Smart Home Gesture Control Application Part2 Report
Name:Wenzhe Zheng
Date:5/2/2021

Introduction:

Students will work to develop a python application classifying Smart Home gestures. Students will gain hands-on experience in training and testing a CNN model for hand gestures. This project, and its successful completion, provides an excellent opportunity to gain further exposure to understand machine learning models.

System Requirements:

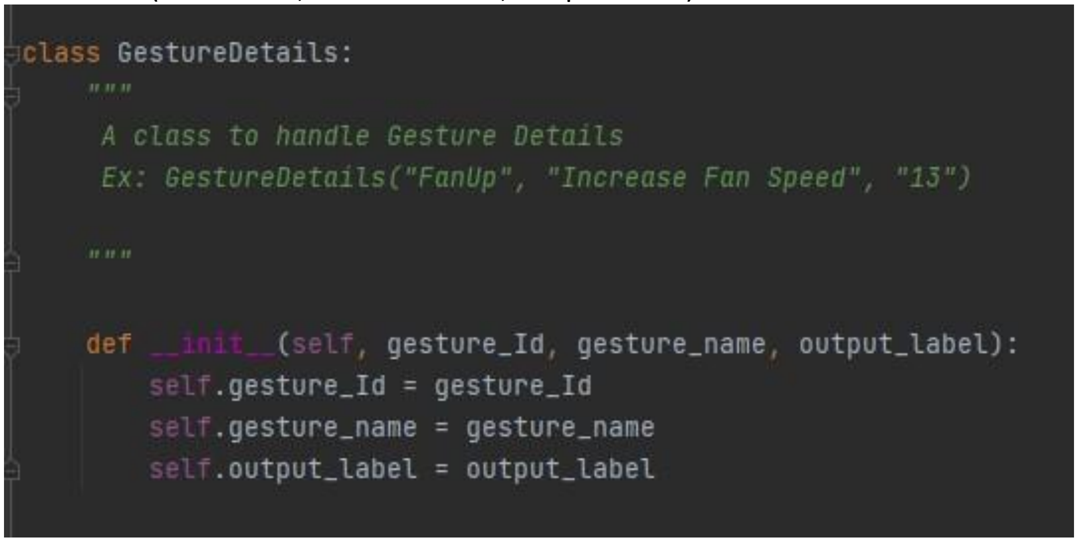
- TensorFlow
- Python 3.6.9
- OpenCV for Python
- Keras

Solution Step:

1. Training set from part 1 is used to prepare a feature vector set for all videos Frames
   - Extract the middle frames of all the training gesture videos
   - Extract feature vector for middle frame image using The CNN model.
   - Save all the feature vectors for all gestures along with their label
2. Test data gesture videos output gesture label will be determined using cosine similarity with all the feature vectors for all gestures
   - Extract the middle frames each test gesture video
   - Extract feature vector for middle frame image using the CNN model.
   - Apply Cosine similarity against training vector set and determine a vector which has minimum cosine difference.
   - Save output label of that video into results.csv

