selvasankari-nn

February 19, 2025

0.1 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.

[90]:
```
#import the required libraries
import pathlib
import os

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import PIL

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential,Model
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPool2D,␣
↪Dropout,BatchNormalization,Rescaling, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import␣
↪ModelCheckpoint,EarlyStopping,ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import load_img
```

[45]:
```
# Defining the path for train and test images
data_dir_train = pathlib.Path("Train/")
data_dir_test = pathlib.Path("Test/")
```

[46]:
```
# Count the number of image in Train and Test directory
# Using the glob to retrieve files/pathnames matching a specified pattern.

#Train Image count
image_count_train = len(list(data_dir_train.glob('*/*.jpg')))
print(image_count_train)
```

```
#Test Image count
image_count_test = len(list(data_dir_test.glob('*/*.jpg')))
print(image_count_test)
```

```
2239
118
```

0.1.1 Load using keras.preprocessing

Let's load these images off disk using the helpful image_dataset_from_directory utility.

0.1.2 Create a dataset

Define some parameters for the loader:

[49]:
```
batch_size = 32
img_height = 180
img_width = 180
```

[50]:
```
## 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,
  validation_split=0.2,
  subset="training",
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 14739 files belonging to 9 classes.
Using 11792 files for training.
```

[51]:
```
## 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 14739 files belonging to 9 classes.
Using 2947 files for validation.
```

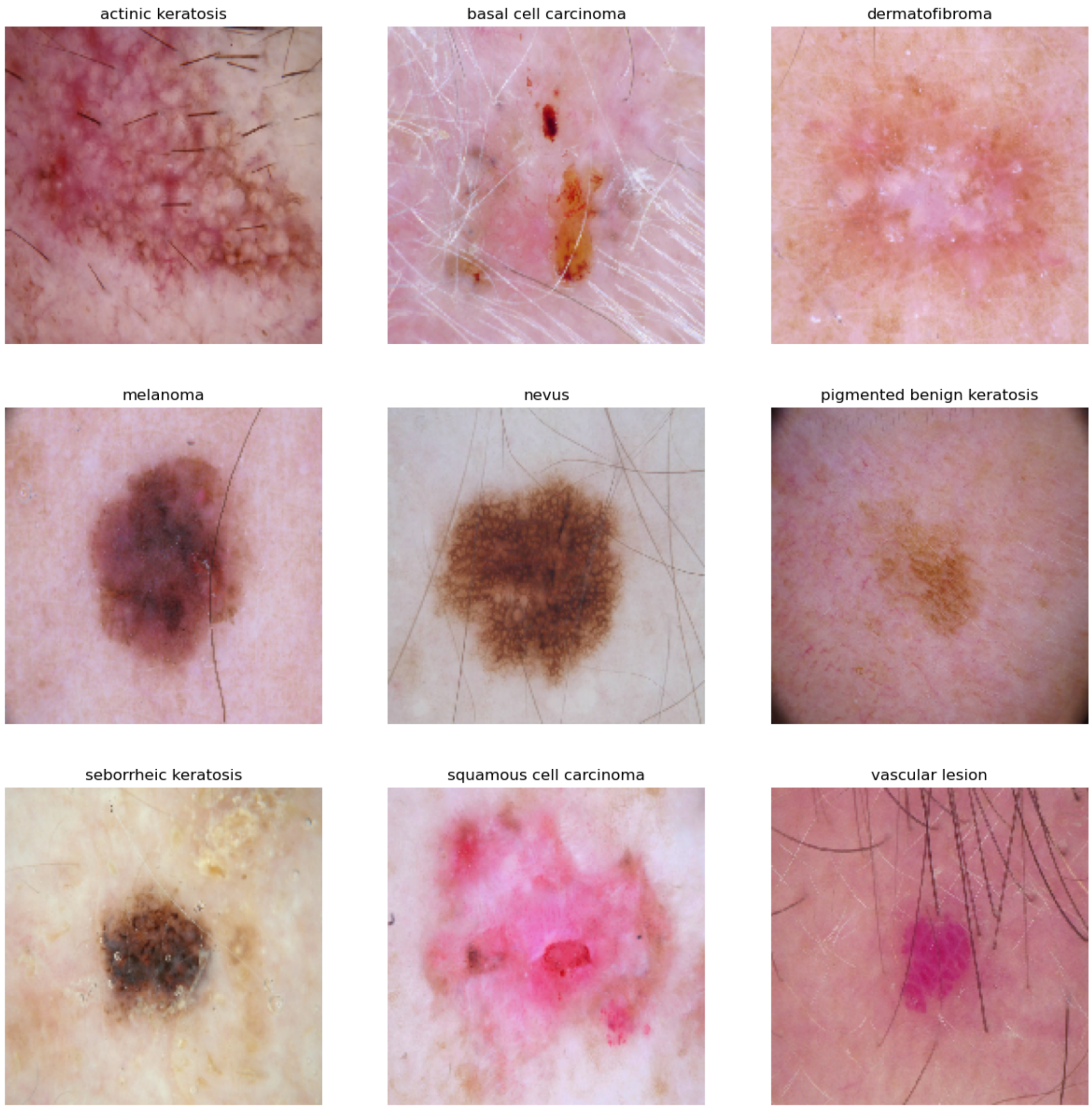
[52]:
```
# List out all the classes of skin cancer and store them in a list.
# You can find the class names in the class_names attribute on these datasets.
# These correspond to the directory names in alphabetical order.
class_names = train_ds.class_names
print(class_names)
```

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

Data Visualization

[54]:
```
#Visualize one instance of all the class present in the dataset.

#image_dataset_from_directory() will return a tf.data.Dataset that yields␣
↪batches of images from the subdirectories.
#label_mode is categorial, the labels are a float32 tensor of shape␣
↪(batch_size, num_classes), representing a one-hot encoding of the class␣
↪index.
image_dataset = tf.keras.preprocessing.
↪image_dataset_from_directory(data_dir_train,batch_size=32,image_size=(180,180),
                                                                         ␣
↪label_mode='categorical',seed=123)

#all the classes of Skin Cancer
class_names = image_dataset.class_names

#Dictionary to store the path of image as per the class
files_path_dict = {}

for c in class_names:
    files_path_dict[c] = list(map(lambda x:str(data_dir_train)+'/'+c+'/'+x,os.
↪listdir(str(data_dir_train)+'/'+c)))

#Visualize image
plt.figure(figsize=(15,15))
index = 0
for c in class_names:
    path_list = files_path_dict[c][:1]
    index += 1
    plt.subplot(3,3,index)
    plt.imshow(load_img(path_list[0],target_size=(180,180)))
    plt.title(c)
    plt.axis("off")
```

```
Found 14739 files belonging to 9 classes.
```



Visualize distribution of classes in the training dataset.

[56]:
```python
def class_distribution_count(directory):

    #count number of image in each classes
    count= []
    for path in pathlib.Path(directory).iterdir():
        if path.is_dir():
            count.append(len([name for name in os.listdir(path)
                              if os.path.isfile(os.path.join(path, name))]))
```

```
    #name of the classes
    sub_directory = [name for name in os.listdir(directory)
                     if os.path.isdir(os.path.join(directory, name))]

    #return dataframe with image count and class.
    return pd.DataFrame(list(zip(sub_directory,count)),columns =['Class', 'No.␣
↪of Image'])

df = class_distribution_count(data_dir_train)
df
```

