121cs0755-assign-6

February 18, 2024

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pathlib
import os
import glob as gb
import cv2
import PIL
import seaborn as sns
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

from tensorflow.keras.callbacks import EarlyStopping ,ReduceLROnPlateau
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D , Dense , Dropout , Flatten ,␣
↪MaxPooling2D , BatchNormalization ,experimental
from tensorflow.keras.utils import to_categorical
from keras.applications.vgg16 import VGG16
from keras.applications.vgg19 import VGG19
from tensorflow import keras
from keras.models import Model
```

[3]:
```
pwd
```

[3]:
```
'C:\\Users\\Admin'
```

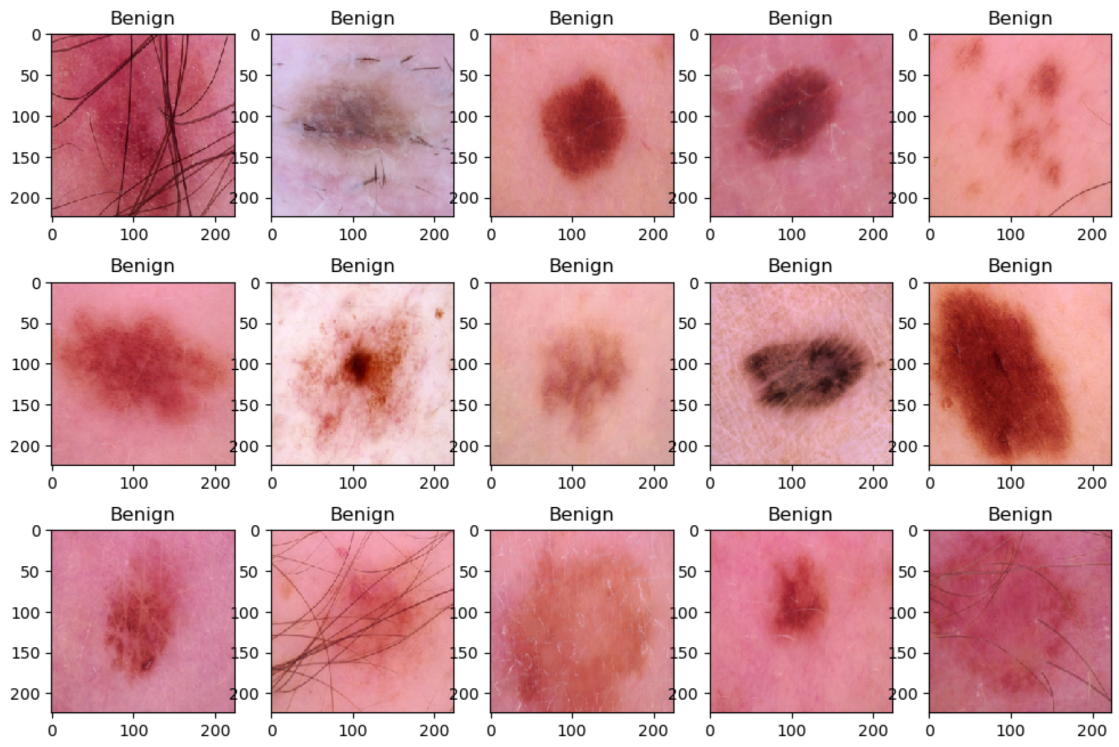
[8]:
```
trainpath = 'C:\\Users\\Admin\\Desktop\\Skin_Cancer\\train'
testpath = 'C:\\Users\\Admin\\Desktop\\Skin_Cancer\\test'
```

[9]:
```
new_size=224
train_images=[]
train_labels=[]
for i in os.listdir(trainpath):#entering train folder
  print("Entering to the folder name:",i)
  files=gb.glob(pathname=str(trainpath+'/' + i + '/*.jpg'))# pointing to all␣
 ↪the .jpg extension image foldetrainpath = 'C:
 ↪\\Users\\Admin\\Desktop\\Skin_Cancer\\train'
testpath = 'C:\\Users\\Admin\\Desktop\\Skin_Cancer\\test'r
  print("Number of images in the folder is",len(files))
  for j in files:# reading each images
      class_cancer={'benign':0,'malignant':1}
      image_raw=cv2.imread(j)
      image=cv2.cvtColor(image_raw,cv2.COLOR_BGR2RGB)
      resize_image=cv2.resize(image,(new_size,new_size))
      train_images.append(list(resize_image))
      train_labels.append(class_cancer[i])
```

