Problem statement:

Melanoma is a dangerous type of skin cancer that can be fatal if not detected in its early stages. Despite representing a small fraction of skin cancer cases, melanoma accounts for approximately 75% of all skin cancer-related deaths. Early and accurate detection is essential to improve survival rates.

```python
#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

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint,EarlyStopping

from tensorflow.keras.preprocessing.image import load_img

# Defining the path for train and test images
data_dir_train = pathlib.Path("Train")
data_dir_test = pathlib.Path("Test")

# 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
```

Data Visualization

```python
#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,bat
ch_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")
```
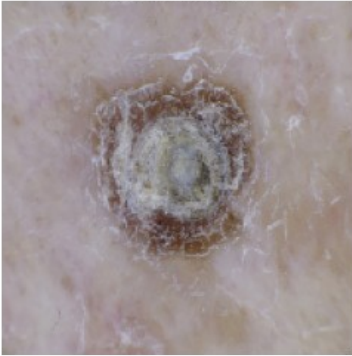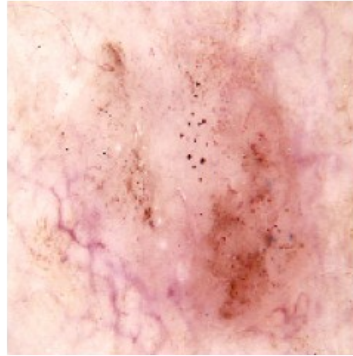
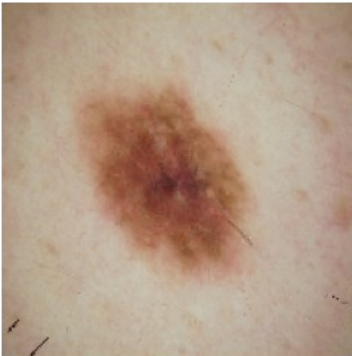Found 2239 files belonging to 9 classes.

actinic keratosis	basal cell carcinoma	dermatofibroma
melanoma	nevus	pigmented benign keratosis
seborrheic keratosis	squamous cell carcinoma	vascular lesion
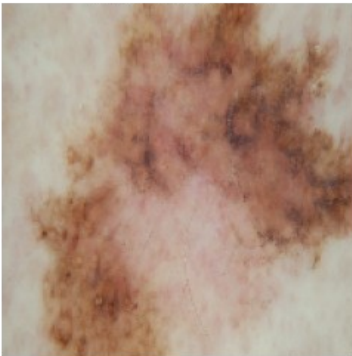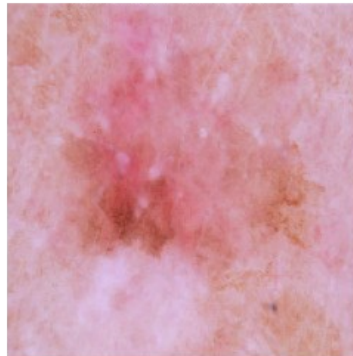
Visualize distribution of classes in the training dataset.

```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))]))
```

```python
    #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
```

