Optical Alphabet Recognition: System Verification and Validation Plan

Hunter Ceranic

April 13, 2024

Revision History

Date	Version	Notes
February 18, 2024	1.0	Initial Revision
February 19, 2024	1.1	Added Input Image Orientation Test
April 10, 2024	1.2	Updated Document according to feedback from Primary and Secondary Reviewers
April 13, 2024	1.3	Updated Document according to detailed comments from Dr. Smith

Contents

1 Symbols, Abbreviations, and Acronyms

symbol	description
A	Assumption
R	Requirement
SRS	System Requirements Specification
T	Test
VnV	Verification and Validation
OAR	Optical Alphabet Recognition

This document describes the Verification and Validation plan for the OAR Project. Testing of the software and its components will be conducted in accordance with this document to improve confidence in the accuracy, understandibility and ability for the software to meet the requirements laid out in the Software Requirement Specifications (SRS).

2 General Information

2.1 Summary

The OAR project will result in software that is able to accurately classify input images of capital-letter English alphabet charcters using logistic regression. An emphasis is placed on understandable code so that OAR can be a launch point for learning about optical character recognition.

2.2 Objectives

The objective is to build confidence in the correctness of the software, and making sure it is comparable to similar soultions (*accuracy*). The correctness of the output is very easy for a user to evaluate themselves, so it is essential the OAR project not only outputs the correct label, but is able to do so at a high confidence probability level. As a secondary goal, the *understandability* of the code is important, as the project is designed as a learning tool. This goal is a subset of making the code *maintainable*, and will require both thorough code writing during implementation and user input to validate. Parts of the code will rely on external libraries for operations like pre-processing input images, however while it would be beneficial to verify the correctness of these libraries, this is out of the scope of the project and it will be assumed these external libraries will have been verified by their implementation teams.

2.3 Relevant Documentation

There are multiple design documents that provide in-depth details for understanding the program being tested. These are as follows:

- Problem Statement (Ceranic, 2024c): States the problem we are trying to solve and introduces the topics that will need be verified.

- SRS (Ceranic, 2024d): States the requirements, assumptions, models, and important terminology that are used in the VnV plan.
- VnV Report (Ceranic, 2024e): States what has been achieved from the VnV plan, and provides results or evidence to help build confidence for correctness.
- MG (Ceranic, 2024a): States the responsibilities and secrets for each module that will be evaluated by the VnV plan.
- MIS (Ceranic, 2024b): Specifies the functional design and states what is needed for each module that will be evaluated by the VnV plan.

3 Plan

In this section, multiple plans are described to test and inspect the software with an emphasis on *accuracy* and *understandability*. Multiple approaches and perspectives will be employed by the VnV team (3.1) to help build confidence in the requirements, to avoid any missed important details, and to deliver on the outlined objectives (2.2).

3.1 Verification and Validation Team

The members of the VnV team as well their individual roles are listed in the following table:

Role	**Name**
Project Supervisor	Dr. Spencer Smith
Author	Hunter Ceranic
Domain Expert	Adrian Sochaniwsky
SRS Reviewer	Phil Du
VnV Plan Reviewer	Goafeng Zhou
MG + MIS Reviewer	Yiding Li

Table 1: Table of the VnV Team Members

3.2 SRS Verification Plan

The SRS will be reviewed by the project supervisor, the SRS reviewer and the author. Most of the feedback has been provided through issues on GitHub, as annotated documents, or by verbal exchange. Throughout the development of the project, the author is expected to make changes needed to the resolve the issues. The reviewers may refer to the SRS checklist (Smith, 2022c). The key objective is to verify that the software requirements and the documentation are sound and coherent to the intended audience as defined in the SRS.

3.3 Design Verification Plan

Design decisions were made with a focus on performance and code understandability, so that the program could also be used as an educational tool for learning about image recognition. The design and implementation is documented in the MG(Ceranic, 2024a)/MIS(Ceranic, 2024b) which will be verified with the MG and MIS (Smith, 2022a,b) checklists. The VnV team will provide their input using these checklists through GitHub issues.

3.4 Verification and Validation Plan Verification Plan

The goal is to uncover any mistakes and reveal any risks (such as misconceptions or coverage gaps) through the supervision and review of the VnV team members. Once most of the work has been done, the work and accompanying documentation shall undergo a vetting process: the VnV team will check whether the documented testing plans and verification process have been accomplished and the requirements fulfilled. The author will then review the documents against the VnV checklist (Smith, 2022d), before a final review by the rest of the VnV team.

3.5 Implementation Verification Plan

Both automated and manual testing will be performed for this project. This will include checking for any unexpected outputs or bugs that may be encountered. To ensure code consistency, linting tools will be implemented as described in 3.6. Furthermore, tests for the program based on requirements

will be performed using an automated test suite. Manual tests will be performed to benchmark the performance of the OAR project against already existing solutions.