```
Entering to the folder name: benign
Number of images in the folder is 25
Entering to the folder name: malignant
Number of images in the folder is 25
```

[10]:
```
w=40
h=30
fig=plt.figure(figsize=(12, 8))
columns = 5
rows = 3

for i in range(1, columns*rows +1):
    ax = fig.add_subplot(rows, columns, i)
    if train_labels[i] == 0:
        ax.title.set_text('Benign')
    else:
        ax.title.set_text('Malignant')
    plt.imshow(train_images[i], interpolation='nearest')
plt.show()
```



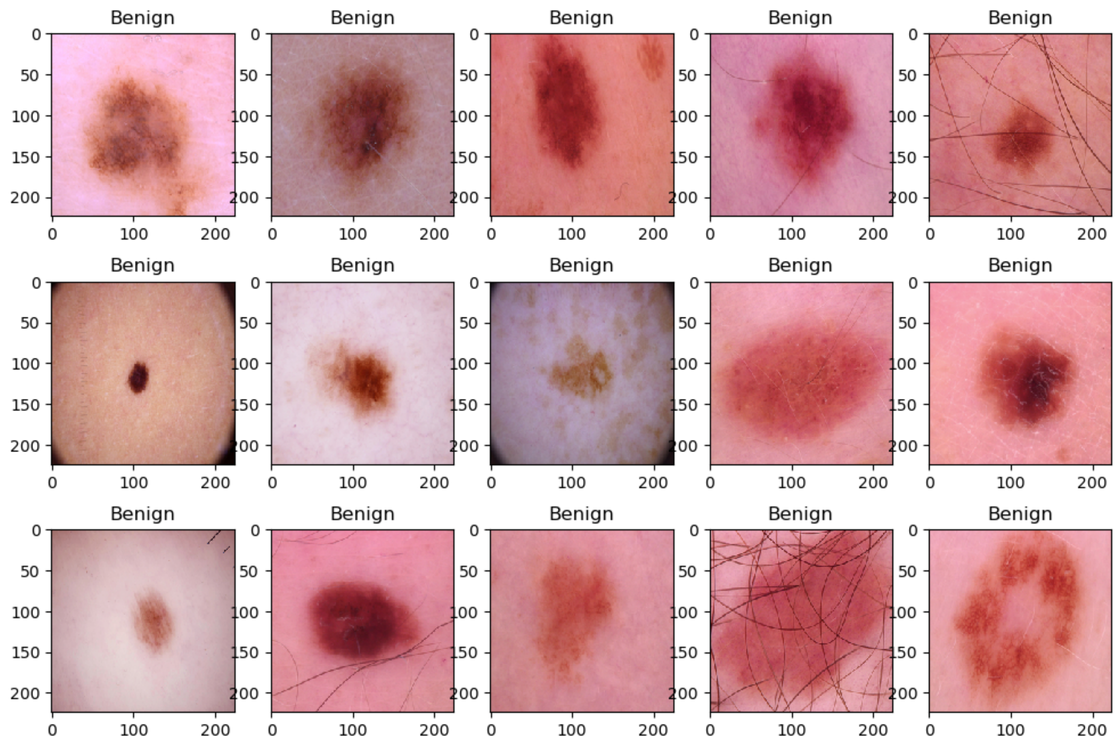
```
[11]: new_size=224
test_images=[]
test_labels=[]
for i in os.listdir(testpath):# entering to the test folder
  print("Entering to the folder name:",i)
  files=gb.glob(pathname=str(testpath +'/' + i + '/*.jpg'))# pointing to all␣
↪the .jpg extension image folder
  print("Number of images in the folder is",len(files))
  for j in files:
      class_cancer={'benign':0,'malignant':1}
      image_raw=cv2.imread(j)
      image=cv2.cvtColor(image_raw,cv2.COLOR_BGR2RGB)
      resize_image=cv2.resize(image,(new_size,new_size))
      test_images.append(list(resize_image))
      test_labels.append(class_cancer[i])
```

```
Entering to the folder name: benign
Number of images in the folder is 25
Entering to the folder name: malignant
Number of images in the folder is 25
```

```
[12]: w=40
h=30
```

```
fig=plt.figure(figsize=(12, 8))
columns = 5
rows = 3

for i in range(1, columns*rows +1):
    ax = fig.add_subplot(rows, columns, i)
    if test_labels[i] == 0:
        ax.title.set_text('Benign')
    else:
        ax.title.set_text('Malignant')
    plt.imshow(test_images[i], interpolation='nearest')
plt.show()
```



[13]:
```
def list_to_array_train(train_images,train_labels):
  return np.array(train_images),np.array(train_labels)

x_train,y_train=list_to_array_train(train_images,train_labels)


def list_to_array_test(test_images,test_labels):
  return np.array(test_images),np.array(test_labels)
```

```
x_test,y_test=list_to_array_test(test_images,test_labels)
```

[15]:
```
print(x_train.shape)
print("*"*20)
print(y_train.shape)
print("*"*20)
print(x_test.shape)
print(y_test.shape)
```

