

**A**

**Project Report**

**On**

**Handwritten Nepali Digit Recognition**

**Submitted By:**

Aashish Pokharel (20786/075)

Milan Chaudhary (20811/075)

Shreedhar Ghimire (20827/075)

**Submitted To**

Shikha Sharma

Lecturer

Orchid International College

DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

ORCHID INTERNATIONAL COLLEGE

# **ACKNOWLEDGEMENT**

It is a tremendous honor to be able to express our deepest appreciation to everyone who assisted us. It is our great privilege and honor to be able to work on this project and gain expert knowledge and assistance from the faculty at Orchid International College.

We specially extend our gratitude towards our supervisors **Er. Dhiraj Kumar Jha** and **Ms. Shikha Sharma** who provided us with sufficient guidance and reminders to inspire us to stay focused and reflect deeply on the purpose of this study. Their recommendations and inputs have always improved this project, and finally for producing a report of excellent quality and complete this project. Their continuous inspiration has made us achieve the target.

We would also like to extend our heartfelt gratitude for the faculty of **Orchid International College, Department of Computer Science and Information Technology** for this opportunity. We would like to express our gratitude to our friends and colleagues too for their selfless efforts in helping us completing the project.

# **ABSTRACT**

Handwritten Nepali Digit Classification is an approach of enabling computers to recognize human written nepali digits through a complicated process of learning called Machine Learning. The system focuses on predicting the handwritten digits in the form of vector images and then classifying it using ANN and then predicting the digit with higher accuracy. The system is a MVC based web application developed using HTML, CSS, JavaScript, ReactJS and Python as the backend framework incorporating Artificial Neural Networks as the primary algorithm implemented.

