1 Reference

1. Detecting Parkinson's Disease with OpenCV, Computer Vision, and the Spiral/Wave Test (https://www.pyimagesearch.com/2019/04/29/detecting-parkinsons-disease-with-opencv-computer-vision-and-the-spiral-wave-test/)
2. Distinguishing Different Stages of Parkinson's Disease Using Composite Index of Speed and Pen-Pressure of Sketching a Spiral, by Zham et al. (https://www.frontiersin.org/articles/10.3389/fneur.2017.00435/full), The researchers found that the drawing speed was slower and the pen pressure lower among Parkinson's patients.
3. Images Dataset, NIATS of Federal University of Uberlândia. (http://www.niats.feelt.ufu.br/en/node/81)
4. Histogram of Oriented Gradients for Human Detection (https://ieeexplore.ieee.org/document/1467360), 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.

Parkinson's disease is a nervous system disorder that affects movement. The disease is progressive and is marked by five different stages (source).

- Stage 1: Mild symptoms that do not typically interfere with daily life, including tremors and movement issues on only one side of the body.
- Stage 2: Symptoms continue to become worse with both tremors and rigidity now affecting both sides of the body. Daily tasks become challenging.
- Stage 3: Loss of balance and movements with falls becoming frequent and common. The patient is still capable of (typically) living independently.
- Stage 4: Symptoms become severe and constraining. The patient is unable to live alone and requires help to perform daily activities.
- Stage 5: Likely impossible to walk or stand. The patient is most likely wheelchair bound and may even experience hallucinations.

While Parkinson's cannot be cured, early detection along with proper medication can significantly improve symptoms and quality of life, making it an important topic as computer vision and machine learning practitioners to explore.

2 Introtroduction

A 2017 study by Zham et al. (https://www.frontiersin.org/articles/10.3389/fneur.2017.00435/full) found that it was possible to detect Parkinson's by asking the patient to draw a spiral and then track:

1. Speed of drawing
2. Pen pressure

The researchers found that the drawing speed was slower and the pen pressure lower among Parkinson's patients — this was especially pronounced for patients with a more acute/advanced forms of the disease.

Dataset is availed by Adriano de Oliveira Andrade and Joao Paulo Folado from the NIATS of Federal University of Uberlândia (http://www.niats.feelt.ufu.br/en/node/81), which consists of 204 images and is pre-split into a training set and a testing set, consisting of:

1. Spiral: 102 images, 72 training, and 30 testing
2. Wave: 102 images, 72 training, and 30 testing

In [14]:
```
# import the necessary packages
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
import sklearn.metrics as skm
from sklearn.metrics import confusion_matrix
from skimage import feature,exposure
from imutils import build_montages
from imutils import paths
import numpy as np
from tqdm import tqdm,tqdm_notebook

import cv2
import os
```

In [2]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

3 HOG Algorithm

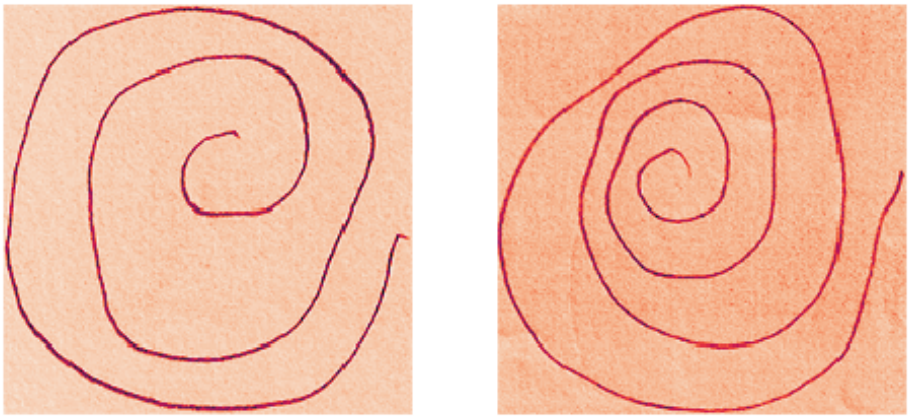
HOG (https://ieeexplore.ieee.org/document/1467360), Histogram of Oriented Gradients, is a structural descriptor that will capture and quantify changes in local gradient in the input image. HOG will naturally be able to quantify how the directions of a both spirals and waves change.

