OPENCV VOICE PATHOLOGY DETECTION COMPETITION

Unveiling the power of image processing and SVM

Alessia Simone

11 - 29 - 2023

Contents

1 Introduction

Note: This project aims to be an instrument to help doctors diagnose voice pathology and for future developments. It won't in any way substitute medical advice. If you are in doubt about your health status please refer to a doctor, just use this application for test purposes only.

This project aims to identify different types of pathology based on a few seconds of vowel sound. In particular, unhealthy voice sounds can be dysphonia and laryngitis pathologies.

This project aims to build an application capable of collecting 5 seconds of "a" vowel sounds with an embedded device, preprocessing it to highlight the most important features which will help the AI algorithm to detect the response. As I will feed the AI algorithm to train it, I needed a completed and well-structured dataset to let the algorithm learn the specific behaviour of the problem.

For this purpose, I used the VOICED dataset[1], containing 208 voice samples (150 pathological, 58 healthy), and the signals consist of 5 seconds of vowel 'a' vocalization without interruption.

The signals have been preprocessed, and a Support Vector Machine has been applied for the binary classification:

- A Butterworth filter[2] has been applied to select only voice frequency,
- A Fast Fourier Transformation[3] has been applied to clean the sound from background noise, as reported on the original data preprocessing,
- Each signal of the dataset has been converted into an image with the Gramian Angular Field technique[4], producing grey images,
- Finally, a Histogram of Oriented Gradients[5] has been applied as a feature extractor

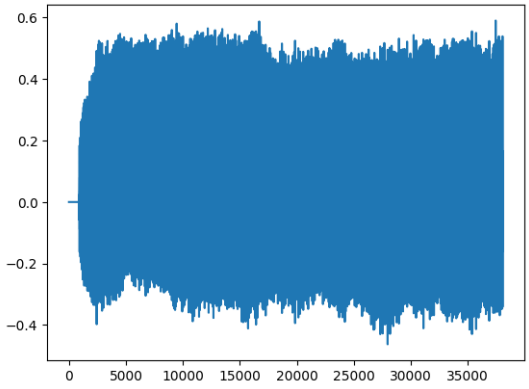

Figure 1: Dysphonia voice signal

The plot above represents the five-second signal of "a" vowel of subject 1: the subject was a 32-year-old researcher with hyperkinetic dysphonia (VHI: 15, RSI:5).

As we can see and as reported in the original dataset description, the signals have been filtered and cleaned from background noise.

In order to use one of the most powerful tools from the OpenCV library to extract features from images, I need to transform the sequence into images maintaining the requested details.

Then, the Gramian Angular Field technique was used to transform the signal sequence into an image, representing the correlation of each time step with past and feature points. For this purpose, the summation GAF image has been applied, resulting in a single channel image of shape 128 x 128.

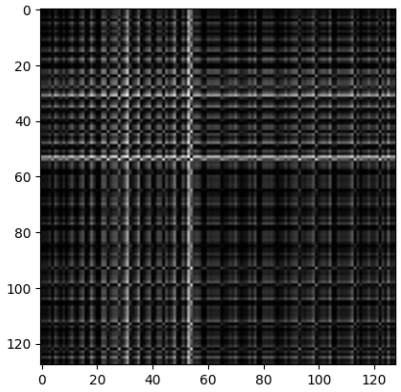

Figure 2: Summation Gramian Angular Field image

In order to detect if the 5-second signal-to-image is healthy or pathological, I will apply a Histogram of Oriented Gradients as a features extractor to detect the meaningful part of the image with the following hyperparameters:

- A window size of 128 x 128 to match the entire image,
- A cell block of size 2 x 2 has 4 pixels in each cell,
- A block of size 8 x 8 has 64 cells in each block,
- A stride of size 1 to move across all the cells in order to capture as many details as possible,
- 18 bins for the histogram

The plot above represents the magnitude and direction of the gradients for each block, highlighting the most important features.

In particular, the HOG extractor divides the image into blocks, in which histograms of oriented gradients are computed. It highlights two important components throw arrows which represent gradients (weights of each pixel on the image compared to the whole image):

- The orientation of the local gradients on the images in terms of luminosity and colours helps to identify the structure of the image,
- The magnitude of the gradients in that direction is represented throw the length of the arrow

Those extracted features are read as numbers by the AI algorithm, which will learn from those features to identify the differences between healthy and pathological voices.

