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
```

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.

```
## 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
```

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

```
import zipfile
import os

zip_path = "/content/gdrive/MyDrive/CNN_assignment.zip"

extract_path = "content/"

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
  zip_ref.extractall(extract_path)
```

```
# Defining the path for train and test images
## Todo: Update the paths of the train and test dataset
data_dir_train = pathlib.Path("/content/content/Skin cancer ISIC The International Skin Imaging Collaboration/Train")
data_dir_test = pathlib.Path('/content/content/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)
```

```
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,
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size
)
```

```
Found 2239 files belonging to 9 classes.
```

```
## 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_test,
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size
)
```

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

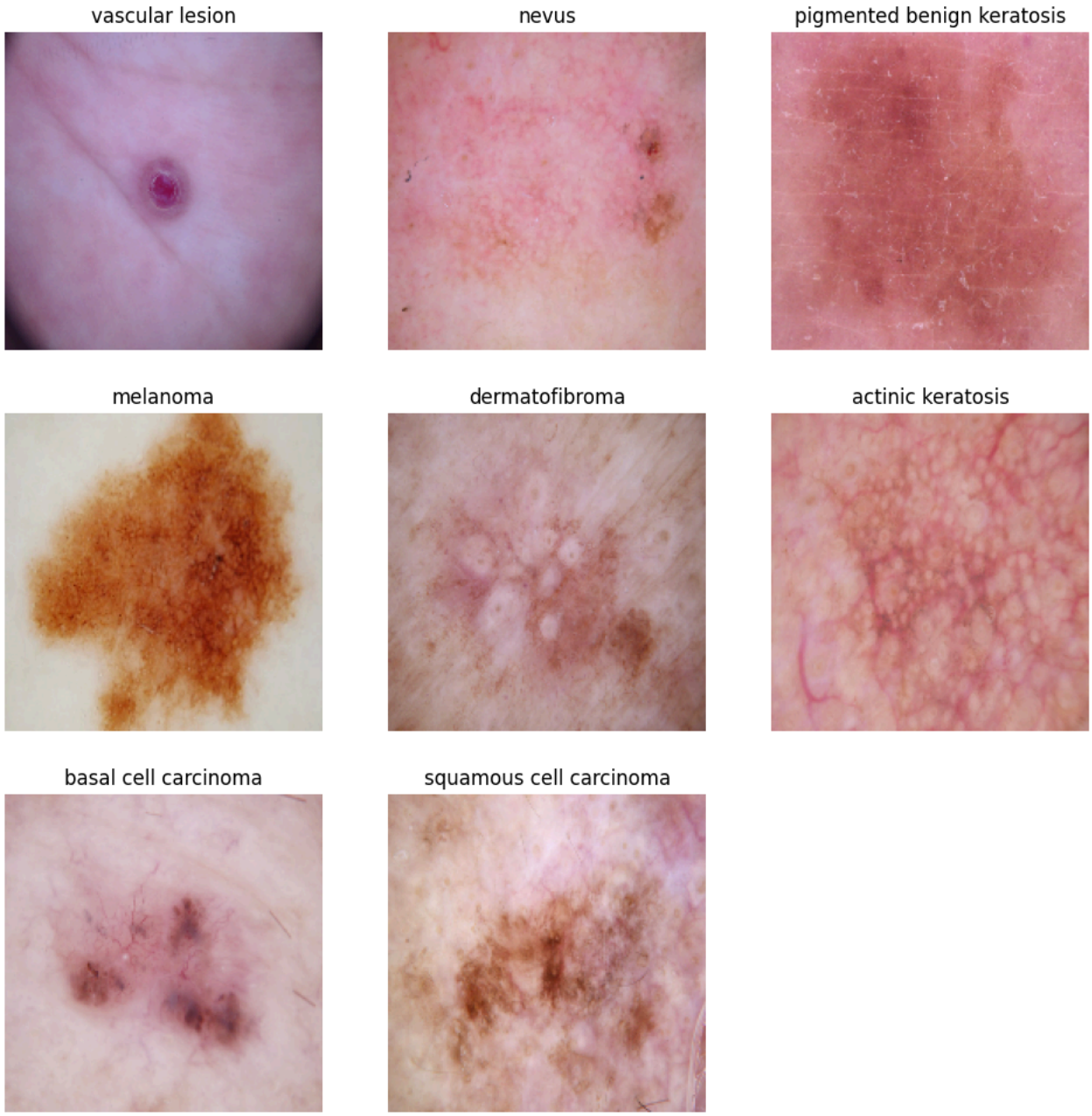
```
# List out all the classes of skin cancer and store them in a list.
# You can find the class names in the class_names attribute on these datasets.
# These correspond to the directory names in alphabetical order.
class_names = train_ds.class_names
print(class_names)
```

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

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

# Get a batch of images and labels
image_batch, label_batch = next(iter(train_ds))

# Dictionary to store the first image of each class
class_images = {}
for img, label in zip(image_batch, label_batch):
    class_name = class_names[label.numpy()]
    if class_name not in class_images:
        class_images[class_name] = img  # Store the first image of the class
    if len(class_images) == len(class_names):  # Stop once we have one of each
        break

# Plot one image per class
plt.figure(figsize=(12, 12))
for i, (class_name, img) in enumerate(class_images.items()):
    plt.subplot(3, 3, i + 1)  # 3x3 grid for 9 classes
    plt.imshow(img.numpy().astype("uint8"))
    plt.title(class_name)
    plt.axis("off")

plt.show()
```



