Smart Home Gesture Control Application

Project Part 2

CSE 535: Mobile Computing

Nada Obaid
MSC Cyber Security
nmobaid@asu.edu

***Abstract*—This project introduces a an application to use gestures to control SmartHome devices providing ease of use and train the application in order to control the devices in convenient way that helps the people in need. The application uses mobile computing and machine learning by CNN model to achieve this.**

Keywords—mobile computing, smart home, machine learning,

I. Introduction

The main purpose of this project is to develop a Python application that allocates certain gestures, apply a pre-trained machine learning model for allocating those gestures, and train and test a CNN model.

II. TECHNOLOGY REQUIREMENTS

1. Python 3.10
2. Tensorflow 2.12.0
3. Keras 2.12.0
4. Pandas 1.5.3
5. Scikit-learn 1.2.1
6. Numpy 1.23.4
7. OpencV for Python 4.8.0.74

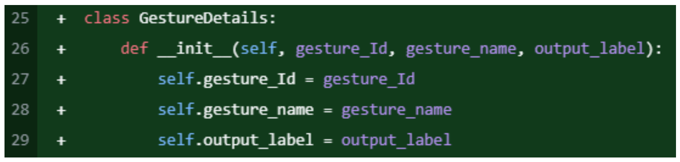
III. SOLUTION

Use the training videos (gestures) from Part 1 of the project to set frames, by extracting the middle frame of the training videos and the feature vector for the middle frame image by the CNN model.

Use the test data videos to get the output label that is resulted from the cosine function by getting the similarity with the feature vectors of all the gesture videos, by extracting the middle frame of the test videos and the feature vector for the middle frame image by the CNN model, then by applying the cosine function to find the similarity with the training vector and getting the vector with the least cosine difference result. The output is in the results.csv file which are the output labels.

IV. SOLUTION STEPS

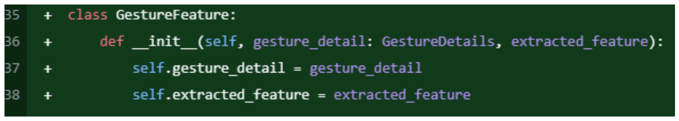
1. Create a class called GestureDetails which is a list of each gesture with its ID, Name, and Output Label:



```
+ class GestureDetails:
+     def __init__(self, gesture_Id, gesture_name, output_label):
+         self.gesture_Id = gesture_Id
+         self.gesture_name = gesture_name
+         self.output_label = output_label
```

Figure 1

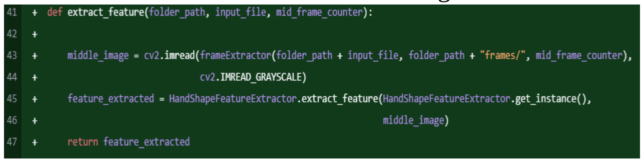
2. Create a class called GestureFeature which manages each gesture and its feature:



```
+ class GestureFeature:
+     def __init__(self, gesture_detail: GestureDetails, extracted_feature):
+         self.gesture_detail = gesture_detail
+         self.extracted_feature = extracted_feature
```

Figure 2

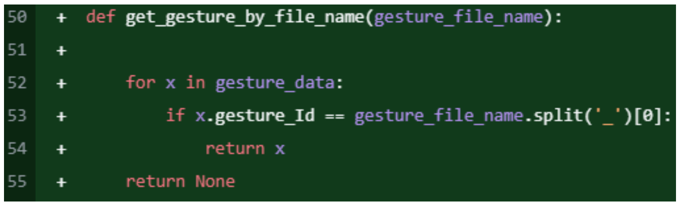
a. The function extract_feature which extracts the features of the middle frame of each image:



```
+ def extract_feature(folder_path, input_file, mid_frame_counter):
+
+     middle_image = cv2.imread(frameExtractor(folder_path + input_file, folder_path + "frames/", mid_frame_counter),
+                               cv2.IMREAD_GRAYSCALE)
+     feature_extracted = HandShapeFeatureExtractor.extract_feature(HandShapeFeatureExtractor.get_instance(),
+                                                                   middle_image)
+     return feature_extracted
```

