Calculating Histogram of Oriented Features (HOGs)

```
import cv2
import os
import json
import numpy as np
from scipy.io import loadmat
from tqdm import tqdm
import pickle

import seaborn as sns
import matplotlib.pyplot as plt

from skimage.feature import hog
from skimage import data, color, exposure, io
from skimage.io import imread
from skimage.transform import resize

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_s
from sklearn.multioutput import MultiOutputClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, Ad
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.base import clone

from imblearn.over_sampling import SMOTE

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropou


%matplotlib inline
```

```
subject_ids = ['SN001', 'SN002', 'SN003', 'SN004', 'SN0
```

```
def extract_hog_features(image_path, target_size=(128,
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is not None:
        # Resize image to the target size
        img_resized = resize(img, target_size, anti_ali
```

```
            # Compute HOG features
            features, _ = hog(img_resized, orientations=9,
                              cells_per_block=(2, 2), block
            return features
        else:
            raise FileNotFoundError(f"The image at {image_p

    def process_subjects_for_hog(subject_ids, json_dir, cro
        hog_features = {}

        for subject_id in tqdm(subject_ids, desc="Processin
            subject_hog_features = []
            json_file_path = os.path.join(json_dir, f'{subj
            if os.path.exists(json_file_path):
                with open(json_file_path, 'r') as file:
                    data = json.load(file)
                    for frame_data in data:
                        img_path = os.path.join(cropped_dir
                        try:
                            features = extract_hog_features
                            subject_hog_features.append(fea
                        except FileNotFoundError:
                            print(f"Warning: The file {img_

            hog_features[subject_id] = subject_hog_features

        return hog_features

    json_dir = 'json'
    cropped_images_dir = 'croppedImg'

    # Process all subjects to compute HOG features
    all_hog_features = process_subjects_for_hog(subject_ids

    # Save the HOG features to a .pkl file
    with open('HOG_features.pkl', 'wb') as f:
        pickle.dump(all_hog_features, f)
```

```
Processing subjects: 100%|██████████| 26/26 [47:21<00:00,
109.31s/it]
```

Visualzing the feature

```
    def compute_and_plot_average_hog(subject_id, target_siz
        cropped_dir = os.path.join("croppedImg", subject_id
        hog_images = []
```

```
            for img_name in tqdm(os.listdir(cropped_dir), desc=
                img_path = os.path.join(cropped_dir, img_name)
                img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE
                if img is not None:
                    # Compute HOG features and the associated v
                    _, hog_image = hog(img, orientations=9, pix
                                       cells_per_block=(2, 2),
                    # Resize the HOG image to the target size
                    resized_hog_image = resize(hog_image, targe
                    hog_images.append(resized_hog_image)

            # Compute the average of the resized HOG images
            if hog_images:
                sns.set(style="whitegrid")
                average_hog_image = np.mean(hog_images, axis=0)
                plt.imshow(average_hog_image, cmap='gray')
                plt.title(f"Average HOG Image for Subject {subj
                plt.axis('off')
                plt.show()

        for subject_id in subject_ids:
            compute_and_plot_average_hog(subject_id)
```

```
Computing HOG for SN001: 100%|██████████| 4845/4845
[16:49<00:00,  4.80it/s]
Computing HOG for SN002: 100%|██████████| 4845/4845
[14:46<00:00,  5.46it/s]
Computing HOG for SN003: 100%|██████████| 4845/4845
[12:00<00:00,  6.73it/s]
Computing HOG for SN004: 100%|██████████| 4845/4845
[10:41<00:00,  7.56it/s]
Computing HOG for SN005: 100%|██████████| 4845/4845
[16:07<00:00,  5.01it/s]
Computing HOG for SN006: 100%|██████████| 4845/4845
[15:52<00:00,  5.09it/s]
Computing HOG for SN007: 100%|██████████| 4845/4845
[16:35<00:00,  4.87it/s]
Computing HOG for SN008: 100%|██████████| 4845/4845
[19:05<00:00,  4.23it/s]
Computing HOG for SN008: 100%|██████████| 4845/4845
[19:03<00:00,  4.24it/s]
Computing HOG for SN010: 100%|██████████| 4844/4844
[15:22<00:00,  5.25it/s]
Computing HOG for SN011: 100%|██████████| 4845/4845
[11:22<00:00,  7.10it/s]
Computing HOG for SN012: 100%|██████████| 4845/4845
[15:52<00:00,  5.09it/s]
```

Computing HOG for SN013: 100%|██████████| 4845/4845
[16:31<00:00, 4.89it/s]
Computing HOG for SN016: 100%|██████████| 4845/4845
[15:24<00:00, 5.24it/s]
Computing HOG for SN017: 100%|██████████| 4845/4845
[11:19<00:00, 7.13it/s]
Computing HOG for SN018: 100%|██████████| 4845/4845
[10:03<00:00, 8.03it/s]
Computing HOG for SN021: 100%|██████████| 4845/4845
[15:15<00:00, 5.29it/s]
Computing HOG for SN023: 100%|██████████| 4845/4845
[17:31<00:00, 4.61it/s]
Computing HOG for SN024: 100%|██████████| 4845/4845
[13:58<00:00, 5.78it/s]
Computing HOG for SN025: 100%|██████████| 4845/4845
[19:11<00:00, 4.21it/s]
Computing HOG for SN026: 100%|██████████| 4845/4845
[14:12<00:00, 5.69it/s]
Computing HOG for SN027: 100%|██████████| 4845/4845
[12:59<00:00, 6.21it/s]
Computing HOG for SN028: 100%|██████████| 4845/4845
[15:20<00:00, 5.26it/s]
Computing HOG for SN029: 100%|██████████| 4845/4845
[18:03<00:00, 4.47it/s]
Computing HOG for SN030: 100%|██████████| 4845/4845
[15:30<00:00, 5.21it/s]
Computing HOG for SN031: 100%|██████████| 4845/4845
[14:24<00:00, 5.61it/s]
Computing HOG for SN032: 100%|██████████| 4845/4845
[14:06<00:00, 5.72it/s]

Average HOG Image for Subject SN001

Average HOG Image for Subject SN002

Average HOG Image for Subject SN003

Average HOG Image for Subject SN004

Average HOG Image for Subject SN005

Average HOG Image for Subject SN006

Average HOG Image for Subject SN007

Average HOG Image for Subject SN008

Average HOG Image for Subject SN008

Average HOG Image for Subject SN010

Average HOG Image for Subject SN011

Average HOG Image for Subject SN012

Average HOG Image for Subject SN013

Average HOG Image for Subject SN016

Average HOG Image for Subject SN017

Average HOG Image for Subject SN018

Average HOG Image for Subject SN021

Average HOG Image for Subject SN023

Average HOG Image for Subject SN024

Average HOG Image for Subject SN025

Average HOG Image for Subject SN026

Average HOG Image for Subject SN027

Average HOG Image for Subject SN028

Average HOG Image for Subject SN029

Average HOG Image for Subject SN030

Average HOG Image for Subject SN031

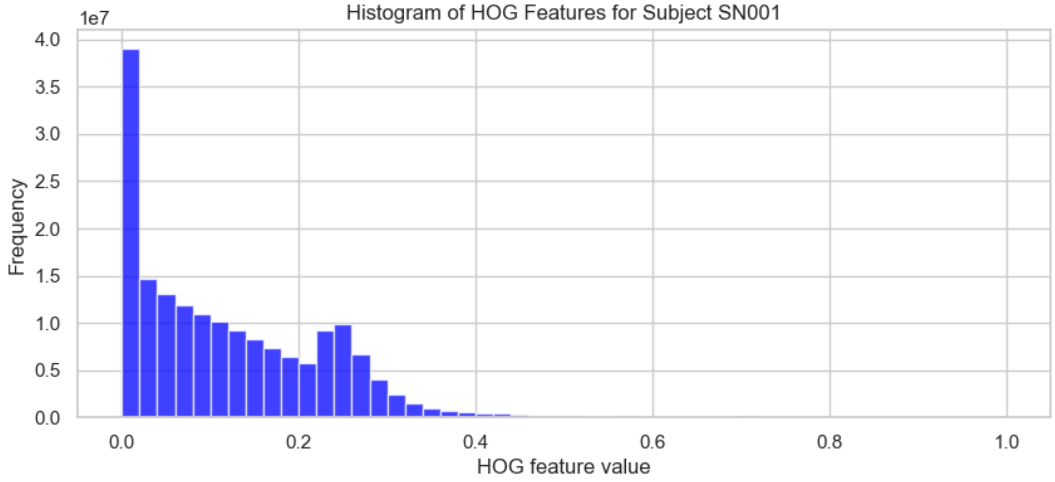
Average HOG Image for Subject SN032

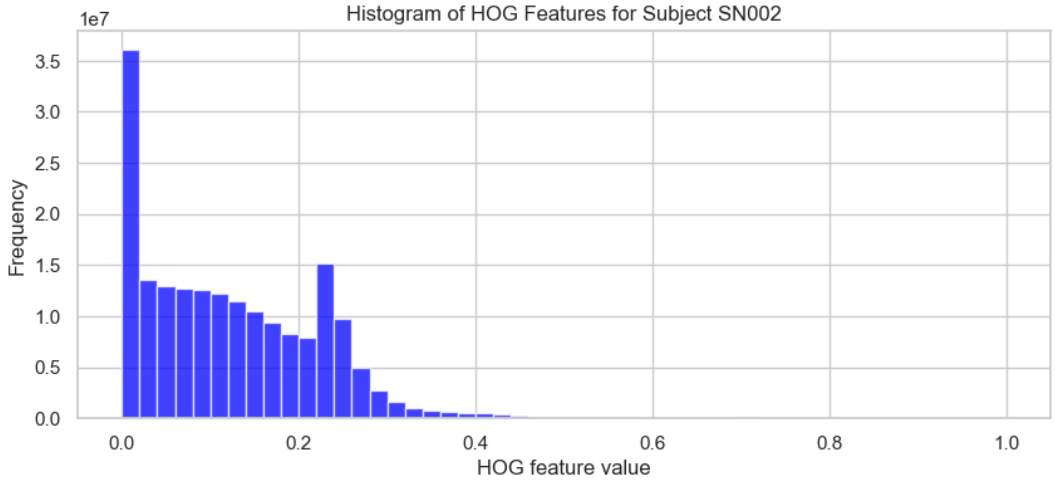
```
def compute_hog_features_and_create_histogram(subject_i
    cropped_dir = os.path.join("croppedImg", subject_id
    hog_features_list = []

    for img_name in tqdm(os.listdir(cropped_dir), desc=
        img_path = os.path.join(cropped_dir, img_name)
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE
        if img is not None:
            hog_features = hog(img, orientations=9, pix
                               cells_per_block=(2, 2),
            hog_features_list.append(hog_features)

    # Concatenate all HOG features from all images to c
    all_hog_features = np.concatenate(hog_features_list

    # Now create the histogram
    plt.figure(figsize=(10, 4))
    n_bins = 50
    plt.hist(all_hog_features, bins=n_bins, facecolor='
    plt.title(f"Histogram of HOG Features for Subject {
    plt.xlabel('HOG feature value')
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()
```

