Introduction

What is Parkinson's Disease?

Parkinson's disease (PD), or simply Parkinson's is a long-term degenerative disorder of the central nervous system that mainly affects the motor system. The symptoms usually emerge slowly and, as the disease worsens, non-motor symptoms become more common. The most obvious early symptoms are tremor, rigidity, slowness of movement, and difficulty with walking,but cognitive and behavioral problems may also occur. Parkinson's disease dementia becomes common in the advanced stages of the disease. Depression and anxiety are also common, occurring in more than a third of people with PD. Other symptoms include sensory, sleep, and emotional problems. The main motor symptoms are collectively called "parkinsonism", or a "parkinsonian syndrome

Dataset Description

The dataset we'll be using here today was curated by Adriano de Oliveira Andrade and Joao Paulo Folado from the NIATS of Federal University of Uberlândia.

The dataset itself consists of images and is pre-split into a training set and a testing set, consisting of:

Spiral: training, and testing

Wave: training, and testing

Approach

Although Deep learning with Convolutional Neural networks seems to be the best approach for this computer vision problem, we have a limited amount of training data and we cannot apply data augmentation as it will lead to a distortion of the results. With this in mind we will rather apply the Histogram of Oriented Gradients Image Descriptor with an ensemble method i.e., Random Forest Classifier and Xgboost

Load the data and extract the features

```
import os
import cv2
import numpy as np
from skimage import feature
import random
import matplotlib.pyplot as plt
%matplotlib inline
```

Quantify the image using a HOG Descriptor

HOG is a structural descriptor that will capture and quantify changes in local gradient in the input image. HOG will naturally be able to quantify how the directions of a both spirals and waves change. And furthermore, HOG will be able to capture if these drawings have more of a “shake” to them, as we might expect from a Parkinson's patient.

The resultant feature vector will then be used to train the classifier

```
def quantify_image(image):
    features = feature.hog(image, orientations=9,
                           pixels_per_cell=(10, 10), cells_per_block=(2, 2),
                           transform_sqrt=True, block_norm="L1")
    return features
```

```
def load_split(path):
    # grab the list of images in the input directory, then initialize
    # the list of data (i.e., images) and class labels
    imagePaths = list(paths.list_images(path))
    data = []
    labels = []
    # loop over the image paths
    for imagePath in imagePaths:
        # extract the class label from the filename
        label = imagePath.split(os.path.sep)[-2]
        # load the input image, convert it to grayscale, and resize
        # it to 200x200 pixels, ignoring aspect ratio
        image = cv2.imread(imagePath)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        image = cv2.resize(image, (200, 200))
        # threshold the image such that the drawing appears as white
        # on a black background
        image = cv2.threshold(image, 0, 255,
                              cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
        # quantify the image
        features = quantify_image(image)
        # update the data and labels lists, respectively
        data.append(features)
        labels.append(label)
    return (np.array(data), np.array(labels))
```

Training the Models

```
!pip install imutils
from imutils import paths
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
     Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
def train_models(dataset):
    # initialize the models
    models = {
        "Rf": {
            "classifier": RandomForestClassifier(random_state=1),
            "accuracy": 0,
            "sensitivity": 0,
            "specificity": 0,
        },
        "Xgb": {
            "classifier": XGBClassifier(),
            "accuracy": 0,
            "sensitivity": 0,
            "specificity": 0,
        }
    }
    # define the path to the testing and training directories
    base_path = "/content/drive/MyDrive/Parkinsons/drawings"
    path = os.path.join(base_path, dataset)
    trainingPath = os.path.join('/content/drive/MyDrive/Parkinsons/drawings/spiral', "training")
    testingPath = os.path.join('/content/drive/MyDrive/Parkinsons/drawings/spiral', "testing")
    # load the data
    (trainX, trainY) = load_split(trainingPath)
    (testX, testY) = load_split(testingPath)
    # encode the labels
    le = LabelEncoder()
    trainY = le.fit_transform(trainY)
    testY = le.transform(testY)

    # train each model and calculate its metrics
    for model in models:
        models[model]["classifier"].fit(trainX, trainY)
        predictions = models[model]["classifier"].predict(testX)
        cm = confusion_matrix(testY, predictions).ravel()
        tn, fp, fn, tp = cm
        models[model]["accuracy"] = (tp + tn) / float(cm.sum())
        models[model]["sensitivity"] = tp / float(tp + fn)
        models[model]["specificity"] = tn / float(tn + fp)

    return models
```

```
# Train the models on the spiral drawings
spiralModels = train_models('spiral')
```

```
# train the model on the wave-form drawings
waveModels = train_models('wave')
```

Models Performance

Spiral Drawings

```
print("Random Forrest vs XGBoost Classifier\n\n")
for metric in ("accuracy", "sensitivity", "specificity"):
    print(f"{metric.capitalize()}: ")
    print("Random Forrest={:.2f}%, XGBoost={:.2f}% \n".format(
        spiralModels['Rf'][metric]*100, spiralModels['Xgb'][metric]*100))
```

```
Random Forrest vs XGBoost Classifier


Accuracy:
Random Forrest=80.00%, XGBoost=73.33%

Sensitivity:
Random Forrest=73.33%, XGBoost=73.33%

Specificity:
Random Forrest=86.67%, XGBoost=73.33%
```

