```
Start coding or generate with AI.
```

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, 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, BatchNormalization, Conv2D, MaxPooling2D
```

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

```
## If you are using the data by mounting the google drive, use the following :
from google.colab import drive
drive.mount('/content/drive')
## Ref:https://towardsdatascience.com/downloading-datasets-into-google-drive-via-google-colab-bcb1b30b0166
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", 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
## Paths of train and test dataset on Google drive

data_dir_train = pathlib.Path("/content/drive/My Drive/Melanoma Detection/Train")
data_dir_test = pathlib.Path("/content/drive/My Drive/Melanoma Detection/Test")

## Paths of train and test dataset on local machine
# data_dir_train = pathlib.Path("Skin cancer ISIC The International Skin Imaging Collaboration/Train")
# data_dir_test = pathlib.Path("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.preprocessing.image_dataset_from_directory(data_dir_train,
                                                               seed=123,
                                                               validation_split=0.2,
                                                               image_size=(img_height,img_width),
                                                               batch_size=batch_size,
                                                               color_mode='rgb',
                                                               subset='training')
```

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.preprocessing.image_dataset_from_directory(data_dir_train,
                                                             seed=123,
                                                             validation_split=0.2,
                                                             image_size=(img_height,img_width),
                                                             batch_size=batch_size,
                                                             color_mode='rgb',
                                                             subset='validation')
```

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

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

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

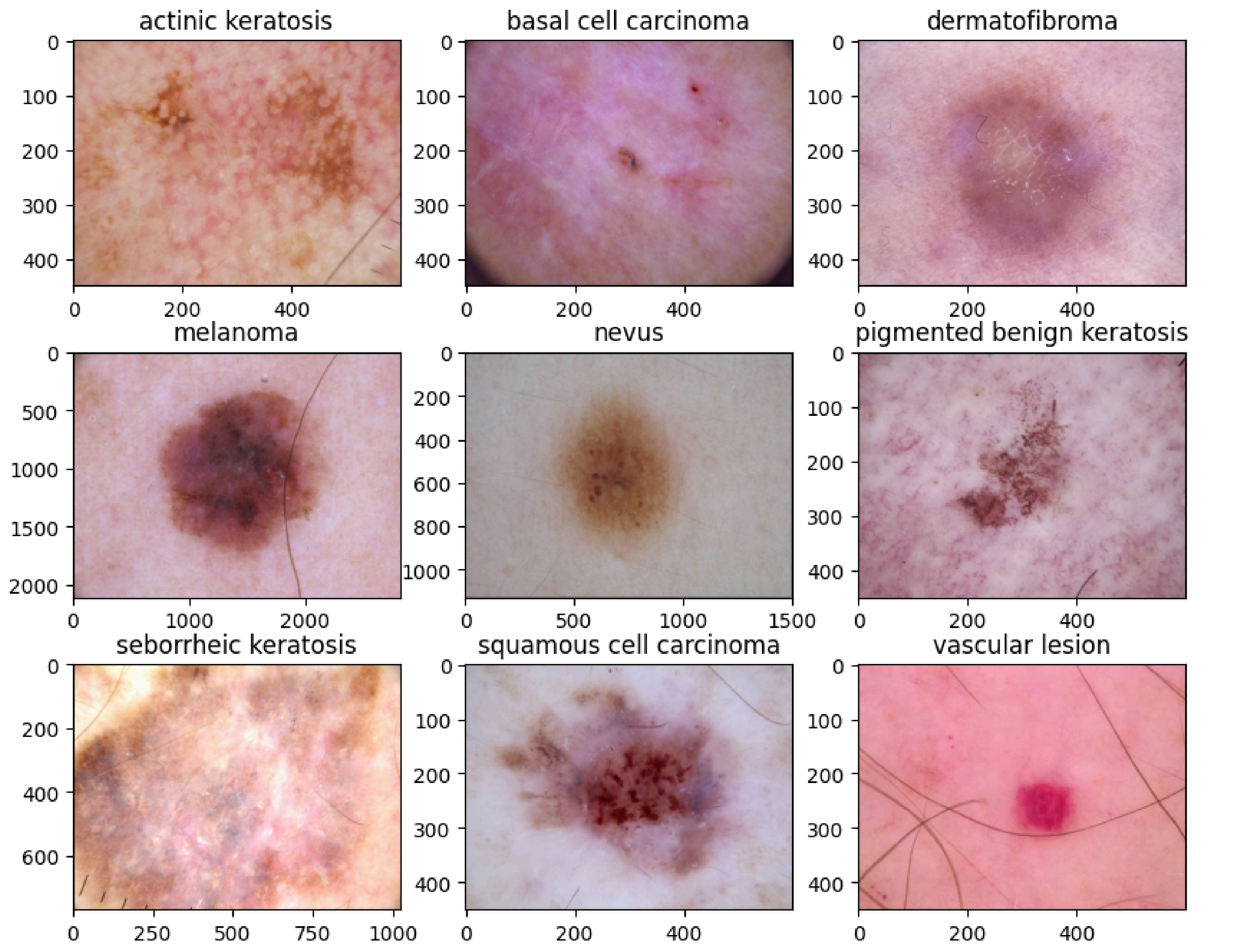
```
# List out all the classes of skin cancer and store them in a list.
# You can find the class names in the class_names attribute on these datasets.
# These correspond to the directory names in alphabetical order.
class_names = train_ds.class_names
print(class_names)
```