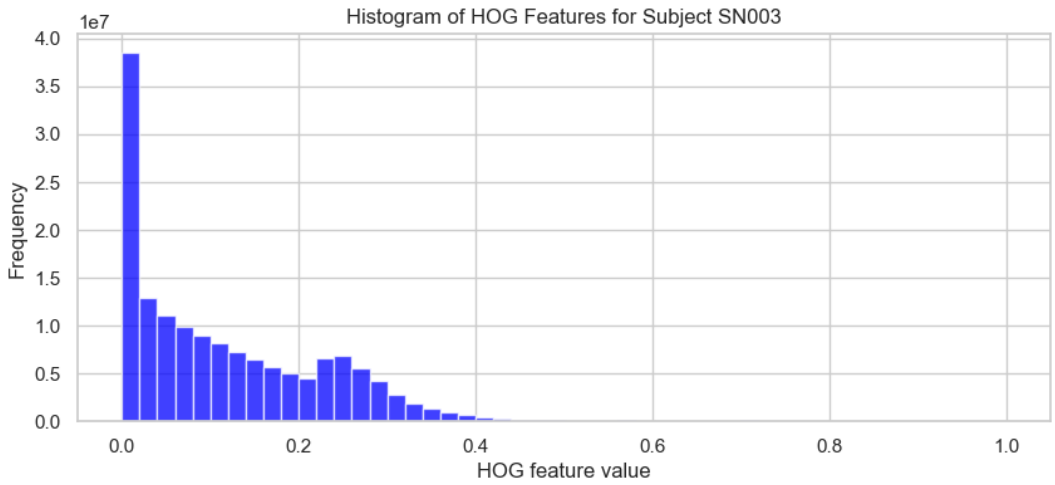
```
        for subject_id in subject_ids:
            compute_hog_features_and_create_histogram(subject_i
```

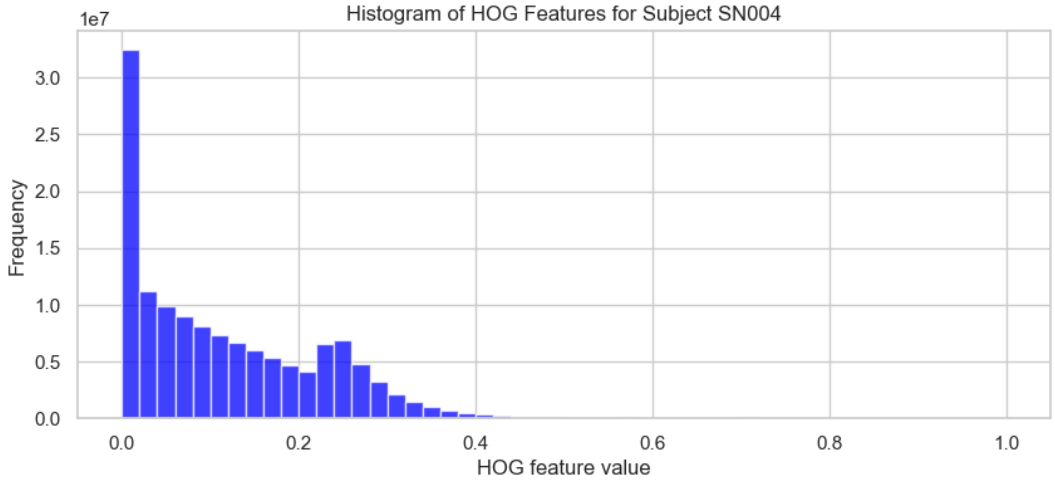
```
Computing HOG for SN001:   0%|          | 0/4845 [00:00<?, ?
it/s]Computing HOG for SN001: 100%|██████████| 4845/4845
[02:15<00:00, 35.86it/s]
Computing HOG for SN002: 100%|██████████| 4845/4845
[02:24<00:00, 33.41it/s]
Computing HOG for SN003: 100%|██████████| 4845/4845
[01:56<00:00, 41.49it/s]
Computing HOG for SN004: 100%|██████████| 4845/4845
[01:43<00:00, 47.04it/s]
Computing HOG for SN005: 100%|██████████| 4845/4845
[02:35<00:00, 31.12it/s]
Computing HOG for SN006: 100%|██████████| 4845/4845
[02:33<00:00, 31.57it/s]
Computing HOG for SN007: 100%|██████████| 4845/4845
[02:39<00:00, 30.31it/s]
Computing HOG for SN008: 100%|██████████| 4845/4845
[03:04<00:00, 26.30it/s]
Computing HOG for SN010: 100%|██████████| 4844/4844
[02:28<00:00, 32.53it/s]
Computing HOG for SN011: 100%|██████████| 4845/4845
[01:49<00:00, 44.24it/s]
Computing HOG for SN012: 100%|██████████| 4845/4845
[02:33<00:00, 31.51it/s]
Computing HOG for SN013: 100%|██████████| 4845/4845
[02:41<00:00, 29.97it/s]
Computing HOG for SN016: 100%|██████████| 4845/4845
[02:31<00:00, 32.06it/s]
Computing HOG for SN017: 100%|██████████| 4845/4845
[01:49<00:00, 44.36it/s]
Computing HOG for SN018: 100%|██████████| 4845/4845
[01:37<00:00, 49.76it/s]
Computing HOG for SN021: 100%|██████████| 4845/4845
[02:25<00:00, 33.19it/s]
Computing HOG for SN023: 100%|██████████| 4845/4845
[02:47<00:00, 28.93it/s]
Computing HOG for SN024: 100%|██████████| 4845/4845
[02:14<00:00, 35.91it/s]
Computing HOG for SN025: 100%|██████████| 4845/4845
[03:03<00:00, 26.35it/s]
Computing HOG for SN026: 100%|██████████| 4845/4845
[02:16<00:00, 35.53it/s]
Computing HOG for SN027: 100%|██████████| 4845/4845
[02:03<00:00, 39.30it/s]
```

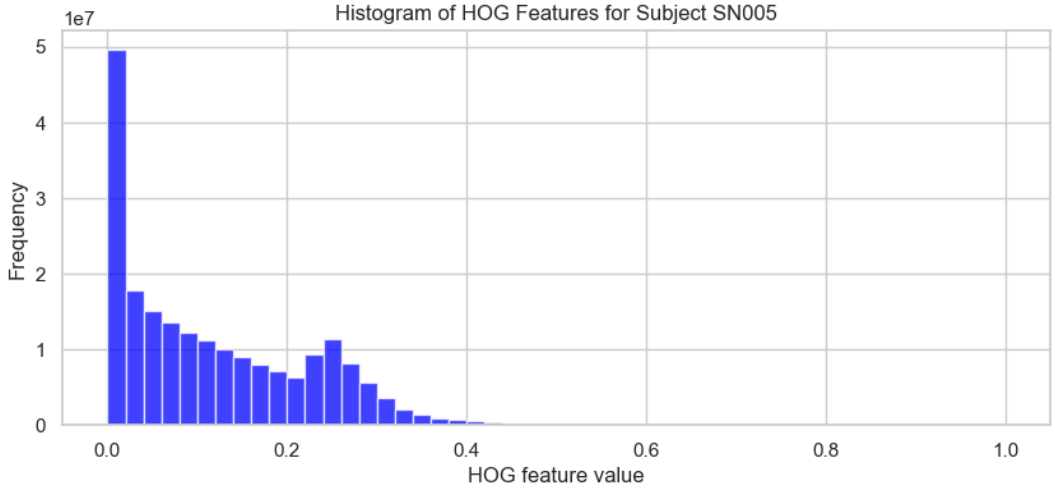
```
Computing HOG for SN028: 100%|██████████| 4845/4845
[02:27<00:00, 32.95it/s]
Computing HOG for SN029: 100%|██████████| 4845/4845
[02:51<00:00, 28.29it/s]
Computing HOG for SN030: 100%|██████████| 4845/4845
[02:27<00:00, 32.85it/s]
Computing HOG for SN031: 100%|██████████| 4845/4845
[02:16<00:00, 35.38it/s]
Computing HOG for SN032: 100%|██████████| 4845/4845
[02:15<00:00, 35.77it/s]
```

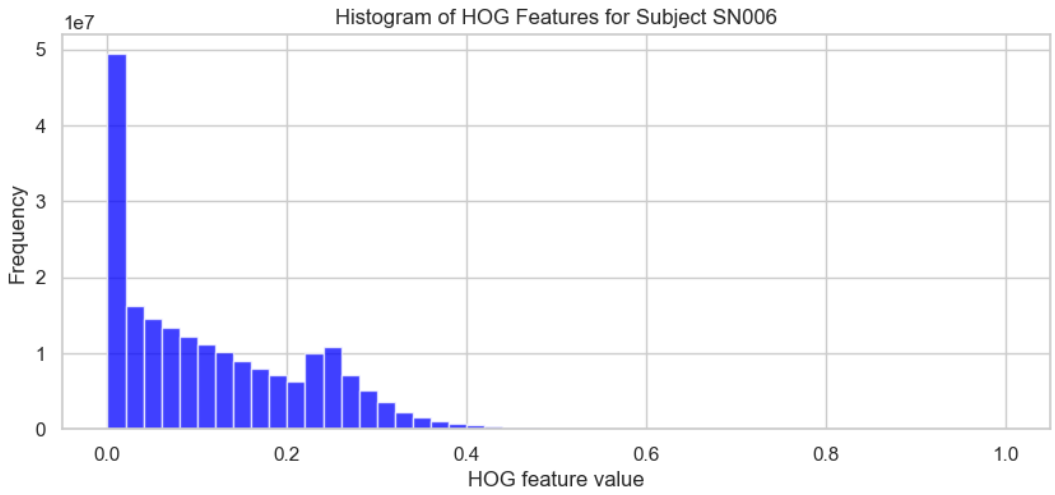


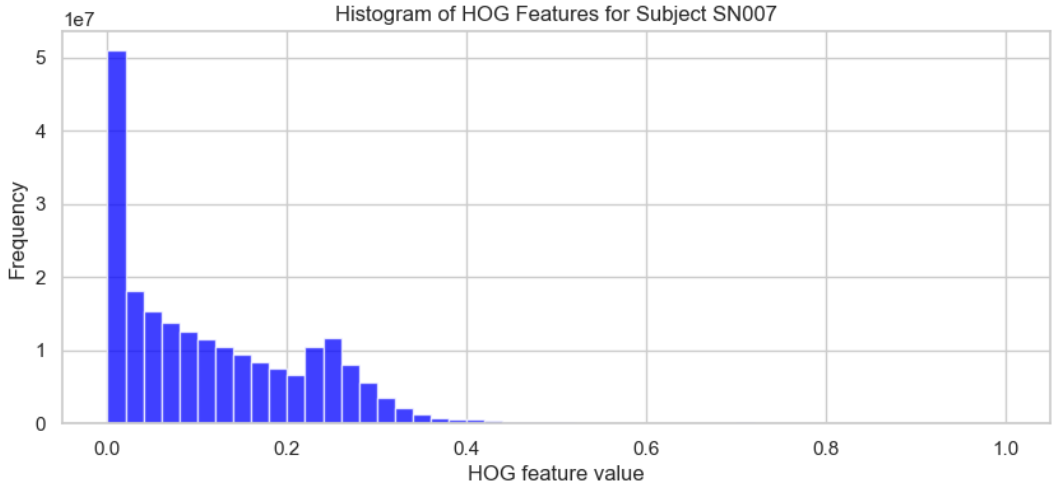


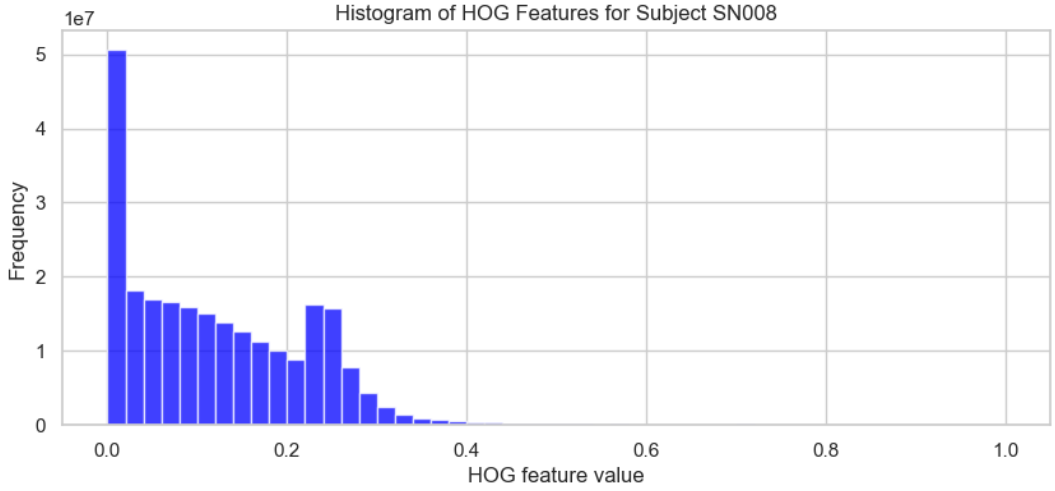

Histogram of HOG Features for Subject SN003
1e7
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
HOG feature value

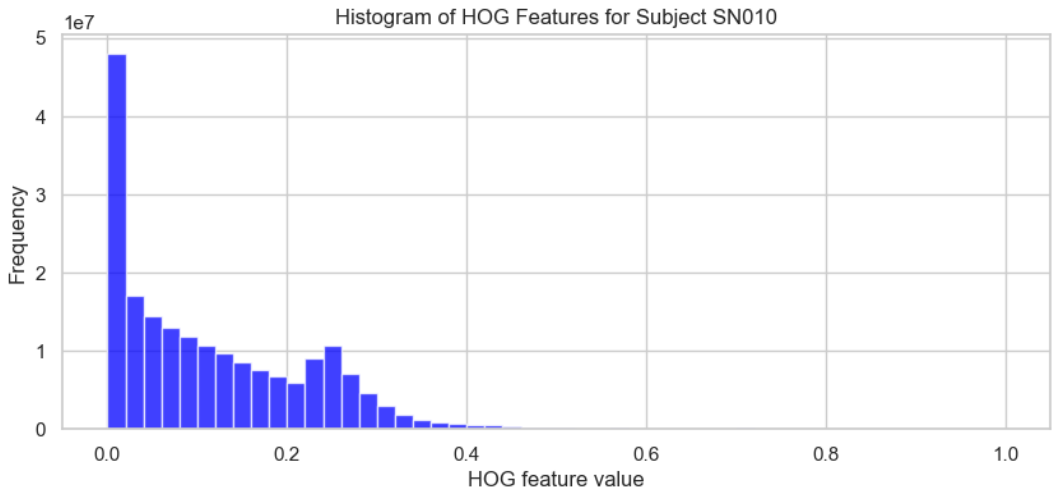

Histogram of HOG Features for Subject SN004
1e7
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
HOG feature value

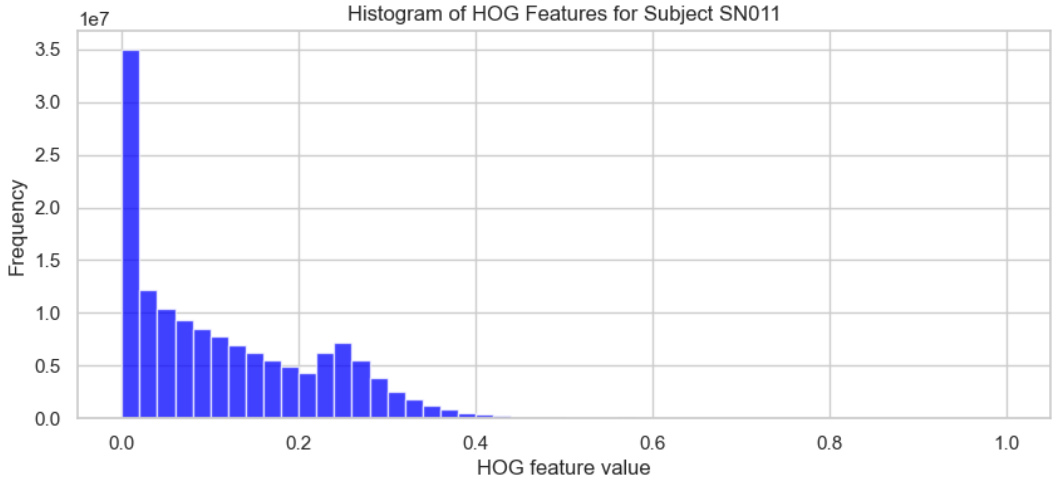

Histogram of HOG Features for Subject SN005
1e7
5
4
3
2
1
0
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
HOG feature value

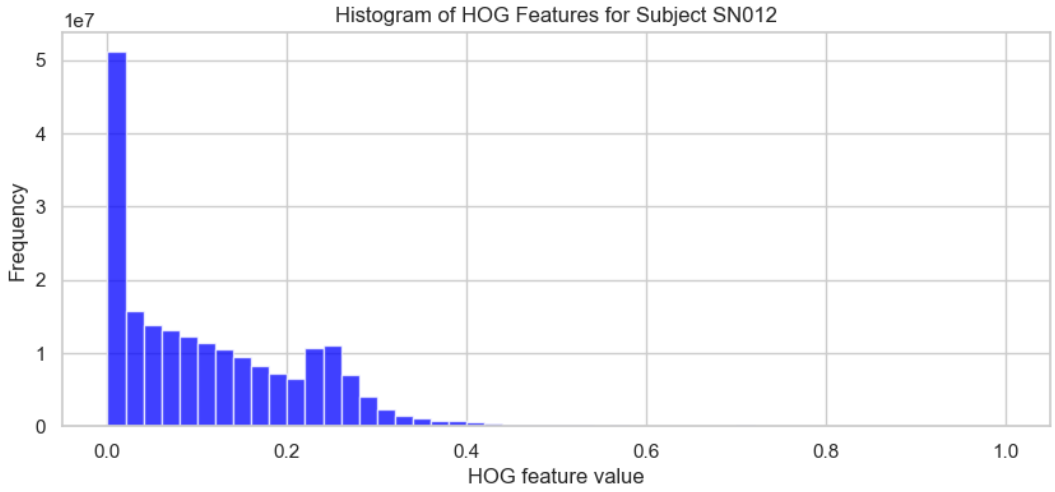

Histogram of HOG Features for Subject SN006
1e7
5
4
3
2
1
0
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
HOG feature value

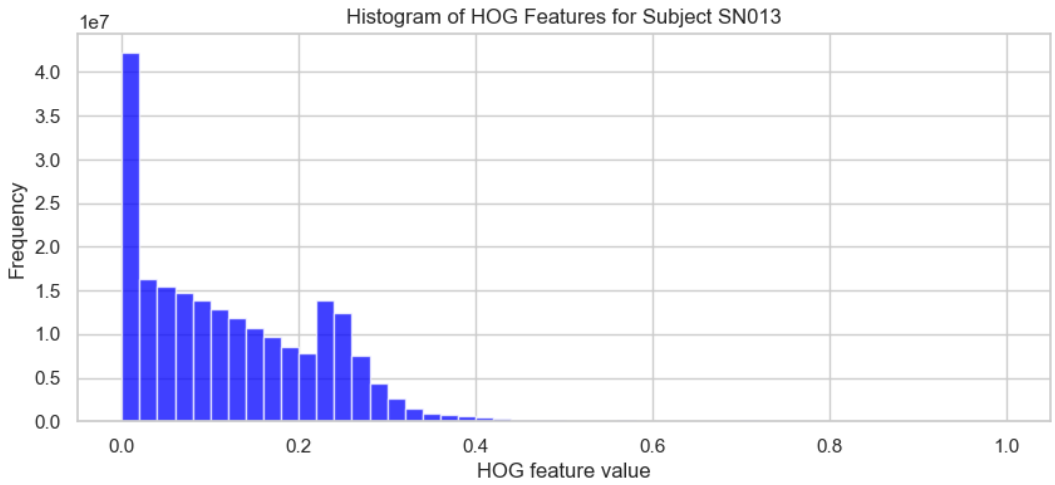

Histogram of HOG Features for Subject SN007
1e7
5
4
3
2
1
0
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
HOG feature value

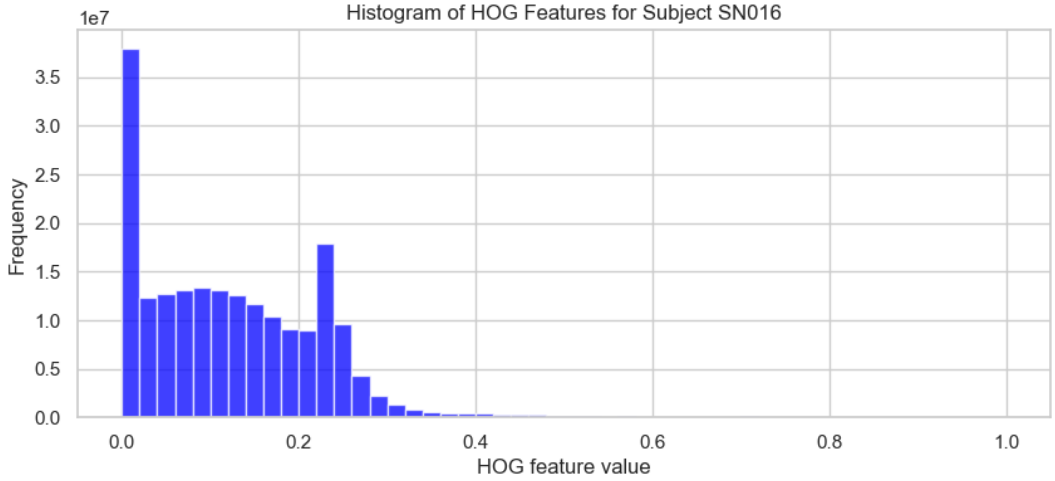

Histogram of HOG Features for Subject SN008
1e7
5
4
3
2
1
0
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
HOG feature value

