Optical Alphabet Recognition: System Verification and Validation Plan for OAR

Hunter Ceranic

February 19, 2024

Revision History

Date	Version	Notes
February 18, 2024	1.0	Initial Revision
February 19, 2024	1.1	Added Input Image Orientation Test

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 (Section 3.1) to help build confidence in the requirements, to avoid any missed important details, and to deliver on the outlined objectives (Section 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. Some input may be given by the expert consultants if time permits. 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 Smith (2022a) and MIS Smith (2022b) 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 inconsistencies, unexpected outputs, or bugs that may be encountered. For all the code implemented linters and coverage

tools will be implemented as described in Section 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 running tests created using the pytest library and code coverage tests using the coverage.py library, 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 Test Description

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). Most, if not 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 set of non-corrupt (valid) PNG, JPG, and BMP image files depicting valid label characters (see 1).

   Output: The predicted label should match the actual label 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.

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. **T2**: 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 in black and white(2), grayscale (3), red (4), green (5) and blue (6).

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

Test Case Derivation: These images cover the cases of greyscale, black and white and 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. **T3**: 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 1) of the following sizes:

   - 8192px × 8192px
   - 4096px × 4096px
   - 1024px × 1024px
   - 20px × 20px
   - 10px × 10px

   Output: In the 8192px × 8192px and 10px × 10px cases the program should provide the user with an error. In all other cases, the predicted label should match the actual label as assigned by the oracle (the author).

   Test Case Derivation: These images cover the range 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.

3. **T4**: Test input for images with different orientations

   Control: Automatic

   Initial State: OAR loaded and idle.

   Input: A set of non-corrupt (valid) image files depicting valid label characters (see 7) at the following orientations:

   - 0°
   - ±15°
   - ±30°
   - ±45°
   - ±60°
   - ±75°
   - ±90°
   - ±105°
   - ±120°
   - ±135°
   - ±150°
   - ±165°
   - 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, 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.

4.1.3 Output Labels

To satisfy R4 (and by extension R3) from the SRS (Ceranic, 2024d), any input image should be properly classified and related to the correct output label 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 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 all 26 valid label characters(1), and a blank image depicting no character (8).

   Output: The predicted label output should match the actual label as assigned by the oracle (the author). The output for the blank image should state "No Label Detected".

   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.

4.1.4 Output Label Confidence Level

To satisfy R5 from the SRS (Ceranic, 2024d), any classified input image should also have an associated valid confidence level in the range of 0-100%.

1. **T6**: 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 an input character from the training set, an input character from the test set, an image depicting multiple characters and a blank image depicting no characters.

   Output: For each case the outputs would be as follows:

   - Training Set Image (10): Confidence $> 90\%$

- Test Set Image (10): Confidence $> 50\%$
- Multiple Character Image (9): Confidence $< 50\%$
- Blank Image (8): Confidence $< 10\%$ (Corresponding to the "No Label Detected" Output)

Test Case Derivation: These images likely cover the range of possible confidence level outputs for the program.

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. **T7**: 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 $< \epsilon$ (see 5.1.1)

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

2. **T8**: 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 $< \epsilon$ (see 5.1.2)

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

3. **T9**: 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 calculation should be $< \epsilon$ (see 5.1.3)

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

4.2.2 Maintainability Testing

1. **T10**: 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 5.3).

   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. **T11**: Static code analysis

   Type: Automatic

   Condition: The software code will be tested through the use of Flake 8.

   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.

3. **T12**: Duck Test Code Review

   Type: Manual

   Condition: The software source code and the code review checklist (see 5.4).

   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 Usability Testing

1. **T13**: Code Useability Survey

   Type: Manual

   Condition: A group of intended users, and the survey questions on topics related to the program quality and clarity.

   Result: The completed user surveys (see 5.5).

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

4.2.4 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 executed in a Docker Image, a container that enables the program to be portable, as it contains all the libraries and dependencies needed to build the code.

   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					
T2	X	X		X					
T4	X	X		X			X		
T3	X	X		X					
T5	X	X		X					
T6	X	X		X	X				
T7			X	X	X	X			
T8			X	X	X	X			
T9			X	X	X	X			
T10								X	
T11								X	
T12								X	
T13							X		
T14									X

Table 2: 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.

Chris Crawford. Emnist (extended mnist) - kaggle. https://www.kaggle.com/datasets/crawford/emnist, 2018.

