Importing Skin Cancer 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
import warnings
warnings.filterwarnings('ignore')
```

Data Reading/Data Understanding

```
from google.colab import drive
drive.mount('/content/gdrive')
```

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

```
!unzip '/content/gdrive/MyDrive/Upgrad/CNN_assignment.zip' -d '/content/gdrive/MyDrive/Upgrad/Dataset/'
```

```
  inflating: /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collaboration/T
  inflating: /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collaboration/T
  inflating: /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collaboration/T
  inflating: /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collaboration/T
  inflating: /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collaboration/T
  inflating: /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collaboration/T
  inflating: /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collaboration/T
  inflating: /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collaboration/T
  inflating: /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collaboration/T
  inflating: /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collaboration/T
```

```
# 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/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Col
data_dir_test = pathlib.Path('/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Coll
```

```
!ls
```

```
gdrive  sample_data
```

Dataset Creation→ Create train & validation dataset from the train directory with a batch size of 32. Also, make sure you resize your images to 180*180

```
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
```

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

Creation of Training dataset

```
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),
  batch_size=batch_size)
```

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

Creation of validation dataset

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

```
Found 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)
```

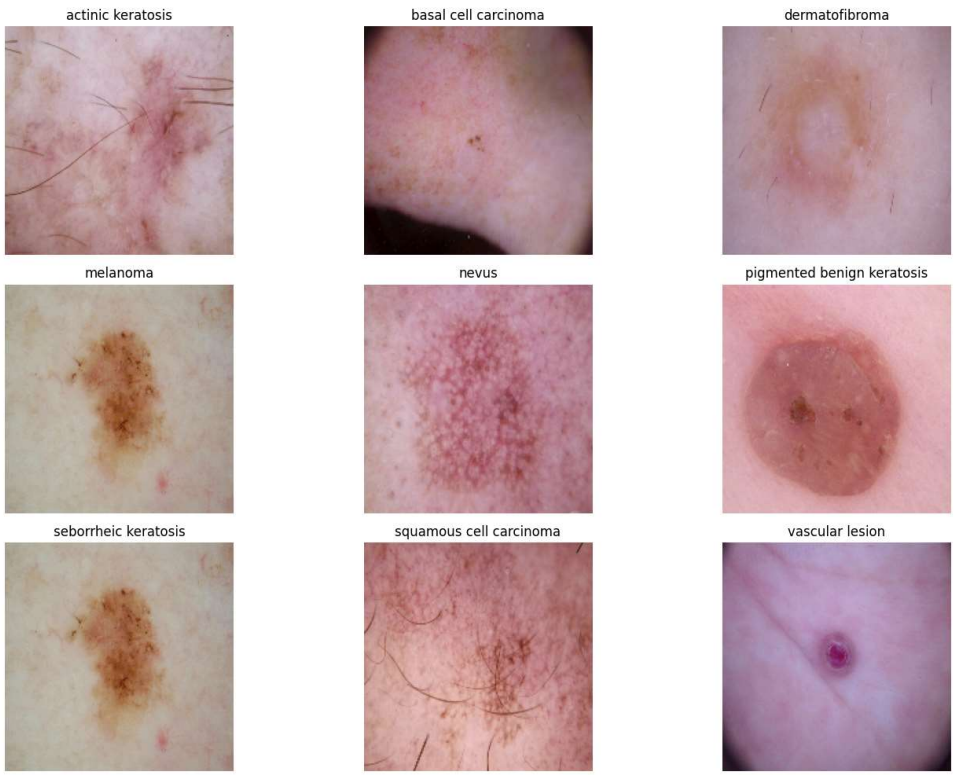
```
['actinic keratosis', 'basal cell carcinoma', 'dermatofibroma', 'melanoma', 'nevus', 'pigmented benign keratosis', 's
```

```
print(type(train_ds))
```

```
<class 'tensorflow.python.data.ops.prefetch_op._PrefetchDataset'>
```

Dataset visualisation → Create a code to visualize one instance of all the nine classes present in the dataset

```
import matplotlib.pyplot as plt

# Create an empty dictionary to store one instance of each class
class_samples = {class_name: None for class_name in class_names}

# Iterate through the dataset to find one instance of each class
for images, labels in train_ds:
    for image, label in zip(images, labels):
        class_name = class_names[label.numpy()]
        if class_samples[class_name] is None:
            class_samples[class_name] = image.numpy()
            break

# Visualize one instance of each class
plt.figure(figsize=(15, 10))
for i, (class_name, image) in enumerate(class_samples.items()):
    plt.subplot(3, 3, i+1)
    plt.imshow(image.astype("uint8"))
    plt.title(class_name)
    plt.axis("off")
plt.tight_layout()
plt.show()
```




```
#print(type(train_ds))
#print(len(train_ds)


#overlaps data preprocessing and model execution while training., Speed up 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)
```

CNN MODEL

```
### Your code goes here
num_classes = 9

#A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one outp
model = Sequential([layers.experimental.preprocessing.Rescaling(1./255, input_shape=(img_height, img_width, 3)),

  #2D convolution layer (e.g. spatial convolution over images).
  layers.Conv2D(16, 3, padding='same', activation='relu'),

  #We slide over the feature map and extract tiles of a specified size.
  #Downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window
  layers.MaxPooling2D(),

  #We slide over the feature map and extract tiles of a specified size.
  layers.Conv2D(32, 3, padding='same', activation='relu'),

  layers.MaxPooling2D(),

  layers.Conv2D(64, 3, padding='same', activation='relu'),

  #We slide over the feature map and extract tiles of a specified size.
  #Advantages of downsampling - Decreased size of input for upcoming layers, Works against overfitting
  layers.MaxPooling2D(),