1. (optional) global image normalisation
2. computing the gradient image in x and y
3. computing gradient histograms
4. normalising across blocks
5. flattening into a feature vector

In [2]:
```
feature.hog?
```

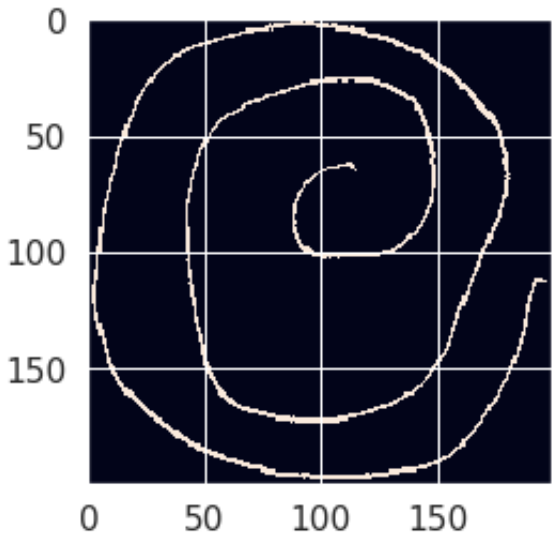
In [80]:
```
def quantify_image(image):
    # compute the histogram of oriented gradients fe
    # the input image
    features = feature.hog(image, orientations=9,
        pixels_per_cell=(10, 10), cells_per_block=(
        transform_sqrt=True, block_norm="L1")

    # return the feature vector
    return features
```

In [81]:
```
fig = plt.figure(figsize=(8, 8))
img1="dataset/spiral/testing/healthy/V01HE01.png"
img2="dataset/spiral/testing/parkinson/V01PE01.png"
img = cv2.imread(img1)
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img = cv2.resize(img, (200, 200))
ax = fig.add_subplot(1, 2, 1, xticks=[], yticks=[])
ax.imshow(img)
img0 = cv2.imread(img2)
img0 = cv2.cvtColor(img0, cv2.COLOR_BGR2GRAY)
img0 = cv2.resize(img0, (200, 200))
ax = fig.add_subplot(1, 2, 2, xticks=[], yticks=[])
ax.imshow(img0)
plt.xticks([]), plt.yticks([]);
```



In [82]:
```
# The thresholding step segments the drawing from the
# making the drawing appear as white foreground on a
image = cv2.threshold(img, 0, 255,
                      cv2.THRESH_BINARY_INV | cv2.THRESH_OTS
plt.imshow(image)
```

Out[82]: <matplotlib.image.AxesImage at 0x1355f1dd8>



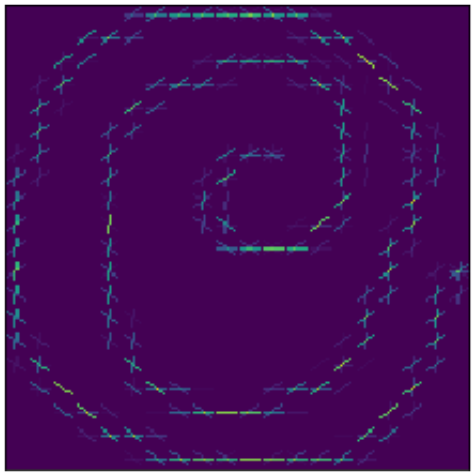
In [83]:
```
# quantify the image by 20736-features
features = quantify_image(image)
len(features)
```

Out[83]: 12996

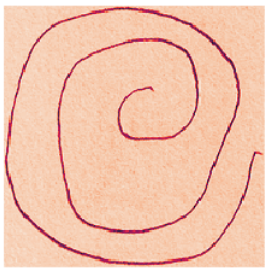
In [85]:
```
features
```

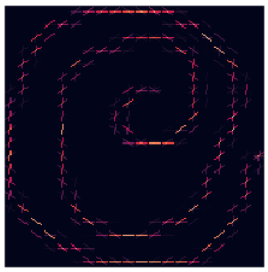
Out[85]: (12996, array([0., 0., 0., ..., 0., 0., 0.]))