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

```python
#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")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f74b43b9390>
```
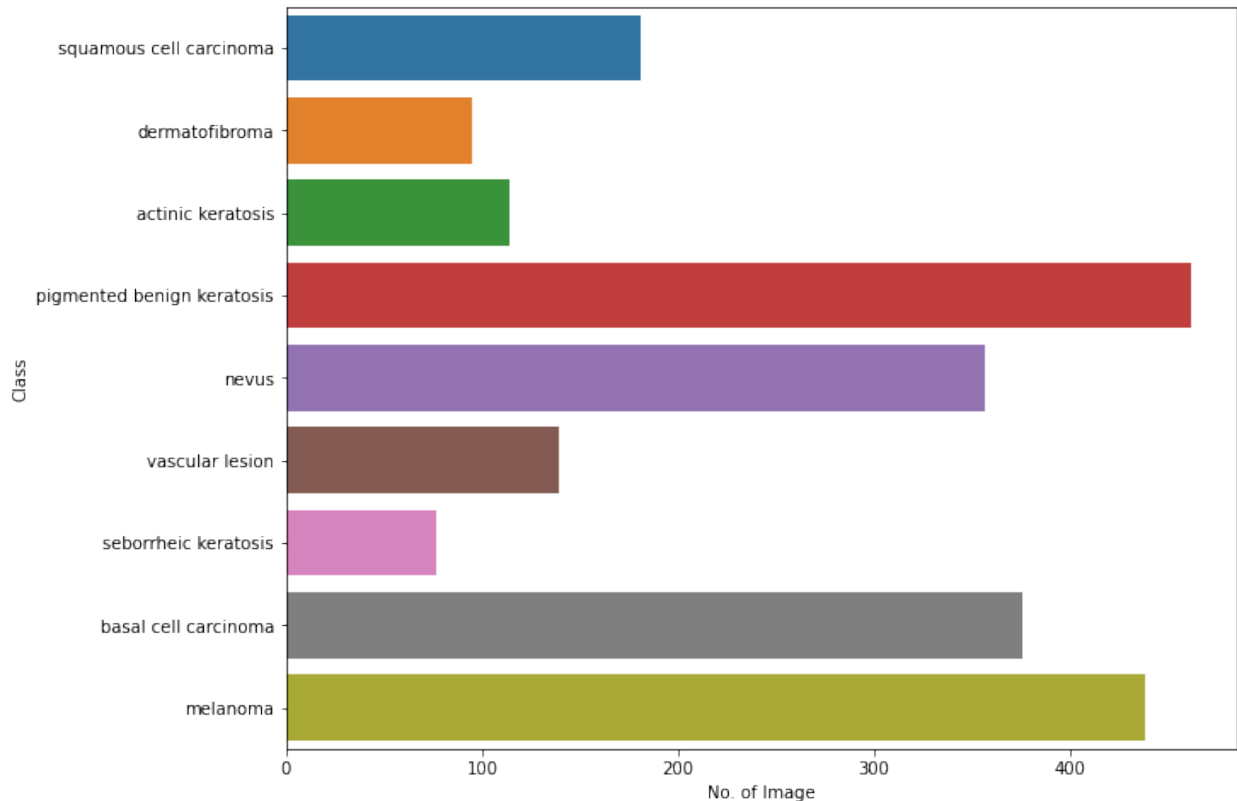
There is a class imbalance to solve this using a python package Augmentor
(https://augmentor.readthedocs.io/en/master/) to add more samples across all classes so that
none of the classes have very few samples.

```
#install Augmentor
!pip install Augmentor

Collecting Augmentor
  Downloading Augmentor-0.2.8-py2.py3-none-any.whl (38 kB)
Requirement already satisfied: numpy>=1.11.0 in
/usr/local/lib/python3.7/dist-packages (from Augmentor) (1.19.5)
Requirement already satisfied: tqdm>=4.9.0 in
/usr/local/lib/python3.7/dist-packages (from Augmentor) (4.62.3)
Requirement already satisfied: Pillow>=5.2.0 in
/usr/local/lib/python3.7/dist-packages (from Augmentor) (7.1.2)
Requirement already satisfied: future>=0.16.0 in
/usr/local/lib/python3.7/dist-packages (from Augmentor) (0.16.0)
Installing collected packages: Augmentor
Successfully installed Augmentor-0.2.8

from pathlib import Path
import Augmentor

# Set path to training dataset
path_to_training_dataset = Path("C:/Users/HP/Desktop/Skin cancer ISIC
```

```
The International Skin Imaging Collaboration/Train")

# Augment images in each class
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) #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/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/actinic keratosis/output.

Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB
size=600x450 at 0x7F75338535D0>: 100%|██████████| 500/500
[00:19<00:00, 26.19 Samples/s]

Initialised with 376 image(s) found.
Output directory set to /content/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/basal cell carcinoma/output.

Processing <PIL.Image.Image image mode=RGB size=600x450 at
0x7F758429DED0>: 100%|██████████| 500/500 [00:19<00:00, 25.43
Samples/s]

Initialised with 95 image(s) found.
Output directory set to /content/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/dermatofibroma/output.

Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB
size=600x450 at 0x7F75336B3710>: 100%|██████████| 500/500
[00:19<00:00, 25.83 Samples/s]

Initialised with 438 image(s) found.
Output directory set to /content/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/melanoma/output.

Processing <PIL.Image.Image image mode=RGB size=1024x768 at
0x7F74B3A78B90>: 100%|██████████| 500/500 [01:33<00:00, 5.34
Samples/s]

Initialised with 357 image(s) found.
Output directory set to /content/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/nevus/output.

Processing <PIL.Image.Image image mode=RGB size=576x768 at
0x7F75338C8890>: 100%|██████████| 500/500 [01:36<00:00, 5.19
Samples/s]