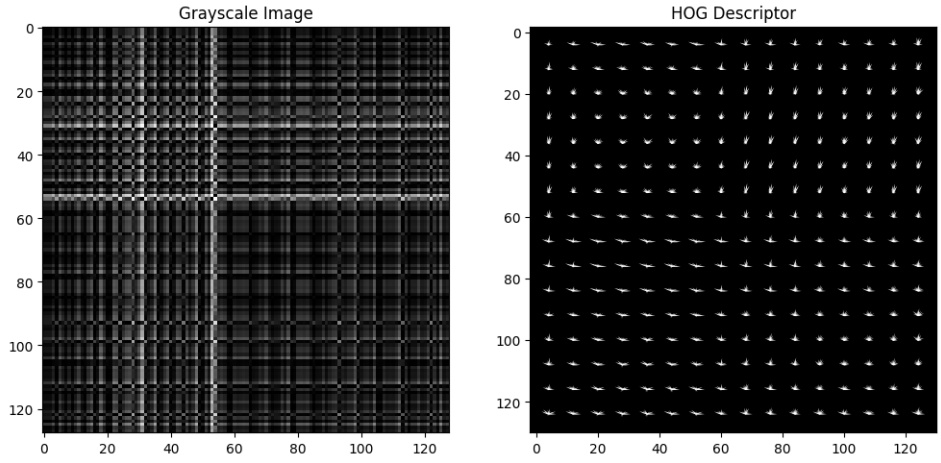

Figure 3: HOG Visualization

2 Preprocessing

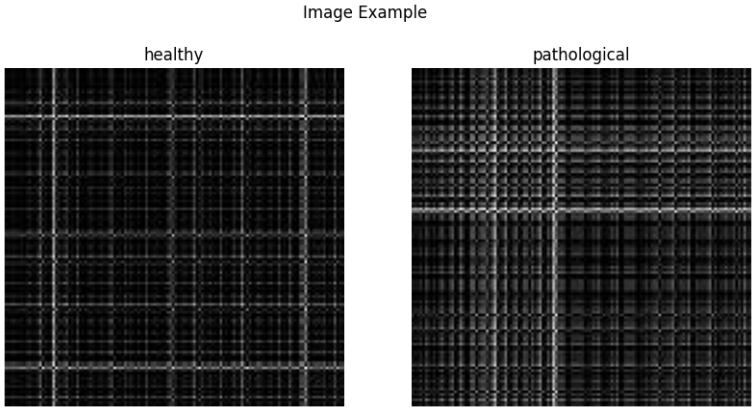

Figure 4: Healthy vs Pathological images

Here is the example of the final image dataset: on the left, there is a randomly picked healthy image and on the right, there is a randomly picked pathological image.

As the original dataset was quite imbalanced, with 55 healthy signals and 150 pathological signals, I needed to balance the images to allow the model to avoid learning higher features from a class only.

One of the most famous ways to augment images is to apply various techniques, like random cropping, horizontal and vertical flipping and rotation, in order to create various types of images without losing any type of information.

In particular, a set of geometric transformations are applied to change the position, size and orientation of the image:

- rotation of 30°,
- shearing the image on its axis by 2,
- translate the image on its axis by 5.

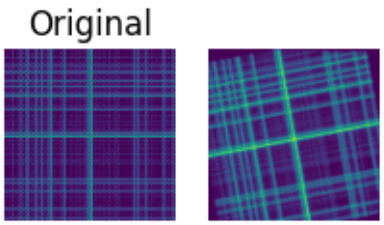

Figure 5: Original vs Augmented Healthy images

3 Training

At this point, the dataset is ready to feed the Support Vector Machine algorithm, which will analyze the extracted features and predict the response:

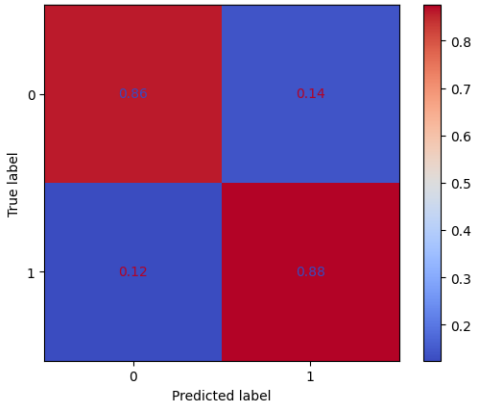

Figure 6: Confusion Matrix

The plot above represents the confusion matrix, which compares the true and predicted labels. As we can see, 86% of corrected healthy images were predicted against 88% of corrected pathological images.