**Keywords: Handwritten Nepali Digit Classification, Machine Learning, ANN, HTML/CSS, ReactJS, Python**

**TABLE OF CONTENTS**

**[ACKNOWLEDGEMENT](#_Toc132607500)** [i](#_Toc132607500)

**[ABSTRACT](#_Toc132607501)** [ii](#_Toc132607501)

**[LIST OF FIGURES](#_Toc132607502)** [v](#_Toc132607502)

**[LIST OF TABLES](#_Toc132607503)** [vi](#_Toc132607503)

**[LIST OF ABBREVIATIONS](#_Toc132607504)** [vii](#_Toc132607504)

**[CHAPTER 1: INTRODUCTION](#_Toc132607505)** [1](#_Toc132607505)

**[1.1](#_Toc132607506)****[Introduction](#_Toc132607506)** [1](#_Toc132607506)

**[CHAPTER 2: LITERATURE REVIEW](#_Toc132607507)** [4](#_Toc132607507)

**[CHAPTER 3: SYSTEM ANALYSIS](#_Toc132607508)** [7](#_Toc132607508)

**[3.1 System Analysis](#_Toc132607509)** [7](#_Toc132607509)

**[3.1.1 Requirement Analysis](#_Toc132607510)** [7](#_Toc132607510)

**[3.1.2 Functional Requirements](#_Toc132607511)** [7](#_Toc132607511)

**[3.1.3 Non-functional Requirements](#_Toc132607512)** [15](#_Toc132607512)

**[3.1.4 Feasibility Analysis](#_Toc132607513)** [15](#_Toc132607513)

**[3.1.5 Analysis](#_Toc132607514)** [18](#_Toc132607514)

**[3.1.6 Dataset Analysis](#_Toc132607515)** [21](#_Toc132607515)

**[CHAPTER 4: SYSTEM DESIGN](#_Toc132607516)** [23](#_Toc132607516)

**[4.1 Design](#_Toc132607517)** [23](#_Toc132607517)

**[4.1.1 Model Architecture](#_Toc132607518)** [23](#_Toc132607518)

**[4.1.2 Model Building Pipeline](#_Toc132607519)** [25](#_Toc132607519)

**[4.1.3 UI Design](#_Toc132607520)** [26](#_Toc132607520)

**[CHAPTER 5: IMPLEMENTATION](#_Toc132607521)** [27](#_Toc132607521)

**[5.1 Tools Used](#_Toc132607522)** [27](#_Toc132607522)

**[5.1.1 Development Tools](#_Toc132607523)** [27](#_Toc132607523)

**[5.1.2 Design and Documentation Tools](#_Toc132607524)** [28](#_Toc132607524)

**[5.2 Methodology](#_Toc132607525)** [29](#_Toc132607525)

**[5.2.1 Data Preparation](#_Toc132607526)** [29](#_Toc132607526)

**[5.2.2. Model Introduction](#_Toc132607527)** [29](#_Toc132607527)

**[5.2.3 Baseline Modelling](#_Toc132607528)** [32](#_Toc132607528)

**[5.2.4 Model Optimization](#_Toc132607529)** [32](#_Toc132607529)

**[5.2.5 Final Model](#_Toc132607530)** [33](#_Toc132607530)

**[5.2.6 Model Deployment](#_Toc132607531)** [35](#_Toc132607531)

**[CHAPTER 6: TESTING](#_Toc132607532)** [36](#_Toc132607532)

**[6.1 Unit Testing](#_Toc132607533)** [36](#_Toc132607533)

**[CHAPTER 7: CONCLUSION AND FUTURE RECOMMENDATIONS](#_Toc132607534)** [38](#_Toc132607534)

**[7.1 Conclusion](#_Toc132607535)** [38](#_Toc132607535)

**[7.2 Future Recommendations](#_Toc132607536)** [38](#_Toc132607536)

**[REFERENCES](#_Toc132607537)** [39](#_Toc132607537)

# **LIST OF FIGURES**

*[Figure 1.1: CPMAI architecture](#_Toc132579967)* [3](#_Toc132579967)

*[Figure 3.1: Use-Case Diagram for Handwritten Nepali Digit Recognition 8](#_Toc132579968)*

*[Figure 3.2: Work Breakdown Structure of Handwritten Nepali Digit Classification System 16](#_Toc132579969)*

*[Figure 3.3: Gantt Chart of the Project 17](#_Toc132579970)*

*[Figure 3.4: Activity Diagram for Handwritten Nepali Digit Classification 19](#_Toc132579971)*

*[Figure 3.5: Sequence Diagram for Handwritten Nepali Digit Classification 20](#_Toc132579972)*

*[Figure 3.6: Folder Structure for dataset 22](#_Toc132579973)*

*[Figure 4.1: Multilayer Perceptron 23](#_Toc132579974)*

*[Figure 4.2: Model Building Pipeline 25](#_Toc132579975)*

*[Figure 5.1: ReLU Activation Function Graph 31](#_Toc132579976)*

*[Figure 5.2: Training accuracy of the final model 34](#_Toc132579977)*

*[Figure 5.3: Training loss of the final model 34](#_Toc132579978)*

# **LIST OF TABLES**

*[Table 3.1: Use Case for Load training datasets](#_Toc132579467)* [9](#_Toc132579467)

*[Table 3.2: Use Case for Preprocess data](#_Toc132579468)* [10](#_Toc132579468)

*[Table 3.3: Use Case for Input Training Dataset](#_Toc132579469)* [11](#_Toc132579469)

*[Table 3.4: Use Case for Train Model](#_Toc132579470)* [12](#_Toc132579470)

*[Table 3.5: Use Case for Load Model](#_Toc132579471)* [13](#_Toc132579471)

*[Table 3.6: Use Case for Save Model](#_Toc132579472)* [14](#_Toc132579472)

*[Table 3.7: Use Case for Predict Class](#_Toc132579473)* [14](#_Toc132579473)

*[Table 5.1: Baseline Modelling](#_Toc132579474)* [32](#_Toc132579474)

*[Table 5.2: Final Model](#_Toc132579475)* [33](#_Toc132579475)

*[Table 6.1: Test scenarios for the image digit classification](#_Toc132579476)* [36](#_Toc132579476)

*[Table 6.2: Test scenarios for Neural Network Initialization](#_Toc132579477)* [37](#_Toc132579477)

# **LIST OF ABBREVIATIONS**

AI : Artificial Intelligence

ANN : Artificial Neural Networks

API : Application Programming Interface

CNN : Convolutional Neural Networks

CPMAI : Cognitive Project Management for Artificial Intelligence

CSS : Cascading Style Sheet

HTML : Hypertext Markup Language

JS : JavaScript

ML : Machine Learning

MLP : Multilayer Perceptron

OCR : Optical Character Recognition

ReLU : Rectified Linear Unit

SDLC : Software Development Life Cycle

SVM : Support Vector Machine

UC : Use Case

UI : User Interface  
UML : Unified Modeling Language

WBS : Work Breakdown Structure

# **CHAPTER 1: INTRODUCTION**

## **Introduction**

Handwritten Nepali Digit Classification is an approach of enabling computers to recognize human written nepali digits through a complicated process of learning called Machine Learning using the concept of ANN.