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
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Define input shape
input_shape = (img_height, img_width, 3)  # Image dimensions with 3 color channels
```

```
# Build CNN Model
model = keras.Sequential([
    # Normalize pixel values
    layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
    # First convolutional block
    layers.Conv2D(32, (3,3), activation='relu', padding='same'),
    layers.MaxPooling2D(2,2),

    # Second convolutional block
    layers.Conv2D(64, (3,3), activation='relu', padding='same'),
    layers.MaxPooling2D(2,2),

    # Third convolutional block
    layers.Conv2D(128, (3,3), activation='relu', padding='same'),
    layers.MaxPooling2D(2,2),

    # Flatten layer to convert 2D feature maps to 1D
    layers.Flatten(),

    # Fully connected layers
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),  # Dropout to prevent overfitting
    layers.Dense(64, activation='relu'),

    # Output layer with softmax activation for 9 classes
    layers.Dense(len(class_names), activation='softmax')
])
```

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

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='sparse_categorical_crossentropy',  # For integer-labeled classes
              metrics=['accuracy'])
```

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

Model: "sequential"

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 180, 180, 3)	0
conv2d (Conv2D)	(None, 180, 180, 32)	896
max_pooling2d (MaxPooling2D)	(None, 90, 90, 32)	0
conv2d_1 (Conv2D)	(None, 90, 90, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 45, 45, 64)	0
conv2d_2 (Conv2D)	(None, 45, 45, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 22, 22, 128)	0
flatten (Flatten)	(None, 61952)	0
dense (Dense)	(None, 128)	7,929,984
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 9)	585

Total params: 8,032,073 (30.64 MB)
Trainable params: 8,032,073 (30.64 MB)
Non-trainable params: 0 (0.00 B)