In [61]:
```
(H, hogImage) = feature.hog(image, orientations=9, pi
                    cells_per_block=(2, 2), transform_
                    visualise=True)
hogImage = exposure.rescale_intensity(hogImage, out_r
hogImage = hogImage.astype("uint8")

plt.imshow(hogImage)
plt.xticks([]), plt.yticks([]);
```



In [86]:
```
fig = plt.figure(figsize=(14, 4))
ax = fig.add_subplot(1, 3, 1, xticks=[], yticks=[])
ax.imshow(img)
ax = fig.add_subplot(1, 3, 2, xticks=[], yticks=[])
ax.imshow(image)
ax = fig.add_subplot(1, 3, 3, xticks=[], yticks=[])
ax.imshow(hogImage)
plt.xticks([]), plt.yticks([]);
```





In [87]:
```
def load_split(path):
	# grab the list of images in the input directory
	# the list of data (i.e., images) and class labe
	imagePaths = list(paths.list_images(path))
	data = []
	labels = []

	# loop over the image paths
	for imagePath in tqdm(imagePaths):
		# extract the class label from the filename
		label = imagePath.split(os.path.sep)[-2]

		# load the input image, convert it to grays
		# it to 200x200 pixels, ignoring aspect rat
		image = cv2.imread(imagePath)
		image = cv2.cvtColor(image, cv2.COLOR_BGR2G
		image = cv2.resize(image, (200, 200))

		# threshold the image such that the drawing
		# on a black background
		image = cv2.threshold(image, 0, 255,
			cv2.THRESH_BINARY_INV | cv2.THRESH_OTS

		# quantify the image
		features = quantify_image(image)

		# update the data and labels lists, respect
		data.append(features)
		labels.append(label)

	# return the data and labels
	return (np.array(data), np.array(labels))
```

In [88]:
```
# define the path to the training and testing directo
trainingPath ="dataset/spiral/training"
testingPath = "dataset/spiral/testing"
```

In [89]:
```
print("[INFO] loading data...")
(trainX, trainY) = load_split(trainingPath)
(testX, testY) = load_split(testingPath)
```

```
  8%|▉          | 6/72 [00:00<00:01, 55.64it/s]

[INFO] loading data...

100%|██████████| 72/72 [00:01<00:00, 59.03it/s]
100%|██████████| 30/30 [00:00<00:00, 60.38it/s]
```

In [7]:
```
trainX
```

Out[7]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [8]:
```
trainY
```

Out[8]:
```
array(['healthy', 'healthy', 'healthy', 'healthy', 'h
ealthy', 'healthy',
       'healthy', 'healthy', 'healthy', 'healthy', 'h
ealthy', 'healthy',
       'healthy', 'healthy', 'healthy', 'healthy', 'h
ealthy', 'healthy',
       'healthy', 'healthy', 'healthy', 'healthy', 'h
ealthy', 'healthy',
       'healthy', 'healthy', 'healthy', 'healthy', 'h
ealthy', 'healthy',
       'healthy', 'healthy', 'healthy', 'healthy', 'h
ealthy', 'healthy',
       'parkinson', 'parkinson', 'parkinson', 'parkin
son', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkin
son', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkin
son', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkin
son', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkin
son', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkin
son', 'parkinson',
       'parkinson', 'parkinson', 'parkinson', 'parkin
son', 'parkinson',
       'parkinson'], dtype='<U9')
```

In [66]:
```
# encode the labels as integers
le = LabelEncoder()
trainY = le.fit_transform(trainY)
testY = le.transform(testY)

# initialize our trials dictionary
trials = {}
```

4 DecisionTree

In [91]:
```
clf=DecisionTreeClassifier(max_depth=3)
clf.fit(trainX,trainY)
clf
```

Out[91]:
```
DecisionTreeClassifier(class_weight=None, criterion='
gini', max_depth=3,
                       max_features=None, max_leaf_no
des=None,
                       min_impurity_decrease=0.0, min
_impurity_split=None,
                       min_samples_leaf=1, min_sample
s_split=2,
                       min_weight_fraction_leaf=0.0,
presort=False,
                       random_state=None, splitter='b
est')
```

5 Metrics

$$\text{accuracy} = \frac{TP + FN}{\text{all samples}}$$

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

$$\text{f1} = \frac{1}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

In [92]:
```
# Evaluation of the model
print(skm.classification_report(testY,clf.predict(tes
```

```
              precision    recall  f1-score   support

     healthy       0.73      0.73      0.73        15
   parkinson       0.73      0.73      0.73        15

    accuracy                           0.73        30
   macro avg       0.73      0.73      0.73        30
weighted avg       0.73      0.73      0.73        30
```

In [ ]:

In [69]:
```
# loop over the number of trials to run
ntrials=5
for i in tqdm_notebook(range(0, ntrials)):
    # train the model
    print("[INFO] training model {} of {}...".format
    model = RandomForestClassifier(n_estimators=100)
    model.fit(trainX, trainY)

    # make predictions on the testing data and initi
    # to store our computed metrics
    predictions = model.predict(testX)
    metrics = {}

    # compute the confusion matrix and and use it to
    # accuracy, sensitivity, and specificity
    cm = confusion_matrix(testY, predictions).flatte
    (tn, fp, fn, tp) = cm
    metrics["acc"] = (tp + tn) / float(cm.sum())
    metrics["sensitivity"] = tp / float(tp + fn)
    metrics["specificity"] = tn / float(tn + fp)

    # loop over the metrics
    for (k, v) in metrics.items():
        # update the trials dictionary with the lis
        # the current metric
        l = trials.get(k, [])
        l.append(v)
        trials[k] = l
```

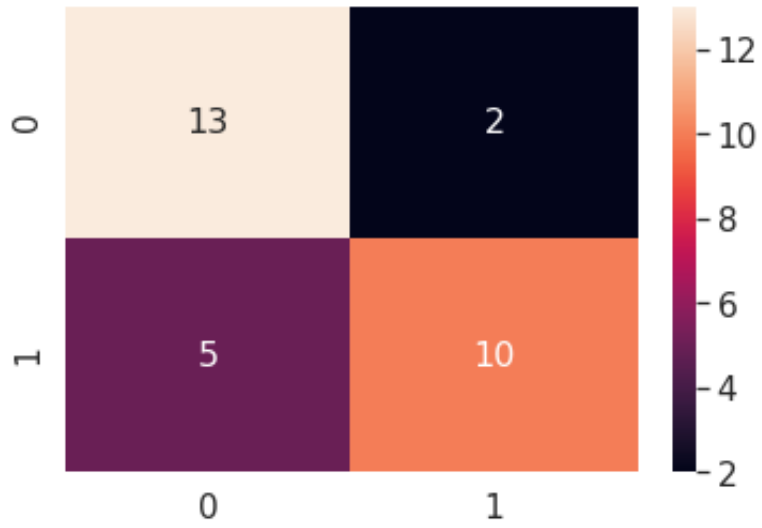
A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
[INFO] training model 1 of 5...
[INFO] training model 2 of 5...
[INFO] training model 3 of 5...
[INFO] training model 4 of 5...
[INFO] training model 5 of 5...
```

In [71]:
```
cmo=confusion_matrix(testY, predictions)
```

In [72]:
```
import seaborn as sns
import pandas as pd
df_cm = pd.DataFrame(cmo, range(2),
                     range(2))
#plt.figure(figsize = (10,7))

sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[72]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1310c8cc0
>
```



In [73]:
```
# loop over our metrics
for metric in ("acc", "sensitivity", "specificity"):
    # grab the list of values for the current metric
    # the mean and standard deviation
    values = trials[metric]
    mean = np.mean(values)
    std = np.std(values)

    # show the computed metrics for the statistic
    print(metric)
    print("=" * len(metric))
    print("μ={:.4f}, σ={:.4f}".format(mean, std))
    print("")
```

```
acc
===
μ=0.8200, σ=0.0452

sensitivity
===========
μ=0.7333, σ=0.0596

specificity
===========
μ=0.9067, σ=0.0327
```

6 New Model of scikit_learn

1. Since 2019/05/17, two new implementations of gradient boosting trees: `ensemble.HistGradientBoostingClassifier` and `ensemble.HistGradientBoostingRegressor` , are supported by `scikit_learning` -0.21.1.

```
# usage
from sklearn.experimental import enable_his
t_gradient_boosting
from sklearn.ensemble import HistGradientBo
ostingRegressor
HistGradientBoostingRegressor(loss='least_s
quares', learning_rate=0.1, max_iter=100, m
ax_leaf_nodes=31, max_depth=None, min_sampl
es_leaf=20, l2_regularization=0.0, max_bins
=256, scoring=None, validation_fraction=0.1
, n_iter_no_change=None, tol=1e-07, verbose
=0, random_state=None)

> pip install -U scikit_learn
```

In [93]:
```
from sklearn.experimental import enable_hist_gradient
from sklearn.ensemble import HistGradientBoostingClas