Figure 2.a

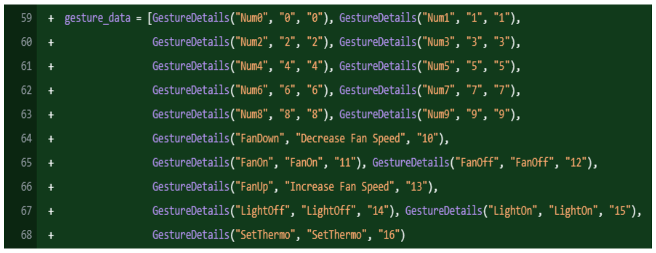
b. The function get_gesture_by_file_name which gets the gesture by its file name:



```
+ def get_gesture_by_file_name(gesture_file_name):
+
+     for x in gesture_data:
+         if x.gesture_Id == gesture_file_name.split('_')[0]:
+             return x
+     return None
```

Figure 2.b

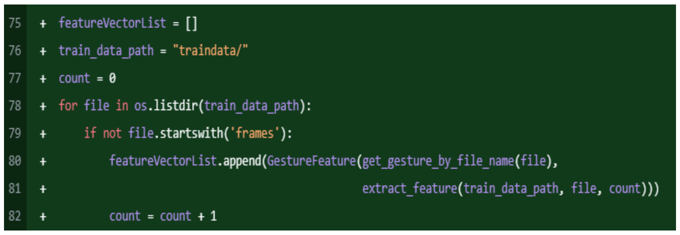
c. The list gesture_data which contains the details of the gestures:



```
+ gesture_data = [GestureDetails("Num0", "0", "0"), GestureDetails("Num1", "1", "1"),
+                 GestureDetails("Num2", "2", "2"), GestureDetails("Num3", "3", "3"),
+                 GestureDetails("Num4", "4", "4"), GestureDetails("Num5", "5", "5"),
+                 GestureDetails("Num6", "6", "6"), GestureDetails("Num7", "7", "7"),
+                 GestureDetails("Num8", "8", "8"), GestureDetails("Num9", "9", "9"),
+                 GestureDetails("FanDown", "Decrease Fan Speed", "10"),
+                 GestureDetails("FanOn", "FanOn", "11"), GestureDetails("FanOff", "FanOff", "12"),
+                 GestureDetails("FanUp", "Increase Fan Speed", "13"),
+                 GestureDetails("LightOff", "LightOff", "14"), GestureDetails("LightOn", "LightOn", "15"),
+                 GestureDetails("SetThermo", "SetThermo", "16")
```

Figure 2.c

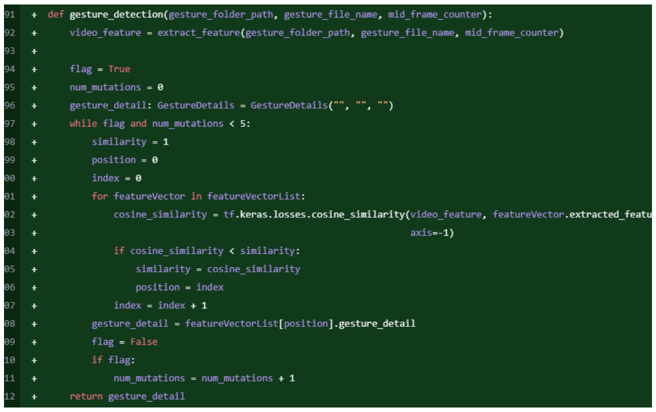
3. Get the training data's middle frame and its gesture details then extract the feature of that middle frame then save the feature vectors into a list:



```
+ featureVectorList = []
+ train_data_path = "traindata/"
+ count = 0
+ for file in os.listdir(train_data_path):
+     if not file.startswith('frames'):
+         featureVectorList.append(GestureFeature(get_gesture_by_file_name(file),
+                                                 extract_feature(train_data_path, file, count)))
+         count = count + 1
```