Train the model

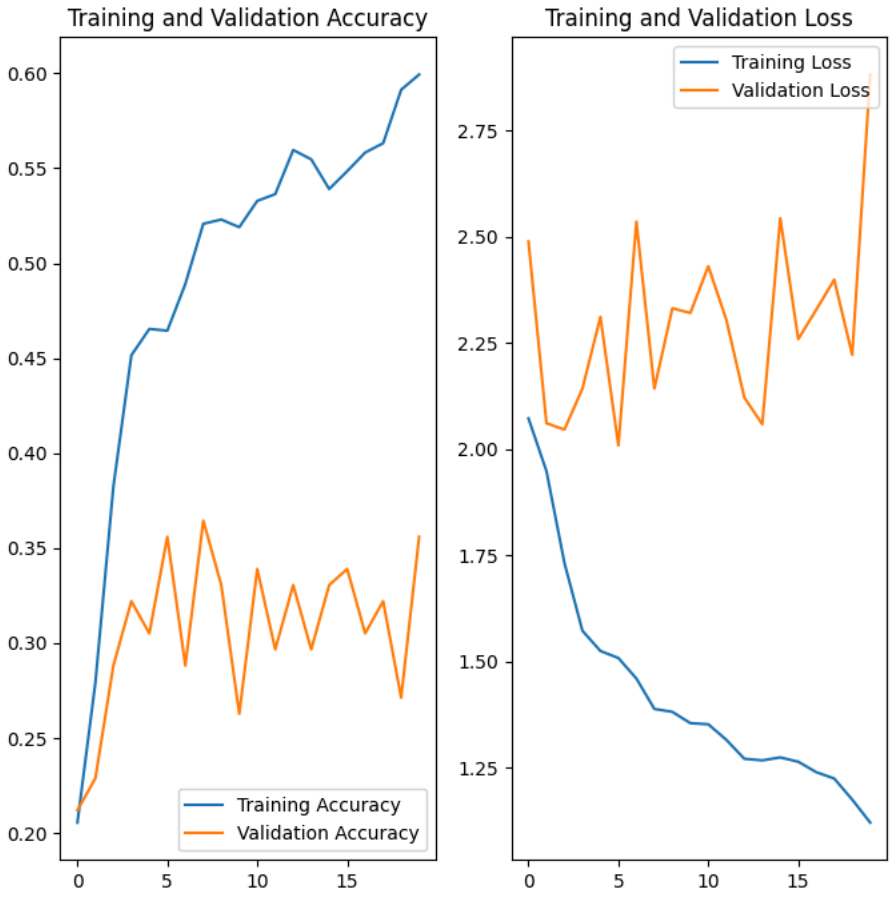
```
epochs = 20
history = model.fit(
  train_ds,
```

```
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 30s 169ms/step - accuracy: 0.1795 - loss: 2.1709 - val_accuracy: 0.2119 - val_loss: 2.4892
Epoch 2/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 14s 30ms/step - accuracy: 0.2636 - loss: 2.0034 - val_accuracy: 0.2288 - val_loss: 2.0611
Epoch 3/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.3589 - loss: 1.7925 - val_accuracy: 0.2881 - val_loss: 2.0463
Epoch 4/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.4310 - loss: 1.6159 - val_accuracy: 0.3220 - val_loss: 2.1427
Epoch 5/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.4678 - loss: 1.5415 - val_accuracy: 0.3051 - val_loss: 2.3115
Epoch 6/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.4936 - loss: 1.4673 - val_accuracy: 0.3559 - val_loss: 2.0089
Epoch 7/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.4788 - loss: 1.4687 - val_accuracy: 0.2881 - val_loss: 2.5358
Epoch 8/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.5242 - loss: 1.3954 - val_accuracy: 0.3644 - val_loss: 2.1431
Epoch 9/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.5359 - loss: 1.3512 - val_accuracy: 0.3305 - val_loss: 2.3319
Epoch 10/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.5382 - loss: 1.3122 - val_accuracy: 0.2627 - val_loss: 2.3209
Epoch 11/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.5263 - loss: 1.3562 - val_accuracy: 0.3390 - val_loss: 2.4310
Epoch 12/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.5392 - loss: 1.2926 - val_accuracy: 0.2966 - val_loss: 2.3049
Epoch 13/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.5419 - loss: 1.3059 - val_accuracy: 0.3305 - val_loss: 2.1218
Epoch 14/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.5627 - loss: 1.2431 - val_accuracy: 0.2966 - val_loss: 2.0587
Epoch 15/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.5251 - loss: 1.2913 - val_accuracy: 0.3305 - val_loss: 2.5439
Epoch 16/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.5453 - loss: 1.2971 - val_accuracy: 0.3390 - val_loss: 2.2592
Epoch 17/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.5672 - loss: 1.2279 - val_accuracy: 0.3051 - val_loss: 2.3285
Epoch 18/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.5510 - loss: 1.2390 - val_accuracy: 0.3220 - val_loss: 2.3992
Epoch 19/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.5922 - loss: 1.2063 - val_accuracy: 0.2712 - val_loss: 2.2224
Epoch 20/20
70/70 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.5961 - loss: 1.1187 - val_accuracy: 0.3559 - val_loss: 2.8824
```

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

Observations:

1. **Training Accuracy consistently higher than Validation Accuracy:** The training accuracy is noticeably and consistently higher than the validation accuracy throughout the training process. This is a potential indicator of overfitting.
2. **Validation Accuracy plateaus and fluctuates:** The validation accuracy seems to plateau or stagnate after a few epochs, showing limited improvement despite continued training. It also exhibits fluctuations, suggesting the model might be struggling to generalize well to unseen data.
3. **Training Loss continues to decrease, Validation Loss plateaus/increases** : The training loss steadily decreases, which is expected. However, the validation loss, after an initial decrease, plateaus and even shows some increase towards the later epochs. This is a strong indication of overfitting, where the model is improving its performance on the training data at the expense of generalization to the validation data.
4. **Gap between Training and Validation Loss:** There's a noticeable and widening gap between the training and validation loss as training progresses, further supporting the suspicion of overfitting.

