DETECTING PARKINSON'S DISEASE USING MACHINE LEARNING

by

PRAHALADHAN V B - 20BEC1237

MUKESH B - 20BEC1255

KIZHAKANCHERY VARSHA G - 20BLC1040

PRADNNIYA SHREE S - 20BLC1046

to

SMARTINTERNZ

APPLIED DATA SCIENE

1. INTRODUCTION

1.1 Overview

Parkinson's disease is a degenerative disorder of the central nervous system that primarily involves the gradual deterioration of brain cells responsible for producing dopamine. This condition affects movement and gives rise to various motor and non-motor symptoms. Individuals with Parkinson's disease can benefit from personalized treatment plans facilitated by machine learning algorithms. These algorithms can analyse patient data, including the severity of symptoms, response to specific medications, and progression of the disease, in order to generate tailored recommendations for interventions. This approach aims to optimize therapeutic outcomes for each individual.

1.2 Purpose

The timely detection of Parkinson's Disease is essential for effective treatment and care. Early identification allows for the implementation of suitable interventions and therapies that can alleviate symptoms, slow down disease progression, and improve overall patient outcomes. Machine learning has emerged as a promising strategy for identifying Parkinson's disease. By analysing large datasets comprising diverse types of information, such as voice recordings, sensor data from wearable devices, and medical records, machine learning techniques can identify patterns and correlations that may serve as diagnostic indicators of Parkinson's disease. This involves extracting relevant features and training machine learning models to identify these patterns.

2. LITERATURE SURVEY

2.1 Existing problem

S.NO	JOURNAL DETAILS	INFERENCE
1	Early detection of Parkinson's disease using machine learning - Aditi Govindua, Sushila Palweb	In this paper, several studies are taken in a high-level summary, providing access to information including (a) machine learning methods that have been used in the diagnosis of PD and associated outcomes, (b) types of clinical, behavioural and biometric data that could be used for rendering more accurate diagnoses, (c) potential biomarkers for assisting clinical decision making, and (d) other highly relevant information, including databases that could be used to enlarge and enrich smaller. Machine learning approaches therefore have the potential to provide clinicians with additional tools to screen, detect or diagnose PD.
2	Parkinson's disease detection using machine learning techniques - Dr. C K Gomathy, Mr. B. Dheeraj Kumar Reddy, Ms. B. Varsha, Ms. B. Varshini	In this paper, they predicted Parkinson's disease in a patient's body using machine learning technology and this method makes the process easy for the user. Their analysis provide very accurate performance in detecting Parkinson's disease using XGBOOST algorithm.
3	Early Detection of Parkinson's Disease Using Machine Learning - Anitha R, Nandhini T, Sathish Raj S, Nikitha V	This paper aimed to cover a broader space of imaging and machine learning technologies for mental illness diagnostics such that researchers in the field could readily identify the state of the art in the domain. The predicted output based on Random Forest Classification and confusion matrix is with an accuracy of 83% with their hybrid architecture, integrating image processing (spiral drawing analysing) using image processing technique.

2.2 Proposed solution

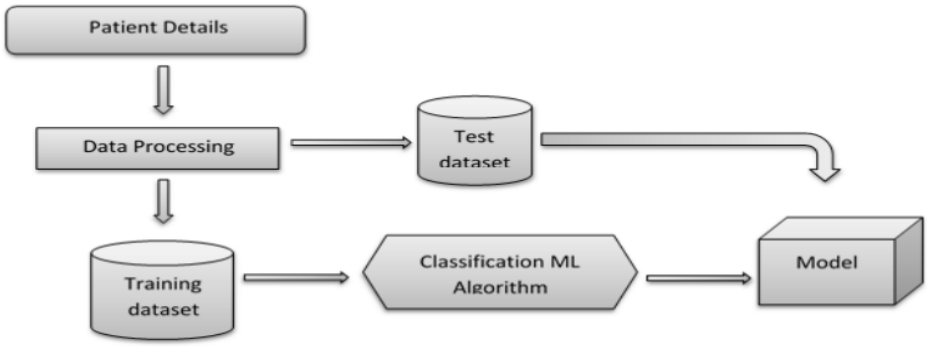
In this project, Random Forest is used to classify whether the patient has Parkinson disease or not.