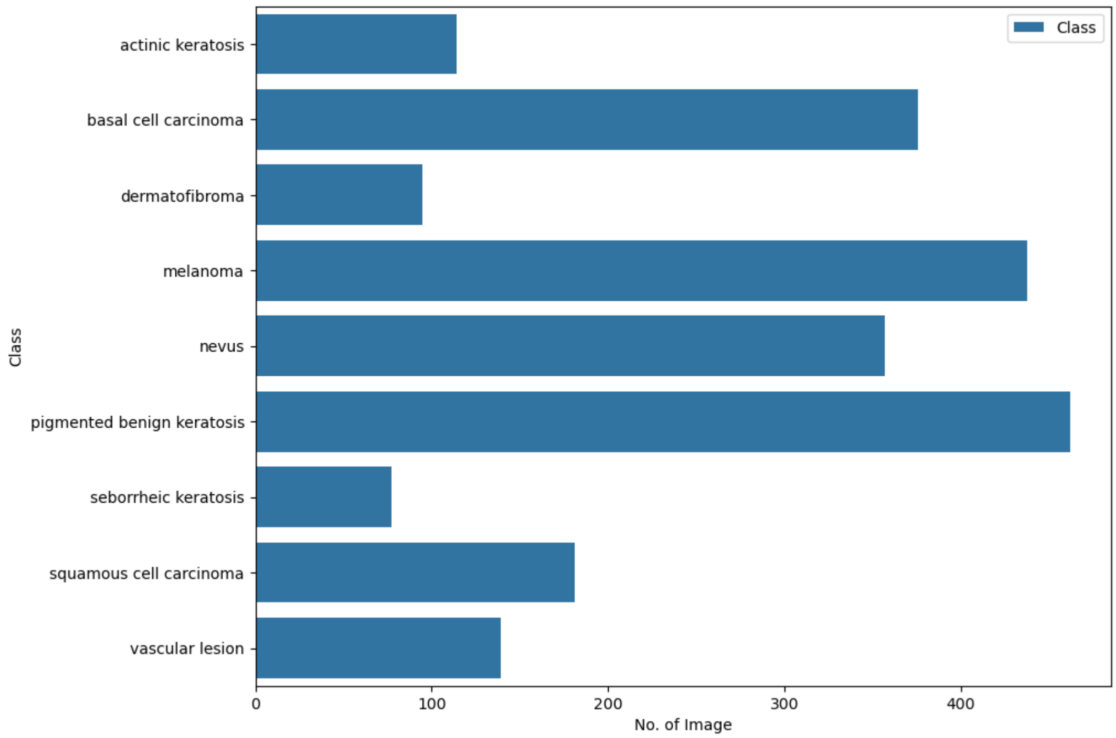
[56]:

	Class	No. of Image
0	actinic keratosis	114
1	basal cell carcinoma	376
2	dermatofibroma	95
3	melanoma	438
4	nevus	357
5	pigmented benign keratosis	462
6	seborrheic keratosis	77
7	squamous cell carcinoma	181
8	vascular lesion	139

[57]:

```
#Visualize the Number of image in each class.
import seaborn as sns
plt.figure(figsize=(10, 8))
sns.barplot(x="No. of Image", y="Class", data=df,
            label="Class")
```

[57]: <Axes: xlabel='No. of Image', ylabel='Class'>



There is a class imbalance to solve this using a python package Augmentor (https://augmentor.readthedocs.io/en/master/) to add more samples across all classes so that none of the classes have very few samples.

[59]:
```
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)
```

0.1.3 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]`

[92]:
```
model = Sequential()
#model.add(layers.Rescaling(1./255, input_shape=(img_height, img_width,3)))
model = Sequential([Input(shape=(img_height, img_width, 3)),  Rescaling(1./
 ↪255)])
model.add(Conv2D(32, kernel_size=(3, 3),padding = 'Same',activation= 'relu'))
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, kernel_size=(3, 3),padding = 'Same',activation ='relu'))
model.add(MaxPool2D(pool_size = (2, 2)))
```

```
model.add(Dropout(0.25))
model.add(Conv2D(64, kernel_size=(3, 3),padding = 'Same',activation ='relu'))
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, kernel_size=(3, 3),padding = 'Same',activation ='relu'))
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Conv2D(16, kernel_size=(7, 7),padding = 'Same',activation= 'relu'))
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, kernel_size=(11,11),padding = 'Same',activation ='relu'))
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(256, kernel_size=(3, 3),padding = 'Same',activation ='relu'))
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dense(9,activation='softmax'))
```

[98]:
```
model.summary()
```

```
Model: "sequential_14"


 Layer (type)                        Output Shape                        ␣
↪Param #

 rescaling_6 (Rescaling)             (None, 180, 180, 3)                     ␣
↪  0

 conv2d_63 (Conv2D)                  (None, 180, 180, 32)                    ␣
↪896

 max_pooling2d_63 (MaxPooling2D)     (None, 90, 90, 32)                      ␣
↪  0

 dropout_54 (Dropout)                (None, 90, 90, 32)                      ␣
↪  0

 conv2d_64 (Conv2D)                  (None, 90, 90, 64)                  ␣
↪18,496

 max_pooling2d_64 (MaxPooling2D)     (None, 45, 45, 64)                      ␣
↪  0
```

```
dropout_55 (Dropout)                (None, 45, 45, 64)                  ␣
↪  0

conv2d_65 (Conv2D)                  (None, 45, 45, 64)                ␣
↪36,928

max_pooling2d_65 (MaxPooling2D)     (None, 22, 22, 64)                  ␣
↪  0

dropout_56 (Dropout)                (None, 22, 22, 64)                  ␣
↪  0

conv2d_66 (Conv2D)                  (None, 22, 22, 64)                ␣
↪36,928

max_pooling2d_66 (MaxPooling2D)     (None, 11, 11, 64)                  ␣
↪  0

conv2d_67 (Conv2D)                  (None, 11, 11, 16)                ␣
↪50,192

max_pooling2d_67 (MaxPooling2D)     (None, 5, 5, 16)                    ␣
↪  0

dropout_57 (Dropout)                (None, 5, 5, 16)                    ␣
↪  0

conv2d_68 (Conv2D)                  (None, 5, 5, 128)                ␣
↪247,936

max_pooling2d_68 (MaxPooling2D)     (None, 2, 2, 128)                   ␣
↪  0

dropout_58 (Dropout)                (None, 2, 2, 128)                   ␣
↪  0

conv2d_69 (Conv2D)                  (None, 2, 2, 256)                ␣
↪295,168

max_pooling2d_69 (MaxPooling2D)     (None, 1, 1, 256)                   ␣
↪  0

dropout_59 (Dropout)                (None, 1, 1, 256)                   ␣
↪  0

flatten_9 (Flatten)                 (None, 256)                         ␣
↪  0
```

```
dense_14 (Dense)                    (None, 128)                    ␣
↪32,896

dense_15 (Dense)                    (None, 9)                      ␣
↪1,161
```

Total params: 720,601 (2.75 MB)

Trainable params: 720,601 (2.75 MB)

Non-trainable params: 0 (0.00 B)

[100]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

[102]:
```
%%time
epochs = 20
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

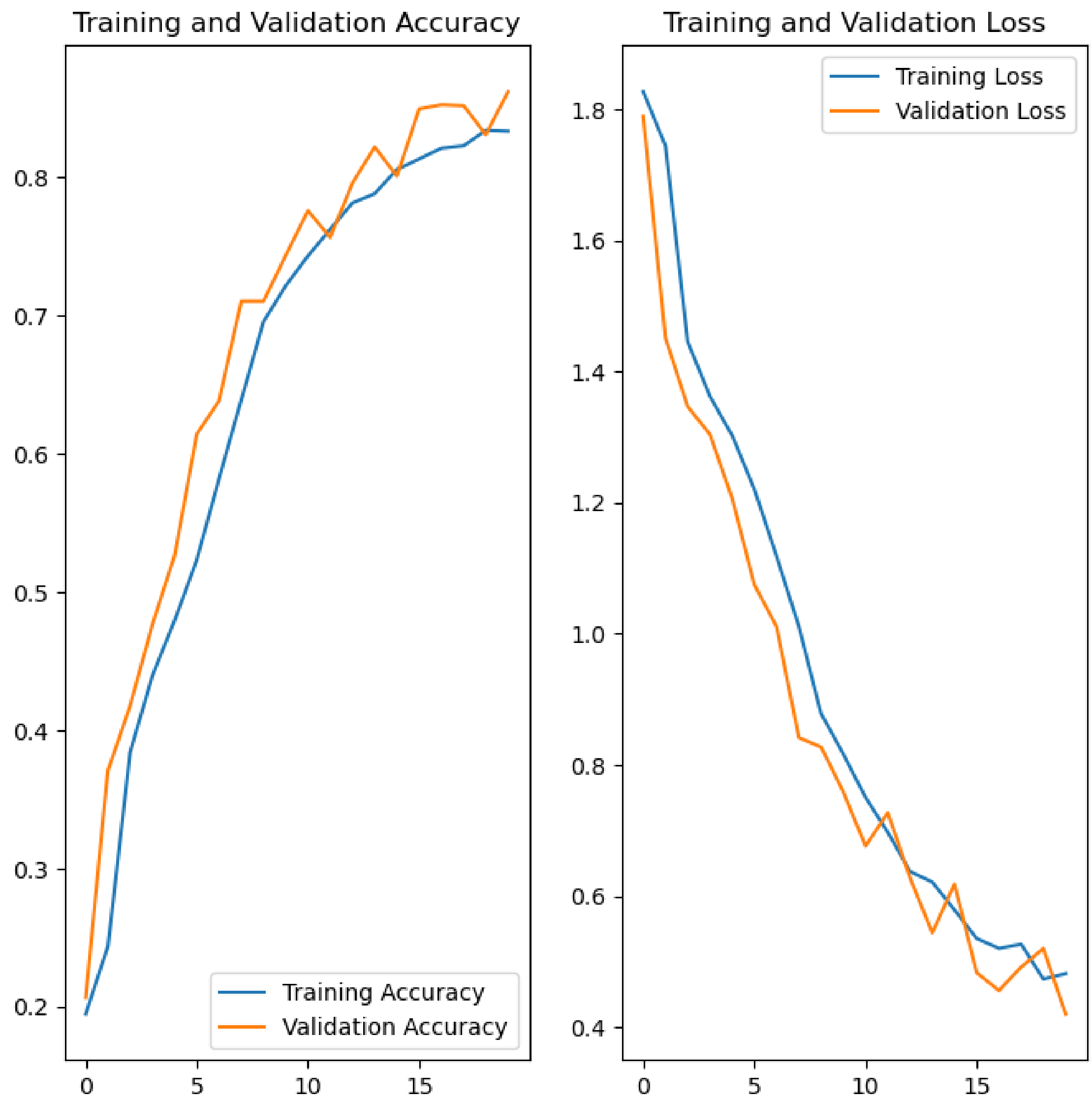
```
Epoch 1/20