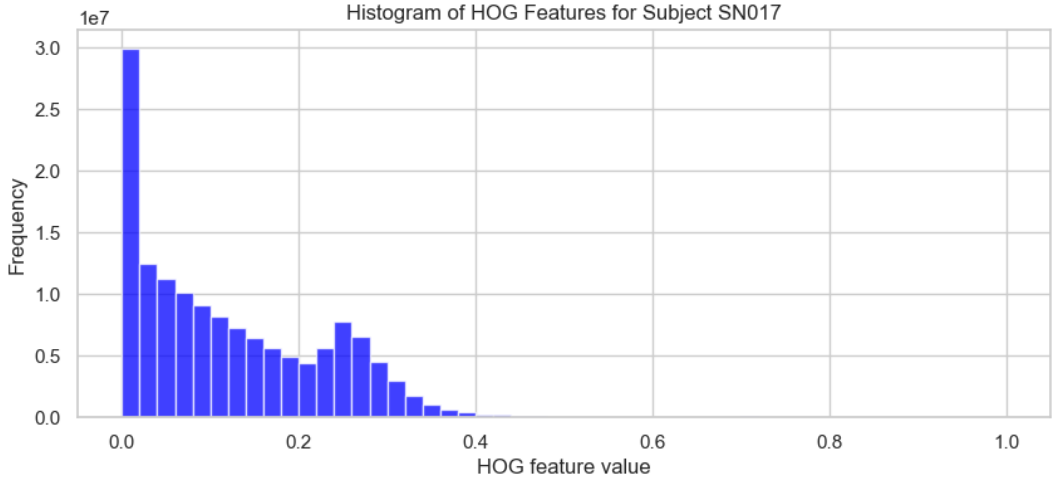

Histogram of HOG Features for Subject SN010
1e7
5
4
3
2
1
0
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
HOG feature value

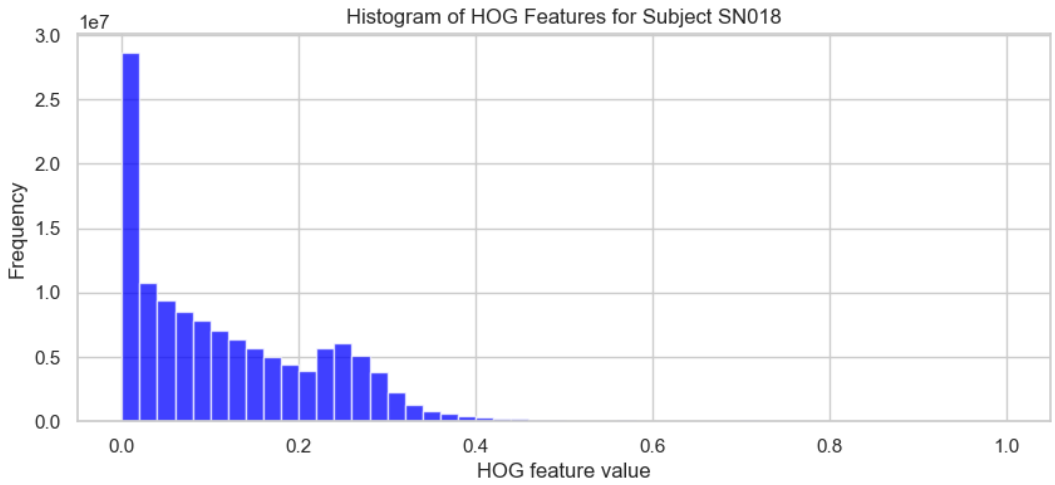

Histogram of HOG Features for Subject SN011
1e7
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
HOG feature value

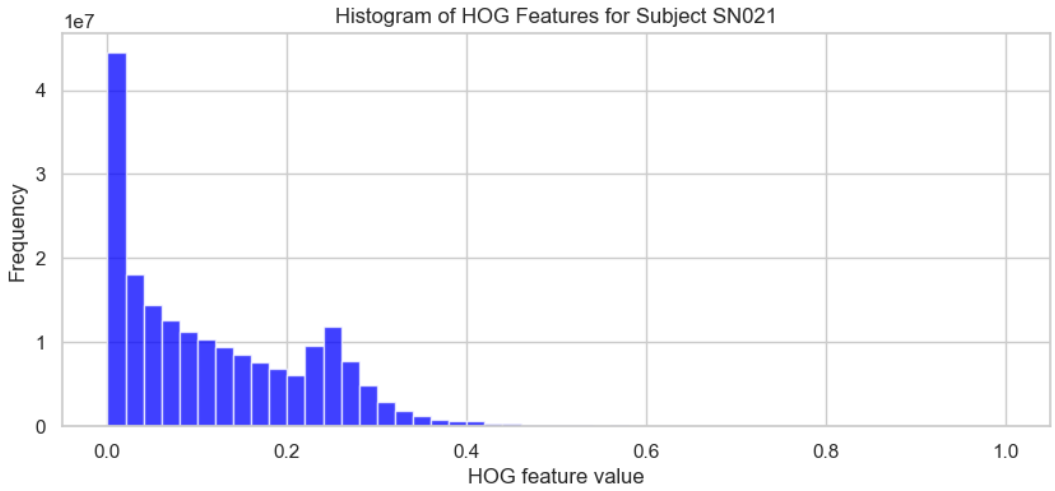

Histogram of HOG Features for Subject SN012
1e7
5
4
3
2
1
0
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
HOG feature value

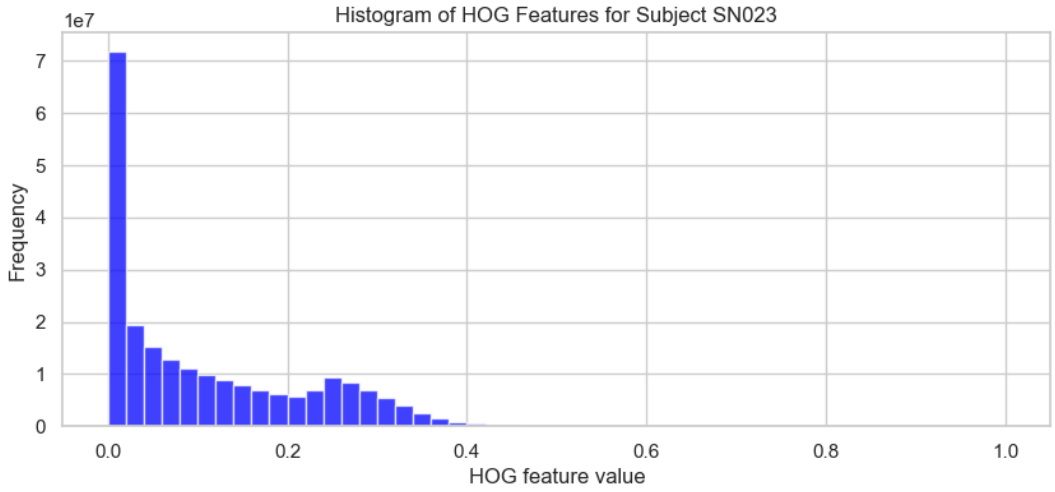

Histogram of HOG Features for Subject SN013
1e7
Frequency
HOG feature value
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0.0
0.2
0.4
0.6
0.8
1.0

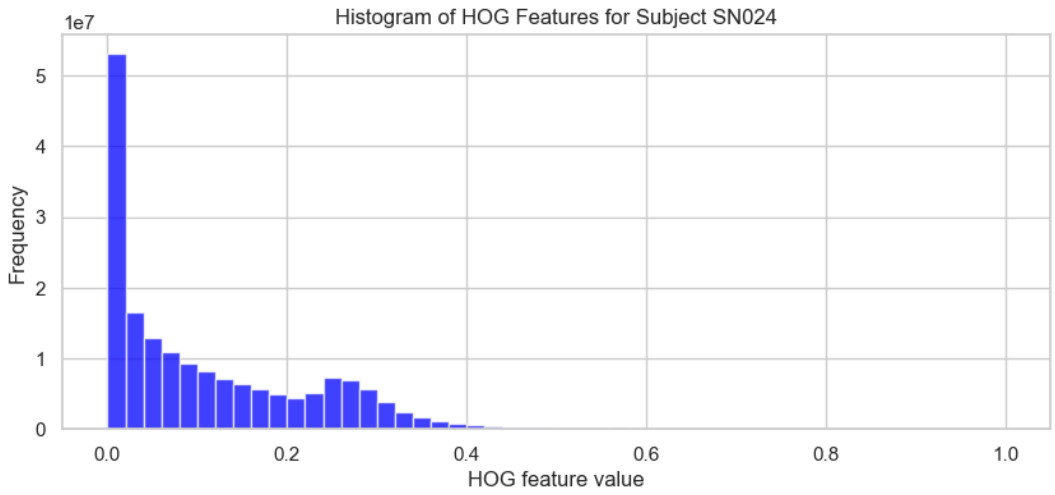

Histogram of HOG Features for Subject SN016
1e7
Frequency
HOG feature value
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0.0
0.2
0.4
0.6
0.8
1.0

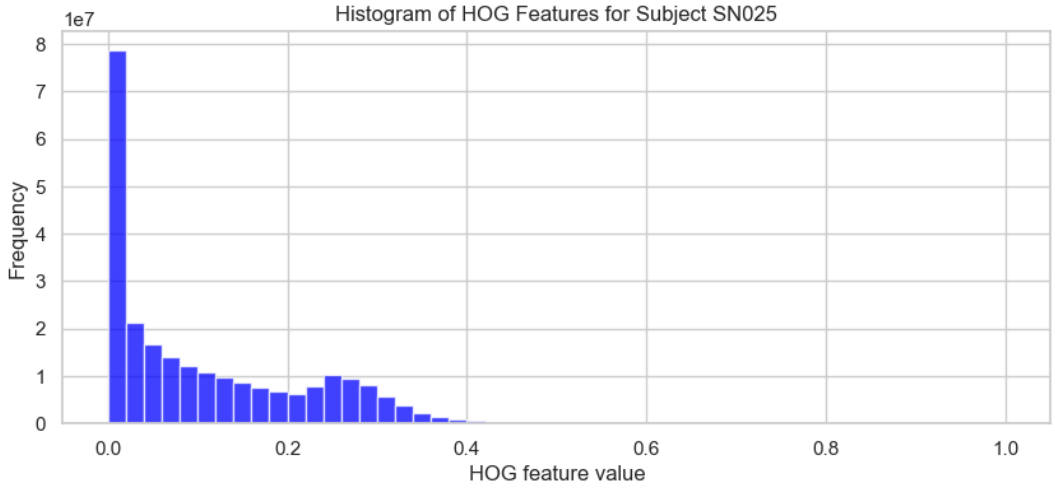

Histogram of HOG Features for Subject SN017
1e7
Frequency
HOG feature value
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.0
0.2
0.4
0.6
0.8
1.0

