Suhas Naik - Upgrad C67 batch

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')
```

```
Mounted at /content/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
data_dir_train = pathlib.Path("/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train")
data_dir_test = pathlib.Path('/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Test')
```

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

```
2249
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,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 2249 files belonging to 9 classes.
Using 1800 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 = val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir_train,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 2249 files belonging to 9 classes.
Using 449 files for validation.
```

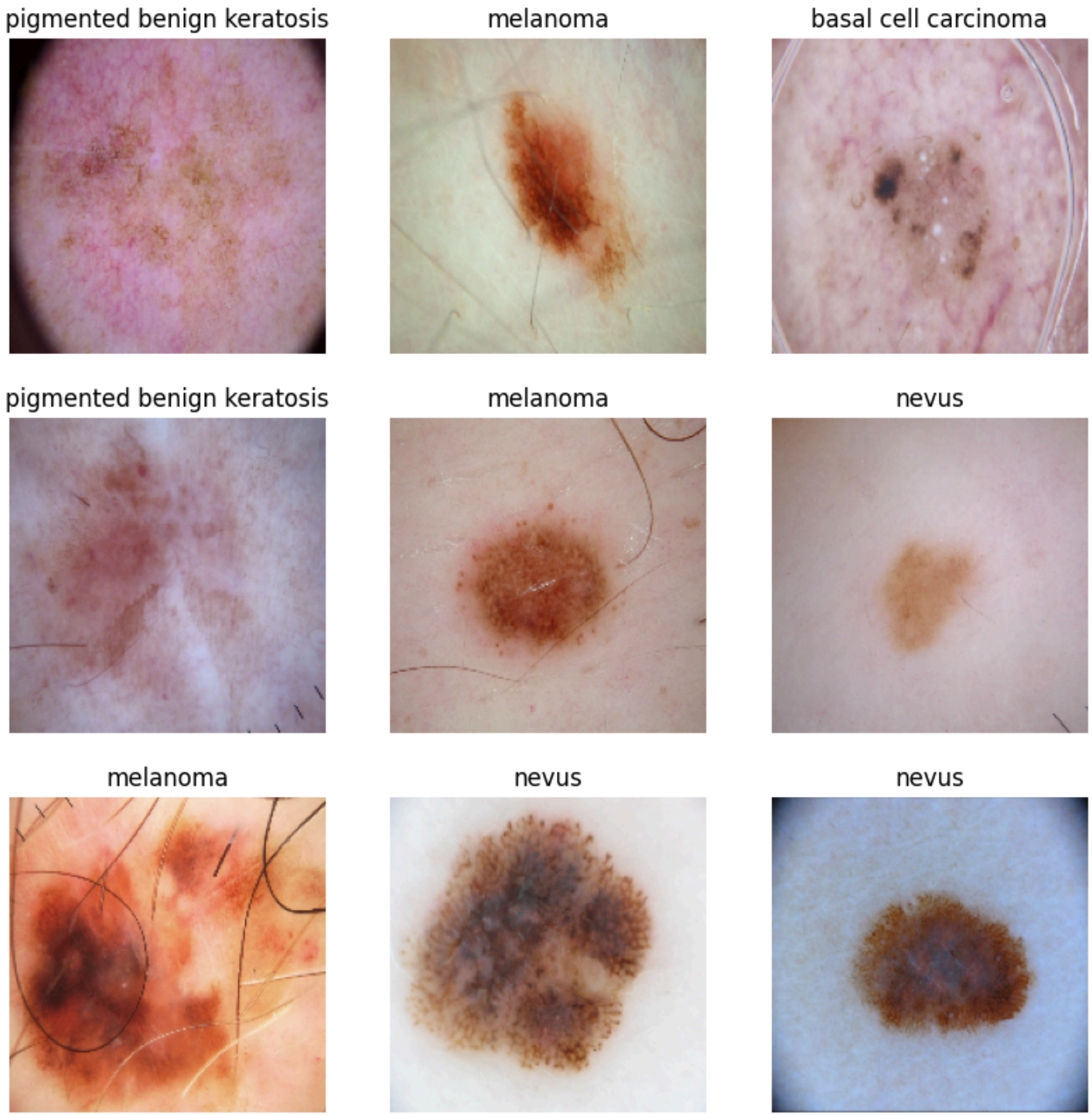
```
# List out all the classes of skin cancer and store them in a list.
# You can find the class names in the class_names attribute on these datasets.
# These correspond to the directory names in alphabetical order.
class_names = train_ds.class_names
print(class_names)
```

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

Visualize the data

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

