Notebook

April 10, 2024

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.

0.0.1 Importing Skin Cancer Data

To do: Take necessary actions to read the data

0.0.2 Importing all the important libraries

[22]:
```
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")           ## Suppress all warnings
```

[23]:
```
# ## 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
```

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.

[24]:
```
##root_dir = pathlib.Path('C:/02 Srikanth/15 Upgrad/06 Deep Learning/CNN Cancer␣
 ↪assignment/Skin_cancer_ISIC')
root_dir = pathlib.Path('Skin_cancer_ISIC')
```

```
data_dir_train = root_dir / 'Train'
data_dir_test = root_dir / 'Test'

# Function to count the number of JPG images in a directory
def count_jpg_images(directory):
    count = 0
    for subdir in directory.iterdir():
        count += len(list(subdir.glob('*.jpg')))
    return count

# Count the number of JPG images in the train and test directories
image_count_train = count_jpg_images(data_dir_train)
image_count_test = count_jpg_images(data_dir_test)

print("Total number of JPG images in train folder:", image_count_train)
print("Total number of JPG images in test folder:", image_count_test)
```

```
Total number of JPG images in train folder: 2239
Total number of JPG images in test folder: 118
```

0.0.3 Load using keras.preprocessing — Let's load these images off disk using the helpful image_dataset_from_directory utility.

0.0.4 Create a dataset - Define some parameters for the loader:

[25]:
```
batch_size = 32
img_height = 180
img_width = 180
```

Use 80% of the images for training, and 20% for validation.

[26]:
```
## 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


##### seed=123 ensures that the shuffling process produces the same random␣
↪order of files each time
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.
```

[27]:
```python
## Write your validation dataset here
## Note use seed=123 while creating your dataset using tf.keras.preprocessing.
 ↪image_dataset_from_directory
## Note, make sure your resize your images to the size img_height*img_width,␣
 ↪while writting the dataset

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

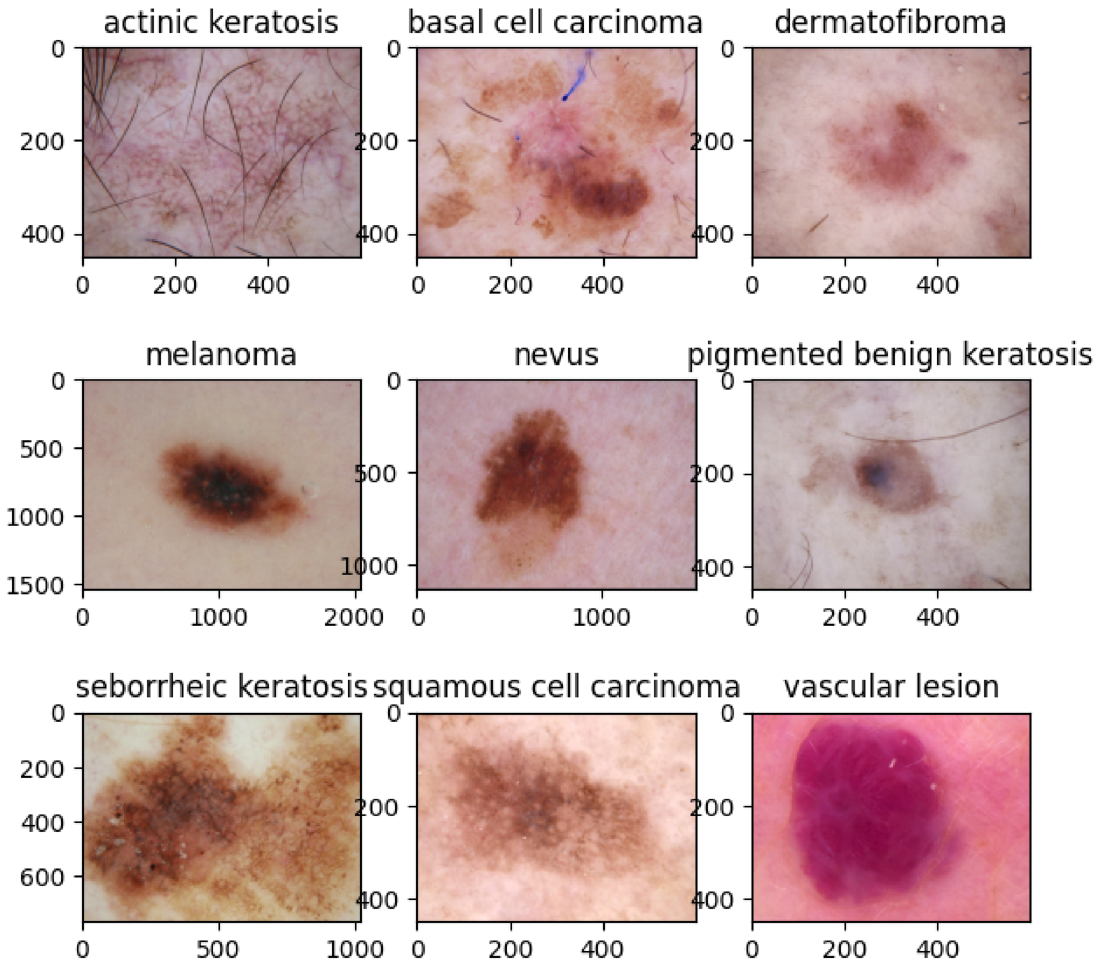
```
Found 2239 files belonging to 9 classes.
Using 447 files for validation.
```

[28]:
```python
# List out all the classes of skin cancer and store them in a list.
# You can find the class names in the class_names attribute on these datasets.
# These correspond to the directory names in alphabetical order.

