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('/content/gdrive')

##Ref:https://towardsdatascience.com/downloading-datasets-into-google-drive-via-google-colab-bcb1b30b0166
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", 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
## Todo: Update the paths of the train and test dataset
data_dir_train = pathlib.Path("/content/gdrive/My Drive/CNN_assignment/Train")
data_dir_test = pathlib.Path('/content/gdrive/My Drive/CNN_assignment/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,
  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_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,
  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', 'nevus', 'pigmented benign keratosis', 'seborrheic keratosis', 'squamous cell carcinoma', 'vascular lesion']
```

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

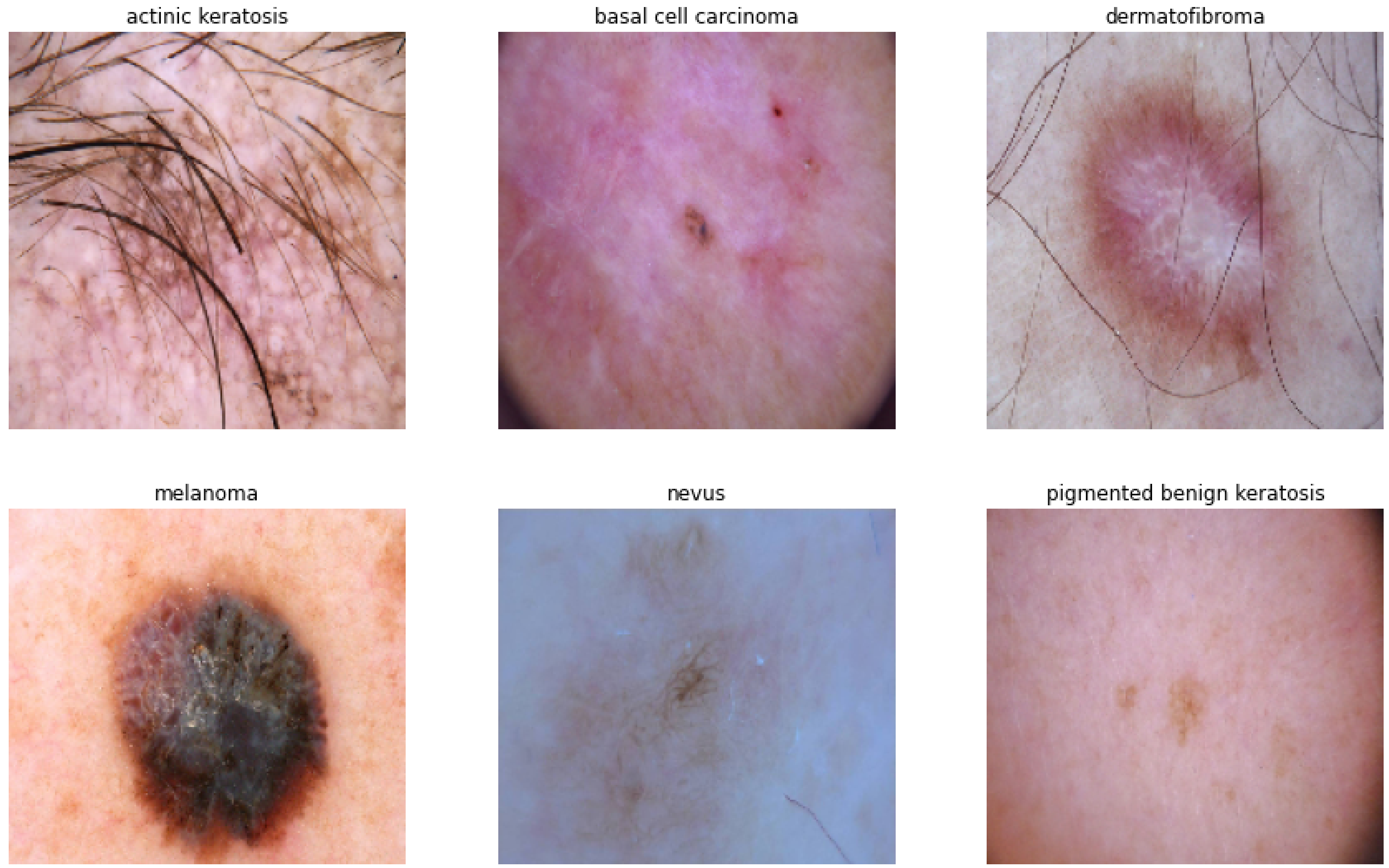
```
Found 118 files belonging to 9 classes.
```

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(str(data_dir_train)+'/'+c)))
```

```
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)))   #Rescaling Layer


#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 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.