```
Initialised with 462 image(s) found.
Output directory set to /content/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/pigmented benign keratosis/output.

Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB
size=600x450 at 0x7F74AE7FFB10>: 100%|████████| 500/500
[00:20<00:00, 24.64 Samples/s]

Initialised with 77 image(s) found.
Output directory set to /content/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/seborrheic keratosis/output.

Processing <PIL.Image.Image image mode=RGB size=1024x768 at
0x7F74B3B58BD0>: 100%|████████| 500/500 [00:47<00:00, 10.46
Samples/s]

Initialised with 181 image(s) found.
Output directory set to /content/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/squamous cell carcinoma/output.

Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB
size=600x450 at 0x7F75336EEF90>: 100%|████████| 500/500
[00:19<00:00, 25.78 Samples/s]

Initialised with 139 image(s) found.
Output directory set to /content/Skin cancer ISIC The International
Skin Imaging Collaboration/Train/vascular lesion/output.

Processing <PIL.Image.Image image mode=RGB size=600x450 at
0x7F7533950C50>: 100%|████████| 500/500 [00:19<00:00, 25.74
Samples/s]
```

```python
#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

```python
# train dataset
train_ds =
tf.keras.preprocessing.image_dataset_from_directory(data_dir_train,
batch_size=32,

image_size=(180,180), label_mode='categorical',

seed=123,subset="training",

validation_split=0.2)
```

```python
#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.
```

```
Found 6739 files belonging to 9 classes.
Using 5392 files for training.
```

```python
# validation dataset
val_ds
=tf.keras.preprocessing.image_dataset_from_directory(data_dir_train,ba
tch_size=32,

image_size=(180,180), label_mode='categorical',

seed=123,subset="validation",

validation_split=0.2)
```

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

```python
#tf.data.experimental.AUTOTUNE defines appropriate number of processes
that are free for working.

#`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)
```

```python
#CNN Model Architecture

#Sequential allows you to create models layer-by-layer
model = Sequential()

model.add(layers.experimental.preprocessing.Rescaling(1./255,input_sha
pe=(180,180,3)))   #Rescaling Layer

#First Convulation layer
model.add(layers.Conv2D(32,kernel_size=(3,3),activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))

#Second Convulation Layer
model.add(layers.Conv2D(64,kernel_size=(3,3),activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))
```

```python
#Third Convulation Layer
model.add(layers.Conv2D(128,kernel_size=(3,3),activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))

#Dropout layer with 50% Fraction of the input units to drop.
model.add(layers.Dropout(0.5))

#Flatten Layer
##Keras.layers.flatten function flattens the multi-dimensional input
tensors into a single dimension.
model.add(layers.Flatten())

#Dense Layer
model.add(layers.Dense(128,activation='relu'))

#Dropout layer with 25% Fraction of the input units to drop.
model.add(layers.Dropout(0.25))

#Dense Layer with softmax activation function.
#Softmax is an activation function that scales numbers/logits into
probabilities.
model.add(layers.Dense(len(class_names),activation='softmax'))

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 180, 180, 3)	0
conv2d (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_1 (Conv2D)	(None, 87, 87, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 43, 43, 64)	0
conv2d_2 (Conv2D)	(None, 41, 41, 128)	73856
max_pooling2d_2 (MaxPooling2	(None, 20, 20, 128)	0
dropout (Dropout)	(None, 20, 20, 128)	0
flatten (Flatten)	(None, 51200)	0
dense (Dense)	(None, 128)	6553728