```
(50, 224, 224, 3)
********************
(50,)
********************
(50, 224, 224, 3)
(50,)
```

[16]:
```
def keras_to_categorical(y_train,y_test):
  return to_categorical(y_train),to_categorical(y_test)
y_train1=y_train
y_test1=y_test
y_train,y_test=keras_to_categorical(y_train,y_test)
```

[17]:
```
y_train1.shape,y_test1.shape
```

[17]:
```
((50,), (50,))
```

Question-1.a

[30]:
```
def model_vgg16():
    VGG_model = VGG16(weights='imagenet', include_top=False,␣
 ↪input_shape=(224,224, 3))
  #Make loaded layers as non-trainable. This is important as we want to work␣
 ↪with pre-trained weights
    for layer in VGG_model.layers:
        layer.trainable = False #True for actual transfer learning
        feature=keras.layers.GlobalAveragePooling2D()(VGG_model.output)
        d1=Dense(units=64,kernel_initializer="glorot_uniform",␣
 ↪activation='relu')(feature)
        d2=Dense(units=32,kernel_initializer="glorot_uniform",␣
 ↪activation='sigmoid')(d1)
        d3=Dense(units=2,kernel_initializer="glorot_uniform",␣
 ↪activation='softmax')(d2)
        output = Model(inputs =VGG_model.input, outputs =d3)
  #output = Model(inputs =VGG_model.input, outputs =feature)

    return output
```

```
model16=model_vgg16()
```

```
[31]: model16.compile(optimizer='Adam', loss='mse', metrics='accuracy')
```

```
[33]: history = model16.fit(x_train, y_train, validation_split=0.2,epochs= 10,␣
      ↪batch_size= 5, verbose=1,validation_data=(x_test,y_test))
```

```
Epoch 1/10
WARNING:tensorflow:From C:\Users\Admin\AppData\Roaming\Python\Python39\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\Admin\AppData\Roaming\Python\Python39\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.

10/10 [==============================] - 11s 1s/step - loss: 0.3156 - accuracy:
0.5200 - val_loss: 0.1994 - val_accuracy: 0.7200
Epoch 2/10
10/10 [==============================] - 10s 1s/step - loss: 0.1592 - accuracy:
0.9400 - val_loss: 0.1762 - val_accuracy: 0.7200
Epoch 3/10
10/10 [==============================] - 10s 1s/step - loss: 0.1013 - accuracy:
0.9200 - val_loss: 0.1339 - val_accuracy: 0.8200
Epoch 4/10
10/10 [==============================] - 9s 998ms/step - loss: 0.0743 -
accuracy: 0.9400 - val_loss: 0.1316 - val_accuracy: 0.8400
Epoch 5/10
10/10 [==============================] - 10s 1s/step - loss: 0.0518 - accuracy:
0.9600 - val_loss: 0.1432 - val_accuracy: 0.8400
Epoch 6/10
10/10 [==============================] - 10s 1s/step - loss: 0.0407 - accuracy:
1.0000 - val_loss: 0.1227 - val_accuracy: 0.8000
Epoch 7/10
10/10 [==============================] - 10s 1s/step - loss: 0.0289 - accuracy:
1.0000 - val_loss: 0.1242 - val_accuracy: 0.8400
Epoch 8/10
10/10 [==============================] - 10s 1s/step - loss: 0.0203 - accuracy:
1.0000 - val_loss: 0.1176 - val_accuracy: 0.8200
Epoch 9/10
10/10 [==============================] - 9s 1s/step - loss: 0.0150 - accuracy:
1.0000 - val_loss: 0.1258 - val_accuracy: 0.8400
Epoch 10/10
10/10 [==============================] - 10s 1s/step - loss: 0.0132 - accuracy:
1.0000 - val_loss: 0.1364 - val_accuracy: 0.8200
```

Question-1.b

```
[37]: def model_vgg16():
        VGG_model = VGG16(weights='imagenet', include_top=False,␣
       ↪input_shape=(224,224, 3))
        #Make loaded layers as non-trainable. This is important as we want to work␣
       ↪with pre-trained weights
        for layer in VGG_model.layers:
          layer.trainable = False #True for actual transfer learning
        feature=keras.layers.GlobalAveragePooling2D()(VGG_model.output)
        ##d1=Dense(units=256,kernel_initializer="glorot_uniform", W_regularizer=l2(0.
       ↪001), activation='relu')(feature)
        ##d2=Dense(units=2,kernel_initializer="glorot_uniform", W_regularizer=l2(0.
       ↪001), activation='softmax')(d1)
        #output = Model(inputs =VGG_model.input, outputs =d2)
        output = Model(inputs =VGG_model.input, outputs =feature)

        return output

      model_FE_16=model_vgg16()
```

```
[38]: model_FE_16.compile(optimizer='Adam', loss='mse', metrics='accuracy')
      train_feature_16=model_FE_16.predict(x_train)
      test_feature_16=model_FE_16.predict(x_test)
```

```
2/2 [==============================] - 5s 2s/step
2/2 [==============================] - 5s 2s/step
```

```
[39]: from sklearn.ensemble import RandomForestClassifier
      rf=RandomForestClassifier()
      rf=rf.fit(train_feature_16,y_train)
      train_pred=rf.predict(train_feature_16)
      test_pred=rf.predict(test_feature_16)
      print("Train Accuracy Score",accuracy_score(train_pred,y_train))
      print("Test Accuracy Score",accuracy_score(test_pred,y_test))
```

```
Train Accuracy Score 1.0
Test Accuracy Score 0.8
```

Question-1.c

```
[49]: def model_vgg19():
        VGG19_model = VGG19(weights='imagenet', include_top=False,␣
       ↪input_shape=(224,224, 3))
        #Make loaded layers as non-trainable. This is important as we want to work␣
       ↪with pre-trained weights
        for layer in VGG19_model.layers:
          layer.trainable = False #True for actual transfer learning
        feature=keras.layers.GlobalAveragePooling2D()(VGG19_model.output)
```