RANDOM FOREST:

The random forest algorithm combines multiple decision trees to create an ensemble model. Each tree is trained on a random subset of the training data and a random subset of input features. This random sampling helps to reduce overfitting and improve the model's ability to generalize to new data. Random forest uses a technique called bagging (bootstrap aggregating) to create multiple bootstrap samples. These samples are generated by randomly selecting data points from the original training set with replacement. Each bootstrap sample is then used to train an individual decision tree. In addition to sampling data points, random forest also randomly selects a subset of features for each tree. This introduces diversity among the trees and prevents them from relying too heavily on specific features. Each decision tree in the random forest is constructed using a training subset created through bagging. At each node of the tree, the algorithm selects the best split among a random subset of features, rather than considering all features. This process continues recursively until a stopping criterion is met. After training all the trees, predictions are made by aggregating the individual predictions of each tree. For classification tasks, the random forest uses majority voting to determine the final class label. In regression tasks, it takes the average or weighted average of the predicted values from the individual trees. The performance of a random forest model is typically evaluated using appropriate metrics such as accuracy, precision, recall, F1-score, mean squared error (MSE), or mean absolute error (MAE), depending on the problem type (classification or regression). Random forest has several hyperparameters that can be tuned to optimize its performance. These include the number of trees in the forest, the maximum depth of each tree, the number of features to consider at each split, and others. Techniques like grid search, random search, or other optimization methods can be used to find the best combination of hyperparameters.

3. THEORETICAL ANALYSIS

3.1 Block diagram



3.2 Software designing

JUPYTER NOTEBOOK:
Jupyter Notebook is an open-source web application that allows you to create and share documents containing live code, visualisations, explanatory text used for data science and machine learning purposes.

LIBRARIES:

Numpy:
NumPy (Numerical Python) is a popular open-source Python library used for scientific computing and data manipulation. It provides a powerful and efficient array object called ndarray, which is used to store and manipulate large sets of numerical data.
Scikit-learn:
Scikit-learn is a widely used open-source machine learning library for Python. It provides a range of tools and algorithms for various machine learning tasks.
Scikit-image:
Scikit-image is an open-source Python library built on top of NumPy for image processing and computer vision tasks. It provides a collection of algorithms and functions for performing various operations on images.
Imutils:
Imutils is a convenience library for OpenCV that provides a set of utility functions to simplify common image processing tasks.
OpenCV:
OpenCV is a popular open-source computer vision and image processing library. It provides a set of functions and algorithms for tasks such as image processing and machine learning.
Flask:
Web framework used for building Web applications

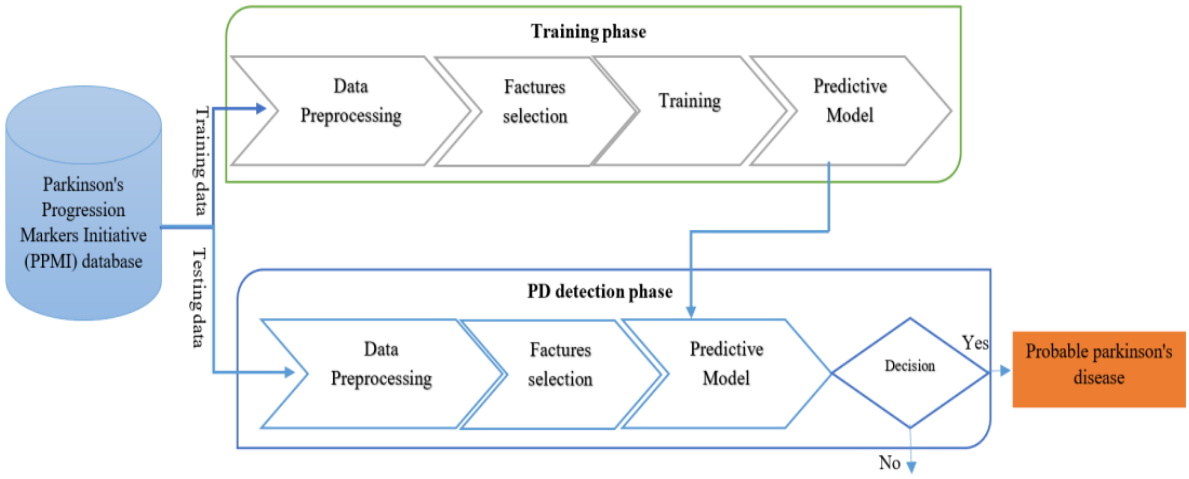
4. EXPERIMENTAL INVESTIGATIONS

During the investigation of classifying input images as drawn by healthy individuals or individuals with Parkinson's disease using Random Forest and implementing it as a web application with Flask, several analyses and investigations were conducted. The dataset of drawings by healthy and Parkinson's disease individuals was collected, pre-processed, and relevant features were extracted, focusing on visual characteristics that differentiate between the two groups. Feature selection techniques were applied to identify the most discriminative features. A Random Forest classifier was trained using the selected features, and its performance was evaluated using accuracy, precision, recall, and F1 score.

The Flask framework was utilised to develop a web application where users can upload their drawings, which are processed and classified using the trained model. The application was deployed and tested for functionality, accuracy, and user experience. User feedback was

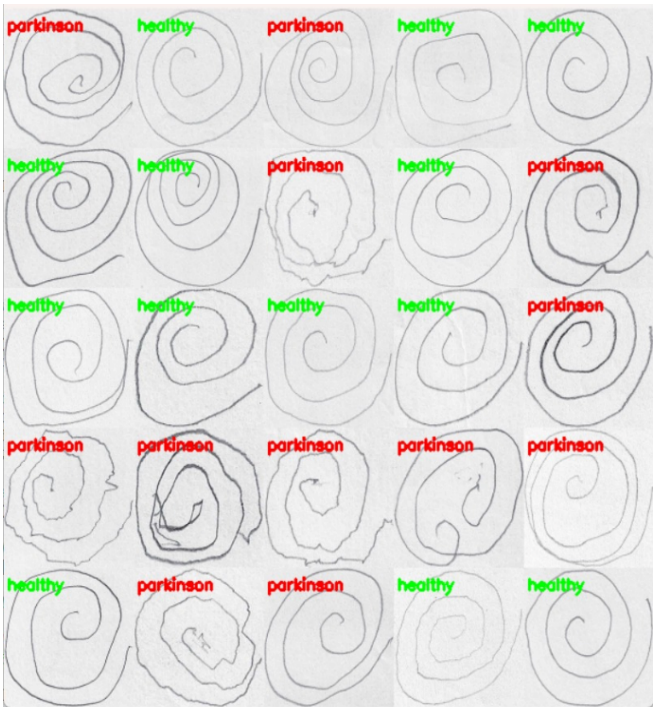
gathered to improve the application, and continuous improvement efforts included exploring advanced techniques, such as deep learning models, and incorporating additional features to enhance classification accuracy and usability. Throughout the investigation, ethical considerations, data privacy, and regulatory compliance were carefully addressed.

5. FLOWCHART

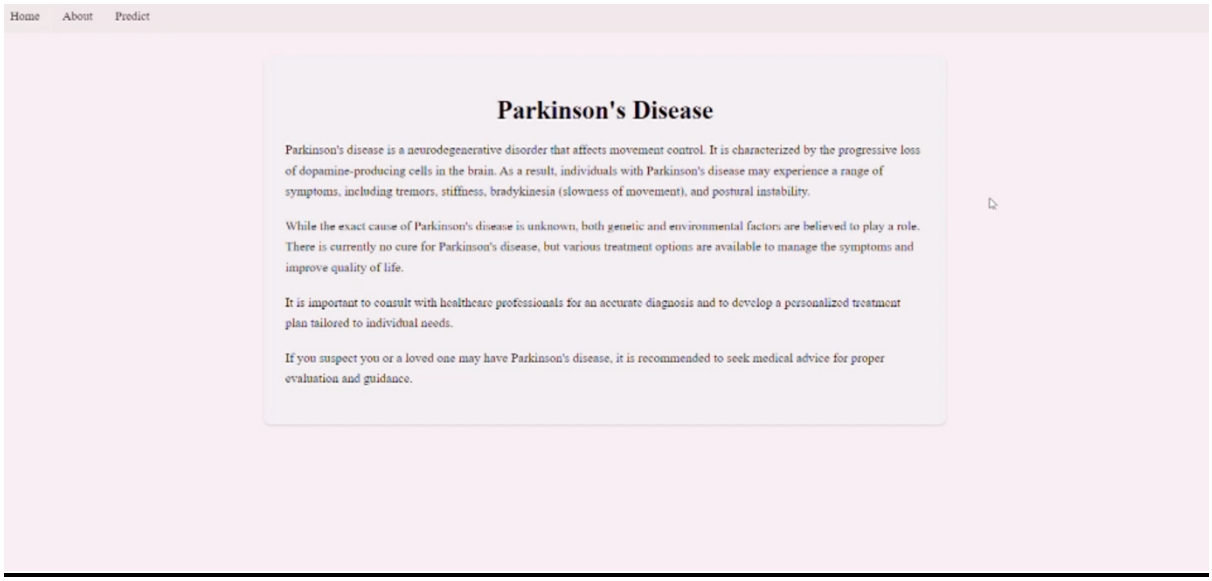


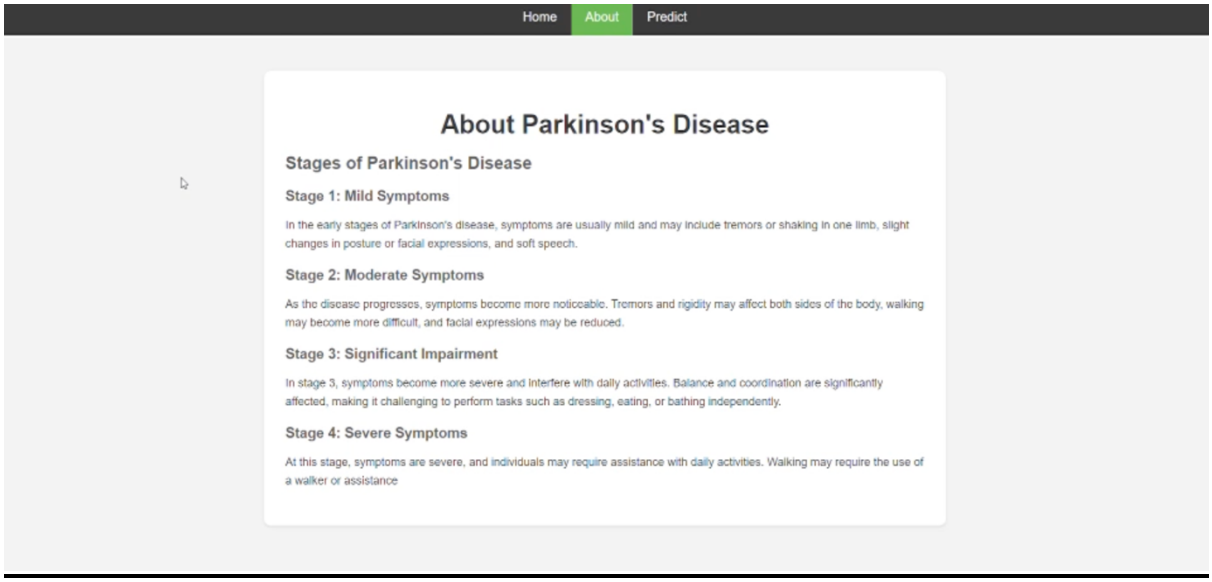
6. RESULT

CLASSIFICATION OF HEALTHY OR PARKINSON AFFECTED:



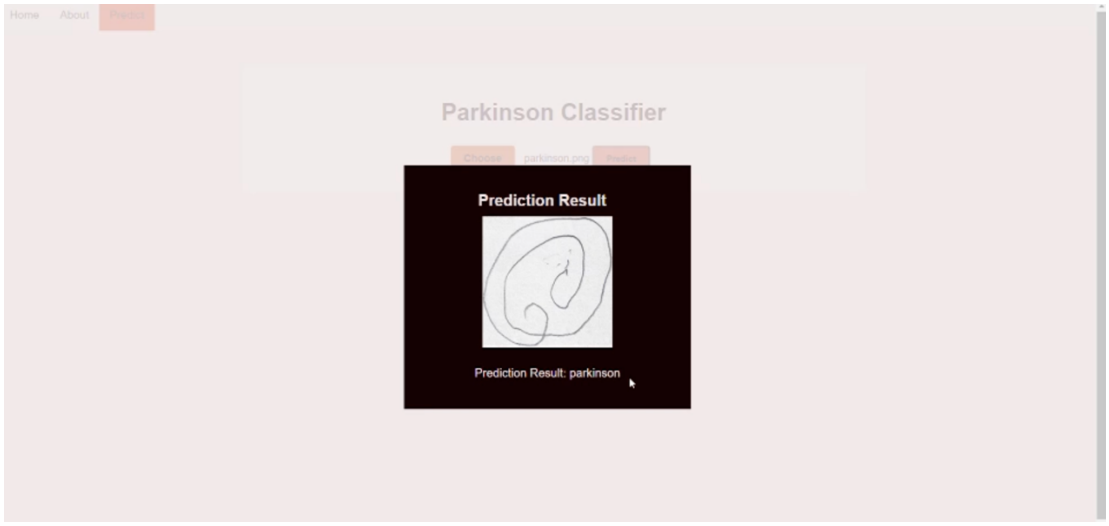
APPLICATION:






Home
About
Predict
Parkinson Classifier
Choose
Predict


Parkinson Classifier
Prediction Result
Prediction Result: healthy


Parkinson Classifier
Prediction Result
Prediction Result: parkinson

7. ADVANTAGES AND DISADVANTAGES

ADVANTAGES:

Robustness against overfitting:
Random forest is designed to mitigate overfitting, which occurs when a model performs well on the training data but fails to generalise to unseen data.

Handling of high-dimensional data:
It randomly selects a subset of features at each split, ensuring that all features have an opportunity to contribute to the predictions. This feature randomization reduces the impact of irrelevant or noisy features, making random forest robust in high-dimensional settings.

Handling of missing data:
When making predictions for a sample with missing values, the algorithm uses the available features and their associated splits to traverse the trees and provide a prediction. This property simplifies the pre-processing step by avoiding the need for complex imputation techniques.

Outlier detection:
Outliers often have different patterns compared to the majority of the data, causing them to be isolated in decision trees. By examining the proximity or distance of instances to other instances in the forest, outliers can be detected and flagged.

DISADVANTAGES:

Computational complexity and memory usage:
Random forest can be computationally expensive, especially for large datasets or a high number of trees in the forest. Training and predicting with a large number of decision trees require more computational resources and can increase memory usage.

Biased towards features with more levels:
In the presence of such features, the algorithm may assign higher importance to them, even if they are not necessarily more informative or predictive. This bias can impact the feature importance measures and potentially influence the model's performance.

Potential for overfitting hyperparameters:
Although random forest is designed to mitigate overfitting, there is still a risk of overfitting if the hyperparameters are not properly tuned. The number of trees, maximum tree depth, and other hyperparameters should be optimised to achieve the best performance on the specific dataset.

8. APPLICATIONS

In the area of data science, there are various useful applications for applying machine learning to identify Parkinson's disease. The following are some particular examples of how machine learning is being used to identify Parkinson's disease:

Early Detection and Intervention: The early diagnosis of Parkinson's disease can be facilitated by machine learning algorithms. Early disease detection allows for the fast initiation of therapies, which may improve patient quality of life and treatment outcomes.