```
dropout_1 (Dropout)              (None, 128)                 0
_____
dense_1 (Dense)                  (None, 9)                   1161
=================================================================
Total params: 6,648,137
Trainable params: 6,648,137
Non-trainable params: 0
_____
```

```python
# vizualizing the model
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='model_plot.png', show_shapes=True,
show_layer_names=True)
```

rescaling_input: InputLayer	input:	[(None, 180, 180, 3)]
	output:	[(None, 180, 180, 3)]

rescaling: Rescaling	input:	(None, 180, 180, 3)
	output:	(None, 180, 180, 3)

conv2d: Conv2D	input:	(None, 180, 180, 3)
	output:	(None, 178, 178, 32)

max_pooling2d: MaxPooling2D	input:	(None, 178, 178, 32)
	output:	(None, 89, 89, 32)

conv2d_1: Conv2D	input:	(None, 89, 89, 32)
	output:	(None, 87, 87, 64)

max_pooling2d_1: MaxPooling2D	input:	(None, 87, 87, 64)
	output:	(None, 43, 43, 64)

conv2d_2: Conv2D	input:	(None, 43, 43, 64)
	output:	(None, 41, 41, 128)

```python
#Compile the Model

#Adam optimization: is a stochastic gradient descent method that is
based on adaptive estimation of first-order and second-order moments.
#categorical_crossentropy: Used as a loss function for multi-class
classification model where there are two or more output labels.

model.compile(optimizer="Adam",loss="categorical_crossentropy",metrics
=["accuracy"])

#ModelCheckpoint callback is used in conjunction with training using
model.fit() to save a model or weights (in a checkpoint file) at some
interval,
#so the model or weights can be loaded later to continue the training
from the state saved.
checkpoint =
ModelCheckpoint("model.h5",monitor="val_accuracy",save_best_only=True,
mode="auto",verbose=1)

#Stop training when a monitored metric has stopped improving.
earlystop =
EarlyStopping(monitor="val_accuracy",patience=5,mode="auto",verbose=1)

# Train the model
epochs = 20
history = model.fit(train_ds, validation_data=val_ds,
epochs=epochs,callbacks=[checkpoint,earlystop])

Epoch 1/20
169/169 [==============================] - 69s 102ms/step - loss:
1.8030 - accuracy: 0.3166 - val_loss: 1.5269 - val_accuracy: 0.4321

Epoch 00001: val_accuracy improved from -inf to 0.43207, saving model
to model.h5
Epoch 2/20
169/169 [==============================] - 12s 69ms/step - loss:
1.4552 - accuracy: 0.4518 - val_loss: 1.3301 - val_accuracy: 0.5308

Epoch 00002: val_accuracy improved from 0.43207 to 0.53081, saving
model to model.h5
Epoch 3/20
169/169 [==============================] - 12s 69ms/step - loss:
1.3019 - accuracy: 0.5109 - val_loss: 1.2471 - val_accuracy: 0.5434

Epoch 00003: val_accuracy improved from 0.53081 to 0.54343, saving
model to model.h5
Epoch 4/20
169/169 [==============================] - 12s 69ms/step - loss:
1.1884 - accuracy: 0.5555 - val_loss: 1.0393 - val_accuracy: 0.6288
```

```
Epoch 00004: val_accuracy improved from 0.54343 to 0.62880, saving
model to model.h5
Epoch 5/20
169/169 [==============================] - 12s 69ms/step - loss:
1.0936 - accuracy: 0.5920 - val_loss: 1.0634 - val_accuracy: 0.6006

Epoch 00005: val_accuracy did not improve from 0.62880
Epoch 6/20
169/169 [==============================] - 12s 69ms/step - loss:
0.9737 - accuracy: 0.6422 - val_loss: 0.8937 - val_accuracy: 0.6711