#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

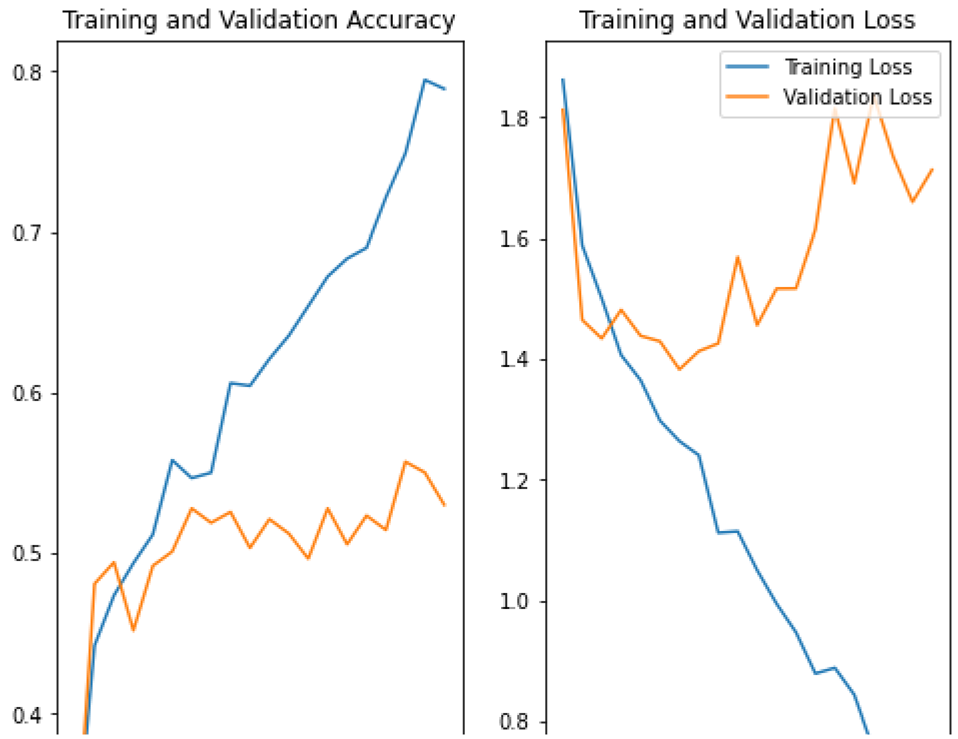
```
epochs = 20
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/20
56/56 [==============================] - 24s 146ms/step - loss: 1.8619 - accuracy: 0.3097 - val_loss: 1.8123 - val_accuracy: 0.3020
Epoch 2/20
56/56 [==============================] - 4s 77ms/step - loss: 1.5882 - accuracy: 0.4425 - val_loss: 1.4648 - val_accuracy: 0.4810
Epoch 3/20
56/56 [==============================] - 4s 77ms/step - loss: 1.5005 - accuracy: 0.4738 - val_loss: 1.4342 - val_accuracy: 0.4944
Epoch 4/20
56/56 [==============================] - 4s 76ms/step - loss: 1.4067 - accuracy: 0.4939 - val_loss: 1.4816 - val_accuracy: 0.4519
Epoch 5/20
56/56 [==============================] - 4s 76ms/step - loss: 1.3652 - accuracy: 0.5117 - val_loss: 1.4386 - val_accuracy: 0.4922
Epoch 6/20
56/56 [==============================] - 4s 77ms/step - loss: 1.2980 - accuracy: 0.5580 - val_loss: 1.4294 - val_accuracy: 0.5011
Epoch 7/20
56/56 [==============================] - 4s 77ms/step - loss: 1.2641 - accuracy: 0.5469 - val_loss: 1.3829 - val_accuracy: 0.5280
Epoch 8/20
56/56 [==============================] - 4s 77ms/step - loss: 1.2406 - accuracy: 0.5502 - val_loss: 1.4131 - val_accuracy: 0.5190
Epoch 9/20
```