The model now is ready to be deployed!

4 Appendix

4.1 Notebook

```
import warnings
warnings.filterwarnings('ignore')
import os
import glob
from tqdm import tqdm
from PIL import Image
import pandas as pd
import numpy as np
#from scipy import signal
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from pyts.image import GramianAngularField
import cv2                                          #<- OPENCV HERE!
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import GridSearchCV
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV, StratifiedKFold
from sklearn.svm import SVC
from sklearn.metrics import classification_report, ConfusionMatrixDisplay
from sklearn.model_selection import cross_val_predict
from joblib import dump

df1 = pd.read_csv("path",  delimiter='\t')
plt.plot(df1)
plt.show()

signal1 = df1.values
scaler = StandardScaler(with_std=False)
scaled = scaler.fit_transform(signal1)
plt.plot(scaled)
plt.show()

dump(scaler, 'C:/Users/aless/Desktop/opencv-competition/scaler.joblib')

signal2 = np.transpose(signal1)
gasf = GramianAngularField(method='summation', image_size=128)
img = gasf.transform(signal2)

#Normalizing the image in range [0, 1]
min_val = np.min(img)
max_val = np.max(img)
```

```
img = 255 * ((img - min_val) / (max_val - min_val))
img = img.astype(np.uint8)
img = img.transpose(1,2,0)
plt.imshow(img, cmap='gray')
plt.show()

#image = cv2.resize(img, (128, 128))
win_size = (128, 128)
cell_size = (8, 8)
block_size = (16, 16)
block_stride = (8, 8)
num_bins = 9
hog = cv2.HOGDescriptor(win_size, block_size, block_stride, cell_size, num_bins)
features = hog.compute(img)
print(features.shape)

hog_descriptor_reshaped = features.reshape(15, 15, 2, 2, 9).transpose((1, 0, 2, 3, 4

# Create an array that will hold the average gradients for each cell
ave_grad = np.zeros((16, 16, 9))
# Create an array that will count the number of histograms per cell
hist_counter = np.zeros((16, 16, 1))

# Add up all the histograms for each cell and count the number of histograms per cel
for i in range(2):
for j in range(2):
ave_grad[i:15 + i,
j:15 + j] += hog_descriptor_reshaped[:, :, i, j, :]
hist_counter[i:15 + i,
j:15 + j] += 1

# Calculate the average gradient for each cell
ave_grad /= hist_counter

# Calculate the total number of vectors we have in all the cells.
len_vecs = ave_grad.shape[0] * ave_grad.shape[1] * ave_grad.shape[2]

# Create an array that has num_bins equally spaced between 0 and 180 degress in radi
deg = np.linspace(0, np.pi, num_bins, endpoint=False)

# Each cell will have a histogram with num_bins. For each cell, plot each bin as a v
# equal to the height of the bin in the histogram, and its angle corresponding to th
# To do this, create rank 1 arrays that will hold the (x,y)-coordinate of all the ve
# image. Also, create the rank 1 arrays that will hold all the (U,V)-components of a
# cells in the image. Create the arrays that will hold all the vector positons and c
```

```
U = np.zeros((len_vecs))
V = np.zeros((len_vecs))
X = np.zeros((len_vecs))
Y = np.zeros((len_vecs))

# Set the counter to zero
counter = 0

# Use the cosine and sine functions to calculate the vector components (U,V) from th
# cosine and sine functions take angles in radians. Calculate the vector positions a
# average gradient array
for i in range(ave_grad.shape[0]):
for j in range(ave_grad.shape[1]):
for k in range(ave_grad.shape[2]):
U[counter] = ave_grad[i, j, k] * np.cos(deg[k])
V[counter] = ave_grad[i, j, k] * np.sin(deg[k])
X[counter] = (cell_size[0] / 2) + (cell_size[0] * i)