```
    d1=Dense(units=64,kernel_initializer="glorot_uniform",␣
 ↪activation='relu')(feature)
    d2=Dense(units=32,kernel_initializer="glorot_uniform",␣
 ↪activation='sigmoid')(d1)
    d3=Dense(units=2,kernel_initializer="glorot_uniform",␣
 ↪activation='softmax')(d2)
    output = Model(inputs =VGG19_model.input, outputs =d3)
    #output = Model(inputs =VGG_model.input, outputs =feature)

    return output

model19=model_vgg19()
```

[50]:
```
model19.compile(optimizer='Adam', loss='mse', metrics='accuracy')
```

[51]:
```
history = model19.fit(x_train, y_train, validation_split=0.2,
                    epochs= 10, batch_size= 5,␣
 ↪verbose=1,validation_data=(x_test,y_test)
                   )
```

```
Epoch 1/10
10/10 [==============================] - 13s 1s/step - loss: 0.2821 - accuracy:
0.6000 - val_loss: 0.1557 - val_accuracy: 0.8000
Epoch 2/10
10/10 [==============================] - 12s 1s/step - loss: 0.1557 - accuracy:
0.7200 - val_loss: 0.1255 - val_accuracy: 0.8600
Epoch 3/10
10/10 [==============================] - 13s 1s/step - loss: 0.0869 - accuracy:
0.9400 - val_loss: 0.1178 - val_accuracy: 0.8400
Epoch 4/10
10/10 [==============================] - 13s 1s/step - loss: 0.0565 - accuracy:
0.9400 - val_loss: 0.1231 - val_accuracy: 0.8400
Epoch 5/10
10/10 [==============================] - 13s 1s/step - loss: 0.0384 - accuracy:
0.9800 - val_loss: 0.1237 - val_accuracy: 0.8400
Epoch 6/10
10/10 [==============================] - 12s 1s/step - loss: 0.0301 - accuracy:
1.0000 - val_loss: 0.1149 - val_accuracy: 0.8200
Epoch 7/10
10/10 [==============================] - 13s 1s/step - loss: 0.0195 - accuracy:
1.0000 - val_loss: 0.1070 - val_accuracy: 0.8400
Epoch 8/10
10/10 [==============================] - 13s 1s/step - loss: 0.0144 - accuracy:
1.0000 - val_loss: 0.1227 - val_accuracy: 0.8200
Epoch 9/10
10/10 [==============================] - 13s 1s/step - loss: 0.0093 - accuracy:
1.0000 - val_loss: 0.1091 - val_accuracy: 0.8400
```

```
Epoch 10/10
10/10 [==============================] - 13s 1s/step - loss: 0.0068 - accuracy:
1.0000 - val_loss: 0.1182 - val_accuracy: 0.8200
```

[43]:
```
def model_vgg19():
  VGG_model = VGG19(weights='imagenet', include_top=False,␣
↪input_shape=(224,224, 3))
  #Make loaded layers as non-trainable. This is important as we want to work␣
↪with pre-trained weights
  for layer in VGG_model.layers:
    layer.trainable = False #True for actual transfer learning
  feature=keras.layers.GlobalAveragePooling2D()(VGG_model.output)
  ##d1=Dense(units=256,kernel_initializer="glorot_uniform", W_regularizer=l2(0.
↪001), activation='relu')(feature)
  ##d2=Dense(units=2,kernel_initializer="glorot_uniform", W_regularizer=l2(0.
↪001), activation='softmax')(d1)
  #output = Model(inputs =VGG_model.input, outputs =d2)
  output = Model(inputs =VGG_model.input, outputs =feature)

  return output

model_FE_19=model_vgg19()
```

[44]:
```
model_FE_19.compile(optimizer='Adam', loss='mse', metrics='accuracy')
train_feature_19=model_FE_19.predict(x_train)
test_feature_19=model_FE_19.predict(x_test)
```

```
2/2 [==============================] - 6s 2s/step
2/2 [==============================] - 6s 2s/step
```

[45]:
```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf=rf.fit(train_feature_19,y_train)
train_pred=rf.predict(train_feature_19)
test_pred=rf.predict(test_feature_19)
print("Train Accuracy Score",accuracy_score(train_pred,y_train))
print("Test Accuracy Score",accuracy_score(test_pred,y_test))
```

```
Train Accuracy Score 1.0
Test Accuracy Score 0.76
```

[46]:
```
final_train=np.hstack((train_feature_16,train_feature_19))
final_test=np.hstack((test_feature_16,test_feature_19))
```

[47]:
```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf=rf.fit(final_train,y_train)
```

```
train_pred=rf.predict(final_train)
test_pred=rf.predict(final_test)
print("Train Accuracy Score",accuracy_score(train_pred,y_train))
print("Test Accuracy Score",accuracy_score(test_pred,y_test))
```