C:\Users\ci381f\DevopsSetupPrograms\Anaconda\Lib\site-
packages\keras\src\backend\tensorflow\nn.py:708: UserWarning:
"`sparse_categorical_crossentropy` received `from_logits=True`, but the `output`
argument was produced by a Softmax activation and thus does not represent
logits. Was this intended?
  output, from_logits = _get_logits(

369/369             225s 543ms/step -
accuracy: 0.1929 - loss: 1.8429 - val_accuracy: 0.2067 - val_loss: 1.7889
Epoch 2/20
369/369             104s 280ms/step -
accuracy: 0.2220 - loss: 1.7858 - val_accuracy: 0.3709 - val_loss: 1.4511
Epoch 3/20
369/369             97s 262ms/step -
accuracy: 0.3696 - loss: 1.4784 - val_accuracy: 0.4177 - val_loss: 1.3466
Epoch 4/20
369/369             92s 248ms/step -
```

```
accuracy: 0.4268 - loss: 1.3858 - val_accuracy: 0.4764 - val_loss: 1.3042
Epoch 5/20
369/369            97s 263ms/step -
accuracy: 0.4841 - loss: 1.3052 - val_accuracy: 0.5263 - val_loss: 1.2067
Epoch 6/20
369/369            96s 260ms/step -
accuracy: 0.5127 - loss: 1.2407 - val_accuracy: 0.6142 - val_loss: 1.0749
Epoch 7/20
369/369            98s 265ms/step -
accuracy: 0.5878 - loss: 1.1131 - val_accuracy: 0.6383 - val_loss: 1.0107
Epoch 8/20
369/369            94s 254ms/step -
accuracy: 0.6307 - loss: 1.0196 - val_accuracy: 0.7102 - val_loss: 0.8418
Epoch 9/20
369/369            88s 240ms/step -
accuracy: 0.6820 - loss: 0.9102 - val_accuracy: 0.7102 - val_loss: 0.8275
Epoch 10/20
369/369            88s 238ms/step -
accuracy: 0.7201 - loss: 0.8096 - val_accuracy: 0.7435 - val_loss: 0.7590
Epoch 11/20
369/369            97s 264ms/step -
accuracy: 0.7386 - loss: 0.7732 - val_accuracy: 0.7757 - val_loss: 0.6764
Epoch 12/20
369/369            98s 267ms/step -
accuracy: 0.7609 - loss: 0.6852 - val_accuracy: 0.7564 - val_loss: 0.7272
Epoch 13/20
369/369            97s 264ms/step -
accuracy: 0.7765 - loss: 0.6472 - val_accuracy: 0.7957 - val_loss: 0.6284
Epoch 14/20
369/369            101s 274ms/step -
accuracy: 0.7829 - loss: 0.6371 - val_accuracy: 0.8219 - val_loss: 0.5442
Epoch 15/20
369/369            99s 268ms/step -
accuracy: 0.7991 - loss: 0.6011 - val_accuracy: 0.8008 - val_loss: 0.6184
Epoch 16/20
369/369            98s 265ms/step -
accuracy: 0.8119 - loss: 0.5381 - val_accuracy: 0.8493 - val_loss: 0.4835
Epoch 17/20
369/369            92s 248ms/step -
accuracy: 0.8172 - loss: 0.5229 - val_accuracy: 0.8524 - val_loss: 0.4562
Epoch 18/20
369/369            89s 242ms/step -
accuracy: 0.8257 - loss: 0.5100 - val_accuracy: 0.8517 - val_loss: 0.4920
Epoch 19/20
369/369            89s 242ms/step -
accuracy: 0.8386 - loss: 0.4670 - val_accuracy: 0.8307 - val_loss: 0.5203
Epoch 20/20
369/369            89s 240ms/step -
```

```
accuracy: 0.8301 - loss: 0.4929 - val_accuracy: 0.8619 - val_loss: 0.4204
CPU times: total: 4h 58min 31s
Wall time: 33min 47s
```

0.1.4 Visualizing the Training Results

[105]:
```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()
```



0.1.5 Observations:

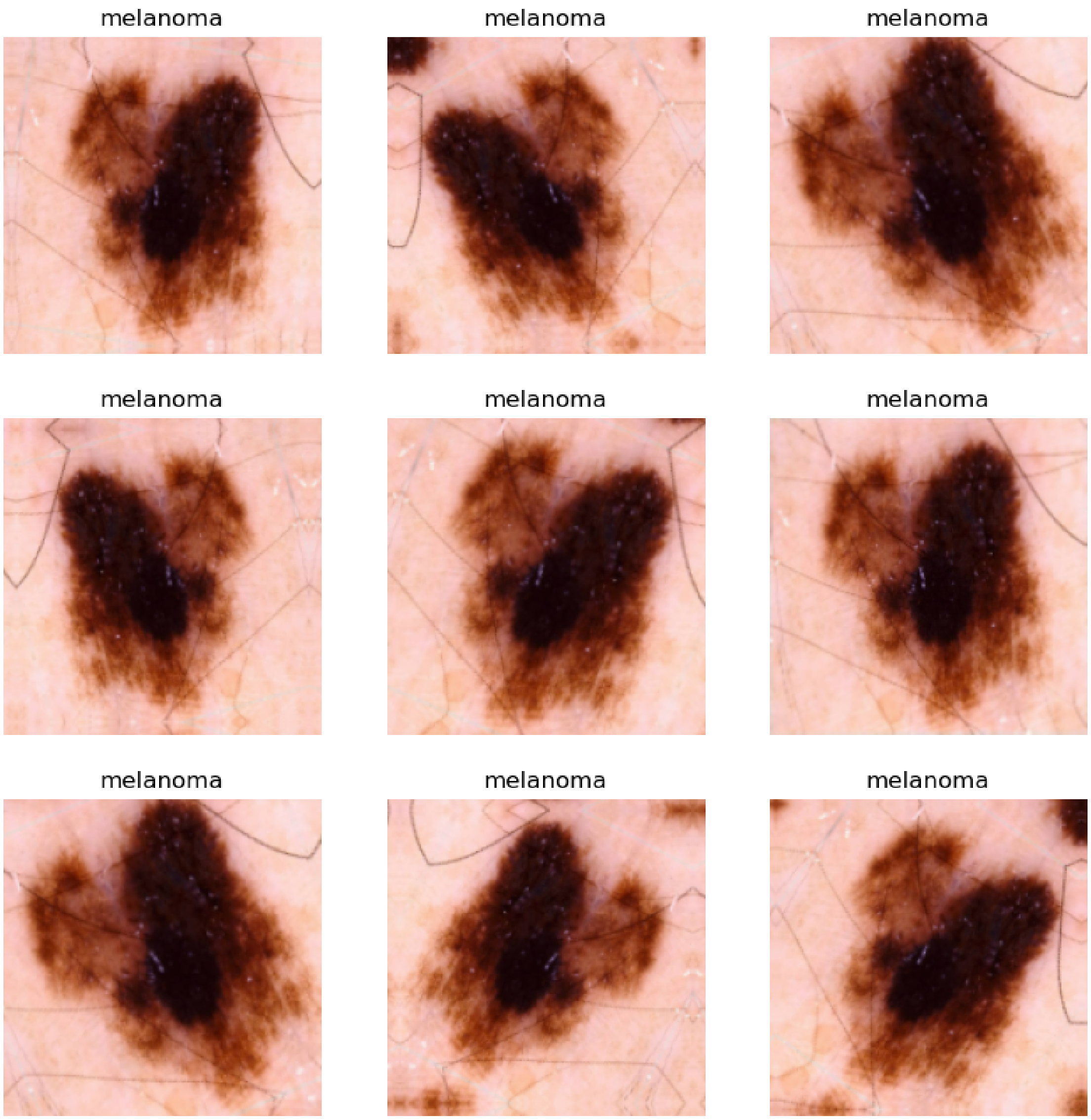
- The model's training accuracy steadily increases to 90%, while validation accuracy remains consistently around 55%.
- The high training accuracy suggests that the model has learned patterns from the training data effectively. However, its poor performance on validation data indicates a lack of generalization, meaning the model is overfitting to the training set.
- To mitigate overfitting, data augmentation techniques will be applied. Given the limited training data, new samples will be generated by introducing slight modifications to existing images, such as horizontal and vertical flips, minor rotations, and other transformations. These augmented images will enhance model robustness and improve its ability to generalize to unseen data.

0.2 After analyzing the model's fit history for signs of underfitting or overfitting, choose an appropriate data augmentation strategy.

[135]:
```
augmentation_data = keras.Sequential([
    layers.InputLayer(shape=(img_height, img_width, 3)),  # Specify input shape␣
 ↪first
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
])
```

0.2.1 Visualize how your data augmentation strategy applies to a single instance of a training image

[137]:
```
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        augmented_images = augmentation_data(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.title(class_names[labels[0]])
        plt.axis("off")
```



0.2.2 Compile the model

Choose an appropirate optimiser and loss function for model training

[143]:
```
target_labels = 9
model = Sequential([
  augmentation_data,
  layers.Rescaling(1./255),
  layers.Conv2D(16, (3, 3), padding='same', activation=tf.nn.relu),
  layers.MaxPooling2D(),
  layers.Conv2D(32, (3, 3), padding='same', activation=tf.nn.relu),
  layers.MaxPooling2D(),
```