```
56/56 [==============================] - 4s 77ms/step - loss: 1.1122 - accuracy: 0.6060 - val_loss: 1.4260 - val_accuracy: 0.5257
Epoch 10/20
56/56 [==============================] - 4s 77ms/step - loss: 1.1146 - accuracy: 0.6044 - val_loss: 1.5692 - val_accuracy: 0.5034
Epoch 11/20
56/56 [==============================] - 4s 77ms/step - loss: 1.0502 - accuracy: 0.6211 - val_loss: 1.4562 - val_accuracy: 0.5213
Epoch 12/20
56/56 [==============================] - 4s 76ms/step - loss: 0.9947 - accuracy: 0.6356 - val_loss: 1.5168 - val_accuracy: 0.5123
Epoch 13/20
56/56 [==============================] - 4s 77ms/step - loss: 0.9472 - accuracy: 0.6540 - val_loss: 1.5171 - val_accuracy: 0.4966
Epoch 14/20
56/56 [==============================] - 4s 76ms/step - loss: 0.8789 - accuracy: 0.6724 - val_loss: 1.6144 - val_accuracy: 0.5280
Epoch 15/20
56/56 [==============================] - 4s 76ms/step - loss: 0.8884 - accuracy: 0.6836 - val_loss: 1.8146 - val_accuracy: 0.5056
Epoch 16/20
56/56 [==============================] - 4s 77ms/step - loss: 0.8438 - accuracy: 0.6903 - val_loss: 1.6907 - val_accuracy: 0.5235
Epoch 17/20
56/56 [==============================] - 4s 77ms/step - loss: 0.7523 - accuracy: 0.7215 - val_loss: 1.8360 - val_accuracy: 0.5145
Epoch 18/20
56/56 [==============================] - 4s 76ms/step - loss: 0.7026 - accuracy: 0.7489 - val_loss: 1.7351 - val_accuracy: 0.5570
Epoch 19/20
56/56 [==============================] - 4s 77ms/step - loss: 0.5516 - accuracy: 0.7946 - val_loss: 1.6596 - val_accuracy: 0.5503
Epoch 20/20
56/56 [==============================] - 4s 77ms/step - loss: 0.5664 - accuracy: 0.7891 - val_loss: 1.7128 - val_accuracy: 0.5302
```

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 overfit, choose an appropriate data augumentation strategy.
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 strategy
```

```
56/56 [==============================] - 2s 31ms/step - loss: 0.3800 - accuracy: 0.8700
14/14 [==============================] - 0s 30ms/step - loss: 1.7128 - accuracy: 0.5302
Accuracy:  0.8699776530265808
Validation Accuracy:  0.5302013158798218
Loss:  0.380046546459198
Validation Loss 1.71280038356781
```

```
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 width)
```

```
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images

image_class = ['nevus','melanoma','basal_cell_caricoma','actinic_keratosis','vasc_lesion','dermatofibroma', 'pigmented_keratosis', 'seborrheic_keratosis', 'squam

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 0x7f05844056d0>
```