class_names = train_ds.class_names
print(class_names)
```

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

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

[29]:
```python
import matplotlib.pyplot as plt
import matplotlib.image as img

plt.figure(figsize=(7,7))
for i in range(9):
  plt.subplot(3, 3, i + 1)
  image = img.imread(str(list(data_dir_train.glob(class_names[i]+'/*.jpg'))[1]))
  plt.title(class_names[i])
  plt.imshow(image)
```



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.

[30]:
```
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]`

My Notes * putting padding = 'Same' to maintain same spatial dimensions as the input image. * when padding='same': If the stride of the convolution operation is 1, the output size will be the same as the input size. * If the stride is greater than 1, the output size will be adjusted accordingly to maintain the spatial dimensions as closely as possible to the input size.

[31]:
```
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
num_classes = 9

model = Sequential([layers.experimental.preprocessing.Rescaling(1./255,␣
 ↪input_shape=(img_height, img_width,3))])
model.add(Conv2D(filters = 32, kernel_size = (5,5), padding = 'Same',␣
 ↪activation ='relu', input_shape = (180, 180, 32)))
model.add(Conv2D(filters = 32, kernel_size = (5,5), padding = 'Same',␣
 ↪activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(filters = 32, kernel_size = (5,5), padding = 'Same',␣
 ↪activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(filters = 32, kernel_size = (5,5), padding = 'Same',␣
 ↪activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(filters = 32, kernel_size = (5,5), padding = 'Same',␣
 ↪activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.30))


model.add(Flatten())
model.add(Dense(num_classes, activation = "softmax"))
```

My Notes 1. *First Conv2D layer (32 filters):* Parameters: $(5 * 5 * 3 + 1) * 32 = 2432$, each with a kernel size of (5,5), and the input shape is (180,180,3)

2. *Second Conv2D layer (32 filters):* Parameters: $(5 * 5 * 32 + 1) * 32 = 25632$, and the input shape is the same as the output shape of the previous layer.

3. *Third Conv2D layer (32 filters):* Parameters: $(5 * 5 * 32 + 1) * 32 = 25632$, and the input shape is the same as the output shape of the previous layer.

4. *Fourth Conv2D layer (32 filters):* Parameters: $(5 * 5 * 32 + 1) * 32 = 25632$, and the input shape is the same as the output shape of the previous layer.

5. *Dropout layer:* The dropout layer does not have any trainable parameters, so 0 parameters.

6. *Flatten layer:* The flatten layer does not have any trainable parameters, so 0 parameters.

7. *Dense layer* (3872 input neurons and 9 output neurons) Parameters: $(3872 + 1) * 9 = 34857$

8. *Total trainable parameters*: $2432 + 25632 + 25632 + 25632 + 34857 = 139{,}817$

0.0.5 Compile the model - Choose an appropirate optimiser and loss function for model training

Being 9 classes defined, this is categorical model example and hence using this specific loss function

[32]:
```
model.compile(optimizer='adam', loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
```

[33]:
```
# 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_5 (Conv2D)           (None, 180, 180, 32)      2432

 conv2d_6 (Conv2D)           (None, 180, 180, 32)      25632

 max_pooling2d_4 (MaxPoolin  (None, 90, 90, 32)        0
 g2D)

 conv2d_7 (Conv2D)           (None, 90, 90, 32)        25632

 max_pooling2d_5 (MaxPoolin  (None, 45, 45, 32)        0
 g2D)

 conv2d_8 (Conv2D)           (None, 45, 45, 32)        25632

 max_pooling2d_6 (MaxPoolin  (None, 22, 22, 32)        0
 g2D)

 conv2d_9 (Conv2D)           (None, 22, 22, 32)        25632

 max_pooling2d_7 (MaxPoolin  (None, 11, 11, 32)        0
 g2D)

 dropout_1 (Dropout)         (None, 11, 11, 32)        0

 flatten_1 (Flatten)         (None, 3872)              0

 dense_1 (Dense)             (None, 9)                 34857

=================================================================
Total params: 139817 (546.16 KB)
Trainable params: 139817 (546.16 KB)
Non-trainable params: 0 (0.00 Byte)
```

