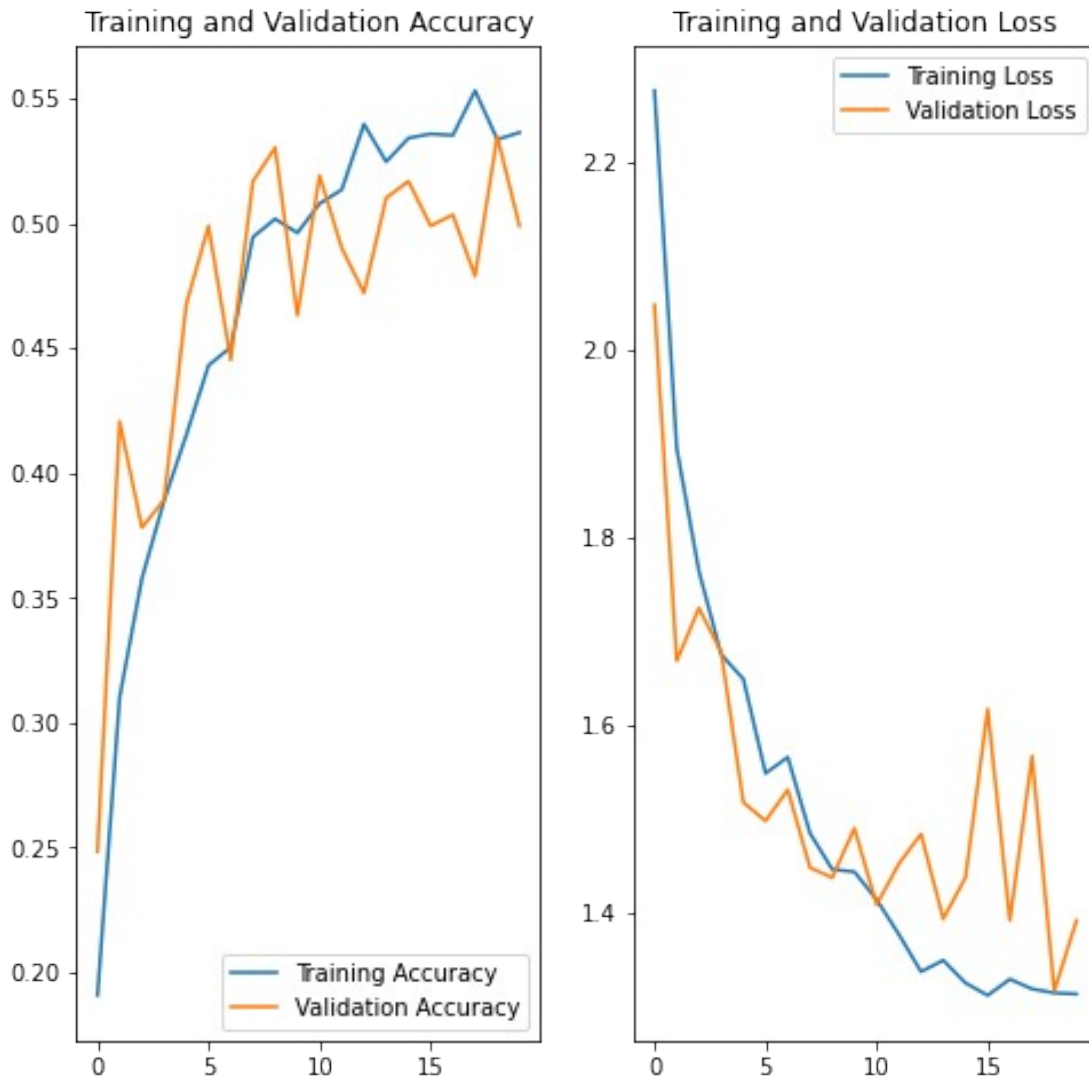
```

accuracy: 0.1908 - val_loss: 2.0474 - val_accuracy: 0.2483
Epoch 2/20
56/56 [=====] - 5s 92ms/step - loss: 1.8945 -
accuracy: 0.3103 - val_loss: 1.6684 - val_accuracy: 0.4206
Epoch 3/20
56/56 [=====] - 5s 92ms/step - loss: 1.7640 -
accuracy: 0.3577 - val_loss: 1.7247 - val_accuracy: 0.3781
Epoch 4/20
56/56 [=====] - 5s 92ms/step - loss: 1.6752 -
accuracy: 0.3890 - val_loss: 1.6779 - val_accuracy: 0.3893
Epoch 5/20
56/56 [=====] - 5s 92ms/step - loss: 1.6490 -
accuracy: 0.4152 - val_loss: 1.5178 - val_accuracy: 0.4676
Epoch 6/20
56/56 [=====] - 5s 93ms/step - loss: 1.5488 -
accuracy: 0.4431 - val_loss: 1.4979 - val_accuracy: 0.4989
Epoch 7/20
56/56 [=====] - 5s 93ms/step - loss: 1.5659 -
accuracy: 0.4503 - val_loss: 1.5313 - val_accuracy: 0.4452
Epoch 8/20
56/56 [=====] - 5s 92ms/step - loss: 1.4849 -
accuracy: 0.4944 - val_loss: 1.4482 - val_accuracy: 0.5168
Epoch 9/20
56/56 [=====] - 5s 93ms/step - loss: 1.4464 -