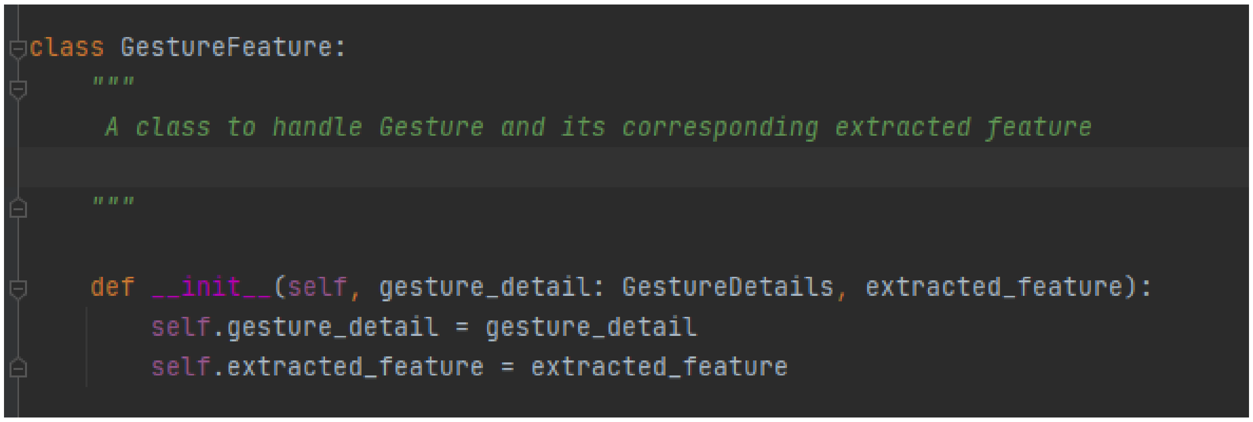
The Implementation Details: Part1: helper functions and classes

1) First we create a class (GestureDetails) to handle the each gesture and its details which are (Gesture ID, Gesture Name, Output Label)



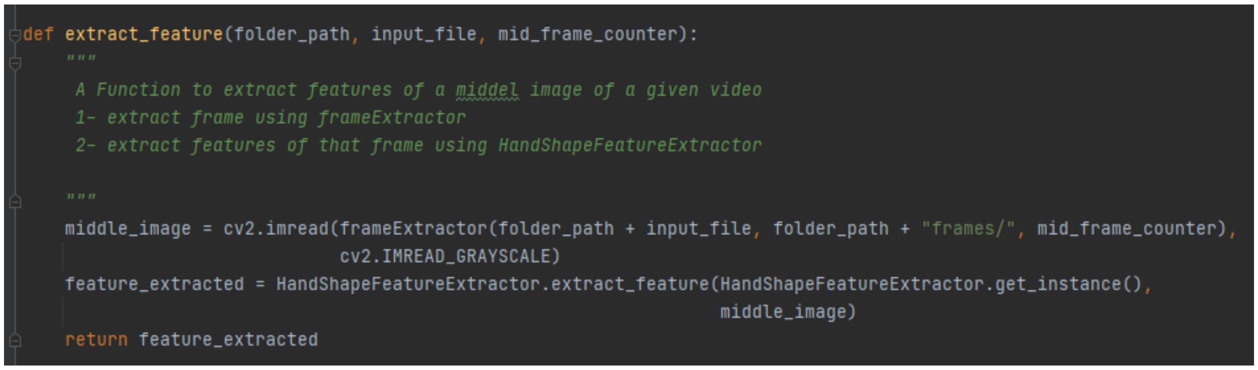
```
class GestureDetails:
    """
    A class to handle Gesture Details
    Ex: GestureDetails("FanUp", "Increase Fan Speed", "13")

    """

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

2) Then we create Class to handle each Gesture and its corresponding extracted feature



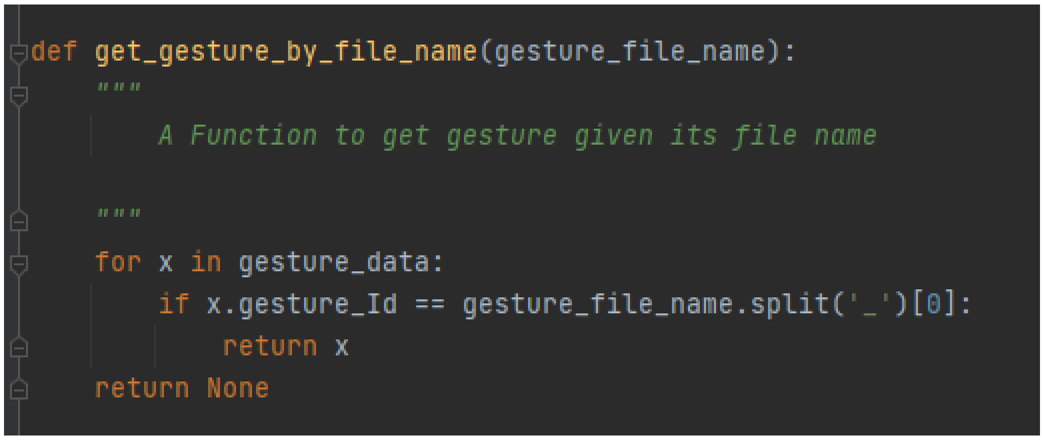
```python
class GestureFeature:
    """
     A class to handle Gesture and its corresponding extracted feature