3.6 Automated Testing and Verification Tools

Continous integration will be implemented for this project using CircleCI, which is easily integrated with GitHub and Python the programming language for the project. CircleCI automates the testing pipeline for the project by building the project, running tests created using the pytest library, and linting, on selected pushes or pulls to the repository (Circle Internet Services, 2024). The test created with pytest are performed by predetermining potential user inputs and comparing them with expected values. In addition to the continous integration suite, Flake 8 will be used as a linter for the code created through the VSCode IDE (Cordasco, 2016).

3.7 Software Validation Plan

Software validation is beyond the scope of the OAR project due to the sheer amount of experimental data needed to be collected to validate the system behaviour not being feasible, given the time constraints.

4 System Tests

This section outlines the tests that will be performed on the program to address the functional and nonfunctional requirements found in the SRS (Ceranic, 2024d). The tests and test images are located here: https://github.com/cer-hunter/OAR-CAS741/tree/main/tests

4.1 Tests for Functional Requirements

In this section, the system tests that will be conducted are described in detail. These tests will be used to verify the fulfillment of the requirements as listed in the SRS (Ceranic, 2024d). All of the tests listed here will be automatically performed unless otherwise stated.

4.1.1 Input Image Type

To satisfy R1 from the SRS (Ceranic, 2024d), any input image of the following formats listed below shall be accepted provided they follow the input data constraints.

- PNG
- JPG
- BMP

1. **T1**: Test input for PNG/JPG/BMP format

   Control: Automatic

   Initial State: OAR loaded and idle.

   Input: A subset of non-corrupt (valid) PNG, JPG, and BMP image files depicting valid label characters (see Table 4, Figure 1).

   Output: The program should give a valid output as assigned by the oracle (the author).

   Test Case Derivation: These are some of the most common image file formats and should be compatible with the software.

   How test will be performed: The automatic test will be performed using pytest, executed by CircleCI.

1. **T2**: Test input for Invalid formats

   Control: Automatic

   Initial State: OAR loaded and idle.

   Input: A subset of directory paths pointing to a PDF image file (see Table 5), an empty file path, and no file path.

   Output: The program should output a ValueError for all of these inputs.

   Test Case Derivation: These are some common input paths that would not be valid given the requirements.

   How test will be performed: The automatic test will be performed using pytest, executed by CircleCI.

4.1.2 Unprocessed Input Images

To satisfy R2 from the SRS (Ceranic, 2024d), regardless of colorscale or pixel size any input image should be able to be accepted to be processed into a format usable by the program. In addition, pixel sizes are constrained to the following ranges:

- *max*: 4096px × 4096px
- *min*: 20px × 20px

1. **T3**: Test input for images with different colour schemes

   Control: Automatic

   Initial State: OAR loaded and idle.

   Input: A set of non-corrupt (valid) image files depicting valid label characters (see Table 6) in black and white (Figure 1), grayscale (Figure 5), red (Figure 4), green (Figure 3) and blue (Figure 2).

   Output: The program should give a valid output as assigned by the oracle (the author), meaning some classification result regardless of whether it is correct.

   Test Case Derivation: These images cover the cases of grayscale, black and white and the primary colours of RGB images which should be compatible with the software.

   How test will be performed: The automatic test will be performed using pytest, executed by CircleCI.

2. **T4**: Test input for images with different pixel sizes

   Control: Automatic

   Initial State: OAR loaded and idle.

   Input: A set of non-corrupt (valid) image files depicting valid label characters (see Table 7, Figure 1) of the following sizes:

   - 8192px × 8192px
   - 4096px × 4096px

- 20px × 20px
- 10px × 10px

Output: In the 8192px × 8192px and 10px × 10px cases the program should provide the user with a Value Error. In all other cases, the program should give a valid output as assigned by the oracle (the author), meaning some classification result regardless of whether it is correct.

Test Case Derivation: These images cover the edge cases of acceptable pixel sizes, as well as outside of the range which should be compatible with the software.

How test will be performed: The automatic test will be performed using pytest, executed by CircleCI.

4.1.3 Output Labels

To satisfy R4 and R5 (and by extension R3) from the SRS (Ceranic, 2024d), any input image should be properly classified with a confidence level from the set of 26 letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

1. **T5**: Test Output of Each Possible Label

   Control: Automatic

   Initial State: OAR loaded and idle.

   Input: A set of non-corrupt (valid) image files depicting all 26 valid label characters (see Table 8, Figure 1).

   Output: The predicted label output should match the actual label as assigned by the oracle (the author).

   Test Case Derivation: These images cover the range of possible label outputs for the program.

   How test will be performed: The automatic test will be performed using pytest, executed by CircleCI.

2. **T6**: Test Output of Degenerate Cases

   Control: Automatic

Initial State: OAR loaded and idle.

Input: A set of non-corrupt (valid) image files (see Table 9) depicting degenerate cases such as multiple letters (Figure **??**), a picture (Figure 8), and a blank image depicting no character (Figure 9).

Output: The output for all degenerate cases should state Not Classified.

Test Case Derivation: These images cover the range of possible valid inputs for the program, that should not produce an associated labal.

How test will be performed: The automatic test will be performed using pytest, executed by CircleCI.

3. **T7**: Test Output for input images with different orientations

   Control: Automatic

   Initial State: OAR loaded and idle.

   Input: A set of non-corrupt (valid) image files depicting valid classifiable label characters generated by paint.net (see Table 10, Figure 6) at the following orientations:

   - 0°
   - ±1°
   - ±2°
   - ±5°
   - ±15°
   - ±45°
   - ±90°
   - ±135°
   - 180°

   Output: In the 0% case, the predicted label should match the actual label as assigned by the oracle (the author). In all other cases, the resulting value is unknown, although likely it will be incorrect, as this test is a degenerate edge case test that is covered by A4.

   Test Case Derivation: These images are not compatible with the software according to A4 in the SRS (Ceranic, 2024d), however detecting

that orientation is skewed will be difficult in the scope of the project. Therefore, this test is performed to evaluate the performance of the program in these degenerate cases.

How test will be performed: The automatic test will be performed using pytest, executed by CircleCI.

4.2 Tests for Nonfunctional Requirements

The following tests will check if the nonfunctional requirements, as defined in the SRS (Ceranic, 2024d), are met. The emphasis is on the *accuracy* (NFR1) and *understandability*, and therby *maintainability* (NFR3), of the software. In the case of OAR, correctness and ease of understanding the code both help contribute to being a better learning tool. To satisfy the *usability* (NFR1) requirement, a quick user survey shall be conducted to establish perspective on ease of use as expanded on below. As for *portability* (NFR4), the user should be run the software on their platform and environment of choice.

4.2.1 Accuracy Testing

1. **T8**: Accuracy Performance Test

   Type: Manual

   Condition: The OAR program, and a program built using the Logistic Regression Function from the scikit-learn library (scikit-learn developers, 2024).

   Result: The result of the accuracy calculation should be output to be compared (see 6.1.1)

   How test will be performed: The confusion matrices of both programs will be compared using the accuracy metric.

2. **T9**: Misclassification Performance Test

   Type: Manual

   Condition: The OAR program, and a program built using the Logistic Regression Function from the scikit-learn library (scikit-learn developers, 2024).

Result: The result of the misclassification calculation should be output to be compared (see 6.1.2)

How test will be performed: The confusion matrices of both programs will be compared using the misclassification metric.

3. **T10**: Precision Performance Test

   Type: Manual

   Condition: The OAR program, and a program built using the Logistic Regression Function from the scikit-learn library (scikit-learn developers, 2024).

   Result: The result of the precision calculations should be output to be compared (see 6.1.3)

   How test will be performed: The confusion matrices of both programs will be compared using the precision metric.

4.2.2 Maintainability Testing

1. **T11**: Code Understandability Survey

   Type: Manual

   Condition: A group of intended users, documentation about the supporting libraries, and the survey questions on topics related to the code quality and clarity.

   Result: The completed user surveys (see 6.2).

   Note: The information collected from the surveys will be aggregated and analyzed for any patterns to change or adapt the code to the results.

   How test will be performed: The users will be given a series of questions to evaluate the style and ease of understanding of the code as well as perceptions about how easy it will be to build upon the code for those who are learning.

2. **T12**: Static Code Analysis

   Type: Automatic

Condition: The software code will be tested through the use of Flake 8 (Cordasco, 2016).

Result: The code linters should list zero warnings, and zero errors.

How test will be performed: Flake 8 is a VSCode Extension and it's messages are present at all times while viewing and writing code. A config file may be used to ignore E402 so that module level imports can be at the top of the file.

3. **T13**: Duck Test Code Review

   Type: Manual

   Condition: The software source code and the code review checklist (see 6.3).

   Result: The code should respect the defined code review checklist.

   How test will be performed: The author with walkthrough the code manually and check for qualities such as consistent styling, loose variables, variable scoping, unreachable code, sufficient commenting, etc.

4.2.3 Portability Testing

1. **T14**: Docker Image Test

   Type: Automatic

   Condition: Code pushed to GitHub, connected to a CicleCI jobflow.

   Result: Program Built Successfully.

   Note: CircleCI tests are built and executed in a Docker Image, a container that enables the program to be run on Linux or Windows systems, and ensures libraries and dependencies needed to build the code can be installed.

   How test will be performed: CircleCI will automatically run this test whenever a jobflow is executed.