0.0.6 Train the model

My notes – changed the epochs to 30

[34]:
```
epochs = 30
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/30
WARNING:tensorflow:From
c:\Users\sndur\AppData\Local\Programs\Python\Python311\Lib\site-
packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue
is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From
c:\Users\sndur\AppData\Local\Programs\Python\Python311\Lib\site-
packages\keras\src\engine\base_layer_utils.py:384: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.

56/56 [==============================] - 40s 645ms/step - loss: 2.0655 -
accuracy: 0.1747 - val_loss: 2.0495 - val_accuracy: 0.2103
Epoch 2/30
56/56 [==============================] - 40s 723ms/step - loss: 2.0037 -
accuracy: 0.2271 - val_loss: 1.9616 - val_accuracy: 0.2148
Epoch 3/30
56/56 [==============================] - 44s 791ms/step - loss: 1.9367 -
accuracy: 0.2662 - val_loss: 1.9115 - val_accuracy: 0.2550
Epoch 4/30
56/56 [==============================] - 45s 812ms/step - loss: 1.8703 -
accuracy: 0.3019 - val_loss: 1.8333 - val_accuracy: 0.3266
Epoch 5/30
56/56 [==============================] - 47s 847ms/step - loss: 1.7789 -
accuracy: 0.3599 - val_loss: 1.6681 - val_accuracy: 0.4094
Epoch 6/30
56/56 [==============================] - 48s 865ms/step - loss: 1.6792 -
accuracy: 0.3867 - val_loss: 1.6559 - val_accuracy: 0.4094
Epoch 7/30
56/56 [==============================] - 51s 912ms/step - loss: 1.5867 -
accuracy: 0.4336 - val_loss: 1.6200 - val_accuracy: 0.3982
Epoch 8/30
56/56 [==============================] - 83s 1s/step - loss: 1.5220 - accuracy:
0.4475 - val_loss: 1.5010 - val_accuracy: 0.4653
Epoch 9/30
```

