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

Importing Skin Cancer Data

To do: Take necessary actions to read the data

Importing all the important libraries

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

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

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

```
    Mounted at /gdrive
```

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

```
# Defining the path for train and test images
## Todo: Update the paths of the train and test dataset
path_to_training_dataset = "/gdrive/My Drive/CNN_assignment/Train"
path_to_test_dataset = '/gdrive/My Drive/CNN_assignment/Test'

data_dir_train = pathlib.Path(path_to_training_dataset)
data_dir_test = pathlib.Path(path_to_test_dataset)
```

```
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_dat
## Note, make sure your resize your images to the size img_height*img_width, while writ
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir_train,
  validation_split=0.2,
  labels='inferred',
  label_mode='categorical',
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

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

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

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

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

```
['actinic keratosis', 'basal cell carcinoma', 'dermatofibroma', 'melanoma', 'nevu
```

```
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir_test,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

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

Visualize the data

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

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

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

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

index = 0
plt.figure(figsize=(15,15))
for c in class_names:
    path_list = files_path_dict[c][:1]
    index += 1
    plt.subplot(3,3,index)
    plt.imshow(load_img(path_list[0],target_size=(180,180)))
    plt.title(c)
    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.

```
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]`

```
### Your code goes here
from tensorflow.keras.layers import BatchNormalization

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

model.add(layers.experimental.preprocessing.Rescaling(1./255,input_shape=(180,180,3)))


#First Convulation layer
model.add(layers.Conv2D(32,kernel_size=(3,3),activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))

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

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

#Dropout layer with 50% Fraction of the input units to drop.
model.add(layers.Dropout(0.5))

#Flatten Layer
##Keras.layers.flatten function flattens the multi-dimensional input tensors into a sir
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.
#Softmax is an activation function that scales numbers/logits into probabilities.
model.add(layers.Dense(len(class_names),activation='softmax'))
```

Compile the model

Choose an appropirate optimiser and loss function for model training

```
### Todo, choose an appropirate optimiser and loss function
model.compile(optimizer='Adam',
```

```
              loss='categorical_crossentropy',
              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, 178, 178, 32)      896

 max_pooling2d (MaxPooling2D  (None, 89, 89, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 87, 87, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 43, 43, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 41, 41, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 20, 20, 128)      0
 2D)

 dropout (Dropout)           (None, 20, 20, 128)       0

 flatten (Flatten)           (None, 51200)             0

 dense (Dense)               (None, 128)               6553728

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 9)                 1161

=================================================================
Total params: 6,648,137
Trainable params: 6,648,137
Non-trainable params: 0
_________________________________________________________________
```

Train the model

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

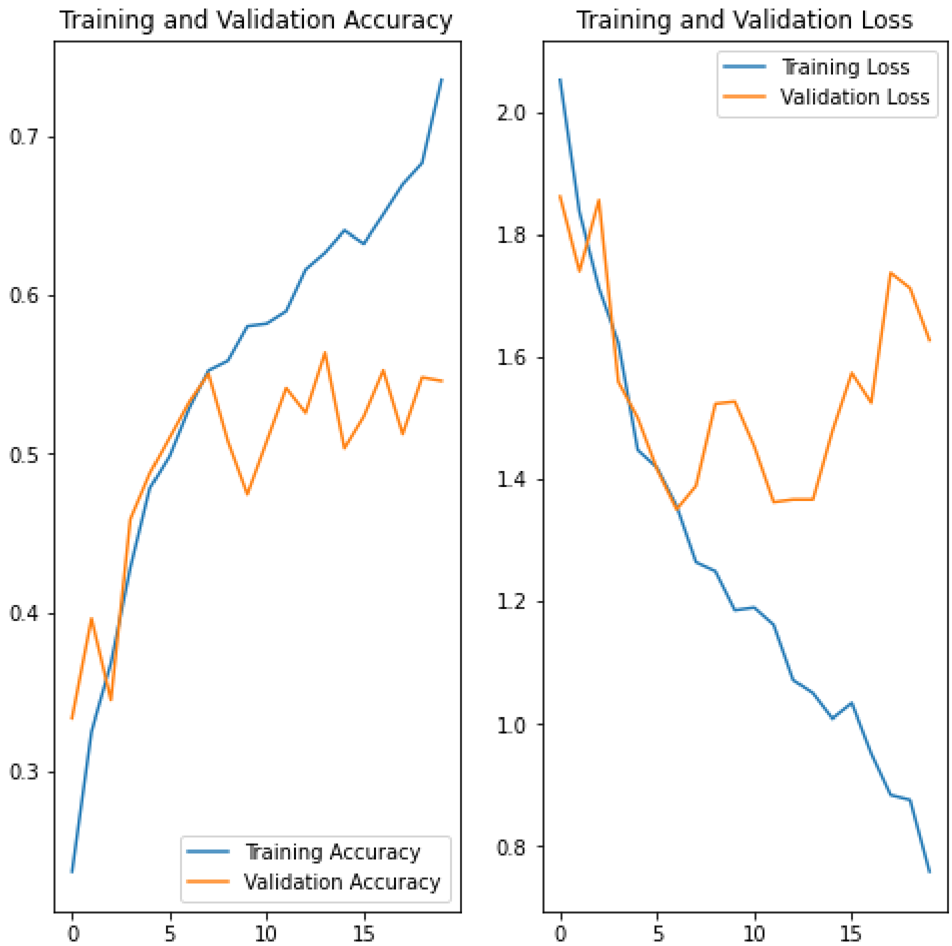
```
Epoch 1/20
56/56 [==============================] - 296s 1s/step - loss: 2.0527 - accuracy:
```

