System Verification and Validation Plan for OCRacle

Phillip Tran

February 21, 2025

Revision History

Date	Version	Notes
February 20, 2025	1.0	Initial Creation

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 Symbols, Abbreviations, and Acronyms

symbol	description
T	Test
OCR	Optical Character Recognition
OAR	Optical Alphabet Recognition, the predecessor to this program
SRS	Software Requirements Specification
VnV	Verification and Validation
MG	Module Guide
MIS	Module Interface Specification
PEP 8	Python Enhancement Proposal 8, the Python style guide
GHA	GitHub Actions

This document outlines the verification and validation plan for the OCRacle program. This document will outline the testing procedures that will be used to ensure that the software meets the requirements outlined in the SRS document.

2 General Information

2.1 Summary

The OCRacle program is being tested. OCRacle is an OCR program that classifies a single handwritten uppercase Latin alphabet character in an image. The project provides a trained model to complete this task as well as a user interface to feed an image into the model and display the results.

2.2 Objectives

The main objective of this project is to build confidence in the software correctness. This will be done by testing the software to ensure that it meets the requirements outlined in the SRS document. This includes testing the accuracy of the software as compared to the OAR predecessor.

For the purposes of this project, the validation of the program's usability will not be tested. This is because the program's user interface will be kept as simple as possible to focus on the OCR functionality.

The project may rely on external libraries for image manipulation or matrix operations. The validation of these libraries will not be tested.

2.3 Challenge Level and Extras

The challenge level for this project is general. Although this project has been done before, implementing the software with higher accuracy than the previous implementation is a challenge.

For the extra task, I will be including a user manual. This will help users understand how to use the program and what to expect from it.

2.4 Relevant Documentation

- Tran (2025) SRS Document: Outlines the requirements for the OCRacle program. This VnV plan will be based on the requirements outlined in this document.

- MG Document: Outlines the modules that compose the OCRacle program. The VnV plan will be based on the modules outlined in this document.
- MIS Document: Outlines the interfaces of the modules that compose the OCRacle program. The VnV plan will be based on the interfaces outlined in this document.

3 Plan

This section outlines the multiple stages of the verification and validation process. First, the VnV team will be introduced. Then the verification plans for the SRS, design, VnV plan, and implementation will be outlined. Finally, a brief overview of automated testing and verification tools will be provided.

3.1 Verification and Validation Team

The following personnel will be involved in the verification and validation of the OCRacle program:

- Phillip Tran: The author of the program. Will be responsible for the verification and validation of the OCRacle program. This includes the creation of the VnV plan, the implementation of the tests, and the analysis of the results.
- Dr. Spencer Smith: The supervisor. Will be responsible for the verification and validation of the OCRacle program. This includes the review of the VnV plan, the review of the tests, and the review of the results.
- Hussein Saad: The domain expert. Will be responsible for the verification and validation of the OCRacle program. This includes the review of the VnV plan, the review of the tests, and the review of the results.

3.2 SRS Verification Plan

To validate the SRS, the domain expert and supervisor have been assigned a GitHub issues to review the document. The author will be responsible for addressing any comments made by the reviewers. As the project progresses the SRS document may be modified, and the reviewers will be assigned a new GitHub issue to review the changes.

To ensure that the SRS document is complete, correct, and consistent, the reviewers can rely on the SRS checklist Smith (2024b).

3.3 Design Verification Plan

The design of the OCRacle program will be verified by the domain expert and supervisor using the MG checklist Smith (2024a) and MIS checklist Smith (2022a).

Reviewers will focus on ensuring that the modules and interfaces are correctly defined for the OCRacle program given the requirements outlined in the SRS

3.4 Verification and Validation Plan Verification Plan

The VnV plan will be reviewed by the domain expert and supervisor using the VnV plan checklist Smith (2022b). The author will be responsible for addressing any comments made by the reviewers. As the project progresses the VnV plan may be modified, and the reviewers will be assigned a new GitHub issue to review the changes.

Reviewers will focus on ensuring that test cases adequately cover any edge cases and are representative of the requirements outlined in the SRS document.

3.5 Implementation Verification Plan

As described in Section 3.6, the OCRacle program will be tested using an automated test suite. Code quality and static type checking will also be enforced using automated tools. To ensure that these automated tests are correct, the VnV team will review the tests and the test results. The VnV team will also review the code quality and static type checking results.

Manual testing in the form of code walkthroughs will also be used to ensure that the code is achieving the desired functionality. The VnV team will review the code walkthroughs to ensure that the code is correct.