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)))   #Rescaling Layer


#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 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.
#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, amsgrad=False)

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_rate_reduction])
```

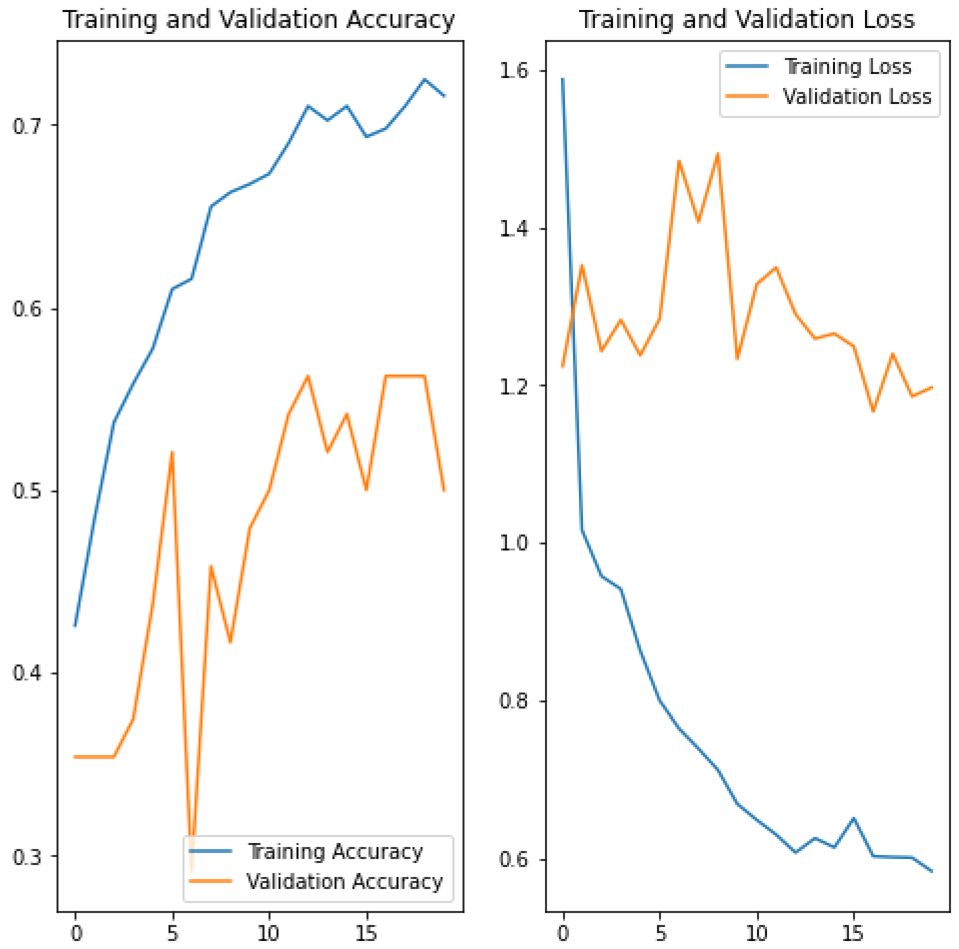
```
Epoch 1/20
14/14 [==============================] - 56s 4s/step - loss: 1.5875 - accuracy: 0.4258 - val_loss: 1.2243 - val_accuracy: 0.3542 - lr: 0.0010
Epoch 2/20
14/14 [==============================] - 35s 3s/step - loss: 1.0164 - accuracy: 0.4843 - val_loss: 1.3523 - val_accuracy: 0.3542 - lr: 0.0010
Epoch 3/20
```

```
14/14 [==============================] - 37s 3s/step - loss: 0.9579 - accuracy: 0.5371 - val_loss: 1.2434 - val_accuracy: 0.3542 - lr: 0.0010
Epoch 4/20
14/14 [==============================] - 37s 3s/step - loss: 0.9414 - accuracy: 0.5584 - val_loss: 1.2830 - val_accuracy: 0.3750 - lr: 0.0010
Epoch 5/20
14/14 [==============================] - 37s 3s/step - loss: 0.8633 - accuracy: 0.5775 - val_loss: 1.2381 - val_accuracy: 0.4375 - lr: 0.0010
Epoch 6/20
14/14 [==============================] - 36s 3s/step - loss: 0.8000 - accuracy: 0.6101 - val_loss: 1.2842 - val_accuracy: 0.5208 - lr: 0.0010
Epoch 7/20
14/14 [==============================] - 37s 3s/step - loss: 0.7650 - accuracy: 0.6157 - val_loss: 1.4841 - val_accuracy: 0.2917 - lr: 0.0010
Epoch 8/20
14/14 [==============================] - 37s 3s/step - loss: 0.7396 - accuracy: 0.6551 - val_loss: 1.4066 - val_accuracy: 0.4583 - lr: 0.0010
Epoch 9/20
14/14 [==============================] - ETA: 0s - loss: 0.7126 - accuracy: 0.6629
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
14/14 [==============================] - 36s 3s/step - loss: 0.7126 - accuracy: 0.6629 - val_loss: 1.4932 - val_accuracy: 0.4167 - lr: 0.0010
Epoch 10/20
14/14 [==============================] - 36s 3s/step - loss: 0.6694 - accuracy: 0.6674 - val_loss: 1.2335 - val_accuracy: 0.4792 - lr: 5.0000e-04
Epoch 11/20
14/14 [==============================] - 36s 3s/step - loss: 0.6492 - accuracy: 0.6730 - val_loss: 1.3282 - val_accuracy: 0.5000 - lr: 5.0000e-04
Epoch 12/20
14/14 [==============================] - 36s 3s/step - loss: 0.6305 - accuracy: 0.6899 - val_loss: 1.3495 - val_accuracy: 0.5417 - lr: 5.0000e-04
Epoch 13/20
14/14 [==============================] - 36s 3s/step - loss: 0.6077 - accuracy: 0.7101 - val_loss: 1.2901 - val_accuracy: 0.5625 - lr: 5.0000e-04
Epoch 14/20
14/14 [==============================] - 36s 3s/step - loss: 0.6259 - accuracy: 0.7022 - val_loss: 1.2592 - val_accuracy: 0.5208 - lr: 5.0000e-04
Epoch 15/20
14/14 [==============================] - 36s 3s/step - loss: 0.6143 - accuracy: 0.7101 - val_loss: 1.2656 - val_accuracy: 0.5417 - lr: 5.0000e-04
Epoch 16/20
14/14 [==============================] - ETA: 0s - loss: 0.6514 - accuracy: 0.6933
Epoch 16: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
14/14 [==============================] - 35s 3s/step - loss: 0.6514 - accuracy: 0.6933 - val_loss: 1.2493 - val_accuracy: 0.5000 - lr: 5.0000e-04
Epoch 17/20
14/14 [==============================] - 36s 3s/step - loss: 0.6031 - accuracy: 0.6978 - val_loss: 1.1667 - val_accuracy: 0.5625 - lr: 2.5000e-04
Epoch 18/20
14/14 [==============================] - 35s 3s/step - loss: 0.6019 - accuracy: 0.7101 - val_loss: 1.2400 - val_accuracy: 0.5625 - lr: 2.5000e-04
Epoch 19/20
14/14 [==============================] - ETA: 0s - loss: 0.6013 - accuracy: 0.7247
Epoch 19: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
14/14 [==============================] - 36s 3s/step - loss: 0.6013 - accuracy: 0.7247 - val_loss: 1.1859 - val_accuracy: 0.5625 - lr: 2.5000e-04
Epoch 20/20
14/14 [==============================] - 36s 3s/step - loss: 0.5843 - accuracy: 0.7157 - val_loss: 1.1971 - val_accuracy: 0.5000 - lr: 1.2500e-04
```

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()
```



