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.

I. Data Reading/Data Understanding Skin Cancer Data

Importing all the important libraries

```
# importing libraries required
import pathlib
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import os
from glob import glob
import math
import PIL
from tensorflow import keras
from tensorflow.keras import layers,regularizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import ReduceLROnPlateau
```

```
# mounting google drive to get dataset from drive
from google.colab import drive
drive.mount('/content/gdrive')
```

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

```
#unzip the dataset zip file and save it in a folder if zip file present in Drive
#!unzip "/content/gdrive/My Drive/Colab Notebooks/CNN_assignment.zip" -d  "/content/gdrive/My Drive/images_melanoma"> /dev/null
```

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
data_dir_train = pathlib.Path("/content/gdrive/My Drive/images_melanoma/Skin cancer ISIC The International Skin Imaging Collaboration/Tr
data_dir_test = pathlib.Path('/content/gdrive/My Drive/images_melanoma/Skin cancer ISIC The International Skin Imaging Collaboration/Tes
```

```
# To check training and test data set size using glob
# Train images count
image_count_train = len(list(data_dir_train.glob('*/*.jpg')))
print("Training images available in dataset: ",image_count_train)
#Test Images count
image_count_test = len(list(data_dir_test.glob('*/*.jpg')))
print("Test images available in dataset: ", image_count_test)
```

```
Training images available in dataset:  2239
Test images available in dataset:  118
```

Load using keras.preprocessing

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

II. Dataset Creation

Define some parameters for the loader as per problem statement:

```
# parameters set as per problem statement
batch_size = 32
img_height = 180
img_width = 180
```

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

```
## loading and preprocessing images for Training from training directory and resize images to 180x180
## used seed=123 to keep consistency in random selection
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir_train,
  seed=123,
  validation_split=0.2,
  subset="training",
  image_size=(img_height, img_width),
  label_mode='categorical',
  batch_size=batch_size)
```

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

```
## loading and preprocessing images for Training from training directory and resize images to 180x180
## used seed=123 to keep consistency in random selection
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',
  batch_size=batch_size)
```

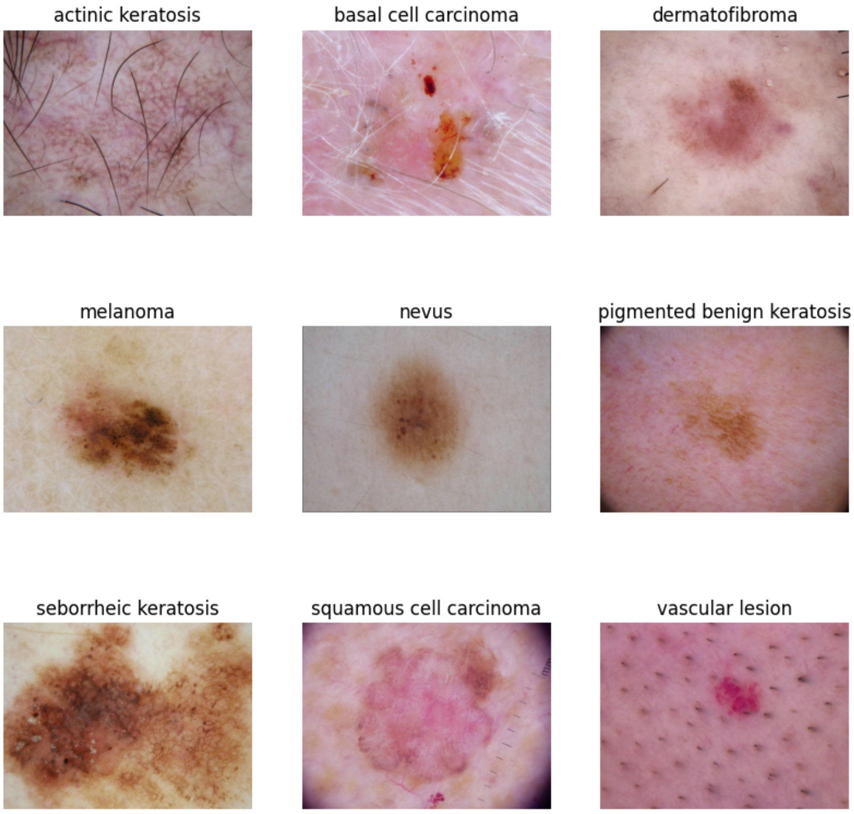
```
Found 2239 files belonging to 9 classes.
Using 447 files for validation.
```

```
# List out all the classes of skin cancer and store them in a list using class_names attribute
class_names = train_ds.class_names
print(class_names)
```

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

III. Dataset visualisation

```
plt.figure(figsize=(10,10))
# to display one image per class
for i in range(len(class_names)):
  plt.subplot(math.ceil(len(class_names)/3),3,i+1)
  class_image = plt.imread(str(list(data_dir_train.glob(f'{class_names[i]}/*.jpg'))[2]))
  plt.title(class_names[i])
  plt.imshow(class_image)
  plt.axis('off')
```



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.