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

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=(10,8))
for i in range(len(class_names)):
  plt.subplot(3,3,i+1)
  image= plt.imread(str(list(data_dir_train.glob(class_names[i]+'/*.jpg'))[1]))
  plt.title(class_names[i])
  plt.imshow(image)
```

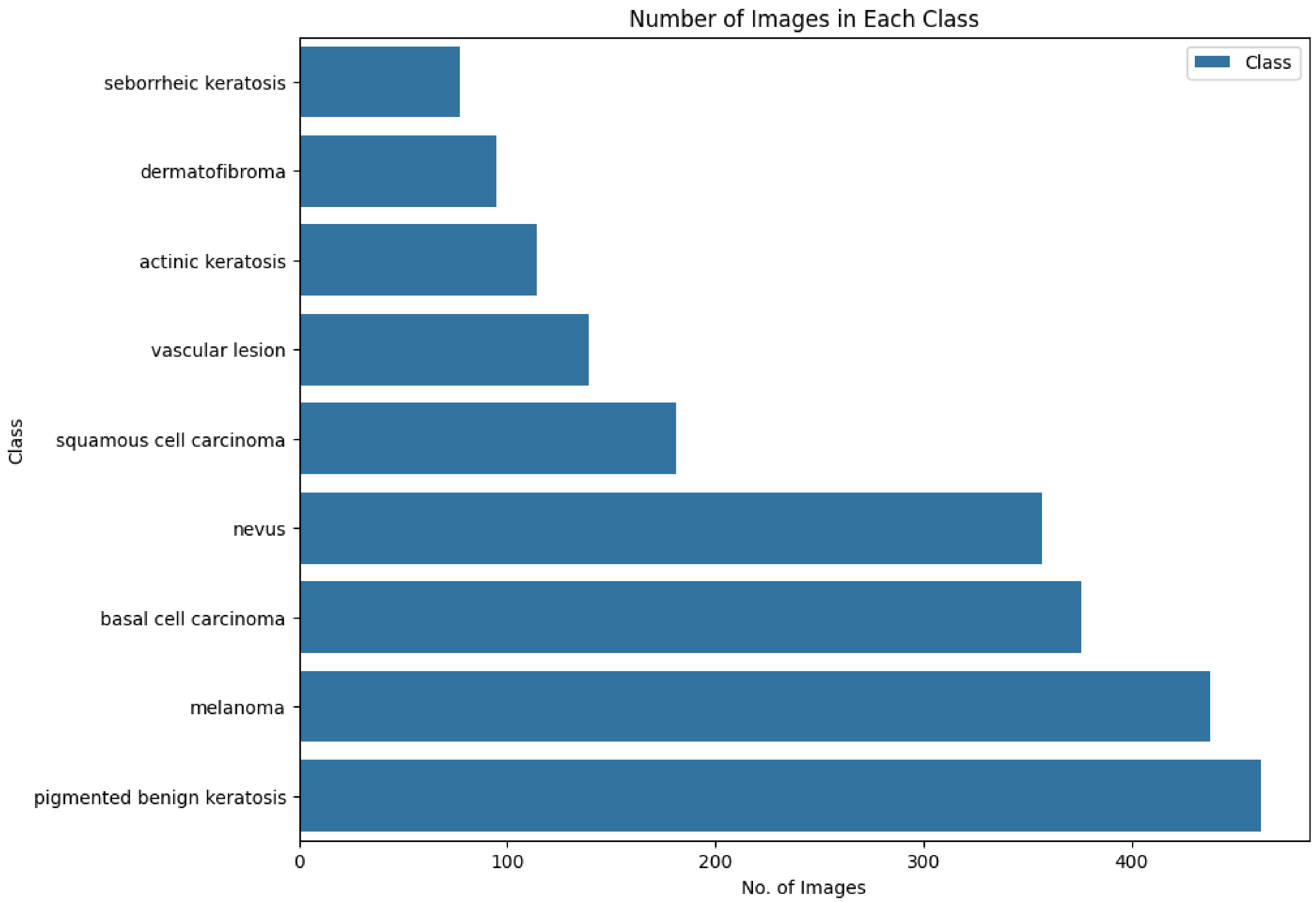


```
# Define a function to count and analyze the distribution of images in each class

def class_distribution_count(directory):
    # Initialize a list to store image counts for each class
    count = []

    # Count the number of images in each class directory
    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))]))

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

    # Create a DataFrame with class names and corresponding image counts
    df = pd.DataFrame(list(zip(sub_directory, count)), columns=['Class', 'No. of Images'])

    # Sort the DataFrame in ascending order of image counts
    df = df.sort_values(by='No. of Images', ascending=True)

    # Return the sorted DataFrame
    return df

# Call the function to get the class distribution and image count
class_distribution_df = class_distribution_count(data_dir_train)