```
import matplotlib.pyplot as plt

### your code goes here, you can use training or validation data to visualize

plt.figure(figsize=(10, 10))
for img,labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(img[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    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
num_classes = 9

model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)), # Changed from layers.experimental.preprocessing.Rescaling to layers.Rescaling
  layers.Conv2D(16, 3, padding='same', activation='relu'),

  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),

  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),

  layers.MaxPooling2D(),

  layers.Flatten(),

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

])
```

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=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])


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

Model: "sequential_1"

Layer (type)	Output Shape	Param #
rescaling_1 (Rescaling)	(None, 180, 180, 3)	0
conv2d_3 (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d_3 (MaxPooling2D)	(None, 90, 90, 16)	0
conv2d_4 (Conv2D)	(None, 90, 90, 32)	4,640
max_pooling2d_4 (MaxPooling2D)	(None, 45, 45, 32)	0
conv2d_5 (Conv2D)	(None, 45, 45, 64)	18,496
max_pooling2d_5 (MaxPooling2D)	(None, 22, 22, 64)	0
flatten_1 (Flatten)	(None, 30976)	0
dense_2 (Dense)	(None, 128)	3,965,056
dense_3 (Dense)	(None, 9)	1,161

Total params: 3,989,801 (15.22 MB)
Trainable params: 3,989,801 (15.22 MB)
Non-trainable params: 0 (0.00 B)

Train the model

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

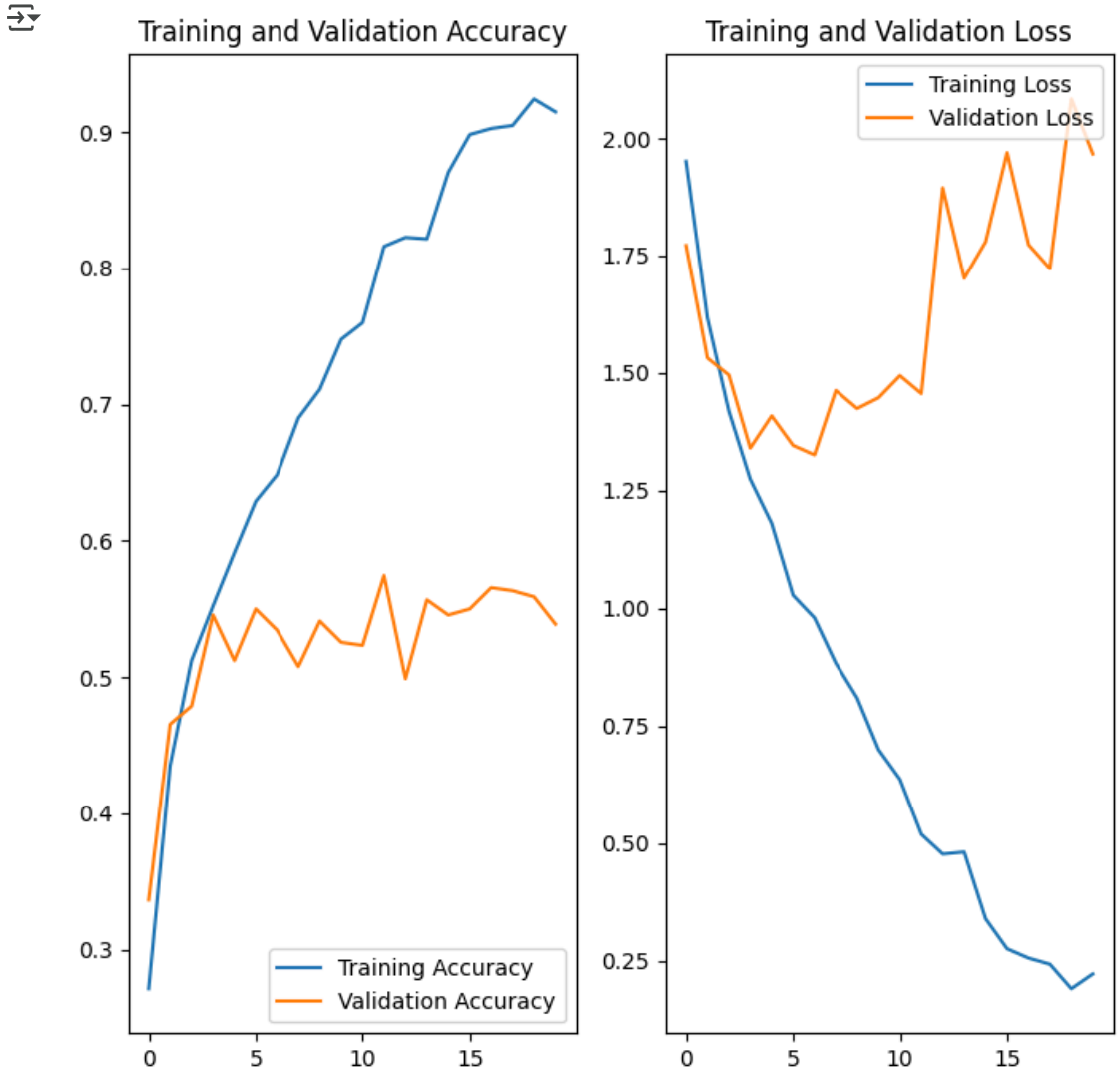
```
Epoch 1/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 368s 2s/step - accuracy: 0.2207 - loss: 2.0511 - val_accuracy: 0.3363 - val_loss: 1.7717
```