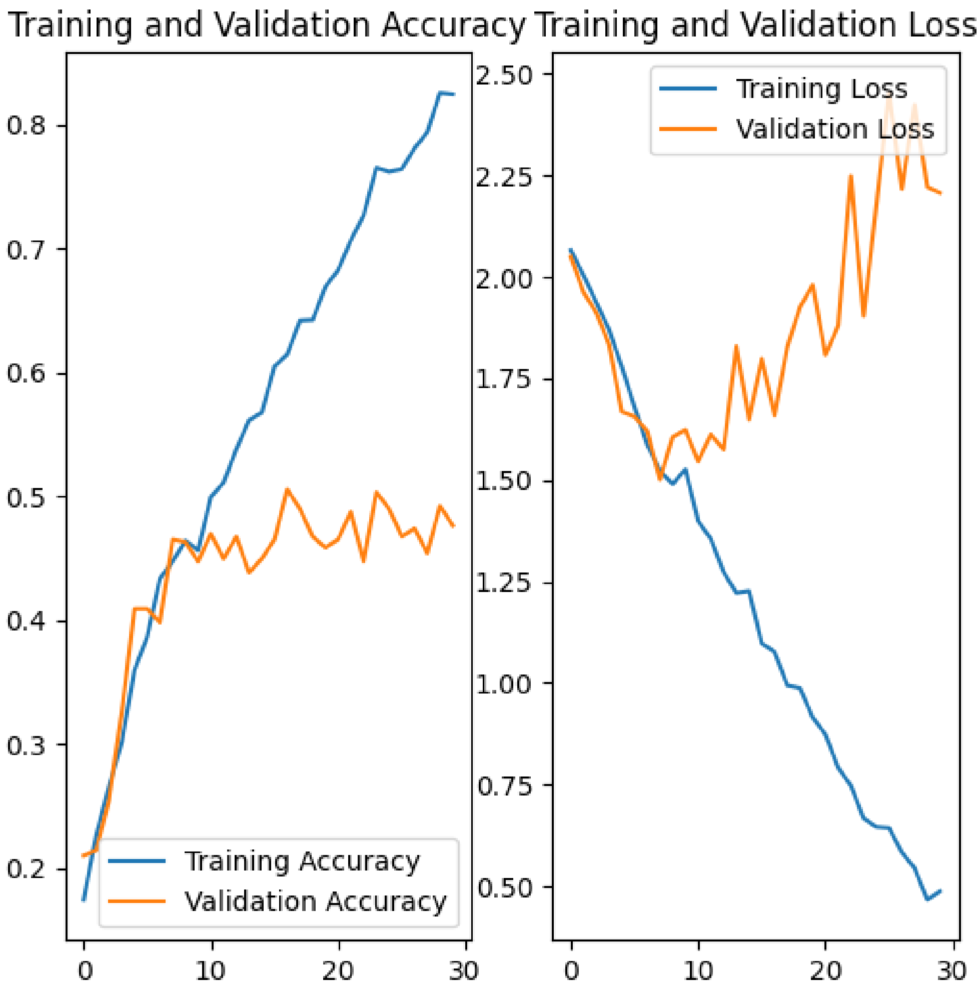
```
56/56 [==============================] - 75s 1s/step - loss: 1.4899 - accuracy:
0.4643 - val_loss: 1.6061 - val_accuracy: 0.4631
Epoch 10/30
56/56 [==============================] - 52s 932ms/step - loss: 1.5259 -
accuracy: 0.4565 - val_loss: 1.6230 - val_accuracy: 0.4474
Epoch 11/30
56/56 [==============================] - 52s 928ms/step - loss: 1.3985 -
accuracy: 0.4994 - val_loss: 1.5452 - val_accuracy: 0.4698
Epoch 12/30
56/56 [==============================] - 55s 993ms/step - loss: 1.3547 -
accuracy: 0.5112 - val_loss: 1.6122 - val_accuracy: 0.4497
Epoch 13/30
56/56 [==============================] - 95s 2s/step - loss: 1.2727 - accuracy:
0.5379 - val_loss: 1.5745 - val_accuracy: 0.4676
Epoch 14/30
56/56 [==============================] - 147s 3s/step - loss: 1.2214 - accuracy:
0.5614 - val_loss: 1.8295 - val_accuracy: 0.4385
Epoch 15/30
56/56 [==============================] - 85s 1s/step - loss: 1.2264 - accuracy:
0.5675 - val_loss: 1.6483 - val_accuracy: 0.4497
Epoch 16/30
56/56 [==============================] - 73s 1s/step - loss: 1.0973 - accuracy:
0.6049 - val_loss: 1.7985 - val_accuracy: 0.4653
Epoch 17/30
56/56 [==============================] - 52s 936ms/step - loss: 1.0765 -
accuracy: 0.6144 - val_loss: 1.6583 - val_accuracy: 0.5056
Epoch 18/30
56/56 [==============================] - 52s 929ms/step - loss: 0.9941 -
accuracy: 0.6417 - val_loss: 1.8269 - val_accuracy: 0.4899
Epoch 19/30
56/56 [==============================] - 52s 932ms/step - loss: 0.9874 -
accuracy: 0.6423 - val_loss: 1.9251 - val_accuracy: 0.4676
Epoch 20/30
56/56 [==============================] - 51s 920ms/step - loss: 0.9150 -
accuracy: 0.6691 - val_loss: 1.9800 - val_accuracy: 0.4586
Epoch 21/30
56/56 [==============================] - 49s 873ms/step - loss: 0.8730 -
accuracy: 0.6825 - val_loss: 1.8076 - val_accuracy: 0.4653
Epoch 22/30
56/56 [==============================] - 49s 878ms/step - loss: 0.7910 -
accuracy: 0.7065 - val_loss: 1.8794 - val_accuracy: 0.4877
Epoch 23/30
56/56 [==============================] - 50s 894ms/step - loss: 0.7474 -
accuracy: 0.7266 - val_loss: 2.2483 - val_accuracy: 0.4474
Epoch 24/30
56/56 [==============================] - 49s 869ms/step - loss: 0.6664 -
accuracy: 0.7651 - val_loss: 1.9036 - val_accuracy: 0.5034
Epoch 25/30
```

```
56/56 [==============================] - 50s 897ms/step - loss: 0.6454 -
accuracy: 0.7617 - val_loss: 2.1767 - val_accuracy: 0.4899
Epoch 26/30
56/56 [==============================] - 50s 902ms/step - loss: 0.6431 -
accuracy: 0.7640 - val_loss: 2.4530 - val_accuracy: 0.4676
Epoch 27/30
56/56 [==============================] - 49s 871ms/step - loss: 0.5827 -
accuracy: 0.7807 - val_loss: 2.2167 - val_accuracy: 0.4743
Epoch 28/30
56/56 [==============================] - 49s 868ms/step - loss: 0.5441 -
accuracy: 0.7935 - val_loss: 2.4235 - val_accuracy: 0.4541
Epoch 29/30
56/56 [==============================] - 49s 869ms/step - loss: 0.4658 -
accuracy: 0.8253 - val_loss: 2.2212 - val_accuracy: 0.4922
Epoch 30/30
56/56 [==============================] - 51s 914ms/step - loss: 0.4869 -
accuracy: 0.8242 - val_loss: 2.2066 - val_accuracy: 0.4765
```