```
# to control degree of parallelism
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)
```

IV. Model Building & training on provided data

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]`

```
# Initialization of Sequential CNN framework
model = Sequential()
# Rescales the pixel values of the input images to the range [0, 1]
model.add(layers.experimental.preprocessing.Rescaling(1./255, input_shape=(img_height, img_width,3)))

# Convolutional Layers
# padding 'same' is selected to ensure no information loss(spatial dimensions of output equals to input)
# RELU activation function is used except for output layer(Softmax)
# First Convulation layer
model.add(layers.Conv2D(32,kernel_size=(3,3),padding='same',activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))

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

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

# Dropout layer with 50% Fraction of the input units to drop.
model.add(layers.Dropout(0.25))

# To flatten the multi-dimensional input tensors into a single dimension
model.add(layers.Flatten())
# Dense Layer
model.add(layers.Dense(128,activation='relu'))
# Dropout layer with 25% Fraction of the input units to drop.
model.add(layers.Dropout(0.25))
# Dense Layer with softmax activation function.
model.add(layers.Dense(len(class_names),activation='softmax'))
```

Compile the model

Choose an appropirate optimiser and loss function for model training

```
#Adam optimization: is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments
#categorical_crossentropy is choosen as loss function.

# To reduce learning rate if no improvement seen in validation accuracy for consecutive 5 epochs

learn_control = ReduceLROnPlateau(monitor='val_accuracy', patience=5,
                                  verbose=1,factor=0.2, min_lr=1e-7)

model.compile(optimizer=tf.keras.optimizers.Adam(),loss=tf.keras.losses.CategoricalCrossentropy(),metrics=["accuracy"])
```

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

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

 conv2d (Conv2D)             (None, 180, 180, 32)      896

 max_pooling2d (MaxPooling2  (None, 90, 90, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 90, 90, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 45, 45, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 45, 45, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 22, 22, 128)       0
 g2D)

 dropout (Dropout)           (None, 22, 22, 128)       0

 flatten (Flatten)           (None, 61952)             0

 dense (Dense)               (None, 128)               7929984

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 9)                 1161
```

```
=================================================================
Total params: 8024393 (30.61 MB)
Trainable params: 8024393 (30.61 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

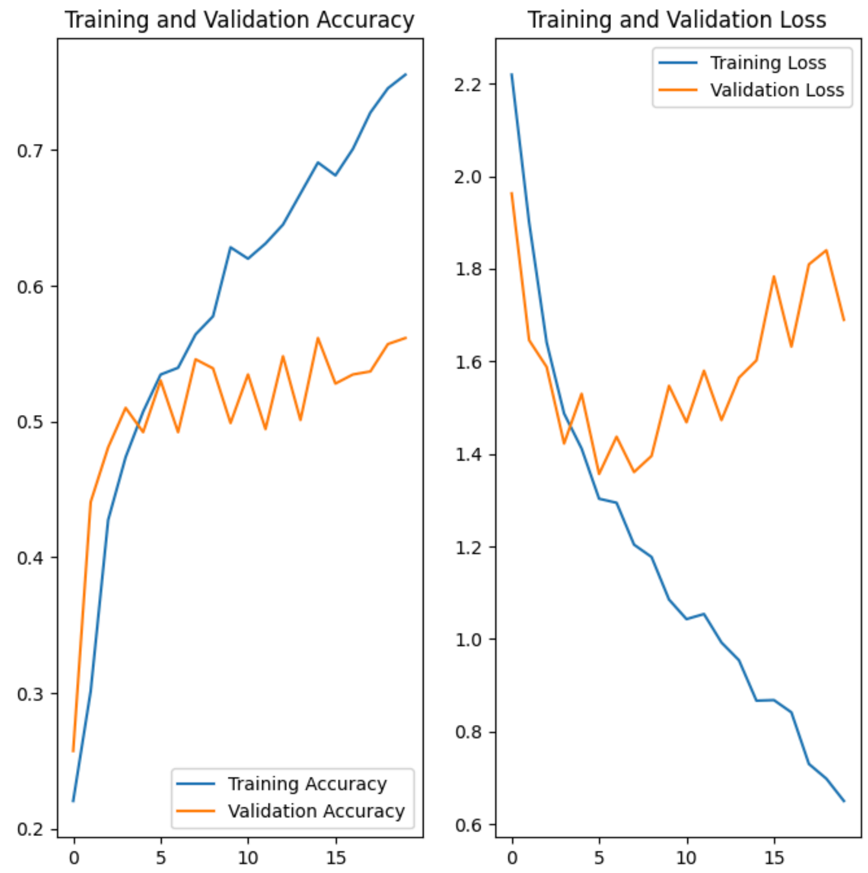
Train the model