4.3 Traceability Between Test Cases and Requirements

Traceability matrices are used to simplify the process of identifying what needs to be changed if a component is modified. An "X" is used to indicate links between items in the table. When a component is changed, the elements marked with an "X" might need to be updated as well.

	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	
T14									X

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

5 Unit Test Description

This section focuses on automated testing for individual modules as defined by the MG Ceranic (2024a) and MIS Ceranic (2024b). The unit tests and test image files are located here: https://github.com/cer-hunter/OAR-CAS741/tree/main/tests

5.1 Unit Testing Scope

All of the tests will be ran automatically through CircleCI **??** unless otherwise stated. The functionality of the Output Calculator (M4), Input Data Read (M5), and Input Classifier (M6) Modules are tested by the system test cases listed above and as such, unit tests for these modules are not needed. Additionally since the Metrics (M11), Input Processing (M12), and Graphical User Interface (M13) modules are all implemented by 3rd party libraries, in interest of time they will not be unit tested and it will be assumed that the functions they supply have already been verified. The OAR Model Testing Module (M10) will be tested manually, due to the nature of the module creating the entire model taking an excessively long time to run (too long for free automated testing resources). Although it is possible to test GUIs and visual fidelity to some degree, this is not planned here to due to limited resources (mainly time), and as such the Application Control (M2) and Graphics Display (M3) modules will not be tested. Finally since the OAR Model Data (M7) Module is a record to be written and read from there are no direct functionality tests that can be performed (these are performed by tests on other modules). Unit Testing will focus mainly on the OAR Model Equations (M8) and OAR Model Training (M9) modules and a suite of inputs are provided for each function (see Table 11).

5.2 Tests for Functional Requirements

The tests for each module to be tested will be listed here, and each test will be part of an automated unit test suite.

5.2.1 Module 8 - OAR Model Equations

This module contributes to R3, R4, and R5 by providing the underlying theory and equations needed to construct the model and classify input images.

The test names are related to the function that it tests.

1. **UT1**: Sigmoid

   Type: Automatic

   Input (see Table 11): (**String**)

   Output: *ValueError*

   Test Case Derivation: The input to the Sigmoid Function must be a float or integer.

   How test will be performed: The automatic test will be performed using pytest, executed by CircleCI.

2. **UT2**: LogLossFunc

   Type: Automatic

   Input (see Table 11): (**Valid Integer, Invalid Float**);
   (**Invalid Integer, Valid Float**);
   (**String, String**)

   Output: *ValueError*

   Test Case Derivation: The input trueVal must be an integer either 0 or 1, and the input predVal must be a float $0 \leq \text{predVal} \leq 1$.

   How test will be performed: The automatic test will be performed using pytest, executed by CircleCI.

3. **UT3**: Predict

   Type: Automatic

   Input (see Table 11): (**Matrix 1, Matrix 2, ValidInteger**);
   (**String, String, ValidInteger**);
   (**Matrix 1, Matrix 1, String**)

   Output: *ValueError*

   Test Case Derivation: The inputs inputImage and weight must be matrices of the same size, and the input bias must be an integer or float.

   How test will be performed: The automatic test will be performed using pytest, executed by CircleCI.

4. **UT4**: GradientW

   Type: Automatic

   Input (see Table 11): (**Matrix 1, String, Matrix 1, ValidInteger, ValidInteger, ValidInteger**);
   (**Matrix 1, InvalidInteger, Matrix 1, ValidInteger, ValidInteger, ValidInteger**);
   (**Matrix 1, ValidInteger, Matrix 1, ValidInteger, String, ValidInteger**);
   (**Matrix 1, ValidInteger, Matrix 1, ValidInteger, ValidInteger, String**)

   Output: *ValueError*

   Test Case Derivation: The inputs inputImage, weight and bias will be tested by **UT3** as the Predict Function is called in this function. The input trueVal must be an integer either 0 or 1, the input regParam must be a float, and the input trainSize must be an integer.

   How test will be performed: The automatic test will be performed using pytest, executed by CircleCI.

5. **UT5**: GradientB

   Type: Automatic

   Input (see Table 11):(**Matrix 1, String, Matrix 1, ValidInteger**);
   (**Matrix 1, InvalidInteger, Matrix 1, ValidInteger**)

   Output: *ValueError*

   Test Case Derivation: The inputs inputImage, weight and bias will be tested by **UT3** as the Predict Function is called in this function. The input trueVal must be an integer either 0 or 1.

   How test will be performed: The automatic test will be performed using pytest, executed by CircleCI.

5.2.2 Module 9 - OAR Model Training

This module contributes to R3 by providing the underlying theory and equations needed to train the classification model. The test names are related to

the function that it tests, and the inputs are derived directly from the code, rather than the MIS document.