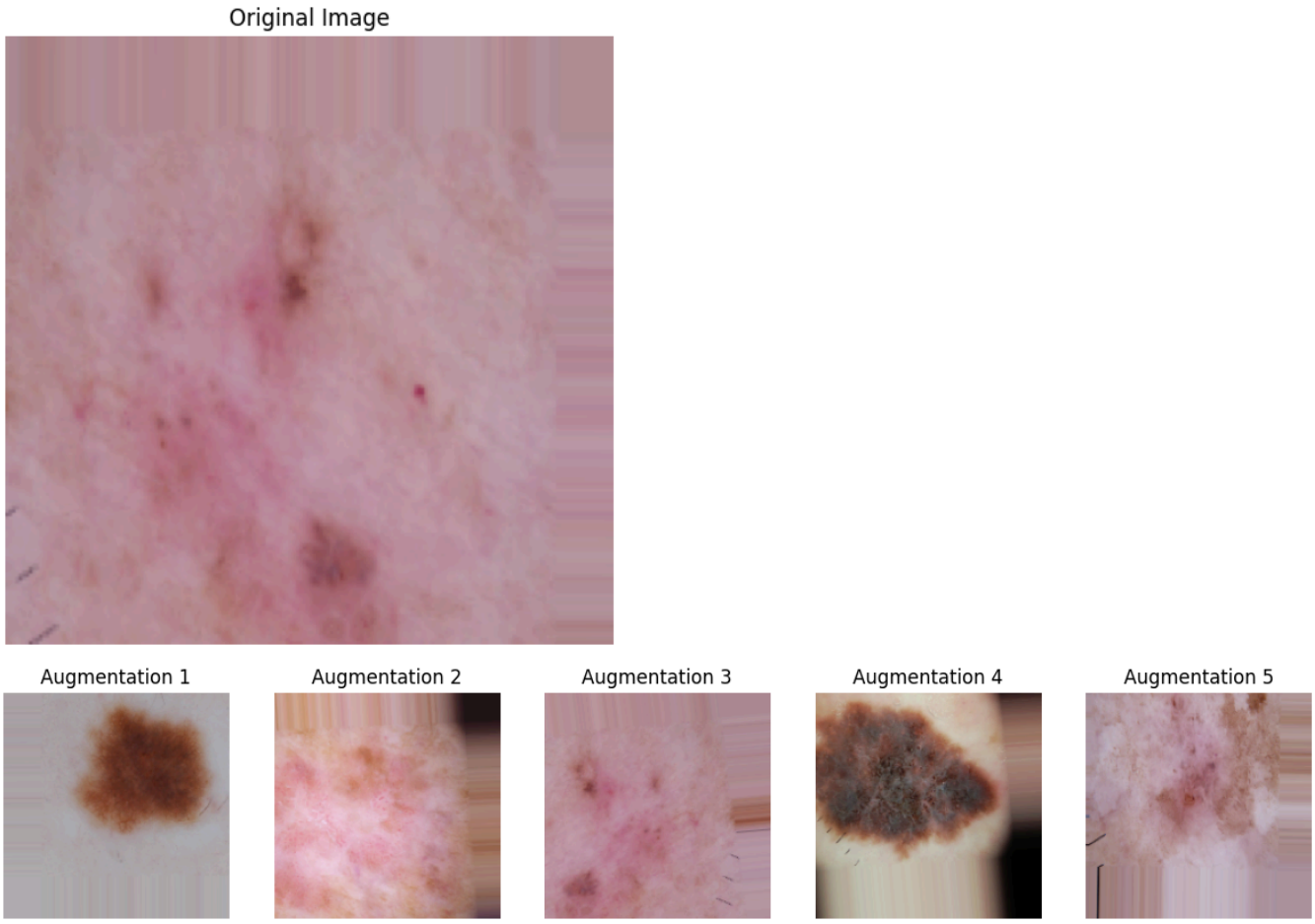
Interpretation and Findings:

Evidence of Overfitting: The trends observed strongly suggest that the model is overfitting the training data. The model is likely learning noise or specific features of the training set that are not representative of the underlying data distribution, leading to poor performance on unseen data (the validation set).