```
# We are using 20 epochs to train the model
epochs = 20
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs,
  callbacks=[learn_control]
)
```

```
Epoch 1/20
56/56 [==============================] - 260s 937ms/step - loss: 2.2192 - accuracy: 0.2204 - val_loss: 1.9627 - val_accuracy: 0.2573
Epoch 2/20
56/56 [==============================] - 3s 59ms/step - loss: 1.8996 - accuracy: 0.3013 - val_loss: 1.6455 - val_accuracy: 0.4407 -
Epoch 3/20
56/56 [==============================] - 3s 54ms/step - loss: 1.6408 - accuracy: 0.4275 - val_loss: 1.5878 - val_accuracy: 0.4810 -
Epoch 4/20
56/56 [==============================] - 3s 57ms/step - loss: 1.4871 - accuracy: 0.4738 - val_loss: 1.4224 - val_accuracy: 0.5101 -
Epoch 5/20
56/56 [==============================] - 3s 59ms/step - loss: 1.4115 - accuracy: 0.5073 - val_loss: 1.5299 - val_accuracy: 0.4922 -
Epoch 6/20
56/56 [==============================] - 3s 55ms/step - loss: 1.3031 - accuracy: 0.5346 - val_loss: 1.3566 - val_accuracy: 0.5302 -
Epoch 7/20
56/56 [==============================] - 3s 55ms/step - loss: 1.2944 - accuracy: 0.5396 - val_loss: 1.4371 - val_accuracy: 0.4922 -
Epoch 8/20
56/56 [==============================] - 3s 55ms/step - loss: 1.2039 - accuracy: 0.5642 - val_loss: 1.3607 - val_accuracy: 0.5459 -
Epoch 9/20
56/56 [==============================] - 3s 60ms/step - loss: 1.1773 - accuracy: 0.5776 - val_loss: 1.3955 - val_accuracy: 0.5391 -
Epoch 10/20
56/56 [==============================] - 3s 55ms/step - loss: 1.0853 - accuracy: 0.6283 - val_loss: 1.5470 - val_accuracy: 0.4989 -
Epoch 11/20
56/56 [==============================] - 3s 54ms/step - loss: 1.0431 - accuracy: 0.6200 - val_loss: 1.4683 - val_accuracy: 0.5347 -
Epoch 12/20
56/56 [==============================] - 3s 54ms/step - loss: 1.0541 - accuracy: 0.6311 - val_loss: 1.5796 - val_accuracy: 0.4944 -
Epoch 13/20
56/56 [==============================] - 3s 56ms/step - loss: 0.9926 - accuracy: 0.6451 - val_loss: 1.4730 - val_accuracy: 0.5481 -
Epoch 14/20
56/56 [==============================] - 3s 55ms/step - loss: 0.9543 - accuracy: 0.6680 - val_loss: 1.5646 - val_accuracy: 0.5011 -
Epoch 15/20
56/56 [==============================] - 3s 55ms/step - loss: 0.8669 - accuracy: 0.6908 - val_loss: 1.6020 - val_accuracy: 0.5615 -
Epoch 16/20
56/56 [==============================] - 3s 59ms/step - loss: 0.8681 - accuracy: 0.6814 - val_loss: 1.7833 - val_accuracy: 0.5280 -
Epoch 17/20
56/56 [==============================] - 3s 55ms/step - loss: 0.8417 - accuracy: 0.7009 - val_loss: 1.6317 - val_accuracy: 0.5347 -
Epoch 18/20
56/56 [==============================] - 3s 55ms/step - loss: 0.7302 - accuracy: 0.7277 - val_loss: 1.8093 - val_accuracy: 0.5369 -
Epoch 19/20
56/56 [==============================] - 3s 55ms/step - loss: 0.6984 - accuracy: 0.7455 - val_loss: 1.8398 - val_accuracy: 0.5570 -
Epoch 20/20
56/56 [==============================] - ETA: 0s - loss: 0.6503 - accuracy: 0.7556
Epoch 20: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
56/56 [==============================] - 3s 55ms/step - loss: 0.6503 - accuracy: 0.7556 - val_loss: 1.6897 - val_accuracy: 0.5615 -
```

Visualizing training results

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

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

epochs_range = range(epochs)

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

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



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

Write your findings here

Findings

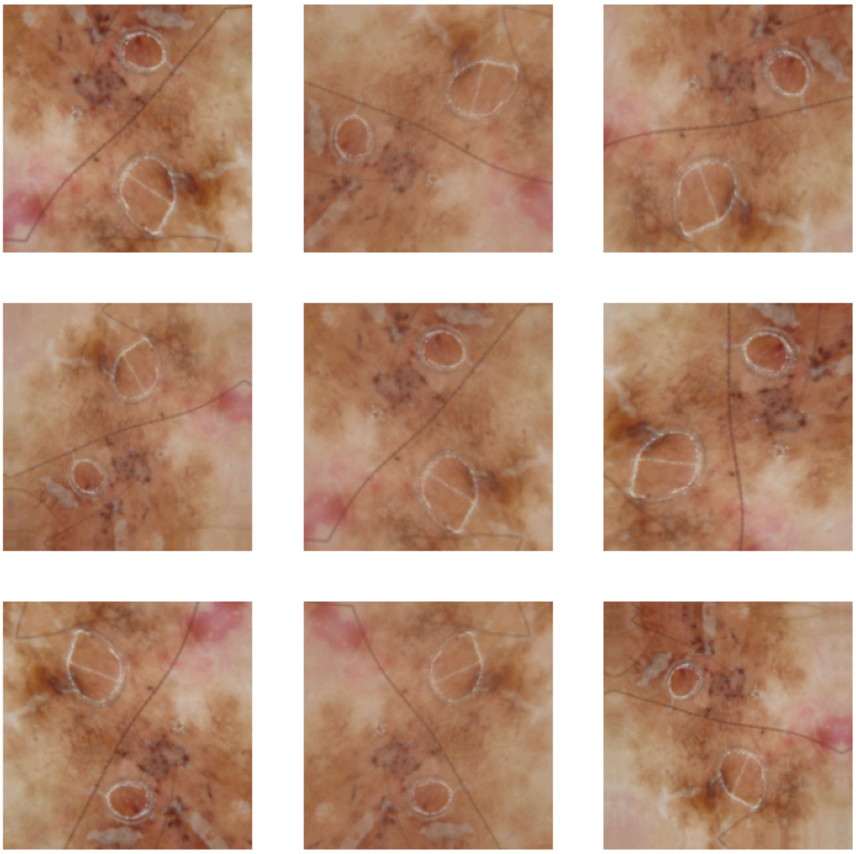
1. The difference between Traing accuracy and validation accuracy kept on increasing through epochs which suggests that current model is overfitting
2. Training loss is very low, but validation loss is fluctuating.

Thus we can collect that the model is not good.

V. Model Building & training on Augmented data

```
# Todo, after you have analysed the model fit history for presence of underfit or overfit, choose an appropriate data augumentation stra
data_augmentation = tf.keras.Sequential(
  [
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical",
                                                 input_shape=(img_height,
                                                              img_width,
                                                              3)),
    layers.experimental.preprocessing.RandomRotation(0.2),
    layers.experimental.preprocessing.RandomZoom(0.2),
    layers.experimental.preprocessing.RandomContrast(0.2)
  ]
)
```

```
# Visualising augmented images for one instance of training image.
plt.figure(figsize=(10, 10))
for image, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(image)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```



Todo:

Create the model, compile and train the model

```
## You can use Dropout layer if there is an evidence of overfitting in your findings
# Initialization of Sequential CNN framework
model = Sequential()
# Rescales the pixel values of the input images to the range [0, 1]
model.add(layers.experimental.preprocessing.Rescaling(1./255, input_shape=(img_height, img_width,3)))
# Data augmentation
model.add(data_augmentation)
# Convolutional Layers
# padding 'same' is selected to ensure no information loss(spatial dimensions of output equals to input)
# RELU activation function is used except for output layer(Softmax)
# First Convulation layer
model.add(layers.Conv2D(32,kernel_size=(3,3),padding='same',activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))

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

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

# Dropout layer with 50% Fraction of the input units to drop.
model.add(layers.Dropout(0.25))

# To flatten the multi-dimensional input tensors into a single dimension
model.add(layers.Flatten())
# Dense Layer
model.add(layers.Dense(128,activation='relu'))
#Dropout layer with 25% Fraction of the input units to drop.
model.add(layers.Dropout(0.25))

# Dense Layer with softmax activation function.
model.add(layers.Dense(len(class_names),activation='softmax'))

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_3 (Conv2D)           (None, 180, 180, 32)      896

 max_pooling2d_3 (MaxPoolin  (None, 90, 90, 32)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 90, 90, 64)        18496

 max_pooling2d_4 (MaxPoolin  (None, 45, 45, 64)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 45, 45, 128)       73856

 max_pooling2d_5 (MaxPoolin  (None, 22, 22, 128)       0
 g2D)

 dropout_2 (Dropout)         (None, 22, 22, 128)       0

 flatten_1 (Flatten)         (None, 61952)             0

 dense_2 (Dense)             (None, 128)               7929984

 dropout_3 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 9)                 1161

=================================================================
Total params: 8024393 (30.61 MB)
Trainable params: 8024393 (30.61 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compiling the model

```
# Compiling the model with learning rate control
learn_control = ReduceLROnPlateau(monitor='val_accuracy', patience=5,
                                  verbose=1,factor=0.2, min_lr=1e-7)

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss=tf.keras.losses.CategoricalCrossentropy(),
              metrics=["accuracy"])
```

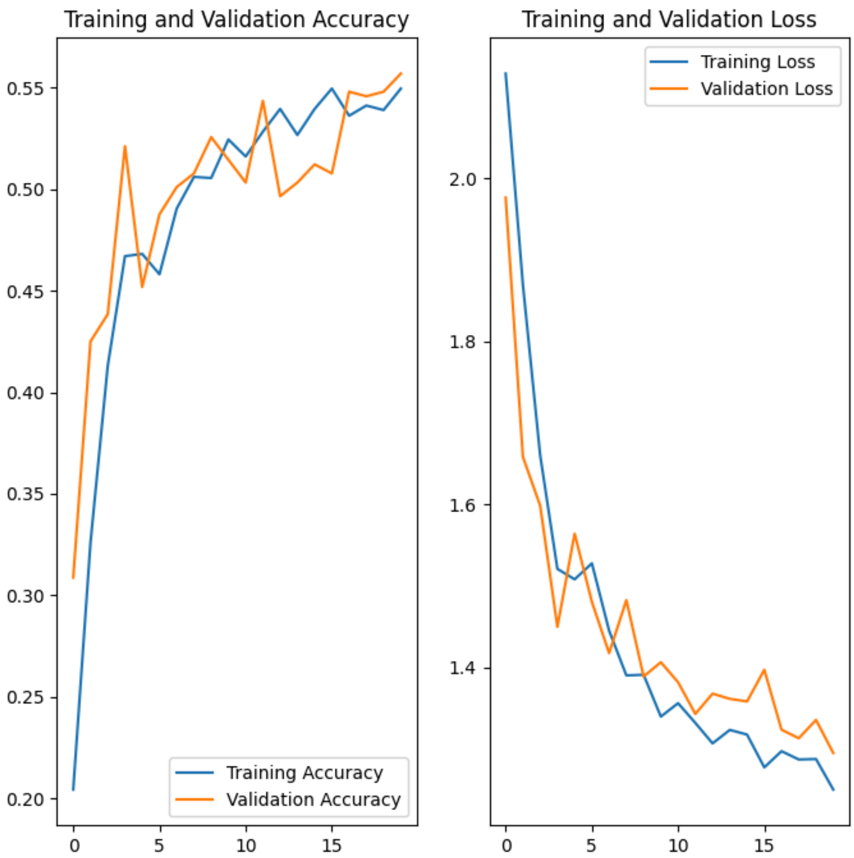
Training the model

```
## Training  Augmented model for 20 epochs
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs,
  callbacks=[learn_control]
)
```

```
Epoch 1/20
56/56 [==============================] - 6s 61ms/step - loss: 2.1284 - accuracy: 0.2042 - val_loss: 1.9763 - val_accuracy: 0.3087 -
Epoch 2/20
56/56 [==============================] - 3s 62ms/step - loss: 1.8693 - accuracy: 0.3259 - val_loss: 1.6583 - val_accuracy: 0.4251 -
Epoch 3/20
56/56 [==============================] - 3s 58ms/step - loss: 1.6604 - accuracy: 0.4129 - val_loss: 1.5985 - val_accuracy: 0.4385 -
Epoch 4/20
56/56 [==============================] - 3s 60ms/step - loss: 1.5209 - accuracy: 0.4671 - val_loss: 1.4498 - val_accuracy: 0.5213 -
Epoch 5/20
56/56 [==============================] - 3s 57ms/step - loss: 1.5081 - accuracy: 0.4682 - val_loss: 1.5640 - val_accuracy: 0.4519 -
Epoch 6/20
56/56 [==============================] - 3s 56ms/step - loss: 1.5277 - accuracy: 0.4581 - val_loss: 1.4804 - val_accuracy: 0.4877 -
Epoch 7/20
56/56 [==============================] - 3s 56ms/step - loss: 1.4451 - accuracy: 0.4905 - val_loss: 1.4176 - val_accuracy: 0.5011 -
Epoch 8/20
56/56 [==============================] - 3s 60ms/step - loss: 1.3902 - accuracy: 0.5061 - val_loss: 1.4825 - val_accuracy: 0.5078 -
Epoch 9/20
56/56 [==============================] - 3s 56ms/step - loss: 1.3910 - accuracy: 0.5056 - val_loss: 1.3889 - val_accuracy: 0.5257 -
Epoch 10/20
56/56 [==============================] - 3s 57ms/step - loss: 1.3397 - accuracy: 0.5246 - val_loss: 1.4064 - val_accuracy: 0.5145 -
Epoch 11/20
56/56 [==============================] - 3s 57ms/step - loss: 1.3561 - accuracy: 0.5162 - val_loss: 1.3819 - val_accuracy: 0.5034 -
Epoch 12/20
56/56 [==============================] - 3s 57ms/step - loss: 1.3319 - accuracy: 0.5285 - val_loss: 1.3430 - val_accuracy: 0.5436 -
Epoch 13/20
56/56 [==============================] - 3s 56ms/step - loss: 1.3069 - accuracy: 0.5396 - val_loss: 1.3678 - val_accuracy: 0.4966 -
Epoch 14/20
56/56 [==============================] - 3s 57ms/step - loss: 1.3233 - accuracy: 0.5268 - val_loss: 1.3615 - val_accuracy: 0.5034 -
Epoch 15/20
56/56 [==============================] - 3s 59ms/step - loss: 1.3177 - accuracy: 0.5396 - val_loss: 1.3582 - val_accuracy: 0.5123 -
Epoch 16/20
56/56 [==============================] - 3s 56ms/step - loss: 1.2774 - accuracy: 0.5497 - val_loss: 1.3971 - val_accuracy: 0.5078 -
Epoch 17/20
56/56 [==============================] - 3s 55ms/step - loss: 1.2973 - accuracy: 0.5363 - val_loss: 1.3236 - val_accuracy: 0.5481 -
Epoch 18/20
56/56 [==============================] - 3s 57ms/step - loss: 1.2871 - accuracy: 0.5413 - val_loss: 1.3132 - val_accuracy: 0.5459 -
Epoch 19/20
56/56 [==============================] - 3s 57ms/step - loss: 1.2876 - accuracy: 0.5391 - val_loss: 1.3356 - val_accuracy: 0.5481 -
Epoch 20/20
56/56 [==============================] - 3s 56ms/step - loss: 1.2502 - accuracy: 0.5497 - val_loss: 1.2950 - val_accuracy: 0.5570 -
```

Visualizing the results

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

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

epochs_range = range(epochs)

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

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



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

Findings

1. The Training accuracy and validation accuracy are almost same. This is a sign of good fit but the accuracy is still very low(underfit). The model requires more epochs to train with class imbalance handled.

VI. Model Building & training on data after handling class imbalance

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.

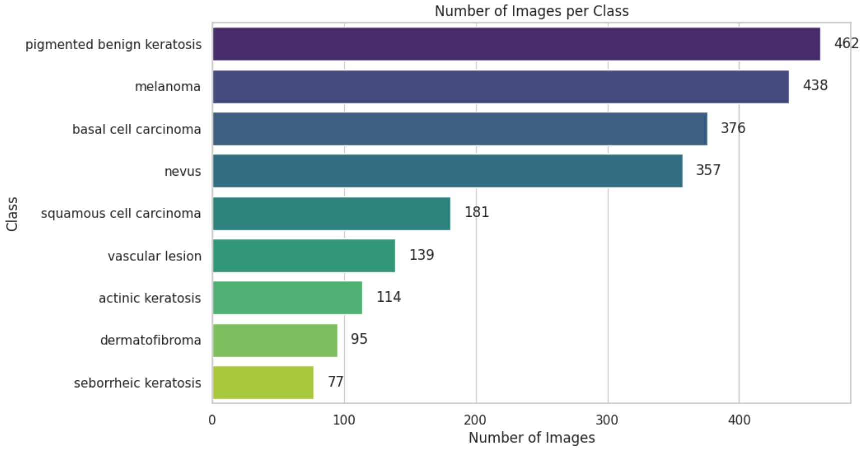
```
# images count in each classes

class_imbalance = pd.DataFrame()
for i in range(len(class_names)):
    cls_name = class_names[i]
    number = len(list(data_dir_train.glob(f'{class_names[i]}/*.jpg')))
    class_imbalance = pd.concat([class_imbalance, pd.DataFrame({'class': [cls_name], 'number': [number]})])

class_imbalance.reset_index(drop=True, inplace=True)
class_imbalance
```

	class	number
0	actinic keratosis	114
1	basal cell carcinoma	376
2	dermatofibroma	95
3	melanoma	438
4	nevus	357
5	pigmented benign keratosis	462
6	seborrheic keratosis	77
7	squamous cell carcinoma	181
8	vascular lesion	139

Next steps: Generate code with class_imbalance View recommended plots

```
class_imbalance_sorted = class_imbalance.sort_values(by='number', ascending=False)

# Plotting
plt.figure(figsize=(10, 6))
sns.set_theme(style="whitegrid")
sns.barplot(x='number', y='class', data=class_imbalance_sorted, palette='viridis')

# Annotate each bar with its count
for i in range(len(class_imbalance_sorted)):
    count = class_imbalance_sorted.iloc[i]['number']
    plt.text(count + 10, i, str(count), va='center', fontsize=12)

plt.xlabel('Number of Images')
plt.ylabel('Class')
plt.title('Number of Images per Class')
plt.show()
```

```
<ipython-input-25-12412dc28a00>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.

  sns.barplot(x='number', y='class', data=class_imbalance_sorted, palette='viridis')
```



Todo: Write your findings here:

- Which class has the least number of samples?

seborrheic keratosis (77)

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

pigmented benign keratosis (462)

Todo: Rectify the class imbalance

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

```
!pip install Augmentor
```

```
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/dist-packages (from Augmentor) (4.66.4)
Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from Augmentor) (1.25.2)
Installing collected packages: Augmentor
Successfully installed Augmentor-0.2.12
```

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

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

```
path_to_training_dataset="/content/gdrive/My Drive/images_melanoma/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=15, max_right_rotation=15)
    p.sample(500) ## We are adding 500 samples per class to make sure that none of the classes are sparse.
```

```
ing <PIL.Image.Image image mode=RGB size=600x450 at 0x7E5D42403E50>: 100%|██████████| 500/500 [00:21<00:00, 22.82 Samples/s]
```

```
essing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=600x450 at 0x7E5D42413D60>: 100%|██████████| 500/500 [00:22<00:00, 21.8

 <PIL.Image.Image image mode=RGB size=600x450 at 0x7E5D426AB4F0>: 100%|██████████| 500/500 [00:20<00:00, 24.61 Samples/s]

Image.Image image mode=RGB size=2048x1536 at 0x7E5DB01E2A10>: 100%|██████████| 500/500 [01:24<00:00,  5.95 Samples/s]

ge.Image image mode=RGB size=1024x768 at 0x7E5DB7F6B370>: 100%|██████████| 500/500 [01:25<00:00,  5.82 Samples/s]

t.Processing <PIL.Image.Image image mode=RGB size=600x450 at 0x7E5D427CA500>: 100%|██████████| 500/500 [00:20<00:00, 24.53 Samples/s

essing <PIL.Image.Image image mode=RGB size=1024x768 at 0x7E5D426DF1F0>: 100%|██████████| 500/500 [00:39<00:00, 12.68 Samples/s]

rocessing <PIL.Image.Image image mode=RGB size=600x450 at 0x7E5D42543670>: 100%|██████████| 500/500 [00:20<00:00, 24.13 Samples/s]

g <PIL.Image.Image image mode=RGB size=600x450 at 0x7E5D426DF670>: 100%|██████████| 500/500 [00:19<00:00, 25.61 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.

```
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.

```
# newly augmented images
path_list = [x for x in glob(os.path.join(data_dir_train, '*','output', '*.jpg'))]
len(path_list)
```

```
4500
```

```
# To get second level directories names which contain output directory with augmented class balanced images
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
len(lesion_list_new)
```

```
4500
```

```
# To map output files to subdirectories and store them as dict
dataframe_dict_new = dict(zip(path_list, lesion_list_new))
```

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

	Path	Label
0	/content/gdrive/My Drive/images_melanoma/Skin ...	squamous cell carcinoma
1	/content/gdrive/My Drive/images_melanoma/Skin ...	squamous cell carcinoma
2	/content/gdrive/My Drive/images_melanoma/Skin ...	squamous cell carcinoma
3	/content/gdrive/My Drive/images_melanoma/Skin ...	squamous cell carcinoma
4	/content/gdrive/My Drive/images_melanoma/Skin ...	squamous cell carcinoma

Next steps: Generate code with path_df View recommended plots

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

```
Label
squamous cell carcinoma       500
actinic keratosis             500
seborrheic keratosis          500
vascular lesion               500
nevus                         500
pigmented benign keratosis    500
dermatofibroma                500
melanoma                      500
basal cell carcinoma          500
Name: count, 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.

Todo: Train the model on the data created using Augmentor

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

```
len(glob(os.path.join(data_dir_train, 'seborrheic keratosis','output', '*.jpg')))
```

```
500
```

Todo: Create a training dataset

```
data_dir_train="/content/gdrive/My Drive/images_melanoma/Skin cancer ISIC The International Skin Imaging Collaboration/Train/"
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir_train,
  seed=123,
  validation_split = 0.2,
  subset = 'training',
  label_mode='categorical',
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

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

Todo: Create a validation dataset

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

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

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

```
## your code goes here

## You can use Dropout layer if there is an evidence of overfitting in your findings
# Initialization of Sequential CNN framework
model = Sequential()
# Rescales the pixel values of the input images to the range [0, 1]
model.add(layers.experimental.preprocessing.Rescaling(1./255, input_shape=(img_height, img_width,3)))

# Convolutional Layers
# padding 'same' is selected to ensure no information loss(spatial dimensions of output equals to input)
# RELU activation function is used except for output layer(Softmax)
# First Convulation layer
model.add(layers.Conv2D(32,kernel_size=(3,3),padding='same',activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))

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

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

# Fourth Convulation Layer
model.add(layers.Conv2D(256,kernel_size=(5,5),padding='same',activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))

# Dropout layer with 50% Fraction of the input units to drop.
model.add(layers.Dropout(0.25))

# To flatten the multi-dimensional input tensors into a single dimension
model.add(layers.Flatten())
# Dense and Dropout Layers
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dropout(0.25))

model.add(layers.Dense(128,activation='relu'))
model.add(layers.Dropout(0.25))

model.add(layers.Dense(64,activation='relu'))
#Dropout layer with 25% Fraction of the input units to drop.
model.add(layers.Dropout(0.25))

# Dense Layer with softmax activation function.
model.add(layers.Dense(len(class_names),activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_4 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_10 (Conv2D)          (None, 180, 180, 32)      896

 max_pooling2d_10 (MaxPooli  (None, 90, 90, 32)        0
 ng2D)

 conv2d_11 (Conv2D)          (None, 90, 90, 64)        18496

 max_pooling2d_11 (MaxPooli  (None, 45, 45, 64)        0
 ng2D)

 conv2d_12 (Conv2D)          (None, 45, 45, 128)       73856

 max_pooling2d_12 (MaxPooli  (None, 22, 22, 128)       0
 ng2D)

 conv2d_13 (Conv2D)          (None, 22, 22, 256)       819456

 max_pooling2d_13 (MaxPooli  (None, 11, 11, 256)       0
 ng2D)

 dropout_11 (Dropout)        (None, 11, 11, 256)       0

 flatten_3 (Flatten)         (None, 30976)             0

 dense_8 (Dense)             (None, 256)               7930112

 dropout_12 (Dropout)        (None, 256)               0

 dense_9 (Dense)             (None, 128)               32896
```

```
 dropout_13 (Dropout)        (None, 128)               0

 dense_10 (Dense)            (None, 64)                8256

 dropout_14 (Dropout)        (None, 64)                0

 dense_11 (Dense)            (None, 9)                 585

=================================================================
Total params: 8884553 (33.89 MB)
Trainable params: 8884553 (33.89 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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

```
# Compiling the model with learning rate control
learn_control = ReduceLROnPlateau(monitor='val_accuracy', patience=5,
                                  verbose=1,factor=0.2, min_lr=1e-7)

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss=tf.keras.losses.CategoricalCrossentropy(),
              metrics=["accuracy"])
```

Todo: Train your model

```
epochs = 30
## Training the model for 30 epochs
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs,
  callbacks=[learn_control]
)
```

```
Epoch 1/30
169/169 [==============================] - 46s 243ms/step - loss: 2.1592 - accuracy: 0.1589 - val_loss: 1.9813 - val_accuracy: 0.
Epoch 2/30
169/169 [==============================] - 44s 252ms/step - loss: 1.8699 - accuracy: 0.2800 - val_loss: 1.6537 - val_accuracy: 0.
Epoch 3/30
169/169 [==============================] - 42s 242ms/step - loss: 1.6907 - accuracy: 0.3427 - val_loss: 1.6721 - val_accuracy: 0.
Epoch 4/30
169/169 [==============================] - 43s 245ms/step - loss: 1.5341 - accuracy: 0.3978 - val_loss: 1.3858 - val_accuracy: 0.
Epoch 5/30
169/169 [==============================] - 42s 240ms/step - loss: 1.4203 - accuracy: 0.4432 - val_loss: 1.2567 - val_accuracy: 0.
Epoch 6/30
169/169 [==============================] - 42s 244ms/step - loss: 1.3651 - accuracy: 0.4750 - val_loss: 1.3883 - val_accuracy: 0.
Epoch 7/30
```