```
  layers.Conv2D(64, (3, 3), padding='same', activation=tf.nn.relu),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation=tf.nn.relu),
  layers.Dense(target_labels)
])
```

[145]:
```
model.summary()
```

Model: "sequential_23"

```
 Layer (type)                           Output Shape                          ␣
↪Param #

 sequential_21 (Sequential)             (None, 180, 180, 3)                     ␣
↪   0

 rescaling_11 (Rescaling)               (None, 180, 180, 3)                     ␣
↪   0

 conv2d_82 (Conv2D)                     (None, 180, 180, 16)                    ␣
↪448

 max_pooling2d_82 (MaxPooling2D)        (None, 90, 90, 16)                      ␣
↪   0

 conv2d_83 (Conv2D)                     (None, 90, 90, 32)                    ␣
↪4,640

 max_pooling2d_83 (MaxPooling2D)        (None, 45, 45, 32)                      ␣
↪   0

 conv2d_84 (Conv2D)                     (None, 45, 45, 64)                   ␣
↪18,496

 max_pooling2d_84 (MaxPooling2D)        (None, 22, 22, 64)                      ␣
↪   0

 dropout_64 (Dropout)                   (None, 22, 22, 64)                      ␣
↪   0

 flatten_14 (Flatten)                   (None, 30976)                           ␣
↪   0
```

```
 dense_21 (Dense)                     (None, 128)                    ␣
↪3,965,056

 dense_22 (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)
```

[11]:
```
#Count total number of image generated by Augmentor.
image_count_train = len(list(data_dir_train.glob('*/output/*.jpg')))
print(image_count_train)
```

4500

Model Building

[147]:
```
from glob import glob

## find the image path for all class labels (lesions)
images_path_list = [ i for i in glob(os.path.join(data_dir_train, '*', '*.
↪jpg')) ]

lesions_list = [ os.path.basename(os.path.dirname(j)) for j in glob(os.path.
↪join(data_dir_train, '*', '*.jpg')) ]
print(len(lesions_list))
```

2239

[149]:
```
# Extract image path and class label in a dictionary
image_dict = dict(zip(images_path_list, lesions_list))
print(list(image_dict.items())[:5])
```

```
[('Train\\actinic keratosis\\ISIC_0025780.jpg', 'actinic keratosis'),
('Train\\actinic keratosis\\ISIC_0025803.jpg', 'actinic keratosis'),
('Train\\actinic keratosis\\ISIC_0025825.jpg', 'actinic keratosis'),
('Train\\actinic keratosis\\ISIC_0025953.jpg', 'actinic keratosis'),
('Train\\actinic keratosis\\ISIC_0025957.jpg', 'actinic keratosis')]
```

[151]:
```
# View the image paths and corresponding class labels in a DataFrame
lesions_df = pd.DataFrame(list(image_dict.items()), columns=['Image Path',␣
↪'Label'])
```

```python
lesions_df.head()
```

[151]:
```
                                 Image Path               Label
0  Train\actinic keratosis\ISIC_0025780.jpg  actinic keratosis
1  Train\actinic keratosis\ISIC_0025803.jpg  actinic keratosis
2  Train\actinic keratosis\ISIC_0025825.jpg  actinic keratosis
3  Train\actinic keratosis\ISIC_0025953.jpg  actinic keratosis
4  Train\actinic keratosis\ISIC_0025957.jpg  actinic keratosis
```

[153]:
```python
## Inspecting the class distribution in the dataset
lesions_df[['Label']].value_counts()
```

[153]:
```
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
```

[155]:
```python
round(lesions_df[['Label']].value_counts(normalize=True)*100, 2)
```

[155]:
```
Label
pigmented benign keratosis    20.63
melanoma                      19.56
basal cell carcinoma          16.79
nevus                         15.94
squamous cell carcinoma        8.08
vascular lesion                6.21
actinic keratosis              5.09
dermatofibroma                 4.24
seborrheic keratosis           3.44
Name: proportion, dtype: float64
```

0.2.3 Observations:

- A clear class imbalance is observed in the training data.
- The class **"seborrheic keratosis"** constitutes the smallest proportion of samples, making up approximately **3.44%**.
- In contrast, the classes **"pigmented benign keratosis"** and **"melanoma"** dominate the dataset, representing approximately **20.63%** and **19.56%** of the data, respectively.

[158]:
```python
path_to_training_dataset = str(data_dir_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)
```

```
Initialised with 114 image(s) found.
Output directory set to Train/actinic keratosis\output.

Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=600x450 at
0x1EA4F2B9430>: 100%| | 500/500 [00:01<00:

Initialised with 376 image(s) found.
Output directory set to Train/basal cell carcinoma\output.

Processing <PIL.Image.Image image mode=RGB size=600x450 at 0x1EA598602F0>:
100%| | 500/500 [00:01<00:00, 366.07 Samples

Initialised with 95 image(s) found.
Output directory set to Train/dermatofibroma\output.

Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=600x450 at
0x1EC41A6ABD0>: 100%| | 500/500 [00:01<00:

Initialised with 438 image(s) found.
Output directory set to Train/melanoma\output.

Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=1024x768 at
0x1EA4F269AF0>: 100%| | 500/500 [00:06<00

Initialised with 357 image(s) found.
Output directory set to Train/nevus\output.

Processing <PIL.Image.Image image mode=RGB size=2725x2082 at 0x1EA4B0C8560>:
100%| | 500/500 [00:06<00:00, 80.64 Sample

Initialised with 462 image(s) found.
Output directory set to Train/pigmented benign keratosis\output.

