```
pwd
```

Out[1]:

```
'/home/wsuser/work'
```

In [2]:

```
import warnings
warnings.filterwarnings("ignore")
```

In [3]:

```
!pip install imutils
```

```
Collecting imutils
  Downloading imutils-0.5.4.tar.gz (17 kB)
Building wheels for collected packages: imutils
  Building wheel for imutils (setup.py) ... done
  Created wheel for imutils: filename=imutils-0.5.4-py3-none-any.whl size=2
5860 sha256=9b27083f9cc4fd096c0ed96e02d664a4fd9cbef45c07309a802fd276271febc
4
  Stored in directory: /tmp/wsuser/.cache/pip/wheels/4b/a5/2d/4a070a801d3a3
d93f033d3ee9728f470f514826e89952df3ea
Successfully built imutils
Installing collected packages: imutils
Successfully installed imutils-0.5.4
```

Image Pre-processing

Importing the necessary libraries

In [4]:

```
import cv2
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from skimage import feature
from imutils import paths
import os
import pickle
```

Functions to load and quantify the images

In [5]:

```
import os, types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It
includes your credentials.
# You might want to remove those credentials before you share the notebook.
cos_client = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='rlyEPAR18kPTMGZRByQGAWk_2w81tdr_D7ky9zEjT6ja',
    ibm_auth_endpoint="https://iam.cloud.ibm.com/oidc/token",
```

```
    config=Config(signature_version='oauth'),
    endpoint_url='https://s3.private.us.cloud-object-
storage.appdomain.cloud')

bucket = 'parkinson39sdiseasedetection-donotdelete-pr-9rhldkrukkwg0y'
object_key = 'dataset.zip'

streaming_body_1 = cos_client.get_object(Bucket=bucket,
Key=object_key)['Body']

# Your data file was loaded into a botocore.response.StreamingBody object.
# Please read the documentation of ibm_boto3 and pandas to learn more about
the possibilities to load the data.
# ibm_boto3 documentation: https://ibm.github.io/ibm-cos-sdk-python/
# pandas documentation: http://pandas.pydata.org/
```

In [6]:

```
from io import BytesIO
import zipfile
unzip = zipfile.ZipFile(BytesIO(streaming_body_1.read()),'r')
file_paths = unzip.namelist()
for path in file_paths:
    unzip.extract(path)
```

In [7]:

```
pwd
```

Out[7]:

```
'/home/wsuser/work'
```

In [8]:

```
def quantify_image(image):
    features = feature.hog(image,
                           orientations=9,
                           pixels_per_cell=(5,5),
                           cells_per_block=(2,2),
                           transform_sqrt=True,
                           block_norm="L1")
    return features
```

In [9]:

```
def load_split(path):
    path_images = list(paths.list_images(path))
    data=[]
    labels=[]

    for path_image in path_images:
        label = path_image.split(os.path.sep)[-2]
        image = cv2.imread(path_image)
        image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
        image = cv2.resize(image, (200,200))
        image = cv2.threshold(image,0,225,cv2.THRESH_BINARY_INV |
cv2.THRESH_OTSU)[1]

        features = quantify_image(image)
        data.append(features)
        labels.append(label)

    return (np.array(data), np.array(labels))
```

Using spiral images

Defining the path for training data and testing data

In [10]:

```
path_training_data = r"/home/wsuser/work/dataset/spiral/training"
path_testing_data = r"/home/wsuser/work/dataset/spiral/testing"
```

Loading the training and testing data

In [11]:

```
(x_train, y_train) = load_split(path_training_data)
(x_test, y_test) = load_split(path_testing_data)
```

Label Encoding

In [12]:

```
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)
print(x_train.shape, y_train.shape)
# 0:healthy,1:Parkinson
```

```
(72, 54756) (72,)
```

Building the model

Training the model

In [13]:

```
model = RandomForestClassifier(n_estimators=100)
model.fit(x_train, y_train)
```