Epoch 00006: val_accuracy improved from 0.62880 to 0.67112, saving
model to model.h5
Epoch 7/20
169/169 [==============================] - 12s 69ms/step - loss:
0.8609 - accuracy: 0.6908 - val_loss: 1.0064 - val_accuracy: 0.6333

Epoch 00007: val_accuracy did not improve from 0.67112
Epoch 8/20
169/169 [==============================] - 12s 69ms/step - loss:
0.8109 - accuracy: 0.7012 - val_loss: 0.8194 - val_accuracy: 0.7016

Epoch 00008: val_accuracy improved from 0.67112 to 0.70156, saving
model to model.h5
Epoch 9/20
169/169 [==============================] - 12s 68ms/step - loss:
0.7206 - accuracy: 0.7350 - val_loss: 0.7097 - val_accuracy: 0.7513

Epoch 00009: val_accuracy improved from 0.70156 to 0.75130, saving
model to model.h5
Epoch 10/20
169/169 [==============================] - 12s 69ms/step - loss:
0.6686 - accuracy: 0.7532 - val_loss: 0.7351 - val_accuracy: 0.7602

Epoch 00010: val_accuracy improved from 0.75130 to 0.76021, saving
model to model.h5
Epoch 11/20
169/169 [==============================] - 12s 69ms/step - loss:
0.6022 - accuracy: 0.7695 - val_loss: 0.7016 - val_accuracy: 0.7580

Epoch 00011: val_accuracy did not improve from 0.76021
Epoch 12/20
169/169 [==============================] - 12s 68ms/step - loss:
0.5660 - accuracy: 0.7897 - val_loss: 0.7897 - val_accuracy: 0.7290

Epoch 00012: val_accuracy did not improve from 0.76021
Epoch 13/20
169/169 [==============================] - 12s 68ms/step - loss:
0.5252 - accuracy: 0.8021 - val_loss: 0.7509 - val_accuracy: 0.7706
```

```
Epoch 00013: val_accuracy improved from 0.76021 to 0.77060, saving
model to model.h5
Epoch 14/20
169/169 [==============================] - 12s 69ms/step - loss:
0.4755 - accuracy: 0.8251 - val_loss: 0.6387 - val_accuracy: 0.8062

Epoch 00014: val_accuracy improved from 0.77060 to 0.80624, saving
model to model.h5
Epoch 15/20
169/169 [==============================] - 12s 69ms/step - loss:
0.4638 - accuracy: 0.8259 - val_loss: 0.6336 - val_accuracy: 0.7751

Epoch 00015: val_accuracy did not improve from 0.80624
Epoch 16/20
169/169 [==============================] - 12s 69ms/step - loss:
0.4002 - accuracy: 0.8453 - val_loss: 0.5632 - val_accuracy: 0.8040

Epoch 00016: val_accuracy did not improve from 0.80624
Epoch 17/20
169/169 [==============================] - 12s 69ms/step - loss:
0.4229 - accuracy: 0.8399 - val_loss: 0.5827 - val_accuracy: 0.8211

Epoch 00017: val_accuracy improved from 0.80624 to 0.82108, saving
model to model.h5
Epoch 18/20
169/169 [==============================] - 12s 68ms/step - loss:
0.4143 - accuracy: 0.8420 - val_loss: 0.5611 - val_accuracy: 0.8114

Epoch 00018: val_accuracy did not improve from 0.82108
Epoch 19/20
169/169 [==============================] - 12s 68ms/step - loss:
0.3800 - accuracy: 0.8542 - val_loss: 0.5537 - val_accuracy: 0.8426

Epoch 00019: val_accuracy improved from 0.82108 to 0.84261, saving
model to model.h5
Epoch 20/20
169/169 [==============================] - 12s 69ms/step - loss:
0.4168 - accuracy: 0.8477 - val_loss: 0.5865 - val_accuracy: 0.8129