accuracy: 0.5017 - val_loss: 1.4376 - val_accuracy: 0.5302
Epoch 10/20
56/56 [=====] - 5s 93ms/step - loss: 1.4438 -
accuracy: 0.4961 - val_loss: 1.4902 - val_accuracy: 0.4631
Epoch 11/20
56/56 [=====] - 5s 93ms/step - loss: 1.4143 -
accuracy: 0.5078 - val_loss: 1.4087 - val_accuracy: 0.5190
Epoch 12/20
56/56 [=====] - 5s 91ms/step - loss: 1.3773 -
accuracy: 0.5134 - val_loss: 1.4523 - val_accuracy: 0.4899
Epoch 13/20
56/56 [=====] - 5s 93ms/step - loss: 1.3375 -
accuracy: 0.5396 - val_loss: 1.4840 - val_accuracy: 0.4720
Epoch 14/20
56/56 [=====] - 5s 91ms/step - loss: 1.3496 -
accuracy: 0.5246 - val_loss: 1.3937 - val_accuracy: 0.5101
Epoch 15/20
56/56 [=====] - 5s 91ms/step - loss: 1.3255 -
accuracy: 0.5340 - val_loss: 1.4377 - val_accuracy: 0.5168
Epoch 16/20
56/56 [=====] - 5s 91ms/step - loss: 1.3123 -
accuracy: 0.5357 - val_loss: 1.6170 - val_accuracy: 0.4989
Epoch 17/20
56/56 [=====] - 5s 91ms/step - loss: 1.3295 -
accuracy: 0.5352 - val_loss: 1.3920 - val_accuracy: 0.5034
Epoch 18/20