    """

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

3) Create a function(extract_feature) to be called to extract features of each of each middle frame of each image



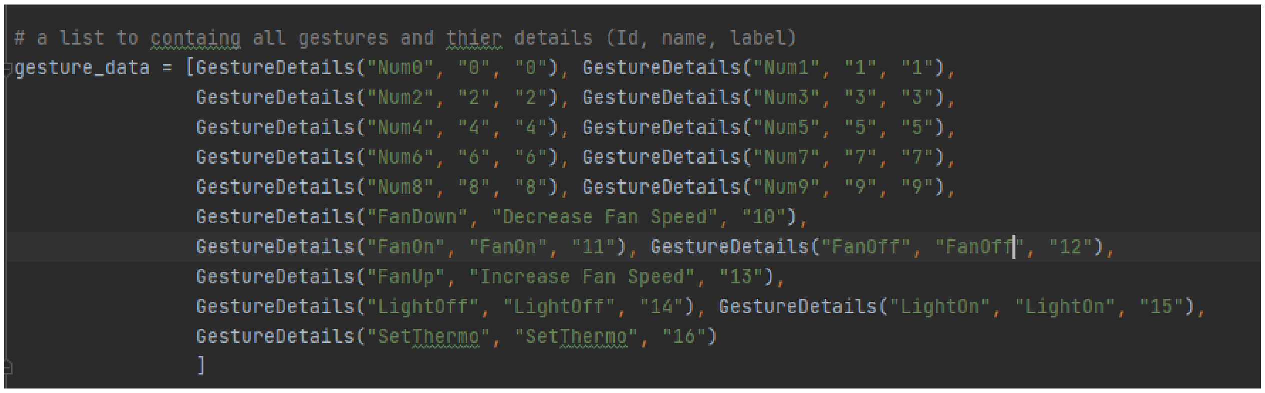
```python
def extract_feature(folder_path, input_file, mid_frame_counter):
    """
     A Function to extract features of a middel image of a given video
     1- extract frame using frameExtractor
     2- extract features of that frame using HandShapeFeatureExtractor

    """
    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
```

4) Create a function to be used to search for a gesture in the given path using its name



```python
def get_gesture_by_file_name(gesture_file_name):
    """
        A Function to get gesture given its file name

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

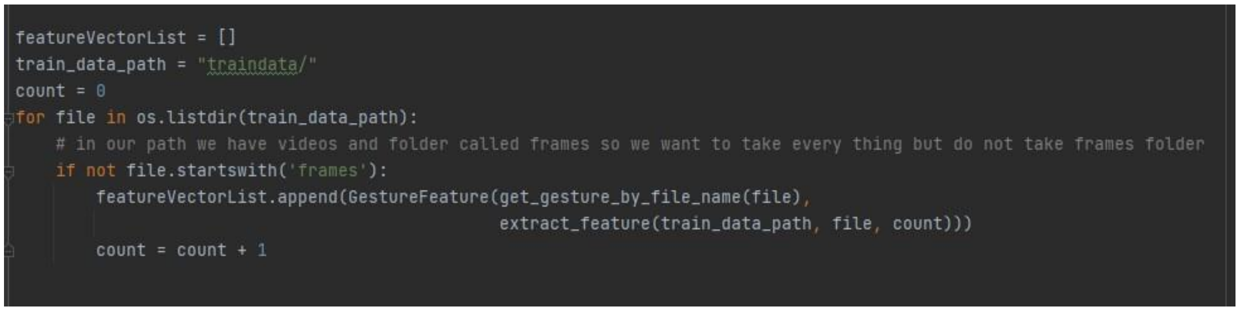
5) Create a list containing all possible gestures and their details