Epoch 00020: val_accuracy did not improve from 0.84261

# Plot the training curves

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'])
```
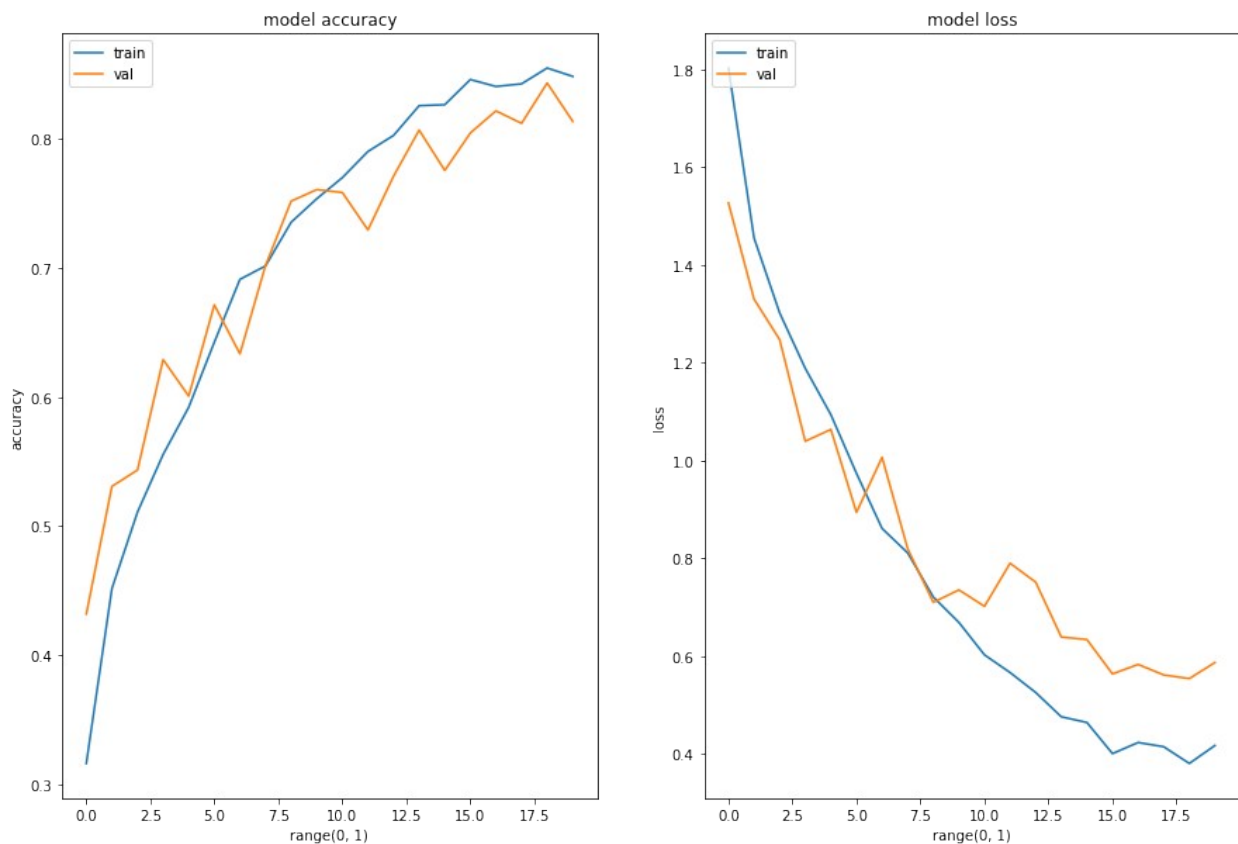
```
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel(epochs_range)
plt.legend(['train', 'val'], loc='upper left')

#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()
```


Model Prediction

```
from glob import glob
Test_image_path = os.path.join(data_dir_test, class_names[1], '*')
Test_image = glob(Test_image_path)
Test_image = load_img(Test_image[-1],target_size=(180,180,3))
plt.imshow(Test_image)
```

```python
plt.grid(False)

img = np.expand_dims(Test_image,axis=0)
pred = model.predict(img)
pred = np.argmax(pred)
pred_class = class_names[pred]
print("Actual Class "+ class_names[1] +'\n'+ "Predictive Class
"+pred_class )
```

```
Actual Class basal cell carcinoma
Predictive Class basal cell carcinoma
```