Executing Pipeline:   0%|
| 0/500 [00:00<?, ? Samples/s]
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~\DevopsSetupPrograms\Anaconda\Lib\site-packages\Augmentor\Pipeline.py:251
  ↪in Pipeline._execute(self, augmentor_image, save_to_disk, multi_threaded)
    243     save_name = augmentor_image.class_label \
    244                 + "_original_" \
    245                 + os.path.basename(augmentor_image.image_path) \
   (…)
```

```
    248                 + "." \
    249                 + (self.save_format if self.save_format else␣
↪augmentor_image.file_format)
--> 251     images[i].save(os.path.join(augmentor_image.output_directory,␣
↪save_name))
    253 else:

File ~\DevopsSetupPrograms\Anaconda\Lib\site-packages\PIL\Image.py:2456, in␣
↪Image.save(self, fp, format, **params)
   2455     else:
-> 2456         fp = builtins.open(filename, "w+b")
   2458 try:

FileNotFoundError: [Errno 2] No such file or directory: 'C:
↪\\Users\\ci381f\\Selva\\AIML_PG\\AIML\\DeepLearning\\Assignment\\CNN_assignment\\Skin␣
↪cancer ISIC The International Skin Imaging Collaboration\\Train\\pigmented␣
↪benign keratosis\\output\\pigmented benign keratosis_original_ISIC_0026174.
↪jpg_ce3a4085-c0b0-4f74-ba5b-75625aded7c2.jpg'

During handling of the above exception, another exception occurred:

AttributeError                            Traceback (most recent call last)
Cell In[158], line 8
      6 p = Augmentor.Pipeline(path_to_training_dataset + i)
      7 p.rotate(probability=0.7, max_left_rotation=10, max_right_rotation=10)
----> 8 p.sample(500)

File ~\DevopsSetupPrograms\Anaconda\Lib\site-packages\Augmentor\Pipeline.py:364␣
↪in Pipeline.sample(self, n, multi_threaded)
    362 with tqdm(total=len(augmentor_images), desc="Executing Pipeline", unit=␣
↪Samples") as progress_bar:
    363     with ThreadPoolExecutor(max_workers=None) as executor:
--> 364         for result in executor.map(self, augmentor_images):
    365             progress_bar.set_description("Processing %s" % result)
    366             progress_bar.update(1)

File ~\DevopsSetupPrograms\Anaconda\Lib\concurrent\futures\_base.py:619, in␣
↪Executor.map.<locals>.result_iterator()
    616 while fs:
    617     # Careful not to keep a reference to the popped future
    618     if timeout is None:
--> 619         yield _result_or_cancel(fs.pop())
    620     else:
    621         yield _result_or_cancel(fs.pop(), end_time - time.monotonic())

File ~\DevopsSetupPrograms\Anaconda\Lib\concurrent\futures\_base.py:317, in␣
↪_result_or_cancel(***failed resolving arguments***)
    315 try:
```

```
    316     try:
--> 317         return fut.result(timeout)
    318     finally:
    319         fut.cancel()

File ~\DevopsSetupPrograms\Anaconda\Lib\concurrent\futures\_base.py:456, in␣
↪Future.result(self, timeout)
    454     raise CancelledError()
    455 elif self._state == FINISHED:
--> 456     return self.__get_result()
    457 else:
    458     raise TimeoutError()

File ~\DevopsSetupPrograms\Anaconda\Lib\concurrent\futures\_base.py:401, in␣
↪Future.__get_result(self)
    399 if self._exception:
    400     try:
--> 401         raise self._exception
    402     finally:
    403         # Break a reference cycle with the exception in self._exception
    404         self = None

File ~\DevopsSetupPrograms\Anaconda\Lib\concurrent\futures\thread.py:58, in␣
↪_WorkItem.run(self)
     55     return
     57 try:
---> 58     result = self.fn(*self.args, **self.kwargs)
     59 except BaseException as exc:
     60     self.future.set_exception(exc)

File ~\DevopsSetupPrograms\Anaconda\Lib\site-packages\Augmentor\Pipeline.py:105 ␣
↪in Pipeline.__call__(self, augmentor_image)
     92 def __call__(self, augmentor_image):
     93     """
     94     Function used by the ThreadPoolExecutor to process the pipeline
     95     using multiple threads. Do not call directly.
   (…)
    103     :return: None
    104     """
--> 105     return self._execute(augmentor_image)

File ~\DevopsSetupPrograms\Anaconda\Lib\site-packages\Augmentor\Pipeline.py:268 ␣
↪in Pipeline._execute(self, augmentor_image, save_to_disk, multi_threaded)
    265             images[i].save(os.path.join(augmentor_image.
↪output_directory, save_name))
    267 except IOError as e:
--> 268     print("Error writing %s, %s. Change save_format to PNG?" %␣
↪(file_name, e.message))
```

```
    269     print("You can change the save format using the␣
↪set_save_format(save_format) function.")
    270     print("By passing save_format=\"auto\", Augmentor can save in the␣
↪correct format automatically.")

AttributeError: 'FileNotFoundError' object has no attribute 'message'
```

[160]:
```
# Verifying the total count of images after the augmentation
image_count_train = len(list(data_dir_train.glob('*/output/*.jpg')))
print(image_count_train)
```

```
17500
```

[162]:
```
# extracting the augmented image paths in a list
path_list_new = [x for x in glob(os.path.join(data_dir_train, '*','output', '*.
↪jpg'))]
path_list_new[:5]
```

[162]:
```
['Train\\actinic keratosis\\output\\actinic
keratosis_original_ISIC_0025780.jpg_038d9efd-ed5c-4cc7-8e74-59a48c289472.jpg',
 'Train\\actinic keratosis\\output\\actinic
keratosis_original_ISIC_0025780.jpg_088c78c4-8ab9-4a2f-95bc-0dacf6f47ac3.jpg',
 'Train\\actinic keratosis\\output\\actinic
keratosis_original_ISIC_0025780.jpg_0b396c97-c996-4ea9-bbce-51ee7a64feac.jpg',
 'Train\\actinic keratosis\\output\\actinic
keratosis_original_ISIC_0025780.jpg_16357d54-f5e4-40d2-9130-818132a3e33f.jpg',
 'Train\\actinic keratosis\\output\\actinic
keratosis_original_ISIC_0025780.jpg_1893db8d-19fa-440c-9219-9b741ed4f220.jpg']
```

[164]:
```
lesion_list_new = [os.path.basename(os.path.dirname(os.path.dirname(y))) for y␣
↪in glob(os.path.join(data_dir_train, '*','output', '*.jpg'))]
lesion_list_new[:5]
```

[164]:
```
['actinic keratosis',
 'actinic keratosis',
 'actinic keratosis',
 'actinic keratosis',
 'actinic keratosis']
```

[166]:
```
dataframe_dict_new = dict(zip(path_list_new, lesion_list_new))
```

[168]:
```
df2 = pd.DataFrame(list(dataframe_dict_new.items()),columns = ['Image␣
↪Path','Label'])
new_df = pd.concat([lesions_df, df2], ignore_index=True)
new_df.shape
```

[168]:
```
(19739, 2)
```

[170]:
```
new_df.head()
```

[170]:
```
                              Image Path              Label
0  Train\actinic keratosis\ISIC_0025780.jpg  actinic keratosis
1  Train\actinic keratosis\ISIC_0025803.jpg  actinic keratosis
2  Train\actinic keratosis\ISIC_0025825.jpg  actinic keratosis
3  Train\actinic keratosis\ISIC_0025953.jpg  actinic keratosis
4  Train\actinic keratosis\ISIC_0025957.jpg  actinic keratosis
```

[172]:
```
# Inspecting the classes after adding 500 samples per label
new_df['Label'].value_counts()
```

[172]:
```
Label
melanoma                      3938
basal cell carcinoma          3876
nevus                         3857
actinic keratosis             3614
dermatofibroma                3595
pigmented benign keratosis     462
squamous cell carcinoma        181
vascular lesion                139
seborrheic keratosis            77
Name: count, dtype: int64
```

[174]:
```
# Inspecting the classes (% age wise) after adding 500 samples per label
round(new_df['Label'].value_counts(normalize=True)*100, 2)
```

[174]:
```
Label
melanoma                      19.95
basal cell carcinoma          19.64
nevus                         19.54
actinic keratosis             18.31
dermatofibroma                18.21
pigmented benign keratosis     2.34
squamous cell carcinoma        0.92
vascular lesion                0.70
seborrheic keratosis           0.39
Name: proportion, dtype: float64
```

0.2.4 Train the model on the data created using Augmentor

[177]:
```
batch_size = 32
img_height = 180
img_width = 180
```

[179]:
```
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 19739 files belonging to 9 classes.
Using 15792 files for training.
```

[185]:
```
model = Sequential([
  augmentation_data,
  layers.Rescaling(1./255),
  layers.Conv2D(16, (3, 3), padding='same', activation=tf.nn.relu),
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Conv2D(32, (3, 3), padding='same', activation=tf.nn.relu),
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Conv2D(64, (3, 3), padding='same', activation=tf.nn.relu),
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation=tf.nn.relu),
  layers.Dense(target_labels)
])
```

[187]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

[189]:
```
%%time
epochs = 20
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/20
494/494              88s 173ms/step -
accuracy: 0.5009 - loss: 2.7371 - val_accuracy: 0.4913 - val_loss: 1.5643
Epoch 2/20
494/494              103s 209ms/step -
```