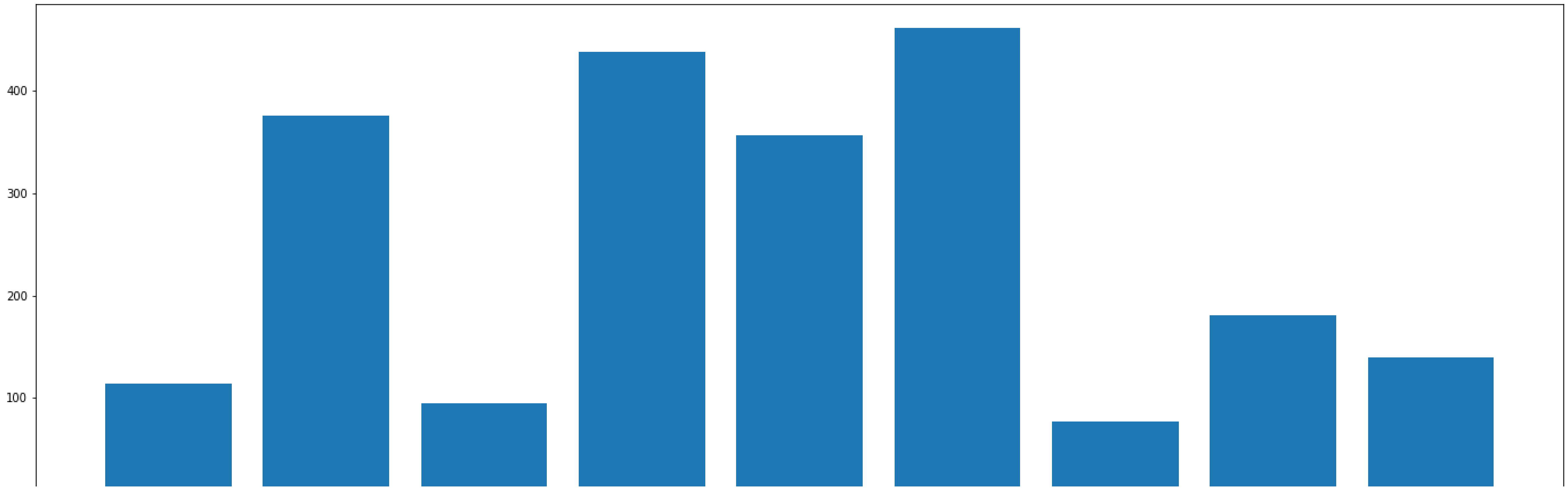
Todo: Write your findings after the model fit, see if there is an evidence of model overfit or underfit. Do you think there is some improvement now as compared to the previous model run?