Wave Drawings

```
print("Random Forrest vs XGBoost Classifier\n\n")
for metric in ("accuracy", "sensitivity", "specificity"):
    print(f"{metric.capitalize()}: ")
    print("Random Forrest={:.2f}%, XGBoost={:.2f}% \n".format(
        waveModels['Rf'][metric]*100, waveModels['Xgb'][metric]*100))
```

```
Random Forrest vs XGBoost Classifier


Accuracy:
Random Forrest=80.00%, XGBoost=73.33%

Sensitivity:
Random Forrest=73.33%, XGBoost=73.33%

Specificity:
Random Forrest=86.67%, XGBoost=73.33%
```

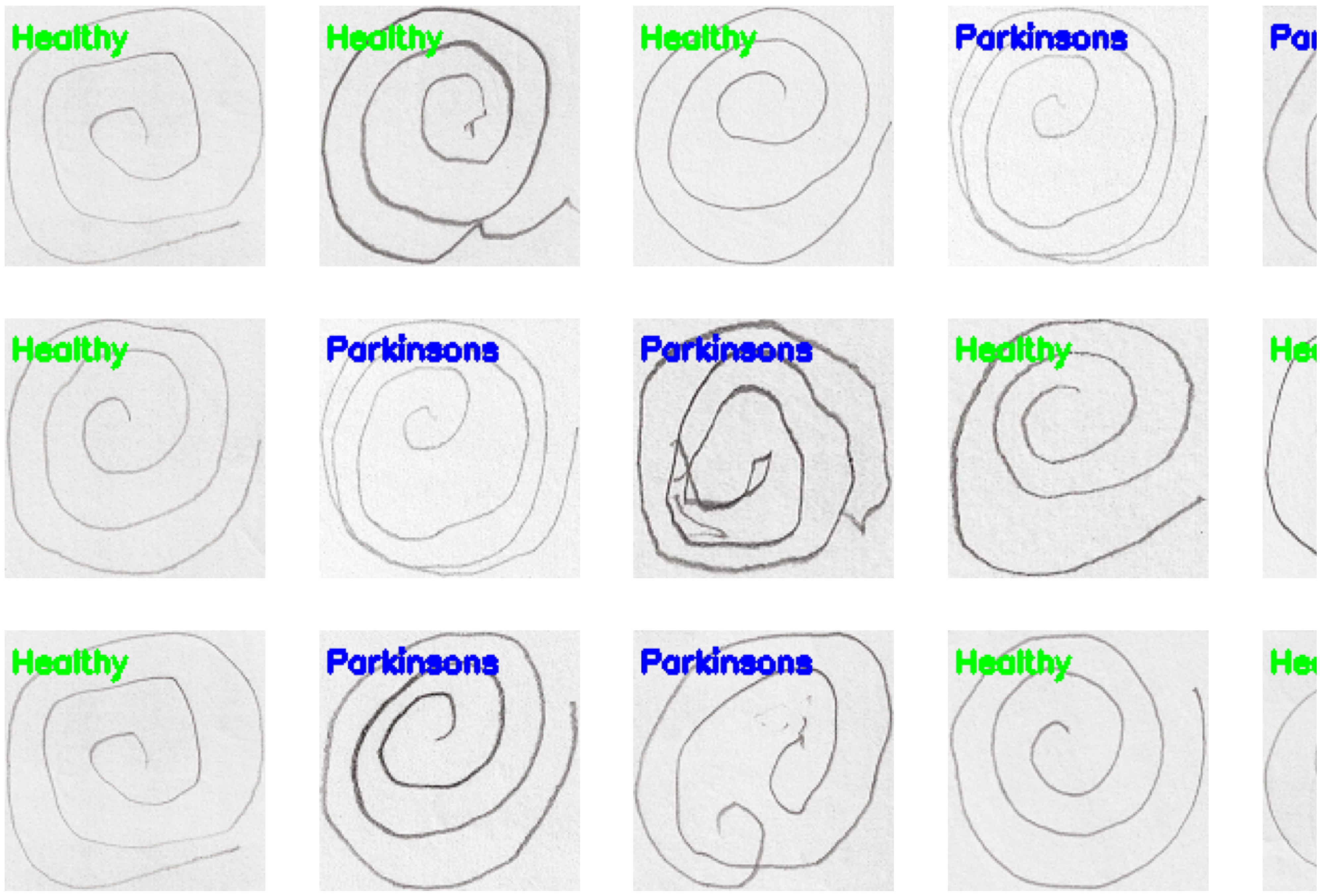
Testing On Sample Images and Visualize the predictions

```
def test_prediction(model, testingPath):
    # get the list of images
    testingPaths = list(paths.list_images(testingPath))
    output_images = []
    # pick 15 images at random
    for _ in range(15):
        image = cv2.imread(random.choice(testingPaths))
        output = image.copy()
        output = cv2.resize(output, (128, 128))
        # pre-process 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
        features = quantify_image(image)
        preds = model.predict([features])
        label = "Parkinsons" if preds[0] else "Healthy"

        # draw the colored class label on the output image and add it to
        # the set of output images
        color = (0, 255, 0) if label == "Healthy" else (0, 0, 255)
        cv2.putText(output, label, (3, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                    color, 2)
        output_images.append(output)
    plt.figure(figsize=(20, 20))
    for i in range(len(output_images)):
        plt.subplot(5, 5, i+1)
        plt.imshow(output_images[i])
        plt.axis("off")
    plt.show()
```