Lack of Underfitting Evidence: There are no clear signs of underfitting. The training accuracy is reasonably high, and the training loss is decreasing, suggesting the model is learning something meaningful from the data.

```
# Todo, after you have analysed the model fit history for presence of underfit or overfit, choose an appropriate data augume
# Your code goes here

from tensorflow.keras.preprocessing.image import ImageDataGenerator

def apply_data_augmentation(train_data):
    # Apply data augmentation techniques
    datagen = ImageDataGenerator(
        rotation_range=20,
```

```
            width_shift_range=0.2,
            height_shift_range=0.2,
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True,
            fill_mode='nearest'
        )

        # Convert the PrefetchDataset to a NumPy array
        # Assuming train_data is your PrefetchDataset
        images, labels = next(iter(train_data.take(1)))  # Get one batch
        images = images.numpy()  # Convert to NumPy array

        # Now you can apply augmentation
        augmented_data = datagen.flow(images, labels, batch_size=images.shape[0]) # Pass labels as well for flow

        return augmented_data

    # Apply data augmentation to training data if overfitting is detected
    augmented_train_data = apply_data_augmentation(train_ds)



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

    import numpy as np

    # Get a batch of images and labels from the augmented data
    augmented_images, augmented_labels = next(iter(augmented_train_data))

    # Select the first image for visualization
    image_to_visualize = augmented_images[0]

    # Display the original image (before augmentation)
    plt.figure(figsize=(6, 6))
    plt.imshow(image_to_visualize.astype("uint8"))
    plt.title("Original Image")
    plt.axis("off")
    plt.show()

    # Apply augmentation and display a few augmented versions
    num_augmentations_to_display = 5  # Number of augmented images to show

    plt.figure(figsize=(15, 5))
    for i in range(num_augmentations_to_display):
        # Use __next__() instead of next()
        augmented_image = augmented_train_data.__next__()[0][0]  # Get the next augmented image
        plt.subplot(1, num_augmentations_to_display, i + 1)
        plt.imshow(augmented_image.astype("uint8"))
        plt.title(f"Augmentation {i + 1}")
        plt.axis("off")

    plt.show()
```



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

# Define input shape
input_shape = (img_height, img_width, 3)  # Image dimensions with 3 color channels

# Build CNN Model with Dropout
model = keras.Sequential([
    # Normalize pixel values
    layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
    # First convolutional block
    layers.Conv2D(32, (3,3), activation='relu', padding='same'),
    layers.MaxPooling2D(2,2),

    # Second convolutional block
    layers.Conv2D(64, (3,3), activation='relu', padding='same'),
    layers.MaxPooling2D(2,2),

    # Third convolutional block
    layers.Conv2D(128, (3,3), activation='relu', padding='same'),
    layers.MaxPooling2D(2,2),

    # Flatten layer to convert 2D feature maps to 1D
    layers.Flatten(),

    # Fully connected layers with Dropout
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),  # Dropout layer to reduce overfitting
    layers.Dense(64, activation='relu'),

    # Output layer with softmax activation for 9 classes
    layers.Dense(len(class_names), activation='softmax')
])
```

Compiling the model

```
## Your code goes here
# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Training the model

```
## Your code goes here, note: train your model for 20 epochs
# Train the model
epochs = 20
history = model.fit(
  augmented_train_data, # Use augmented training data
  validation_data=val_ds,
  epochs=epochs
)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `P
  self._warn_if_super_not_called()
Epoch 1/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step - accuracy: 0.0625 - loss: 2.2137 - val_accuracy: 0.1356 - val_loss: 2.2650
Epoch 2/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 328ms/step - accuracy: 0.0625 - loss: 2.2505 - val_accuracy: 0.1780 - val_loss: 2.1233
Epoch 3/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 313ms/step - accuracy: 0.0312 - loss: 2.1003 - val_accuracy: 0.2203 - val_loss: 2.1311
Epoch 4/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 320ms/step - accuracy: 0.2188 - loss: 2.0993 - val_accuracy: 0.1441 - val_loss: 2.1222
Epoch 5/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 333ms/step - accuracy: 0.1875 - loss: 2.0190 - val_accuracy: 0.1356 - val_loss: 2.1277
Epoch 6/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 347ms/step - accuracy: 0.3125 - loss: 1.9218 - val_accuracy: 0.1356 - val_loss: 2.1849
Epoch 7/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 315ms/step - accuracy: 0.2188 - loss: 2.0757 - val_accuracy: 0.1356 - val_loss: 2.1863
Epoch 8/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 310ms/step - accuracy: 0.0938 - loss: 1.9019 - val_accuracy: 0.1356 - val_loss: 2.1813
Epoch 9/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 303ms/step - accuracy: 0.1562 - loss: 1.9385 - val_accuracy: 0.1356 - val_loss: 2.1628
Epoch 10/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 317ms/step - accuracy: 0.2188 - loss: 1.9276 - val_accuracy: 0.1356 - val_loss: 2.1574
Epoch 11/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 342ms/step - accuracy: 0.1562 - loss: 2.0444 - val_accuracy: 0.1356 - val_loss: 2.1447
Epoch 12/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 315ms/step - accuracy: 0.2188 - loss: 1.9541 - val_accuracy: 0.1356 - val_loss: 2.1451
Epoch 13/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 301ms/step - accuracy: 0.1875 - loss: 1.9124 - val_accuracy: 0.1356 - val_loss: 2.1655
Epoch 14/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 305ms/step - accuracy: 0.3125 - loss: 1.8644 - val_accuracy: 0.1356 - val_loss: 2.2268
Epoch 15/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 399ms/step - accuracy: 0.2500 - loss: 1.8472 - val_accuracy: 0.1356 - val_loss: 2.2852
Epoch 16/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 460ms/step - accuracy: 0.1250 - loss: 1.9185 - val_accuracy: 0.1356 - val_loss: 2.2912
Epoch 17/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 485ms/step - accuracy: 0.1250 - loss: 1.9816 - val_accuracy: 0.1356 - val_loss: 2.2129
Epoch 18/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 483ms/step - accuracy: 0.2188 - loss: 1.7952 - val_accuracy: 0.1780 - val_loss: 2.1811
Epoch 19/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 627ms/step - accuracy: 0.1875 - loss: 1.8928 - val_accuracy: 0.1780 - val_loss: 2.1714
Epoch 20/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 608ms/step - accuracy: 0.2812 - loss: 1.7869 - val_accuracy: 0.1780 - val_loss: 2.2107
```