```
Train Accuracy Score 1.0
Test Accuracy Score 0.86
```

[48]:
```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc=dtc.fit(final_train,y_train)
train_pred=dtc.predict(final_train)
test_pred=dtc.predict(final_test)
print("Train Accuracy Score",accuracy_score(train_pred,y_train))
print("Test Accuracy Score",accuracy_score(test_pred,y_test))
```

```
Train Accuracy Score 1.0
Test Accuracy Score 0.76
```

Question-2.a

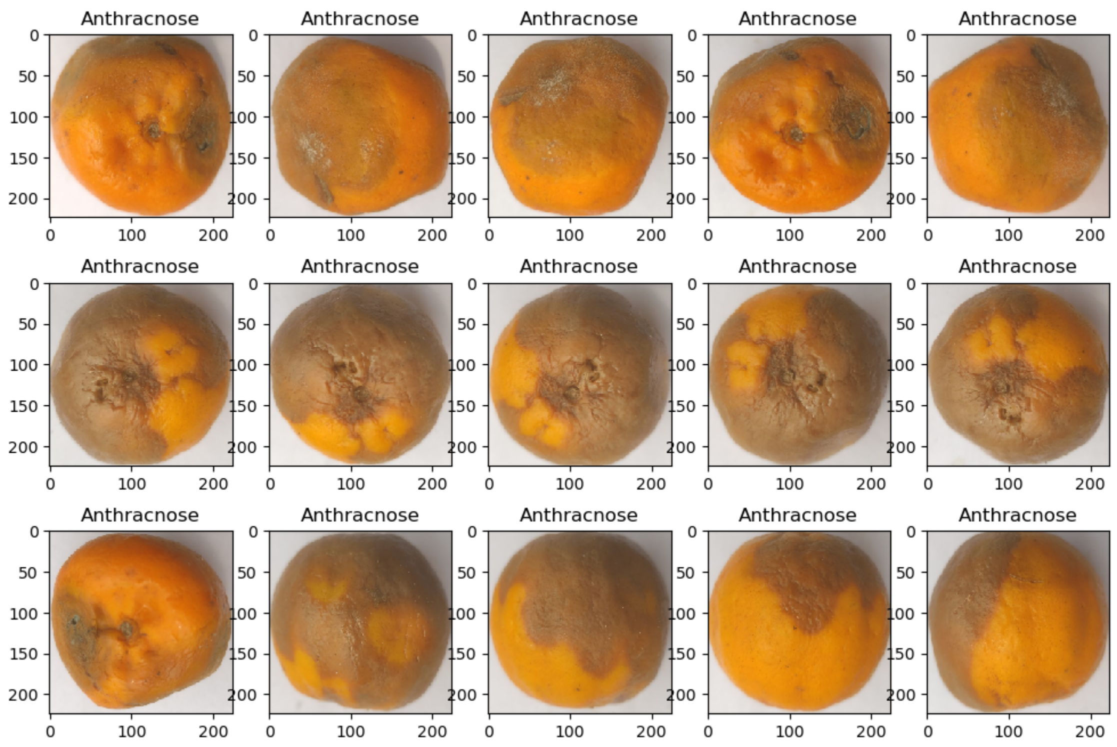
[54]:
```
train1path = 'C:\\Users\\Admin\\Desktop\\Orange_Dataset\\train'
test1path = 'C:\\Users\\Admin\\Desktop\\Orange_Dataset\\test'
```

[55]:
```
new_size=224
train_images=[]
train_labels=[]
for i in os.listdir(trainpath):#entering train folder
  print("Entering to the folder name:",i)
  files=gb.glob(pathname=str(train1path+'/' + i + '/*.jpg'))# pointing to all␣
 ↪the .jpg extension image folder
  print("Number of images in the folder is",len(files))
  for j in files:# reading each images
      class_cancer={'Anthracnose':0,'BlackSpot':1,'Healthy':2}
      image_raw=cv2.imread(j)
      image=cv2.cvtColor(image_raw,cv2.COLOR_BGR2RGB)
      resize_image=cv2.resize(image,(new_size,new_size))
      train_images.append(list(resize_image))
      train_labels.append(class_cancer[i])
```

```
Entering to the folder name: Anthracnose
Number of images in the folder is 30
Entering to the folder name: BlackSpot
Number of images in the folder is 30
Entering to the folder name: Healthy
Number of images in the folder is 30
```