Out[13]:

```
RandomForestClassifier()
```

Testing the model

In [14]:

```
testingPaths = list(paths.list_images(path_testing_data))
idxs = np.arange(0, len(testingPaths))
idxs = np.random.choice(idxs, size=(25,), replace=False)
images = []
```

In [15]:

```
for i in idxs:
    # loading the testing image, clone it, and resize it
    image = cv2.imread(testingPaths[i])
    output = image. copy()
    output = cv2. resize(output, (128, 128))

    # pre-processing the image
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image, (200, 200))
```

```
    image = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY_INV |
cv2.THRESH_OTSU) [1]

    # quantify the image and make predictions based on the extracted
    # features using the last trained Random Forest
    features = quantify_image(image)
    preds = model.predict([features])

    label = label_encoder.inverse_transform(preds)[0]

    # draw the colored class label on the output image and add it to the
set of output images
    if label == "healthy":
        color = (0, 255, 0)
    else:
        color = (0, 0, 255)
    cv2.putText(output, label, (3, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
color, 2)
    images.append(output)
```

Evaluating the model

In [16]:

```
predictions = model.predict(x_test)    # predictions on the testing data

cm = confusion_matrix(y_test, predictions).flatten ()    # computing the
confusion matrix
print(cm)
(tn, fp, fn, tp) = cm

accuracy = (tp + tn) / float(cm.sum())     # computing the accuracy
print(accuracy)
```

```
[12  3  5 10]
0.7333333333333333
```

Saving the model

In [17]:

```
pickle.dump(model,open('parkinson.pkl', 'wb'))
```

In [18]:

```
!tar -zcvf parkinsonsmodel.tgz parkinson.pkl
```

```
parkinson.pkl
```

In [19]:

```
ls
```

```
dataset/  parkinson.pkl  parkinsonsmodel.tgz
```

In [20]:

```
!pip install watson-machine-learning-client --upgrade

from ibm_watson_machine_learning import APIClient
wml_credentials = {
                    "url" : "https://us-south.ml.cloud.ibm.com",
```

```
                    "apikey" :
"K36t7CdPrC_R4VWU_knCWisv6ePwSofTJbYqDvwtHL54"
                    }
client = APIClient(wml_credentials)
```

In [22]:

```
def guid_from_space_name(client,space_name):
    space = client.spaces.get_details()
    print(space)
    return (next(item for item in space['resources'] if
item['entity']['name']==space_name)['metadata']['id'])
```

In [23]:

```
space_uid = guid_from_space_name(client,'ParkinsonsDiseaseDetection')
print('Space UID = ' + space_uid)
client.set.default_space(space_uid)
```