The project intends to implement Artificial Neural Network model, that will be able to recognize the handwritten Nepali digits with high accuracy.

* 1. **Problem Statement**

In a fast-changing digital world, we need to adapt to the new challenges thrown to us in a daily basis. The tasks of humans are being replaced by robots in every field whether it be finance, medicine, education and much more. So, it makes sense for a system that will be able to recognize the tasks performed by humans as they are the ones being replaced by the system being designed, recognizing the characters written by humans being one of them.

It is almost impossible to design a system to recognize handwritten digits with 100% accuracy, but still there are studies being conducted on a regular basis to improve on the accuracy we have. It is particularly hard for the Nepali Characters and Devanagari script as a whole.

It makes sense for the project to be chosen such that it is simple yet worthy enough for a final year project. So, the authors intend to build a model to recognize the Nepali digits only so that the model is specific to a task but will be good enough to generalize to other characters if necessary.

* 1. **Objectives**

Following are the objectives for the project:

* To identify Nepali digits using Deep Learning Approaches,
* To perform tasks such as hyperparameter tuning, normalization, regularization, etc. to create an efficient model,
* To integrate the developed model into a given system where user can provide the input and the model gives the predicted outcome.
  1. **Scope and Limitation**

The project tries to implement a system that will be able to take inputs from user and recognize which nepali digit is given as input.

The System has following limitations:

● The system will not be able to recognize Nepali characters other than the digits

* 1. **Development Methodology**

CPMAI (Cognitive Project Management for AI) is the best-practice methodology for AI projects proposed by Cognilytica.  
This methodology consists of 6 phases.

Phase 1: Business Understanding

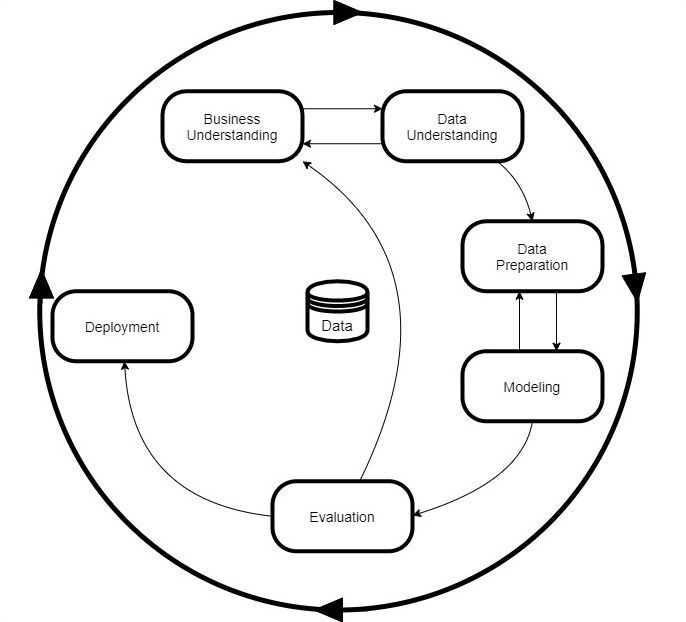
Phase 2: Data Understanding

Phase 3: Data Preparation

Phase 4: Modeling

Phase 5: Model Evaluation

Phase 6: Model Deployment



*Figure 0:1.1: CPMAI architecture*

**Report Organization**

This is the process by which the project report has been organized. The project follows the SDLC process which consists of:

* Requirement Analysis
* Planning
* Analysis
* Design
* Implementation
* Testing
* Deployment and Maintenance

# **CHAPTER 2: LITERATURE REVIEW**

Handwritten digit classification using Neural Networks has been a topic of research for a long time. It has seen a lot of progress in the recent years due to availability of resources for the computation.