# Display the DataFrame
class_distribution_df
```

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

```
# Visualize the Number of Images in Each Class

# Import the seaborn library for data visualization
import seaborn as sns

# Set the size of the plot figure
```

```
    plt.figure(figsize=(10, 8))

    # Create a bar plot using seaborn
    # x-axis: Number of Images
    # y-axis: Class names
    # Data source: df DataFrame
    # Label: "Class"
    sns.barplot(x="No. of Images", y="Class", data= class_distribution_df, label="Class")

    # Add a title to the plot
    plt.title("Number of Images in Each Class")

    # Display the plot
    plt.show()
```



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]

Model Building & training :

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). Choosing an appropriate optimiser and loss function for model training Training the model for ~20 epochs Plotting Graph for findings after the model fit to check if there is any evidence of model overfit or underfit.

```
# 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'))
```

Compile the

Choose an appropirate optimiser and loss function for model training

```
# 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)   │         1,792 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)    │ (None, 90, 90, 64)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)               │ (None, 90, 90, 128)    │        73,856 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 45, 45, 128)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)               │ (None, 259200)         │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 256)            │    66,355,456 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 9)              │         2,313 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 66,433,417 (253.42 MB)
 Trainable params: 66,433,417 (253.42 MB)
```

Train the model

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

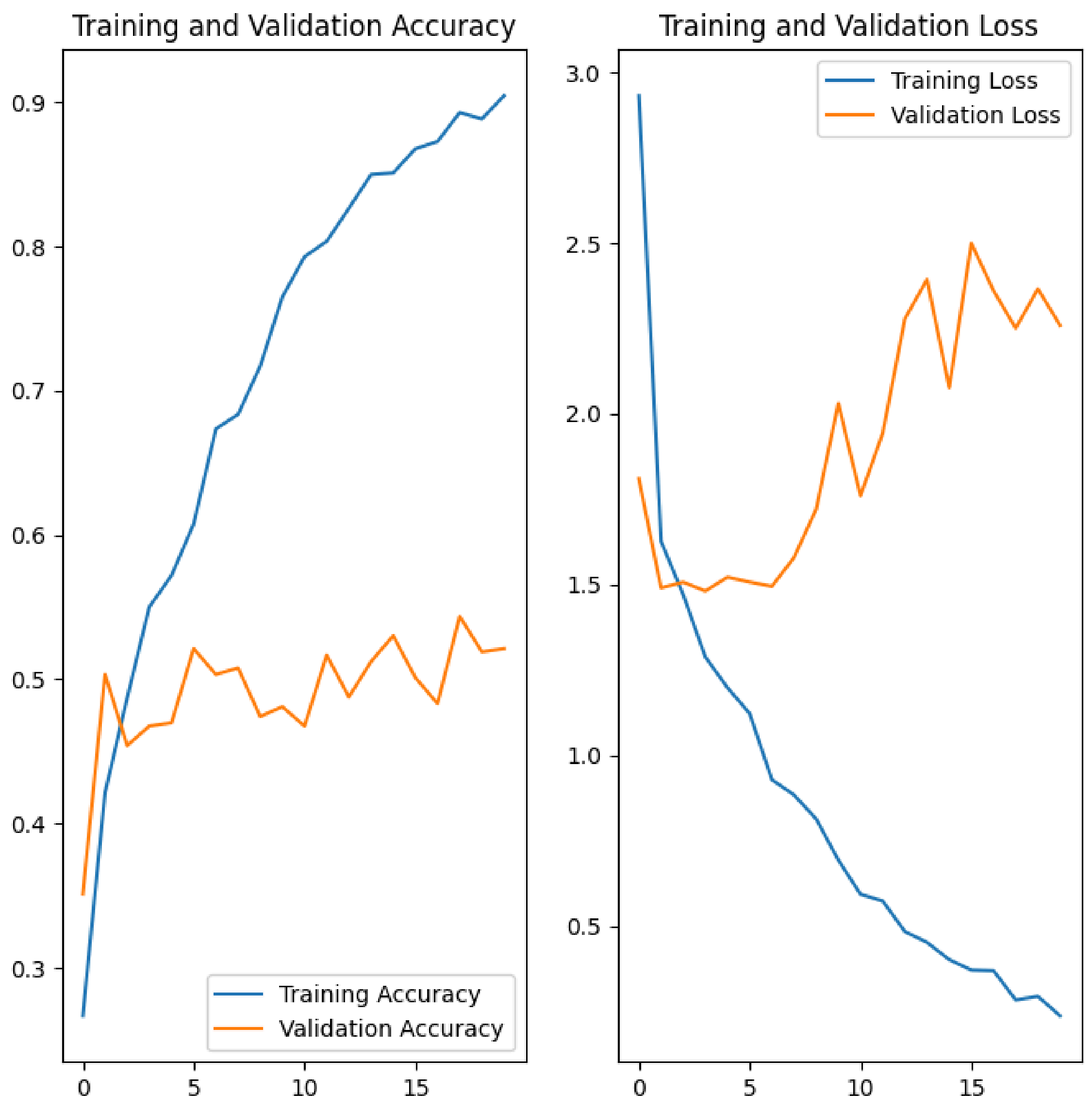
```
Epoch 1/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 423s 3s/step - accuracy: 0.2092 - loss: 5.2105 - val_accuracy: 0.3512 - val_loss: 1.8102
Epoch 2/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 66ms/step - accuracy: 0.3875 - loss: 1.6825 - val_accuracy: 0.5034 - val_loss: 1.4902
Epoch 3/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step - accuracy: 0.4911 - loss: 1.4545 - val_accuracy: 0.4541 - val_loss: 1.5069
Epoch 4/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 64ms/step - accuracy: 0.5568 - loss: 1.3051 - val_accuracy: 0.4676 - val_loss: 1.4811
Epoch 5/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step - accuracy: 0.5679 - loss: 1.1955 - val_accuracy: 0.4698 - val_loss: 1.5216
Epoch 6/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 64ms/step - accuracy: 0.6093 - loss: 1.1355 - val_accuracy: 0.5213 - val_loss: 1.5073
Epoch 7/20
```