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_train)+'/'+c)))

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
```

```
    Collecting Augmentor
      Downloading Augmentor-0.2.9-py2.py3-none-any.whl (38 kB)
```

```
Requirement already satisfied: future>=0.16.0 in /usr/local/lib/python3.7/dist-packages (from Augmentor) (0.16.0)
Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.7/dist-packages (from Augmentor) (1.21.5)
Requirement already satisfied: tqdm>=4.9.0 in /usr/local/lib/python3.7/dist-packages (from Augmentor) (4.62.3)
Requirement already satisfied: Pillow>=5.2.0 in /usr/local/lib/python3.7/dist-packages (from Augmentor) (7.1.2)
Installing collected packages: Augmentor
Successfully installed Augmentor-0.2.9
```

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/CNN_assignment/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/gdrive/My Drive/CNN_assignment/Train/actinic keratosis/output.Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=600x450 at 0x7F0509407890>:
Initialised with 376 image(s) found.
Output directory set to /content/gdrive/My Drive/CNN_assignment/Train/basal cell carcinoma/output.Processing <PIL.Image.Image image mode=RGB size=600x450 at 0x7F050942EE10>: 100%|█████████
Initialised with 95 image(s) found.
Output directory set to /content/gdrive/My Drive/CNN_assignment/Train/dermatofibroma/output.Processing <PIL.Image.Image image mode=RGB size=600x450 at 0x7F050A074550>: 100%|██████████| 500
Initialised with 438 image(s) found.
Output directory set to /content/gdrive/My Drive/CNN_assignment/Train/melanoma/output.Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=2198x1603 at 0x7F050A015110>: 100%|█
Initialised with 357 image(s) found.
Output directory set to /content/gdrive/My Drive/CNN_assignment/Train/nevus/output.Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=600x450 at 0x7F050A074B10>: 100%|██████
Initialised with 462 image(s) found.
Output directory set to /content/gdrive/My Drive/CNN_assignment/Train/pigmented benign keratosis/output.Processing <PIL.Image.Image image mode=RGB size=600x450 at 0x7F05093B9B10>: 100%|███
Initialised with 77 image(s) found.
Output directory set to /content/gdrive/My Drive/CNN_assignment/Train/seborrheic keratosis/output.Processing <PIL.Image.Image image mode=RGB size=1024x768 at 0x7F0586505F10>: 100%|████████
Initialised with 181 image(s) found.
Output directory set to /content/gdrive/My Drive/CNN_assignment/Train/squamous cell carcinoma/output.Processing <PIL.Image.Image image mode=RGB size=600x450 at 0x7F050A629710>: 100%|██████
Initialised with 139 image(s) found.
Output directory set to /content/gdrive/My Drive/CNN_assignment/Train/vascular lesion/output.Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=600x450 at 0x7F050A62DAD0>: 1
```

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(os.path.join(data_dir_train, '*','output', '*.jpg'))]
# 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()
```

```
    actinic keratosis             500
    nevus                         500
    dermatofibroma                500
    squamous cell carcinoma       500
    vascular lesion               500
    seborrheic keratosis          500
    pigmented benign keratosis    500
    basal cell carcinoma          500
    melanoma                      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

```
data_dir_train="/content/gdrive/My Drive/CNN_assignment/Train"
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir_train,
  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)))   #Rescaling Layer


#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 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.
#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, amsgrad=False)

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_reduction])
```

```
169/169 [==============================] - 40s 232ms/step - loss: 0.3861 - accuracy: 0.8431 - val_loss: 0.7190 - val_accuracy: 0.7587 - lr: 5.0000e-04
Epoch 27/50
169/169 [==============================] - 41s 233ms/step - loss: 0.3747 - accuracy: 0.8498 - val_loss: 0.6217 - val_accuracy: 0.7795 - lr: 5.0000e-04
Epoch 28/50
168/169 [============================>.] - ETA: 0s - loss: 0.3701 - accuracy: 0.8501
Epoch 28: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
169/169 [==============================] - 42s 242ms/step - loss: 0.3707 - accuracy: 0.8500 - val_loss: 0.5614 - val_accuracy: 0.8099 - lr: 5.0000e-04
Epoch 29/50
169/169 [==============================] - 42s 243ms/step - loss: 0.3255 - accuracy: 0.8681 - val_loss: 0.5255 - val_accuracy: 0.8293 - lr: 2.5000e-04
Epoch 30/50
169/169 [==============================] - 42s 243ms/step - loss: 0.3150 - accuracy: 0.8668 - val_loss: 0.6231 - val_accuracy: 0.7944 - lr: 2.5000e-04
Epoch 31/50
169/169 [==============================] - 42s 242ms/step - loss: 0.3016 - accuracy: 0.8759 - val_loss: 0.4846 - val_accuracy: 0.8537 - lr: 2.5000e-04
Epoch 32/50
169/169 [==============================] - 42s 238ms/step - loss: 0.2858 - accuracy: 0.8858 - val_loss: 0.5243 - val_accuracy: 0.8359 - lr: 2.5000e-04
Epoch 33/50
169/169 [==============================] - 42s 241ms/step - loss: 0.2957 - accuracy: 0.8809 - val_loss: 0.5602 - val_accuracy: 0.8322 - lr: 2.5000e-04
Epoch 34/50
168/169 [============================>.] - ETA: 0s - loss: 0.2830 - accuracy: 0.8815
```