1. **UT6**: Train

   Type: Automatic

   Input (see Table 11): (**Matrix 1, ValidInteger, Matrix 1, ValidInteger, ValidFloat, String, ValidInteger**)

   Output: *ValueError*

   Test Case Derivation: The inputs inputImage, trueVal, weight, bias, regParam, and trainSize will be tested by **UT3**, **UT4** and **UT5** as the Predict, GradientW and GradientB Functions are called in this function. The input alpha must be a float.

   How test will be performed: The automatic test will be performed using pytest, executed by CircleCI.

5.3 Tests for Nonfunctional Requirements

There are no Unit Tests for Non-Functionl Requirements.

5.4 Traceability Between Test Cases and Modules

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13
T1					X							X	
T2					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 3: Traceability Matrix Showing the Connections Between the Tests and Requirements

References

Hunter Ceranic. Module guide. https://github.com/cer-hunter/OAR-CAS741/blob/main/docs/Design/SoftArchitecture/MG.pdf, 2024a.

Hunter Ceranic. Module interface specification. https://github.com/cer-hunter/OAR-CAS741/blob/main/docs/Design/SoftDetailedDes/MIS.pdf, 2024b.

Hunter Ceranic. Problem statement and goals. https://github.com/cer-hunter/OAR-CAS741/blob/main/docs/ProblemStatementAndGoals/ProblemStatement.pdf, 2024c.

Hunter Ceranic. System requirements specification. https://github.com/cer-hunter/OAR-CAS741/blob/main/docs/SRS/SRS.pdf, 2024d.

Hunter Ceranic. Verification and validation report. https://github.com/cer-hunter/OAR-CAS741/blob/main/docs/VnVReport/VnVReport.pdf, 2024e.

Inc. Circle Internet Services. Circleci - continous integration product and features. https://circleci.com/product/, 2024.

Ian Stapleton Cordasco. Flake8: Your tool for style guide enforcement. https://flake8.pycqa.org/en/latest/, 2016.

scikit-learn developers. scikit-learn: Machine learnsing in python. https://scikit-learn.org/stable/, 2024.

Dr. S. Smith. MG checklist, 2022a. URL https://github.com/smiths/capTemplate/blob/main/docs/Checklists/MG-Checklist.pdf.

Dr. S. Smith. MIS checklist, 2022b. URL https://github.com/smiths/capTemplate/blob/main/docs/Checklists/MIS-Checklist.pdf.

Dr. S. Smith. SRS and CA checklist, 2022c. URL https://github.com/smiths/capTemplate/blob/main/docs/Checklists/SRS-Checklist.pdf.

Dr. S. Smith. System verification and validation plan checklist, 2022d. URL https://github.com/smiths/capTemplate/blob/main/docs/Checklists/VnV-Checklist.pdf.

6 Appendix

6.1 Calculation Definitions

6.1.1 Accuracy Calculation

Using generated confusion matrices the metrics for a given label prediction can be produced. The percent of predictions made that represent True Positive (TA), False Negative (FN), False Positive (FP), and True Negative (TN) values can be obtained from that we define the definition of accuracy as:

$$\frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

We can then use this definition to evaluate the overall accuracy of the OAR model, by comparing confusion matrices of the OAR model and any Library model using the following equation which calculates the relative error between the two models:

$$\frac{|OARAccuracy - LibraryAccuracy|}{|LibraryAccuracy|} \times 100\% \tag{2}$$

6.1.2 Misclassification Calculation

Using generated confusion matrices the metrics for a given label prediction can be produced. The percent of predictions made that represent True Positive (TA), False Negative (FN), False Positive (FP), and True Negative (TN) values can be obtained from that we define the definition of misclassification as:

$$\frac{FP + FN}{TP + TN + FP + FN} \tag{3}$$

We can then use this definition to evaluate the overall misclassifications of the OAR model, by comparing confusion matrices of the OAR model and any Library model using the following equation which calculates the relative error between the two models:

$$\frac{|OARMisclassification - LibraryMisclassification|}{|LibraryMisclassification|} \times 100\% \tag{4}$$

6.1.3 Precision Calculation

Using generated confusion matrices the metrics for a given label prediction can be produced. The percent of predictions made that represent True Positive (TA), False Negative (FN), False Positive (FP), and True Negative (TN) values can be obtained from that we define the definition of precision as:

$$\frac{TP}{TP + FP} \tag{5}$$

We can then use this definition to evaluate the overall precision of the OAR model, by comparing confusion matrices of the OAR model and any Library model using the following equation which calculates the relative error between the two models:

$$\frac{|OARPrecision - LibraryPrecision|}{|LibraryPrecision|} \times 100\% \tag{6}$$

6.2 Understandability Survey Questions

- On a scale from 1 to 10, how easy was it to understand the code?
- Were there any OAR specific functions or variable names that caused confusion?
- Were there any parts of the code that seemed not useful or uneccessary?
- What part of the code was the most confusing/unclear?
- Was the use of comments adequate?
- Do you think you understand the code enough to be able to build upon it further?
- Would you consider using the software as a learning or teaching tool?
- Do you have any extra comments or thoughts you'd like to share?

