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.

Build a multiclass classification model using a custom convolutional neural network in TensorFlow.

Importing Skin Cancer Data ¶

Step 1:- Importing all the important libraries

In [1]:

```
#import the required libraries
import pathlib
import os, cv2

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import PIL

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

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint,EarlyStopping

from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, BatchNormalizati

import warnings
warnings.filterwarnings("ignore")
```

In [2]:

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

## Data set Ref : https://towardsdatascience.com/downloading-datasets-into-google-drive-v
```

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

In [3]:

```
#unziping the dataset
!unzip "/content/gdrive/MyDrive/CNN_assignment.zip" > /dev/null
```

Reading Data and Understanding it

Step 2:- Defining the path for train and test images

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.

In [4]:

```
# Defining the path for train and test images,
dataset_train = pathlib.Path("/content/Skin cancer ISIC The International Skin Imaging Co
dataset_test = pathlib.Path("/content/Skin cancer ISIC The International Skin Imaging Col
```

In [5]:

```
# Training Image Count
image_count_train = len(list(dataset_train.glob('*/*.jpg')))
print(image_count_train)
```

```
2239
```

In [6]:

```
# Testing Image Count
image_count_test = len(list(dataset_test.glob('*/*.jpg')))
print(image_count_test)
```

```
118
```

Load using keras.preprocessing

Step 3:- Let's load these images off disk using the helpful image_dataset_from_directory utility.

In [7]:

```
# Defined some parameters for the loader
batch_size = 32
img_height = 180
img_width = 180
```

Note:- Here we are using 80% of the images for training, and 20% for validation.

In [8]:

```
# Loading the training data
# using seed=123 while creating dataset using tf.keras.preprocessing.image_dataset_from_d
# resizing images to the size img_height*img_width, while writting the dataset
train_ds = tf.keras.preprocessing.image_dataset_from_directory(dataset_train,
                                                               seed=123,
                                                               validation_split=0.2,
                                                               image_size=(img_height,img
                                                               batch_size=batch_size,
                                                               color_mode='rgb',
                                                               subset='training')
```

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

In [9]:

```
# Loading the validation data
# using seed=123 while creating dataset using tf.keras.preprocessing.image_dataset_from_d
# resizing images to the size img_height*img_width, while writting the dataset
val_ds = tf.keras.preprocessing.image_dataset_from_directory(dataset_train,
                                                             seed=123,
                                                             validation_split=0.2,
                                                             image_size=(img_height,img_w
                                                             batch_size=batch_size,
                                                             color_mode='rgb',
                                                             subset='validation')
```

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

In [10]:

```
# Loading the testing data
# using seed=123 while creating dataset using tf.keras.preprocessing.image_dataset_from_d
# resizing images to the size img_height*img_width, while writting the dataset
test_ds = tf.keras.preprocessing.image_dataset_from_directory(dataset_test,
                                                              seed=123,
                                                              image_size=(img_height,img_w
                                                              batch_size=batch_size,
                                                              color_mode='rgb')
```

```
Found 118 files belonging to 9 classes.
```

Here we are listing out all the classes of skin cancer and store them in a list.

In [11]:

```
# These correspond to the directory names in alphabetical order.
class_names = train_ds.class_names
print(class_names)
```

```
['actinic keratosis', 'basal cell carcinoma', 'dermatofibroma', 'melanom
a', 'nevus', 'pigmented benign keratosis', 'seborrheic keratosis', 'squamo
us cell carcinoma', 'vascular lesion']
```

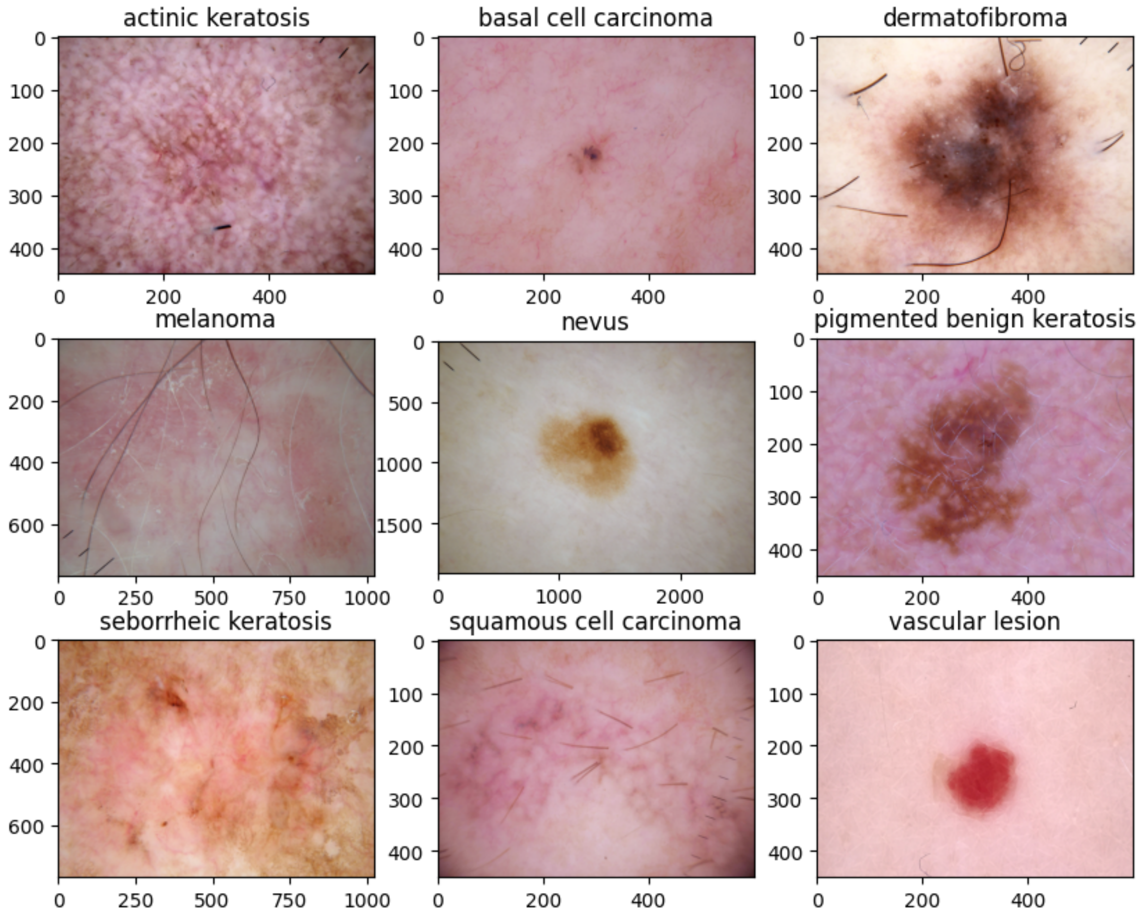
Visualize the data

Step 4:- Todo, create a code to visualize one instance of all the nine classes present in the dataset

In [12]:

```
import matplotlib.pyplot as plt

### your code goes here, you can use training or validation data to visualize
plt.figure(figsize=(10,8))
for i in range(len(class_names)):
  plt.subplot(3,3,i+1)
  image= plt.imread(str(list(dataset_train.glob(class_names[i]+'/*.jpg'))[1]))
  plt.title(class_names[i])
  plt.imshow(image)
```



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.

In [13]:

```
# Configuing the dataset for the performance

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)

# `Dataset.cache()` keeps the images in memory after they're loaded off disk during the f
# `Dataset.prefetch()` overlaps data preprocessing and model execution while training.
```

Creating a Model

Note:- Creating 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]`

Model 1

Step 5:- Model Building & training :

Step i : Creating a CNN model, which can accurately detect 9 classes present in the dataset. While building the model, rescaling images to normalize pixel values between (0,1).

Step ii : Choosing an appropriate optimiser and loss function for model training.

Step iii : Training the model for ~20 epochs.

Step iv : Plotting Graph for findings after the model fit to check if there is any evidence of model overfit or underfit.

In [14]:

```
# CNN Model
model=models.Sequential()
# scaling the pixel values from 0-255 to 0-1
model.add(layers.Rescaling(scale=1./255,input_shape=(180,180,3)))

# Convolution layer with 64 features, 3x3 filter and relu activation with 2x2 pooling
model.add(layers.Conv2D(64,(3,3),padding = 'same',activation='relu'))
model.add(layers.MaxPooling2D())

# Convolution layer with 128 features, 3x3 filter and relu activation with 2x2 pooling
model.add(layers.Conv2D(128,(3,3),padding = 'same',activation='relu'))
model.add(layers.MaxPooling2D())

model.add(layers.Flatten())
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dense(9,activation='softmax'))
```

In [15]:

```
# Compiling the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 rescaling (Rescaling)       (None, 180, 180, 3)       0         
                                                                 
 conv2d (Conv2D)             (None, 180, 180, 64)      1792      
                                                                 
 max_pooling2d (MaxPooling2  (None, 90, 90, 64)        0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (None, 90, 90, 128)       73856     
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 45, 45, 128)       0         
 g2D)                                                            
                                                                 
 flatten (Flatten)           (None, 259200)            0         
                                                                 
 dense (Dense)               (None, 256)               66355456  
                                                                 
 dense_1 (Dense)             (None, 9)                 2313      
                                                                 
=================================================================
Total params: 66433417 (253.42 MB)
Trainable params: 66433417 (253.42 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [16]:

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

```
Epoch 1/20
56/56 [==============================] - 30s 119ms/step - loss: 3.6686 - a
ccuracy: 0.2271 - val_loss: 2.1130 - val_accuracy: 0.2975
Epoch 2/20
56/56 [==============================] - 4s 66ms/step - loss: 1.8029 - acc
uracy: 0.3599 - val_loss: 1.6467 - val_accuracy: 0.4206
Epoch 3/20
56/56 [==============================] - 4s 67ms/step - loss: 1.5229 - acc
uracy: 0.4526 - val_loss: 1.5535 - val_accuracy: 0.4698
Epoch 4/20
56/56 [==============================] - 4s 67ms/step - loss: 1.4124 - acc
uracy: 0.5123 - val_loss: 1.7493 - val_accuracy: 0.4698
Epoch 5/20
56/56 [==============================] - 4s 67ms/step - loss: 1.2951 - acc
uracy: 0.5469 - val_loss: 1.4310 - val_accuracy: 0.5615
Epoch 6/20
56/56 [==============================] - 4s 67ms/step - loss: 1.1781 - acc
uracy: 0.5809 - val_loss: 1.6459 - val_accuracy: 0.4922
Epoch 7/20
56/56 [==============================] - 4s 68ms/step - loss: 1.0319 - acc
uracy: 0.6278 - val_loss: 1.6226 - val_accuracy: 0.5257
Epoch 8/20
56/56 [==============================] - 4s 67ms/step - loss: 0.9509 - acc
uracy: 0.6523 - val_loss: 1.5594 - val_accuracy: 0.5257
Epoch 9/20
56/56 [==============================] - 4s 67ms/step - loss: 0.8339 - acc
uracy: 0.7026 - val_loss: 1.6372 - val_accuracy: 0.5101
Epoch 10/20
56/56 [==============================] - 4s 68ms/step - loss: 0.7210 - acc
uracy: 0.7439 - val_loss: 1.7543 - val_accuracy: 0.5011
Epoch 11/20
56/56 [==============================] - 4s 68ms/step - loss: 0.5974 - acc
uracy: 0.7824 - val_loss: 1.8170 - val_accuracy: 0.5056
Epoch 12/20
56/56 [==============================] - 4s 67ms/step - loss: 0.5846 - acc
uracy: 0.8092 - val_loss: 1.8750 - val_accuracy: 0.5369
Epoch 13/20
56/56 [==============================] - 4s 68ms/step - loss: 0.4701 - acc
uracy: 0.8292 - val_loss: 2.1649 - val_accuracy: 0.5034
Epoch 14/20
56/56 [==============================] - 4s 70ms/step - loss: 0.4753 - acc
uracy: 0.8371 - val_loss: 2.1784 - val_accuracy: 0.4609
Epoch 15/20
56/56 [==============================] - 4s 67ms/step - loss: 0.3794 - acc
uracy: 0.8638 - val_loss: 2.0870 - val_accuracy: 0.4922
Epoch 16/20
56/56 [==============================] - 4s 67ms/step - loss: 0.3514 - acc
uracy: 0.8633 - val_loss: 2.1906 - val_accuracy: 0.5056
Epoch 17/20
56/56 [==============================] - 4s 70ms/step - loss: 0.2898 - acc
uracy: 0.8929 - val_loss: 2.4301 - val_accuracy: 0.5391
Epoch 18/20
56/56 [==============================] - 4s 69ms/step - loss: 0.2487 - acc
uracy: 0.9102 - val_loss: 2.2169 - val_accuracy: 0.4855
Epoch 19/20
56/56 [==============================] - 4s 67ms/step - loss: 0.2589 - acc
uracy: 0.8968 - val_loss: 2.3014 - val_accuracy: 0.5391
Epoch 20/20
56/56 [==============================] - 4s 67ms/step - loss: 0.2344 - acc
uracy: 0.9129 - val_loss: 2.3308 - val_accuracy: 0.5436
```

In [17]:

```
#Plotting the graph of outcome
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()
```

Findings from the Graph:

- As the training accuracy increases linearly over time, where as the validation accuracy stall at ~54% accuracy in training process.
- As the training loss dereases with epochs the validation loss increases
- The plots show that training accuracy and validation accuracy are off by large margins, and the model has achieved around **~54%** accuracy on the validation set.
- The difference in accuracy between training and validation accuracy is **noticeable** which is a sign of overfitting.