  #Flattening - Convert into 1D feature vector.  Flattens all its structure to create a single long feature vector
  ##Flattens the input. Does not affect the batch size.
  layers.Flatten(),

  #fully connected layer
  #A hidden layer in which each node is connected to every node in the subsequent hidden layer.
  #A fully connected layer is also known as a dense layer.

  layers.Dense(128, activation='relu'),

  #Dense is the only actual network layer in that model. A Dense layer feeds all outputs from the previous layer to all i
  #It's the most basic layer in neural networks. A Dense(10) has ten neurons. A Dense(512) has 512 neurons.
  #Dense implements the operation: output = activation(dot(input, kernel)
  #Dense Layer - A dense layer represents a matrix vector multiplication.  each input node is connected to each output no
  layers.Dense(num_classes)
  #Dense Layer - A dense layer represents a matrix vector multiplication.  each input node is connected to each output no
])



# 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, 16)      448

 max_pooling2d (MaxPooling2  (None, 90, 90, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_1 (MaxPoolin  (None, 45, 45, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 22, 22, 64)        0
 g2D)

 flatten (Flatten)           (None, 30976)             0

 dense (Dense)               (None, 128)               3965056

 dense_1 (Dense)             (None, 9)                 1161

=================================================================
Total params: 3989801 (15.22 MB)
Trainable params: 3989801 (15.22 MB)
```