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?

Observations:

Training Accuracy fluctuates wildly: The training accuracy shows significant fluctuations and spikes throughout the training process. This is unusual and could indicate instability in the training.

Validation Accuracy is very low and stagnant: The validation accuracy is extremely low (around 0.14) and shows almost no improvement throughout training. It's practically flat.

Training Loss fluctuates and doesn't decrease consistently: Similar to the training accuracy, the training loss exhibits large fluctuations and doesn't show a clear, consistent downward trend.

Validation Loss is high and relatively stagnant: The validation loss is high (above 2.0) and, like the validation accuracy, shows minimal improvement.

Interpretation and Findings:

Clear Evidence of Underfitting: The extremely low and stagnant validation accuracy, along with the high and relatively unchanging validation loss, strongly suggest that the model is underfitting the data. This means the model is not learning the underlying patterns in the data and is failing to generalize.

Possible Reasons for Underfitting:

Model Complexity: The model might be too simple to capture the complexity of the data. It could lack the capacity to learn the necessary features. **Insufficient Training:** Although it's hard to tell definitively without knowing the training duration, it's possible that the model hasn't been trained for long enough to converge (though the lack of any improvement suggests otherwise). **Problem with the Data:** There might be issues with the data itself, such as incorrect labels, insufficient data, or highly noisy data, which could hinder the learning process. **Bug in the**

Code: It's also possible that there is a bug in the code related to the model training, data loading, or evaluation process. Inappropriate Hyperparameters: The model's hyperparameters might be poorly chosen, preventing effective learning.

Comparison with the Previous Run: **bold text** The previous model run showed signs of overfitting, where the training accuracy was significantly higher than the validation accuracy, and the validation loss plateaued or increased.

Is there Improvement? No, the model has gotten significantly worse. This run shows severe underfitting, a much worse outcome than the previous overfitting. The model is essentially not learning anything meaningful.

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.

```
## Your code goes here.

from glob import glob  # Import glob

# Assuming 'data_dir_train' is the path to your training dataset directory
# Get a list of all image paths
path_list = [x for x in glob(os.path.join(data_dir_train, '*/*.jpg'))]

# Extract class labels from the paths (assuming your directory structure is class_name/image.jpg)
lesion_list = [os.path.basename(os.path.dirname(y)) for y in glob(os.path.join(data_dir_train, '*/*.jpg'))]

# Create a pandas DataFrame to store the data
dataframe_dict = dict(zip(path_list, lesion_list))
original_df = pd.DataFrame(list(dataframe_dict.items()), columns=['Path', 'Label'])

# Get the class distribution
class_distribution = original_df['Label'].value_counts()

print(class_distribution)
```

```
Label
pigmented benign keratosis    462
melanoma                      438
basal cell carcinoma          376
nevus                         357
squamous cell carcinoma       181
vascular lesion               139
actinic keratosis             114
dermatofibroma                 95
seborrheic keratosis           77
Name: count, dtype: int64
```

Todo: Write your findings here:

- Which class has the least number of samples?

Seborrheic keratosis has the least number of samples with only 77 images.

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

The following classes dominate the data:

1. Pigmented benign keratosis: With 462 samples, it has the highest number of images.
2. Melanoma: With 438 samples, it is the second most dominant class.
3. Basal cell carcinoma: With 376 samples, it has a substantial number of images.
4. Nevus: With 357 samples, it is also a significant portion of the dataset.