Y[counter] = (cell_size[1] / 2) + (cell_size[1] * j)
counter = counter + 1
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Display the image
ax1.set(title='Grayscale Image')
ax1.imshow(img, cmap='gray')

# Plot the feature vector (HOG Descriptor)
ax2.set(title='HOG Descriptor')
ax2.quiver(Y, X, U, V, color='white', headwidth=0, headlength=0, scale_units='inches
ax2.invert_yaxis()
ax2.set_aspect(aspect=1)
ax2.set_facecolor('black')

source_folder = "path1"
target_folder = "path2"

# Create the target folder if it doesn't exist
os.makedirs(target_folder, exist_ok=True)

# List all files in the source folder
file_list = os.listdir(source_folder)

for filename in tqdm(file_list, desc="Processing files"):
    if filename.endswith('.txt'):
        # Read the data from the file
        file_path = os.path.join(source_folder, filename)
```

```
        data = pd.read_csv(file_path,  delimiter='\t')

        array = data.values
        array = np.transpose(array)

        gasf = GramianAngularField(method='summation', image_size=128)
        img = gasf.transform(array)

        # Save the produced image in the target folder as jpg
        target_path = os.path.join(target_folder, filename.replace('.txt', '.jpg'))

        # Convert numpy array to PIL Image and save as jpg
        img = img.reshape((128, 128))  # Reshape to (128 x 128)
        img = (img - img.min()) / (img.max() - img.min()) * 255 #ensure range [0, 25
        img = img.astype(np.uint8)
        pil_img = Image.fromarray(img, mode='L')  # 'L' mode for grayscale
        pil_img.save(target_path)

print('Folders inside the main directory:', os.listdir("path"))

classes = os.listdir("path")
sum_images = []
for i in classes:
    img_base_path = "path" + str(i)
    images = os.listdir(img_base_path)
    sum_images.append(len(images))
plt.bar(classes, sum_images)
plt.title('Distribution of classes')
plt.show()

images = []
paths = []

fig,ax = plt.subplots(1,2,figsize = (10, 5))
ax = ax.ravel()

for idx, i in enumerate(classes):
    img_base_path = "path" + str(i) #link of the subfolders
    timages = os.listdir(img_base_path) #list of images inside each subfolders
    images_path_to_display = os.path.join(img_base_path, str(timages[0]))
    paths.append(images_path_to_display)
    img = Image.open(images_path_to_display)
    images.append(img)
    image = images[idx]
    ax[idx].axis('off')
```

```
    ax[idx].imshow(image,cmap = 'gray')
    ax[idx].set_title(str(i))

plt.suptitle('Image Example')
plt.show()

path = 'path'

neg_imgs = glob.glob(path + '/*.jpg')

print(len(neg_imgs))

def affine_transform(img, ang_max, shear_max, trans_max):
    '''
    Apply random affine transformations on an image.
    Credits: https://medium.com/@dnemutlu/hog-feature-descriptor-263313c3b40d

    Args:
        img: 1 image array with 3 channels.
        ang_max: integer value to represent max and min values for random
                 angle rotations.
        shear_max: integer value to represent max and min values for random
                   sheering.
        trans_max: integer value to represent max and min values for random
                   translation.

    '''


    rows = img.shape[0]
    cols = img.shape[1]

    # rotation
    rot = np.random.uniform(ang_max) - ang_max / 2
    rot_matrix = cv2.getRotationMatrix2D((cols / 2, rows / 2), rot , 1)

    # shearing
    pts1 = np.float32([[ 10, 10], [20, 10], [10, 20]])
    pt1 = 10 + shear_max * np.random.uniform() - shear_max / 2
    pt2 = 20 + shear_max * np.random.uniform() - shear_max / 2