```
# a list to containg all gestures and thier details (Id, name, label)
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")
                ]
```

The Implementation Details: Part2: Get the penultimate layer for the training data

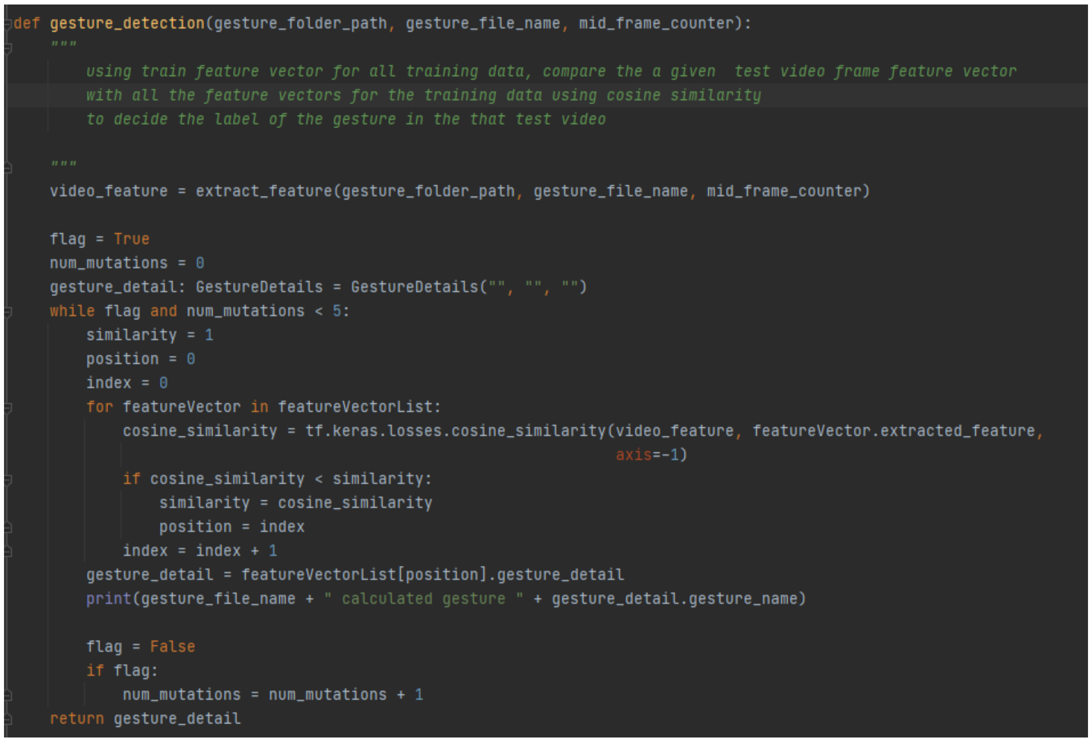
- Loop over each video in the training videos path and get their middle frame and from it get the gesture details and then extract feature of that middle frame
- Then save these feature vectors into a list