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="To do"
import Augmentor
for i in class_names:
    p = Augmentor.Pipeline(str(data_dir_train) + '/' + 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/content/Skin cancer ISIC The International Skin Imaging Collaboration/Train/actinic ker
Initialised with 376 image(s) found.
Output directory set to /content/content/Skin cancer ISIC The International Skin Imaging Collaboration/Train/basal cell
Initialised with 95 image(s) found.
Output directory set to /content/content/Skin cancer ISIC The International Skin Imaging Collaboration/Train/dermatofibr
Initialised with 438 image(s) found.
Output directory set to /content/content/Skin cancer ISIC The International Skin Imaging Collaboration/Train/melanoma/ou
Initialised with 357 image(s) found.
Output directory set to /content/content/Skin cancer ISIC The International Skin Imaging Collaboration/Train/nevus/outpu
Initialised with 462 image(s) found.
Output directory set to /content/content/Skin cancer ISIC The International Skin Imaging Collaboration/Train/pigmented b
Initialised with 77 image(s) found.
Output directory set to /content/content/Skin cancer ISIC The International Skin Imaging Collaboration/Train/seborrheic
Initialised with 181 image(s) found.
Output directory set to /content/content/Skin cancer ISIC The International Skin Imaging Collaboration/Train/squamous ce
Initialised with 139 image(s) found.
Output directory set to /content/content/Skin cancer ISIC The International Skin Imaging Collaboration/Train/vascular le
```

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.

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

```
 '/content/content/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/output/melanoma_original_ISIC_0010143.jpg_b8b086e1-629c-4d95-97f6-7c3cea177ad5.jpg',
 '/content/content/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/output/melanoma_original_ISIC_0010400.jpg_c8bdc5b0-384e-4b92-8110-c633e2b6a63e.jpg',
 '/content/content/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/output/melanoma_original_ISIC_0000405.jpg_5e2257d6-f3aa-4f55-a56a-8aee751e5fb4.jpg',
 '/content/content/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/output/melanoma_original_ISIC_0000292.jpg_d61c7ae2-af1f-4b7c-8e13-bcae2785b099.jpg',
 '/content/content/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/output/melanoma_original_ISIC_0010119.jpg_83f4c4fa-6bd1-4994-833d-05d9ef4b4c59.jpg',
 '/content/content/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/output/melanoma_original_ISIC_0010478.jpg_3758e904-b4bf-42d4-8156-4573e09ed9b1.jpg',
 '/content/content/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/output/melanoma_original_ISIC_0010729.jpg_57b7ea12-aa90-430d-9a78-f375d0747729.jpg',
 '/content/content/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/output/melanoma_original_ISIC_0010548.jpg_cc158895-ee2f-4a7d-8f72-3aaf154fe93a.jpg',
 '/content/content/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/output/melanoma_original_ISIC_0000142.jpg_0669bff8-7c3a-47ff-8d27-2a23448a52d0.jpg',
 '/content/content/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/output/melanoma_original_ISIC_0009960.jpg_52429867-4cc2-4b2c-82c1-e438615ca272.jpg',
 '/content/content/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/output/melanoma_original_ISIC_0001102.jpg_6eb2c1dd-e550-4f81-94c3-f9bb82453bfe.jpg',
 '/content/content/Skin cancer ISIC The International Skin Imaging
Collaboration/Train/melanoma/output/melanoma_original_ISIC_0010089.jpg_9d470016-fb49-46e9-ac66-7a5f27916d93.jpg',
 ...]
```

Double-click (or enter) to edit

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

```
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
 'pigmented benign keratosis',
```

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


df2 = pd.DataFrame(list(dataframe_dict_new.items()), columns=['Path', 'Label'])
new_df = pd.concat([original_df, df2], ignore_index=True)


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

Label	count
pigmented benign keratosis	962
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="/content/content/Skin cancer ISIC The International Skin Imaging Collaboration/Train"
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir_train,
  seed=123,
  validation_split = 0.2,
  subset = "training", # Todo choose the correct parameter value, so that only training data is refered to,,
  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", # Todo choose the correct parameter value, so that only validation data is refered to,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

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

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

```
## your code goes here

# Define input shape
input_shape = (img_height, img_width, 3)  # Image dimensions with 3 color channels

# Build CNN Model with Normalization
```

```
model = keras.Sequential([
    # Rescaling layer for normalization
    layers.Rescaling(1./255, input_shape=input_shape),

    # Convolutional layers
    layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(2, 2),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(2, 2),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(2, 2),

    # Flatten and Dense layers
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),  # Dropout for regularization
    layers.Dense(64, activation='relu'),

    # Output layer
    layers.Dense(len(class_names), activation='softmax')
])
```

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

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