```
Epoch 2/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 17s 17ms/step - accuracy: 0.4075 - loss: 1.6704 - val_accuracy: 0.4655 - val_loss: 1.5319
Epoch 3/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.5069 - loss: 1.4056 - val_accuracy: 0.4788 - val_loss: 1.4956
Epoch 4/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.5636 - loss: 1.2576 - val_accuracy: 0.5457 - val_loss: 1.3402
Epoch 5/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.5907 - loss: 1.1716 - val_accuracy: 0.5122 - val_loss: 1.4086
Epoch 6/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.6340 - loss: 1.0284 - val_accuracy: 0.5501 - val_loss: 1.3458
Epoch 7/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.6481 - loss: 0.9644 - val_accuracy: 0.5345 - val_loss: 1.3256
Epoch 8/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.6999 - loss: 0.8898 - val_accuracy: 0.5078 - val_loss: 1.4627
Epoch 9/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.6958 - loss: 0.8485 - val_accuracy: 0.5412 - val_loss: 1.4243
Epoch 10/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.7800 - loss: 0.6174 - val_accuracy: 0.5256 - val_loss: 1.4472
Epoch 11/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.7550 - loss: 0.6096 - val_accuracy: 0.5234 - val_loss: 1.4943
Epoch 12/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.8229 - loss: 0.5304 - val_accuracy: 0.5746 - val_loss: 1.4561
Epoch 13/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.8422 - loss: 0.4442 - val_accuracy: 0.4989 - val_loss: 1.8943
Epoch 14/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.8125 - loss: 0.5148 - val_accuracy: 0.5568 - val_loss: 1.7016
Epoch 15/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.8735 - loss: 0.3418 - val_accuracy: 0.5457 - val_loss: 1.7795
Epoch 16/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9062 - loss: 0.2682 - val_accuracy: 0.5501 - val_loss: 1.9694
Epoch 17/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9050 - loss: 0.2456 - val_accuracy: 0.5657 - val_loss: 1.7734
Epoch 18/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9065 - loss: 0.2172 - val_accuracy: 0.5635 - val_loss: 1.7223
Epoch 19/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9346 - loss: 0.1703 - val_accuracy: 0.5590 - val_loss: 2.0835
Epoch 20/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9282 - loss: 0.1899 - val_accuracy: 0.5390 - val_loss: 1.9666
```

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

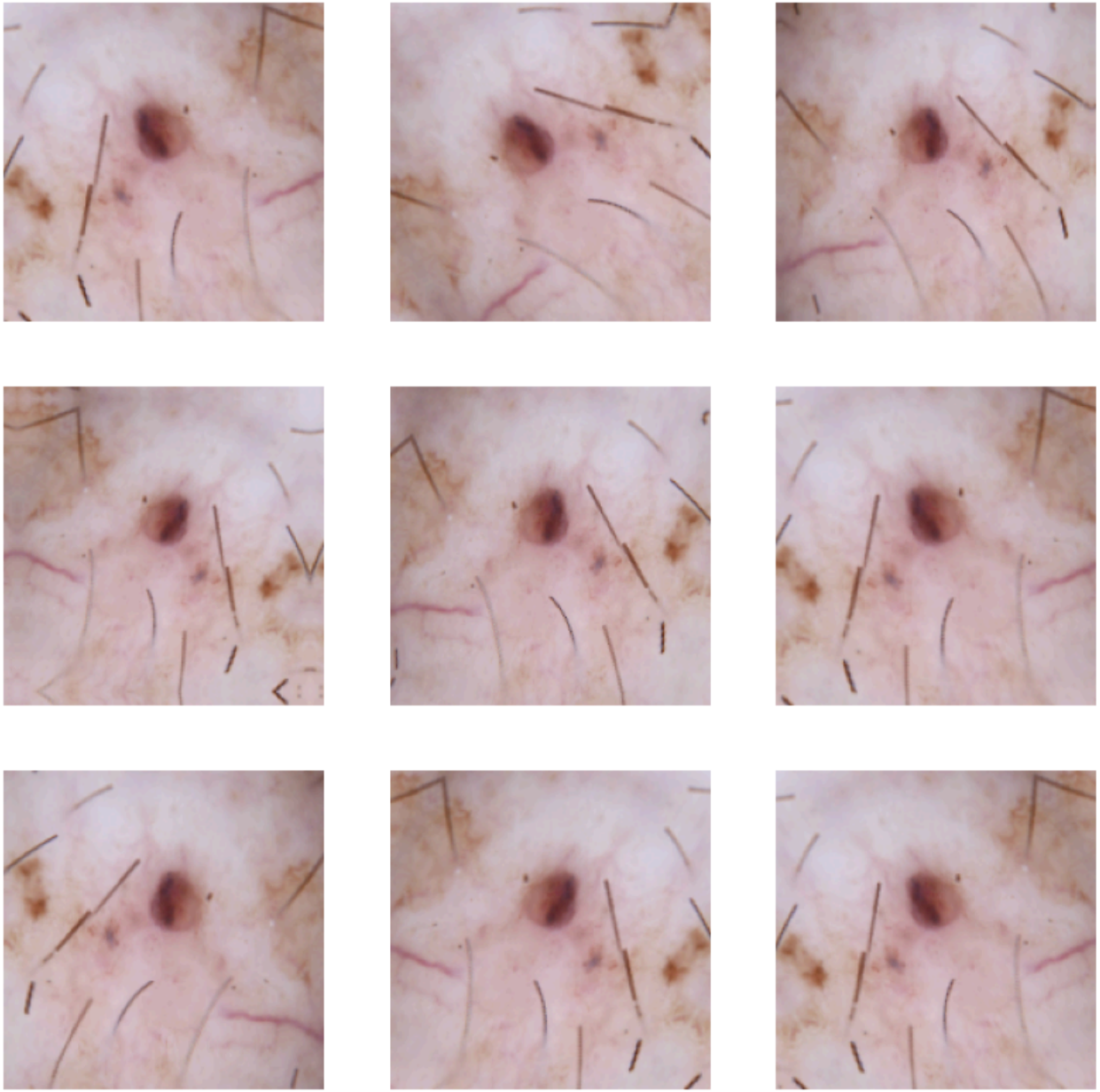
Inference from Model 1: As we can see from the plots, training accuracy and validation accuracy are off by large margin and the model has achieved only around 55% accuracy on the validation set.

Overfitting: In the plots above, the training accuracy is increasing linearly over time, whereas validation accuracy is only around 55% in the training process. This difference is a clear indicator of overfitting

```
# Todo, after you have analysed the model fit history for presence of underfit or overfit, choose an appropriate data augumentation strategy.
# Your code goes here
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal", # Changed from layers.experimental.preprocessing.RandomFlip to layers.RandomFlip
                                                 input_shape=(img_height,
                                                              img_width,
                                                              3)),
    layers.RandomRotation(0.1), # Changed from layers.experimental.preprocessing.RandomRotation to layers.RandomRotation
    layers.RandomZoom(0.1), # Changed from layers.experimental.preprocessing.RandomZoom to layers.RandomZoom
  ]
)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/preprocessing/tf_data_layer.py:19: UserWarning: Do not pass an `input_shape`/`input_dim` arg
  super().__init__(**kwargs)
```

```
# Todo, visualize how your augmentation strategy works for one instance of training image.
# Your code goes here
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")
```



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
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255), # Changed from layers.experimental.preprocessing.Rescaling to layers.Rescaling
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

Compiling the model

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

```
model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
sequential_2 (Sequential)	(None, 180, 180, 3)	0
rescaling_2 (Rescaling)	(None, 180, 180, 3)	0
conv2d_6 (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d_6 (MaxPooling2D)	(None, 90, 90, 16)	0
conv2d_7 (Conv2D)	(None, 90, 90, 32)	4,640
max_pooling2d_7 (MaxPooling2D)	(None, 45, 45, 32)	0
conv2d_8 (Conv2D)	(None, 45, 45, 64)	18,496
max_pooling2d_8 (MaxPooling2D)	(None, 22, 22, 64)	0
dropout (Dropout)	(None, 22, 22, 64)	0
flatten_2 (Flatten)	(None, 30976)	0
dense_4 (Dense)	(None, 128)	3,965,056
dense_5 (Dense)	(None, 9)	1,161

Total params: 3,989,801 (15.22 MB)
Trainable params: 3,989,801 (15.22 MB)
Non-trainable params: 0 (0.00 B)

Training the model

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

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

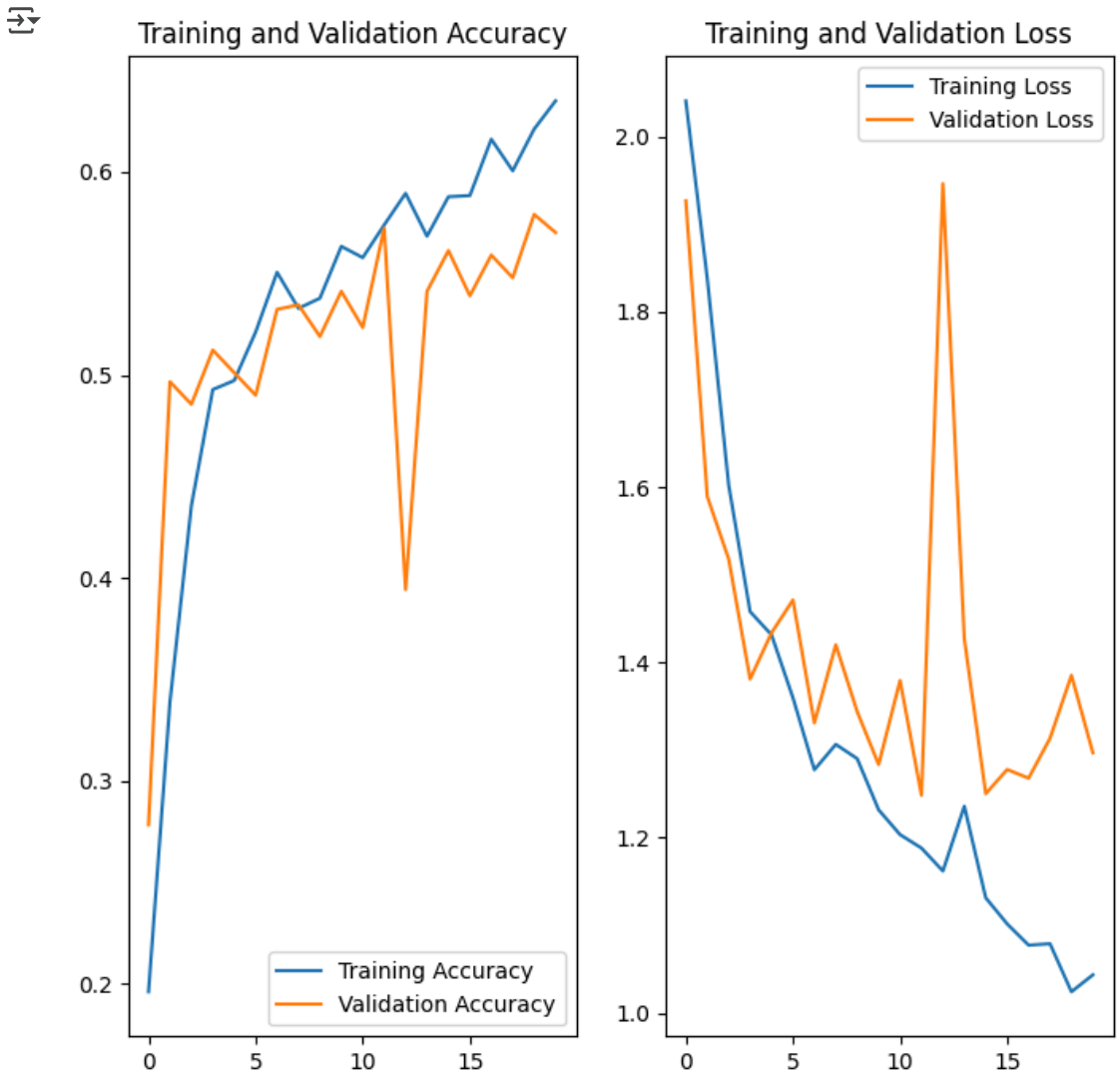
```
Epoch 1/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 5s 33ms/step - accuracy: 0.1648 - loss: 2.1267 - val_accuracy: 0.2784 - val_loss: 1.9264
Epoch 2/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.3022 - loss: 1.8826 - val_accuracy: 0.4967 - val_loss: 1.5892
Epoch 3/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.4221 - loss: 1.6045 - val_accuracy: 0.4855 - val_loss: 1.5179
```

```
Epoch 4/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.4828 - loss: 1.4868 - val_accuracy: 0.5122 - val_loss: 1.3810
Epoch 5/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.5108 - loss: 1.3881 - val_accuracy: 0.5011 - val_loss: 1.4337
Epoch 6/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.5099 - loss: 1.3886 - val_accuracy: 0.4900 - val_loss: 1.4712
Epoch 7/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.5548 - loss: 1.2753 - val_accuracy: 0.5323 - val_loss: 1.3310
Epoch 8/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.5342 - loss: 1.2793 - val_accuracy: 0.5345 - val_loss: 1.4201
Epoch 9/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - accuracy: 0.5393 - loss: 1.2486 - val_accuracy: 0.5189 - val_loss: 1.3438
Epoch 10/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.5561 - loss: 1.2287 - val_accuracy: 0.5412 - val_loss: 1.2835
Epoch 11/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.5472 - loss: 1.1965 - val_accuracy: 0.5234 - val_loss: 1.3794
Epoch 12/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.5784 - loss: 1.1645 - val_accuracy: 0.5724 - val_loss: 1.2484
Epoch 13/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.5976 - loss: 1.1191 - val_accuracy: 0.3942 - val_loss: 1.9459
Epoch 14/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.5281 - loss: 1.3893 - val_accuracy: 0.5412 - val_loss: 1.4269
Epoch 15/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.5951 - loss: 1.1075 - val_accuracy: 0.5612 - val_loss: 1.2503
Epoch 16/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.5914 - loss: 1.1064 - val_accuracy: 0.5390 - val_loss: 1.2778
Epoch 17/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.5955 - loss: 1.1243 - val_accuracy: 0.5590 - val_loss: 1.2679
Epoch 18/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - accuracy: 0.6163 - loss: 1.0569 - val_accuracy: 0.5479 - val_loss: 1.3134
Epoch 19/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.6089 - loss: 1.0416 - val_accuracy: 0.5791 - val_loss: 1.3853
Epoch 20/20
57/57 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.6364 - loss: 1.0512 - val_accuracy: 0.5702 - val_loss: 1.2969
```