    pts2 = np.float32([[pt1, 10], [pt2, pt1], [10, pt2]])
    shear_matrix = cv2.getAffineTransform(pts1, pts2)

    # translation
    trans_x = trans_max * np.random.uniform() - trans_max / 2
    trans_y = trans_max * np.random.uniform() - trans_max / 2
```

```
    trans_matrix = np.float32([[1  , 0, trans_x], [0  , 1, trans_y]])

    # affine transformations
    img = cv2.warpAffine(img, rot_matrix,(cols, rows))
    img = cv2.warpAffine(img, trans_matrix,(cols, rows))
    img = cv2.warpAffine(img, shear_matrix,(cols, rows))

    return img

# augment by affine transformations
aug_images = []
for i,path in enumerate(neg_imgs):
    # read image
    img = mpimg.imread(path)
    # append original image
    aug_images.append(img)
    # apply random transformations for 5 times
    for j in range(1):
        # transform
        trans_img = affine_transform(img, 30, 2, 5)
        # append to list
        aug_images.append(trans_img)
    if i%10 == 0:
        plt.figure(figsize=(8,4))
        plt.subplot(1,6,1)
        plt.imshow(img)
        plt.axis('off')
        plt.title('Original')
        for k in range(2,3):
            plt.subplot(1,6,k)
            plt.imshow(aug_images[-1*(k-1)])
            plt.axis('off')
        plt.show()
        plt.close()

for i, img in enumerate(aug_images):
    pil_img = Image.fromarray(img, mode='L')  # 'L' mode for grayscale
    new_path = os.path.join(path, str(i) + '.jpg')
    pil_img.save(new_path)

# Specify the parameters for our HOG descriptor
win_size = (128, 128)
cell_size = (2, 2)
block_size = (2, 2)
block_stride = (1, 1)
```

```
num_bins = 18

# Set the parameters of the HOG descriptor using the variables defined above
hog = cv2.HOGDescriptor(win_size, block_size, block_stride, cell_size, num_bins)

# Initialize variables
img_folder_1 = 'C:\\Users\\aless\\Desktop\\opencv-competition\\imgs\\healthy'
img_folder_2 = 'C:\\Users\\aless\\Desktop\\opencv-competition\\imgs\\pathological'

imgs = []
labels = []

# Process images in folder 1
for img_name in os.listdir(img_folder_1):
    img_path = os.path.join(img_folder_1, img_name)
    img = cv2.imread(img_path)
    img = cv2.resize(img, (128, 128))
    if img is not None:
        img = hog.compute(img)
        imgs.append(img)
        labels.append(0)

# Process images in folder 2
for img_name in os.listdir(img_folder_2):
    img_path = os.path.join(img_folder_2, img_name)
    img = cv2.imread(img_path)
    img = cv2.resize(img, (128, 128))
    if img is not None:
        img = hog.compute(img)
        imgs.append(img)
        labels.append(1)

# Convert the lists to NumPy arrays
imgs = np.array(imgs)
labels = np.array(labels)

# Print the shape of the arrays to verify
print("Images shape:", imgs.shape)
print("Labels shape:", labels.shape)

scaler = MinMaxScaler()
x = scaler.fit_transform(imgs)

dump(scaler, 'C:/Users/aless/Desktop/opencv-competition/scaler2.joblib')
```

```
clf = SVC(random_state=46, kernel='linear', gamma = 0.0)
cv = StratifiedKFold(shuffle=True, random_state=42)
param_grid = [{'C': [0.0001, 0.001, 0.01, 1, 10, 100]}]
search = HalvingGridSearchCV(clf, param_grid, cv = cv, random_state=42, scoring = 'f1
print(search.best_params_)
print(search.best_score_)

clf = search.best_estimator_
y_pred = cross_val_predict(clf, x, labels, cv=cv)
print(classification_report(labels, y_pred))

ConfusionMatrixDisplay.from_predictions(labels, y_pred, cmap='coolwarm', normalize =
plt.show()

dump(clf, 'C:/Users/aless/Desktop/opencv-competition/clf.joblib')
```

