System Test Plan

For

EasyNN

Version/Author	Date
Daniel Wilczak	10/18
Jack Nguyen	10/18
Liam Kehoe	10/18
Nathan Foster	10/18

Table of Contents

1. Introduction

1.1 Purpose

This document is a test plan for the EasyNN System Testing, produced by the System Testing team. It describes the testing strategy and approach to testing the team will use to verify that the application meets the established requirements of the business before release.

1.2 Objectives

- Meets the requirements, specifications, and Business rules.
- Supports the intended business functions and achieves the required standards.
- Satisfies the Entrance Criteria for User Acceptance Testing.

2. Functional Scope

The Modules in the scope of testing for the EasyNN System Testing are mentioned in the document attached in the following path: Overall strategy, execution plan, traceability matrix

3. Overall Strategy and Approach

3.1 Testing Strategy

EasyNN System Testing will include testing of all functionalities that are in scope (Refer to Functional Scope Section) identified. System testing activities will include the testing of new functionalities, modified functionalities, screen level validations, workflows, functionality access, testing of internal & external interfaces.

3.2 System Testing Entrance Criteria

The system should be in a place where it can physically compile and execute, and perform the user's desired outcomes. The EasyNN backend, as well as the GUI for the user to be able to use the system and operate with one of the models.

3.3 Testing Types

3.3.1 Usability Testing

User interface attributes, cosmetic presentation, and the content will be tested for accuracy and general usability. The goal of Usability Testing is to ensure that the User Interface is comfortable to use and provides the user with consistent and appropriate access and navigation through the functions of the application (e.g., access keys, consistent tab order, readable fonts, etc.)

The user interface will have specific functionalities (specifics undefined), and everything will operate as intended.

The model will show it's output once the user requests it.

3.3.2 Functional Testing

The objective of this test is to ensure that each element of the component meets the functional

requirements of the business as outlined in the:

- Business / Functional Requirements
- Business rules or conditions
- Other functional documents produced during the project i.e. resolution to issues/change requests/feedback

3.4 Suspension Criteria and Resumption Requirements

This section will specify the criteria that will be used to suspend all or a portion of the testing activities on the items associated with this test plan.

3.4.1 Suspension Criteria

Testing will be suspended if running the model causes a fatal system error and the code will not compile. All testing should be halted and repairing the cause of the system error will become the top priority.

3.4.2 Resumption Requirements

Resumption of testing will be possible when the functionality that caused the suspension of testing has been retested successfully.

4. Execution Plan

4.1 Execution Plan

The execution plan will detail the test cases to be executed. The Execution plan will be put together to ensure that all the requirements are covered. The execution plan will be designed to accommodate some changes if necessary if testing is incomplete on any day. All the test cases of the projects under test in this release are arranged in a logical order depending upon their interdependency.

1. EasyNN will be downloaded via pip
2. Feed a dataset through an untrained model, and receive an accuracy
3. Save a newly trained model
4. Check if the neural network casts inputs to float tensors
5. Check software for operations used and if they result in tensors
6. Check software for operations used and if they result in vectors
7. Check software for operations used and if they result in 3D tensors
8. Check software for optimizer and machine learning usage
9. Check neural network software and documentation on how to define its topology

5. Traceability Matrix & Defect Tracking

5.1 Traceability Matrix

List of requirements, corresponding test cases

Requirement Number (from SRS)	Covered by Test:
3.1.1	2
3.1.2	2
3.1.3	3
3.1.4	2
3.1.5	2
3.1.6	2
3.2.1	4
3.2.2	5
3.2.3	6
3.2.4	7
3.2.5	2
3.2.6	8
3.2.7	9
3.6.1	3
3.6.2	3
3.6.3	3
3.6.4	3
3.6.5	2
3.6.6	2

5.2 Defect Severity Definitions

Critical	The defect causes a catastrophic or severe error that results in major problems and the functionality rendered is unavailable to the user. A manual procedure cannot be either implemented or a high effort is required to remedy the defect. Examples of a critical defect are as follows: ● System abends ● Data cannot flow through a business function/lifecycle ● Data is corrupted or cannot post to the database
Medium	The defect that does not seriously impair system function can be categorized as a medium Defect. A manual procedure requiring medium effort can be implemented to remedy the defect. Examples of a medium defect are as follows: ● Form navigation is incorrect ● Field labels are not consistent with global terminology

Low	The defect is cosmetic or has little to no impact on system functionality. A manual procedure requiring low effort can be implemented to remedy the defect. Examples of a low defect are as follows: ● Repositioning of fields on screens ● Text font on reports is incorrect

6. Environment

6.1 Environment

- Local machine
- Text editor
- Mnist dataset
- Cifar10 dataset

7. Assumptions

- The user will have a computer with an internet connection.
- The user will have pip installed on their computer.
- The user will have Python version 3.9.5 or later installed on their computer.