```
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compile the model

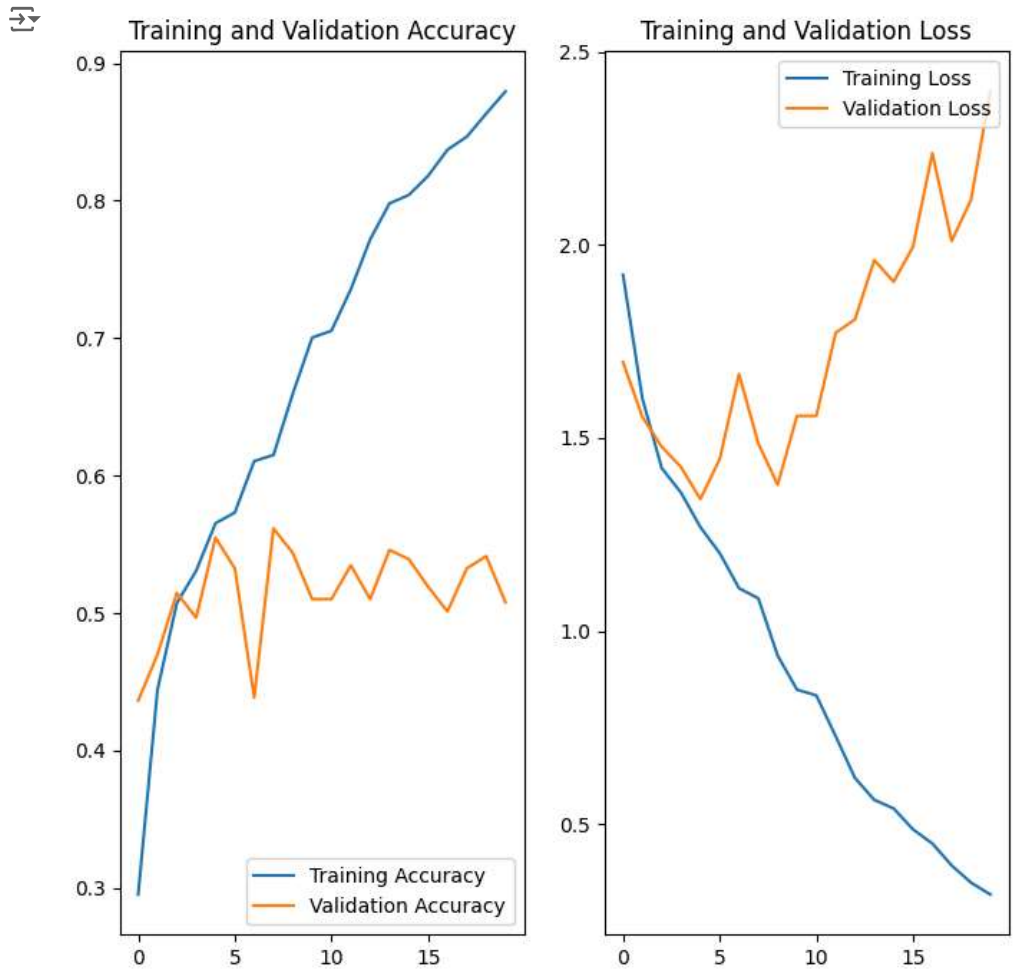
```
### Todo, choose an appropirate optimiser and loss function
#RMSprop. RMSprop is a very effective, but currently unpublished adaptive learning rate method
#Adam. Adam is a recently proposed update that looks a bit like RMSProp with momentum. The (simplified) update looks as f
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Train the model for ~20 epochs

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

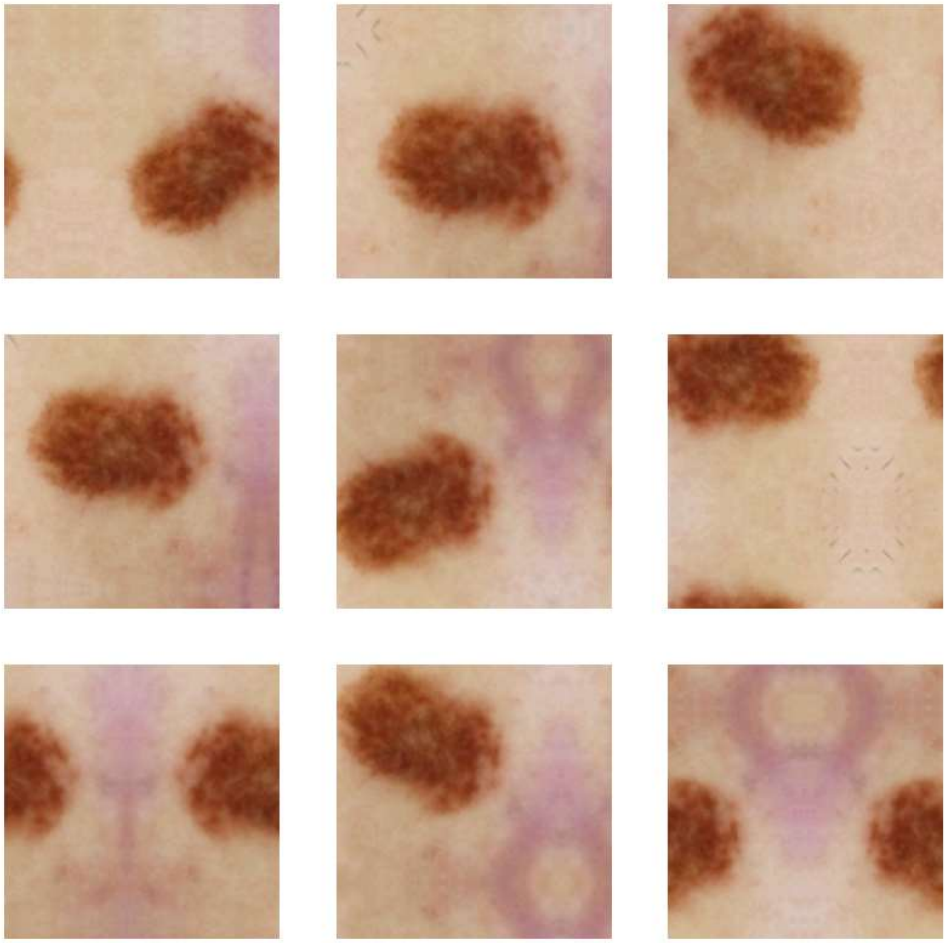
```
Epoch 1/20
56/56 [==============================] - 87s 1s/step - loss: 1.9225 - accuracy: 0.2952 - val_loss: 1.6963 - val_accur
Epoch 2/20
56/56 [==============================] - 66s 1s/step - loss: 1.6025 - accuracy: 0.4442 - val_loss: 1.5545 - val_accur
Epoch 3/20
56/56 [==============================] - 68s 1s/step - loss: 1.4222 - accuracy: 0.5073 - val_loss: 1.4775 - val_accur
Epoch 4/20
56/56 [==============================] - 66s 1s/step - loss: 1.3585 - accuracy: 0.5307 - val_loss: 1.4247 - val_accur
Epoch 5/20
56/56 [==============================] - 71s 1s/step - loss: 1.2694 - accuracy: 0.5653 - val_loss: 1.3418 - val_accur
Epoch 6/20
56/56 [==============================] - 68s 1s/step - loss: 1.2021 - accuracy: 0.5731 - val_loss: 1.4449 - val_accur
Epoch 7/20
56/56 [==============================] - 67s 1s/step - loss: 1.1111 - accuracy: 0.6105 - val_loss: 1.6655 - val_accur
Epoch 8/20
56/56 [==============================] - 65s 1s/step - loss: 1.0848 - accuracy: 0.6150 - val_loss: 1.4858 - val_accur
Epoch 9/20
56/56 [==============================] - 66s 1s/step - loss: 0.9365 - accuracy: 0.6596 - val_loss: 1.3789 - val_accur
Epoch 10/20
56/56 [==============================] - 68s 1s/step - loss: 0.8485 - accuracy: 0.7003 - val_loss: 1.5571 - val_accur
Epoch 11/20
56/56 [==============================] - 66s 1s/step - loss: 0.8335 - accuracy: 0.7054 - val_loss: 1.5571 - val_accur
Epoch 12/20
56/56 [==============================] - 64s 1s/step - loss: 0.7279 - accuracy: 0.7355 - val_loss: 1.7722 - val_accur
Epoch 13/20
56/56 [==============================] - 65s 1s/step - loss: 0.6196 - accuracy: 0.7718 - val_loss: 1.8070 - val_accur
Epoch 14/20
56/56 [==============================] - 64s 1s/step - loss: 0.5630 - accuracy: 0.7980 - val_loss: 1.9605 - val_accur
Epoch 15/20
56/56 [==============================] - 66s 1s/step - loss: 0.5408 - accuracy: 0.8041 - val_loss: 1.9043 - val_accur
Epoch 16/20
56/56 [==============================] - 63s 1s/step - loss: 0.4867 - accuracy: 0.8181 - val_loss: 1.9951 - val_accur
Epoch 17/20
56/56 [==============================] - 70s 1s/step - loss: 0.4502 - accuracy: 0.8371 - val_loss: 2.2374 - val_accur
Epoch 18/20
56/56 [==============================] - 65s 1s/step - loss: 0.3934 - accuracy: 0.8465 - val_loss: 2.0101 - val_accur
Epoch 19/20
56/56 [==============================] - 65s 1s/step - loss: 0.3490 - accuracy: 0.8633 - val_loss: 2.1164 - val_accur
Epoch 20/20
56/56 [==============================] - 65s 1s/step - loss: 0.3182 - accuracy: 0.8795 - val_loss: 2.3971 - val_accur
```

```
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()
```



High Training Accuracy and low validation accuracy. The valdation loss is very high. The model seems overfitting