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.

5 Appendix

5.1 Calculation Definitions

5.1.1 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:

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

5.1.2 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:

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

5.1.3 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:

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

5.2 Symbolic Parameters

These are *symbolic* values used in the descriptions and text in this document that are subject to change. They are listed and defined here to make it easy to change and reduce the possibility of mistakes when updating this document.

- $\epsilon = 0.1$ or 10% (used for error calculations)

5.3 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?

5.4 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?

5.5 Usability Survey Questions

- What operating system do you use?
- Is OAR running smoothly on your computer?
- On a scale from 1 to 10, how easy was it to use the software?
- Was the output of the program satisfactory?
- What was the least enjoyable feature to use?
- Was everything legible and clearly displayed? If not, what was not?
- 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?

5.6 Figures

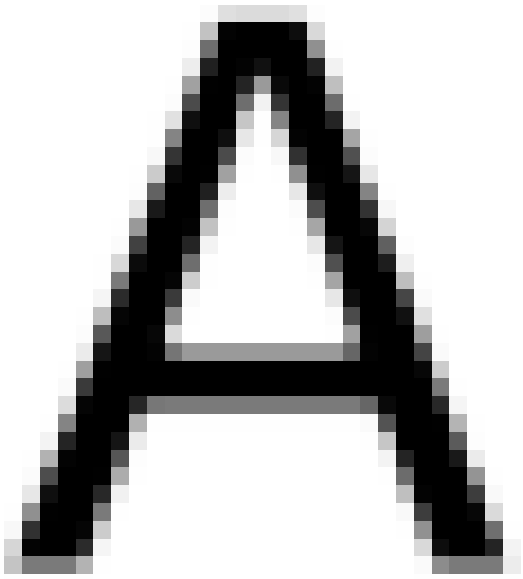

Figure 1: This image has been converted into .bmp, .jpg, and .png formats, and similar images exist for each letter of the English alphabet. This example picture can also be rescaled using paint.net