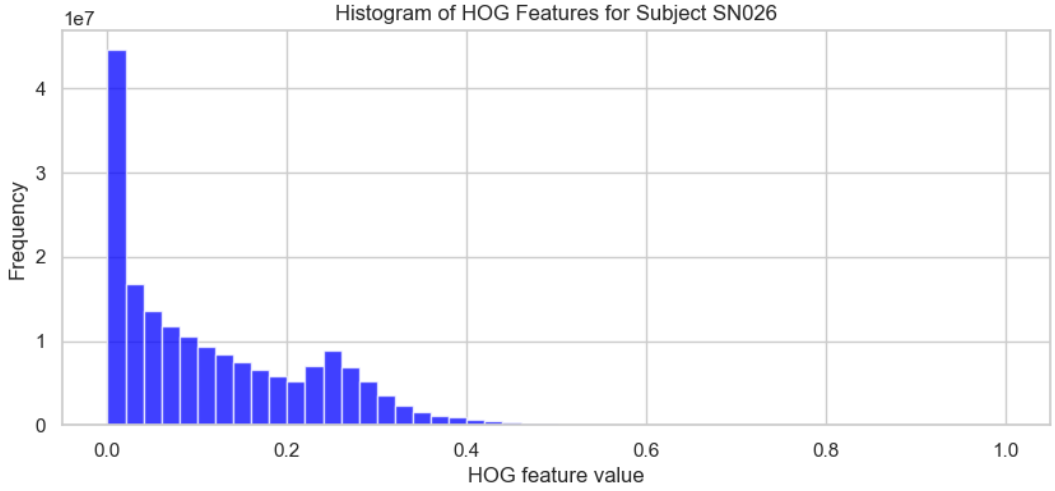

Histogram of HOG Features for Subject SN018
1e7
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
HOG feature value

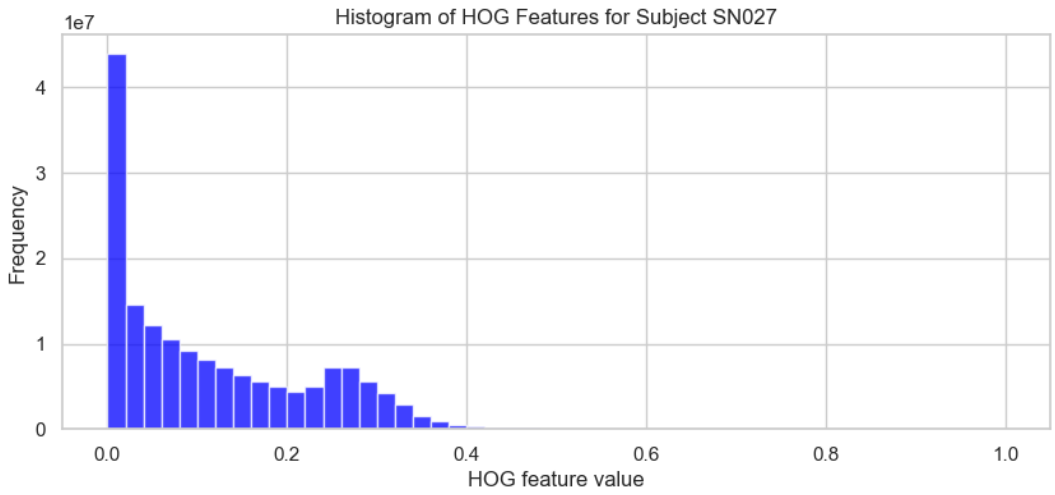

Histogram of HOG Features for Subject SN021
1e7
4
3
2
1
0
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
HOG feature value

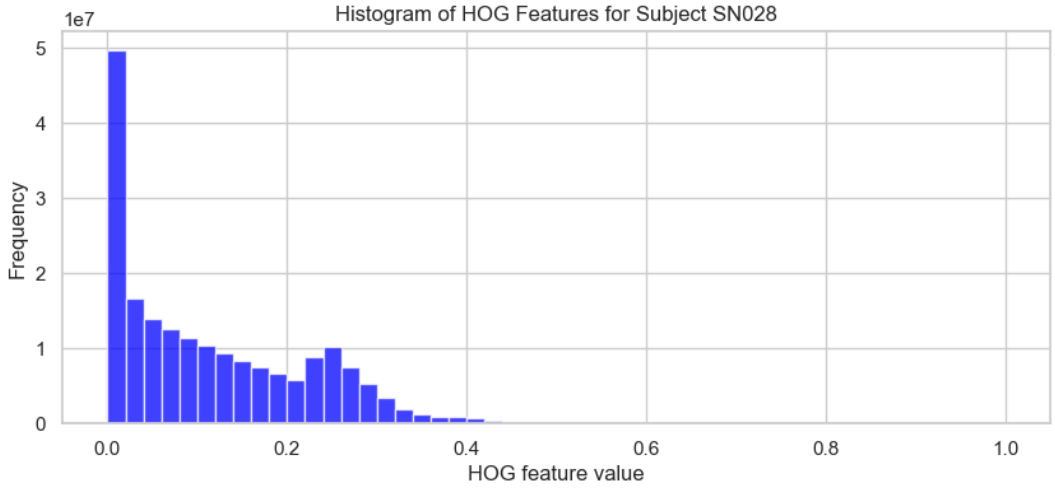

Histogram of HOG Features for Subject SN023
1e7
7
6
5
4
3
2
1
0
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
HOG feature value

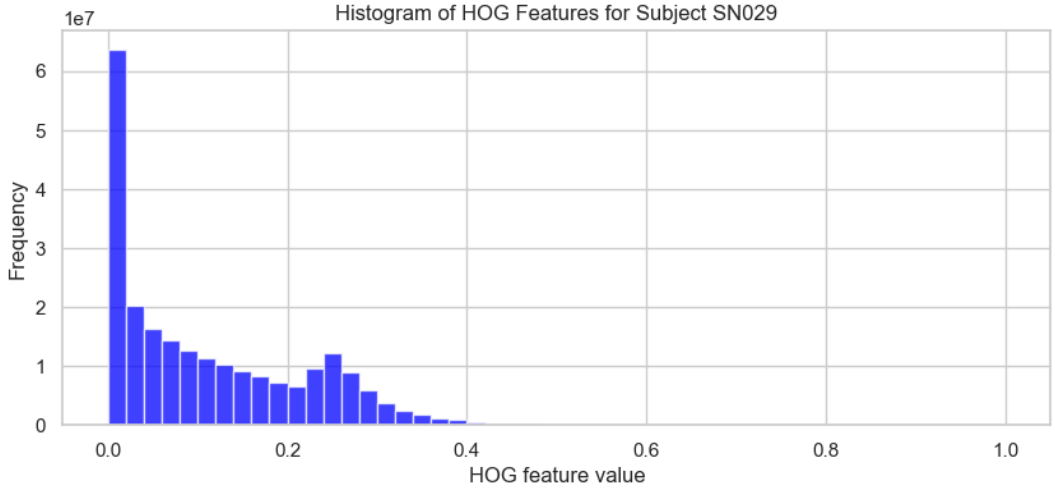

Histogram of HOG Features for Subject SN024
1e7
Frequency
HOG feature value
0
1
2
3
4
5
0.0
0.2
0.4
0.6
0.8
1.0

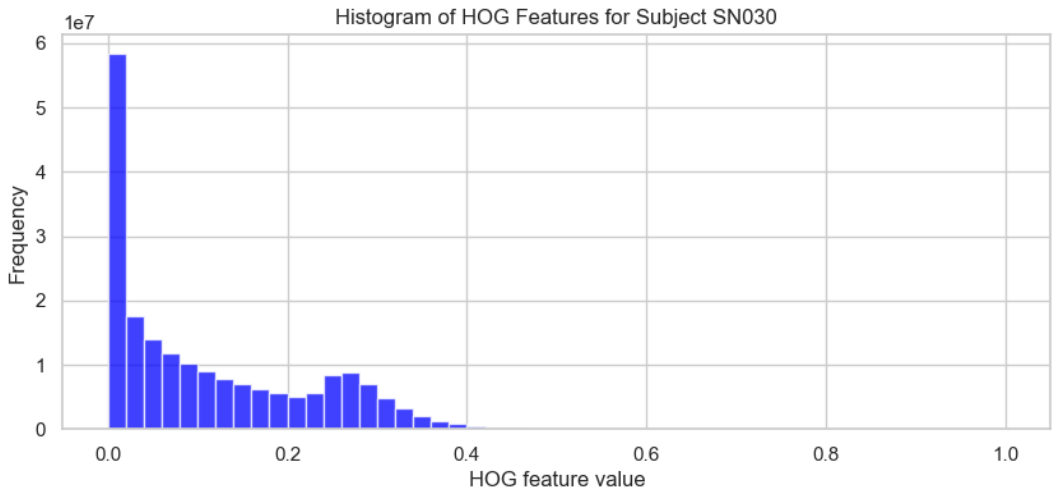

Histogram of HOG Features for Subject SN025
1e7
Frequency
HOG feature value
0
1
2
3
4
5
6
7
8
0.0
0.2
0.4
0.6
0.8
1.0

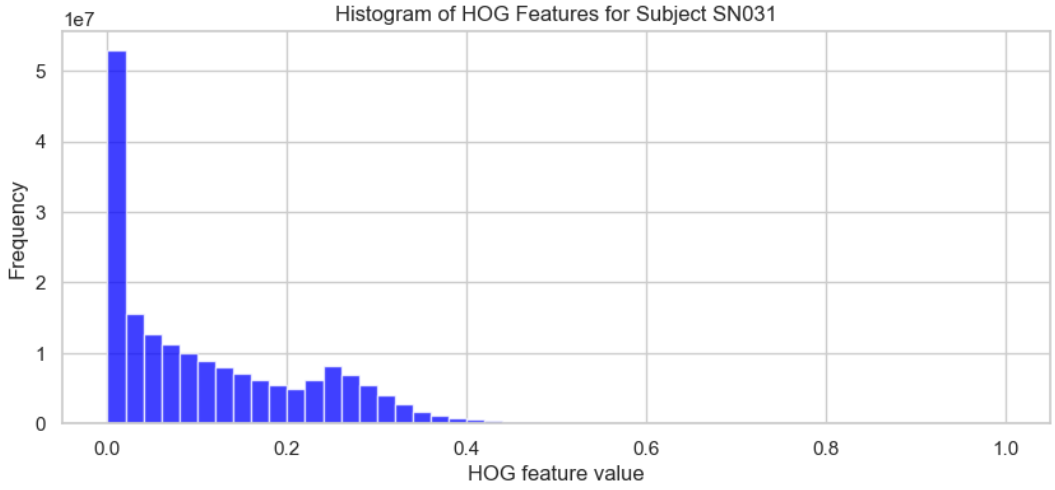

Histogram of HOG Features for Subject SN026
1e7
Frequency
HOG feature value
0
1
2
3
4
0.0
0.2
0.4
0.6
0.8
1.0

