Assignment-5 – Testing and Debugging

The project is written using Python language, version 3 along with OpenCV libraries. Python comes with a rich testing library/module that helps in unit testing the code along with developing the project in Test Driven Development approach (TDD)

Pycharm was used as the python editor which has rich support for testing and debugging.

1. Unit testing using UnitTest framework : This approach has been widely used in the project to help maintain code relatively bug free. In order to implement this approach, we need to import the unittest module in the test project. The test module should follow the below rules…
   1. Import unittest module
   2. The name of the python module should start with ‘test\_’, else the unit test library won’t treat it as test method
   3. Declare a class starting with ‘Test….’ Like in the below snapshot
   4. The initialization or class level variables can be declared and intantiated in the setup() and setUpClass() methods…

@classmethod  
def setUpClass(cls):  
 cls.tester = sr.app.test\_client()

A screenshot of a cell phone

Description automatically generated

In the above code, I am testing the flask web service calls by using unit testing framework. The module name is test\_StockResource.py and the class name is TestStoreResource.

Each definition (method), ‘**def’** is a test scenario and it tests the functionality of the actual module which StoreResource.py.

Similarly, there is another test module called test\_StockServices.py, which contains similar ‘def’ methods that test the relevant module StockServices.py

Client testing: There are two parts to the project, client and server. The client runs on RaspberryPI and captures pictures and makes a call to server running web service. The client is tested by allowing it send images to server as encoded image document. The client python module has its own test case scenarios written in unittest module.

1. a. Debugging using PyCharm debugger: PyCharm comes with built-in debugger engine that helps set breakpoints and evaluate the variable values.

I have set up breakpoints at critical code locations where the possibility of failure is more and tried to check the values to ensure right data is being passed to these variables. The below snapshot shows how to set up breakpoints for debugging.

A screenshot of a cell phone

Description automatically generated

Once the runtime stops at the required breakpoint, we can evaluate the value of the variables as shown below…

A screenshot of a cell phone

Description automatically generated

b. PDB Process: can also be performed using Python Debugger tool. However, that can be used if we don’t have advanced editors like PyCharm. The process is very similar like setting up breakpoints and evaluating the variables values to ensure expected values are being passed.

Bugs Identified during testing:

Unit testing has caught some subtle bugs like…

1. When the file name provided is incorrect or could be found, in which case, it throws the below error.

A screenshot of a cell phone

Description automatically generated

Fix: The fix was to provide the right path when running in debugging mode.

img = cv2.imread('test-images/' + filename)

1. The image processing also identified unwanted objects like humans etc when the code was expected to identify only physical objects. The reason was because the model used for training contained other labels along with objects.

**Fix**: The fix was to add an array object that ignores all the unwanted labels and let OpenvCV consider only the needed labels like below…

IGNORE = set(["background", "aeroplane", "bicycle", "bird", "boat",  
 "bus", "car", "cat", "chair", "cow", "diningtable",  
 "dog", "horse", "motorbike", "person", "pottedplant", "sheep",  
 "sofa", "train", "tvmonitor"])

1. The file name was hardcoded which ended up overwriting the previous file thus losing the previous images

**Fix**: Hardcoding is removed and the selected filename is passed dynamically to definition (def)

fileName = request.args.get('filename')  
supObj = ss.objectDetection(nparr, net, args, fileName)

1. The client code failed to capture the connection error.

**Fix**: Added try/except block and ensured the connection error is caught.

All the above issues were fixed and tests were rerun to ensure bugs are fixed and retested. Some fine tuning is currently in progress will be complete and submitted before the deadline.

Total lines of Executable code: 2106 loc including runting injection of json proto.