```
### Your code goes here
num_classes = 9

model = Sequential([
  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  #We slide over the feature map and extract tiles of a specified size.
  layers.MaxPooling2D(),
  layers.Conv2D(128, 3, padding='same', activation='relu'),
  #We slide over the feature map and extract tiles of a specified size.
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  #We slide over the feature map and extract tiles of a specified size.
  layers.MaxPooling2D(),
  #Advantages of downsampling - Decreased size of input for upcoming layers, Works against overfitting
  layers.Flatten(),
  #Flattening - Convert into 1D feature vector.  Flattens all its structure to create a single long feature vector
  layers.Dense(128, activation='relu'),
  #Dense Layer - A dense layer represents a matrix vector multiplication.  each input node is connected to each output no
  layers.Dense(num_classes)
  #Dense Layer - A dense layer represents a matrix vector multiplication.  each input node is connected to each output no
])


# Todo, after you have analysed the model fit history for presence of underfit or overfit, choose an appropriate data aug
data_augmentation = keras.Sequential(
  [
    layers.experimental.preprocessing.RandomFlip("horizontal",
                                                 input_shape=(img_height,
                                                              img_width,
                                                              3)),
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1),
    layers.experimental.preprocessing.RandomTranslation(1,.5,fill_mode="reflect",interpolation="bilinear",seed=None,fill_
    layers.experimental.preprocessing.RandomCrop(img_height,img_width),
  ]
)


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



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

model = Sequential([
  data_augmentation,
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(128, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(256, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])


## Your code goes here
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])


## Your code goes here, note: train your model for 20 epochs
epochs = 20
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/20
56/56 [==============================] - 466s 8s/step - loss: 2.1752 - accuracy: 0.2109 - val_loss: 1.9587 - val_accu
Epoch 2/20
56/56 [==============================] - 456s 8s/step - loss: 1.8943 - accuracy: 0.3019 - val_loss: 1.7770 - val_accu
Epoch 3/20
56/56 [==============================] - 460s 8s/step - loss: 1.7287 - accuracy: 0.3482 - val_loss: 1.7122 - val_accu
Epoch 4/20
56/56 [==============================] - 458s 8s/step - loss: 1.7428 - accuracy: 0.3571 - val_loss: 1.6455 - val_accu
Epoch 5/20
56/56 [==============================] - 452s 8s/step - loss: 1.6607 - accuracy: 0.3733 - val_loss: 1.6473 - val_accu
Epoch 6/20
56/56 [==============================] - 452s 8s/step - loss: 1.5867 - accuracy: 0.4062 - val_loss: 1.6351 - val_accu
Epoch 7/20
56/56 [==============================] - 453s 8s/step - loss: 1.5698 - accuracy: 0.4213 - val_loss: 1.5971 - val_accu
Epoch 8/20
56/56 [==============================] - 455s 8s/step - loss: 1.5903 - accuracy: 0.4235 - val_loss: 1.6650 - val_accu
Epoch 9/20
56/56 [==============================] - 458s 8s/step - loss: 1.6184 - accuracy: 0.3968 - val_loss: 1.5952 - val_accu
Epoch 10/20
56/56 [==============================] - 469s 8s/step - loss: 1.6245 - accuracy: 0.3996 - val_loss: 1.6156 - val_accu
Epoch 11/20
56/56 [==============================] - 455s 8s/step - loss: 1.5633 - accuracy: 0.4196 - val_loss: 1.5938 - val_accu
Epoch 12/20
56/56 [==============================] - 441s 8s/step - loss: 1.5420 - accuracy: 0.4431 - val_loss: 1.6780 - val_accu
Epoch 13/20
56/56 [==============================] - 449s 8s/step - loss: 1.5185 - accuracy: 0.4526 - val_loss: 1.5462 - val_accu
Epoch 14/20
56/56 [==============================] - 452s 8s/step - loss: 1.4849 - accuracy: 0.4548 - val_loss: 1.6588 - val_accu
Epoch 15/20
56/56 [==============================] - 449s 8s/step - loss: 1.4982 - accuracy: 0.4548 - val_loss: 1.6894 - val_accu
Epoch 16/20
56/56 [==============================] - 453s 8s/step - loss: 1.4568 - accuracy: 0.4704 - val_loss: 1.6209 - val_accu
Epoch 17/20
56/56 [==============================] - 451s 8s/step - loss: 1.4821 - accuracy: 0.4738 - val_loss: 1.5560 - val_accu
Epoch 18/20
56/56 [==============================] - 447s 8s/step - loss: 1.4798 - accuracy: 0.4671 - val_loss: 1.6146 - val_accu
Epoch 19/20
56/56 [==============================] - 456s 8s/step - loss: 1.4846 - accuracy: 0.4576 - val_loss: 1.4798 - val_accu
Epoch 20/20
56/56 [==============================] - 451s 8s/step - loss: 1.5157 - accuracy: 0.4515 - val_loss: 1.5822 - val_accu
```

```
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()
```

Training and validation accuracy is almost same. The Training and validation loss is almost same. The issue of Overfitting in the earlier model is attended in this model

```
# For convenience, let us set up the path for the training and validation sets
train_dir = os.path.join('/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collabor
val_dir = os.path.join('/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collaborat
```

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Setting batch size and image size
batch_size = 100
IMG_SHAPE = 224

# Create training images generator
#Generate batches of tensor image data with real-time data augmentation.
#https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator
image_gen_train = ImageDataGenerator(
                    rescale=1./255,
                    rotation_range=45,
                    width_shift_range=.15,
                    height_shift_range=.15,
                    horizontal_flip=True,
                    zoom_range=0.5
                    )
#https://keras.io/api/preprocessing/image/
#Then calling image_dataset_from_directory(main_directory, labels='inferred') will return a tf.data.Dataset that yields b
train_data_gen = image_gen_train.flow_from_directory(
                                              batch_size=batch_size,
                                              directory=train_dir,
                                              shuffle=True,
                                              target_size=(IMG_SHAPE,IMG_SHAPE),
                                              class_mode='sparse'
                                              )

# Create validation images generator
image_gen_val = ImageDataGenerator(rescale=1./255)
val_data_gen = image_gen_val.flow_from_directory(batch_size=batch_size,
                                                 directory=val_dir,
                                                 target_size=(IMG_SHAPE, IMG_SHAPE),
                                                 class_mode='sparse')
```