```
Epoch 2/20
56/56 [==============================] - 2s 40ms/step - loss: 1.8364 - accuracy: 
Epoch 3/20
56/56 [==============================] - 2s 39ms/step - loss: 1.7130 - accuracy: 
Epoch 4/20
56/56 [==============================] - 2s 39ms/step - loss: 1.6233 - accuracy: 
Epoch 5/20
56/56 [==============================] - 2s 39ms/step - loss: 1.4477 - accuracy: 
Epoch 6/20
56/56 [==============================] - 2s 39ms/step - loss: 1.4186 - accuracy: 
Epoch 7/20
56/56 [==============================] - 2s 39ms/step - loss: 1.3562 - accuracy: 
Epoch 8/20
56/56 [==============================] - 2s 39ms/step - loss: 1.2641 - accuracy: 
Epoch 9/20
56/56 [==============================] - 2s 39ms/step - loss: 1.2492 - accuracy: 
Epoch 10/20
56/56 [==============================] - 2s 39ms/step - loss: 1.1854 - accuracy: 
Epoch 11/20
56/56 [==============================] - 2s 39ms/step - loss: 1.1898 - accuracy: 
Epoch 12/20
56/56 [==============================] - 2s 39ms/step - loss: 1.1615 - accuracy: 
Epoch 13/20
56/56 [==============================] - 2s 39ms/step - loss: 1.0711 - accuracy: 
Epoch 14/20
56/56 [==============================] - 2s 39ms/step - loss: 1.0498 - accuracy: 
Epoch 15/20
56/56 [==============================] - 2s 39ms/step - loss: 1.0083 - accuracy: 
Epoch 16/20
56/56 [==============================] - 2s 39ms/step - loss: 1.0337 - accuracy: 
Epoch 17/20
56/56 [==============================] - 2s 39ms/step - loss: 0.9516 - accuracy: 
Epoch 18/20
56/56 [==============================] - 2s 39ms/step - loss: 0.8838 - accuracy: 
Epoch 19/20
56/56 [==============================] - 2s 39ms/step - loss: 0.8757 - accuracy: 
Epoch 20/20
56/56 [==============================] - 2s 39ms/step - loss: 0.7584 - accuracy: 
```

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

```
# Todo, after you have analysed the model fit history for presence of underfit or overf
loss, accuracy = model.evaluate(train_ds, verbose=1,)
loss_v, accuracy_v = model.evaluate(val_ds, verbose=1)

print("Accuracy: ", accuracy)
print("Validation Accuracy: ",accuracy_v)
print("Loss: ",loss)
print("Validation Loss", loss_v)

# Thus we can clearly that model Overfit and we need to chose right data augumentation
```

```
56/56 [==============================] - 1s 17ms/step - loss: 0.5957 - accuracy: (
14/14 [==============================] - 0s 16ms/step - loss: 1.6275 - accuracy: (
Accuracy:  0.8002232313156128
Validation Accuracy:  0.5458613038063049
Loss:  0.5956795811653137
Validation Loss 1.6274607181549072
```