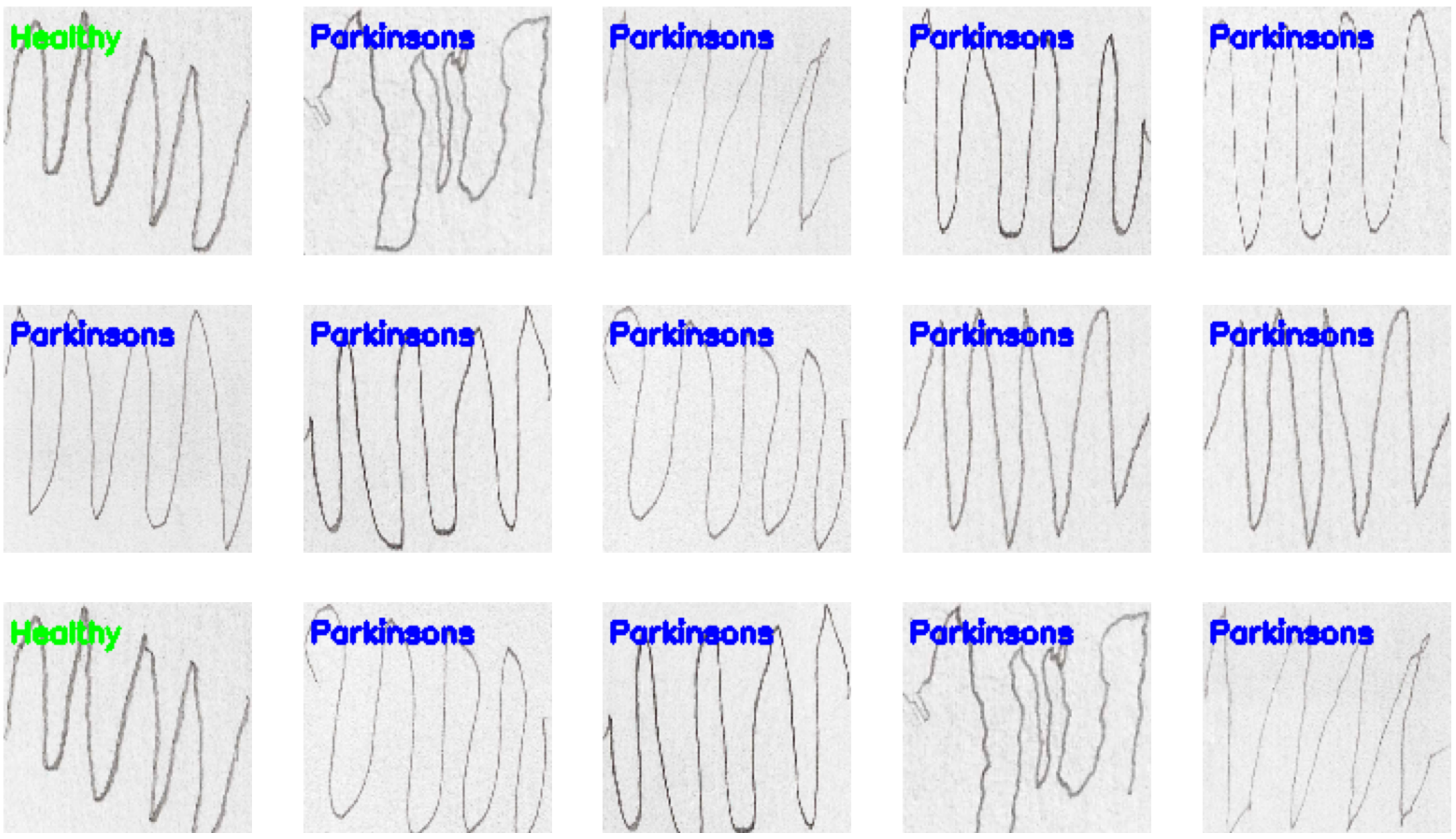
Spiral images

```
testingPath = os.path.sep.join(["/content/drive/MyDrive/Parkinsons/drawings/spiral", "testing"])
test_prediction(spiralModels['Rf']['classifier'], testingPath)
```



Wave Images

```
testingPath = os.path.sep.join(["/content/drive/MyDrive/Parkinsons/drawings/wave", "testing"])
test_prediction(waveModels['Rf']['classifier'], testingPath)
```


Healthy
Parkinsons
Parkinsons
Parkinsons
Parkinsons
Parkinsons
Parkinsons
Parkinsons
Parkinsons
Parkinsons
Healthy
Parkinsons
Parkinsons
Parkinsons
Parkinsons