4.2 Script

```
import serial
from joblib import load
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import time
from pyts.image import GramianAngularField
import pandas as pd
import numpy as np
import pandas as pd
import cv2
from scipy.signal import butter, lfilter
from scipy.fft import fft, ifft

clf = load('C:/Users/aless/Desktop/opencv-competition/clf.joblib')
scaler2 = load('C:/Users/aless/Desktop/opencv-competition/scaler2.joblib')

#------------------------------------------------------------------------------
# Initialize communication with Arduino
arduino = serial.Serial('COM9', 9600)

# Create a vector to store signal
signal = []

start_time = time.time()
print("Put your device at 20cm from your face and say 'a' with a constant tone for 5
while time.time() - start_time < 5:
```

```
    # Read a line from Arduino
    data = arduino.readline()
    if data:
        string = data.decode()
        string2 = string.strip()
        num = int(string2)
        signal.append(num)

# Close communication with Arduino
#-------------------------------------------------------------------------------
arduino.close()
print("Signal Received")
#-------------------------------------------------------------------------------
# 1) Clean signal
signal = np.array(signal)
signal = -signal
def butter_bandpass_filter(data, lowcut, highcut, sampling_rate, order=4):
    nyquist = 0.5 * sampling_rate
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band', analog=False)
    filtered_data = lfilter(b, a, data)
    return filtered_data
lowcut = 85
highcut = 255
filtered_signal = butter_bandpass_filter(signal, lowcut, highcut, sampling_rate=1000
#-------------------------------------------------------------------------------
# 2) Denoise signal
spectrum = fft(signal)
mean_spectrum = np.mean(np.abs(spectrum))
spectrum_denoised = spectrum - mean_spectrum
signal_denoised = ifft(spectrum_denoised).real
#-------------------------------------------------------------------------------
# 3) Standardize
mean = np.mean(signal_denoised)
sd_sign = signal_denoised-mean
sd_sign = sd_sign[10:]
#-------------------------------------------------------------------------------
# 4) Gramian Angular Field
signal = pd.DataFrame(sd_sign)
array = signal.values
array = np.transpose(array)
gasf = GramianAngularField(method='summation', image_size=128)
img = gasf.transform(array)
img = (img - img.min()) / (img.max() - img.min()) * 255 #ensure range [0, 255]
```

```
img = img.astype(np.uint8)
img = img.transpose(1,2,0)
#-------------------------------------------------------------------------------
# 5) HOG
win_size = (128, 128)
cell_size = (2, 2)
block_size = (2, 2)
block_stride = (1, 1)
num_bins = 18
hog = cv2.HOGDescriptor(win_size, block_size, block_stride, cell_size, num_bins)
image = cv2.resize(img, (128, 128))
features = hog.compute(image)
#-------------------------------------------------------------------------------
# 6) Min-Max Scaling
features = features[np.newaxis, :]
scaled = scaler2.transform(features)
#-------------------------------------------------------------------------------
# 7) Linear Support Vector Machine
prediction = clf.predict(scaled)
if prediction == 0:
    prediction = 'Healthy'
else:
    prediction = 'Pathological'
#-------------------------------------------------------------------------------
# 8) Display
fig, axes = plt.subplots(2, 3, figsize=(12, 6))

axes[0,0].plot(signal)
axes[0,0].set_title('Step 1) Original Signal')

axes[0,1].plot(filtered_signal)
axes[0,1].set_title('Step 2) Filtered Signal')

axes[0,2].plot(signal_denoised)
axes[0,2].set_title('Step 3) Denoised Signal')

axes[1,0].plot(sd_sign)
axes[1,0].set_title('Step 4) Standardized Signal')

axes[1,1].imshow(image, cmap='gray')
axes[1,1].axis('off')
axes[1,1].set_title('Step 5) GAF Image')

# Reshape the HOG features into a 2D array
hog_features = features.reshape(-1, num_bins)
```

```
# Create a blank image to visualize HOG features
hog_image = np.zeros((128, 128), dtype=float)
# Iterate through HOG features and draw HOG descriptors as gradients
cell_size_x, cell_size_y = cell_size
for i in range(hog_features.shape[0]):
    for j in range(num_bins):
        magnitude = hog_features[i][j]
        angle_radians = j * np.pi / num_bins
        x = int(i % (128 / cell_size_x)) * cell_size_x + cell_size_x // 2
        y = int(i // (128 / cell_size_y)) * cell_size_y + cell_size_y // 2
        x1 = x + int(magnitude * np.cos(angle_radians) * cell_size_x / 2)
        y1 = y + int(magnitude * np.sin(angle_radians) * cell_size_y / 2)
        cv2.line(hog_image, (y, x), (y1, x1), 255, 1)

axes[1,2].imshow(hog_image, cmap='gray')
axes[1,2].axis('off')
axes[1,2].set_title('Step 6) HOG Features')

plt.suptitle(str(prediction))
plt.show(block=True)
```

References

[1] U. Cesari, G. De Pietro, E. Marciano, C. Niri, G. Sannino, and L. Verde, "A new database of healthy and pathological voices," *Computers Electrical Engineering*, vol. 68, pp. 310–321, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790617338739

[2] S. Buttwrworth, "On the theory of filters amplifiers," *The wireless engineer*, pp. 536–541, 1930. [Online]. Available: https://www.changpuak.ch/electronics/downloads/On_the_Theory_of_Filter_Amplifiers.pdf

[3] E. O. Brigham and R. E. Morrow, "The fast fourier transform," *IEEE Spectrum*, vol. 4, no. 12, pp. 63–70, 1967.

[4] Z. Wang and T. Oates, "Imaging time-series to improve classification and imputation," 2015.

[5] R. K. McConnell, "Method of and apparatus for pattern recognition." [Online]. Available: https://www.osti.gov/biblio/6007283