It is clear that validation accuracy is less than model accuracy, a case of overfitting

0.0.7 Visualizing training results

[36]:
```
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=(6, 6))
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()
```



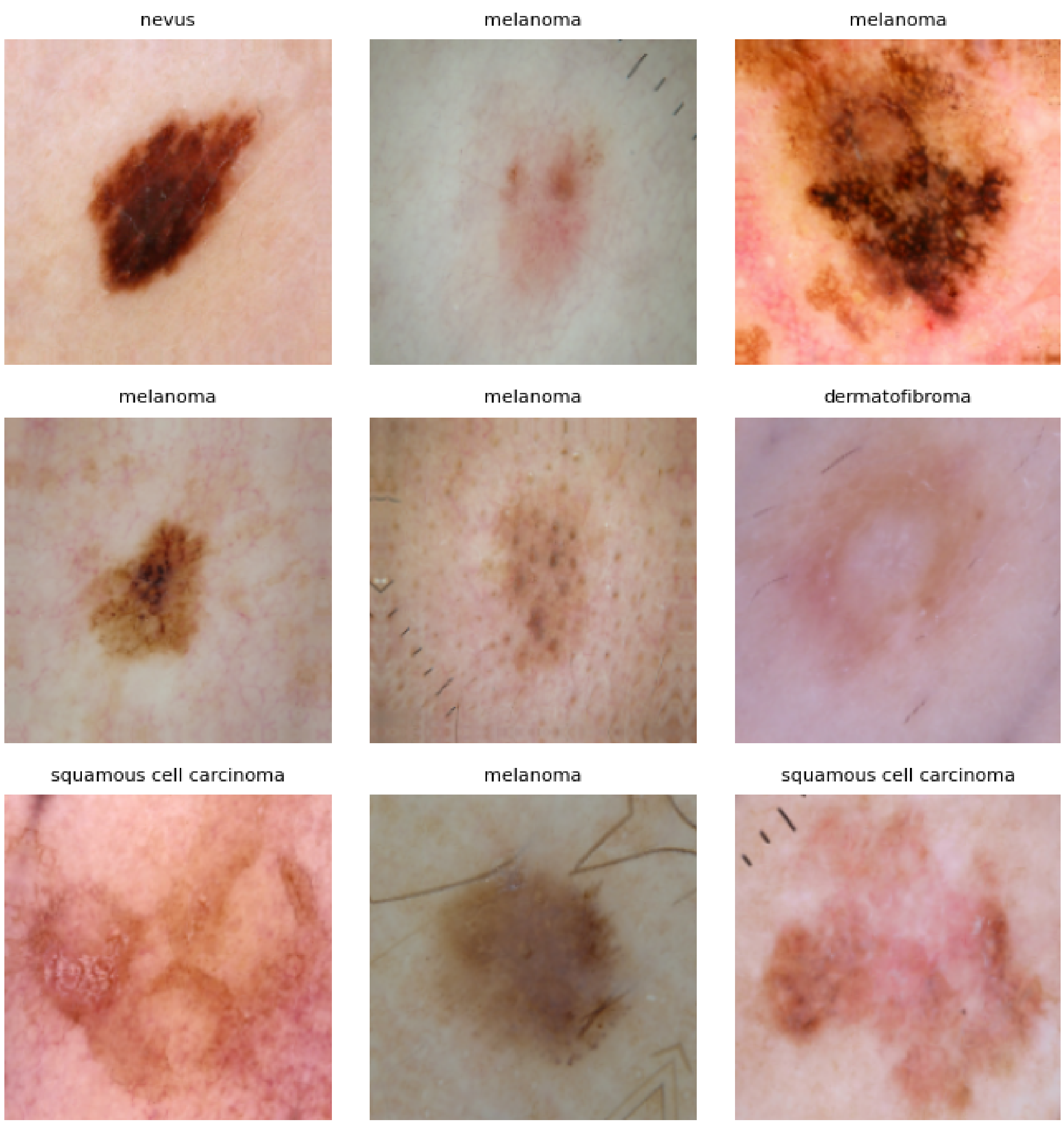
Todo: Write your findings after the model fit, see if there is an evidence of model overfit or underfit My Notes

The training accuracy is much higher than the validation accuracy, and the training loss is much lower than the validation loss, it indicates overfitting. In this case, the model has learned to fit the training data too well, but it doesn't generalize well to unseen data. ##### Hence applying data augumentation strategy