In [1] the author discusses the various algorithms that could be used for Character Recognition such as logistic regression, Support Vector Machines, etc. although these may provide analytical and computational properties but that their practical applicability is limited by the curse of dimensionality. In order to apply such models to large scale problems, it is necessary to adapt the basic functions to the data. The approach is to fix the number of basic functions in advance but allow them to be adaptive, in other words to use parametric forms for the basic functions in which the parameter values are adapted during training. The most successful model of this type in the context of pattern recognition is the feed-forward neural network, also known as the multilayer perceptron.

David Rumelhart, Geoffrey Hinton, and Ronald Williams in [2] suggested that neural networks with backpropagation could be used for character recognition. This paper introduced the backpropagation algorithm, which is a method for training neural networks to minimize error on a supervised learning task, such as character recognition. Prior to this paper, neural networks had been used for character recognition, but the backpropagation algorithm made it possible to train large and deep networks more effectively, which led to significant improvements in performance.

The implementation of Neural Networks in Character Recognition can be found as early as 1998 by LeCun et al. [3] with an error rate as low as 12% using a single perceptron model. Further use of deeper Neural Networks has been done and results with error less than 1%.

Handwritten Nepali Digit Classification has been a topic of research for the recognition of the Nepali digits. This has been a difficult task because of the complexity and variations in the handwritten Nepali Devanagari digits.

In [4] by Yadav, Cuadrado and Morato, in 2013 used ANN’s for Devanagari OCR and achieved an accuracy of 90% in character recognition. However, the given accuracy is for only 5 fonts. In this paper, they propose an OCR for printed Hindi text in Devanagari script, using Artificial Neural Network (ANN), which improves its efficiency. One of the major reasons for the poor recognition rate is error in character segmentation. In this work, three feature extraction techniques-: histogram of projection based on mean distance, histogram of projection based on pixel value, and vertical zero crossing, have been used to improve the rate of recognition. These feature extraction techniques are powerful enough to extract features of even distorted characters/symbols. For development of the neural classifier, a back-propagation neural network with two hidden layers is used. The classifier is trained and tested for printed Hindi texts. A performance of approximately 90% correct recognition rate is achieved. [4]

In another study by Nirajan Pant and Balkrishna Bal, in [1], proposed a hybrid OCR system for printed Nepali text using the Random Forest (RF) Machine Learning technique. It incorporates two different approaches of OCR – the Holistic and the Character level recognition. The system first tries to recognize a word as a whole; if it is not confident about the word being recognized, then the character level recognition is performed. The recognition rates of approximately 78.87% and 94.80% were achieved for character level recognition method and the Hybrid method respectively. They attempted to minimize the segmentation errors by reducing the segmentation tasks. [1]

Similarly, in [5], Sharma and Bhattarai in 2017 has shown a high character recognition accuracy using Convolutional Neural Networks. However, upon analysis of their confusion matrix, we found that they represented the character ‘ङ’ (nga) as ‘ड’ (Da) (a combination of two characters ‘ड’ and ‘.’), which resulted in a high rate of error for that character, especially since 70% of their dataset was generated artificially. This study uses Tesseract and ANN with some modifications, wherever necessary, for Nepali script.

Likewise, in a study by Owais Mujtaba Khandey and Dr. Samad Dadvandipour, [6] covers the work done in handwritten digit recognition and the various classifiers that have been developed. Methods like MLP, SVM, Bayesian networks, and Random forests were discussed with their accuracy and are empirically evaluated. Boosted LetNet 4, an ensemble of various classifiers, has shown maximum efficiency among these methods. The boosted LeNet 4 method performs the best with the accuracy of 99.3% and is the best among the methods that have been studied in this paper. The only tradeoff is the training time, which is very large and is about five weeks. The operational/actual recognition time is 0.05 ms.