HGBC=HistGradientBoostingClassifier(learning_rate=0.0
                                    max_depth=3)
HGBC.fit(trainX,trainY)
HGBC
```

Out[93]:
```
HistGradientBoostingClassifier(l2_regularization=0.0,
learning_rate=0.01,
                               loss='auto', max_bins=
256, max_depth=3,
                               max_iter=100, max_leaf
_nodes=31,
                               min_samples_leaf=5, n_
iter_no_change=None,
                               random_state=None, sco
ring=None, tol=1e-07,

validation_fraction=0.1, verbose=0)
```

In [94]:
```
# Evaluation of the model↔
```

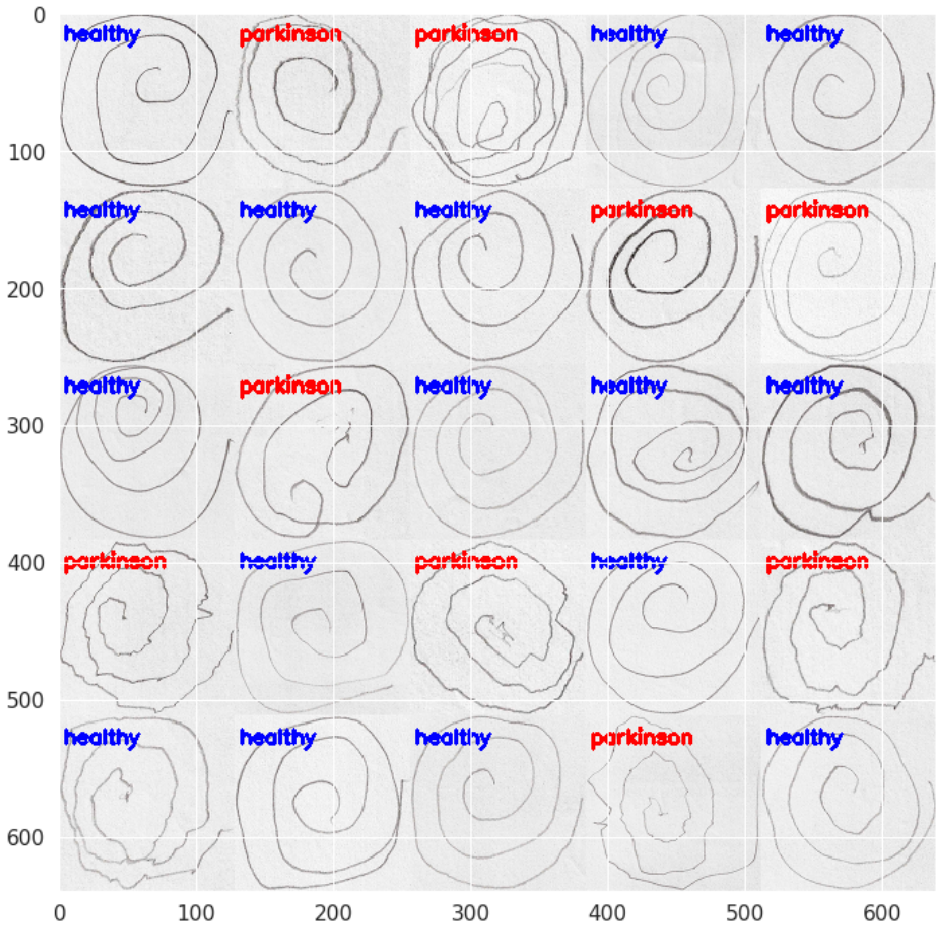
	precision	recall	f1-score	support
healthy	0.83	0.67	0.74	15
parkinson	0.72	0.87	0.79	15
accuracy			0.77	30
macro avg	0.78	0.77	0.76	30
weighted avg	0.78	0.77	0.76	30

In [76]:
```
# randomly select a few images and then initialize th
# for the montage
testingPaths = list(paths.list_images(testingPath))
idxs = np.arange(0, len(testingPaths))
idxs = np.random.choice(idxs, size=(25,), replace=Fal
images = []
```

In [77]:
```
# loop over the testing samples
for i in idxs:
    # load the testing image, clone it, and resize i
    image = cv2.imread(testingPaths[i])
    output = image.copy()
    output = cv2.resize(output, (128, 128))

    # pre-process the image in the same manner we di
    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
    # features using the last trained Random Forest
    features = quantify_image(image)
    preds = model.predict([features])
    label = le.inverse_transform(preds)[0]

    # draw the colored class label on the output ima
    # the set of output images
    color = (255, 0, 0) if label == 'parkinson' else

    cv2.putText(output, label, (3, 20),cv2.FONT_HERS
    images.append(output)
```

In [78]:
```
# create a montage using 128x128 "tiles" with 5 rows
montage = build_montages(images, (128, 128), (5, 5))[

# show the output montage
cv2.imshow("Output", montage)
#cv2.waitKey(0)
```

In [79]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,12))
plt.imshow(montage)

plt.show()
```



In [ ]: