System Verification and Validation Plan for OCRacle

Phillip Tran

April 17, 2025

Revision History

Date	Version	Notes
February 20, 2025	1.0	Initial Creation
April 1, 2025	2.0	Added unit tests
April 5, 2025	2.1	Addressed feedback from Dr. Spencer Smith
April 7, 2025	2.2	Addressed Feedback from Hussein Saad

Contents

List of Tables

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 (Tran, 2025e).

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 model's predicted character.

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. Validation of the software's maintainability will also be conducted.

For the purposes of this project, the validation of the program's usability will not be heavily 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 video code walkthrough. This will explain how the code works to achieve the desired functionality and give technical users an understanding of how to train, evaluate, and test the model.

2.4 Relevant Documentation

- SRS Document (Tran, 2025e): Outlines the requirements for the OCRacle program. This VnV plan will be based on the requirements outlined in this document.
- MG Document (Tran, 2025a): Outlines the modules that compose the OCRacle program. The VnV plan will be based on the modules outlined in this document.
- MIS Document (Tran, 2025b): 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 instructor. 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 instructor 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 instructor using the MG checklist (Smith, 2024a) and MIS checklist (Smith, 2022b).

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

3.4 Verification and Validation Plan Verification Plan

The VnV plan will be reviewed by the domain expert and instructor using the VnV plan checklist (Smith, 2022c). 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. This will be the primary responsibility of the author, with the domain expert and instructor providing feedback of the code walkthroughs to ensure that the code is correct. In order to facilitate this, the author will be responsible for creating a video that explains the purpose of each module, briefly discusses the code, and details how each component contributes to the overall functionality of the program. This is also treated as the extra task for the project, as outlined in the Problem Statement (Tran, 2025c) and Reflection and Traceability Report (Tran, 2025d).

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 run using the pytest framework. 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 pyright 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. The main focus of the project is not user acceptance, but rather the accuracy of the OCR functionality.

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 ready for input.

   Input: A single uppercase Latin alphabet character in JPEG format. This letter should come from the "test" subset of the EMNIST letters dataset. [1]

   Output: The system accepts the image as an input without errors.

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

   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 ready for input.

   Input: A single uppercase Latin alphabet character in PNG format. This letter should come from the "test" subset of the EMNIST letter dataset.

[1] To learn more about this dataset, refer to Section 6.1. Anywhere a single sample is required for the test, the author has arbitrarily chosen the letter "Y" from the EMNIST letters dataset.

Output: The system accepts the mage as an input without errors.

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

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 ready for input.

   Input: A file in any format other than JPEG or PNG.

   Output: The system rejects the file, displaying an appropriate error message.

   Test Case Derivation: Based on R1, the program must reject files that are in formats other than JPEG or PNG.

   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 ready for input.

   Input: A valid image containing a single uppercase Latin alphabet character. This image should come from the "test" subset of the EMNIST letter dataset.

   Output: As described in IM1, the system pre-processed the image for the classification task in the specified manner.

   Test Case Derivation: Based on R2, the program must pre-process images for classification. The test will verify that the image is pre-processed according to R2 in the SRS document (Tran, 2025e).

   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, which covers R3.

Single Character Prediction

1. T5: Character Prediction

   Control: Automatic

   Initial State: The OCRacle system is ready for input.

   Input: An image containing a single uppercase Latin alphabet character. This image should come from the "test" subset of the EMNIST letter dataset.

   Output: The predicted character, which should match the character in the image. The predicted character corresponds to the character with the highest probability in the probability distribution.

   Test Case Derivation: Based on R3, the program should correctly predict single characters from prepared images. The prediction produced by the model will be compared to the image's true label.

   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 distribution should contain the probability of each character in the alphabet, and the sum of the probabilities should be 1, since the input is assumed to always be a valid character as specified in the SRS under Assumption 2 (Tran, 2025e). These tests cover R4.

Probability Vector Validity

1. T6: Probability Vector Sum

   Control: Automatic

Initial State: The OCRacle system is ready for input.

Input: An image containing a single uppercase Latin alphabet character. This image should come from the "test" subset of the EMNIST letter dataset.

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

Test Case Derivation: Based on R4, the program must output a probability distribution 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 system is ready for input.

   Input: An image containing a single uppercase Latin alphabet character. This image should come from the "test" subset of the EMNIST letter dataset.

   Output: A probability distribution 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 distribution where the length is the number of possible classification labels.

   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. This covers R5.

Readable Character Output

1. T8: Readable Character Output

   Control: Automatic

   Initial State: The OCRacle system is ready for input.

Input: An image containing a single uppercase Latin alphabet character. This image should come from the "test" subset of the EMNIST letter dataset.

Output: The predicted character is displayed in a human-readable format. A human-readable format is defined as displaying P_{pred}, which is the character with the highest probability in the probability distribution P.

Test Case Derivation: Based on R5, the program must display character predictions in a readable format. To validate this, this test will ensure that the final output function correctly returns a character belonging in $\{A...Z\}$ as the predicted character.

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 ready for input. 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: All image samples from the "train" subset of the EMNIST letters dataset.

Output/Result: The overall accuracy percentage of the predictions made by the OCRacle system. This overall accuracy report is compared to OAR's overall accuracy percentage as reported the OAR VnV Report to determine if OCRacle performed better.

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 accuracy will be calculated using the formula dictated in Section 6.2. The overall accuracy percentage from this test is compared against the OAR predecessor's overall accuracy percentage.

4.2.2 Usability Testing

User Manual Usability Test

1. T10: User Manual Usability Test

   Type: Dynamic, Manual

   Initial State: The codebase and user manual are available.

   Input/Condition: All members of the VnV team, which are assumed to have basic command line skills. At a minimum, the members of the team have the equivalent knowledge contained in the MIT Missing Semester.

   Output/Result: Feedback on ease of use, clarity of instructions, and any difficulties encountered by the users.

   How test will be performed: To support NFR2, users will follow the user manual to set up and run the OCRacle program. The major tasks that a user should be able to complete are: 1. setting up a Python virtual environment, 2. install dependencies, 3. running the test suite, 4. training the model, and 5. using OCRacle to identify a single uppercase Latin alphabet character in an image. 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 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: Pyright Static Type Checker Test

   Type: Static, Automatic

   Initial State: The pyright static type checker is set up and ready to run.

   Input/Condition: The codebase of the OCRacle project.

   Output/Result: A report on type errors, 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 static type checker will check the codebase for any issues.

3. T13: Code Review for Modularity

   Type: Static, Manual

   Initial State: The codebase is available.

   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 utilize the Source Code Checklist (Smith, 2022a)

to identify sections of the code that are not easily modifiable or understandable. Any issues encountered by the reviewers will be documented via GitHub issues.

4.2.4 Portability Testing

Cross-Platform Compatibility Test

1. T14: Cross-Platform Compatibility Test

   Type: Dynamic, Automatic

   Initial State: The OCRacle system and its dependencies are installed in the GHA environment.

   Input/Condition: This system is running in the GHA environment.

   Output/Result: The GHA environment is able to run all automatic tests described in the VnV Plan without any issues.

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

4.3 Traceability Between Test Cases and Requirements

The following table outlines the traceability between the test cases and the requirements outlined in the SRS document. Note: The table has been generated using ChatGPT 4o. The output has been manually validated to ensure that it is correct. [2]

5 Unit Test Description

The unit tests will be based on the modules and interfaces outlined in the MG (Tran, 2025a) and MIS (Tran, 2025b) documents. The unit tests will be created using the pytest framework. These tests will be run in a GHA

[2]The following query was used: "Fill out the following traceability matrix given the following table template and the information from this document: [table template and information from this document]."

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

environment on every pull request, on a Windows, Linux, and MacOS environment.

Unit tests will focus on validating the functionality of modules that have not already been validated by the system tests. Since these modules have a static input with a known output, there will be a focus on validating normal behavior without the need for edge cases.

5.1 Unit Testing Scope

The following modules already have their functionality validated by the System Tests. As such, it is not necessary to create Unit Tests for these modules:

- Input Format Module (M3)
- Image Preprocessing Module (M5)
- Prediction Model Module (M6)
- Model Output Module (M7)

- Accuracy Metrics Module (M9)

The following modules are implemented by a third party and are not considered to be part of the OCRacle program. As such, it is not necessary to create Unit Tests for these modules:

- Hardware Hiding Modules (M1)
- Graphical User Interface Module (M8)

The unit tests will be created to validate the behavior of the following modules:

- Model Training Module (M4)
- Data Loading Module (M10)

5.2 Tests for Functional Requirements

These unit tests validate the behavior of supporting modules to ensure that the OCRacle program meets the functional requirements outlined in the SRS document (Tran, 2025e).

5.2.1 Model Training Module (M4)

These tests ensure that the module is able to produce a model that conforms to R3 as outlined in the SRS document (Tran, 2025e).

Because the Model Training Module (M4) produces the Prediction Model Module (M6), as long as the tests for the Prediction Model Module (M6) pass, then this module's behavior is considered validated.

1. T15: Model Training

   Type: Dynamic, Automatic

   Initial State: THe OCRacle system is ready for input.

   Input: The "train" subset of the EMNIST letters dataset is available.

   Output: The modules produces the Prediction Model Module (M6), which is a model that has been trained on the "train" subset of the EMNIST letters dataset.

Test Case Derivation: Based on R3, the program must be able to produce a model that is capable of taking in the specified input and producing the specified output in IM3 as described in the SRS document (Tran, 2025e).

How test will be performed: The test will be run on GHA. The system will train the model on the "train" subset of the EMNIST letters dataset to produce the Prediction Model Module (M6). Then, the rest of the system tests will run. This validates the behavior of the Model Training Module (M4) to produce a model in accordance with the requirements outlined in the SRS document (Tran, 2025e).

5.2.2 Data Loading Module (M10)

These tests ensure that the module is able to load the "train" subset of the EMNIST letters dataset to support the training of the model. This fulfills R3 as outlined in the SRS document (Tran, 2025e).

1. T16: Load Train Subset

   Type: Dynamic, Automatic

   Initial State: The OCRacle system is ready for input.

   Input/Condition: The "train" subset of the EMNIST letters dataset is available.

   Output/Result: The module is able to load the "train" subset of the EMNIST letters, which is a tuple containing 20,800 images and their corresponding labels.

   Test Case Derivation: Based on R3, the program must have the "train" subset of the EMNIST letters dataset loaded in order to train the model.

   How test will be performed: The test will be run on GHA. The system will load the "train" subset of the EMNIST letters dataset and check that all of the data has been loaded correctly, such that the format of the images corresponds to DD2 as specified in the SRS document (Tran, 2025e). The system will check tha the size of the dataset is 20,800 image/label pairs and that the labels are correctly formatted as an integer from 0-25 inclusive corresponding to the letters A-Z.

5.3 Input Format Module (M3)

These tests ensure that the input module adequately handles the DimensionsError and FileNotFound exceptions identified in the MIS document (Tran, 2025b).

1. T17: File Not Found

   Type: Dynamic, Automatic

   Initial State: The OCRacle system is ready for input.

   Input/Condition: A file path to a file that does not exist.

   Output/Result: The system raises a FileNotFoundError exception.

   Test Case Derivation: This test case is derived from the MIS document (Tran, 2025b), which states that the system should raise a FileNotFoundError exception if the file does not exist.

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

2. T18: Invalid Dimensions

   Type: Dynamic, Automatic

   Initial State: The OCRacle system is ready for input.

   Input/Condition: A file with dimensions outside of the boundaries identified by the MIS document (Tran, 2025b).

   Output/Result: The system raises a DimensionsError exception.

   Test Case Derivation: This test case is derived from the MIS document (Tran, 2025b), which states that the system should raise a DimensionsError exception if the file does not have the correct dimensions.

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

5.4 Tests for Nonfunctional Requirements

These unit tests validate the behavior of supporting modules to ensure that the OCRacle program meets the nonfunctional requirements outlined in the SRS document (Tran, 2025e).

5.4.1 Data Loading Module (M10)

These tests ensure that the module is able to load the "test" subset of the EMNIST letters dataset to support the testing of the model's accuracy. This Fulfills NFR1 as outlined in the SRS document (Tran, 2025e).

1. T17: Load Test Subset

   Type: Dynamic, Automatic

   Initial State: The OCRacle system is ready for input.