```
[56]: w=40
h=30
fig=plt.figure(figsize=(12, 8))
columns = 5
rows = 3

for i in range(1, columns*rows +1):
    ax = fig.add_subplot(rows, columns, i)
    if train_labels[i] == 0:
        ax.title.set_text('Anthracnose')
    elif train_labels[i] == 1:
        ax.title.set_text('BlackSpot')
    else:
        ax.title.set_text('Healthy')
    plt.imshow(train_images[i], interpolation='nearest')
plt.show()
```



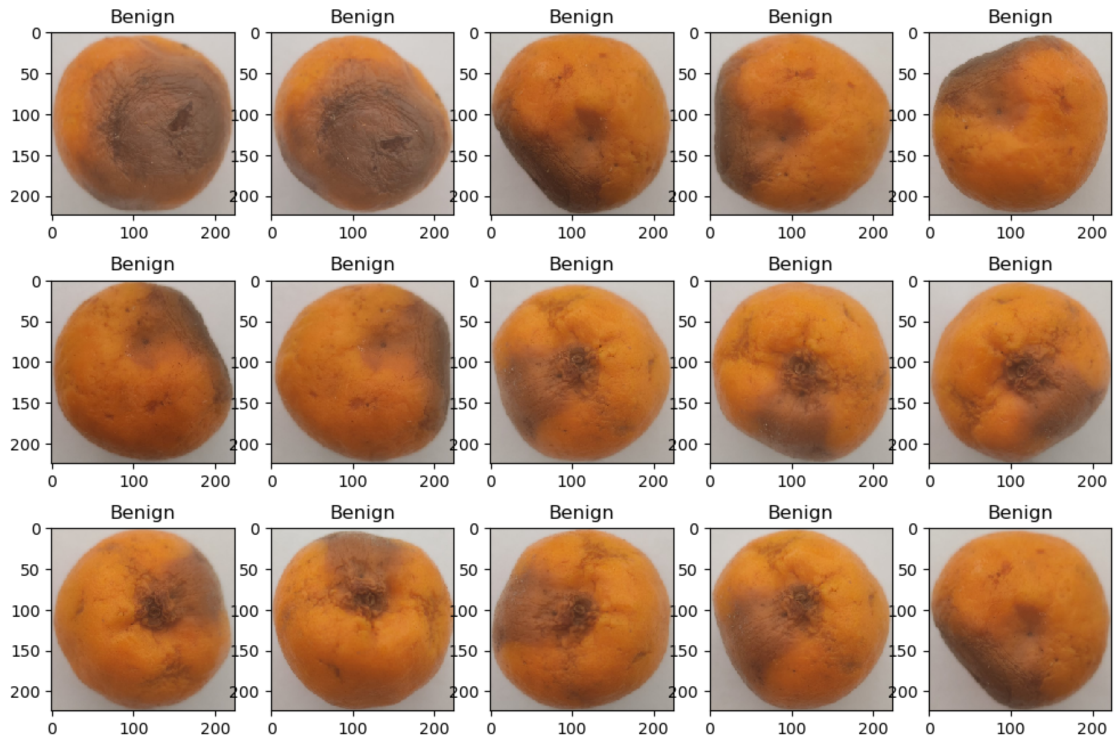
```
[58]: new_size=224
test_images=[]
test_labels=[]
for i in os.listdir(testpath):# entering to the test folder
  print("Entering to the folder name:",i)
```

```
 files=gb.glob(pathname=str(test1path +'/' + i + '/*.jpg'))# pointing to all␣
↪the .jpg extension image folder
 print("Number of images in the folder is",len(files))
 for j in files:
     class_cancer={'Anthracnose':0,'BlackSpot':1,'Healthy':2}
     image_raw=cv2.imread(j)
     image=cv2.cvtColor(image_raw,cv2.COLOR_BGR2RGB)
     resize_image=cv2.resize(image,(new_size,new_size))
     test_images.append(list(resize_image))
     test_labels.append(class_cancer[i])
```

```
Entering to the folder name: Anthracnose
Number of images in the folder is 25
Entering to the folder name: BlackSpot
Number of images in the folder is 25
Entering to the folder name: Healthy
Number of images in the folder is 25
```

[59]:
```
w=40
h=30
fig=plt.figure(figsize=(12, 8))
columns = 5
rows = 3

for i in range(1, columns*rows +1):
    ax = fig.add_subplot(rows, columns, i)
    if test_labels[i] == 0:
        ax.title.set_text('Benign')
    else:
        ax.title.set_text('Malignant')
    plt.imshow(test_images[i], interpolation='nearest')
plt.show()
```



[60]:
```
def list_to_array_train(train_images,train_labels):
  return np.array(train_images),np.array(train_labels)

X_train,y_train=list_to_array_train(train_images,train_labels)


def list_to_array_test(test_images,test_labels):
  return np.array(test_images),np.array(test_labels)


X_test,y_test=list_to_array_test(test_images,test_labels)
```

[61]:
```
print(X_train.shape)
print("*"*20)
print(y_train.shape)
print("*"*20)
print(X_test.shape)
print(y_test.shape)
```

```
(90, 224, 224, 3)
********************
(90,)
********************
(75, 224, 224, 3)
```

```
(75,)
```

[62]:
```
def keras_to_categorical(y_train,y_test):
  return to_categorical(y_train),to_categorical(y_test)
y_train1=y_train
y_test1=y_test
y_train,y_test=keras_to_categorical(y_train,y_test)
```

[63]:
```
y_train1.shape,y_test1.shape
```

[63]:
```
((90,), (75,))
```

[76]:
```
def model_vgg16():
  VGG_model = VGG16(weights='imagenet', include_top=False,␣
 ↪input_shape=(224,224, 3))
  #Make loaded layers as non-trainable. This is important as we want to work␣
 ↪with pre-trained weights
  for layer in VGG_model.layers:
    layer.trainable = False #True for actual transfer learning
    feature=keras.layers.GlobalAveragePooling2D()(VGG_model.output)
    d1=Dense(units=64,kernel_initializer="glorot_uniform",␣
 ↪activation='relu')(feature)
    d2=Dense(units=32,kernel_initializer="glorot_uniform",␣
 ↪activation='sigmoid')(d1)
    d3=Dense(units=3,kernel_initializer="glorot_uniform",␣
 ↪activation='softmax')(d2)
    output = Model(inputs =VGG_model.input, outputs =d3)
  #output = Model(inputs =VGG_model.input, outputs =feature)