Choosing an appropriate data augmentation strategy to resolve underfitting/overfitting

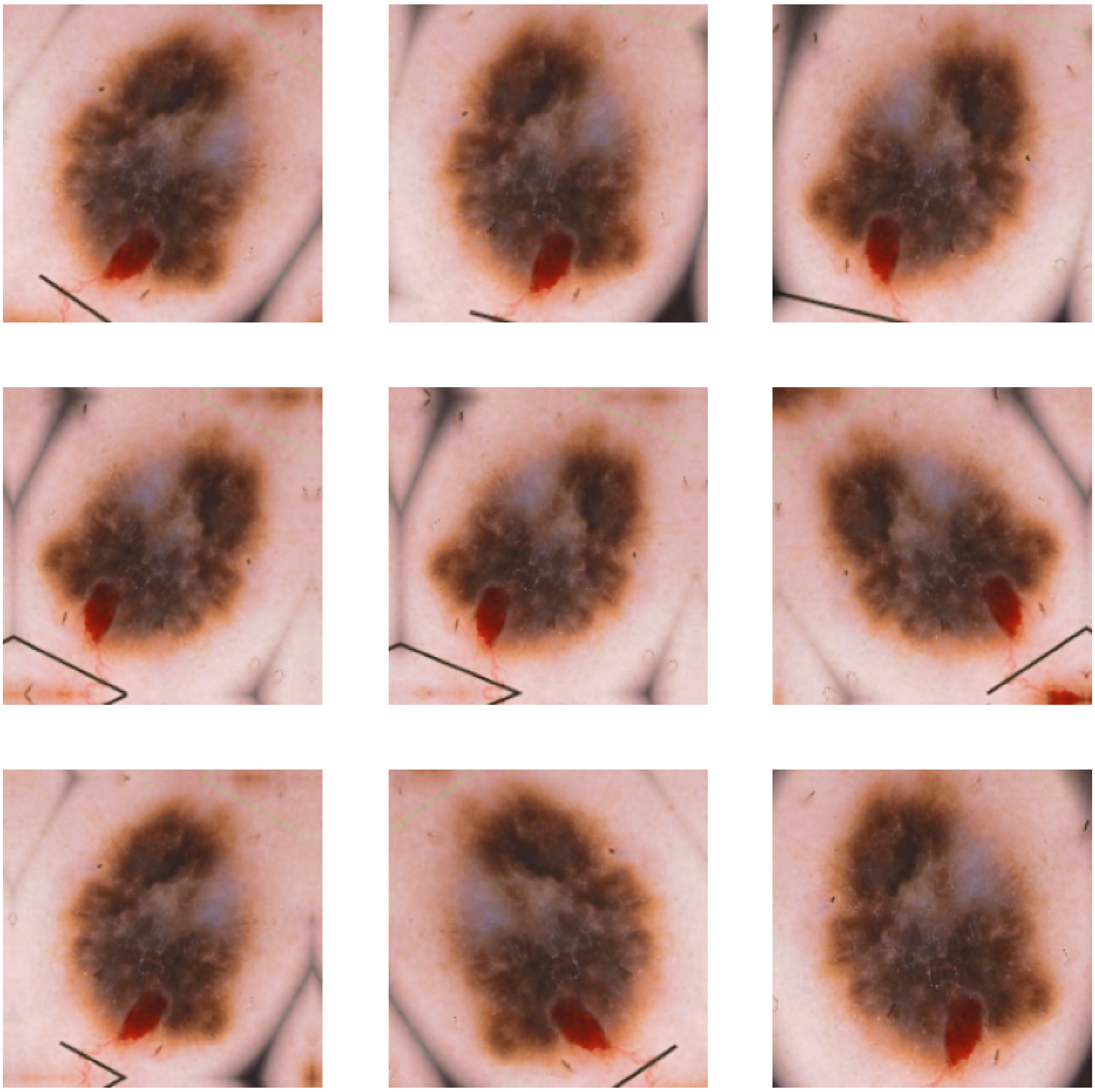
Note:- Overfitting generally occurs when there are a small number of training examples. Data augmentation takes the approach of generating additional training data from your existing examples by augmenting them using random transformations that yield believable-looking images. This helps expose the model to more aspects of the data and generalize better.

In [18]:

```
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",input_shape=(img_height,img_width,3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)
```

In [19]:

```
# visualizing how your augmentation strategy works for one instance of training 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

Step 6:- Model Building & training on the augmented data :

Step i :Creating a CNN model, which can accurately detect 9 classes present in the dataset. While building the model, rescaling images to normalize pixel values between (0,1).

Step ii :Choosing an appropriate optimiser and loss function for model training.

Step iii :Training the model for ~20 epochs.

Step iv :Plotting Graph for findings after the model fit to check if there is any evidence of model overfit or underfit.

In [20]:

```
# CNN Model
model=models.Sequential()
# scaling the pixel values from 0-255 to 0-1
model.add(layers.Rescaling(scale=1./255,input_shape=(180,180,3)))

# adding the augmentation layer before the convolution layer
model.add(data_augmentation)

# Convolution layer with 64 features, 3x3 filter and relu activation with 2x2 pooling
model.add(layers.Conv2D(64,(3,3),padding = 'same',activation='relu'))
model.add(layers.MaxPooling2D())

# Convolution layer with 128 features, 3x3 filter and relu activation with 2x2 pooling
model.add(layers.Conv2D(128,(3,3),padding = 'same',activation='relu'))
model.add(layers.MaxPooling2D())

model.add(layers.Flatten())
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dense(9,activation='softmax'))
```

In [21]:

```
# Compiling the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0         
                                                                 
 sequential_1 (Sequential)   (None, 180, 180, 3)       0         
                                                                 
 conv2d_2 (Conv2D)           (None, 180, 180, 64)      1792      
                                                                 
 max_pooling2d_2 (MaxPoolin  (None, 90, 90, 64)        0         
 g2D)                                                            
                                                                 
 conv2d_3 (Conv2D)           (None, 90, 90, 128)       73856     
                                                                 
 max_pooling2d_3 (MaxPoolin  (None, 45, 45, 128)       0         
 g2D)                                                            
                                                                 
 flatten_1 (Flatten)         (None, 259200)            0         
                                                                 
 dense_2 (Dense)             (None, 256)               66355456  
                                                                 
 dense_3 (Dense)             (None, 9)                 2313      
                                                                 
=================================================================
Total params: 66433417 (253.42 MB)
Trainable params: 66433417 (253.42 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [22]:

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

```
Epoch 1/20
56/56 [==============================] - 6s 73ms/step - loss: 3.4813 - acc
uracy: 0.2204 - val_loss: 1.8785 - val_accuracy: 0.3132
Epoch 2/20
56/56 [==============================] - 4s 69ms/step - loss: 1.7863 - acc
uracy: 0.3460 - val_loss: 2.0024 - val_accuracy: 0.3490
Epoch 3/20
56/56 [==============================] - 4s 72ms/step - loss: 1.6703 - acc
uracy: 0.3895 - val_loss: 1.5889 - val_accuracy: 0.4139
Epoch 4/20
56/56 [==============================] - 4s 70ms/step - loss: 1.5283 - acc
uracy: 0.4576 - val_loss: 1.4816 - val_accuracy: 0.4922
Epoch 5/20
56/56 [==============================] - 4s 70ms/step - loss: 1.4496 - acc
uracy: 0.4860 - val_loss: 1.5125 - val_accuracy: 0.4497
Epoch 6/20
56/56 [==============================] - 4s 71ms/step - loss: 1.3934 - acc
uracy: 0.5073 - val_loss: 1.3750 - val_accuracy: 0.5280
Epoch 7/20
56/56 [==============================] - 4s 69ms/step - loss: 1.3080 - acc
uracy: 0.5340 - val_loss: 1.3912 - val_accuracy: 0.5168
Epoch 8/20
56/56 [==============================] - 4s 69ms/step - loss: 1.2870 - acc
uracy: 0.5396 - val_loss: 1.3837 - val_accuracy: 0.5168
Epoch 9/20
56/56 [==============================] - 4s 72ms/step - loss: 1.2752 - acc
uracy: 0.5396 - val_loss: 1.3947 - val_accuracy: 0.5123
Epoch 10/20
56/56 [==============================] - 4s 70ms/step - loss: 1.2714 - acc
uracy: 0.5413 - val_loss: 1.4499 - val_accuracy: 0.5347
Epoch 11/20
56/56 [==============================] - 4s 69ms/step - loss: 1.2556 - acc
uracy: 0.5575 - val_loss: 1.3693 - val_accuracy: 0.5324
Epoch 12/20
56/56 [==============================] - 4s 71ms/step - loss: 1.1828 - acc
uracy: 0.5737 - val_loss: 1.5253 - val_accuracy: 0.5078
Epoch 13/20
56/56 [==============================] - 4s 70ms/step - loss: 1.1783 - acc
uracy: 0.5725 - val_loss: 1.3796 - val_accuracy: 0.5503
Epoch 14/20
56/56 [==============================] - 4s 69ms/step - loss: 1.1567 - acc
uracy: 0.5837 - val_loss: 1.4155 - val_accuracy: 0.5101
Epoch 15/20
56/56 [==============================] - 4s 71ms/step - loss: 1.2430 - acc
uracy: 0.5619 - val_loss: 1.4175 - val_accuracy: 0.5347
Epoch 16/20
56/56 [==============================] - 4s 70ms/step - loss: 1.1106 - acc
uracy: 0.6060 - val_loss: 1.4434 - val_accuracy: 0.5369
Epoch 17/20
56/56 [==============================] - 4s 69ms/step - loss: 1.1140 - acc
uracy: 0.5988 - val_loss: 1.3365 - val_accuracy: 0.5459
Epoch 18/20
56/56 [==============================] - 4s 70ms/step - loss: 1.0574 - acc
uracy: 0.6183 - val_loss: 1.3891 - val_accuracy: 0.5369
Epoch 19/20
56/56 [==============================] - 4s 73ms/step - loss: 1.0800 - acc
uracy: 0.6088 - val_loss: 1.4504 - val_accuracy: 0.5213
Epoch 20/20
56/56 [==============================] - 4s 70ms/step - loss: 1.0308 - acc
uracy: 0.6200 - val_loss: 1.4217 - val_accuracy: 0.5280
```

In [23]:

```
#plotting the graph of outcome
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()
```