   Input/Condition: The "test" subset of the EMNIST letters dataset is available.

   Output/Result: The module is able to load the "test" subset of the EMNIST letters, which is a tuple containing 14,800 images and their corresponding labels.

   How test will be performed: The test will be run on GHA. The system will load the "test" subset of the EMNIST letters dataset and check that the data has been loaded correctly, such that the format of the images corresponds to DD2 as specified in the SRS document (Tran, 2025e). The system will check that the size of the dataset is 14,800 image/label pairs and that the labels are correctly formatted as an integer from 0-25 inclusive corresponding to the letters A-Z.

5.5 Traceability Between Test Cases and Modules

The following table outlines the traceability between test cases and modules outlined in the MG (Tran, 2025a). Note: This table has been generated using ChatGPT 4o. The output has been manually validated to ensure that it is correct. [3]

[3]The following query was used: "Here is an example of a table in LaTeX: [table template]. Please create a new table for tracking the tracability of test cases and modules in the same table format. Here is the information you'll need to complete this: [list of modules and the tests that validate each one]."

Test Case	M1	M2	M3	M4	M5	M6	M7	M9	M10
T1			X						
T2			X						
T3			X						
T4					X				
T5			X				X		
T6						X			
T7						X			
T8							X		
T9								X	
T10	X	X	X	X	X	X	X	X	X
T11	X	X	X	X	X	X	X	X	X
T12	X	X	X	X	X	X	X	X	X
T13	X	X	X	X	X	X	X	X	X
T14	X	X	X	X	X	X	X	X	X
T15				X					
T16									X
T17									X
T18			X						
T19			X						

Table 2: Traceability Matrix of Test Cases to Modules

References

Hunter Ceranic. Optical alphabet recognition: System verification and validation plan. 2024.

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

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

Spencer Smith. System verification and validation plan check-

list. https://github.com/ptrandev/OCRacle/blob/main/docs/Checklists/VnV-Checklist.pdf, 2022c.

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. Module guide. https://github.com/ptrandev/OCRacle/blob/main/docs/Design/SoftArchitecture/MG.pdf, 2025a.

Phillip Tran. Module interface specification. https://github.com/ptrandev/OCRacle/blob/main/docs/Design/SoftDetailedDes/MIS.pdf, 2025b.

Phillip Tran. Problem statement and goals. https://github.com/ptrandev/OCRacle/blob/main/docs/ProblemStatementAndGoals/ProblemStatement.pdf, 2025c.

Phillip Tran. Reflection and traceability report on ocracle. https://github.com/ptrandev/OCRacle/blob/main/docs/ReflectAndTrace/ReflectAndTrace.pdf, 2025d.

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

6 Appendix

6.1 EMNIST Letter Dataset

The EMNIST Letters dataset contains handwritten uppercase Latin alphabet characters. This dataset is split into a "train" subset consisting of 20,800 image/label pairs and a "test" subset consisting of 14,800 image/label pairs. For the purpose of validating the OCRacle program, images from the "test" subset will be used for all system tests.

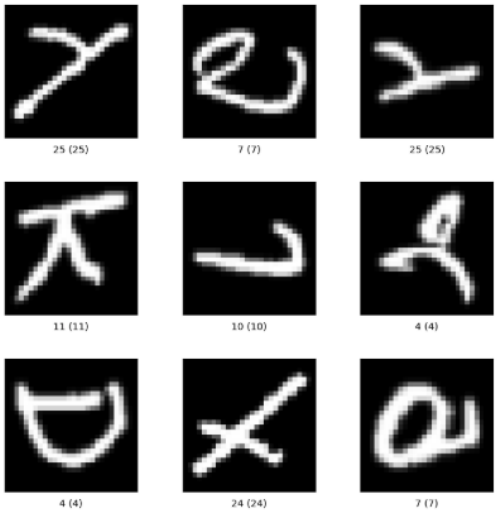

Figure 1: A sample of the EMNIST Letter dataset.

6.2 Accuracy Calculation

To calculate the accuracy of the model, the categorical accuracy of the model will be used. To perform the calculation, the categorical_accuracy function from the Keras library will be used. This is the mathematical equivalent to the accuracy calculation described in the OAR project's VnV Plan (Ceranic, 2024).