```
56/56 ━━━━━━━━━━━━━━━━━━━━ 5s 64ms/step - accuracy: 0.6630 - loss: 0.9545 - val_accuracy: 0.5034 - val_loss: 1.4947
Epoch 8/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 65ms/step - accuracy: 0.6910 - loss: 0.8528 - val_accuracy: 0.5078 - val_loss: 1.5790
Epoch 9/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 5s 66ms/step - accuracy: 0.7186 - loss: 0.7986 - val_accuracy: 0.4743 - val_loss: 1.7210
Epoch 10/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 65ms/step - accuracy: 0.7695 - loss: 0.7126 - val_accuracy: 0.4810 - val_loss: 2.0308
Epoch 11/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 65ms/step - accuracy: 0.7976 - loss: 0.5768 - val_accuracy: 0.4676 - val_loss: 1.7604
Epoch 12/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 65ms/step - accuracy: 0.8068 - loss: 0.5696 - val_accuracy: 0.5168 - val_loss: 1.9430
Epoch 13/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 5s 65ms/step - accuracy: 0.8313 - loss: 0.4694 - val_accuracy: 0.4877 - val_loss: 2.2782
Epoch 14/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 5s 63ms/step - accuracy: 0.8595 - loss: 0.4039 - val_accuracy: 0.5123 - val_loss: 2.3941
Epoch 15/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step - accuracy: 0.8431 - loss: 0.4158 - val_accuracy: 0.5302 - val_loss: 2.0772
Epoch 16/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 5s 65ms/step - accuracy: 0.8810 - loss: 0.3342 - val_accuracy: 0.5011 - val_loss: 2.4998
Epoch 17/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step - accuracy: 0.8737 - loss: 0.3752 - val_accuracy: 0.4832 - val_loss: 2.3607
Epoch 18/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 5s 63ms/step - accuracy: 0.8914 - loss: 0.2823 - val_accuracy: 0.5436 - val_loss: 2.2508
Epoch 19/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 65ms/step - accuracy: 0.8815 - loss: 0.2990 - val_accuracy: 0.5190 - val_loss: 2.3664
Epoch 20/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step - accuracy: 0.9248 - loss: 0.2061 - val_accuracy: 0.5213 - val_loss: 2.2592
```

Double-click (or enter) to edit

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()
```



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

1. The model's training accuracy shows a steady increase of upto 90%, while validation accuracy remains consistently around 50%.
2. A high training accuracy suggests the model has effectively captured the noise within the data.
3. As the training accuracy increases linearly over time, where as the validation accuracy stall at 50% accuracy in training process. Training loss decreases 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 50% accuracy on the validation set.
4. The difference in accuracy between training and validation accuracy is noticeable which is a sign of overfitting.