Out[24]:

```
'SUCCESS'
```

In [25]:

```
client.software_specifications.list()
```

```
------------------------------  ------------------------------------  ----
NAME                            ASSET_ID                              TYPE
default_py3.6                   0062b8c9-8b7d-44a0-a9b9-46c416adcbd9  base
kernel-spark3.2-scala2.12       020d69ce-7ac1-5e68-ac1a-31189867356a  base
pytorch-onnx_1.3-py3.7-edt      069ea134-3346-5748-b513-49120e15d288  base
scikit-learn_0.20-py3.6         09c5a1d0-9c1e-4473-a344-eb7b665ff687  base
spark-mllib_3.0-scala_2.12      09f4cff0-90a7-5899-b9ed-1ef348aebdee  base
pytorch-onnx_rt22.1-py3.9       0b848dd4-e681-5599-be41-b5f6fccc6471  base
ai-function_0.1-py3.6           0cdb0f1e-5376-4f4d-92dd-da3b69aa9bda  base
shiny-r3.6                      0e6e79df-875e-4f24-8ae9-62dcc2148306  base
tensorflow_2.4-py3.7-horovod    1092590a-307d-563d-9b62-4eb7d64b3f22  base
pytorch_1.1-py3.6               10ac12d6-6b30-4ccd-8392-3e922c096a92  base
tensorflow_1.15-py3.6-ddl       111e41b3-de2d-5422-a4d6-bf776828c4b7  base
runtime-22.1-py3.9              12b83a17-24d8-5082-900f-0ab31fbfd3cb  base
scikit-learn_0.22-py3.6         154010fa-5b3b-4ac1-82af-4d5ee5abbc85  base
default_r3.6                    1b70aec3-ab34-4b87-8aa0-a4a3c8296a36  base
pytorch-onnx_1.3-py3.6          1bc6029a-cc97-56da-b8e0-39c3880dbbe7  base
kernel-spark3.3-r3.6            1c9e5454-f216-59dd-a20e-474a5cdf5988  base
pytorch-onnx_rt22.1-py3.9-edt   1d362186-7ad5-5b59-8b6c-9d0880bde37f  base
tensorflow_2.1-py3.6            1eb25b84-d6ed-5dde-b6a5-3fbdf1665666  base
spark-mllib_3.2                 20047f72-0a98-58c7-9ff5-a77b012eb8f5  base
tensorflow_2.4-py3.8-horovod    217c16f6-178f-56bf-824a-b19f20564c49  base
runtime-22.1-py3.9-cuda         26215f05-08c3-5a41-a1b0-da66306ce658  base
do_py3.8                        295addb5-9ef9-547e-9bf4-92ae3563e720  base
autoai-ts_3.8-py3.8             2aa0c932-798f-5ae9-abd6-15e0c2402fb5  base
tensorflow_1.15-py3.6           2b73a275-7cbf-420b-a912-eae7f436e0bc  base
kernel-spark3.3-py3.9           2b7961e2-e3b1-5a8c-a491-482c8368839a  base
pytorch_1.2-py3.6               2c8ef57d-2687-4b7d-acce-01f94976dac1  base
spark-mllib_2.3                 2e51f700-bca0-4b0d-88dc-5c6791338875  base
pytorch-onnx_1.1-py3.6-edt      32983cea-3f32-4400-8965-dde874a8d67e  base
spark-mllib_3.0-py37            36507ebe-8770-55ba-ab2a-eafe787600e9  base
spark-mllib_2.4                 390d21f8-e58b-4fac-9c55-d7ceda621326  base
xgboost_0.82-py3.6              39e31acd-5f30-41dc-ae44-60233c80306e  base
pytorch-onnx_1.2-py3.6-edt      40589d0e-7019-4e28-8daa-fb03b6f4fe12  base
default_r36py38                 41c247d3-45f8-5a71-b065-8580229facf0  base
autoai-ts_rt22.1-py3.9          4269d26e-07ba-5d40-8f66-2d495b0c71f7  base
```

```
autoai-obm_3.0                 42b92e18-d9ab-567f-988a-4240ba1ed5f7  base
pmml-3.0_4.3                   493bcb95-16f1-5bc5-bee8-81b8af80e9c7  base
spark-mllib_2.4-r_3.6          49403dff-92e9-4c87-a3d7-a42d0021c095  base
xgboost_0.90-py3.6             4ff8d6c2-1343-4c18-85e1-689c965304d3  base
pytorch-onnx_1.1-py3.6         50f95b2a-bc16-43bb-bc94-b0bed208c60b  base
autoai-ts_3.9-py3.8            52c57136-80fa-572e-8728-a5e7cbb42cde  base
spark-mllib_2.4-scala_2.11     55a70f99-7320-4be5-9fb9-9edb5a443af5  base
spark-mllib_3.0                5c1b0ca2-4977-5c2e-9439-ffd44ea8ffe9  base
autoai-obm_2.0                 5c2e37fa-80b8-5e77-840f-d912469614ee  base
spss-modeler_18.1              5c3cad7e-507f-4b2a-a9a3-ab53a21dee8b  base
cuda-py3.8                     5d3232bf-c86b-5df4-a2cd-7bb870a1cd4e  base
autoai-kb_3.1-py3.7            632d4b22-10aa-5180-88f0-f52dfb6444d7  base
pytorch-onnx_1.7-py3.8         634d3cdc-b562-5bf9-a2d4-ea90a478456b  base
spark-mllib_2.3-r_3.6          6586b9e3-ccd6-4f92-900f-0f8cb2bd6f0c  base
tensorflow_2.4-py3.7           65e171d7-72d1-55d9-8ebb-f813d620c9bb  base
spss-modeler_18.2              687eddc9-028a-4117-b9dd-e57b36f1efa5  base
-----------------------------  ------------------------------------  ----
Note: Only first 50 records were displayed. To display more use 'limit' par
ameter.
```