```
featureVectorList = []
train_data_path = "traindata/"
count = 0
for file in os.listdir(train_data_path):
    # in our path we have videos and folder called frames so we want to take every thing but do not take frames folder
    if not file.startswith('frames'):
        featureVectorList.append(GestureFeature(get_gesture_by_file_name(file),
                                                extract_feature(train_data_path, file, count)))
        count = count + 1
```

The Implementation Details: Part3: Recognize the gesture using cosine similarity

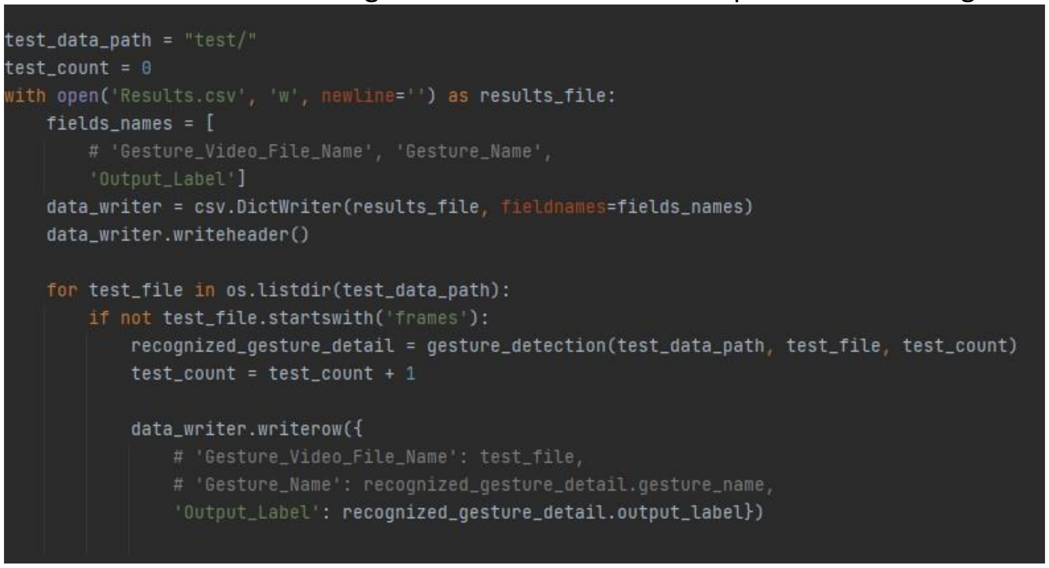
- Create a function to recognize the gesture of the testing videos using cosine similarity with the feature vectors of the training images



```python
def gesture_detection(gesture_folder_path, gesture_file_name, mid_frame_counter):
    """
        using train feature vector for all training data, compare the a given  test video frame feature vector
        with all the feature vectors for the training data using cosine similarity
        to decide the label of the gesture in the that test video

    """
    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
        print(gesture_file_name + " calculated gesture " + gesture_detail.gesture_name)

        flag = False
        if flag:
            num_mutations = num_mutations + 1
    return gesture_detail
```

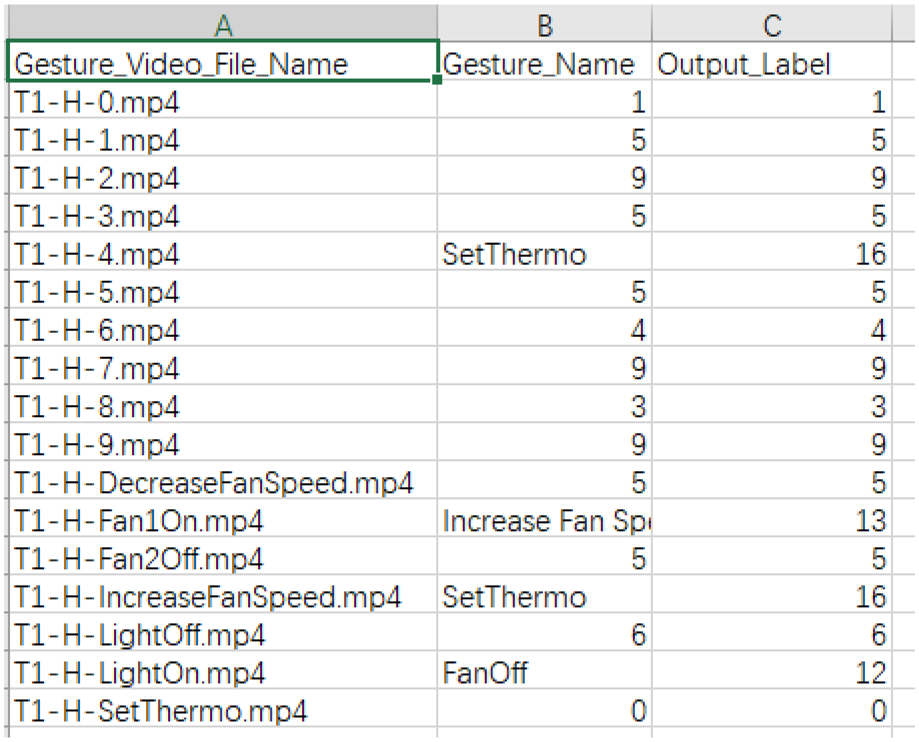
The Implementation Details: Part4: Get the penultimate layer for the testing data

- Loop over each video in the testing videos path and get their middle frame and from it get extract feature of that middle frame and compare it using cosine similarity with the feature vectors of all training videos then decide the output label for that gesture