    return output

model16=model_vgg16()
```

[77]:
```
model16.compile(optimizer='Adam', loss='mse', metrics='accuracy')
```

[78]:
```
import numpy as np
res=np.zeros(5)
for i in range(5):
    history = model16.fit(X_train, y_train, validation_split=0.2,epochs= 10,␣
 ↪batch_size= 5, verbose=1,validation_data=(X_test,y_test))
    res[i]=acc
```

```
Epoch 1/10
18/18 [==============================] - 54s 3s/step - loss: 0.2288 - accuracy:
0.3889 - val_loss: 0.2262 - val_accuracy: 0.3333
Epoch 2/10
18/18 [==============================] - 52s 3s/step - loss: 0.2253 - accuracy:
```

```
0.3333 - val_loss: 0.2236 - val_accuracy: 0.3333
Epoch 3/10
18/18 [==============================] - 52s 3s/step - loss: 0.2224 - accuracy:
0.3556 - val_loss: 0.2229 - val_accuracy: 0.3333
Epoch 4/10
18/18 [==============================] - 51s 3s/step - loss: 0.2233 - accuracy:
0.3333 - val_loss: 0.2226 - val_accuracy: 0.3333
Epoch 5/10
18/18 [==============================] - 51s 3s/step - loss: 0.2232 - accuracy:
0.3333 - val_loss: 0.2223 - val_accuracy: 0.3333
Epoch 6/10
18/18 [==============================] - 51s 3s/step - loss: 0.2226 - accuracy:
0.2778 - val_loss: 0.2223 - val_accuracy: 0.3333
Epoch 7/10
18/18 [==============================] - 51s 3s/step - loss: 0.2225 - accuracy:
0.2667 - val_loss: 0.2222 - val_accuracy: 0.3333
Epoch 8/10
18/18 [==============================] - 52s 3s/step - loss: 0.2225 - accuracy:
0.2556 - val_loss: 0.2222 - val_accuracy: 0.3333
Epoch 9/10
18/18 [==============================] - 54s 3s/step - loss: 0.2232 - accuracy:
0.2556 - val_loss: 0.2224 - val_accuracy: 0.3333
Epoch 10/10
18/18 [==============================] - 55s 3s/step - loss: 0.2231 - accuracy:
0.2556 - val_loss: 0.2223 - val_accuracy: 0.3333
```

Question-2.b

[79]:
```
def model_vgg16():
    VGG_model = VGG16(weights='imagenet', include_top=False,␣
 ↪input_shape=(224,224, 3))
    #Make loaded layers as non-trainable. This is important as we want to work␣
 ↪with pre-trained weights
    for layer in VGG_model.layers:
        layer.trainable = False #True for actual transfer learning
    feature=keras.layers.GlobalAveragePooling2D()(VGG_model.output)
    ##d1=Dense(units=256,kernel_initializer="glorot_uniform", W_regularizer=l2(0.
 ↪001), activation='relu')(feature)
    ##d2=Dense(units=2,kernel_initializer="glorot_uniform", W_regularizer=l2(0.
 ↪001), activation='softmax')(d1)
    #output = Model(inputs =VGG_model.input, outputs =d2)
    output = Model(inputs =VGG_model.input, outputs =feature)

    return output

model_FE_16=model_vgg16()
```

[80]:
```
model_FE_16.compile(optimizer='Adam', loss='mse', metrics='accuracy')
train_feature_16=model_FE_16.predict(X_train)
test_feature_16=model_FE_16.predict(X_test)
```

```
3/3 [==============================] - 9s 3s/step
3/3 [==============================] - 7s 2s/step
```

[81]:
```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf=rf.fit(train_feature_16,y_train)
train_pred=rf.predict(train_feature_16)
test_pred=rf.predict(test_feature_16)
print("Train Accuracy Score",accuracy_score(train_pred,y_train))
print("Test Accuracy Score",accuracy_score(test_pred,y_test))
```

```
Train Accuracy Score 1.0
Test Accuracy Score 1.0
```

Question-2.c

[85]:
```
#vgg-119 model
def model_vgg19():
  VGG19_model = VGG19(weights='imagenet', include_top=False,␣
 ↪input_shape=(224,224, 3))
  #Make loaded layers as non-trainable. This is important as we want to work␣
 ↪with pre-trained weights
  for layer in VGG19_model.layers:
    layer.trainable = False #True for actual transfer learning
  feature=keras.layers.GlobalAveragePooling2D()(VGG19_model.output)
  d1=Dense(units=64,kernel_initializer="glorot_uniform",␣
 ↪activation='relu')(feature)
  d2=Dense(units=32,kernel_initializer="glorot_uniform",␣
 ↪activation='sigmoid')(d1)
  d3=Dense(units=3,kernel_initializer="glorot_uniform",␣
 ↪activation='softmax')(d2)
  output = Model(inputs =VGG19_model.input, outputs =d3)
  #output = Model(inputs =VGG_model.input, outputs =feature)

  return output

model19=model_vgg19()
```

[86]:
```
model19.compile(optimizer='Adam', loss='mse', metrics='accuracy')
```

[87]:
```
history = model19.fit(X_train, y_train, validation_split=0.2,
                      epochs= 10, batch_size= 5,␣
 ↪verbose=1,validation_data=(X_test,y_test))
```