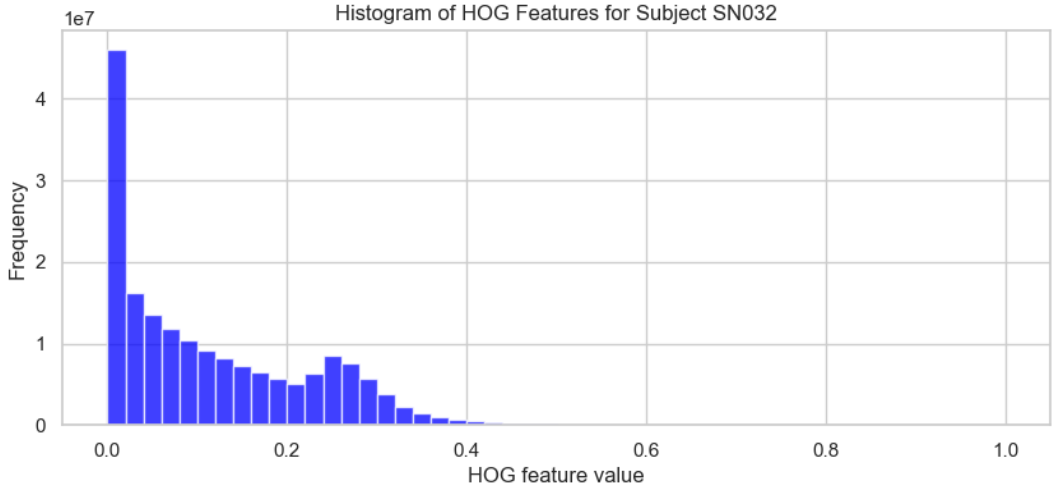

Histogram of HOG Features for Subject SN027
1e7
Frequency
HOG feature value
0
1
2
3
4
0.0
0.2
0.4
0.6
0.8
1.0


Histogram of HOG Features for Subject SN028
1e7
Frequency
HOG feature value
0
1
2
3
4
5
0.0
0.2
0.4
0.6
0.8
1.0


Histogram of HOG Features for Subject SN029
1e7
Frequency
HOG feature value
0
1
2
3
4
5
6
0.0
0.2
0.4
0.6
0.8
1.0





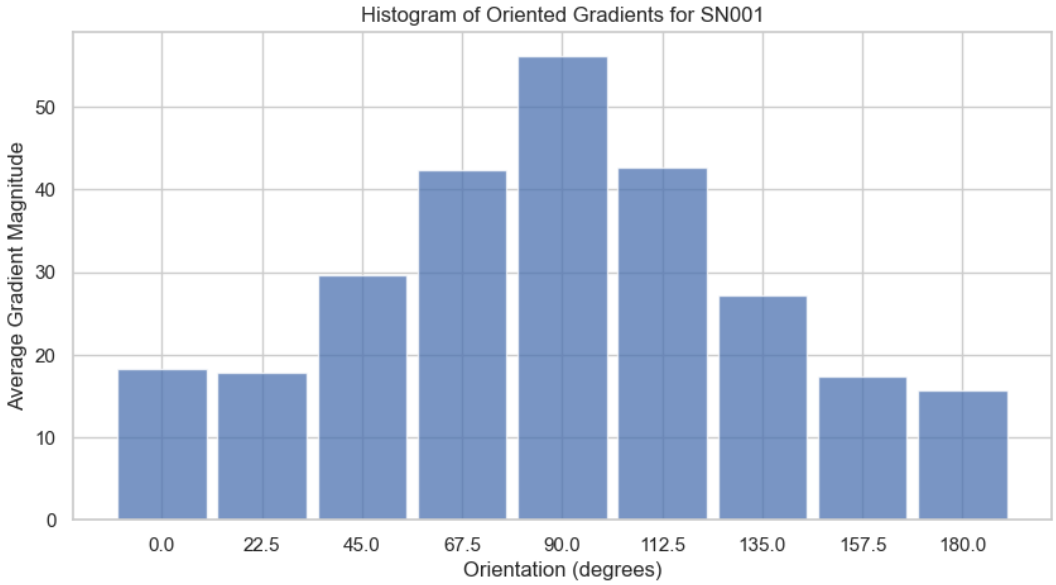


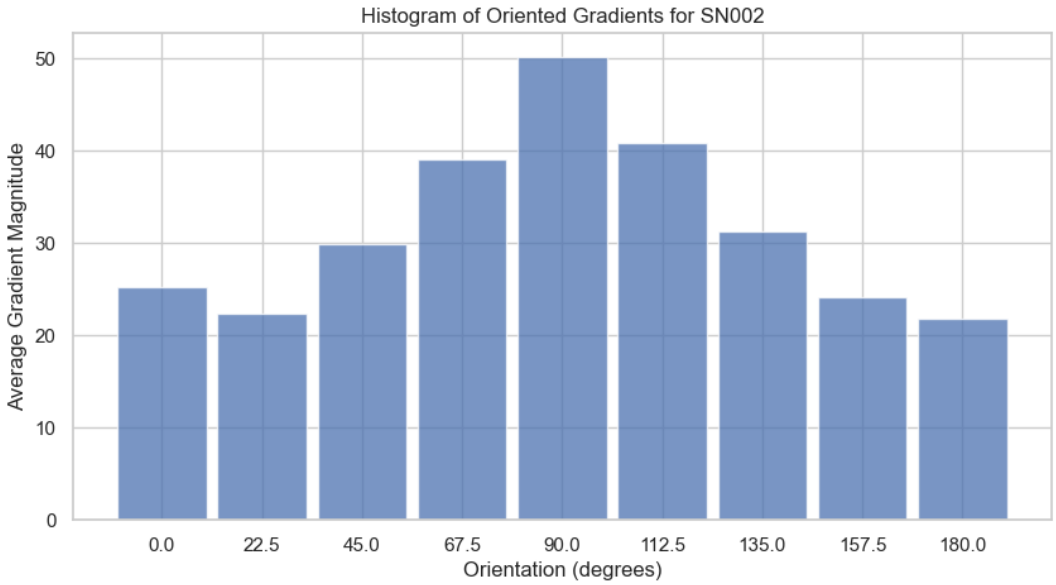

```
        def calculate_gradient_histograms(img, cell_size=(8, 8)
            # Compute HOG features without visualization, which
            hog_features = hog(img, orientations=n_bins, pixels
                               cells_per_block=(1, 1), block_no
            # The hog_features here is a 3D array: (n_cells_row
            # Sum histograms over all cells
```

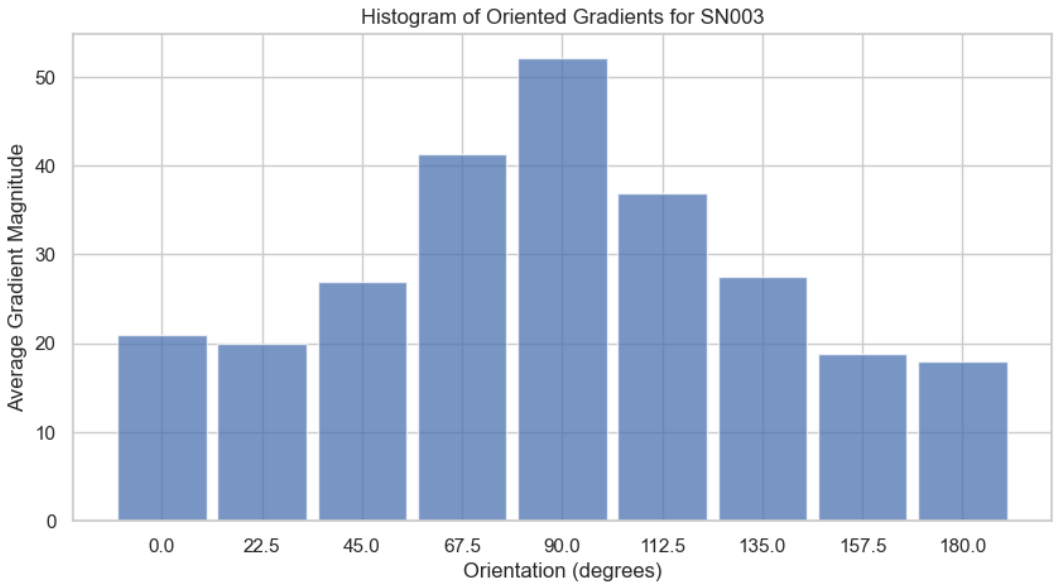
```
            histogram = hog_features.sum(axis=(0, 1))
            return histogram

        def calculate_histograms_for_subject(subject_id, image_
            cropped_dir = os.path.join("croppedImg", subject_id
            histograms = []

            for img_name in tqdm(os.listdir(cropped_dir), desc=
                img_path = os.path.join(cropped_dir, img_name)
                img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE
                if img is not None:
                    # Resize the image
                    img_resized = resize(img, image_size, anti_
                    # Calculate HOG features
                    histogram = calculate_gradient_histograms(i
                    histograms.append(histogram)

            # Normalize the histograms by the number of images
            normalized_histogram = np.sum(histograms, axis=0) /
            return normalized_histogram

        def plot_histograms(histogram, subject_id, n_bins=9):
            # Plotting
            plt.figure(figsize=(10, 5))
            plt.title(f'Histogram of Oriented Gradients for {su
            bin_centers = np.linspace(0, 180, n_bins)

            histogram = histogram.ravel()

            # Create the bar graph with the histogram values
            plt.bar(bin_centers, histogram[:n_bins], width=180/

            plt.xlabel('Orientation (degrees)')
            plt.ylabel('Average Gradient Magnitude')
            plt.xticks(bin_centers)
            plt.grid(True)
            plt.show()

        for subject_id in subject_ids:
            normalized_histogram = calculate_histograms_for_sub
            plot_histograms(normalized_histogram, subject_id)
```

```
Calculating histograms for SN001:   0%|          | 0/4845
[00:00<?, ?it/s]Calculating histograms for SN001: 100%|
██████████| 4845/4845 [00:23<00:00, 204.10it/s]
Calculating histograms for SN002: 100%|██████████| 4845/4845
[00:23<00:00, 202.02it/s]
```

```
Calculating histograms for SN003: 100%|██████████| 4845/4845
[00:22<00:00, 211.19it/s]
```