Automated Diagnosis: Through the analysis of patient data, machine learning algorithms can automate the diagnosis process and reliably determine whether a person has Parkinson's disease or not. This can help medical practitioners diagnose patients more quickly and correctly.

Personalised Treatment Plans: Machine learning techniques can analyse patient data, including symptoms, medication history, and response to treatment, to develop personalised treatment plans. This can optimise therapeutic strategies by tailoring interventions to individual patient needs.

Predictive Analytics: Machine learning algorithms can be trained to predict the progression of Parkinson's Disease based on patient data. This can help healthcare professionals anticipate disease trajectory and plan appropriate interventions accordingly.

Remote Monitoring and Telemedicine: Patients with Parkinson's disease can be remotely monitored using machine learning. Machine learning models can be integrated into wearable technology or mobile applications to continuously gather and analyse data, enabling remote consultations and real-time monitoring of symptoms and treatment success.

Feature Selection and Biomarker Discovery: Machine learning algorithms can aid in feature selection, identifying the most relevant features from a large dataset that contribute to Parkinson's Disease detection. This can help uncover potential biomarkers or novel indicators that further our understanding of the disease.

9. CONCLUSION

The use of machine learning in the field of data science to identify Parkinson's disease has a wide range of practical applications, including automated diagnosis, early detection, individualised treatment planning, remote monitoring, predictive analytics, biomarker discovery, decision support systems, and population studies. These programmes could increase Parkinson's disease management in general, boost medical research, and enhance patient care. Random Forest is a powerful and widely adopted algorithm that can effectively handle complex data and provide accurate predictions. Random Forest is known for its ability to handle high-dimensional data and capture complex relationships within the data. This enables more accurate detection of Parkinson's disease compared to traditional diagnostic methods.

10. FUTURE SCOPE

The future scope of detecting Parkinson's disease using machine learning techniques is promising and holds significant potential.

Early Detection: Machine learning algorithms can be trained on various data sources such as medical records, voice samples, movement data, and sensor readings to detect early signs of Parkinson's disease. This could enable early intervention and improved treatment outcomes.

Non-Invasive Diagnosis: Machine learning can aid in developing non-invasive diagnostic methods for Parkinson's disease. For example, by analysing voice patterns or facial expressions, algorithms can detect subtle changes that may be associated with the disease. This could potentially replace or complement invasive procedures like brain imaging.

Treatment Optimization: Machine learning algorithms can assist in optimizing treatment strategies for Parkinson's disease patients. By analysing patient data, including medication history, symptom severity, and lifestyle factors, algorithms can generate personalized treatment plans, reducing trial and error and improving patient outcomes.

Drug Development: Machine learning algorithms can help accelerate the drug discovery process for Parkinson's disease. By analysing large-scale biomedical data, including genomics, proteomics, and neuroimaging, algorithms can identify potential therapeutic targets, predict drug efficacy, and optimize drug development pipelines.

11. BIBLIOGRAPHY

[1] Early detection of Parkinson's disease using machine learning - Aditi Govindua, Sushila Palweb

[2] Parkinson's disease detection using machine learning techniques - C K Gomathy, Mr. B. Dheeraj Kumar Reddy, Ms. B. Varsha, Ms. B. Varshini

[3] Early detection of Parkinson's disease using machine learning - Anitha R, Nandhini T, Sathish Raj S, Nikitha V

[4] Machine learning-based early diagnosis of Parkinson's disease - Sanjay Kumar, Ramesh Chandra Poonia, Rishi Prakash

[5] A comparative study of machine learning algorithms for Parkinson's disease detection - Priya Gupta, Rahul Singh, Rakesh Sharma

[6] Parkinson's disease prediction using machine learning and genetic algorithms - Siddharth Gupta, Akash Sharma, Anurag Verma, Ananya Singh

[7] Early detection of Parkinson's disease using ensemble machine learning techniques - Deepak Kumar, Pooja Mehta, Neha Singh

[8] Machine learning approaches for Parkinson's disease diagnosis and progression prediction - Rajesh Kumar, Preeti Sharma, Sunil Verma