6.3 Duck Test Checklist

- Does the code follow a consistent style?
- Are there any variables that should not be global?
- Are function names not too long (i.e. less than 40 characters)?
- Are most functions well named and clear in what they do?
- Are there any long lines of that can be split into multiple lines?
- Are global variables and functions documented?
- Is each function modular (not too long and sufficiently broken down to accomplish a single task)?
- Are comments plentiful, and written in a way that makes it easy for beginners to follow?
- Is the code sorted into separate files where reasonable?
- Is there any duplicate that code be avoided?
- Is there any use of jargon, domain-specific, or unclear terms?
- Is all of the code reachable?
- Is the control flow convoluted?
- Is there any unnecessarily obscure code? If necessary, is it commented and explained?
- Could programmer other than the author read the code, and sufficeintly understand it to build upon it?
- Is there any leftover commented code that is not useful?

6.4 Figures

Hyperlink	Description	Expected Output
A.bmp	.BMP file	Input Processed Successfully
A.jpg	.JPG file	Input Processed Successfully
A.png	.PNG file	Input Processed Successfully

Table 4: Table of Valid Extension Images

Hyperlink	Description	Expected Output
A.pdf	.PDF file	ValueError
Empty	Empty Folder	ValueError
None	No Path Passed	ValueError

Table 5: Table of Invalid Extension Images

Hyperlink	Description	Expected Output
A.png	Black and White A	Input Processed Successfully
Blue A	Blue A	Input Processed Successfully
Green A	Green A	Input Processed Successfully
Red A	Red A	Input Processed Successfully
Gray A	Gray A	Input Processed Successfully

Table 6: Table of Images with Different Colours

Hyperlink	Description	Expected Output
Too Big	Image of A of size 8192px × 8192px	Value Error
Big Edge	Image of A of size 4096px × 4096px	Input Processed Successfully
Small Edge	Image of A of size 20px × 20px	Input Processed Successfully
Too Small	Image of A of size 10px × 10px	Value Error

Table 7: Table of Images of Different Sizes in pixels

Hyperlink	Description	Expected Output
A.jpg	Letter A	Classification and Confidence Level Displayed
B.jpg	Letter B	Classification and Confidence Level Displayed
C.jpg	Letter C	Classification and Confidence Level Displayed
D.jpg	Letter D	Classification and Confidence Level Displayed
E.jpg	Letter E	Classification and Confidence Level Displayed
F.jpg	Letter F	Classification and Confidence Level Displayed
G.jpg	Letter G	Classification and Confidence Level Displayed
H.jpg	Letter H	Classification and Confidence Level Displayed
I.jpg	Letter I	Classification and Confidence Level Displayed
J.jpg	Letter J	Classification and Confidence Level Displayed
K.jpg	Letter K	Classification and Confidence Level Displayed
L.jpg	Letter L	Classification and Confidence Level Displayed
M.jpg	Letter M	Classification and Confidence Level Displayed
N.jpg	Letter N	Classification and Confidence Level Displayed
O.jpg	Letter O	Classification and Confidence Level Displayed
P.jpg	Letter P	Classification and Confidence Level Displayed
Q.jpg	Letter Q	Classification and Confidence Level Displayed
R.jpg	Letter R	Classification and Confidence Level Displayed
S.jpg	Letter S	Classification and Confidence Level Displayed
T.jpg	Letter T	Classification and Confidence Level Displayed
U.jpg	Letter U	Classification and Confidence Level Displayed
V.jpg	Letter V	Classification and Confidence Level Displayed
W.jpg	Letter W	Classification and Confidence Level Displayed
X.jpg	Letter X	Classification and Confidence Level Displayed
Y.jpg	Letter Y	Classification and Confidence Level Displayed
Z.jpg	Letter Z	Classification and Confidence Level Displayed

Table 8: Table of Images of Different Letters

Hyperlink	Description	Expected Output
ABC.jpg	Image of Letters A, B and C	Classification and Confidence level displayed stating
hunter.jpg	Picture of the Author	Classification and Confidence level displayed stating
Blank.jpg	Blank Image	Classification and Confidence level displayed stating