3.6 Automated Testing and Verification Tools

GHA will be used to automate the testing of the OCRacle program. For each pull request, the tests will be run. The tests will include unit tests, functional tests, and nonfunctional tests. The tests will be created using the Python unittest framework, which is included in the Python standard library. The tests will be run using the pytest framework, which is included in the Python standard library. The tests will consist of predetermined inputs and expected outputs. The tests will be run on GHA. If any of the tests fail, the pull request will be rejected.

To enforce code quality, the ruff linter and code formatter will be used to ensure that the code follows the PEP 8 style guide and is formatted correctly. In addition to this tool, the mypy static type checker will be used to ensure that the code is correctly typed. For each pull request, the linter, code formatter, and static type checker will be run on GHA. If any of these tools fail, the pull request will be rejected.

3.7 Software Validation Plan

There are no plans for software validation at this time, since it is considered out of scope for this project.

4 System Tests

This section outlines the system tests that will be used to verify the OCRacle program. The tests will be divided into functional and nonfunctional tests.

4.1 Tests for Functional Requirements

THis section contains the tests that verify the functional requirements outlined in the SRS document.

4.1.1 Input Processing Tests

These tests cover R1 and R2 from the SRS document. These requirements state that the program must accept images in JPEG and PNG format and that the program must be able to pre-process these images for classification.

JPEG and PNG Format Acceptance

1. T1: JPEG Format Acceptance

   Control: Automatic

   Initial State: The OCRacle system is running and awaiting input.

   Input: An image in JPEG format containing a single uppercase Latin character.

   Output: The system accepts the image without error and processes it for classification.

   Test Case Derivation: Based on R1, the program must accept images in JPEG format for processing.

   How test will be performed: This automatic test will be run on GHA.

2. T2: PNG Format Acceptance

   Control: Automatic

   Initial State: The OCRacle system is running and awaiting input.

   Input: An image in PNG format containing a single uppercase Latin character.

   Output: The system accepts the image without error and processes it for classification.

   Test Case Derivation: Based on R1, the program must accept images in PNG format for processing.

   How test will be performed: This automatic test will be run on GHA.

3. T3: Non-Supported Format Rejection

   Control: Automatic

Initial State: The OCRacle system is running and awaiting input.

Input: An image in an unsupported format, which is any image format other than JPEG or PNG.

Output: The system rejects the image and displays an error message.

Test Case Derivation: Based on R1, the program must reject images in unsupported formats.

How test will be performed: This automatic test will be run on GHA.

Image Pre-Processing

1. T4: Image Pre-Processing

   Control: Automatic

   Initial State: The OCRacle system is running and awaiting input.

   Input: A valid image containing a single uppercase Latin character.

   Output: The system pre-processes the image for classification, as described in IM1 from the SRS document.

   Test Case Derivation: Based on R2, the program must pre-process images for classification. The system will compare the output from the test with a known correct output to ensure that the pre-processing is correct.

   How test will be performed: This automatic test will be run on GHA.

4.1.2 Character Prediction Tests

As specified in the SRS document, the program must be able to predict a single uppercase Latin character from an image. These tests will verify that the program can correctly predict characters from images.

Single Character Prediction

1. T5: Character Prediction

   Control: Automatic

Initial State: The OCRacle model is trained and ready to predict.

Input: A 28x28 pixel image containing a single uppercase Latin character.

Output: The predicted character, which should match the character in the image.

Test Case Derivation: Based on R3, the program should correctly predict single characters from prepared images. We will use a dataset of known images and their correct label to verify the correctness of the prediction.

How test will be performed: The automatic test will be run on GHA.

4.1.3 Probability Vector Output Tests

These tests ensure that the program outputs a correctly formatted probability vector. The probability vector should contain the probability of each character in the alphabet, and the sum of the probabilities should be 1.

Probability Vector Validity

1. T6: Probability Vector Sum

   Control: Automatic

   Initial State: The OCRacle model is trained and ready to predict.

   Input: A 28x28 pixel image containing a single uppercase Latin character.

   Output: A probability vector where the sum of the probabilities is 1.

   Test Case Derivation: Based on R4, the program must output a probability vector that sums to 1.

   How test will be performed: The automatic test will be run on GHA.

2. T7: Probability Vector Length
   Control: Automatic

   Initial State: The OCRacle model is trained and ready to predict.

   Input: A 28x28 pixel image containing a single uppercase Latin character.

Output: A probability vector where the length is equal to the number of characters in the alphabet, which is 26.

Test Case Derivation: Based on R4, the program must output a probability vector where the length is equal to the number of characters in the alphabet.

How test will be performed: The automatic test will be run on GHA.

4.1.4 Human-Readable Format Tests

These tests ensure that the program outputs the predicted character in a human-readable format.