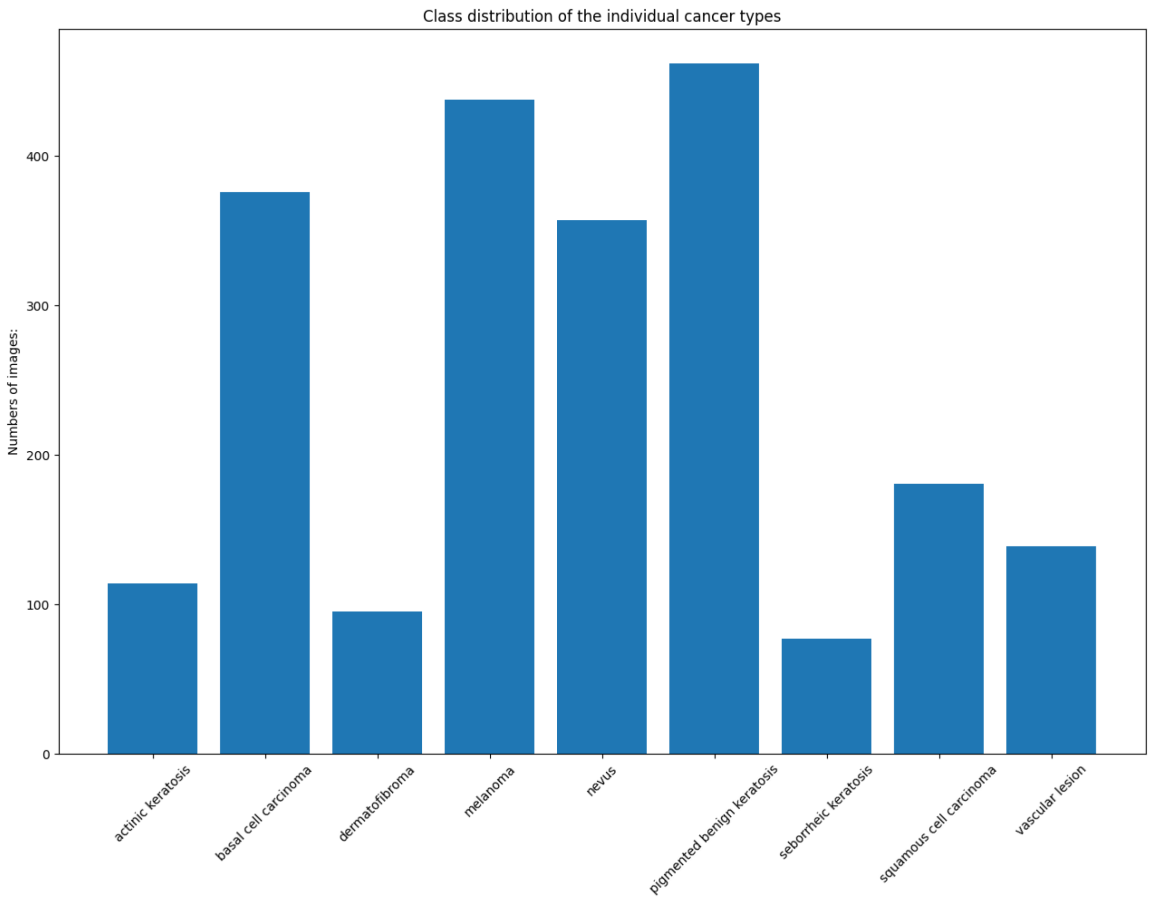
Findings from the graph:

- As the training accuracy increases linearly over time, where as the validation accuracy increases and stall at ~52% accuracy in training process.
- As the training loss decreases with epochs the validation loss decreases and stalls.
- The plots show that gap between training accuracy and validation accuracy have decreased from previous model, and it has achieved around **~52%** accuracy on the validation set.
- The difference in accuracy between training and validation accuracy is still **slightly noticeable** which is a sign of overfitting.

In [24]:

```
# Plot the images to check if all the cancer types are equally distributed
fig = plt.figure(figsize=(12,8))
ax = fig.add_axes([0,0,1,1])
x=[]
y=[]
for i in range(len(class_names)):
  x.append(class_names[i])
  y.append(len(list(dataset_train.glob(class_names[i]+'/*.jpg'))))

ax.bar(x,y)
ax.set_ylabel('Numbers of images:')
ax.set_title('Class distribution of the individual cancer types')
plt.xticks(rotation=45)
plt.show()
```



Model 3

Step 7:- Model Building & training on the augmented data with dropout :

Step i : Creating a CNN model, which can accurately detect 9 classes present in the dataset. While building the model, rescaling images to normalize pixel values between (0,1).

Step ii :Choosing an appropriate optimiser and loss function for model training.

Step iii :Training the model for ~20 epochs.

Step iv :Plotting Graph for findings after the model fit to check if there is any evidence of model overfit or underfit.

In [25]:

```
# CNN Model
model=models.Sequential()
# scaling the pixel values from 0-255 to 0-1
model.add(layers.Rescaling(scale=1./255,input_shape=(180,180,3)))
model.add(data_augmentation)

# Convolution layer with 64 features, 3x3 filter and relu activation with 2x2 pooling
model.add(layers.Conv2D(64,(3,3),padding = 'same',activation='relu'))
model.add(layers.MaxPooling2D())

# Convolution layer with 128 features, 3x3 filter and relu activation with 2x2 pooling
model.add(layers.Conv2D(128,(3,3),padding = 'same',activation='relu'))
model.add(layers.MaxPooling2D())
#adding a 20% dropout after the convolution layers
model.add(layers.Dropout(0.2))

model.add(layers.Flatten())
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dense(9,activation='softmax'))
```

In [26]:

```
# Compiling the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_2 (Rescaling)     (None, 180, 180, 3)       0

 sequential_1 (Sequential)   (None, 180, 180, 3)       0

 conv2d_4 (Conv2D)           (None, 180, 180, 64)      1792

 max_pooling2d_4 (MaxPoolin  (None, 90, 90, 64)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 90, 90, 128)       73856

 max_pooling2d_5 (MaxPoolin  (None, 45, 45, 128)       0
 g2D)

 dropout (Dropout)           (None, 45, 45, 128)       0

 flatten_2 (Flatten)         (None, 259200)            0

 dense_4 (Dense)             (None, 256)               66355456

 dense_5 (Dense)             (None, 9)                 2313

=================================================================
Total params: 66433417 (253.42 MB)
Trainable params: 66433417 (253.42 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [27]:

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