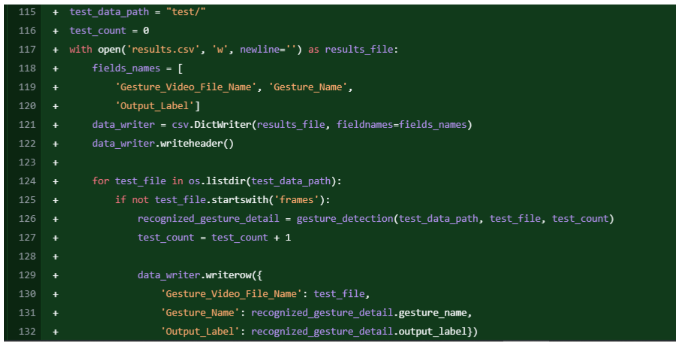
Figure 3

4. Create the function gesture_detection to train the feature vector for the training data which compares it with the test video using the cosine function to decide the label of the gesture in that test video:



```
+ def gesture_detection(gesture_folder_path, gesture_file_name, mid_frame_counter):
+     video_feature = extract_feature(gesture_folder_path, gesture_file_name, mid_frame_counter)
+
+     flag = True
+     num_mutations = 0
+     gesture_detail: GestureDetails = GestureDetails("", "", "")
+     while flag and num_mutations < 5:
+         similarity = 1
+         position = 0
+         index = 0
+         for featureVector in featureVectorList:
+             cosine_similarity = tf.keras.losses.cosine_similarity(video_feature, featureVector.extracted_featu
+                                                                   axis=-1)
+             if cosine_similarity < similarity:
+                 similarity = cosine_similarity
+                 position = index
+             index = index + 1
+         gesture_detail = featureVectorList[position].gesture_detail
+         flag = False
+         if flag:
+             num_mutations = num_mutations + 1
+     return gesture_detail
```

Figure 4.a

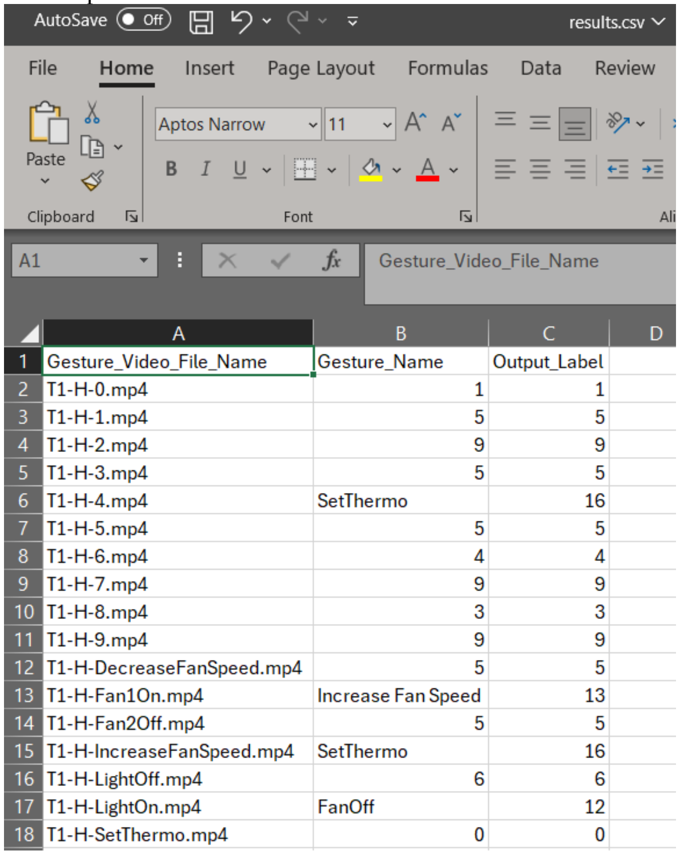


```
+ test_data_path = "test/"
+ test_count = 0
+ with open('results.csv', 'w', newline='') as results_file:
+     fields_names = [
+         'Gesture_Video_File_Name', 'Gesture_Name',
+         'Output_Label']
+     data_writer = csv.DictWriter(results_file, fieldnames=fields_names)
+     data_writer.writeheader()
+
+     for test_file in os.listdir(test_data_path):
+         if not test_file.startswith('frames'):
+             recognized_gesture_detail = gesture_detection(test_data_path, test_file, test_count)
+             test_count = test_count + 1
+
+             data_writer.writerow({
+                 'Gesture_Video_File_Name': test_file,
+                 'Gesture_Name': recognized_gesture_detail.gesture_name,
+                 'Output_Label': recognized_gesture_detail.output_label})
```