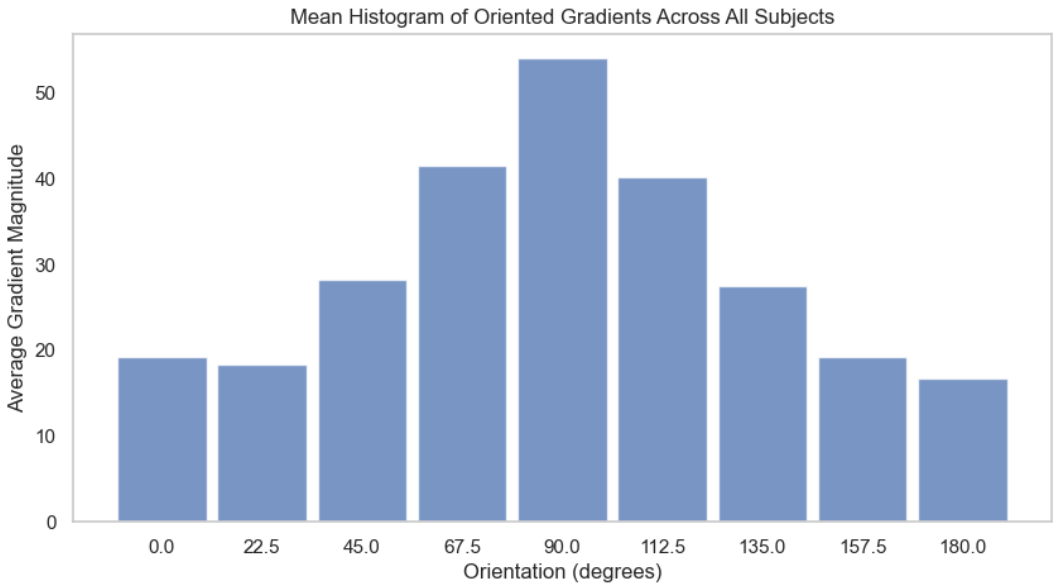





```
def compute_mean_histogram(subjectIds, image_size=(64,
    all_histograms = []

    for subject_id in subjectIds:
        subject_histogram = calculate_histograms_for_su
        all_histograms.append(subject_histogram)

    # Compute the mean histogram across all subjects
    mean_histogram = np.mean(all_histograms, axis=0)
    return mean_histogram

def plot_mean_histogram(mean_histogram, n_bins=9):
    plt.figure(figsize=(10, 5))
    plt.title('Mean Histogram of Oriented Gradients Acr
    bin_centers = np.linspace(0, 180, n_bins)

    # Make sure the histogram array is 1D
    mean_histogram = mean_histogram.ravel()

    plt.bar(bin_centers, mean_histogram[:n_bins], width
    plt.xlabel('Orientation (degrees)')
    plt.ylabel('Average Gradient Magnitude')
    plt.xticks(bin_centers)
    plt.grid(False)
    plt.show()

    sns.set(style="whitegrid")

mean_histogram = compute_mean_histogram(subject_ids)
plot_mean_histogram(mean_histogram)
```

Calculating histograms for SN001: 100%|██████████| 4845/4845 [00:21<00:00, 221.62it/s]
Calculating histograms for SN002: 100%|██████████| 4845/4845 [00:22<00:00, 217.97it/s]
Calculating histograms for SN003: 100%|██████████| 4845/4845 [00:21<00:00, 224.60it/s]
Calculating histograms for SN004: 100%|██████████| 4845/4845 [00:21<00:00, 222.32it/s]
Calculating histograms for SN005: 100%|██████████| 4845/4845 [00:24<00:00, 201.15it/s]
Calculating histograms for SN006: 100%|██████████| 4845/4845 [00:23<00:00, 202.37it/s]
Calculating histograms for SN007: 100%|██████████| 4845/4845 [00:24<00:00, 201.41it/s]
Calculating histograms for SN008: 100%|██████████| 4845/4845 [00:25<00:00, 191.24it/s]
Calculating histograms for SN008: 100%|██████████| 4845/4845 [00:23<00:00, 205.51it/s]
Calculating histograms for SN010: 100%|██████████| 4844/4844 [00:23<00:00, 203.76it/s]
Calculating histograms for SN011: 100%|██████████| 4845/4845 [00:22<00:00, 217.21it/s]
Calculating histograms for SN012: 100%|██████████| 4845/4845 [00:24<00:00, 200.47it/s]
Calculating histograms for SN013: 100%|██████████| 4845/4845 [00:24<00:00, 199.93it/s]
Calculating histograms for SN016: 100%|██████████| 4845/4845 [00:24<00:00, 194.37it/s]
Calculating histograms for SN017: 100%|██████████| 4845/4845 [00:22<00:00, 217.01it/s]
Calculating histograms for SN018: 100%|██████████| 4845/4845 [00:21<00:00, 222.94it/s]
Calculating histograms for SN021: 100%|██████████| 4845/4845 [00:23<00:00, 204.74it/s]
Calculating histograms for SN023: 100%|██████████| 4845/4845 [00:24<00:00, 198.91it/s]
Calculating histograms for SN024: 100%|██████████| 4845/4845 [00:23<00:00, 209.36it/s]
Calculating histograms for SN025: 100%|██████████| 4845/4845 [00:25<00:00, 192.89it/s]
Calculating histograms for SN026: 100%|██████████| 4845/4845 [00:23<00:00, 208.82it/s]
Calculating histograms for SN027: 100%|██████████| 4845/4845 [00:22<00:00, 214.35it/s]
Calculating histograms for SN028: 100%|██████████| 4845/4845 [00:23<00:00, 205.59it/s]
Calculating histograms for SN029: 100%|██████████| 4845/4845 [00:24<00:00, 196.87it/s]

```
Calculating histograms for SN030: 100%|██████████| 4845/4845
[00:23<00:00, 206.08it/s]
Calculating histograms for SN031: 100%|██████████| 4845/4845
[00:23<00:00, 205.39it/s]
Calculating histograms for SN032: 100%|██████████| 4845/4845
[00:23<00:00, 202.12it/s]
```




```
        combined_features = []
        labels = []

        with open('HOG.pkl', 'wb') as f:
            pickle.dump(combined_features, f)
        with open('labels.pkl', 'wb') as f:
            pickle.dump(labels, f)
```

Machine Learning Aspect

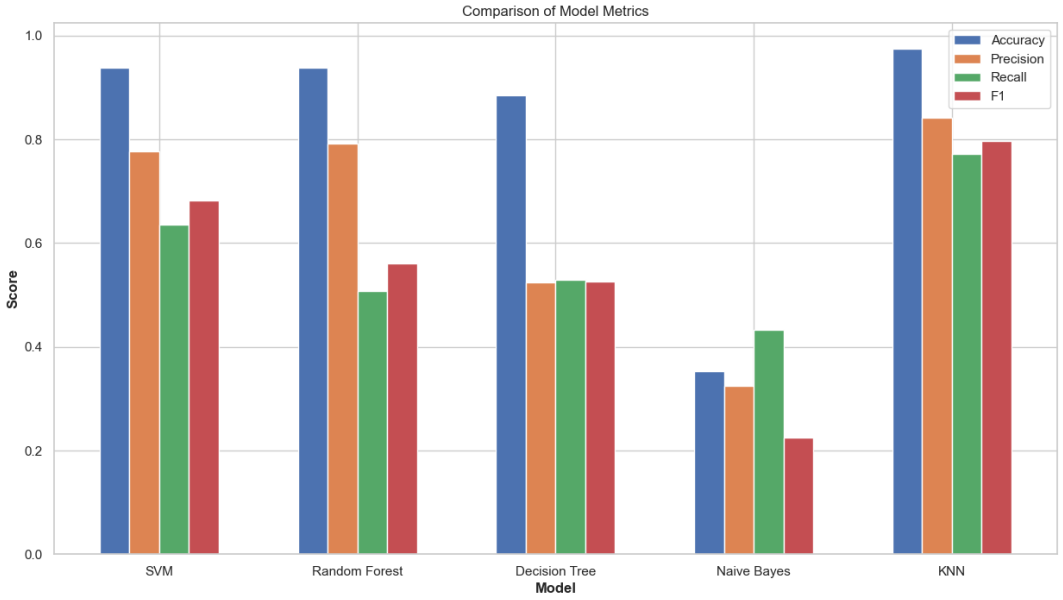
Comparing various ML Classifiers to find the best classifier on the dataset.

```
        X = np.array(combined_features)
        y = np.array(labels)

        # Remove AUs with only one class
        var_threshold = 0.0  # Threshold for variance
        filtered_indices = [i for i in range(y.shape[1]) if np.
        y_filtered = y[:, filtered_indices]

        X_train, X_test, y_train, y_test = train_test_split(X,

        # Define the models to compare
```

```
models = {
    'SVM': MultiOutputClassifier(SVC(kernel='linear', d
    'Random Forest': MultiOutputClassifier(RandomForest
    'Decision Tree': MultiOutputClassifier(DecisionTree
    'Naive Bayes': MultiOutputClassifier(GaussianNB()),
    'KNN': MultiOutputClassifier(KNeighborsClassifier()
}

# Initialize a dictionary to store the metrics for each
model_metrics = {model_name: {'accuracy': [], 'precisio

# Train each model and calculate metrics
for model_name, model in models.items():
    print(f'Training {model_name}...')
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Calculate metrics for each label and take the ave
    accuracies = [accuracy_score(y_test[:, i], y_pred[:
    precisions = [precision_score(y_test[:, i], y_pred[
    recalls = [recall_score(y_test[:, i], y_pred[:, i],
    f1s = [f1_score(y_test[:, i], y_pred[:, i], average

    model_metrics[model_name]['accuracy'].append(np.mea
    model_metrics[model_name]['precision'].append(np.me
    model_metrics[model_name]['recall'].append(np.mean(
    model_metrics[model_name]['f1'].append(np.mean(f1s)

# Plotting the metrics
plt.figure(figsize=(15, 8))
barWidth = 0.15
positions = np.arange(len(models))

for idx, metric in enumerate(['accuracy', 'precision',
    means = [model_metrics[model_name][metric][0] for m
    plt.bar(positions + idx * barWidth, means, width=ba

sns.set(style="whitegrid")
plt.xlabel('Model', fontweight='bold')
plt.ylabel('Score', fontweight='bold')
plt.xticks(positions + 1.5 * barWidth, models.keys())
plt.legend()
plt.title('Comparison of Model Metrics')
plt.show()
```

```
Training SVM...
```

```
# Set the style of seaborn
sns.set(style="whitegrid")

# Plotting the metrics
plt.figure(figsize=(15, 8))
barWidth = 0.15
positions = np.arange(len(models))

for idx, metric in enumerate(['accuracy', 'precision',
    means = [model_metrics[model_name][metric][0] for m
    plt.bar(positions + idx * barWidth, means, width=ba

plt.xlabel('Model', fontweight='bold')
plt.ylabel('Score', fontweight='bold')
plt.xticks(positions + 1.5 * barWidth, models.keys())
plt.legend()
plt.title('Comparison of Model Metrics')
plt.show()
```