```
Epoch 1/20
56/56 [==============================] - 8s 117ms/step - loss: 3.2003 - ac
curacy: 0.1925 - val_loss: 2.0524 - val_accuracy: 0.1902
Epoch 2/20
56/56 [==============================] - 6s 112ms/step - loss: 2.0077 - ac
curacy: 0.2260 - val_loss: 1.9569 - val_accuracy: 0.2864
Epoch 3/20
56/56 [==============================] - 6s 112ms/step - loss: 2.0182 - ac
curacy: 0.2489 - val_loss: 1.9144 - val_accuracy: 0.2886
Epoch 4/20
56/56 [==============================] - 6s 110ms/step - loss: 1.8440 - ac
curacy: 0.2829 - val_loss: 1.7105 - val_accuracy: 0.4295
Epoch 5/20
56/56 [==============================] - 6s 112ms/step - loss: 1.7119 - ac
curacy: 0.3783 - val_loss: 1.7272 - val_accuracy: 0.4072
Epoch 6/20
56/56 [==============================] - 6s 112ms/step - loss: 1.6448 - ac
curacy: 0.4079 - val_loss: 1.6728 - val_accuracy: 0.4183
Epoch 7/20
56/56 [==============================] - 6s 112ms/step - loss: 1.5662 - ac
curacy: 0.4526 - val_loss: 1.5137 - val_accuracy: 0.4787
Epoch 8/20
56/56 [==============================] - 6s 115ms/step - loss: 1.5093 - ac
curacy: 0.4743 - val_loss: 1.5816 - val_accuracy: 0.4832
Epoch 9/20
56/56 [==============================] - 6s 111ms/step - loss: 1.4558 - ac
curacy: 0.4872 - val_loss: 1.4930 - val_accuracy: 0.4989
Epoch 10/20
56/56 [==============================] - 6s 113ms/step - loss: 1.4075 - ac
curacy: 0.5067 - val_loss: 1.4167 - val_accuracy: 0.5168
Epoch 11/20
56/56 [==============================] - 6s 114ms/step - loss: 1.3741 - ac
curacy: 0.5073 - val_loss: 1.4887 - val_accuracy: 0.5056
Epoch 12/20
56/56 [==============================] - 6s 111ms/step - loss: 1.4012 - ac
curacy: 0.4955 - val_loss: 1.4562 - val_accuracy: 0.5213
Epoch 13/20
56/56 [==============================] - 6s 113ms/step - loss: 1.3476 - ac
curacy: 0.5145 - val_loss: 1.5387 - val_accuracy: 0.4989
Epoch 14/20
56/56 [==============================] - 6s 112ms/step - loss: 1.3801 - ac
curacy: 0.5045 - val_loss: 1.4854 - val_accuracy: 0.5034
Epoch 15/20
56/56 [==============================] - 6s 112ms/step - loss: 1.3755 - ac
curacy: 0.5100 - val_loss: 1.3829 - val_accuracy: 0.5324
Epoch 16/20
56/56 [==============================] - 6s 111ms/step - loss: 1.3238 - ac
curacy: 0.5290 - val_loss: 1.4311 - val_accuracy: 0.4989
Epoch 17/20
56/56 [==============================] - 6s 111ms/step - loss: 1.3784 - ac
curacy: 0.5156 - val_loss: 1.3800 - val_accuracy: 0.5459
Epoch 18/20
56/56 [==============================] - 6s 109ms/step - loss: 1.2658 - ac
curacy: 0.5519 - val_loss: 1.4253 - val_accuracy: 0.5391
Epoch 19/20
56/56 [==============================] - 6s 112ms/step - loss: 1.2742 - ac
curacy: 0.5469 - val_loss: 1.4183 - val_accuracy: 0.5436
Epoch 20/20
56/56 [==============================] - 6s 109ms/step - loss: 1.2662 - ac
curacy: 0.5458 - val_loss: 1.4105 - val_accuracy: 0.5190
```

In [28]:

```
#plotting the graph of outcome
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()
```

Findings from the graph:

- As the training accuracy increases linearly over time, where as the validation accuracy increases and stall at ~51% accuracy in training process.
- **As the training loss decreases with epochs the validation loss decreases**
- The plots show that gap between training accuracy and validation accuracy have decreased from previous model, and it has achieved around **~51%** accuracy on the validation set.
- The difference in accuracy between training and validation accuracy is **very less**

Observation :- We can clearly see that the overfitting of the model has redused significantly when compared the earlier models.

Step 8:- 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.

Class distribution:

Examining the current class distribution in the training dataset

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.

In [29]:

```
for i in range(len(class_names)):
  print(class_names[i],' - ',len(list(dataset_train.glob(class_names[i]+'/*.jpg'))))
```

```
actinic keratosis  -  114
basal cell carcinoma  -  376
dermatofibroma  -  95
melanoma  -  438
nevus  -  357
pigmented benign keratosis  -  462
seborrheic keratosis  -  77
squamous cell carcinoma  -  181
vascular lesion  -  139
```

Step 9:- Todo: Write your findings here:

- Which class has the least number of samples?

- Seborrheic keratosis has the least number with 77 samples.

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

- pigmented benign keratosis dominates with 462 samples in total.

Step 10 :- Handling class imbalances:

Rectifing class imbalances present in the training dataset with Augmentor library.

In [30]:

```
!pip install Augmentor
```

```
Collecting Augmentor
  Downloading Augmentor-0.2.12-py2.py3-none-any.whl (38 kB)
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/di
st-packages (from Augmentor) (4.66.1)
Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.10/
dist-packages (from Augmentor) (1.23.5)
Installing collected packages: Augmentor
Successfully installed Augmentor-0.2.12
```

In [31]:

```
path_to_training_dataset="/content/Skin cancer ISIC The International Skin Imaging Collab
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 cla
```

```
Initialised with 114 image(s) found.
Output directory set to /content/Skin cancer ISIC The International Skin I
maging Collaboration/Train/actinic keratosis/output.

Processing <PIL.Image.Image image mode=RGB size=600x450 at 0x7B75ED364580
>: 100%|██████████| 500/500 [00:16<00:00, 29.66 Samples/s]

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

Processing <PIL.Image.Image image mode=RGB size=600x450 at 0x7B76746C3A60
>: 100%|██████████| 500/500 [00:19<00:00, 26.27 Samples/s]

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

Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=600x450
at 0x7B75ED39F4F0>: 100%|██████████| 500/500 [00:21<00:00, 23.26 Samples/
s]

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

Processing <PIL.Image.Image image mode=RGB size=1024x768 at 0x7B75ED124070
>: 100%|██████████| 500/500 [01:22<00:00,  6.09 Samples/s]

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

Processing <PIL.Image.Image image mode=RGB size=919x802 at 0x7B75ED2106A0
>: 100%|██████████| 500/500 [01:29<00:00,  5.57 Samples/s]

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

Processing <PIL.Image.Image image mode=RGB size=600x450 at 0x7B75ED3821D0
>: 100%|██████████| 500/500 [00:18<00:00, 26.94 Samples/s]

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

Processing <PIL.Image.Image image mode=RGB size=1024x768 at 0x7B75ED41F910
>: 100%|██████████| 500/500 [00:40<00:00, 12.39 Samples/s]

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

```
Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=600x450
at 0x7B75ED2FA0E0>: 100%|██████████| 500/500 [00:14<00:00, 33.98 Samples/
s]

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

Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=600x450
at 0x7B75ED27C1F0>: 100%|██████████| 500/500 [00:15<00:00, 32.59 Samples/
s]
```

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.

In [32]:

```
dataset_train =("/content/Skin cancer ISIC The International Skin Imaging Collaboration/T
```

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

In [33]:

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

In [34]:

```
dict_new = dict(zip(path_list, lesion_list_new))
df = pd.DataFrame(list(dict_new.items()),columns = ['Path','Label'])
```

Note:- 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.