Table 9: Table of Images of Degenerate Input Cases

Hyperlink	Description	Expected Output
angle0.png	Image at 0 degrees Orientation	Classification and Confidence Level Displa
angle1p.png	Image at +1 degrees Orientation	Classification and Confidence Level Displa
angle1n.png	Image at -1 degrees Orientation	Classification and Confidence Level Displa
angle2p.png	Image at +2 degrees Orientation	Classification and Confidence Level Displa
angle2n.png	Image at -2 degrees Orientation	Classification and Confidence Level Displa
angle5p.png	Image at +5 degrees Orientation	Classification and Confidence Level Displa
angle5n.png	Image at -5 degrees Orientation	Classification and Confidence Level Displa
angle15p.png	Image at +15 degrees Orientation	Classification and Confidence Level Displa
angle15n.png	Image at -15 degrees Orientation	Classification and Confidence Level Displa
angle45p.png	Image at +45 degrees Orientation	Classification and Confidence Level Displa
angle45n.png	Image at -45 degrees Orientation	Classification and Confidence Level Displa
angle90p.png	Image at +90 degrees Orientation	Classification and Confidence Level Displa
angle90n.png	Image at -90 degrees Orientation	Classification and Confidence Level Displa
angle135p.png	Image at +135 degrees Orientation	Classification and Confidence Level Displa
angle135n.png	Image at -135 degrees Orientation	Classification and Confidence Level Displa
angle180.png	Image at 180 degrees Orientation	Classification and Confidence Level Displa

Table 10: Table of Images of Different Angles

Name	Code Description
String	A string input that takes the form: *test*
Valid Integer	An integer of the value: 1
Valid Float	A float of the value: 0.5
Invalid Integer	An integer of the value: 2
Invalid Float	A float of the value: -5.4
Matrix 1	A matrix of ones of the size 3×1
Matrix 2	A matrix of zeros of the size 2×2

Table 11: Table of Unit Test Inputs

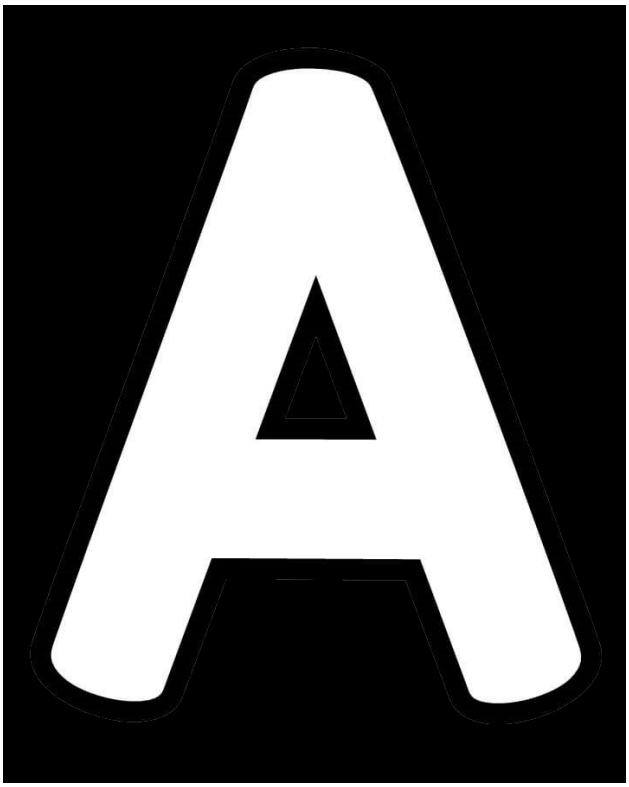

Figure 1: This image has been converted into .bmp, .jpg, and .png formats, and similar images exist for each letter of the English alphabet. This example picture can also be rescaled using paint.net