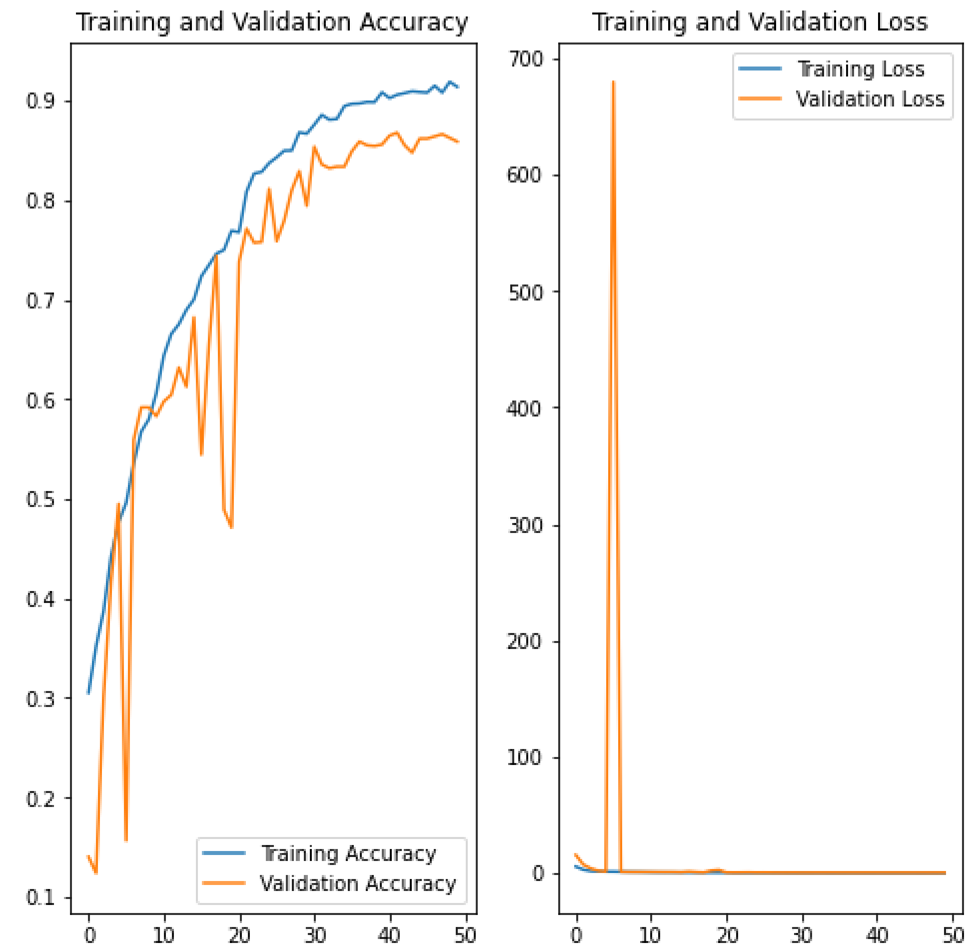
```
168/169 [============================>.] - ETA: 0s - loss: 0.2830 - accuracy: 0.8815
Epoch 34: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
169/169 [==============================] - 42s 241ms/step - loss: 0.2828 - accuracy: 0.8817 - val_loss: 0.5334 - val_accuracy: 0.8337 - lr: 2.5000e-04
Epoch 35/50
169/169 [==============================] - 42s 238ms/step - loss: 0.2492 - accuracy: 0.8947 - val_loss: 0.5808 - val_accuracy: 0.8337 - lr: 1.2500e-04
Epoch 36/50
169/169 [==============================] - 42s 242ms/step - loss: 0.2414 - accuracy: 0.8969 - val_loss: 0.5153 - val_accuracy: 0.8493 - lr: 1.2500e-04
Epoch 37/50
169/169 [==============================] - 42s 242ms/step - loss: 0.2598 - accuracy: 0.8973 - val_loss: 0.5298 - val_accuracy: 0.8589 - lr: 1.2500e-04
Epoch 38/50
169/169 [==============================] - 41s 238ms/step - loss: 0.2476 - accuracy: 0.8987 - val_loss: 0.5169 - val_accuracy: 0.8552 - lr: 1.2500e-04
Epoch 39/50
169/169 [==============================] - 43s 246ms/step - loss: 0.2385 - accuracy: 0.8986 - val_loss: 0.5270 - val_accuracy: 0.8545 - lr: 1.2500e-04
Epoch 40/50
168/169 [============================>.] - ETA: 0s - loss: 0.2245 - accuracy: 0.9083
Epoch 40: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
169/169 [==============================] - 42s 241ms/step - loss: 0.2242 - accuracy: 0.9084 - val_loss: 0.5475 - val_accuracy: 0.8560 - lr: 1.2500e-04
Epoch 41/50
169/169 [==============================] - 42s 244ms/step - loss: 0.2292 - accuracy: 0.9026 - val_loss: 0.5216 - val_accuracy: 0.8649 - lr: 6.2500e-05
Epoch 42/50
169/169 [==============================] - 41s 236ms/step - loss: 0.2216 - accuracy: 0.9060 - val_loss: 0.5197 - val_accuracy: 0.8679 - lr: 6.2500e-05
Epoch 43/50
169/169 [==============================] - 42s 241ms/step - loss: 0.2162 - accuracy: 0.9076 - val_loss: 0.5412 - val_accuracy: 0.8552 - lr: 6.2500e-05
Epoch 44/50
169/169 [==============================] - 42s 241ms/step - loss: 0.2225 - accuracy: 0.9095 - val_loss: 0.5411 - val_accuracy: 0.8478 - lr: 6.2500e-05
Epoch 45/50
168/169 [============================>.] - ETA: 0s - loss: 0.2258 - accuracy: 0.9085
Epoch 45: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
169/169 [==============================] - 43s 246ms/step - loss: 0.2255 - accuracy: 0.9088 - val_loss: 0.5235 - val_accuracy: 0.8619 - lr: 6.2500e-05
Epoch 46/50
169/169 [==============================] - 42s 241ms/step - loss: 0.2260 - accuracy: 0.9082 - val_loss: 0.5241 - val_accuracy: 0.8619 - lr: 3.1250e-05
Epoch 47/50
169/169 [==============================] - 42s 241ms/step - loss: 0.2001 - accuracy: 0.9151 - val_loss: 0.5269 - val_accuracy: 0.8641 - lr: 3.1250e-05
Epoch 48/50
169/169 [==============================] - ETA: 0s - loss: 0.2103 - accuracy: 0.9082
Epoch 48: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
169/169 [==============================] - 43s 246ms/step - loss: 0.2103 - accuracy: 0.9082 - val_loss: 0.5050 - val_accuracy: 0.8664 - lr: 3.1250e-05
Epoch 49/50
169/169 [==============================] - 43s 244ms/step - loss: 0.2013 - accuracy: 0.9190 - val_loss: 0.5302 - val_accuracy: 0.8627 - lr: 1.5625e-05
Epoch 50/50
169/169 [==============================] - 41s 238ms/step - loss: 0.2054 - accuracy: 0.9139 - val_loss: 0.5462 - val_accuracy: 0.8589 - lr: 1.5625e-05
```

```
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 [==============================] - 31s 175ms/step - loss: 0.0915 - accuracy: 0.9616
43/43 [==============================] - 8s 150ms/step - loss: 0.5462 - accuracy: 0.8589
Accuracy:  0.9616097807884216
Validation Accuracy:  0.8589457869529724
Loss:  0.09146161377429962
Validation Loss 0.5461551547050476
```

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 revious 2 models

0s completed at 1:23 AM