```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images

image_class = ['nevus','melanoma','basal_cell_caricoma','actinic_keratosis','vasc_lesic

train_batches = datagen.flow_from_directory(data_dir_train,
    target_size = (180,180),
    classes = image_class,
    batch_size = 64
 )

valid_batches = datagen.flow_from_directory(data_dir_test,
    target_size = (180,180),
    classes = image_class,
    batch_size = 64
)
```

```
Found 890 images belonging to 9 classes.
Found 48 images belonging to 9 classes.
```

```
train_batches
```

```
<keras.preprocessing.image.DirectoryIterator at 0x7f9634b28e50>
```

Todo:

Create the model, compile and train the model

```
## You can use Dropout layer if there is an evidence of overfitting in your findings

## Your code goes here

#Sequential allows you to create models layer-by-layer
```

```
model = Sequential()

model.add(layers.experimental.preprocessing.Rescaling(1./255,input_shape=(180,180,3)))


#First Convulation layer
model.add(layers.Conv2D(32,kernel_size=(3,3),activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))
# Adding Dropout Layer
model.add(layers.Dropout(0.25))

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

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

#Dropout layer with 50% Fraction of the input units to drop.
model.add(layers.Dropout(0.5))

#Flatten Layer
##Keras.layers.flatten function flattens the multi-dimensional input tensors into a sir
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.
#Softmax is an activation function that scales numbers/logits into probabilities.
model.add(layers.Dense(len(class_names),activation='softmax'))
```

Compiling the model

```
## Your code goes here
from tensorflow.keras.optimizers import Adam

optimizer = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0

model.compile(loss='categorical_crossentropy',
                  optimizer=optimizer,
                  metrics=['accuracy'])


# Set a learning rate annealer
from tensorflow.keras.callbacks import ReduceLROnPlateau
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
    patience=3,
```

```
    verbose=1,
    factor=0.5,
    min_lr=0.00001)
```

Training the model

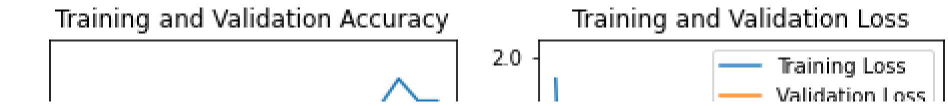
```
## Your code goes here, note: train your model for 20 epochs
epochs = 20

history = model.fit(train_batches,
  epochs = epochs, verbose = 1, validation_data=valid_batches , callbacks=[learning_rat
```

```
Epoch 1/20
14/14 [==============================] - 54s 4s/step - loss: 1.9623 - accuracy: 0
Epoch 2/20
14/14 [==============================] - 31s 2s/step - loss: 1.0564 - accuracy: 0
Epoch 3/20
14/14 [==============================] - 31s 2s/step - loss: 0.9937 - accuracy: 0
Epoch 4/20
14/14 [==============================] - 31s 2s/step - loss: 0.9973 - accuracy: 0
Epoch 5/20
14/14 [==============================] - 31s 2s/step - loss: 0.9358 - accuracy: 0
Epoch 6/20
14/14 [==============================] - 32s 2s/step - loss: 0.8426 - accuracy: 0
Epoch 7/20
14/14 [==============================] - 31s 2s/step - loss: 0.7390 - accuracy: 0
Epoch 8/20
14/14 [==============================] - 31s 2s/step - loss: 0.6929 - accuracy: 0
Epoch 9/20
14/14 [==============================] - 31s 2s/step - loss: 0.7062 - accuracy: 0
Epoch 10/20
14/14 [==============================] - ETA: 0s - loss: 0.6927 - accuracy: 0.671
Epoch 10: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
14/14 [==============================] - 33s 2s/step - loss: 0.6927 - accuracy: 0
Epoch 11/20
14/14 [==============================] - 32s 2s/step - loss: 0.6205 - accuracy: 0
Epoch 12/20
14/14 [==============================] - 31s 2s/step - loss: 0.6178 - accuracy: 0
Epoch 13/20
14/14 [==============================] - ETA: 0s - loss: 0.6270 - accuracy: 0.682
Epoch 13: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
14/14 [==============================] - 31s 2s/step - loss: 0.6270 - accuracy: 0
Epoch 14/20
14/14 [==============================] - 31s 2s/step - loss: 0.6012 - accuracy: 0
Epoch 15/20
14/14 [==============================] - 32s 2s/step - loss: 0.6051 - accuracy: 0
Epoch 16/20
14/14 [==============================] - ETA: 0s - loss: 0.5989 - accuracy: 0.713
Epoch 16: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
14/14 [==============================] - 31s 2s/step - loss: 0.5989 - accuracy: 0
Epoch 17/20
14/14 [==============================] - 31s 2s/step - loss: 0.5871 - accuracy: 0
Epoch 18/20
14/14 [==============================] - 31s 2s/step - loss: 0.5885 - accuracy: 0
Epoch 19/20
14/14 [==============================] - 31s 2s/step - loss: 0.5860 - accuracy: 0
```