```
accuracy: 0.6599 - loss: 0.9769 - val_accuracy: 0.7296 - val_loss: 0.7642
Epoch 3/20
494/494             108s 219ms/step -
accuracy: 0.7220 - loss: 0.8145 - val_accuracy: 0.7126 - val_loss: 0.8088
Epoch 4/20
494/494             109s 221ms/step -
accuracy: 0.7635 - loss: 0.7137 - val_accuracy: 0.6518 - val_loss: 1.0071
Epoch 5/20
494/494             112s 227ms/step -
accuracy: 0.7846 - loss: 0.6507 - val_accuracy: 0.7754 - val_loss: 0.6481
Epoch 6/20
494/494             109s 220ms/step -
accuracy: 0.8017 - loss: 0.5998 - val_accuracy: 0.7011 - val_loss: 0.9134
Epoch 7/20
494/494             108s 219ms/step -
accuracy: 0.8149 - loss: 0.5475 - val_accuracy: 0.6566 - val_loss: 1.1859
Epoch 8/20
494/494             115s 233ms/step -
accuracy: 0.8190 - loss: 0.5289 - val_accuracy: 0.8076 - val_loss: 0.5697
Epoch 9/20
494/494             107s 217ms/step -
accuracy: 0.8354 - loss: 0.4990 - val_accuracy: 0.7625 - val_loss: 0.7254
Epoch 10/20
494/494             103s 209ms/step -
accuracy: 0.8527 - loss: 0.4558 - val_accuracy: 0.7855 - val_loss: 0.7865
Epoch 11/20
494/494             89s 180ms/step -
accuracy: 0.8574 - loss: 0.4396 - val_accuracy: 0.7838 - val_loss: 0.6680
Epoch 12/20
494/494             84s 170ms/step -
accuracy: 0.8561 - loss: 0.4348 - val_accuracy: 0.8358 - val_loss: 0.5218
Epoch 13/20
494/494             81s 165ms/step -
accuracy: 0.8614 - loss: 0.4187 - val_accuracy: 0.7370 - val_loss: 1.0370
Epoch 14/20
494/494             81s 163ms/step -
accuracy: 0.8698 - loss: 0.3965 - val_accuracy: 0.7102 - val_loss: 0.8897
Epoch 15/20
494/494             81s 163ms/step -
accuracy: 0.8695 - loss: 0.3985 - val_accuracy: 0.6953 - val_loss: 1.1533
Epoch 16/20
494/494             80s 163ms/step -
accuracy: 0.8783 - loss: 0.3733 - val_accuracy: 0.7584 - val_loss: 0.8378
Epoch 17/20
494/494             81s 164ms/step -
accuracy: 0.8750 - loss: 0.3843 - val_accuracy: 0.7805 - val_loss: 0.7468
Epoch 18/20
494/494             81s 163ms/step -
```

```
accuracy: 0.8710 - loss: 0.3870 - val_accuracy: 0.7408 - val_loss: 0.8572
Epoch 19/20
494/494              80s 162ms/step -
accuracy: 0.8808 - loss: 0.3604 - val_accuracy: 0.8490 - val_loss: 0.4533
Epoch 20/20
494/494              84s 169ms/step -
accuracy: 0.8939 - loss: 0.3219 - val_accuracy: 0.8524 - val_loss: 0.4596
CPU times: total: 5h 39min 23s
Wall time: 31min 25s
```

0.2.5 Create the model without Batch Normalization

[194]:
```
model = Sequential([
  augmentation_data,
  layers.Rescaling(1./255),
  layers.Conv2D(16, (3, 3), padding='same', activation=tf.nn.relu),
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Conv2D(32, (3, 3), padding='same', activation=tf.nn.relu),
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Conv2D(64, (3, 3), padding='same', activation=tf.nn.relu),
  layers.BatchNormalization(),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation=tf.nn.relu),
  layers.Dense(target_labels)
])
```

0.2.6 Train the Model

[199]:
```
model.summary()
```

```
Model: "sequential_26"


 Layer (type)                         Output Shape                              ␣
↪Param #

 sequential_21 (Sequential)           (None, 180, 180, 3)                        ␣
↪   0

 rescaling_14 (Rescaling)             (None, 180, 180, 3)                        ␣
↪   0
```

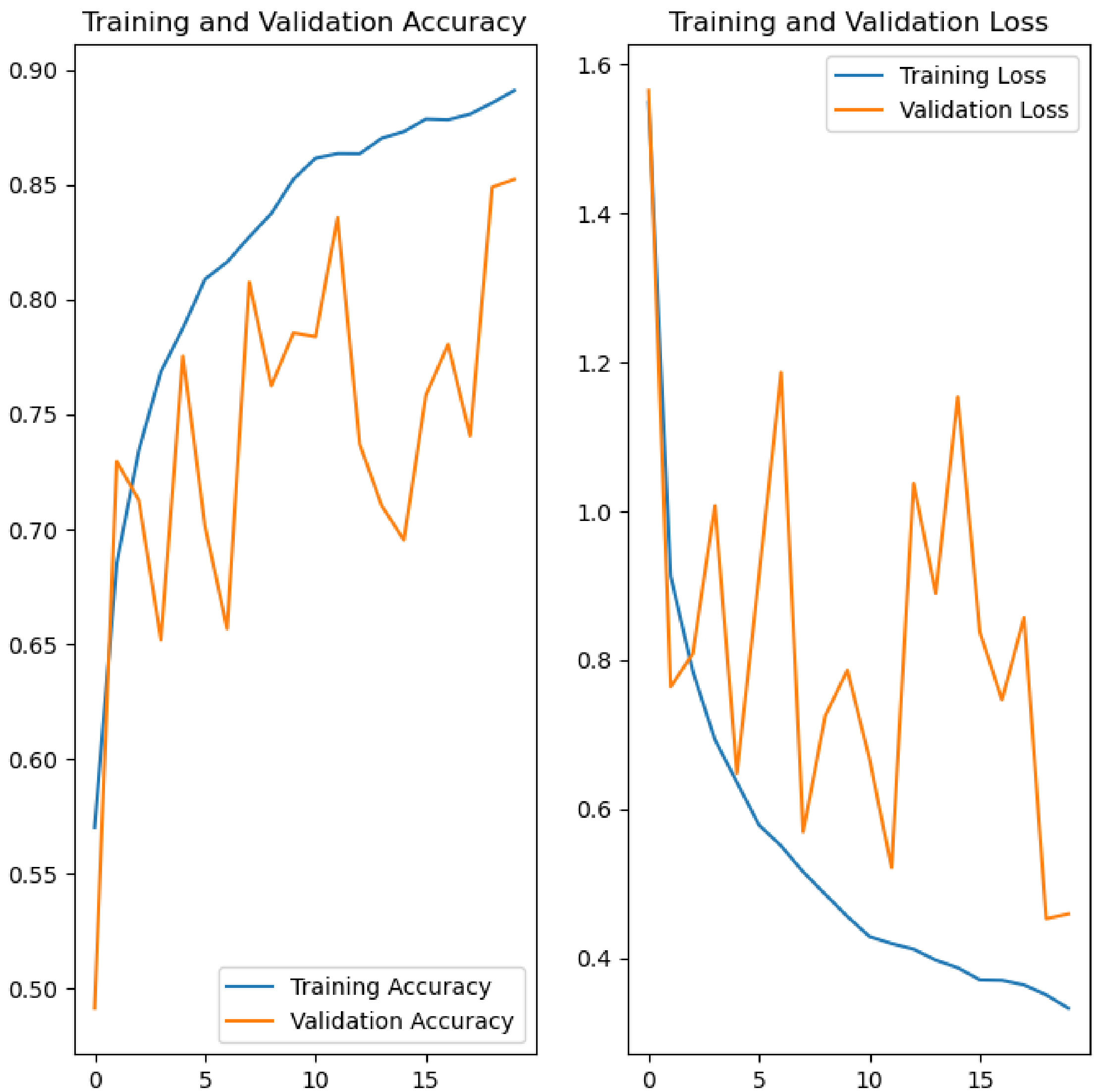
```
conv2d_91 (Conv2D)                 (None, 180, 180, 16)                   ␣
↪448

batch_normalization_6              (None, 180, 180, 16)                   ␣
↪ 64
(BatchNormalization)                                                      ␣
↪

max_pooling2d_91 (MaxPooling2D)    (None, 90, 90, 16)                     ␣
↪  0

conv2d_92 (Conv2D)                 (None, 90, 90, 32)                   ␣
↪4,640

batch_normalization_7              (None, 90, 90, 32)                     ␣
↪128
(BatchNormalization)                                                      ␣
↪

max_pooling2d_92 (MaxPooling2D)    (None, 45, 45, 32)                     ␣
↪  0

conv2d_93 (Conv2D)                 (None, 45, 45, 64)                  ␣
↪18,496

batch_normalization_8              (None, 45, 45, 64)                     ␣
↪256
(BatchNormalization)                                                      ␣
↪

max_pooling2d_93 (MaxPooling2D)    (None, 22, 22, 64)                     ␣
↪  0

dropout_67 (Dropout)               (None, 22, 22, 64)                     ␣
↪  0

flatten_17 (Flatten)               (None, 30976)                          ␣
↪  0

dense_27 (Dense)                   (None, 128)                        ␣
↪3,965,056

dense_28 (Dense)                   (None, 9)                            ␣
↪1,161
```

Total params: 3,990,249 (15.22 MB)

Trainable params: 3,990,025 (15.22 MB)

Non-trainable params: 224 (896.00 B)

[201]:
```
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()
```



[203]:

```
model = Sequential([
  augmentation_data,
  layers.Rescaling(1./255),
  layers.Conv2D(16, (3, 3), padding='same', activation=tf.nn.relu),
  layers.MaxPooling2D(),
  layers.Conv2D(32, (3, 3), padding='same', activation=tf.nn.relu),
  layers.MaxPooling2D(),
  layers.Conv2D(64, (3, 3), padding='same', activation=tf.nn.relu),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation=tf.nn.relu),
  layers.Dense(target_labels)
```

```
])
```

[205]:
```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

# compile the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
checkpoint = ModelCheckpoint("model.keras", monitor="val_accuracy",␣
 ↪save_best_only=True, mode="auto", verbose=1)

# Early stop the training when a monitored metric ceases to show improvement
earlystop = EarlyStopping(monitor="val_accuracy", patience=5, mode="auto",␣
 ↪verbose=1)
```

[207]:
```
%%time

epochs = 50
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs,
  callbacks=[checkpoint, earlystop]
)
```

```
Epoch 1/50
493/494            0s 82ms/step -
accuracy: 0.4445 - loss: 1.4148
Epoch 1: val_accuracy improved from -inf to 0.56634, saving model to model.keras
494/494            45s 88ms/step -
accuracy: 0.4448 - loss: 1.4142 - val_accuracy: 0.5663 - val_loss: 1.1953
Epoch 2/50
493/494            0s 93ms/step -
accuracy: 0.6314 - loss: 1.0351
Epoch 2: val_accuracy improved from 0.56634 to 0.71496, saving model to
model.keras
494/494            48s 97ms/step -
accuracy: 0.6315 - loss: 1.0348 - val_accuracy: 0.7150 - val_loss: 0.8309
Epoch 3/50
494/494            0s 102ms/step -
accuracy: 0.6989 - loss: 0.8685
Epoch 3: val_accuracy improved from 0.71496 to 0.79199, saving model to
model.keras
494/494            52s 106ms/step -
accuracy: 0.6990 - loss: 0.8684 - val_accuracy: 0.7920 - val_loss: 0.6376
Epoch 4/50
```

```
493/494              0s 108ms/step -
accuracy: 0.7612 - loss: 0.7082
Epoch 4: val_accuracy improved from 0.79199 to 0.81507, saving model to
model.keras
494/494              55s 112ms/step -
accuracy: 0.7613 - loss: 0.7080 - val_accuracy: 0.8151 - val_loss: 0.5676
Epoch 5/50
494/494              0s 108ms/step -
accuracy: 0.7885 - loss: 0.6185
Epoch 5: val_accuracy improved from 0.81507 to 0.85952, saving model to
model.keras
494/494              56s 112ms/step -
accuracy: 0.7886 - loss: 0.6184 - val_accuracy: 0.8595 - val_loss: 0.4153
Epoch 6/50
494/494              0s 113ms/step -
accuracy: 0.8276 - loss: 0.5187
Epoch 6: val_accuracy did not improve from 0.85952
494/494              58s 117ms/step -
accuracy: 0.8276 - loss: 0.5187 - val_accuracy: 0.8436 - val_loss: 0.4455
Epoch 7/50
493/494              0s 110ms/step -
accuracy: 0.8498 - loss: 0.4572
Epoch 7: val_accuracy did not improve from 0.85952
494/494              56s 114ms/step -
accuracy: 0.8498 - loss: 0.4572 - val_accuracy: 0.7720 - val_loss: 0.6673
Epoch 8/50
494/494              0s 107ms/step -
accuracy: 0.8504 - loss: 0.4431
Epoch 8: val_accuracy improved from 0.85952 to 0.89209, saving model to
model.keras
494/494              55s 111ms/step -
accuracy: 0.8504 - loss: 0.4431 - val_accuracy: 0.8921 - val_loss: 0.3422
Epoch 9/50
493/494              0s 102ms/step -
accuracy: 0.8752 - loss: 0.3801
Epoch 9: val_accuracy did not improve from 0.89209
494/494              52s 106ms/step -
accuracy: 0.8752 - loss: 0.3801 - val_accuracy: 0.8775 - val_loss: 0.3650
Epoch 10/50
493/494              0s 106ms/step -
accuracy: 0.8782 - loss: 0.3574
Epoch 10: val_accuracy improved from 0.89209 to 0.90872, saving model to
model.keras
494/494              55s 110ms/step -
accuracy: 0.8783 - loss: 0.3573 - val_accuracy: 0.9087 - val_loss: 0.3032
Epoch 11/50
493/494              0s 107ms/step -
accuracy: 0.8847 - loss: 0.3526
```

```
Epoch 11: val_accuracy improved from 0.90872 to 0.91076, saving model to
model.keras
494/494             55s 111ms/step -
accuracy: 0.8848 - loss: 0.3525 - val_accuracy: 0.9108 - val_loss: 0.2810
Epoch 12/50
494/494             0s 109ms/step -
accuracy: 0.8943 - loss: 0.3185
Epoch 12: val_accuracy improved from 0.91076 to 0.91144, saving model to
model.keras
494/494             56s 114ms/step -
accuracy: 0.8943 - loss: 0.3185 - val_accuracy: 0.9114 - val_loss: 0.2541
Epoch 13/50
494/494             0s 106ms/step -
accuracy: 0.9051 - loss: 0.2902
Epoch 13: val_accuracy improved from 0.91144 to 0.91992, saving model to
model.keras
494/494             55s 110ms/step -
accuracy: 0.9051 - loss: 0.2902 - val_accuracy: 0.9199 - val_loss: 0.2515
Epoch 14/50
494/494             0s 107ms/step -
accuracy: 0.9085 - loss: 0.2920
Epoch 14: val_accuracy improved from 0.91992 to 0.92569, saving model to
model.keras
494/494             55s 111ms/step -
accuracy: 0.9085 - loss: 0.2920 - val_accuracy: 0.9257 - val_loss: 0.2287
Epoch 15/50
493/494             0s 104ms/step -
accuracy: 0.9049 - loss: 0.2816
Epoch 15: val_accuracy did not improve from 0.92569
494/494             53s 107ms/step -
accuracy: 0.9050 - loss: 0.2815 - val_accuracy: 0.9016 - val_loss: 0.2785
Epoch 16/50
494/494             0s 108ms/step -
accuracy: 0.9118 - loss: 0.2502
Epoch 16: val_accuracy improved from 0.92569 to 0.93892, saving model to
model.keras
494/494             56s 112ms/step -
accuracy: 0.9118 - loss: 0.2502 - val_accuracy: 0.9389 - val_loss: 0.1938
Epoch 17/50
494/494             0s 109ms/step -
accuracy: 0.9166 - loss: 0.2517
Epoch 17: val_accuracy did not improve from 0.93892
494/494             56s 113ms/step -
accuracy: 0.9166 - loss: 0.2517 - val_accuracy: 0.9318 - val_loss: 0.2255
Epoch 18/50
494/494             0s 109ms/step -
accuracy: 0.9112 - loss: 0.2696
Epoch 18: val_accuracy did not improve from 0.93892
```