Running Random Forest Classifier

```
# Train a multi-output Random Forest classifier
random_forest = RandomForestClassifier(n_estimators=20)
multi_target_rf = MultiOutputClassifier(random_forest,
multi_target_rf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = multi_target_rf.predict(X_test)
```

```
        # Calculate accuracy for each AU
        accuracies = [accuracy_score(y_test[:, i], y_pred[:, i]
        average_accuracy = np.mean(accuracies)
        print(f'Average Accuracy: {average_accuracy}')
        for i, acc in enumerate(accuracies):
            print(f'Accuracy for AU{filtered_indices[i]+1}: {ac
```

```
Average Accuracy: 0.9322445201452676
Accuracy for AU1: 0.8813007384304912
Accuracy for AU2: 0.9739026739439527
Accuracy for AU4: 0.8723570150896666
Accuracy for AU5: 0.9932119433105536
Accuracy for AU6: 0.9223501353024813
Accuracy for AU9: 0.9342292345090125
Accuracy for AU12: 0.9234967665000229
Accuracy for AU17: 0.9944503050038985
Accuracy for AU20: 0.9949089574829152
Accuracy for AU25: 0.8633674265009402
Accuracy for AU26: 0.9011145255240105
```

```
        from sklearn.metrics import precision_score, recall_sco

        # Calculate precision for each AU
        precision = [precision_score(y_test[:, i], y_pred[:, i]
        recall = [recall_score(y_test[:, i], y_pred[:, i], aver
        f1 = [f1_score(y_test[:, i], y_pred[:, i], average='mac

        average_recall = np.mean(recall)
        print(f'Average Recall: {average_recall}')
        for i, rec in enumerate(recall):
            print(f'Recall for AU{filtered_indices[i]+1}: {rec}

        average_f1 = np.mean(f1)
        print(f'Average F1: {average_f1}')
        for i, f1 in enumerate(f1):
            print(f'F1 for AU{filtered_indices[i]+1}: {f1}')

        average_precision = np.mean(precision)
        print(f'Average Precision: {average_precision}')
        for i, prec in enumerate(precision):
            print(f'Precision for AU{filtered_indices[i]+1}: {p
```

```
Average Recall: 0.4981076177032133
Recall for AU1: 0.4505783722425043
Recall for AU2: 0.32246875129449015
Recall for AU4: 0.4732670247470409
Recall for AU5: 0.37806065621842455
```

```
Recall for AU6: 0.5163770056039456
Recall for AU9: 0.5041949845943307
Recall for AU12: 0.6183569665706727
Recall for AU17: 0.366120218579235
Recall for AU20: 0.3333333333333333
Recall for AU25: 0.6217408684709421
Recall for AU26: 0.8946856130804285
Average F1: 0.552270530489258
F1 for AU1: 0.5319042177636241
F1 for AU2: 0.3764087330647401
F1 for AU4: 0.5587925909302742
F1 for AU5: 0.41017745270142164
F1 for AU6: 0.6190476970991656
F1 for AU9: 0.6048738876511303
F1 for AU12: 0.6966942854267592
F1 for AU17: 0.39210679077404675
F1 for AU20: 0.3324903051761929
F1 for AU25: 0.654944447351674
F1 for AU26: 0.897535427442809
Average Precision: 0.7714358632273356
Precision for AU1: 0.8568791911835661
Precision for AU2: 0.7336521322958514
Precision for AU4: 0.8343142454531788
Precision for AU5: 0.6341292730806527
Precision for AU6: 0.8943491061745309
Precision for AU9: 0.8770822036144178
Precision for AU12: 0.8863175338680452
Precision for AU17: 0.6648157496214033
Precision for AU20: 0.3316515304406324
Precision for AU25: 0.8711290462281704
Precision for AU26: 0.9014744835402415

/opt/homebrew/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1334:
UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/homebrew/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1334:
UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/homebrew/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1334:
UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division`
```

```
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Running a K-NN Classifier

```
from sklearn.model_selection import train_test_split
from sklearn.multioutput import MultiOutputClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import numpy as np

# Define and train the KNN classifier within a multi-ou
knn = KNeighborsClassifier(n_neighbors=5)
multi_target_knn = MultiOutputClassifier(knn, n_jobs=-1
multi_target_knn.fit(X_train, y_train)

# Make predictions on the test set
y_pred_knn = multi_target_knn.predict(X_test)

# Calculate accuracy for each AU
accuracies_knn = [accuracy_score(y_test[:, i], y_pred_k
average_accuracy_knn = np.mean(accuracies_knn)
print(f'Average Accuracy with KNN: {average_accuracy_kn
for i, acc in enumerate(accuracies_knn):
    print(f'Accuracy for AU{filtered_indices[i]+1} with
```

```
Average Accuracy with KNN: 0.9749700833496643
Accuracy for AU1 with KNN: 0.9521166811906618
Accuracy for AU2 with KNN: 0.9889464752556988
Accuracy for AU4 with KNN: 0.9528505251570885
Accuracy for AU5 with KNN: 0.9971563546300968
Accuracy for AU6 with KNN: 0.9748658441498876
Accuracy for AU9 with KNN: 0.9763335320827409
Accuracy for AU12 with KNN: 0.9732605604733293
Accuracy for AU17 with KNN: 0.9981195248360317
Accuracy for AU20 with KNN: 0.9970646241342934
Accuracy for AU25 with KNN: 0.9483557308627253
Accuracy for AU26 with KNN: 0.9656010640737513
```

```
from sklearn.metrics import precision_score, recall_sco

# Calculate precision for each AU
precision = [precision_score(y_test[:, i], y_pred_knn[:
recall = [recall_score(y_test[:, i], y_pred_knn[:, i],
f1 = [f1_score(y_test[:, i], y_pred_knn[:, i], average=
```

```
        average_recall = np.mean(recall)
        print(f'Average Recall: {average_recall}')
        for i, rec in enumerate(recall):
            print(f'Recall for AU{filtered_indices[i]+1}: {rec}

        average_f1 = np.mean(f1)
        print(f'Average F1: {average_f1}')
        for i, f1 in enumerate(f1):
            print(f'F1 for AU{filtered_indices[i]+1}: {f1}')

        average_precision = np.mean(precision)
        print(f'Average Precision: {average_precision}')
        for i, prec in enumerate(precision):
            print(f'Precision for AU{filtered_indices[i]+1}: {p
```

```
Average Recall: 0.7720419015932145
Recall for AU1: 0.7791508797661016
Recall for AU2: 0.6866764880824787
Recall for AU4: 0.8067094519233265
Recall for AU5: 0.6622406809762337
Recall for AU6: 0.8676578847501538
Recall for AU9: 0.8046795673376249
Recall for AU12: 0.8793045557443574
Recall for AU17: 0.6037174608698284
Recall for AU20: 0.5698021508962309
Recall for AU25: 0.8681851332332116
Recall for AU26: 0.9643366639458122
Average F1: 0.7963725993980357
F1 for AU1: 0.801644157149407
F1 for AU2: 0.7194225472061132
F1 for AU4: 0.8261523447919691
F1 for AU5: 0.7198793938597162
F1 for AU6: 0.8849961242924121
F1 for AU9: 0.8295989995515552
F1 for AU12: 0.8896647927217078
F1 for AU17: 0.6224918777678395
F1 for AU20: 0.6204536247246007
F1 for AU25: 0.8810583704336977
F1 for AU26: 0.9647363608793746
Average Precision: 0.8420798540848654
Precision for AU1: 0.8287425965068317
Precision for AU2: 0.7651773069274626
Precision for AU4: 0.847668700044793
Precision for AU5: 0.9434388001817796
Precision for AU6: 0.9035164147021562
Precision for AU9: 0.8582544114534228
Precision for AU12: 0.9006162144449708
```

```
Precision for AU17: 0.6441317424111324
Precision for AU20: 0.7094392328872382
Precision for AU25: 0.8967194883429892
Precision for AU26: 0.9651734870307425

/opt/homebrew/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1334:
UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Using Bagging Classifier

```
        # Train a multi-output AdaBoost classifier with RandomF
        ada_boost = AdaBoostClassifier(BaggingClassifier(n_esti
        multi_target_ada = MultiOutputClassifier(ada_boost, n_j
        multi_target_ada.fit(X_train, y_train)

        # Make predictions on the test set
        y_pred = multi_target_ada.predict(X_test)

        # Calculate accuracy, precision, recall, and F1 score f
        accuracies = [accuracy_score(y_test[:, i], y_pred[:, i]
        precisions = [precision_score(y_test[:, i], y_pred[:, i
        recalls = [recall_score(y_test[:, i], y_pred[:, i], ave
        f1_scores = [f1_score(y_test[:, i], y_pred[:, i], avera

        # Calculate and print average metrics
        average_accuracy = np.mean(accuracies)
        average_precision = np.mean(precisions)
        average_recall = np.mean(recalls)
        average_f1_score = np.mean(f1_scores)

        print(f'Average Accuracy: {average_accuracy}')
        print(f'Average Precision: {average_precision}')
        print(f'Average Recall: {average_recall}')
        print(f'Average F1 Score: {average_f1_score}')

        for i, (acc, prec, rec, f1) in enumerate(zip(accuracies
            print(f'AU{filtered_indices[i]+1}: Accuracy={acc},
```

:

Using SMOTE

```
class MultiOutputAdaBoost:
    def __init__(self, base_estimator=None, n_estimator
        self.base_estimator = base_estimator if base_es
        self.n_estimators = n_estimators
        self.estimators_ = []

    def fit(self, X, Y):
        self.estimators_ = []
        smote = SMOTE()  # Initialize SMOTE
        for i in range(Y.shape[1]):
            X_resampled, y_resampled = smote.fit_resamp
            model = AdaBoostClassifier(base_estimator=c
            model.fit(X_resampled, y_resampled)
            self.estimators_.append(model)
        return self

    def predict(self, X):
        predictions = []
        for model in self.estimators_:
            predictions.append(model.predict(X).reshape
        return np.hstack(predictions)

# Initialize the multi-output AdaBoost classifier with
multi_output_adaboost = MultiOutputAdaBoost(base_estima

# Train the classifier
multi_output_adaboost.fit(X_train, y_train)

# Make predictions on the test set
y_pred = multi_output_adaboost.predict(X_test)

# Calculate accuracy for each AU and average accuracy
accuracies = [accuracy_score(y_test[:, i], y_pred[:, i]
average_accuracy = np.mean(accuracies)

# Print the results
print(f'Average Accuracy: {average_accuracy}')
for i, acc in enumerate(accuracies):
    print(f'Accuracy for AU{labels[i]+1}: {acc}')
```

```
# Calculate precision for each AU
precision = [precision_score(y_test[:, i], y_pred[:, i]
recall = [recall_score(y_test[:, i], y_pred[:, i], aver
f1 = [f1_score(y_test[:, i], y_pred[:, i], average='mac

average_recall = np.mean(recall)
print(f'Average Recall: {average_recall}')
```

```
for i, rec in enumerate(recall):
    print(f'Recall for AU{labels[i]+1}: {rec}')

average_f1 = np.mean(f1)
print(f'Average F1: {average_f1}')
for i, f1 in enumerate(f1):
    print(f'F1 for AU{labels[i]+1}: {f1}')

average_precision = np.mean(precision)
print(f'Average Precision: {average_precision}')
for i, prec in enumerate(precision):
    print(f'Precision for AU{labels[i]+1}: {prec}')
```

```
# Create RNN model
model = Sequential([
    LSTM(50, input_shape=(1, X_train.shape[2]), return_
    Dropout(0.2),
    LSTM(50),
    Dropout(0.2),
    Dense(y_filtered.shape[1], activation='sigmoid')
])

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='ad

# Fit the model on training data
model.fit(X_train, y_train, epochs=50, batch_size=64, v

# Evaluate the model
scores = model.evaluate(X_test, y_test, verbose=0)
print(f'RNN Model Accuracy: {scores[1]}')
```