Figure 4.b

V. RESULTS

The output is in the results.csv file:



	A	B	C	D
1	Gesture_Video_File_Name	Gesture_Name	Output_Label	
2	T1-H-0.mp4	1	1	
3	T1-H-1.mp4	5	5	
4	T1-H-2.mp4	9	9	
5	T1-H-3.mp4	5	5	
6	T1-H-4.mp4	SetThermo	16	
7	T1-H-5.mp4	5	5	
8	T1-H-6.mp4	4	4	
9	T1-H-7.mp4	9	9	
10	T1-H-8.mp4	3	3	
11	T1-H-9.mp4	9	9	
12	T1-H-DecreaseFanSpeed.mp4	5	5	
13	T1-H-Fan1On.mp4	Increase Fan Speed	13	
14	T1-H-Fan2Off.mp4	5	5	
15	T1-H-IncreaseFanSpeed.mp4	SetThermo	16	
16	T1-H-LightOff.mp4	6	6	
17	T1-H-LightOn.mp4	FanOff	12	
18	T1-H-SetThermo.mp4	0	0	

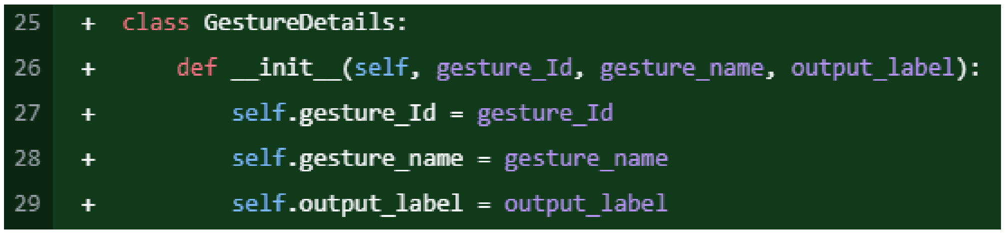
Figure 5

VI. CONCLUSION

The purpose of this project is to increase convenience and create more accessible SmartHome devices for people in need, such as the elderly and people of determination. The project provides a hands-on experience to develop a mobile application to control SmartHome devices with gestures and develop a RESTful application service to classify its gestures using mobile computing and machine learning.

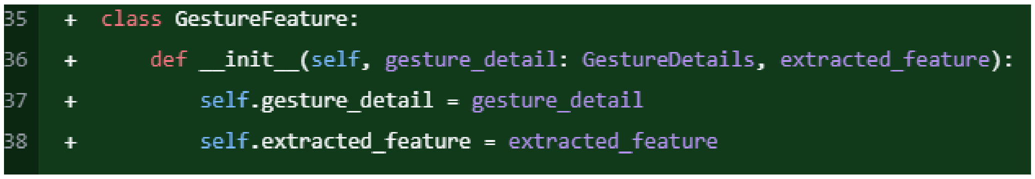
This page contains the screenshots from above for clarity:

Figure 1:



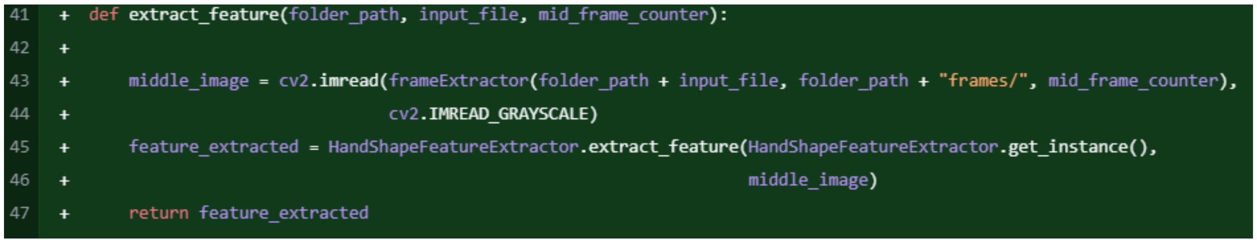
```
+ class GestureDetails:
+     def __init__(self, gesture_Id, gesture_name, output_label):
+         self.gesture_Id = gesture_Id
+         self.gesture_name = gesture_name
+         self.output_label = output_label
```