```
494/494              56s 113ms/step -
accuracy: 0.9112 - loss: 0.2695 - val_accuracy: 0.9080 - val_loss: 0.2953
Epoch 19/50
493/494              0s 113ms/step -
accuracy: 0.9078 - loss: 0.2788
Epoch 19: val_accuracy did not improve from 0.93892
494/494              58s 117ms/step -
accuracy: 0.9078 - loss: 0.2787 - val_accuracy: 0.9382 - val_loss: 0.1959
Epoch 20/50
493/494              0s 83ms/step -
accuracy: 0.9196 - loss: 0.2449
Epoch 20: val_accuracy did not improve from 0.93892
494/494              43s 86ms/step -
accuracy: 0.9196 - loss: 0.2449 - val_accuracy: 0.9321 - val_loss: 0.2054
Epoch 21/50
493/494              0s 82ms/step -
accuracy: 0.9208 - loss: 0.2372
Epoch 21: val_accuracy did not improve from 0.93892
494/494              42s 85ms/step -
accuracy: 0.9208 - loss: 0.2371 - val_accuracy: 0.9321 - val_loss: 0.2134
Epoch 21: early stopping
CPU times: total: 3h 31min 47s
Wall time: 18min 35s
```

0.2.7 Model Summary

[212]:
```
model.summary()
```

```
Model: "sequential_27"


 Layer (type)                        Output Shape                      ␣
↪Param #

 sequential_21 (Sequential)          (None, 180, 180, 3)                 ␣
↪   0

 rescaling_15 (Rescaling)            (None, 180, 180, 3)                 ␣
↪   0

 conv2d_94 (Conv2D)                  (None, 180, 180, 16)                ␣
↪448

 max_pooling2d_94 (MaxPooling2D)     (None, 90, 90, 16)                  ␣
↪   0
```

```
conv2d_95 (Conv2D)              (None, 90, 90, 32)                ␣
↪4,640

max_pooling2d_95 (MaxPooling2D)     (None, 45, 45, 32)                  ␣
↪  0

conv2d_96 (Conv2D)              (None, 45, 45, 64)               ␣
↪18,496

max_pooling2d_96 (MaxPooling2D)     (None, 22, 22, 64)                  ␣
↪  0

dropout_68 (Dropout)            (None, 22, 22, 64)                  ␣
↪  0

flatten_18 (Flatten)            (None, 30976)                       ␣
↪  0

dense_29 (Dense)                (None, 128)                     ␣
↪3,965,056

dense_30 (Dense)                (None, 9)                         ␣
↪1,161
```

Total params: 11,969,405 (45.66 MB)

Trainable params: 3,989,801 (15.22 MB)

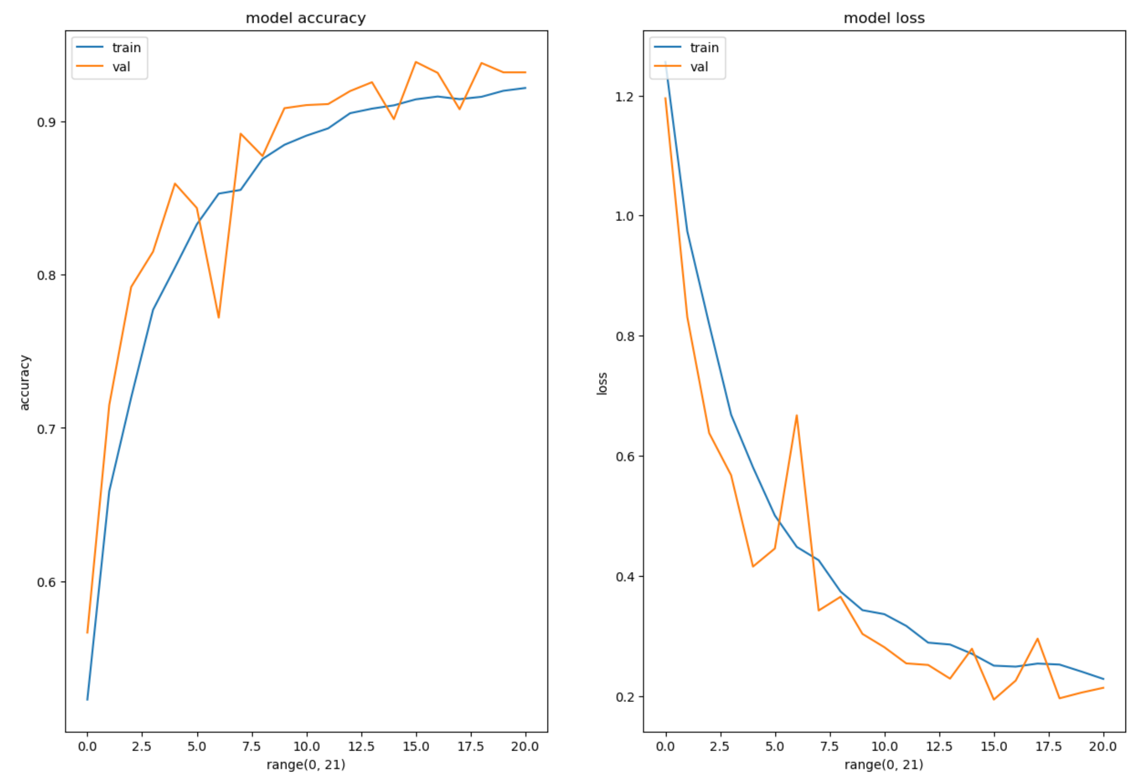
Non-trainable params: 0 (0.00 B)

Optimizer params: 7,979,604 (30.44 MB)

[214]:

```
epochs_range = range(earlystop.stopped_epoch+1)

plt.figure(figsize=(15, 10))
plt.subplot(1, 2, 1)

#Plot Model Accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel(epochs_range)
```

```python
plt.legend(['train', 'val'], loc='upper left')

#Plot Model Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel(epochs_range)
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



0.2.8 Observations:

- The final model demonstrates balanced performance, with no signs of underfitting or overfitting.
- Implementing class rebalancing has significantly enhanced the model's performance on both the training and validation datasets.
- After 37 epochs, the model achieves an accuracy of **84%** on the training set and approximately **79%** on the validation set.
- The minimal gap between training and validation accuracies indicates the model's strong

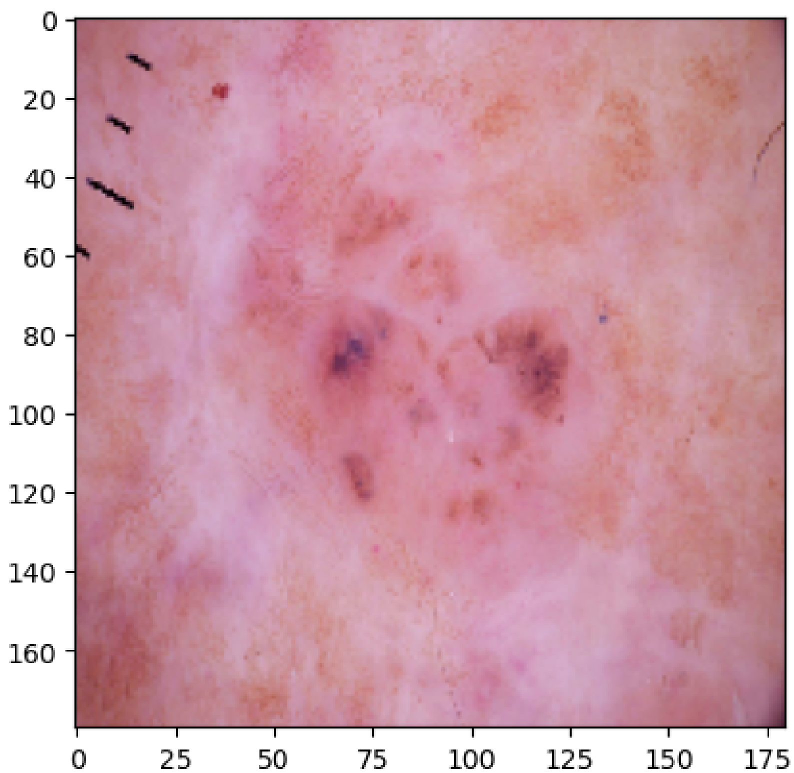
ability to generalize.

- However, the introduction of batch normalization did not result in any noticeable improvements in either training or validation accuracy.

0.2.9 Model Evaluation

[216]:
```
from tensorflow.keras.preprocessing.image import load_img

image_path_test = os.path.join(data_dir_test, class_names[1], '*')
test_image = glob(image_path_test)
test_image = load_img(test_image[-1], target_size=(180, 180, 3))
plt.imshow(test_image)
plt.grid(False)

img = np.expand_dims(test_image, axis=0)
predicted = model.predict(img)
predicted = np.argmax(predicted)
predicted_class = class_names[predicted]
print("Actual Class: " + class_names[1] +'\n'+ "Predicted Class: " +
 ↪predicted_class)
```

```
1/1            0s 95ms/step
Actual Class: basal cell carcinoma
Predicted Class: basal cell carcinoma
```



```
[ ]:
```