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()
```



Model 2 : After data augmentation and adding dropping layer,overfitting has been reduced significantly

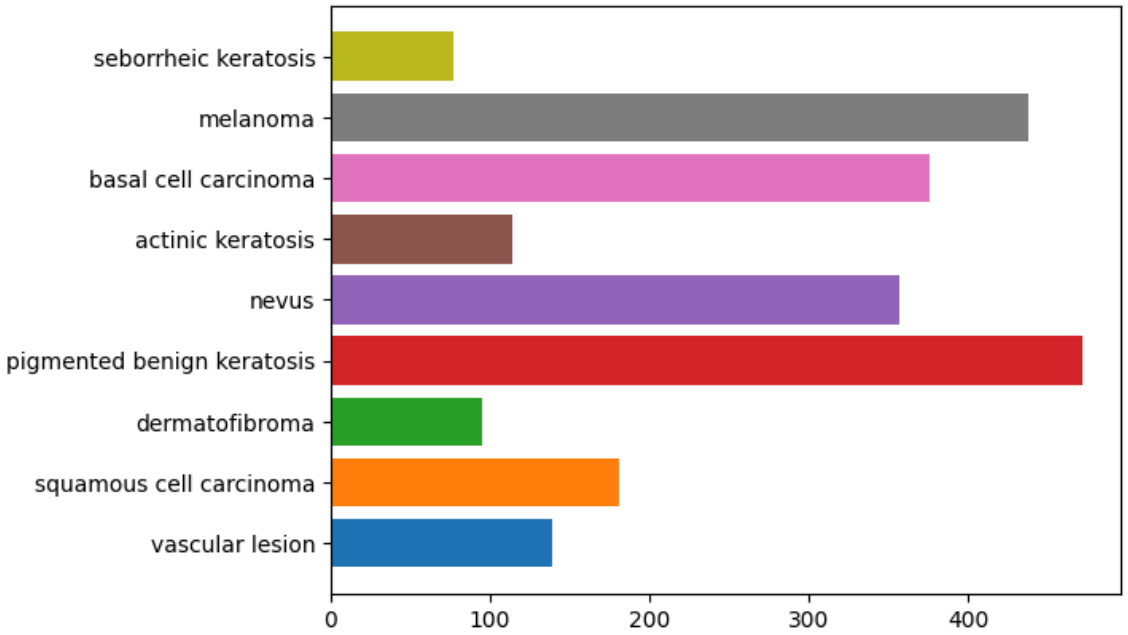
Todo: Find the distribution of classes in the training dataset.

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.

Counting each class