Figure 2:



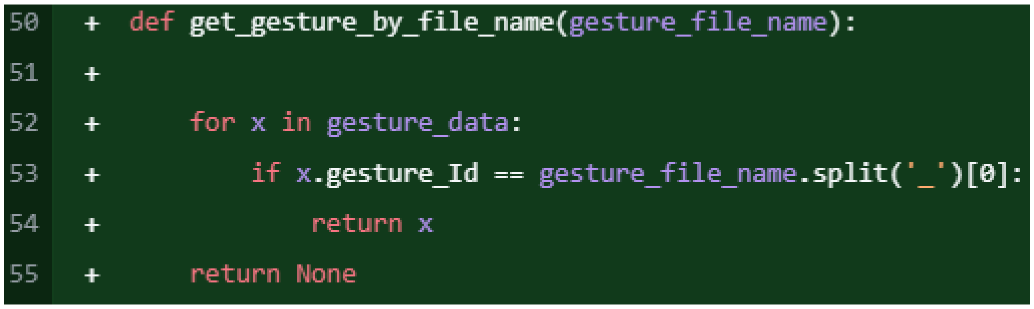
```
+ class GestureFeature:
+     def __init__(self, gesture_detail: GestureDetails, extracted_feature):
+         self.gesture_detail = gesture_detail
+         self.extracted_feature = extracted_feature
```

Figure 2.a:



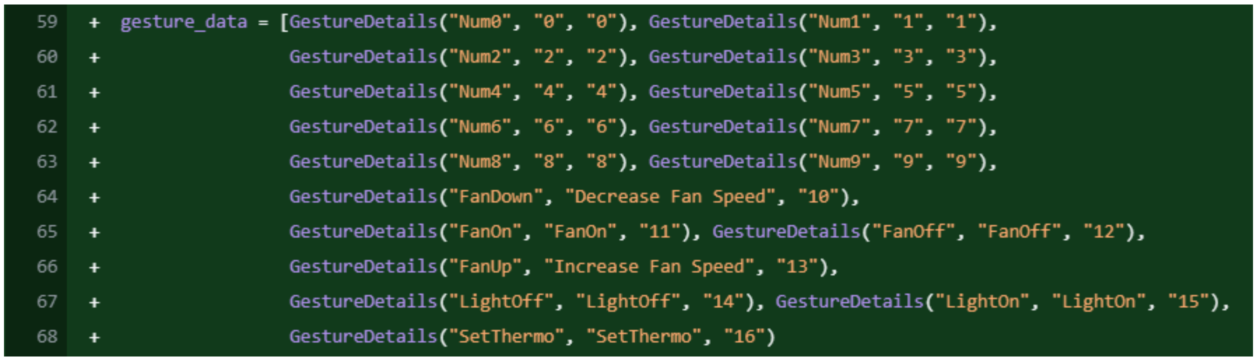
```
+ def extract_feature(folder_path, input_file, mid_frame_counter):
+
+     middle_image = cv2.imread(frameExtractor(folder_path + input_file, folder_path + "frames/", mid_frame_counter),
+                               cv2.IMREAD_GRAYSCALE)
+     feature_extracted = HandShapeFeatureExtractor.extract_feature(HandShapeFeatureExtractor.get_instance(),
+                                                                   middle_image)
+     return feature_extracted
```

Figure 2.b:



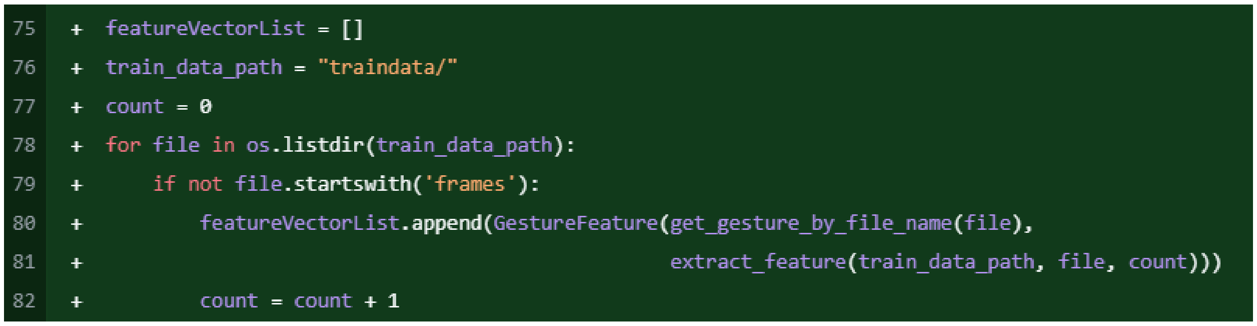
```
+ def get_gesture_by_file_name(gesture_file_name):
+
+     for x in gesture_data:
+         if x.gesture_Id == gesture_file_name.split('_')[0]:
+             return x
+     return None
```

Figure 2.c:



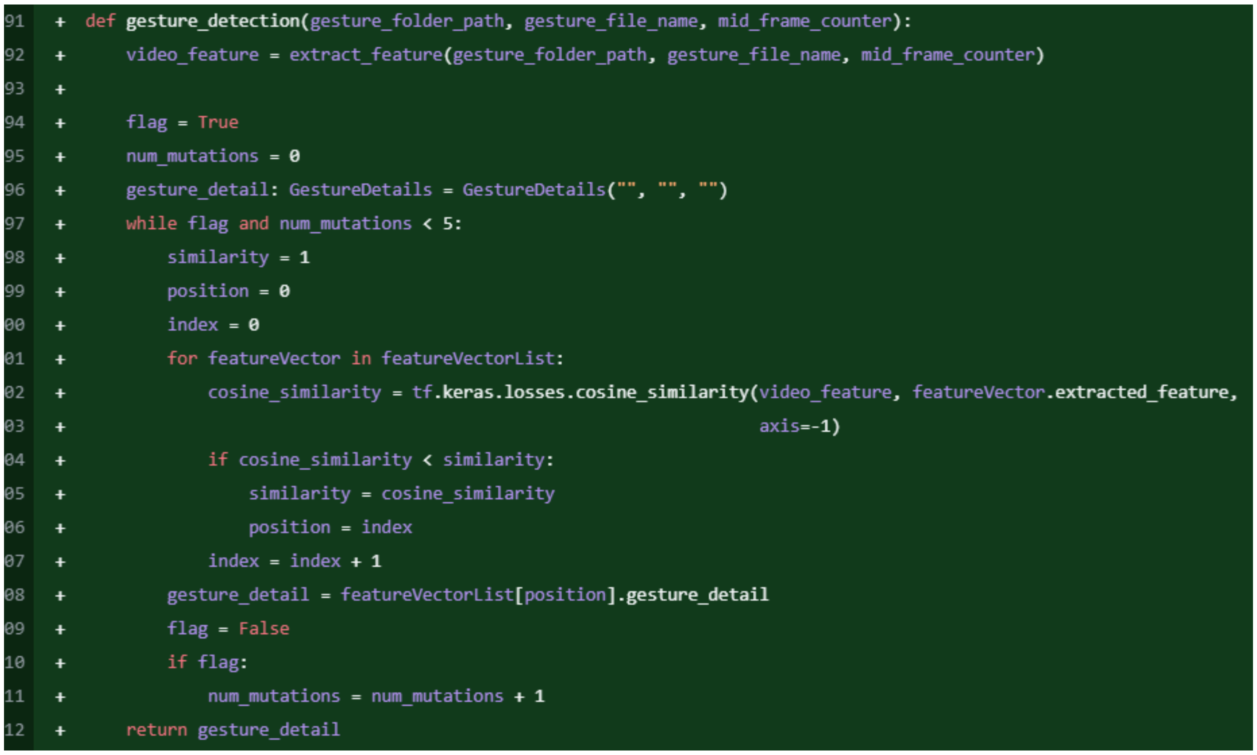
```
+ gesture_data = [GestureDetails("Num0", "0", "0"), GestureDetails("Num1", "1", "1"),
+                 GestureDetails("Num2", "2", "2"), GestureDetails("Num3", "3", "3"),
+                 GestureDetails("Num4", "4", "4"), GestureDetails("Num5", "5", "5"),
+                 GestureDetails("Num6", "6", "6"), GestureDetails("Num7", "7", "7"),
+                 GestureDetails("Num8", "8", "8"), GestureDetails("Num9", "9", "9"),
+                 GestureDetails("FanDown", "Decrease Fan Speed", "10"),
+                 GestureDetails("FanOn", "FanOn", "11"), GestureDetails("FanOff", "FanOff", "12"),
+                 GestureDetails("FanUp", "Increase Fan Speed", "13"),
+                 GestureDetails("LightOff", "LightOff", "14"), GestureDetails("LightOn", "LightOn", "15"),
+                 GestureDetails("SetThermo", "SetThermo", "16")
```

Figure 3:



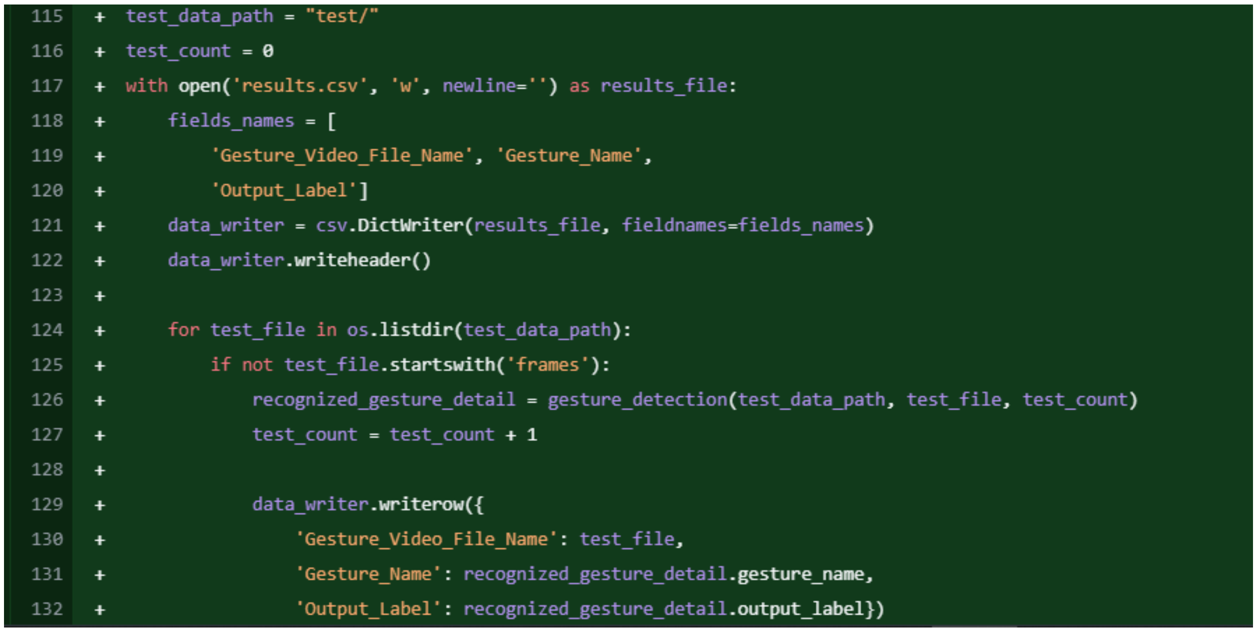
```
+ featureVectorList = []
+ train_data_path = "traindata/"
+ count = 0
+ for file in os.listdir(train_data_path):
+     if not file.startswith('frames'):
+         featureVectorList.append(GestureFeature(get_gesture_by_file_name(file),
+                                                 extract_feature(train_data_path, file, count)))
+         count = count + 1
```

Figure 4.a:



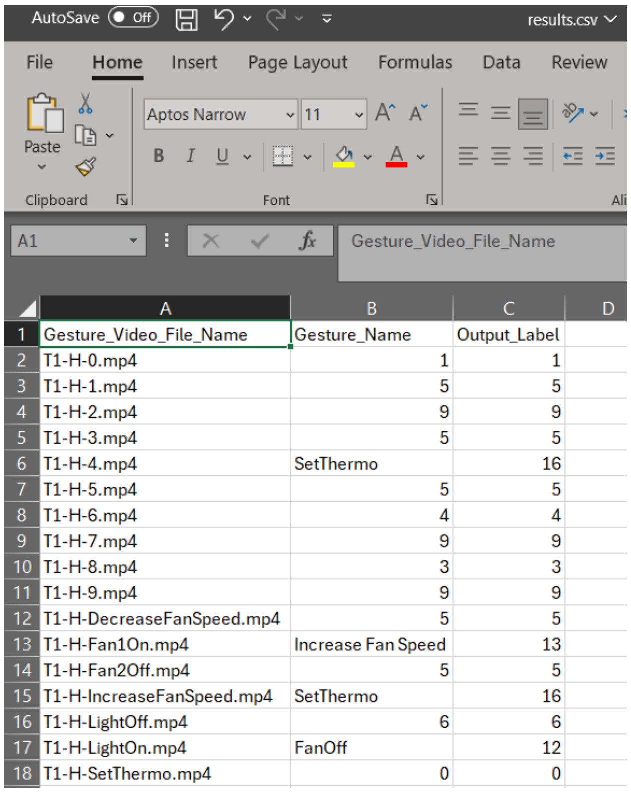
```
+ def gesture_detection(gesture_folder_path, gesture_file_name, mid_frame_counter):
+     video_feature = extract_feature(gesture_folder_path, gesture_file_name, mid_frame_counter)
+
+     flag = True
+     num_mutations = 0
+     gesture_detail: GestureDetails = GestureDetails("", "", "")
+     while flag and num_mutations < 5:
+         similarity = 1
+         position = 0
+         index = 0
+         for featureVector in featureVectorList:
+             cosine_similarity = tf.keras.losses.cosine_similarity(video_feature, featureVector.extracted_feature,
+                                                                   axis=-1)
+             if cosine_similarity < similarity:
+                 similarity = cosine_similarity
+                 position = index
+             index = index + 1
+         gesture_detail = featureVectorList[position].gesture_detail
+         flag = False
+         if flag:
+             num_mutations = num_mutations + 1
+     return gesture_detail
```

Figure 4.b:



```
+ test_data_path = "test/"
+ test_count = 0
+ with open('results.csv', 'w', newline='') as results_file:
+     fields_names = [
+         'Gesture_Video_File_Name', 'Gesture_Name',
+         'Output_Label']
+     data_writer = csv.DictWriter(results_file, fieldnames=fields_names)
+     data_writer.writeheader()
+
+     for test_file in os.listdir(test_data_path):
+         if not test_file.startswith('frames'):
+             recognized_gesture_detail = gesture_detection(test_data_path, test_file, test_count)
+             test_count = test_count + 1
+
+             data_writer.writerow({
+                 'Gesture_Video_File_Name': test_file,
+                 'Gesture_Name': recognized_gesture_detail.gesture_name,
+                 'Output_Label': recognized_gesture_detail.output_label})
```

Figure 5:



Gesture_Video_File_Name

	A	B	C
1	Gesture_Video_File_Name	Gesture_Name	Output_Label
2	T1-H-0.mp4	1	1
3	T1-H-1.mp4	5	5
4	T1-H-2.mp4	9	9
5	T1-H-3.mp4	5	5
6	T1-H-4.mp4	SetThermo	16
7	T1-H-5.mp4	5	5
8	T1-H-6.mp4	4	4
9	T1-H-7.mp4	9	9
10	T1-H-8.mp4	3	3
11	T1-H-9.mp4	9	9
12	T1-H-DecreaseFanSpeed.mp4	5	5
13	T1-H-Fan1On.mp4	Increase Fan Speed	13
14	T1-H-Fan2Off.mp4	5	5
15	T1-H-IncreaseFanSpeed.mp4	SetThermo	16
16	T1-H-LightOff.mp4	6	6
17	T1-H-LightOn.mp4	FanOff	12
18	T1-H-SetThermo.mp4	0	0