```
Epoch 20/20
14/14 [==============================] - ETA: 0s - loss: 0.5796 - accuracy: 0.724
Epoch 20: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
14/14 [==============================] - 33s 2s/step - loss: 0.5796 - accuracy: 0
```

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()
```

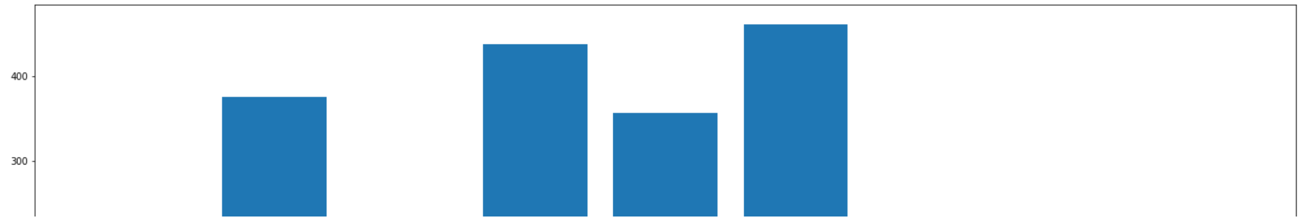


Todo: Write your findings after the model fit, see if there is an evidence of model overfit or underfit. Do you think there is some improvement now as compared to the previous model run?

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.

```
import matplotlib.pyplot as plt

data = dict()

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

for i in data:
  data[i] = len(data[i])

f = plt.figure()
f.set_figwidth(24)
f.set_figheight(8)

plt.bar(range(len(data)), list(data.values()), align='center')
plt.xticks(range(len(data)), list(data.keys()))
plt.show()
```



Todo: Write your findings here:

- Which class has the least number of samples? seborrheic_keratosis

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

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
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whee
Collecting Augmentor
  Downloading Augmentor-0.2.10-py2.py3-none-any.whl (38 kB)
Requirement already satisfied: tqdm>=4.9.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: Pillow>=5.2.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: future>=0.16.0 in /usr/local/lib/python3.7/dist-pa
Installing collected packages: Augmentor
Successfully installed Augmentor-0.2.10
```

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.

```
import Augmentor
for i in class_names:
    print(i)
    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
```

```
actinic keratosis
Initialised with 114 image(s) found.
Output directory set to /gdrive/My Drive/CNN_assignment/Train/actinic keratosis/o
basal cell carcinoma
```

```
Initialised with 376 image(s) found.
Output directory set to /gdrive/My Drive/CNN_assignment/Train/basal cell carcinom
dermatofibroma
Initialised with 95 image(s) found.
Output directory set to /gdrive/My Drive/CNN_assignment/Train/dermatofibroma/outp
melanoma
Initialised with 438 image(s) found.
Output directory set to /gdrive/My Drive/CNN_assignment/Train/melanoma/output.Pro
nevus
Initialised with 357 image(s) found.
Output directory set to /gdrive/My Drive/CNN_assignment/Train/nevus/output.Proces
pigmented benign keratosis
Initialised with 462 image(s) found.
Output directory set to /gdrive/My Drive/CNN_assignment/Train/pigmented benign ke
seborrheic keratosis
Initialised with 77 image(s) found.
Output directory set to /gdrive/My Drive/CNN_assignment/Train/seborrheic keratosi
squamous cell carcinoma
Initialised with 181 image(s) found.
Output directory set to /gdrive/My Drive/CNN_assignment/Train/squamous cell carci
vascular lesion
Initialised with 139 image(s) found.
Output directory set to /gdrive/My Drive/CNN_assignment/Train/vascular lesion/out
```

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.

```
import os
from glob import glob

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


lesion_list_new = [os.path.basename(os.path.dirname(os.path.dirname(y))) for y in glob(
# lesion_list_new


dataframe_dict_new = dict(zip(path_list, lesion_list_new))


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

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

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

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

Todo: Train the model on the data created using Augmentor

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

Todo: Create a training dataset

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  path_to_training_dataset,
  seed=123,
  validation_split = 0.2,
  subset = 'training',
  labels='inferred',
  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',
  labels='inferred',
  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)

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

model.add(layers.experimental.preprocessing.Rescaling(1./255,input_shape=(180,180,3)))


#First Convulation layer
model.add(layers.Conv2D(32,kernel_size=(3,3),activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))
# Adding Normalisation
model.add(BatchNormalization())
# Adding Dropout Layer
model.add(layers.Dropout(0.25))

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

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

#Dropout layer with 50% Fraction of the input units to drop.
model.add(layers.Dropout(0.5))

#Flatten Layer
##Keras.layers.flatten function flattens the multi-dimensional input tensors into a sir
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.
#Softmax is an activation function that scales numbers/logits into probabilities.
model.add(layers.Dense(len(class_names),activation='softmax'))
```

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

```
optimizer = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0

model.compile(loss='categorical_crossentropy',
                  optimizer=optimizer,
                  metrics=['accuracy'])
```

Todo: Train your model

```
epochs = 50

learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
    patience=3,
    verbose=1,
    factor=0.5,
    min_lr=0.00001)

batch_size = 10
history = model.fit(train_ds,
  epochs = epochs, verbose = 1, validation_data=val_ds , callbacks=[learning_rate_reduc
```

```
Epoch 1/50
169/169 [==============================] - 33s 185ms/step - loss: 5.6292 - accu
Epoch 2/50
169/169 [==============================] - 33s 188ms/step - loss: 2.5246 - accu
Epoch 3/50
169/169 [==============================] - 31s 178ms/step - loss: 2.1554 - accu
Epoch 4/50
169/169 [==============================] - 31s 179ms/step - loss: 1.5981 - accu
Epoch 5/50
169/169 [==============================] - 31s 180ms/step - loss: 1.4563 - accu
Epoch 6/50
169/169 [==============================] - 33s 187ms/step - loss: 1.3628 - accu
Epoch 7/50
169/169 [==============================] - 31s 179ms/step - loss: 1.2525 - accu
Epoch 8/50
169/169 [==============================] - 31s 179ms/step - loss: 1.1895 - accu
Epoch 9/50
169/169 [==============================] - 31s 178ms/step - loss: 1.1645 - accu
Epoch 10/50
169/169 [==============================] - ETA: 0s - loss: 1.1810 - accuracy: 0
Epoch 10: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
169/169 [==============================] - 32s 186ms/step - loss: 1.1810 - accu
Epoch 11/50
169/169 [==============================] - 31s 178ms/step - loss: 0.9796 - accu
Epoch 12/50
169/169 [==============================] - 31s 177ms/step - loss: 0.8990 - accu
Epoch 13/50
169/169 [==============================] - 31s 178ms/step - loss: 0.8585 - accu
Epoch 14/50
169/169 [==============================] - 32s 187ms/step - loss: 0.8031 - accu
Epoch 15/50
169/169 [==============================] - 31s 177ms/step - loss: 0.7748 - accu
Epoch 16/50
169/169 [==============================] - 31s 178ms/step - loss: 0.7371 - accu
```