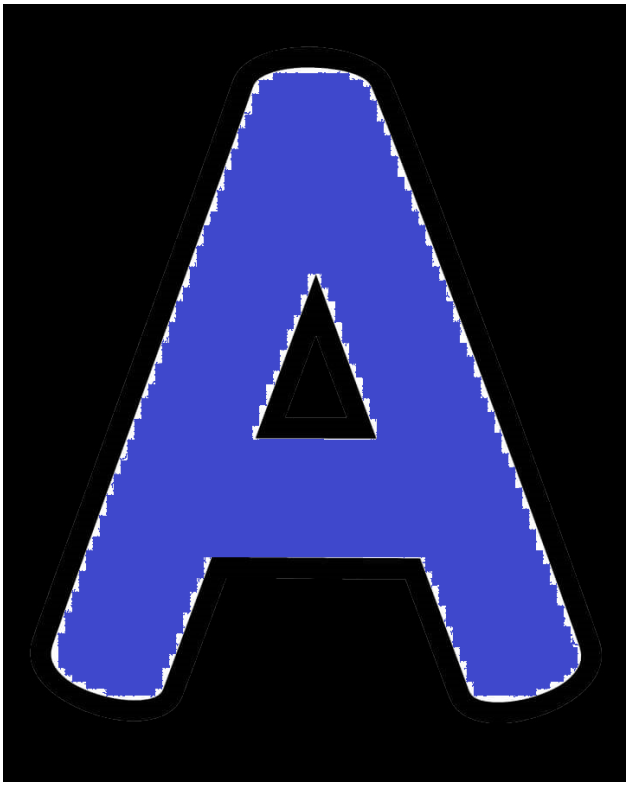

Figure 2: This image depicts a Blue A

Figure 3: This image depicts a Green A

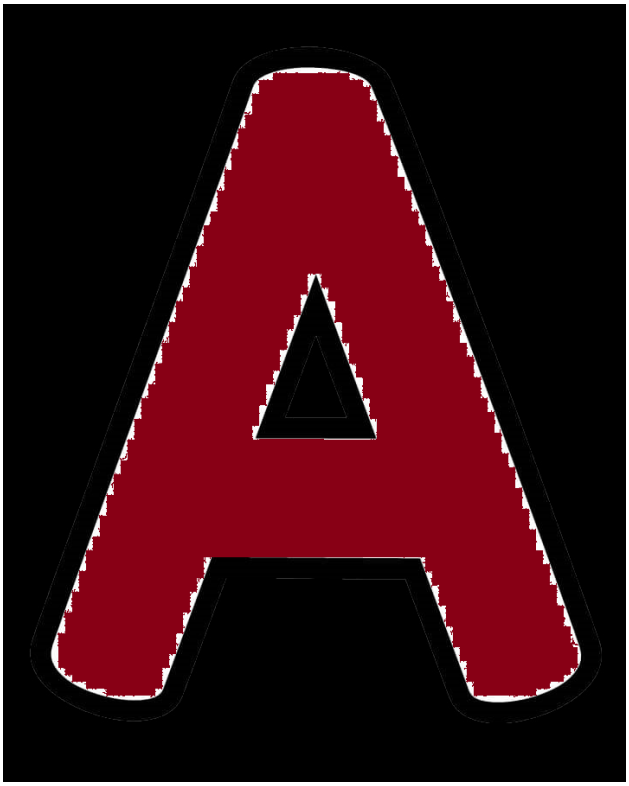

Figure 4: This image depicts a Red A to satisfy

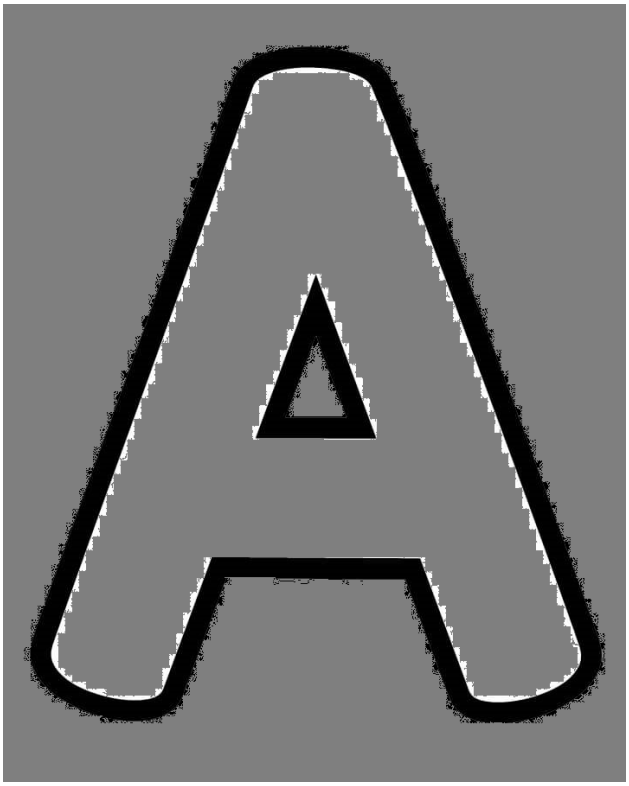

Figure 5: This image depicts a Grayscale A to satisfy

Figure 6: This image has been rotated to a 90° angle. Rotations are performed using paint.net

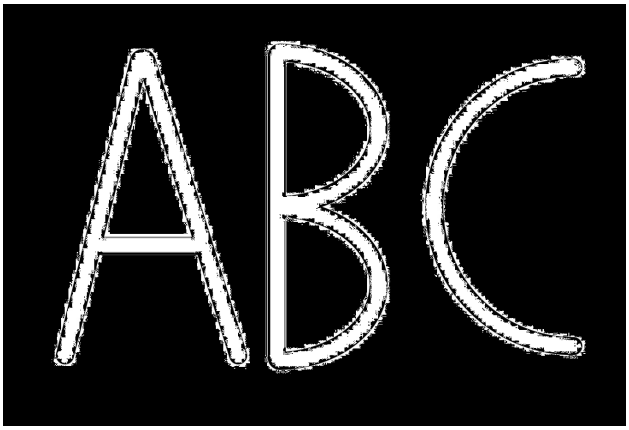

Figure 7: This image depicts the characters A, B and C

Figure 8: This image depicts the Author, Hunter Ceranic, to be used as a degenerate input case

Figure 9: This is a blank image