[37]:
```
# Todo, after you have analysed the model fit history for presence of underfit␣
↪or overfit,
# choose an appropriate data augumentation strategy.
data_augmentation = tf.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal"),
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1),
])
```

[40]:

```
# Todo, visualize how your augmentation strategy works for one instance of␣
↪training image.
plt.figure(figsize=(7, 7))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(data_augmentation(images)[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]], fontsize=8)  # Adjust font size to 8
        plt.axis("off")
        plt.tight_layout()  # Adjust layout to prevent overlapping
plt.show()
```



Applying model on data_augmentation

[46]:
```
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from tensorflow.keras.layers.experimental.preprocessing import Rescaling,␣
↪RandomRotation, RandomZoom

model = Sequential([
    Rescaling(1./255, input_shape=(img_height, img_width, 3)),
    data_augmentation,  # Apply data augmentation
    Conv2D(32, (5, 5), padding='same', activation='relu'),
    Conv2D(32, (5, 5), padding='same', activation='relu'),
    MaxPool2D(pool_size=(2, 2)),
    Conv2D(32, (5, 5), padding='same', activation='relu'),
    MaxPool2D(pool_size=(2, 2)),
    Conv2D(32, (5, 5), padding='same', activation='relu'),
    MaxPool2D(pool_size=(2, 2)),
    Conv2D(32, (5, 5), padding='same', activation='relu'),
    MaxPool2D(pool_size=(2, 2)),
    Dropout(0.30),
    Flatten(),
    Dense(num_classes, activation='softmax')
])
```

0.0.8 Compiling the model

[48]:
```
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'])

# Display model summary
model.summary()
```

```
Model: "sequential_4"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_4 (Rescaling)     (None, 180, 180, 3)       0

 sequential_2 (Sequential)   (None, 180, 180, 3)       0

 conv2d_17 (Conv2D)          (None, 180, 180, 32)      2432

 conv2d_18 (Conv2D)          (None, 180, 180, 32)      25632

 max_pooling2d_12 (MaxPooli  (None, 90, 90, 32)        0
 ng2D)

 conv2d_19 (Conv2D)          (None, 90, 90, 32)        25632
```

```
 max_pooling2d_13 (MaxPooli  (None, 45, 45, 32)        0
 ng2D)

 conv2d_20 (Conv2D)          (None, 45, 45, 32)        25632

 max_pooling2d_14 (MaxPooli  (None, 22, 22, 32)        0
 ng2D)

 conv2d_21 (Conv2D)          (None, 22, 22, 32)        25632

 max_pooling2d_15 (MaxPooli  (None, 11, 11, 32)        0
 ng2D)

 dropout_3 (Dropout)         (None, 11, 11, 32)        0

 flatten_3 (Flatten)         (None, 3872)              0

 dense_3 (Dense)             (None, 9)                 34857

=================================================================
Total params: 139817 (546.16 KB)
Trainable params: 139817 (546.16 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

0.0.9 Training the model

1 My notes — Code written in the above block

[49]:
```
epochs = 30
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/30
56/56 [==============================] - 37s 640ms/step - loss: 2.0634 -
accuracy: 0.2020 - val_loss: 2.0580 - val_accuracy: 0.1924
Epoch 2/30
56/56 [==============================] - 39s 702ms/step - loss: 2.0252 -
accuracy: 0.2003 - val_loss: 2.0422 - val_accuracy: 0.2058
Epoch 3/30
56/56 [==============================] - 44s 782ms/step - loss: 2.0171 -
accuracy: 0.2132 - val_loss: 2.0362 - val_accuracy: 0.1633
Epoch 4/30
56/56 [==============================] - 112s 2s/step - loss: 2.0124 - accuracy:
```