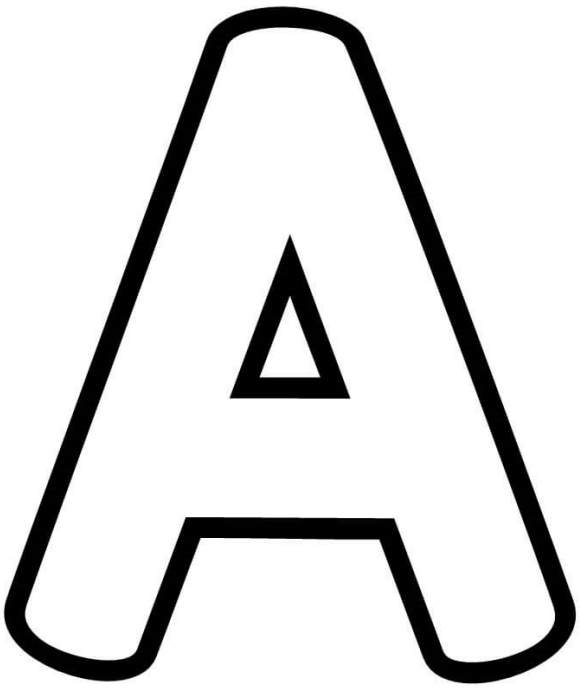

Figure 2: This image depicts a Black and White A to satisfy

Figure 3: This image depicts a Grayscale A to satisfy

Figure 4: This image depicts a Red A to satisfy

Figure 5: This image depicts a Green A

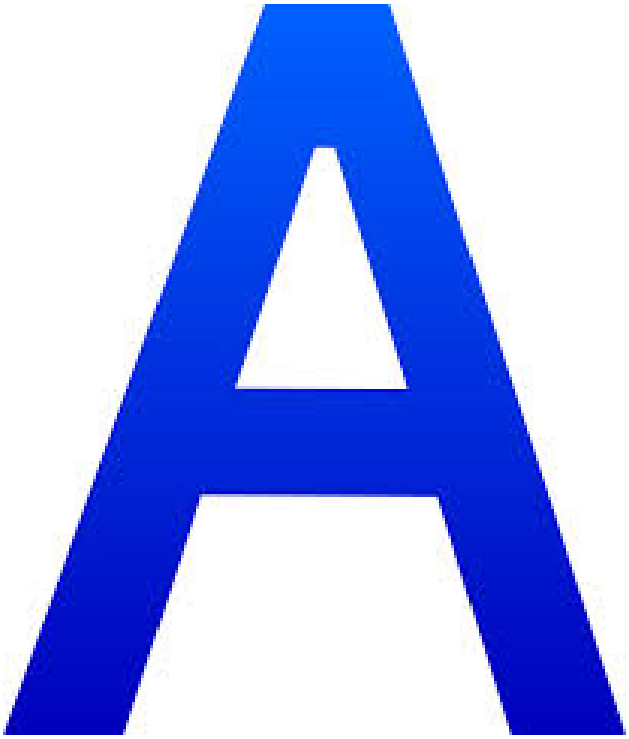

Figure 6: This image depicts a Blue A

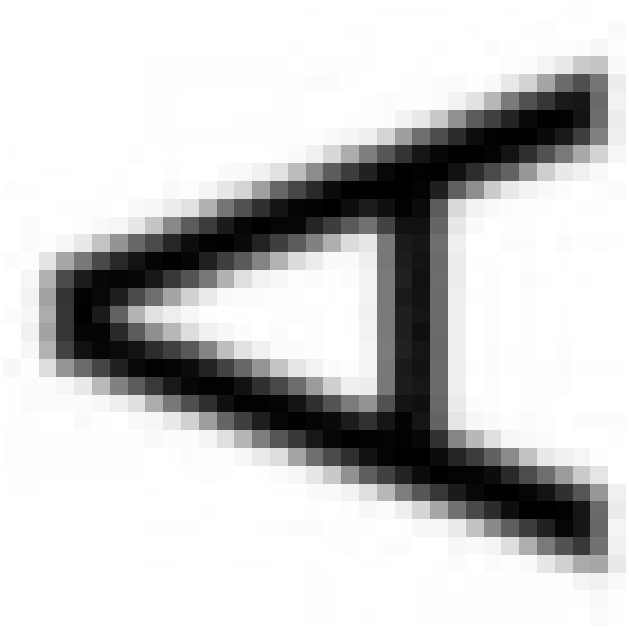

Figure 7: This image has been rotated to a 90° angle. Rotations are performed using paint.net

Figure 8: This is a blank image

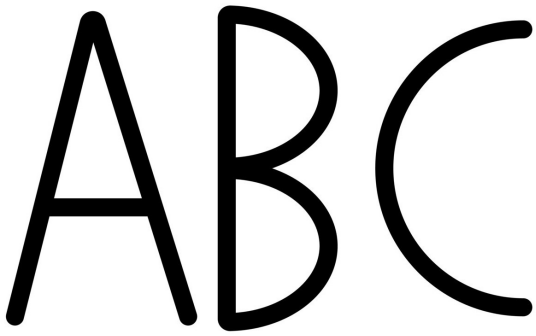

Figure 9: This image depicts the characters A, B and C

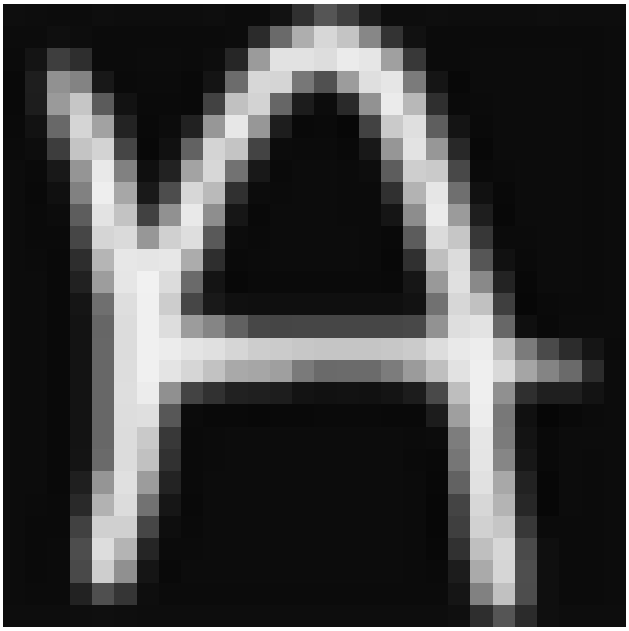

Figure 10: This image depicts a sample character from the EMNIST data set used for training and testing the OAR model (Crawford, 2018)