```
56/56 [=====] - 5s 91ms/step - loss: 1.3191 -  
accuracy: 0.5530 - val_loss: 1.5669 - val_accuracy: 0.4787  
Epoch 19/20  
56/56 [=====] - 5s 92ms/step - loss: 1.3149 -  
accuracy: 0.5335 - val_loss: 1.3170 - val_accuracy: 0.5347  
Epoch 20/20  
56/56 [=====] - 5s 91ms/step - loss: 1.3137 -  
accuracy: 0.5363 - val_loss: 1.3919 - val_accuracy: 0.4989
```

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()
```



- After using data augmentation and dropout layer overfitting issue is reduce.
- Model Performance is still not increased. Will check the distribution of classes in the training set to check is there have class imbalance.

Class Imbalance Detection

```
def class_distribution_count(directory):
    #count number of image in each classes
    count= []
    for path in pathlib.Path(directory).iterdir():
        if path.is_dir():
            count.append(len([name for name in os.listdir(path)
                               if os.path.isfile(os.path.join(path,
name))]))
```

```

#name of the classes
sub_directory = [name for name in os.listdir(directory)
                  if os.path.isdir(os.path.join(directory, name))]

#return dataframe with image count and class.
return pd.DataFrame(list(zip(sub_directory,count)),columns
=['Class', 'No. of Image'])

```

```

df = class_distribution_count(data_dir_train)
df

```

	Class	No. of Image
0	melanoma	438
1	seborrheic keratosis	77
2	squamous cell carcinoma	181
3	vascular lesion	139
4	pigmented benign keratosis	462
5	nevus	357
6	basal cell carcinoma	376
7	dermatofibroma	95
8	actinic keratosis	114

```

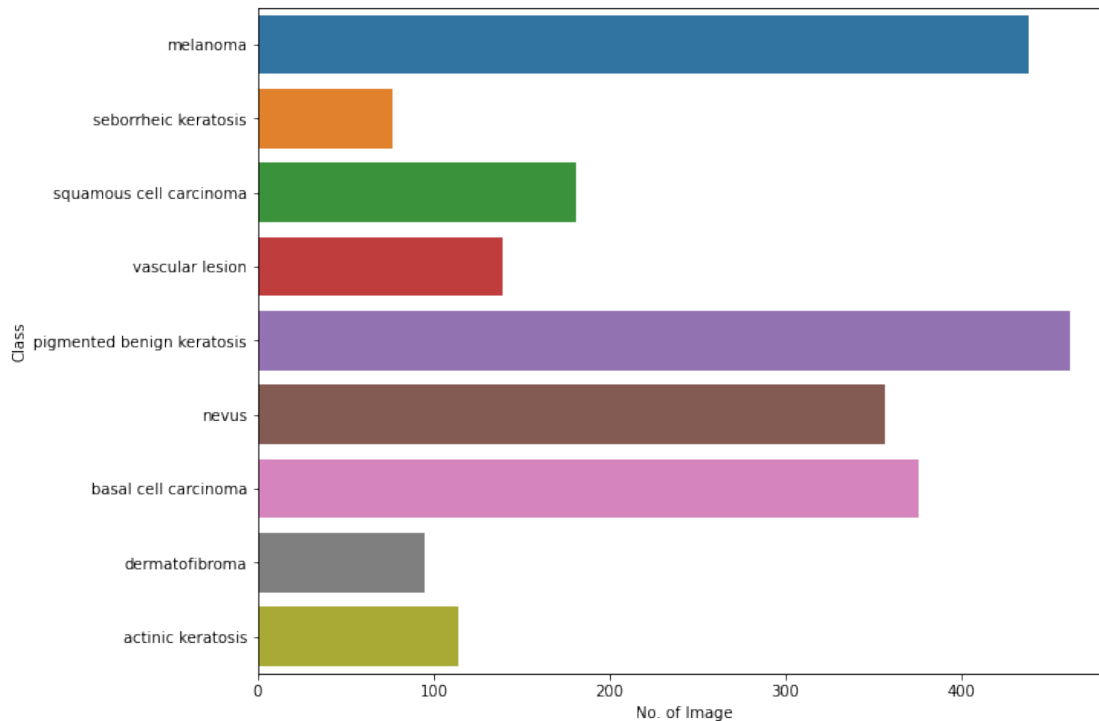
#Visualize the Number of image in each class.
import seaborn as sns
plt.figure(figsize=(10, 8))
sns.barplot(x="No. of Image", y="Class", data=df,
            label="Class")

```

```

<matplotlib.axes._subplots.AxesSubplot at 0x7f7d5879b610>

```



```
!pip install Augmentor
```

Collecting Augmentor

```
Downloading Augmentor-0.2.8-py2.py3-none-any.whl (38 kB)
Requirement already satisfied: future>=0.16.0 in
/usr/local/lib/python3.7/dist-packages (from Augmentor) (0.16.0)
Requirement already satisfied: Pillow>=5.2.0 in
/usr/local/lib/python3.7/dist-packages (from Augmentor) (7.1.2)
Requirement already satisfied: tqdm>=4.9.0 in
/usr/local/lib/python3.7/dist-packages (from Augmentor) (4.62.3)
Requirement already satisfied: numpy>=1.11.0 in
/usr/local/lib/python3.7/dist-packages (from Augmentor) (1.19.5)
Installing collected packages: Augmentor
Successfully installed Augmentor-0.2.8
```

```
path_to_training_dataset="/content/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.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/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/actinic keratosis/output.

Processing <PIL.Image.Image image mode=RGB size=600x450 at
0x7F7DD4F36FD0>: 100%|██████████| 500/500 [00:19<00:00, 25.31
Samples/s]

Initialised with 376 image(s) found.
Output directory set to /content/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/basal cell carcinoma/output.

Processing <PIL.Image.Image image mode=RGB size=600x450 at
0x7F7DD4FE2450>: 100%|██████████| 500/500 [00:20<00:00, 24.98
Samples/s]

Initialised with 95 image(s) found.
Output directory set to /content/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/dermatofibroma/output.

Processing <PIL.Image.Image image mode=RGB size=600x450 at
0x7F7E4F34FBD0>: 100%|██████████| 500/500 [00:19<00:00, 25.43
Samples/s]

Initialised with 438 image(s) found.
Output directory set to /content/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/melanoma/output.

Processing <PIL.Image.Image image mode=RGB size=3072x2304 at
0x7F7DD4D7BFD0>: 100%|██████████| 500/500 [01:53<00:00, 4.40
Samples/s]

Initialised with 357 image(s) found.
Output directory set to /content/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/nevus/output.

Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB
size=1504x1129 at 0x7F7E4F4DE050>: 100%|██████████| 500/500
[01:14<00:00, 6.71 Samples/s]

Initialised with 462 image(s) found.
Output directory set to /content/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/pigmented benign keratosis/output.

Processing <PIL.Image.Image image mode=RGB size=600x450 at
0x7F7D587ECE90>: 100%|██████████| 500/500 [00:20<00:00, 24.64
Samples/s]

Initialised with 77 image(s) found.
Output directory set to /content/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/seborrheic keratosis/output.

Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB
size=1024x768 at 0x7F7D57FCFED0>: 100%|██████████| 500/500
[00:47<00:00, 10.54 Samples/s]

Initialised with 181 image(s) found.
Output directory set to /content/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/squamous cell carcinoma/output.

Processing <PIL.Image.Image image mode=RGB size=600x450 at
0x7F7EC49553D0>: 100%|██████████| 500/500 [00:19<00:00, 25.63
Samples/s]

Initialised with 139 image(s) found.
Output directory set to /content/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/vascular lesion/output.

Processing <PIL.Image.Image image mode=RGB size=600x450 at
0x7F7E4F2C73D0>: 100%|██████████| 500/500 [00:19<00:00, 25.70
Samples/s]

```
#Count total number of image generated by Augmentor.
image_count_train = len(list(data_dir_train.glob('*output/*.jpg')))
print(image_count_train)
```

4500

```
from glob import glob
path_list = [x for x in glob(os.path.join(data_dir_train,
    '*', 'output', '*.jpg'))]
#path_list

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

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

```
#dataframe that store path and label of the images generated by
Augmentor
```

```
df2 = pd.DataFrame(list(dataframe_dict_new.items()), columns =
['Path', 'Label'])
```

```
#label count.
```

```
df2['Label'].value_counts()
```

pigmented benign keratosis	500
seborrheic keratosis	500
dermatofibroma	500
melanoma	500
squamous cell carcinoma	500
actinic keratosis	500
vascular lesion	500
nevus	500
basal cell carcinoma	500
Name: Label, dtype: int64	

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.

Train the model on the data created using Augmentor

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

Create a training dataset

```
data_dir_train="/content/Skin cancer ISIC The International Skin  
Imaging Collaboration/Train/"
```

#Training dataset.

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(  
    data_dir_train,  
    seed=123,  
    validation_split = 0.2,    #20% fraction of data to reserve for  
validation.  
    subset = "training",  
    image_size=(img_height, img_width), label_mode='categorical',  
    #label_mode='categorical' means that the labels are encoded as a  
categorical vector  
    batch_size=batch_size)
```

Found 6739 files belonging to 9 classes.
Using 5392 files for training.

Create a validation dataset

#Validation dataset.

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(  
    data_dir_train,  
    seed=123,  
    validation_split = 0.2,  
    subset = "validation",  
    image_size=(img_height, img_width), label_mode='categorical',  
    #label_mode='categorical' means that the labels are encoded as a  
categorical vector  
    batch_size=batch_size)
```

Found 6739 files belonging to 9 classes.
Using 1347 files for validation.

Create model

#Model

```
model3 = Sequential()
```

```
model3.add(rescale)    #Rescaling Layer
```

#First Convulation layer

```
model3.add(layers.Conv2D(32, kernel_size=(2,2), activation='relu'))
model3.add(layers.MaxPool2D(pool_size=(2,2)))
model3.add(layers.Dropout(0.25))
```

#Second Convulation Layer

```
model3.add(layers.Conv2D(64, kernel_size=(2,2), activation='relu'))
model3.add(layers.MaxPool2D(pool_size=(2,2)))
model3.add(layers.Dropout(0.25))
```

#Third Convulation Layer

```
model3.add(layers.Conv2D(128, kernel_size=(2,2), activation='relu'))
model3.add(layers.MaxPool2D(pool_size=(2,2)))
```

#Flatten Layer

```
model3.add(layers.Flatten())
```

#Dense Layer

```
model3.add(layers.Dense(512, activation='relu'))
```

#Dropout layer

```
model3.add(layers.Dropout(0.25))
```

#Batch normalization: is a method used to make artificial neural networks faster and more stable through normalization of the layers' inputs by re-centering and re-scaling.

```
model3.add(layers.BatchNormalization())
```

#Dense Layer

```
model3.add(layers.Dense(128, activation='relu'))
```

#Dropout layer with 50% Fraction of the input units to drop.

```
model3.add(layers.Dropout(0.50))
```

#Batch normalization

```
model3.add(layers.BatchNormalization())
```

#Dense layer with Softmax activation function.

```
model3.add(layers.Dense(len(class_names), activation='softmax'))
```

Compile Model

```
model3.compile(optimizer='Adam',
               loss="categorical_crossentropy",
               metrics=['accuracy'])
```

Train your model

```
epochs = 50
history =
model3.fit(train_ds, epochs=epochs, validation_data=val_ds, verbose=1)
```

Epoch 1/50
169/169 [=====] - 33s 181ms/step - loss:
2.4487 - accuracy: 0.1825 - val_loss: 4.1381 - val_accuracy: 0.1359
Epoch 2/50
169/169 [=====] - 32s 186ms/step - loss:
1.8023 - accuracy: 0.3418 - val_loss: 2.4640 - val_accuracy: 0.1693
Epoch 3/50
169/169 [=====] - 32s 186ms/step - loss:
1.5857 - accuracy: 0.3986 - val_loss: 1.9945 - val_accuracy: 0.2517
Epoch 4/50
169/169 [=====] - 33s 187ms/step - loss:
1.5265 - accuracy: 0.4201 - val_loss: 1.5896 - val_accuracy: 0.3831
Epoch 5/50
169/169 [=====] - 33s 187ms/step - loss:
1.4334 - accuracy: 0.4609 - val_loss: 1.3672 - val_accuracy: 0.4937
Epoch 6/50
169/169 [=====] - 32s 185ms/step - loss:
1.3552 - accuracy: 0.4800 - val_loss: 1.3817 - val_accuracy: 0.4722
Epoch 7/50
169/169 [=====] - 32s 187ms/step - loss:
1.2997 - accuracy: 0.5056 - val_loss: 1.4254 - val_accuracy: 0.4640
Epoch 8/50
169/169 [=====] - 32s 185ms/step - loss:
1.2361 - accuracy: 0.5254 - val_loss: 1.2275 - val_accuracy: 0.5449
Epoch 9/50
169/169 [=====] - 32s 186ms/step - loss:
1.2146 - accuracy: 0.5493 - val_loss: 1.2341 - val_accuracy: 0.5301
Epoch 10/50
169/169 [=====] - 32s 183ms/step - loss:
1.1354 - accuracy: 0.5695 - val_loss: 1.2274 - val_accuracy: 0.5271
Epoch 11/50
169/169 [=====] - 32s 183ms/step - loss:
1.0676 - accuracy: 0.5972 - val_loss: 1.1729 - val_accuracy: 0.5612
Epoch 12/50
169/169 [=====] - 32s 184ms/step - loss:
1.0303 - accuracy: 0.6191 - val_loss: 1.1371 - val_accuracy: 0.5672
Epoch 13/50
169/169 [=====] - 32s 183ms/step - loss:
0.9785 - accuracy: 0.6404 - val_loss: 0.9895 - val_accuracy: 0.6310
Epoch 14/50
169/169 [=====] - 33s 187ms/step - loss:
0.8914 - accuracy: 0.6647 - val_loss: 1.0143 - val_accuracy: 0.6058
Epoch 15/50
169/169 [=====] - 31s 179ms/step - loss:
0.8498 - accuracy: 0.6843 - val_loss: 0.9272 - val_accuracy: 0.6496
Epoch 16/50
169/169 [=====] - 31s 181ms/step - loss:
0.7827 - accuracy: 0.7027 - val_loss: 0.9905 - val_accuracy: 0.6429
Epoch 17/50
169/169 [=====] - 32s 182ms/step - loss:

0.7659 - accuracy: 0.7111 - val_loss: 0.8056 - val_accuracy: 0.6986
Epoch 18/50
169/169 [=====] - 32s 184ms/step - loss:
0.6931 - accuracy: 0.7378 - val_loss: 0.8763 - val_accuracy: 0.6704
Epoch 19/50
169/169 [=====] - 32s 184ms/step - loss:
0.6807 - accuracy: 0.7489 - val_loss: 0.8052 - val_accuracy: 0.7030
Epoch 20/50
169/169 [=====] - 32s 183ms/step - loss:
0.6751 - accuracy: 0.7522 - val_loss: 0.8132 - val_accuracy: 0.6875
Epoch 21/50
169/169 [=====] - 32s 182ms/step - loss:
0.5856 - accuracy: 0.7838 - val_loss: 0.7683 - val_accuracy: 0.7179
Epoch 22/50
169/169 [=====] - 33s 187ms/step - loss:
0.5487 - accuracy: 0.8012 - val_loss: 1.1119 - val_accuracy: 0.6236
Epoch 23/50
169/169 [=====] - 32s 184ms/step - loss:
0.5309 - accuracy: 0.8023 - val_loss: 0.8205 - val_accuracy: 0.6941
Epoch 24/50
169/169 [=====] - 32s 185ms/step - loss:
0.5050 - accuracy: 0.8121 - val_loss: 0.6481 - val_accuracy: 0.7595
Epoch 25/50
169/169 [=====] - 33s 187ms/step - loss:
0.4980 - accuracy: 0.8157 - val_loss: 0.7069 - val_accuracy: 0.7313
Epoch 26/50
169/169 [=====] - 32s 186ms/step - loss:
0.4924 - accuracy: 0.8123 - val_loss: 0.7194 - val_accuracy: 0.7416
Epoch 27/50
169/169 [=====] - 31s 179ms/step - loss:
0.4326 - accuracy: 0.8374 - val_loss: 0.5811 - val_accuracy: 0.7810
Epoch 28/50
169/169 [=====] - 32s 182ms/step - loss:
0.4078 - accuracy: 0.8513 - val_loss: 0.6638 - val_accuracy: 0.7491
Epoch 29/50
169/169 [=====] - 32s 183ms/step - loss:
0.4247 - accuracy: 0.8464 - val_loss: 0.6210 - val_accuracy: 0.7780
Epoch 30/50
169/169 [=====] - 32s 186ms/step - loss:
0.3875 - accuracy: 0.8572 - val_loss: 0.5844 - val_accuracy: 0.7929
Epoch 31/50
169/169 [=====] - 32s 185ms/step - loss:
0.3672 - accuracy: 0.8672 - val_loss: 0.6045 - val_accuracy: 0.7892
Epoch 32/50
169/169 [=====] - 32s 183ms/step - loss:
0.3238 - accuracy: 0.8809 - val_loss: 0.5658 - val_accuracy: 0.7877
Epoch 33/50
169/169 [=====] - 32s 184ms/step - loss:
0.3468 - accuracy: 0.8718 - val_loss: 0.6590 - val_accuracy: 0.7810
Epoch 34/50

169/169 [=====] - 32s 183ms/step - loss:
0.3277 - accuracy: 0.8820 - val_loss: 0.7535 - val_accuracy: 0.7350
Epoch 35/50
169/169 [=====] - 32s 184ms/step - loss:
0.3568 - accuracy: 0.8644 - val_loss: 0.6257 - val_accuracy: 0.7788
Epoch 36/50
169/169 [=====] - 32s 184ms/step - loss:
0.3045 - accuracy: 0.8843 - val_loss: 0.7786 - val_accuracy: 0.7372
Epoch 37/50
169/169 [=====] - 32s 184ms/step - loss:
0.2791 - accuracy: 0.8939 - val_loss: 0.5336 - val_accuracy: 0.8211
Epoch 38/50
169/169 [=====] - 32s 184ms/step - loss:
0.2966 - accuracy: 0.8874 - val_loss: 0.5314 - val_accuracy: 0.8144
Epoch 39/50
169/169 [=====] - 32s 184ms/step - loss:
0.2881 - accuracy: 0.8921 - val_loss: 0.5872 - val_accuracy: 0.8144
Epoch 40/50
169/169 [=====] - 32s 181ms/step - loss:
0.2850 - accuracy: 0.8971 - val_loss: 0.5198 - val_accuracy: 0.8233
Epoch 41/50
169/169 [=====] - 32s 187ms/step - loss:
0.3055 - accuracy: 0.8826 - val_loss: 0.5687 - val_accuracy: 0.8174
Epoch 42/50
169/169 [=====] - 32s 183ms/step - loss:
0.2729 - accuracy: 0.8971 - val_loss: 0.5346 - val_accuracy: 0.8203
Epoch 43/50
169/169 [=====] - 31s 181ms/step - loss:
0.2646 - accuracy: 0.8999 - val_loss: 0.5081 - val_accuracy: 0.8337
Epoch 44/50
169/169 [=====] - 32s 182ms/step - loss:
0.3819 - accuracy: 0.8624 - val_loss: 0.5588 - val_accuracy: 0.8137
Epoch 45/50
169/169 [=====] - 32s 182ms/step - loss:
0.3193 - accuracy: 0.8789 - val_loss: 0.6199 - val_accuracy: 0.7973
Epoch 46/50
169/169 [=====] - 31s 181ms/step - loss:
0.2699 - accuracy: 0.9006 - val_loss: 0.5255 - val_accuracy: 0.8382
Epoch 47/50
169/169 [=====] - 31s 179ms/step - loss:
0.2617 - accuracy: 0.9045 - val_loss: 0.5703 - val_accuracy: 0.8077
Epoch 48/50
169/169 [=====] - 32s 181ms/step - loss:
0.2378 - accuracy: 0.9078 - val_loss: 0.5595 - val_accuracy: 0.8181
Epoch 49/50
169/169 [=====] - 32s 181ms/step - loss:
0.2516 - accuracy: 0.9050 - val_loss: 0.5245 - val_accuracy: 0.8293
Epoch 50/50
169/169 [=====] - 32s 186ms/step - loss:
0.2257 - accuracy: 0.9121 - val_loss: 0.5234 - val_accuracy: 0.8337