```
## your code goes here
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Todo: Train your model

```
epochs = 30
## Your code goes here, use 50 epochs.
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)
```

```
Epoch 2/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 36s 136ms/step - accuracy: 0.3220 - loss: 1.7418 - val_accuracy: 0.4410 - val_loss: 1.4941
Epoch 3/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 41s 136ms/step - accuracy: 0.3975 - loss: 1.5705 - val_accuracy: 0.4358 - val_loss: 1.4750
Epoch 4/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 45s 162ms/step - accuracy: 0.4178 - loss: 1.4737 - val_accuracy: 0.5308 - val_loss: 1.3485
Epoch 5/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 28s 164ms/step - accuracy: 0.4534 - loss: 1.4126 - val_accuracy: 0.4699 - val_loss: 1.3871
Epoch 6/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 36s 136ms/step - accuracy: 0.4817 - loss: 1.3711 - val_accuracy: 0.5160 - val_loss: 1.3314
Epoch 7/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 42s 143ms/step - accuracy: 0.5096 - loss: 1.2760 - val_accuracy: 0.5754 - val_loss: 1.2137
Epoch 8/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 40s 136ms/step - accuracy: 0.5414 - loss: 1.2132 - val_accuracy: 0.5857 - val_loss: 1.1352
Epoch 9/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 23s 138ms/step - accuracy: 0.5500 - loss: 1.1916 - val_accuracy: 0.6147 - val_loss: 1.0849
Epoch 10/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 41s 141ms/step - accuracy: 0.5973 - loss: 1.0491 - val_accuracy: 0.5226 - val_loss: 1.3835
Epoch 11/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 28s 163ms/step - accuracy: 0.6187 - loss: 1.0179 - val_accuracy: 0.6563 - val_loss: 0.9273
Epoch 12/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 36s 134ms/step - accuracy: 0.6473 - loss: 0.9143 - val_accuracy: 0.6978 - val_loss: 0.8563
Epoch 13/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 22s 128ms/step - accuracy: 0.6585 - loss: 0.8907 - val_accuracy: 0.7283 - val_loss: 0.8[illegible]
Epoch 14/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 47s 164ms/step - accuracy: 0.6883 - loss: 0.8224 - val_accuracy: 0.7216 - val_loss: 0.8285
Epoch 15/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 36s 133ms/step - accuracy: 0.7063 - loss: 0.7462 - val_accuracy: 0.7342 - val_loss: 0.8242
Epoch 16/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 42s 142ms/step - accuracy: 0.7270 - loss: 0.7244 - val_accuracy: 0.7691 - val_loss: 0.7453
Epoch 17/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 40s 133ms/step - accuracy: 0.7435 - loss: 0.6940 - val_accuracy: 0.7647 - val_loss: 0.8011
Epoch 18/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 41s 133ms/step - accuracy: 0.7532 - loss: 0.6378 - val_accuracy: 0.7803 - val_loss: 0.7213
Epoch 19/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 46s 162ms/step - accuracy: 0.7607 - loss: 0.6187 - val_accuracy: 0.7647 - val_loss: 0.7852
Epoch 20/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 38s 141ms/step - accuracy: 0.7749 - loss: 0.5800 - val_accuracy: 0.8025 - val_loss: 0.7223
```

```
Epoch 22/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 24s 142ms/step - accuracy: 0.8051 - loss: 0.4911 - val_accuracy: 0.8018 - val_loss: 0.7568
Epoch 23/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 41s 144ms/step - accuracy: 0.8122 - loss: 0.4788 - val_accuracy: 0.8055 - val_loss: 0.7668
Epoch 24/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 39s 131ms/step - accuracy: 0.8147 - loss: 0.4728 - val_accuracy: 0.8129 - val_loss: 0.6862
Epoch 25/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 42s 136ms/step - accuracy: 0.8247 - loss: 0.4490 - val_accuracy: 0.8352 - val_loss: 0.7141
Epoch 26/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 46s 163ms/step - accuracy: 0.8357 - loss: 0.4146 - val_accuracy: 0.8137 - val_loss: 1.0204
Epoch 27/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 36s 134ms/step - accuracy: 0.8503 - loss: 0.3858 - val_accuracy: 0.8233 - val_loss: 0.7347
Epoch 28/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 40s 130ms/step - accuracy: 0.8418 - loss: 0.4177 - val_accuracy: 0.8411 - val_loss: 0.6750
Epoch 29/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 42s 134ms/step - accuracy: 0.8544 - loss: 0.3794 - val_accuracy: 0.8285 - val_loss: 0.7473
Epoch 30/30
169/169 ━━━━━━━━━━━━━━━━━━━━ 41s 135ms/step - accuracy: 0.8512 - loss: 0.3964 - val_accuracy: 0.7936 - val_loss: 0.7875
```

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()
```