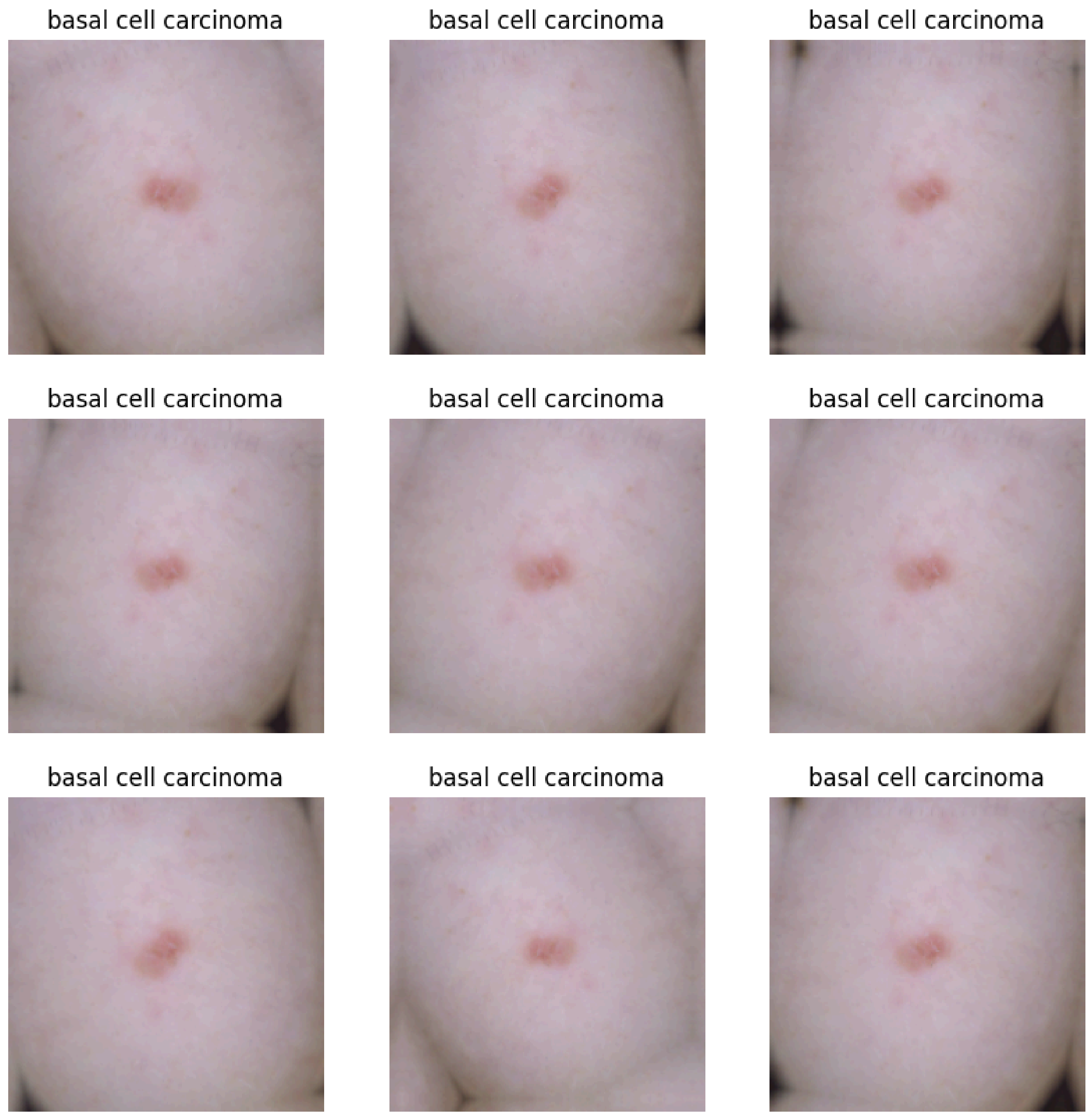
These observations confirm the presence of overfitting.

Choosing an appropriate data augmentation strategy to resolve underfitting/overfitting

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.

```
# After you have analysed the model fit history for presence of underfit or overfit, choose an appropriate data augumentation strategy.

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

```
# visualizing how your augmentation strategy works for one instance of training image.
plt.figure(figsize=(10, 10))
for images, labels 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.title(class_names[labels[0]])
        plt.axis("off")
```



Model Building & training on the augmented data

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).

Choosing an appropriate optimiser and loss function for model training Training the model for ~20 epochs

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

CNN MODEL

```
## You can use Dropout layer if there is an evidence of overfitting in your findings
# 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.Dropout(0.2))
model.add(layers.Flatten())
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dense(9,activation='softmax'))
```

Compiling the model

Double-click (or enter) to edit

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              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)	1,792
max_pooling2d_6 (MaxPooling2D)	(None, 90, 90, 64)	0
conv2d_7 (Conv2D)	(None, 90, 90, 128)	73,856
max_pooling2d_7 (MaxPooling2D)	(None, 45, 45, 128)	0
dropout_2 (Dropout)	(None, 45, 45, 128)	0
flatten_3 (Flatten)	(None, 259200)	0
dense_6 (Dense)	(None, 256)	66,355,456
dense_7 (Dense)	(None, 9)	2,313

```
 Total params: 66,433,417 (253.42 MB)
 Trainable params: 66,433,417 (253.42 MB)
```

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 ━━━━━━━━━━━━━━━━━━━━ 12s 139ms/step - accuracy: 0.2100 - loss: 5.5349 - val_accuracy: 0.3244 - val_loss: 1.8363
Epoch 2/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 7s 131ms/step - accuracy: 0.3242 - loss: 1.8183 - val_accuracy: 0.3400 - val_loss: 1.7485
Epoch 3/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 7s 132ms/step - accuracy: 0.3415 - loss: 1.7508 - val_accuracy: 0.4787 - val_loss: 1.5417
Epoch 4/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 7s 131ms/step - accuracy: 0.4658 - loss: 1.5291 - val_accuracy: 0.4116 - val_loss: 1.6872
Epoch 5/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 7s 130ms/step - accuracy: 0.4546 - loss: 1.5341 - val_accuracy: 0.5324 - val_loss: 1.4177
Epoch 6/20
```

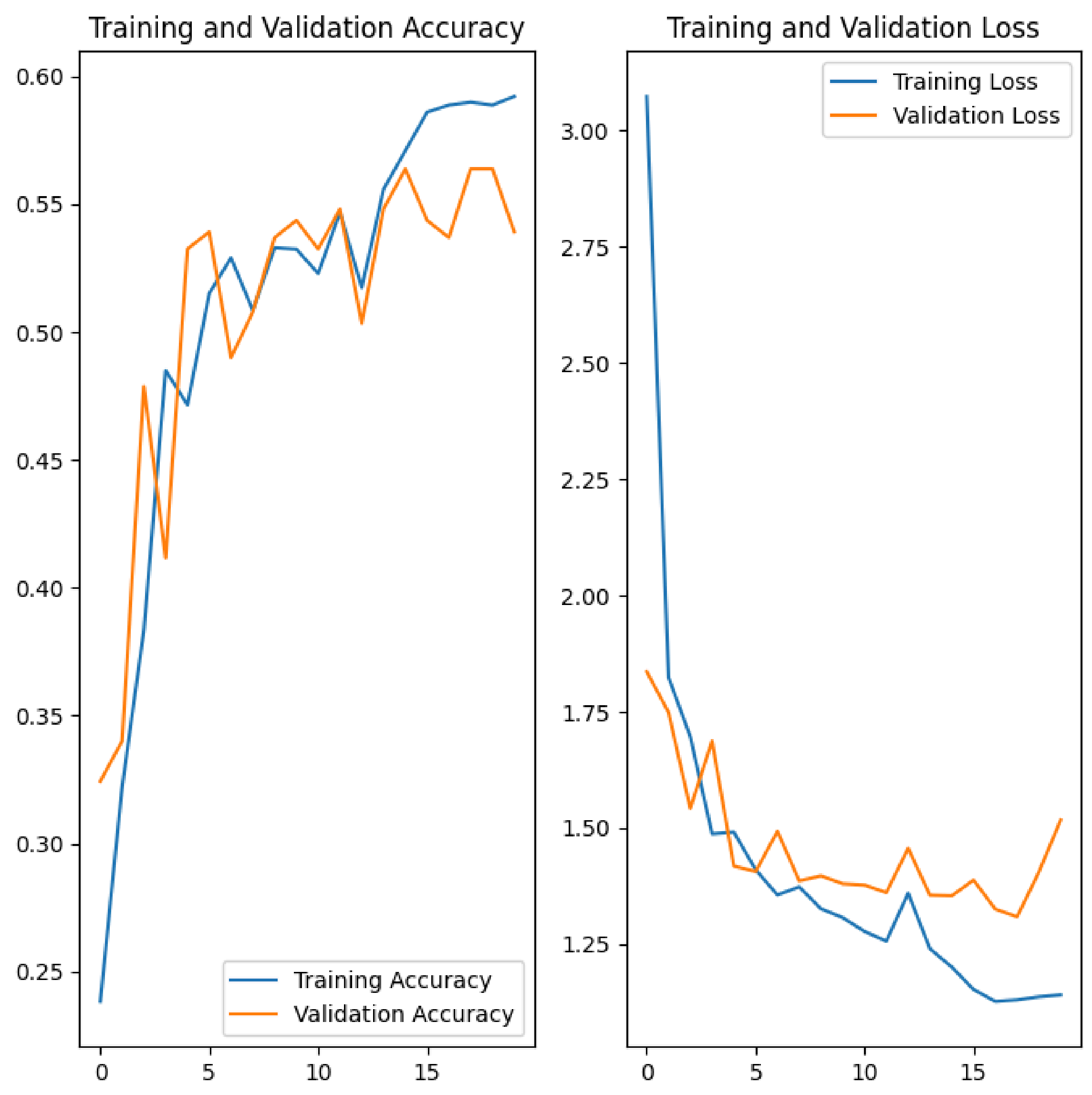
```
56/56 ━━━━━━━━━━━━━━━━━━━━ 10s 131ms/step - accuracy: 0.4960 - loss: 1.4340 - val_accuracy: 0.5391 - val_loss: 1.4060
Epoch 7/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 10s 129ms/step - accuracy: 0.5392 - loss: 1.3148 - val_accuracy: 0.4899 - val_loss: 1.4925
Epoch 8/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 10s 130ms/step - accuracy: 0.4937 - loss: 1.4120 - val_accuracy: 0.5078 - val_loss: 1.3860
Epoch 9/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 7s 132ms/step - accuracy: 0.5438 - loss: 1.3055 - val_accuracy: 0.5369 - val_loss: 1.3968
Epoch 10/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 7s 130ms/step - accuracy: 0.5373 - loss: 1.3270 - val_accuracy: 0.5436 - val_loss: 1.3797
Epoch 11/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 7s 131ms/step - accuracy: 0.5363 - loss: 1.2331 - val_accuracy: 0.5324 - val_loss: 1.3769
Epoch 12/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 7s 132ms/step - accuracy: 0.5571 - loss: 1.2218 - val_accuracy: 0.5481 - val_loss: 1.3608
Epoch 13/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 7s 130ms/step - accuracy: 0.5203 - loss: 1.3548 - val_accuracy: 0.5034 - val_loss: 1.4561
Epoch 14/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 10s 129ms/step - accuracy: 0.5666 - loss: 1.2087 - val_accuracy: 0.5481 - val_loss: 1.3552
Epoch 15/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 7s 129ms/step - accuracy: 0.5658 - loss: 1.2195 - val_accuracy: 0.5638 - val_loss: 1.3539
Epoch 16/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 7s 128ms/step - accuracy: 0.5856 - loss: 1.1373 - val_accuracy: 0.5436 - val_loss: 1.3876
Epoch 17/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 7s 128ms/step - accuracy: 0.5936 - loss: 1.1210 - val_accuracy: 0.5369 - val_loss: 1.3252
Epoch 18/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 7s 127ms/step - accuracy: 0.5799 - loss: 1.1427 - val_accuracy: 0.5638 - val_loss: 1.3087
Epoch 19/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 7s 127ms/step - accuracy: 0.6044 - loss: 1.0828 - val_accuracy: 0.5638 - val_loss: 1.4049
Epoch 20/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 7s 127ms/step - accuracy: 0.5822 - loss: 1.1492 - val_accuracy: 0.5391 - val_loss: 1.5169
```