Visualize the model results

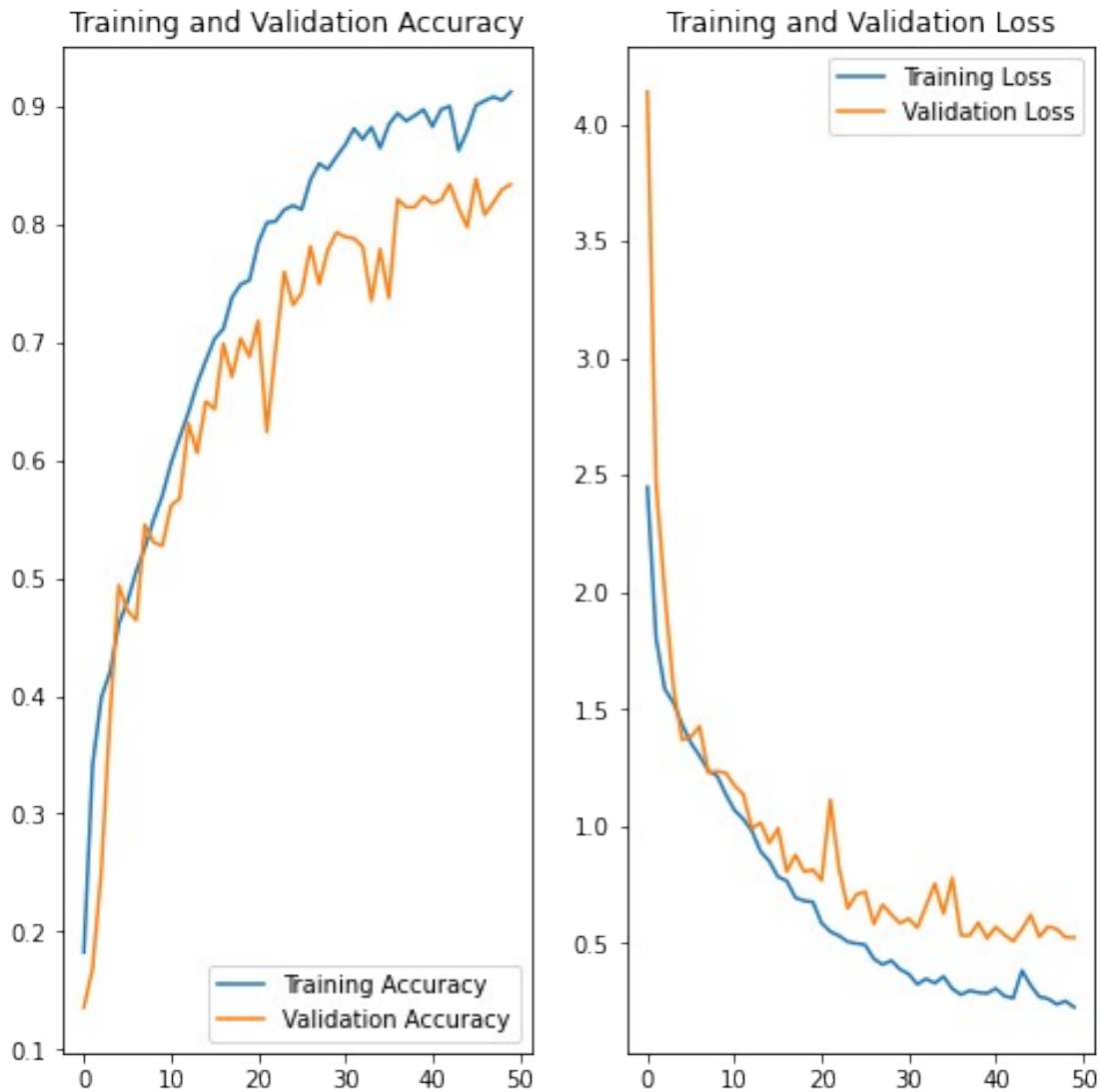
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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()
```

- As per the final model (model3) Training accuracy and validation accuracy increases.
- Model overfitting issue is solved.
- Class rebalance helps in augmentation and achieving the best Training and validation accuracy.