Verification and Validation Report: OAR

Hunter Ceranic

April 14, 2024

1 Revision History

Date	Version	Notes
April 14, 2024	1.0	Initial Revision

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
UT	Unit Test
VnV	Verification and Validation

Contents

This document reports the results of executing the Verification and Validation Plan (**?**).

3 Functional Requirements Evaluation

For the System Functional Requirement tests, the Output Calculate, Input Data Read and Classify Modules were tested with a suite of input images as mentioned in the VnV Plan (**?**). The source code for the tests can be found here: https://github.com/cer-hunter/OAR-CAS741/blob/main/tests/test_req.py. All the tests passed as can be seen in the Test Report 11.1.

However, **T5** and **T7** had some interesting results that should be discussed.

Although performance was not a goal of **T5** it is interesting to note only 3 letters were correctly identified (A, H and M), while the rest were either misclassifed or not classified. On this set of test images it could be said that the model had a 11% accuracy score and a 89% misclassification score.

For **T7** the result of the test was left purposefully vague as it wasn't possible to predict what the output could be for differently oriented letters. If you compare the test output to the test suite itself you can see that the model can accurately predict letters with greater than 50% confidence up to $\pm 15°$. There is one $135°$ image that was classified properly with a high confidence, however I am strongly led to believe this is a fluke, and not an actual feature of the program.

4 Nonfunctional Requirements Evaluation

This section covers the evaluation of the nonfunctional requirements.

4.1 Model Performance

Unfortunately in the interest of time **T8**, **T9** and **T10** could not be fully executed. However accuracy, misclassification and precision were still calculated for the model overall, though comparisons to existing sci-kit learn logistic regression models still need to be completed.

The results are as follows:

1. **T8**: Accuracy Performance Test
   Best Accuracy (Letter M) = 98.7%
   Worst Accuracy (Letter T) = 96.4%
   Overall Accuracy (Entire Model) = 67.4%

2. **T9**: Misclassification Performance Test
   Best Misclassification (Letter M) = 1.3%
   Worst Misclassification (Letter A) = 3.7%
   Overall Misclassification (Entire Model) = 32.5%

3. **T10**: Precision Performance Test
   Best Precision (Letter M) = 87.6%
   Worst Precision (Letter A) = 55.3%
   Note: Precision could not be claculated for the overall model due to the differnce in the way the confusion matrix was generated for the overall model vs for each individual label model. Please see the code in oarTest.py and metrics.py for more details.

4.2 Maintainability

Unfortunately (due to lack of time) the understandability survey **T11** was unable to be conducted yet, so it will still need to be done as part of future work. The results of the rest of the Maintability tests are as follows:

1. **T12**: Static Code Analysis
   Flake8 passed successfully in the CircleCI workflow. Please see https://github.com/cer-hunter/OAR-CAS741/blob/main/.circleci/config.yml to view the configuration of Flake8 used.

2. **T13**: Duck Test Code Review
   The results of the checklist can be seen here:

   - Does the code follow a consistent style?: Yes
   - Are there any variables that should not be global?: No

- Are function names not too long (i.e. less than 40 characters)?: No
- Are most functions well named and clear in what they do?: Yes
- Are there any long lines of that can be split into multiple lines?: No
- Are global variables and functions documented?: Yes
- Is each function modular (not too long and sufficiently broken down to accomplish a single task)?: Not oarTest.py, for future work I would like to decompose this module further.
- Are comments plentiful, and written in a way that makes it easy for beginners to follow?: Yes
- Is the code sorted into separate files where reasonable?: Yes
- Is there any duplicate that code be avoided?: Yes in the GUI, however the GUI was not the main focus of the project, so fixing that can also be left to future work.
- Is there any use of jargon, domain-specific, or unclear terms?: No, as long as the documentation is used in reference to the code.
- Is all of the code reachable?: Yes
- Is the control flow convoluted?: No
- Is there any unnecessarily obscure code? If necessary, is it commented and explained?: I don't think so, most confusing parts of code are commented
- Could a programmer other than the author read the code, and sufficeintly understand it to build upon it?: Yes
- Is there any leftover commented code that is not useful?: No

4.3 Portability

OAR successfully built in the CircleCI Docker Image, so it passes the portability test.

5 Unit Testing

Most files, excluding the 3rd-party library, top-level and GUI modules had corresponding unit tests. All tests were performed in the same file, and each commit to the main branch must pass all the tests. All the tests passed as seen in the Test Report 11.1.

6 Changes Due to Testing

There were a few bugs that were caught as a result of testing, mostly issue when it came to error handling and module imports.

7 Automated Testing

The tests were set up to automatically run in CircleCI whenever a commit was pushed to the `main` branch. The configuration for the CI/CD environment can be found at https://github.com/cer-hunter/OAR-CAS741/blob/main/.circleci/config.yml and the test pipeline can be viewed at https://app.circleci.com/pipelines/github/cer-hunter/OAR-CAS741.

8 Trace to Requirements

9 Trace to Modules

10 Code Coverage Metrics

The code coverage report was given by coverage.py as:

```
Name                 Stmts   Miss  Cover
--------------------------------------
src/__init__.py          1      0   100%
src/classify.py         38      6    84%
src/input.py            23      2    91%
src/oarTrain.py         14      2    86%
src/oarUtils.py         45      2    96%
src/output.py           16      3    81%
```

	R1	R2	R3	R4	R5	NFR1	NFR2	NFR3	NFR4
T1	X	X		X			X		
T2	X	X		X					
T3	X	X		X			X		
T4	X	X		X					
T5	X	X	X	X	X				
T6	X	X	X	X	X				
T7	X	X	X	X	X		X		
T8			X	X	X	X			
T9			X	X	X	X			
T10			X	X	X	X			
T11							X	X	
T12								X	
T13							X	X	
T14									X

Table 1: Traceability Matrix Showing the Connections Between the Tests and Requirements

```
src/preprocess.py      12      0   100%
tests/__init__.py       0      0   100%
tests/test_req.py     157      0   100%
---------------------------------------
TOTAL                 306     15    95%
```

As can be seen 95% of the code that was tested was covered. From what I can gather most of the parts that were "missed" referred to checking if the model.json file was empty or not for the classify.py module and some lines of code that were written to ensure the module imports were connected during testing and when running.

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13
T1					X							X	
T2				X	X	X	X	X				X	
T3					X							X	
T4					X							X	
T5				X	X	X	X	X				X	
T6				X	X	X	X	X				X	
T7				X	X	X	X	X				X	
T8							X			X	X		
T9							X			X	X		
T10							X			X	X		
T11	X	X	X	X	X	X	X	X	X	X	X	X	X
T12	X	X	X	X	X	X	X	X	X	X	X	X	X
T13	X	X	X	X	X	X	X	X	X	X	X	X	X
T14	X	X	X	X	X	X	X	X	X	X	X	X	X
UT1-5								X					
UT6									X				

Table 2: Traceability Matrix Showing the Connections Between the Tests and Requirements

11 Appendix

11.1 Test Report

```
============================= test session starts ==============================
platform linux -- Python 3.11.0, pytest-8.1.1, pluggy-1.4.0
rootdir: /home/circleci/project
configfile: pytest.ini
testpaths: tests
collected 13 items

tests/test_req.py .............                                          [100%]

================================== PASSES ==================================
```

```
------------------------------ test_input_format ------------------------------
----------------------------- Captured stdout call -----------------------------
---------- TEST PASS ----------
--------------------------- test_invalid_input_format ---------------------------
----------------------------- Captured stdout call -----------------------------
---------- TEST PASS ----------
------------------------------ test_input_colors ------------------------------
----------------------------- Captured stdout call -----------------------------
---------- TEST PASS ----------
------------------------------ test_input_size -------------------------------
----------------------------- Captured stdout call -----------------------------
---------- TEST PASS ----------
------------------------------ test_output_labels ------------------------------
----------------------------- Captured stdout call -----------------------------
THE LETTER A 71.37
NOT CLASSIFIED 94.25
THE LETTER K 30.53
THE LETTER X 31.54
THE LETTER X 73.27
THE LETTER X 66.39
THE LETTER X 32.26
THE LETTER H 35.0
THE LETTER X 61.76
THE LETTER X 70.22
THE LETTER X 71.73
THE LETTER X 70.44
THE LETTER M 96.94
THE LETTER X 73.86
THE LETTER X 23.76
THE LETTER T 59.84
THE LETTER U 40.4
THE LETTER X 74.78
THE LETTER X 65.34
THE LETTER X 57.71
THE LETTER X 35.49
THE LETTER X 76.61
THE LETTER N 15.21
THE LETTER X 86.15
THE LETTER X 76.24
THE LETTER X 70.45
```

```
---------- TEST PASS ----------
------------------------------ test_output_degen ------------------------------
----------------------------- Captured stdout call -----------------------------
---------- TEST PASS ----------
------------------------------ test_output_angles ------------------------------
----------------------------- Captured stdout call -----------------------------
THE LETTER A 71.32
THE LETTER A 69.35
THE LETTER A 71.84
THE LETTER A 66.07
THE LETTER A 73.7
THE LETTER A 50.85
THE LETTER A 73.75
NOT CLASSIFIED 92.3
THE LETTER A 73.56
NOT CLASSIFIED 94.99
NOT CLASSIFIED 94.3
THE LETTER A 12.37
NOT CLASSIFIED 96.8
THE LETTER O 11.92
THE LETTER A 86.92
THE LETTER V 13.69
---------- TEST PASS ----------
---------------------------------- test_sigmoid ----------------------------------
----------------------------- Captured stdout call -----------------------------
---------- TEST PASS ----------
------------------------------ test_logLossFunc ------------------------------
----------------------------- Captured stdout call -----------------------------
---------- TEST PASS ----------
---------------------------------- test_predict ----------------------------------
----------------------------- Captured stdout call -----------------------------
---------- TEST PASS ----------
------------------------------ test_gradientW ------------------------------
----------------------------- Captured stdout call -----------------------------
---------- TEST PASS ----------
------------------------------ test_gradientB ------------------------------
----------------------------- Captured stdout call -----------------------------
---------- TEST PASS ----------
---------------------------------- test_train ----------------------------------
----------------------------- Captured stdout call -----------------------------
```

```
---------- TEST PASS ----------
----- generated xml file: /home/circleci/project/test-results/results.xml ------
============================== 13 passed in 1.87s ==============================
```