Readable Character Output

1. T8: Readable Character Output

   Control: Automatic

   Initial State: The OCRacle system is ready to display results.

   Input: A correctly formatted input image.

   Output: The predicted character is displayed in a human-readable format.

   Test Case Derivation: Based on R5, the program must display character predictions in a readable format. We check to see if the UI displays the same character that the model predicted.

   How test will be performed: The automatic test will be run on GHA.

4.2 Tests for Nonfunctional Requirements

This section outlines the tests that will be used to verify the nonfunctional requirements outlined in the SRS document. This includes tests for accuracy, usability, maintainability, and portability.

4.2.1 Accuracy Testing

These tests ensure that the program is accurate in its predictions. It also compares the program's accuracy to the accuracy of the OAR predecessor to ensure that the program is an improvement.

Accuracy Measurement

1. T9: Accuracy Measurement

   Type: Dynamic, Automated

   Initial State: The OCRacle system is trained and ready for testing. The previous OAR project's accuracy metrics are available for comparison. This is an extension of T5, which tests the program's ability to predict characters.

   Input/Condition: A set of test images from the EMNIST dataset that differ from the training set.

   Output/Result: The system outputs a confusion matrix and overall accuracy percentage. Report the relative error compared to previous benchmarks.

   How test will be performed: To support NFR1, the test will be run on GHA. The system will predict the characters in the test images and compare the predictions to the known correct labels. The confusion matrix and overall accuracy percentage will be calculated and compared to the OAR predecessor's accuracy metrics, which have been previously calculated.

4.2.2 Usability Testing

User Manual Usability Test

1. T10: User Manual Usability Test

   Type: Dynamic, Manual

   Initial State: The user manual is prepared and available. The codebase is available for setup and execution.

   Input/Condition: A group of test users with basic command line skills, from the VnV team.

   Output/Result: Feedback on ease of use, clarity of instructions, and any difficulties encountered. The survey will be included in the Appendix.

How test will be performed: To support NFR2, users will follow the user manual to set up and run the OCRacle program. Any issues encountered by the users will be documented via GitHub issues.

4.2.3 Maintainability Testing

Code Review for Modularity

1. T11: Ruff Linter Usability Test

   Type: Static, Automatic

   Initial State: The ruff linter is set up and ready to run.

   Input/Condition: The codebase of the OCRacle project.

   Output/Result: A report on code quality, identifying any issues that need to be addressed. Issues that can automatically be fixed will be fixed. Any issues that cannot be automatically fixed will be documented via failure of the test.

   How test will be performed: To support NFR3, the test will be run on GHA. The linter will check the codebase for any issues.

2. T12: Code Review for Modularity

   Type: Static, Manual

   Initial State: The source code is complete and available for review.

   Input/Condition: The codebase of the OCRacle project.

   Output/Result: A report on code modularity, identifying code sections that are not easily modifiable or understandable.

   How test will be performed: To support NFR3, the VnV team will review the codebase and identify sections that are not easily modifiable or understandable. Any issues will be document via GitHub issues.

4.2.4 Portability Testing

Cross-Platform Compatibility Test

1. T13: Cross-Platform Compatibility Test

   Type: Dynamic, Automatic

   Initial State: The OCRacle system is installed in a Docker environment, which provides cross-platform compatibility.

   Input/Condition: The program and its dependencies are set up on a Docker environment.

   Output/Result: Confirmation that the program runs correctly on the Docker environment.

   How test will be performed: To support NFR4, the test will be run on GHA, which is a dockerized environment. As long as all tests pass, the program is considered to be cross-platform compatible.

4.3 Traceability Between Test Cases and Requirements

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

Table 1: Test Cases to Requirements Matrix

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

Initial State:

Input:

Output: [The expected result for the given inputs —SS]

Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

Spencer Smith. Mis checklist. https://github.com/ptrandev/OCRacle/blob/main/docs/Checklists/MIS-Checklist.pdf, 2022a.

Spencer Smith. System verification and validation plan checklist. https://github.com/ptrandev/OCRacle/blob/main/docs/Checklists/VnV-Checklist.pdf, 2022b.

Spencer Smith. Mg checklist. https://github.com/ptrandev/OCRacle/blob/main/docs/Checklists/MG-Checklist.pdf, 2024a.

Spencer Smith. Srs and ca checklist. https://github.com/ptrandev/OCRacle/blob/main/docs/Checklists/SRS-Checklist.pdf, 2024b.

Phillip Tran. System requirements specification. https://github.com/ptrandev/OCRacle/blob/main/docs/SRS/SRS.pdf, 2025.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]