```
Epoch 17/50
169/169 [==============================] - 31s 178ms/step - loss: 0.7305 - accu
Epoch 18/50
169/169 [==============================] - ETA: 0s - loss: 0.6828 - accuracy: 0
Epoch 18: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
169/169 [==============================] - 32s 186ms/step - loss: 0.6828 - accu
Epoch 19/50
169/169 [==============================] - 31s 177ms/step - loss: 0.6127 - accu
Epoch 20/50
169/169 [==============================] - 31s 177ms/step - loss: 0.5805 - accu
Epoch 21/50
169/169 [==============================] - 31s 178ms/step - loss: 0.5403 - accu
Epoch 22/50
169/169 [==============================] - 32s 185ms/step - loss: 0.5106 - accu
Epoch 23/50
169/169 [==============================] - 31s 179ms/step - loss: 0.4960 - accu
Epoch 24/50
169/169 [==============================] - 31s 178ms/step - loss: 0.4895 - accu
Epoch 25/50
169/169 [==============================] - 31s 178ms/step - loss: 0.4801 - accu
Epoch 26/50
169/169 [==============================] - 32s 186ms/step - loss: 0.4715 - accu
Epoch 27/50
```

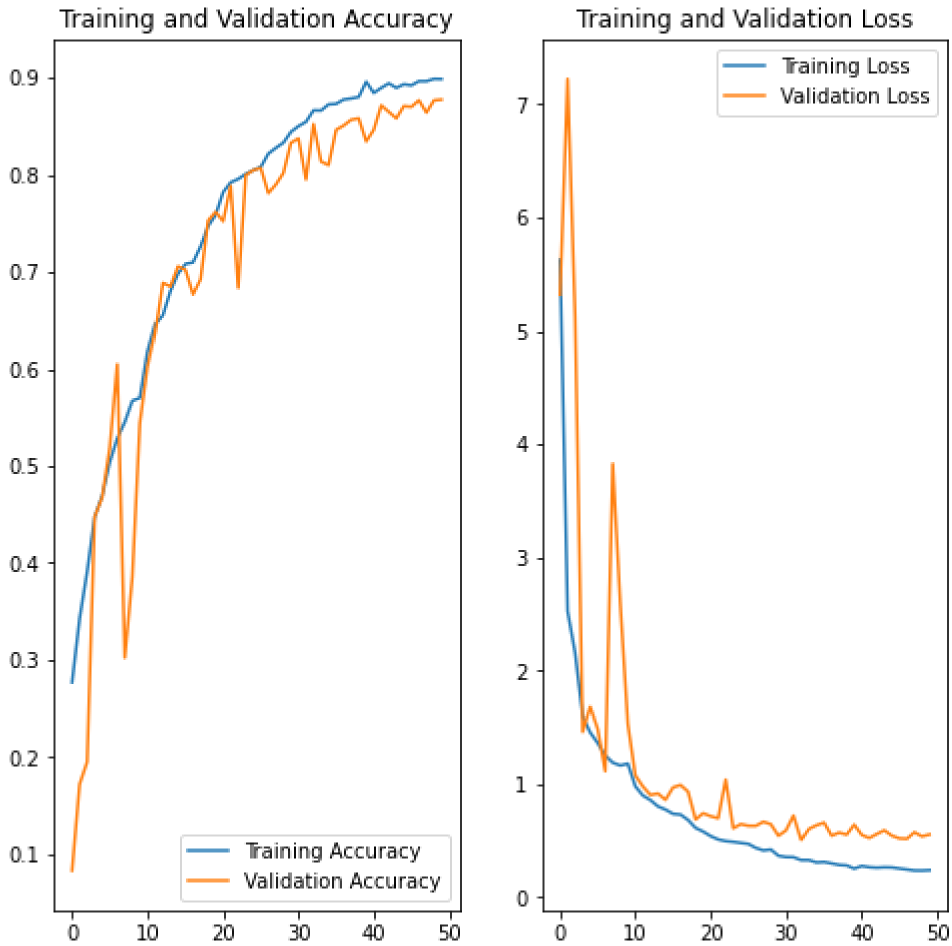
```
loss, accuracy = model.evaluate(train_ds, verbose=1,)
loss_v, accuracy_v = model.evaluate(val_ds, verbose=1)

print("Accuracy: ", accuracy)
print("Validation Accuracy: ",accuracy_v)
print("Loss: ",loss)
print("Validation Loss", loss_v)
```

```
169/169 [==============================] - 25s 139ms/step - loss: 0.1062 - accura
43/43 [==============================] - 6s 114ms/step - loss: 0.5542 - accuracy:
Accuracy:  0.9604970216751099
Validation Accuracy:  0.8775055408477783
Loss:  0.10621771961450577
Validation Loss 0.5542238354682922
```

Todo: Visualize the model results

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

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

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
```

```
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

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



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

The class rebalance helped in reducing overfititng of the data and thus it was given very good results compared to previous 2 models

0s completed at 22:20