```
## Your code goes here.
from pathlib import Path
```

```
import os
path = '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train'
mn = 0
folders = ([name for name in os.listdir(path)
            if os.path.isdir(os.path.join(path, name))]) # get all directories
for folder in folders:
    contents = os.listdir(os.path.join(path,folder)) # get list of contents
    if len(contents) >= mn: # if greater than the limit, print folder and number of contents
        plt.barh(folder,len(contents))
```



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

```
Successfully installed Augmentor-0.2.12
```

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

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/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration'
path_to_training_dataset = path_to_training_dataset + "/Train"
print(path_to_training_dataset)
```

```
/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train
```

```
import Augmentor
## for i in class_names:
##    p = Augmentor.Pipeline(path_to_training_dataset + i)
##    p.rotate(probability=0.7, max_left_rotation=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.

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/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/actinic keratosis
Initialised with 376 image(s) found.
Output directory set to /content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/basal cell carci
Initialised with 95 image(s) found.
Output directory set to /content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/dermatofibroma/ou
Initialised with 438 image(s) found.
Output directory set to /content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/melanoma/output.P
Initialised with 357 image(s) found.
Output directory set to /content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/nevus/output.Proc
Initialised with 472 image(s) found.
Output directory set to /content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/pigmented benign
Initialised with 77 image(s) found.
Output directory set to /content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/seborrheic kerato
Initialised with 181 image(s) found.
Output directory set to /content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell ca
Initialised with 139 image(s) found.
Output directory set to /content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/vascular lesion/
```

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

```
print(data_dir_train)
```

```
/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train
```

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 glob
```

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

```
 '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell
carcinoma/output/squamous cell carcinoma_original_ISIC_0031191.jpg_c1a73f0b-2959-43c7-9a13-03abdb7142b6.jpg',
 '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell
carcinoma/output/squamous cell carcinoma_original_ISIC_0030341.jpg_06119f6b-d0e7-4f00-88a3-238adc45f285.jpg',
 '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell
carcinoma/output/squamous cell carcinoma_original_ISIC_0025948.jpg_a13a6169-2a5b-4668-b6d1-6c0c98967d95.jpg',
 '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell
carcinoma/output/squamous cell carcinoma_original_ISIC_0025831.jpg_d93de62c-b039-45b7-a827-e0ba43f8f57e.jpg',
 '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell
carcinoma/output/squamous cell carcinoma_original_ISIC_0032014.jpg_1d6c3195-e7f9-462d-b3a7-ad52679bdac6.jpg',
 '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell
carcinoma/output/squamous cell carcinoma_original_ISIC_0030714.jpg_7e88285f-a445-4f7b-9efb-e0c8d88e6ccb.jpg',
 '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell
carcinoma/output/squamous cell carcinoma_original_ISIC_0028644.jpg_85d7e6f2-035e-4753-a438-9bf0fabf335f.jpg',
 '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell
carcinoma/output/squamous cell carcinoma_original_ISIC_0026083.jpg_6cd0bca6-2009-4367-8d38-54b957b12fa9.jpg',
 '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell
carcinoma/output/squamous cell carcinoma_original_ISIC_0029549.jpg_bed0646d-9851-4761-8023-1b89a60d3a8b.jpg',
 '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell
carcinoma/output/squamous cell carcinoma_original_ISIC_0028158.jpg_6dcf3105-e2ab-47d9-b02c-57a7cc460ec1.jpg',
 '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell
carcinoma/output/squamous cell carcinoma_original_ISIC_0027529.jpg_7a76643f-f397-43a5-a117-2222670d77b1.jpg',
 '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell
carcinoma/output/squamous cell carcinoma_original_ISIC_0029851.jpg_73e6457c-f5fc-4b5e-a9a2-b66e9126ac2c.jpg',
 '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell
carcinoma/output/squamous cell carcinoma_original_ISIC_0030821.jpg_195097a4-75ba-4d88-ba5a-4d64501e9981.jpg',
 '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell
carcinoma/output/squamous cell carcinoma_original_ISIC_0030953.jpg_05cf510b-07e2-482e-8b8c-9fed118d86f0.jpg',
 '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell
carcinoma/output/squamous cell carcinoma_original_ISIC_0029362.jpg_b10b92db-e5e5-45c1-bd97-9d7a22ac525c.jpg',
 '/content/gdrive/MyDrive/Melonela Cancer/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous cell
carcinoma/output/squamous cell carcinoma_original_ISIC_0026927.jpg_8808810c-0f9e-4651-a67f-e3a848b68951.jpg',
 ...]
```