```
0.2132 - val_loss: 2.0025 - val_accuracy: 0.2371
Epoch 5/30
56/56 [==============================] - 94s 2s/step - loss: 2.0419 - accuracy:
0.2227 - val_loss: 2.0080 - val_accuracy: 0.2013
Epoch 6/30
56/56 [==============================] - 40s 720ms/step - loss: 1.9304 -
accuracy: 0.2578 - val_loss: 1.8603 - val_accuracy: 0.3423
Epoch 7/30
56/56 [==============================] - 47s 845ms/step - loss: 1.8738 -
accuracy: 0.2907 - val_loss: 1.8148 - val_accuracy: 0.2998
Epoch 8/30
56/56 [==============================] - 51s 909ms/step - loss: 1.8122 -
accuracy: 0.3326 - val_loss: 1.7895 - val_accuracy: 0.3400
Epoch 9/30
56/56 [==============================] - 51s 917ms/step - loss: 1.7784 -
accuracy: 0.3421 - val_loss: 1.7213 - val_accuracy: 0.3714
Epoch 10/30
56/56 [==============================] - 48s 865ms/step - loss: 1.8066 -
accuracy: 0.3186 - val_loss: 1.7238 - val_accuracy: 0.3535
Epoch 11/30
56/56 [==============================] - 49s 879ms/step - loss: 1.7091 -
accuracy: 0.3744 - val_loss: 1.6269 - val_accuracy: 0.3937
Epoch 12/30
56/56 [==============================] - 49s 874ms/step - loss: 1.6534 -
accuracy: 0.3945 - val_loss: 1.6089 - val_accuracy: 0.4385
Epoch 13/30
56/56 [==============================] - 50s 902ms/step - loss: 1.7021 -
accuracy: 0.3750 - val_loss: 1.6617 - val_accuracy: 0.3870
Epoch 14/30
56/56 [==============================] - 50s 892ms/step - loss: 1.5949 -
accuracy: 0.4202 - val_loss: 1.5586 - val_accuracy: 0.4519
Epoch 15/30
56/56 [==============================] - 50s 892ms/step - loss: 1.6218 -
accuracy: 0.4124 - val_loss: 2.1010 - val_accuracy: 0.2260
Epoch 16/30
56/56 [==============================] - 49s 869ms/step - loss: 1.6797 -
accuracy: 0.3912 - val_loss: 1.5312 - val_accuracy: 0.4541
Epoch 17/30
56/56 [==============================] - 52s 937ms/step - loss: 1.5523 -
accuracy: 0.4252 - val_loss: 1.5908 - val_accuracy: 0.4385
Epoch 18/30
56/56 [==============================] - 53s 953ms/step - loss: 1.5463 -
accuracy: 0.4392 - val_loss: 1.5287 - val_accuracy: 0.4653
Epoch 19/30
56/56 [==============================] - 50s 893ms/step - loss: 1.5223 -
accuracy: 0.4609 - val_loss: 1.5463 - val_accuracy: 0.4564
Epoch 20/30
56/56 [==============================] - 53s 939ms/step - loss: 1.4886 -
```

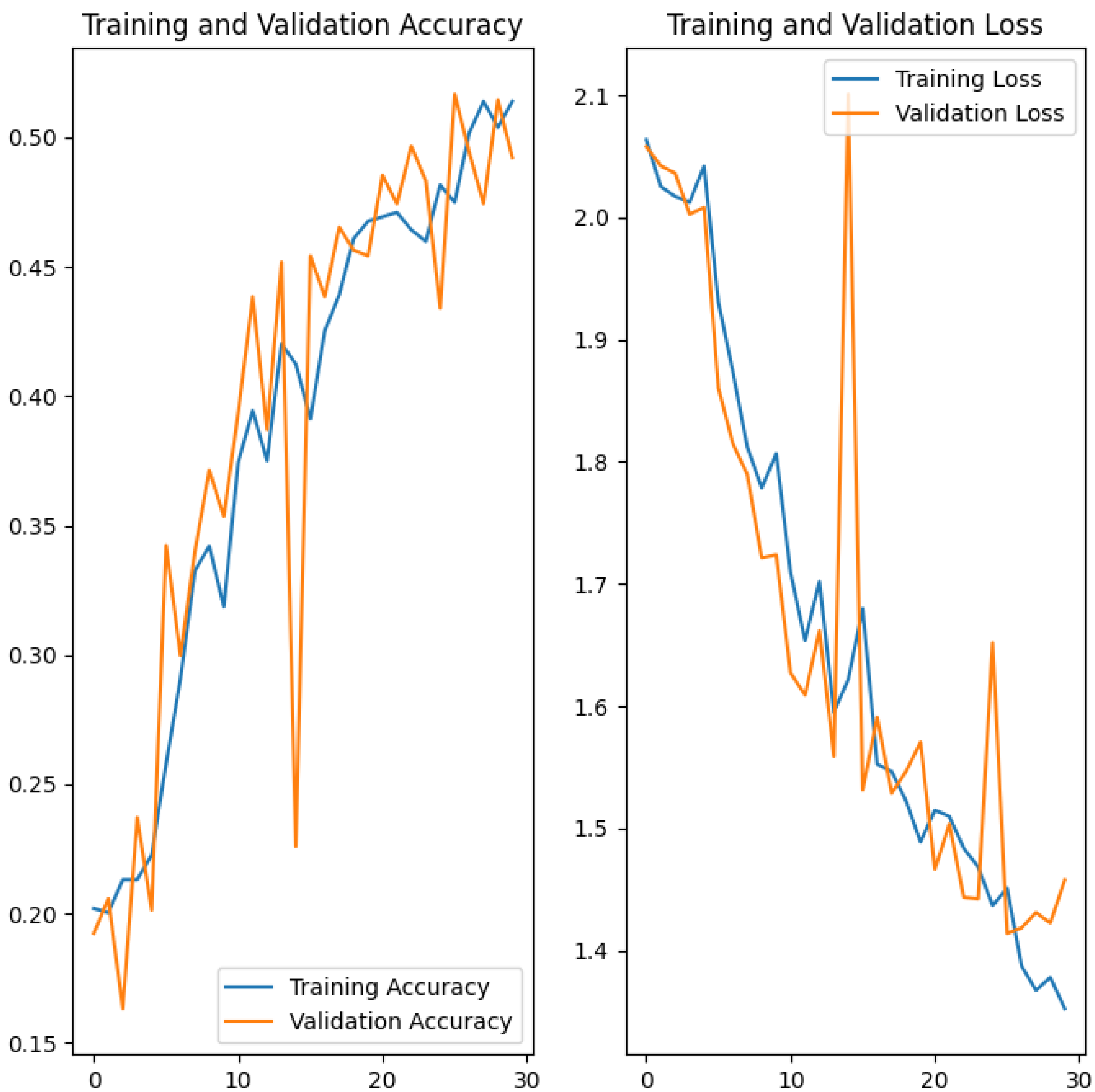
```
accuracy: 0.4676 - val_loss: 1.5703 - val_accuracy: 0.4541
Epoch 21/30
56/56 [==============================] - 54s 958ms/step - loss: 1.5145 -
accuracy: 0.4693 - val_loss: 1.4662 - val_accuracy: 0.4855
Epoch 22/30
56/56 [==============================] - 53s 945ms/step - loss: 1.5096 -
accuracy: 0.4710 - val_loss: 1.5034 - val_accuracy: 0.4743
Epoch 23/30