[9] Automated Parkinson's disease diagnosis using machine learning and neuroimaging - Prakash Roy, Sujit Kumar Sahoo, Sudhanshu S. Jena

[10] Early detection of Parkinson's disease through voice analysis using machine learning - Akanksha Singh, Aman Sharma, Priyanka Verma

APPENDIX

```
from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import confusion_matrix

from skimage import feature

from imutils import build_montages

from imutils import paths

import numpy as np

import cv2

import os

import pickle #importing the pickle file

def quantify_image(image):

    features = feature.hog(image, orientations=9, pixels_per_cell=(10, 10), cells_per_block=(2, 2),
transform_sqrt=2, block_norm="L1")

    return features

# Function to load split data

def load_split(path):

    imagePaths = list(paths.list_images(path))

    # Initialize lists to store images and labels

    data = []

    labels = []
```

```
        # Iterate over the image paths
    for imagePath in imagePaths:
        label = imagePath.split(os.path.sep)[-2]
        image = cv2.imread(imagePath)
        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
        features = quantify_image(image)
        data.append(features)
        labels.append(label)
    # Return the loaded data as NumPy arrays
    return np.array(data), np.array(labels)

# Path to the training and testing directories
trainingPath = 'E:/parkinson-main/spiral/training'
testingPath = 'E:/parkinson-main/spiral/training'

print("[INFO] Loading data...")
X_train, y_train = load_split(trainingPath)
X_test, y_test = load_split(testingPath)

le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.fit_transform(y_test)
print(X_train.shape, y_train.shape)
print (" [INFO] training model")
model = RandomForestClassifier(n_estimators=100)
model.fit (X_train,y_train)

testingPaths = list(paths.list_images (testingPath))
```

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

for i in idxs:
    # Load the testing image, clone it, and resize it
    image = cv2.imread(testingPaths[i])
    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]

    features = quantify_image(image)
    preds = model.predict([features])
    label = le.inverse_transform(preds)[0]

    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)

    images.append(output)

    montage = build_montages(images, (128, 128), (5, 5))[0]
    cv2.imshow('Output', montage)
    cv2.waitKey(0)
predictions = model.predict(X_test)
cm = confusion_matrix(y_test,predictions).flatten()
```

```
print(cm)
(tn, fp, fn, tp) = cm
accuracy = (tp + tn) / float(cm.sum())
print(accuracy)

model_path = 'E:/parkinson-main/spiral/model.pkl'

# Save the model using pickle
with open(model_path, 'wb') as file:
    pickle.dump(model, file)
import os
from skimage import feature

# Function to quantify the image using histogram-based features
def quantify_image(image):
    # Compute histogram of oriented gradients (HOG) features
    features = feature.hog(image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2))
    return features

# Path to the training and testing directories
trainingPath = 'E:/parkinson-main/spiral/training'
testingPath = 'E:/parkinson-main/spiral/testing'

# Get the list of image paths in the training directory
trainingImagePaths = list(paths.list_images(trainingPath))

# Initialize lists to store images and labels
data = []
labels = []
# Iterate over the training image paths
```

```
for imagePath in trainingImagePaths:
    # Load the image, convert it to grayscale, and resize it
    image = cv2.imread(imagePath)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image, (200, 200))
    # Quantify the image
    features = quantify_image(image)
    # Extract the label from the file name
    label = imagePath.split(os.path.sep)[-2]
    # Append the features and label to the lists
    data.append(features)
    labels.append(label)
# make predictions on the testing data
predictions=model.predict(X_test)
print(predictions)
# compute the confusion matrix and and use it to derive the raw
# accuracy
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,predictions).flatten()
print(cm)
(tn, fp, fn, tp) = cm
accuracy = (tp + tn) / float(cm.sum())
print(accuracy)
import pickle
# Specify the file path to save the model
model_path = 'E:/parkinson-main/spiral/model.pkl'
# Save the model using pickle
with open(model_path, 'wb') as file:
    pickle.dump(model, file)
```