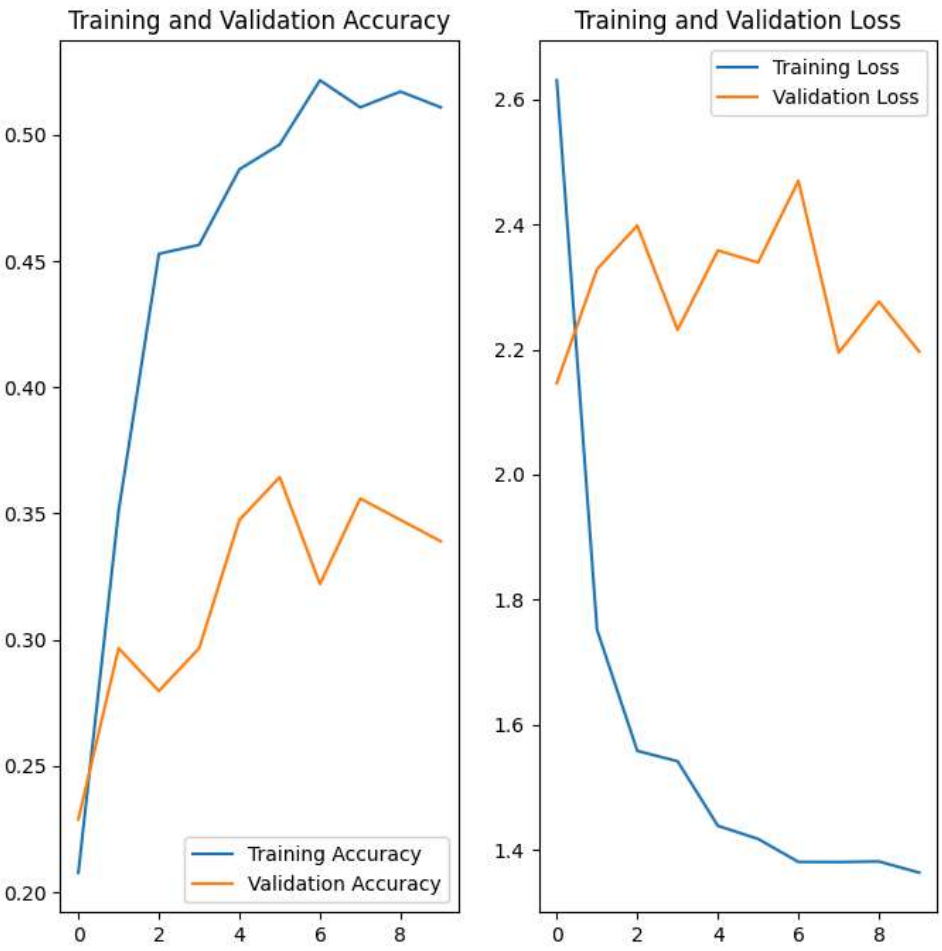
```
Found 2239 images belonging to 9 classes.
Found 118 images belonging to 9 classes.
```

```
#Create a CNN model
#Experiment #1
#A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output
import numpy as np
import glob
import shutil
import matplotlib.pyplot as plt

# Import layers explicitly to keep our code compact
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D


model = Sequential()

#2D convolution layer (e.g. spatial convolution over images).
model.add(Conv2D(16, 3, padding='same', activation='relu', input_shape=(IMG_SHAPE,IMG_SHAPE, 3)))
#Downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window (of
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, 3, padding='same', activation='relu'))

#Downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window (of
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, 3, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

#Flattens the input. Does not affect the batch size.
model.add(Flatten())

#https://keras.io/api/layers/regularization_layers/dropout/
#The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps p
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
```

```
model.add(Dropout(0.2))

#Just your regular densely-connected NN layer.
#Dense is the only actual network layer in that model. A Dense layer feeds all outputs from the previous layer to all its r
#It's the most basic layer in neural networks. A Dense(10) has ten neurons. A Dense(512) has 512 neurons.
#Dense implements the operation: output = activation(dot(input, kernel)
model.add(Dense(9))

# Compile the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])




# Train the model
epochs = 20

history = model.fit(
  train_data_gen,
  validation_data=val_data_gen,
  epochs=10
)
```

```
Epoch 1/10
23/23 [==============================] - 231s 10s/step - loss: 2.6308 - accuracy: 0.2077 - val_loss: 2.1464 - val_acc
Epoch 2/10
23/23 [==============================] - 205s 9s/step - loss: 1.7519 - accuracy: 0.3510 - val_loss: 2.3286 - val_accu
Epoch 3/10
23/23 [==============================] - 212s 9s/step - loss: 1.5579 - accuracy: 0.4529 - val_loss: 2.3984 - val_accu
Epoch 4/10
23/23 [==============================] - 206s 9s/step - loss: 1.5413 - accuracy: 0.4565 - val_loss: 2.2315 - val_accu
Epoch 5/10
23/23 [==============================] - 206s 9s/step - loss: 1.4381 - accuracy: 0.4864 - val_loss: 2.3587 - val_accu
Epoch 6/10
23/23 [==============================] - 216s 9s/step - loss: 1.4171 - accuracy: 0.4962 - val_loss: 2.3393 - val_accu
Epoch 7/10
23/23 [==============================] - 203s 9s/step - loss: 1.3803 - accuracy: 0.5217 - val_loss: 2.4700 - val_accu
Epoch 8/10
23/23 [==============================] - 212s 9s/step - loss: 1.3801 - accuracy: 0.5109 - val_loss: 2.1951 - val_accu
Epoch 9/10
23/23 [==============================] - 203s 9s/step - loss: 1.3811 - accuracy: 0.5172 - val_loss: 2.2769 - val_accu
Epoch 10/10
23/23 [==============================] - 206s 9s/step - loss: 1.3631 - accuracy: 0.5109 - val_loss: 2.1967 - val_accu
```

```
import matplotlib.pyplot as plt
epochs=10
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()
```




```
## Your code goes here.
from glob import glob
path_list = [x for x in glob(os.path.join(data_dir_train, '*', '*.jpg'))]
lesion_list = [os.path.basename(os.path.dirname(y)) for y in glob(os.path.join(data_dir_train, '*', '*.jpg'))]
len(path_list)
```

```
2239
```

CLASS DISTRIBUTION

```
dataframe_dict_original = dict(zip(path_list, lesion_list))
original_df = pd.DataFrame(list(dataframe_dict_original.items()),columns = ['Path','Label'])
original_df
```

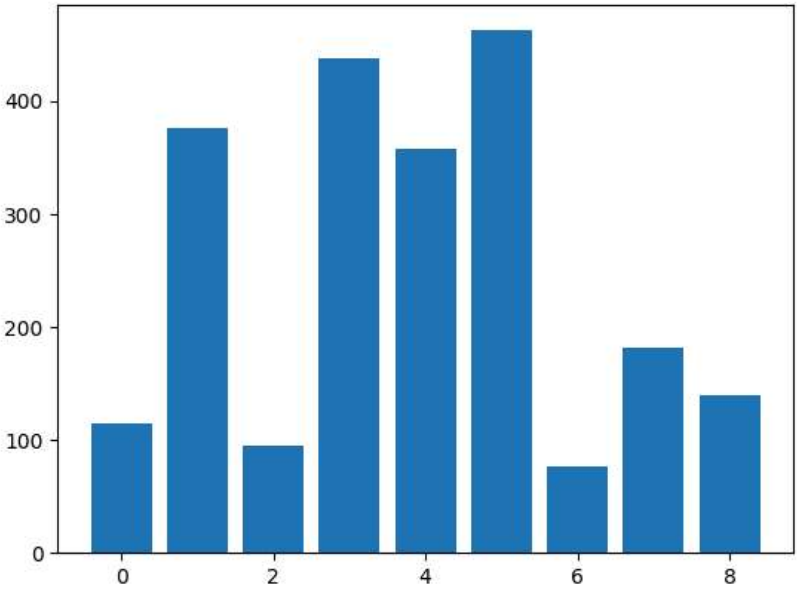
	Path	Label
0	/content/gdrive/MyDrive/Upgrad/Dataset/Skin ca...	actinic keratosis
1	/content/gdrive/MyDrive/Upgrad/Dataset/Skin ca...	actinic keratosis
2	/content/gdrive/MyDrive/Upgrad/Dataset/Skin ca...	actinic keratosis
3	/content/gdrive/MyDrive/Upgrad/Dataset/Skin ca...	actinic keratosis
4	/content/gdrive/MyDrive/Upgrad/Dataset/Skin ca...	actinic keratosis
...	...	...
2234	/content/gdrive/MyDrive/Upgrad/Dataset/Skin ca...	vascular lesion
2235	/content/gdrive/MyDrive/Upgrad/Dataset/Skin ca...	vascular lesion
2236	/content/gdrive/MyDrive/Upgrad/Dataset/Skin ca...	vascular lesion
2237	/content/gdrive/MyDrive/Upgrad/Dataset/Skin ca...	vascular lesion
2238	/content/gdrive/MyDrive/Upgrad/Dataset/Skin ca...	vascular lesion

2239 rows × 2 columns

Next steps: Generate code with original_df | View recommended plots

```
from sklearn.preprocessing import LabelEncoder
from collections import Counter
# split into input and output elements
X, y = original_df['Path'], original_df['Label']
# label encode the target variable
y = LabelEncoder().fit_transform(y)
# summarize distribution
counter = Counter(y)
for k,v in counter.items():
    per = v / len(y) * 100
    print('Class=%d, n=%d (%.3f%%)' % (k, v, per))
# plot the distribution
plt.bar(counter.keys(), counter.values())
plt.show()
```

```
Class=0, n=114 (5.092%)
Class=1, n=376 (16.793%)
Class=2, n=95 (4.243%)
Class=3, n=438 (19.562%)
Class=4, n=357 (15.945%)
Class=5, n=462 (20.634%)
Class=6, n=77 (3.439%)
Class=7, n=181 (8.084%)
Class=8, n=139 (6.208%)
```



Seborrheic keratosis has the least number of samples.

Pigmented benign keratosis has the highest number of samples

The classes 'basal cell carcinoma', 'melanoma', 'nevus', 'pigmented benign keratosis', dominate the data in terms proportionate number of samples.

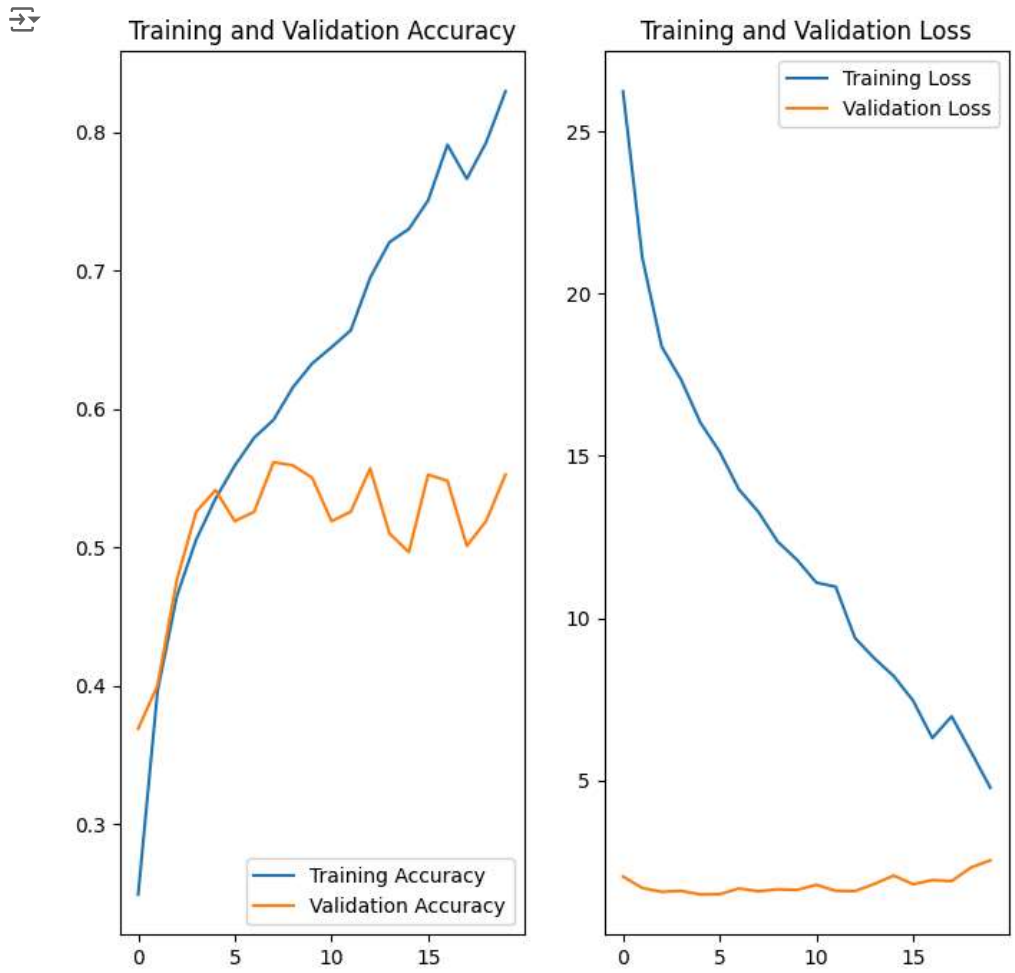
```
#https://datascience.stackexchange.com/questions/13490/how-to-set-class-weights-for-imbalanced-classes-in-keras

from sklearn.utils import class_weight
#Class=0, n=114 (5.092%)
#Class=1, n=376 (16.793%)
#Class=2, n=95 (4.243%)
#Class=3, n=438 (19.562%)
#Class=4, n=357 (15.945%)
#Class=5, n=462 (20.634%)
#Class=6, n=77 (3.439%)
#Class=7, n=181 (8.084%)
#Class=8, n=139 (6.208%)

class_weight = {0:5.09,
                1:16.79,
                2:4.24,
                3:19.56,
                4:15.94,
                5:20.63,
                6:3.43,
                7:8.08,
                8:6.20}

#class_weights = class_weight.compute_class_weight('balanced',np.unique(y_train),y_train)

### Your code goes here
num_classes = 9

#A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one outp
model = Sequential([
  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(img_height, img_width, 3)),

  #2D convolution layer (e.g. spatial convolution over images).
  layers.Conv2D(16, 3, padding='same', activation='relu'),

  #We slide over the feature map and extract tiles of a specified size.
  #Downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window
  layers.MaxPooling2D(),

  #We slide over the feature map and extract tiles of a specified size.
  layers.Conv2D(32, 3, padding='same', activation='relu'),

  layers.MaxPooling2D(),

  layers.Conv2D(64, 3, padding='same', activation='relu'),

  #We slide over the feature map and extract tiles of a specified size.
  #Advantages of downsampling - Decreased size of input for upcoming layers, Works against overfitting
  layers.MaxPooling2D(),

  #Flattening - Convert into 1D feature vector.  Flattens all its structure to create a single long feature vector
  ##Flattens the input. Does not affect the batch size.
  layers.Flatten(),

  #fully connected layer
  #A hidden layer in which each node is connected to every node in the subsequent hidden layer.
  #A fully connected layer is also known as a dense layer.

  layers.Dense(128, activation='relu'),

  #Dense is the only actual network layer in that model. A Dense layer feeds all outputs from the previous layer to all i
  #It's the most basic layer in neural networks. A Dense(10) has ten neurons. A Dense(512) has 512 neurons.
  #Dense implements the operation: output = activation(dot(input, kernel)
  #Dense Layer - A dense layer represents a matrix vector multiplication.  each input node is connected to each output no
  layers.Dense(num_classes)
  #Dense Layer - A dense layer represents a matrix vector multiplication.  each input node is connected to each output no
])


### Todo, choose an appropirate optimiser and loss function
#RMSprop. RMSprop is a very effective, but currently unpublished adaptive learning rate method
#Adam. Adam is a recently proposed update that looks a bit like RMSProp with momentum. The (simplified) update looks as f
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
```

```
              metrics=['accuracy'])

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

```
Epoch 1/20
56/56 [==============================] - 75s 1s/step - loss: 26.2302 - accuracy: 0.2494 - val_loss: 2.0401 - val_accu
Epoch 2/20
56/56 [==============================] - 65s 1s/step - loss: 21.0990 - accuracy: 0.3962 - val_loss: 1.6898 - val_accu
Epoch 3/20
56/56 [==============================] - 68s 1s/step - loss: 18.3680 - accuracy: 0.4648 - val_loss: 1.5692 - val_accu
Epoch 4/20
56/56 [==============================] - 65s 1s/step - loss: 17.3535 - accuracy: 0.5056 - val_loss: 1.6003 - val_accu
Epoch 5/20
56/56 [==============================] - 66s 1s/step - loss: 16.0312 - accuracy: 0.5352 - val_loss: 1.4891 - val_accu
Epoch 6/20
56/56 [==============================] - 65s 1s/step - loss: 15.1268 - accuracy: 0.5592 - val_loss: 1.4970 - val_accu
Epoch 7/20
56/56 [==============================] - 66s 1s/step - loss: 13.9632 - accuracy: 0.5792 - val_loss: 1.6745 - val_accu
Epoch 8/20
56/56 [==============================] - 64s 1s/step - loss: 13.2800 - accuracy: 0.5921 - val_loss: 1.5877 - val_accu
Epoch 9/20
56/56 [==============================] - 67s 1s/step - loss: 12.3600 - accuracy: 0.6155 - val_loss: 1.6477 - val_accu
Epoch 10/20
56/56 [==============================] - 64s 1s/step - loss: 11.7998 - accuracy: 0.6328 - val_loss: 1.6280 - val_accu
Epoch 11/20
56/56 [==============================] - 66s 1s/step - loss: 11.0973 - accuracy: 0.6445 - val_loss: 1.7887 - val_accu
Epoch 12/20
56/56 [==============================] - 65s 1s/step - loss: 10.9703 - accuracy: 0.6568 - val_loss: 1.6016 - val_accu
Epoch 13/20
56/56 [==============================] - 68s 1s/step - loss: 9.3915 - accuracy: 0.6948 - val_loss: 1.5936 - val_accur
Epoch 14/20
56/56 [==============================] - 65s 1s/step - loss: 8.7653 - accuracy: 0.7204 - val_loss: 1.8121 - val_accur
Epoch 15/20
56/56 [==============================] - 67s 1s/step - loss: 8.2196 - accuracy: 0.7299 - val_loss: 2.0727 - val_accur
Epoch 16/20
56/56 [==============================] - 64s 1s/step - loss: 7.4679 - accuracy: 0.7506 - val_loss: 1.8053 - val_accur
Epoch 17/20
56/56 [==============================] - 70s 1s/step - loss: 6.3087 - accuracy: 0.7907 - val_loss: 1.9318 - val_accur
Epoch 18/20
56/56 [==============================] - 64s 1s/step - loss: 6.9750 - accuracy: 0.7662 - val_loss: 1.9006 - val_accur
Epoch 19/20
56/56 [==============================] - 67s 1s/step - loss: 5.8864 - accuracy: 0.7924 - val_loss: 2.3207 - val_accur
Epoch 20/20
56/56 [==============================] - 65s 1s/step - loss: 4.7789 - accuracy: 0.8292 - val_loss: 2.5396 - val_accur
```

Double-click (or enter) to edit

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

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

epochs_range = range(epochs)

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

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




```
!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
```

```
#https://github.com/mdbloice/Augmentor
#https://github.com/mdbloice/Augmentor
datapath = '/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collaboration/Train'
import Augmentor
p = Augmentor.Pipeline(datapath)
#Every function requires you to specify a probability, which is used to decide if an operation is applied to an image as
p.rotate(probability=0.7, max_left_rotation=10, max_right_rotation=10)
p.zoom(probability=0.5, min_factor=1.1, max_factor=1.5)
p.sample(150)
p.process()
```

```
Initialised with 2239 image(s) found.
Output directory set to /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collab
Processing <PIL.Image.Image image mode=RGB size=600x450 at 0x7C1DCF2F6DD0>: 100%|██████████| 2239/2239 [05:02<00:00,
```

```
path_to_training_dataset="/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collaborat
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/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collab
Initialised with 376 image(s) found.
Output directory set to /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collab
Initialised with 95 image(s) found.
Output directory set to /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collab
Initialised with 438 image(s) found.
Output directory set to /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collab
Initialised with 357 image(s) found.
Output directory set to /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collab
Initialised with 462 image(s) found.
Output directory set to /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collab
Initialised with 77 image(s) found.
Output directory set to /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collab
Initialised with 181 image(s) found.
Output directory set to /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collab
Initialised with 139 image(s) found.
Output directory set to /content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging Collab
```

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

```
2239
```

```
import glob
```

```
path_list_new = glob.glob(os.path.join(data_dir_train, '*/*.jpg'))
```

```
path_list_new
```

```
 '/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/ISIC_0011013.jpg',
 '/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/ISIC_0011030.jpg',
 '/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/ISIC_0011031.jpg',
 '/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/ISIC_0011032.jpg',
 '/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/ISIC_0011039.jpg',
 '/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/ISIC_0011040.jpg',
 '/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/ISIC_0011042.jpg',
 '/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/ISIC_0011043.jpg',
 '/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/ISIC_0011045.jpg',
 '/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/ISIC_0011052.jpg',
 '/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/ISIC_0011056.jpg',
 '/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/ISIC_0011057.jpg',
 '/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/ISIC_0011077.jpg',
 '/content/gdrive/MyDrive/Upgrad/Dataset/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/ISIC_0011094.jpg',
```