Visualizing the results

Double-click (or enter) to edit

```
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()
```



```
Start coding or generate with AI.
```

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?

1. Training accuracy decreased from 90% in earlier model run to near 60% , the validation accuracy increased to near 57% in accuracy in training process . The gap is shrinking.
2. As the training loss decreases with epochs the validation loss decreases
3. We can clearly see that the overfitting of the model has reduced significantly when compared the earlier model.

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.

Examine the current class distribution in the training dataset & explaine the following:

Which class has the least number of samples?

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

```
## Your code goes here.
for i in range(len(class_names)):
  print(class_names[i],' - ',len(list(data_dir_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
```

- Which class has the least number of samples?

seborrheic keratosis has the least number with 77 samples **bold text**

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

pigmented benign keratosis and melanona with 462 and 438 samples respectively **bold text**

Handling class imbalances

Rectifing class imbalances present in the training dataset with Augmentor library

```
!pip install Augmentor
```

```
Collecting Augmentor
  Downloading Augmentor-0.2.12-py2.py3-none-any.whl.metadata (1.3 kB)
Requirement already satisfied: Pillow>=5.2.0 in /usr/local/lib/python3.11/dist-packages (from Augmentor) (11.1.0)
Requirement already satisfied: tqdm>=4.9.0 in /usr/local/lib/python3.11/dist-packages (from Augmentor) (4.67.1)
Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from Augmentor) (1.26.4)
Downloading Augmentor-0.2.12-py2.py3-none-any.whl (38 kB)
Installing collected packages: Augmentor
Successfully installed Augmentor-0.2.12
```

To use Augmentor, the following general procedure is followed:

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

```
path_to_training_dataset="/content/drive/My Drive/Melanoma Detection/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/drive/My Drive/Melanoma Detection/Train/actinic keratosis/output.Processing <PIL.Image.Image image mode=RGB s
Initialised with 376 image(s) found.
Output directory set to /content/drive/My Drive/Melanoma Detection/Train/basal cell carcinoma/output.Processing <PIL.Image.Image image mode=RG
Initialised with 95 image(s) found.
Output directory set to /content/drive/My Drive/Melanoma Detection/Train/dermatofibroma/output.Processing <PIL.Image.Image image mode=RGB size
Initialised with 438 image(s) found.
Output directory set to /content/drive/My Drive/Melanoma Detection/Train/melanoma/output.Processing <PIL.Image.Image image mode=RGB size=962x6
Initialised with 357 image(s) found.
Output directory set to /content/drive/My Drive/Melanoma Detection/Train/nevus/output.Processing <PIL.Image.Image image mode=RGB size=767x576
Initialised with 462 image(s) found.
Output directory set to /content/drive/My Drive/Melanoma Detection/Train/pigmented benign keratosis/output.Processing <PIL.Image.Image image m
Initialised with 77 image(s) found.
Output directory set to /content/drive/My Drive/Melanoma Detection/Train/seborrheic keratosis/output.Processing <PIL.Image.Image image mode=RG
Initialised with 181 image(s) found.
Output directory set to /content/drive/My Drive/Melanoma Detection/Train/squamous cell carcinoma/output.Processing <PIL.Image.Image image mode
Initialised with 139 image(s) found.
Output directory set to /content/drive/My Drive/Melanoma Detection/Train/vascular lesion/output.Processing <PIL.Image.Image image mode=RGB siz
```

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.

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

```
4500
```

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

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

Create a training dataset

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir_train, batch_size=32, image_size=(180, 180), label_mode='categorical',
    seed=123, subset="training", validation_split=0.2
)
```

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

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

Create a 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),
  batch_size=batch_size)
```

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

Model Building & training on the rectified class imbalance data

1. 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).
2. Choosing an appropriate optimiser and loss function for model training Training the model for ~30 epochs
3. Plotting Graph for findings after the model fit to check if there is any evidence of model overfit or underfit.