In [35]:

```
# initializing the parameter to load the images
batch_size = 32
img_height = 180
img_width = 180
```

In [36]:

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

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

Note:- So here we can see we have added around 4500 new images using augmentor. So now the total no of images are 4500 + 2239 = 6739 images

In [37]:

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

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

Model 4

Step 11:- Model Building & training on the rectified class imbalance data :

Step i : Creating a CNN model, which can accurately detect 9 classes present in the dataset. While building the model, rescaling images to normalize pixel values between (0,1).

Step ii : Choosing an appropriate optimiser and loss function for model training.

Step iii : Training the model for ~30 epochs.

Step iv : Plotting Graph for findings after the model fit to check if there is any evidence of model overfit or underfit.

In [38]:

```
# CNN Model
model=models.Sequential()
# scaling the pixel values from 0-255 to 0-1
model.add(layers.Rescaling(scale=1./255,input_shape=(180,180,3)))
model.add(data_augmentation)

# Convolution layer with 64 features, 3x3 filter and relu activation with 2x2 pooling
model.add(layers.Conv2D(64,(3,3),padding = 'same',activation='relu'))
#model.add(BatchNormalization())
model.add(layers.MaxPooling2D())

# Convolution layer with 128 features, 3x3 filter and relu activation with 2x2 pooling
model.add(layers.Conv2D(128,(3,3),padding = 'same',activation='relu'))
#model.add(BatchNormalization())
model.add(layers.MaxPooling2D())
#adding a 20% dropout after the convolution layers
model.add(layers.Dropout(0.2))

model.add(layers.Flatten())
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dense(9,activation='softmax'))
```

In [39]:

```
# Compiling the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
model.summary()
```

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

 sequential_1 (Sequential)   (None, 180, 180, 3)       0

 conv2d_6 (Conv2D)           (None, 180, 180, 64)      1792

 max_pooling2d_6 (MaxPoolin  (None, 90, 90, 64)        0
 g2D)

 conv2d_7 (Conv2D)           (None, 90, 90, 128)       73856

 max_pooling2d_7 (MaxPoolin  (None, 45, 45, 128)       0
 g2D)

 dropout_1 (Dropout)         (None, 45, 45, 128)       0

 flatten_3 (Flatten)         (None, 259200)            0

 dense_6 (Dense)             (None, 256)               66355456

 dense_7 (Dense)             (None, 9)                 2313

=================================================================
Total params: 66433417 (253.42 MB)
Trainable params: 66433417 (253.42 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [40]:

```
# Training the model
epochs = 30
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/30
169/169 [==============================] - 45s 252ms/step - loss: 2.3659 -
accuracy: 0.2546 - val_loss: 1.5766 - val_accuracy: 0.4031
Epoch 2/30
169/169 [==============================] - 31s 178ms/step - loss: 1.5978 -
accuracy: 0.4011 - val_loss: 1.4744 - val_accuracy: 0.4454
Epoch 3/30
169/169 [==============================] - 32s 183ms/step - loss: 1.4436 -
accuracy: 0.4572 - val_loss: 1.4222 - val_accuracy: 0.4662
Epoch 4/30
169/169 [==============================] - 31s 179ms/step - loss: 1.3588 -
accuracy: 0.4889 - val_loss: 1.2965 - val_accuracy: 0.5197
Epoch 5/30
169/169 [==============================] - 36s 210ms/step - loss: 1.2875 -
accuracy: 0.5087 - val_loss: 1.2858 - val_accuracy: 0.5160
Epoch 6/30
169/169 [==============================] - 30s 175ms/step - loss: 1.2419 -
accuracy: 0.5328 - val_loss: 1.2797 - val_accuracy: 0.5234
Epoch 7/30
169/169 [==============================] - 31s 177ms/step - loss: 1.2016 -
accuracy: 0.5458 - val_loss: 1.2382 - val_accuracy: 0.5367
Epoch 8/30
169/169 [==============================] - 30s 175ms/step - loss: 1.1529 -
accuracy: 0.5625 - val_loss: 1.1887 - val_accuracy: 0.5501
Epoch 9/30
169/169 [==============================] - 36s 211ms/step - loss: 1.1138 -
accuracy: 0.5768 - val_loss: 1.1442 - val_accuracy: 0.5620
Epoch 10/30
169/169 [==============================] - 31s 178ms/step - loss: 1.0794 -
accuracy: 0.5953 - val_loss: 1.0480 - val_accuracy: 0.6043
Epoch 11/30
169/169 [==============================] - 30s 175ms/step - loss: 1.0347 -
accuracy: 0.6120 - val_loss: 1.0590 - val_accuracy: 0.5976
Epoch 12/30
169/169 [==============================] - 31s 182ms/step - loss: 0.9755 -
accuracy: 0.6328 - val_loss: 1.0638 - val_accuracy: 0.6266
Epoch 13/30
169/169 [==============================] - 36s 210ms/step - loss: 0.9481 -
accuracy: 0.6471 - val_loss: 1.0054 - val_accuracy: 0.6451
Epoch 14/30
169/169 [==============================] - 31s 176ms/step - loss: 0.9599 -
accuracy: 0.6411 - val_loss: 0.9685 - val_accuracy: 0.6578
Epoch 15/30
169/169 [==============================] - 31s 180ms/step - loss: 0.8658 -
accuracy: 0.6788 - val_loss: 0.9961 - val_accuracy: 0.6444
Epoch 16/30
169/169 [==============================] - 31s 179ms/step - loss: 0.8470 -
accuracy: 0.6808 - val_loss: 0.9479 - val_accuracy: 0.6696
Epoch 17/30
169/169 [==============================] - 31s 181ms/step - loss: 0.8494 -
accuracy: 0.6812 - val_loss: 0.9653 - val_accuracy: 0.6607
Epoch 18/30
169/169 [==============================] - 31s 182ms/step - loss: 0.7752 -
accuracy: 0.7131 - val_loss: 0.9332 - val_accuracy: 0.6733
Epoch 19/30
169/169 [==============================] - 31s 178ms/step - loss: 0.7558 -
accuracy: 0.7129 - val_loss: 0.9441 - val_accuracy: 0.6741
Epoch 20/30
169/169 [==============================] - 31s 179ms/step - loss: 0.7207 -
accuracy: 0.7331 - val_loss: 0.9055 - val_accuracy: 0.6934
Epoch 21/30
```

```
169/169 [==============================] - 31s 180ms/step - loss: 0.6906 -
accuracy: 0.7515 - val_loss: 0.8846 - val_accuracy: 0.6956
Epoch 22/30
169/169 [==============================] - 36s 210ms/step - loss: 0.6798 -
accuracy: 0.7465 - val_loss: 0.8904 - val_accuracy: 0.7194
Epoch 23/30
169/169 [==============================] - 31s 177ms/step - loss: 0.6612 -
accuracy: 0.7585 - val_loss: 0.9295 - val_accuracy: 0.6986
Epoch 24/30
169/169 [==============================] - 30s 175ms/step - loss: 0.6132 -
accuracy: 0.7659 - val_loss: 0.8380 - val_accuracy: 0.7283
Epoch 25/30
169/169 [==============================] - 31s 177ms/step - loss: 0.6092 -
accuracy: 0.7750 - val_loss: 0.7350 - val_accuracy: 0.7558
Epoch 26/30
169/169 [==============================] - 36s 206ms/step - loss: 0.5918 -
accuracy: 0.7836 - val_loss: 0.7785 - val_accuracy: 0.7461
Epoch 27/30
169/169 [==============================] - 30s 175ms/step - loss: 0.5728 -
accuracy: 0.7902 - val_loss: 0.7731 - val_accuracy: 0.7558
Epoch 28/30
169/169 [==============================] - 30s 174ms/step - loss: 0.5619 -
accuracy: 0.7953 - val_loss: 0.8256 - val_accuracy: 0.7439
Epoch 29/30
169/169 [==============================] - 30s 174ms/step - loss: 0.5607 -
accuracy: 0.7953 - val_loss: 0.7899 - val_accuracy: 0.7513
Epoch 30/30
169/169 [==============================] - 36s 208ms/step - loss: 0.5185 -
accuracy: 0.8042 - val_loss: 0.7488 - val_accuracy: 0.7513
```