```python
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})
```

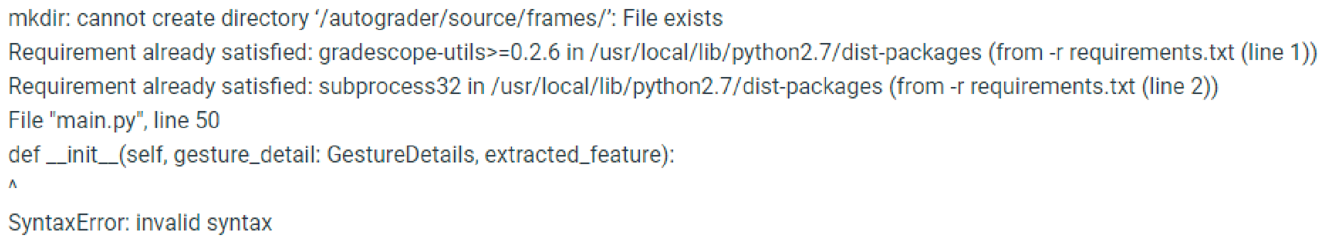
The results:



Gesture_Video_File_Name	Gesture_Name	Output_Label
T1-H-0.mp4	1	1
T1-H-1.mp4	5	5
T1-H-2.mp4	9	9
T1-H-3.mp4	5	5
T1-H-4.mp4	SetThermo	16
T1-H-5.mp4	5	5
T1-H-6.mp4	4	4
T1-H-7.mp4	9	9
T1-H-8.mp4	3	3
T1-H-9.mp4	9	9
T1-H-DecreaseFanSpeed.mp4	5	5
T1-H-Fan1On.mp4	Increase Fan Sp	13
T1-H-Fan2Off.mp4	5	5
T1-H-IncreaseFanSpeed.mp4	SetThermo	16
T1-H-LightOff.mp4	6	6
T1-H-LightOn.mp4	FanOff	12
T1-H-SetThermo.mp4	0	0

GradeScope Upload Failure Analysis:

I got gradescope error like that :



```
mkdir: cannot create directory '/autograder/source/frames/': File exists
Requirement already satisfied: gradescope-utils>=0.2.6 in /usr/local/lib/python2.7/dist-packages (from -r requirements.txt (line 1))
Requirement already satisfied: subprocess32 in /usr/local/lib/python2.7/dist-packages (from -r requirements.txt (line 2))
File "main.py", line 50
def __init__(self, gesture_detail: GestureDetails, extracted_feature):
^
SyntaxError: invalid syntax
```

I think the main reason for the upload failure is because the version I use is Python 3.6, and the error here prompts me that Python 2.7 does not have these things. The most likely reason is that the version is inconsistent.