Model without normalization

```
# 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'))


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

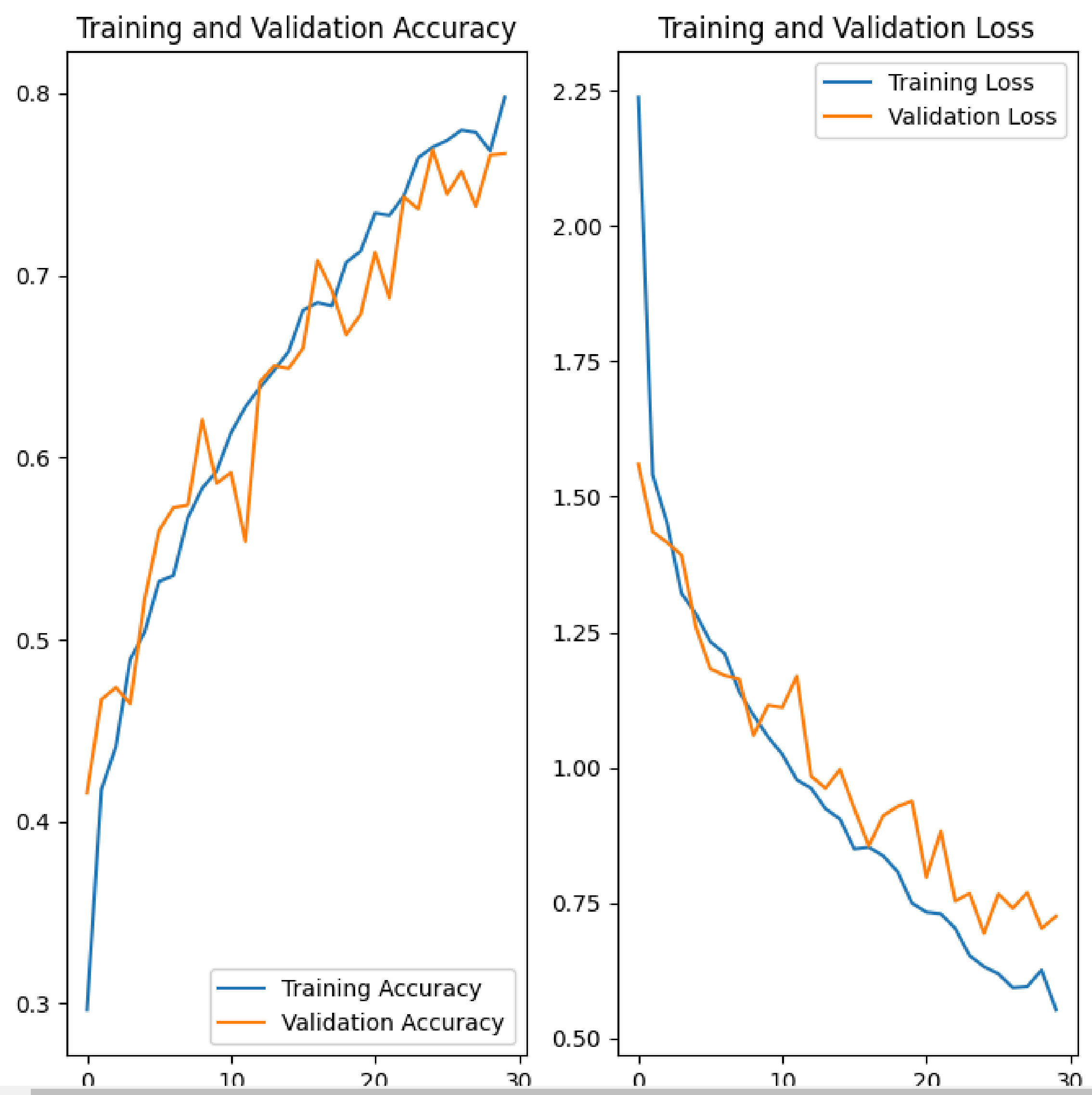
```
Model: "sequential_5"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃          Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━┩
│ rescaling_4 (Rescaling)              │ (None, 180, 180, 3)         │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ sequential_1 (Sequential)            │ (None, 180, 180, 3)         │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ conv2d_8 (Conv2D)                    │ (None, 180, 180, 64)        │            1,792 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ max_pooling2d_8 (MaxPooling2D)       │ (None, 90, 90, 64)          │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ conv2d_9 (Conv2D)                    │ (None, 90, 90, 128)         │           73,856 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ max_pooling2d_9 (MaxPooling2D)       │ (None, 45, 45, 128)         │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dropout_3 (Dropout)                  │ (None, 45, 45, 128)         │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ flatten_4 (Flatten)                  │ (None, 259200)              │                0 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dense_8 (Dense)                      │ (None, 256)                 │       66,355,456 │
├──────────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dense_9 (Dense)                      │ (None, 9)                   │            2,313 │
└──────────────────────────────────────┴─────────────────────────────┴──────────────────┘
 Total params: 66,433,417 (253.42 MB)
 Trainable params: 66,433,417 (253.42 MB)
```

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

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()
```



Model with normalization

```
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'))


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

```
Model: "sequential_6"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ rescaling_5 (Rescaling)              │ (None, 180, 180, 3)         │               0 │
```

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