In [41]:

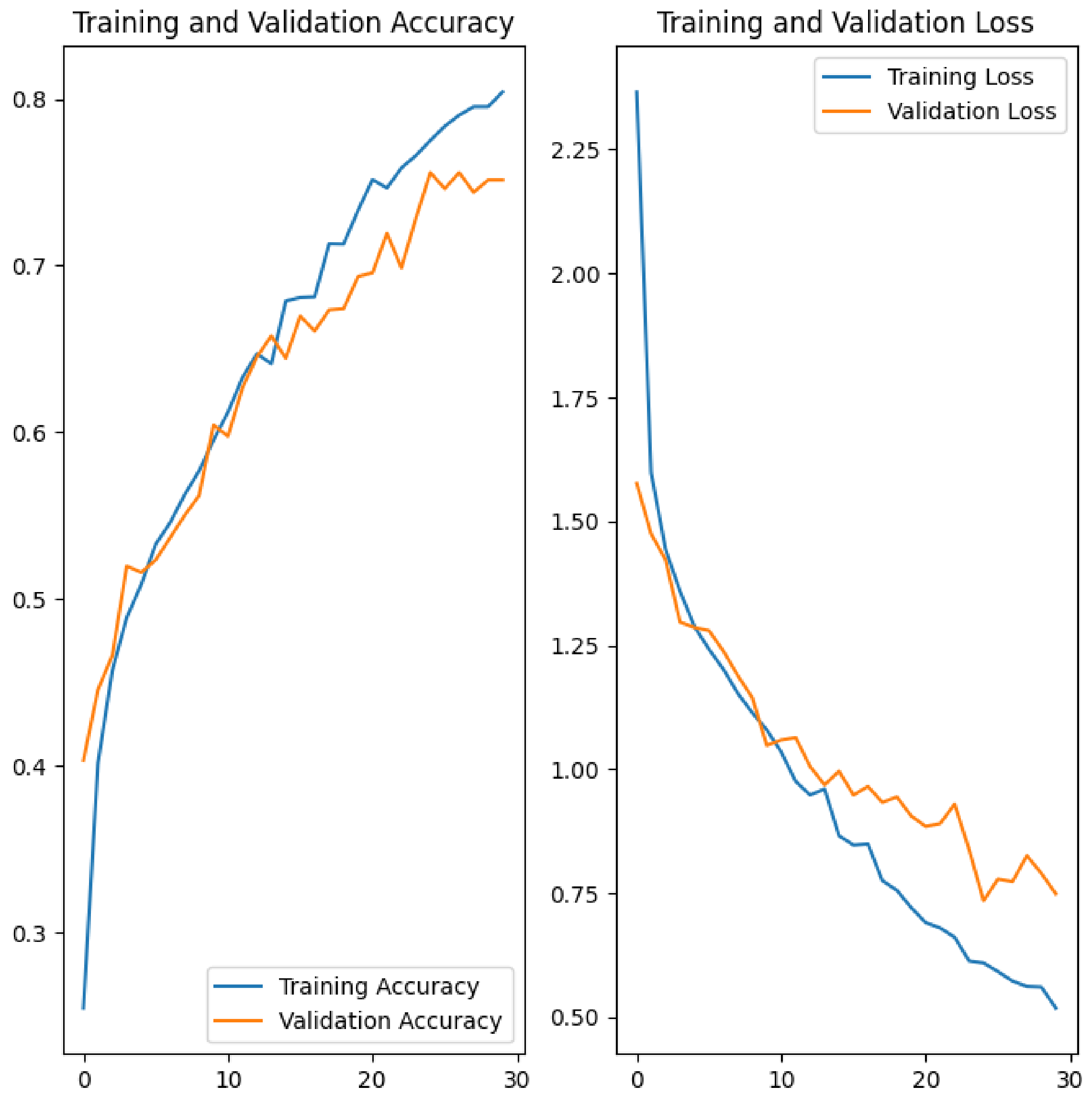
```
# Visualizing model results
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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()
```



Conclusion from the above analysis

- The training accuracy increases linearly over time, where as the validation accuracy increases in training process.

- The training loss decreases with epochs, where as the validation loss also decreases with epochs.

- The plots show that gap between training accuracy and validation accuracy have decreased significantly from previous model.

- The difference in accuracy between training and validation accuracy is very less.

- After training the model for around 50 epochs, the following results were achieved:

- **Training Accuracy: ~80%**
- **Validation Accuracy: ~75%**
- **Training Loss: 0.5**
- **Validation Loss: 0.7**

Note :- Class rebalancing not only got rid of overfitting it also improved the accuracy from ~54% to ~80%.