```
lesion_list_new = [os.path.basename(os.path.dirname(os.path.dirname(y))) for y in glob.glob(os.path.join(data_dir_train, '*','output', '*.jpg'))]
lesion_list_new
```

```
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
 'vascular lesion',
```

```
dataframe_dict_new = dict(zip(path_list, lesion_list_new))
```

```
import pandas as pd
import glob
import os
```

```
path_list = [x for x in glob.glob(os.path.join(data_dir_train, '*', '*.jpg'))]
lesson_list = [os.path.basename(os.path.dirname(y)) for y in glob.glob(os.path.join(data_dir_train, '*', '*.jpg'))]
dataframe_dict = dict(zip(path_list, lesson_list))
original_df = pd.DataFrame(list(dataframe_dict.items()),columns = ['Path','Label'])

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

# Previous code to create original_df and df2
new_df = pd.concat([original_df, df2], ignore_index=True)
```

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

Label	count
pigmented benign keratosis	972
melanoma	938
basal cell carcinoma	876
nevus	857
squamous cell carcinoma	681
vascular lesion	639
actinic keratosis	614
dermatofibroma	595
seborrheic keratosis	577

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= os.path.join(data_dir_train)
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 6749 files belonging to 9 classes.
Using 5400 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',
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 6749 files belonging to 9 classes.
Using 1349 files for validation.
```

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

```
## your code goes here
normalization_layer = layers.Rescaling(1./255) # 'Rescaling' is now directly under 'layers'
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixels values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
```

```
0.0047389013 0.9998366
```

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

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

Todo: Train your model

```
epochs = 50
## Your code goes here, use 50 epochs.

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

```
Epoch 38/50
169/169 ━━━━━━━━━━━━━━━━━━━━ 39s 220ms/step - accuracy: 0.8580 - loss: 0.3786 - val_accuracy: 0.7910 - val_loss: 0.7046
Epoch 39/50
169/169 ━━━━━━━━━━━━━━━━━━━━ 36s 210ms/step - accuracy: 0.8700 - loss: 0.3541 - val_accuracy: 0.8280 - val_loss: 0.5734
Epoch 40/50
169/169 ━━━━━━━━━━━━━━━━━━━━ 41s 243ms/step - accuracy: 0.8665 - loss: 0.3571 - val_accuracy: 0.8162 - val_loss: 0.6260
Epoch 41/50
169/169 ━━━━━━━━━━━━━━━━━━━━ 81s 238ms/step - accuracy: 0.8720 - loss: 0.3487 - val_accuracy: 0.8354 - val_loss: 0.5754
Epoch 42/50
169/169 ━━━━━━━━━━━━━━━━━━━━ 41s 240ms/step - accuracy: 0.8744 - loss: 0.3311 - val_accuracy: 0.8310 - val_loss: 0.6075
Epoch 43/50
169/169 ━━━━━━━━━━━━━━━━━━━━ 41s 238ms/step - accuracy: 0.8929 - loss: 0.2821 - val_accuracy: 0.8191 - val_loss: 0.6451
Epoch 44/50
169/169 ━━━━━━━━━━━━━━━━━━━━ 40s 235ms/step - accuracy: 0.8718 - loss: 0.3239 - val_accuracy: 0.7709 - val_loss: 0.8231
Epoch 45/50
169/169 ━━━━━━━━━━━━━━━━━━━━ 41s 235ms/step - accuracy: 0.8759 - loss: 0.3383 - val_accuracy: 0.7872 - val_loss: 0.7730
Epoch 46/50
169/169 ━━━━━━━━━━━━━━━━━━━━ 38s 220ms/step - accuracy: 0.8580 - loss: 0.3785 - val_accuracy: 0.8073 - val_loss: 0.6718
Epoch 47/50
169/169 ━━━━━━━━━━━━━━━━━━━━ 39s 233ms/step - accuracy: 0.8830 - loss: 0.3094 - val_accuracy: 0.8354 - val_loss: 0.5827
Epoch 48/50
169/169 ━━━━━━━━━━━━━━━━━━━━ 38s 217ms/step - accuracy: 0.8712 - loss: 0.3222 - val_accuracy: 0.8302 - val_loss: 0.6085
Epoch 49/50
169/169 ━━━━━━━━━━━━━━━━━━━━ 44s 235ms/step - accuracy: 0.8749 - loss: 0.3419 - val_accuracy: 0.8503 - val_loss: 0.5644
```