56/56 [==============================] - 50s 898ms/step - loss: 1.4836 -
accuracy: 0.4643 - val_loss: 1.4434 - val_accuracy: 0.4966
Epoch 24/30
56/56 [==============================] - 51s 916ms/step - loss: 1.4681 -
accuracy: 0.4598 - val_loss: 1.4420 - val_accuracy: 0.4832
Epoch 25/30
56/56 [==============================] - 101s 2s/step - loss: 1.4366 - accuracy:
0.4816 - val_loss: 1.6516 - val_accuracy: 0.4340
Epoch 26/30
56/56 [==============================] - 110s 2s/step - loss: 1.4508 - accuracy:
0.4749 - val_loss: 1.4138 - val_accuracy: 0.5168
Epoch 27/30
56/56 [==============================] - 126s 2s/step - loss: 1.3872 - accuracy:
0.5017 - val_loss: 1.4182 - val_accuracy: 0.4944
Epoch 28/30
56/56 [==============================] - 72s 1s/step - loss: 1.3672 - accuracy:
0.5140 - val_loss: 1.4309 - val_accuracy: 0.4743
Epoch 29/30
56/56 [==============================] - 57s 1s/step - loss: 1.3776 - accuracy:
0.5039 - val_loss: 1.4224 - val_accuracy: 0.5145
Epoch 30/30
56/56 [==============================] - 53s 955ms/step - loss: 1.3522 -
accuracy: 0.5140 - val_loss: 1.4576 - val_accuracy: 0.4922
```

If we see here, there is not overfitting now between training dataset accuracy and validation dataset accuracy, ran against 30 epochs

1.0.1 Visualizing the results

[50]:
```python
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: 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.

[52]:
```
import numpy as np

# Initialize a dictionary to store class counts
class_counts = {}

# Iterate through the training dataset
for images, labels in train_ds:
    # Flatten the labels tensor and convert it to a NumPy array
    labels_array = np.array(labels)
    # Count occurrences of each class label
    unique_labels, counts = np.unique(labels_array, return_counts=True)
    # Update the class counts dictionary
    for label, count in zip(unique_labels, counts):
        if label not in class_counts:
            class_counts[label] = count
        else:
            class_counts[label] += count

# Display the class counts along with class names
for label, count in class_counts.items():
    class_name = class_names[label]
    print(f"{class_name}: {count} samples")
```

```
actinic keratosis: 92 samples
basal cell carcinoma: 309 samples
dermatofibroma: 77 samples
melanoma: 352 samples
nevus: 277 samples
pigmented benign keratosis: 370 samples
seborrheic keratosis: 58 samples
squamous cell carcinoma: 142 samples
vascular lesion: 115 samples
```

1.0.2 My Notes

1. Which class has the least number of samples? seborrheic keratosis (58 samples)

2. Which classes dominate the data in terms proportionate number of samples? pigmented benign keratosis (370 samples)