In [26]:

```
software_spec_uid=client.software_specifications.get_uid_by_name("default_p
y3.6")
software_spec_uid
```

Out[26]:

```
'0062b8c9-8b7d-44a0-a9b9-46c416adcbd9'
```

In [ ]:

```
model_details =
client.repository.store_model(model='parkinsonsmodel.tgz',meta_props={
client.repository.ModelMetaNames.NAME: "parkinson",
client.repository.ModelMetaNames.TYPE: "default_py3.6",
client.repository.ModelMetaNames.SOFTWARE_SPEC_UID: software_spec_uid})
model_id = client.repository.get_model_uid(model_details)
```

Using wave images

Defining the path for wave training data and testing data

In [58]:

```
path_training_data = r"dataset/wave/training"
path_testing_data = r"dataset/wave/testing"
```

Loading the training and testing data

In [61]:

```
(x_train, y_train) = load_split(path_training_data)
(x_test, y_test) = load_split(path_testing_data)
```

Label Encoding

In [62]:

```
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)
print(x_train.shape, y_train.shape)
```

```
# 0:healthy,1:Parkinson
```

```
(72, 54756) (72,)
```

Building the model

Training the model

In [63]:

```
model = RandomForestClassifier(n_estimators=100)
model.fit(x_train, y_train)
```

Out[63]:

```
RandomForestClassifier()
```

Testing the model

In [64]:

```
testingPaths = list(paths.list_images(path_testing_data))
idxs = np.arange(0, len(testingPaths))
idxs = np.random.choice(idxs, size=(25,), replace=False)
images = []
```

In [65]:

```
for i in idxs:
    # loading the testing image, clone it, and resize it
    image = cv2.imread(testingPaths[i])
    output = image. copy()
    output = cv2. resize(output, (128, 128))

    # pre-processing the image
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image, (200, 200))
    image = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY_INV |
cv2.THRESH_OTSU) [1]

    # quantify the image and make predictions based on the extracted
    # features using the last trained Random Forest
    features = quantify_image(image)
    preds = model.predict([features])

    label = label_encoder.inverse_transform(preds)[0]

    # draw the colored class label on the output image and add it to the
set of output images
    if label == "healthy":
        color = (0, 255, 0)
    else:
        color = (0, 0, 255)
    cv2.putText(output, label, (3, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
color, 2)
    images.append(output)
```

Evaluating the model

In [66]:

```
predictions = model.predict(x_test)    # predictions on the testing data

cm = confusion_matrix(y_test, predictions).flatten ()    # computing the
confusion matrix
print(cm)
(tn, fp, fn, tp) = cm

accuracy = (tp + tn) / float(cm.sum())     # computing the accuracy
print(accuracy)
```

```
[10  5  5 10]
0.6666666666666666
```

Saving the model

In [67]:

```
pickle.dump(model,open('parkinson_w.pkl', 'wb'))
```

In [68]:

```
!tar -zcvf parkinsons-detection-model_s.tgz parkinson_w.pkl
```

```
parkinson_w.pkl
```

In [69]:

```
ls
```

```
dataset/       parkinsons-detection-model_s.tgz  parkinson_w.pkl
parkinson.pkl  parkinsonsmodel.tgz
```