```
Epoch 1/10
18/18 [==============================] - 22s 1s/step - loss: 0.1832 - accuracy:
0.5333 - val_loss: 0.0930 - val_accuracy: 1.0000
Epoch 2/10
18/18 [==============================] - 21s 1s/step - loss: 0.0460 - accuracy:
1.0000 - val_loss: 0.0267 - val_accuracy: 1.0000
Epoch 3/10
18/18 [==============================] - 22s 1s/step - loss: 0.0162 - accuracy:
1.0000 - val_loss: 0.0194 - val_accuracy: 1.0000
Epoch 4/10
18/18 [==============================] - 22s 1s/step - loss: 0.0106 - accuracy:
1.0000 - val_loss: 0.0127 - val_accuracy: 1.0000
Epoch 5/10
18/18 [==============================] - 21s 1s/step - loss: 0.0078 - accuracy:
1.0000 - val_loss: 0.0096 - val_accuracy: 1.0000
Epoch 6/10
18/18 [==============================] - 22s 1s/step - loss: 0.0061 - accuracy:
1.0000 - val_loss: 0.0076 - val_accuracy: 1.0000
Epoch 7/10
18/18 [==============================] - 22s 1s/step - loss: 0.0049 - accuracy:
1.0000 - val_loss: 0.0062 - val_accuracy: 1.0000
Epoch 8/10
18/18 [==============================] - 21s 1s/step - loss: 0.0042 - accuracy:
1.0000 - val_loss: 0.0053 - val_accuracy: 1.0000
Epoch 9/10
18/18 [==============================] - 21s 1s/step - loss: 0.0036 - accuracy:
1.0000 - val_loss: 0.0045 - val_accuracy: 1.0000
Epoch 10/10
18/18 [==============================] - 21s 1s/step - loss: 0.0031 - accuracy:
1.0000 - val_loss: 0.0040 - val_accuracy: 1.0000
```

[88]:
```
#Feature extraction of vgg-19

def model_vgg19():
  VGG_model = VGG19(weights='imagenet', include_top=False,␣
 ↪input_shape=(224,224, 3))
  #Make loaded layers as non-trainable. This is important as we want to work␣
 ↪with pre-trained weights
  for layer in VGG_model.layers:
    layer.trainable = False #True for actual transfer learning
  feature=keras.layers.GlobalAveragePooling2D()(VGG_model.output)
  ##d1=Dense(units=256,kernel_initializer="glorot_uniform", W_regularizer=l2(0.
 ↪001), activation='relu')(feature)
  ##d2=Dense(units=2,kernel_initializer="glorot_uniform", W_regularizer=l2(0.
 ↪001), activation='softmax')(d1)
  #output = Model(inputs =VGG_model.input, outputs =d2)
  output = Model(inputs =VGG_model.input, outputs =feature)
  return output

model_FE_19=model_vgg19()
```

[89]:
```
model_FE_19.compile(optimizer='Adam', loss='mse', metrics='accuracy')
train_feature_19=model_FE_19.predict(X_train)
test_feature_19=model_FE_19.predict(X_test)
```

```
3/3 [==============================] - 10s 3s/step
3/3 [==============================] - 9s 3s/step
```

[90]:
```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf=rf.fit(train_feature_19,y_train)
train_pred=rf.predict(train_feature_19)
test_pred=rf.predict(test_feature_19)
print("Train Accuracy Score",accuracy_score(train_pred,y_train))
print("Test Accuracy Score",accuracy_score(test_pred,y_test))
```

```
Train Accuracy Score 1.0
Test Accuracy Score 1.0
```

[ ]:
```
#fusion of features in vgg16 and vgg19
```

[92]:
```
final_train=np.hstack((train_feature_16,train_feature_19))
final_test=np.hstack((test_feature_16,test_feature_19))
```

[93]:
```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf=rf.fit(final_train,y_train)
train_pred=rf.predict(final_train)
test_pred=rf.predict(final_test)
print("Train Accuracy Score",accuracy_score(train_pred,y_train))
print("Test Accuracy Score",accuracy_score(test_pred,y_test))
```

```
Train Accuracy Score 1.0
Test Accuracy Score 1.0
```

[94]:
```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc=dtc.fit(final_train,y_train)
train_pred=dtc.predict(final_train)
test_pred=dtc.predict(final_test)
print("Train Accuracy Score",accuracy_score(train_pred,y_train))
print("Test Accuracy Score",accuracy_score(test_pred,y_test))
```

```
Train Accuracy Score 1.0
Test Accuracy Score 1.0
```

[ ]:

```

```