Moreover, in the study by Nikita Singh, in 2018, the image partitioning technique is used for piecewise Histogram of Oriented Gradients (HOG) features extraction. To train the neural network, a feature vector composed of HOG features of all partitions is used. The proposed approach achieves the maximum of 99.27% classification accuracy in training and is able to recognize the different handwritten Devanagari characters with an average accuracy of 97.06%. The proposed approach may be useful in the application for blind people to read the handwritten contents. [7]

# **CHAPTER 3: SYSTEM ANALYSIS**

## **3.1 System Analysis**

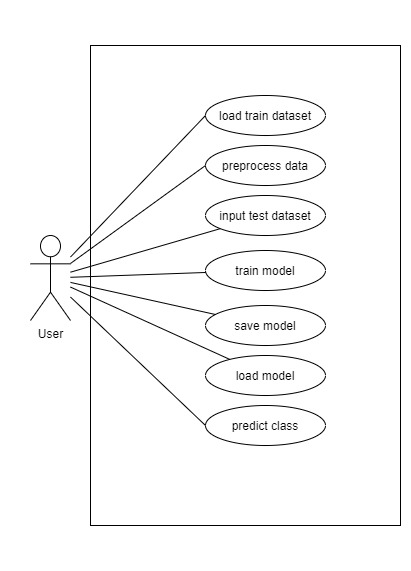
### **3.1.1 Requirement Analysis**

Requirements analysis, also called requirements engineering, is the process of determining user expectations for a new or modified product. These features, called requirements, must be quantifiable, relevant and detailed. It involves frequent communication with system users to determine specific feature expectations, resolution of conflict or ambiguity in requirements as demanded by the various users or groups of users, avoidance of feature creep and documentation of all aspects of the project development process from start to finish.

### **3.1.2 Functional Requirements**

A Functional Requirement is a description of the service that the software must offer. It describes a software system or its component. A function is nothing but inputs to the software system, its behavior, and outputs. It can be calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform.

The following is the use case diagram that describes different functionalities of the system and interaction between actors



*Figure 3.1: Use-Case Diagram for Handwritten Nepali Digit Recognition*

**Use Case Description:**

*Table 3.1: Use Case for Load training datasets*

|  |  |
| --- | --- |
| Use-Case Identifier | UC1 - Load training datasets |
| Primary Actor | User |
| Secondary Actor | None |
| Description | It loads the required size of the training datasets from the local directory. |
| Pre-condition | All the required dependencies should be imported and the data files should be in the local directory or the system should be connected to the internet. |
| Success scenario | The training data is successfully loaded into the system |
| Failure scenario | The training data is not loaded |

*Table 3.2: Use Case for Preprocess data*

|  |  |
| --- | --- |
| Use-Case Identifier | UC2 – Preprocess data |
| Primary Actor | User |
| Secondary Actor | None |
| Description | It preprocesses the loaded data from the local directory by cleaning the data and making it suitable for a machine learning model which also increases its accuracy and efficiency. |
| Pre-condition | The training data should be loaded and input to the model. |
| Success scenario | The data indices are properly manifested |
| Failure scenario | The data are not properly manifested and indexed |

*Table 3.3: Use Case for Input Training Dataset*

|  |  |
| --- | --- |
| Use-Case Identifier | UC3 – Input training dataset |
| Primary Actor | User |
| Secondary Actor | None |
| Description | It puts the datasets into the model for the training process. |
| Pre-condition | All the required files should be in the local directory or the system should be connected to the internet. |
| Success scenario | The preprocessed data is successfully put into the system |
| Failure scenario | The system cannot be trained as planned |

*Table 3.4: Use Case for Train Model*

|  |  |
| --- | --- |
| Use-Case Identifier | UC4 – Train model |
| Primary Actor | User |
| Secondary Actor | None |
| Description | It trains the model using the training dataset available in the local directory. |
| Pre-condition | All the required data should be in the local directory or the system should be connected to the internet. |
| Success scenario | The system gets trained properly |
| Failure scenario | The system remains untrained |

*Table 3.5: Use Case for Load Model*

|  |  |
| --- | --- |
| Use-Case Identifier | UC5 – Load model |
| Primary Actor | User |
| Secondary Actor | None |
| Description | It loads the model that is trained with the training data after the training phase. |
| Pre-condition | The model should be saved and available. |
| Success scenario | The model is loaded to get the test data |
| Failure scenario | The model cannot be loaded for further tests |

*Table 3.6: Use Case for Save Model*

|  |  |
| --- | --- |
| Use-Case Identifier | UC6 – Save model |
| Primary Actor | User |
| Secondary Actor | None |
| Description | It saves the model that is trained from the available training datasets. |
| Pre-condition | The model must be trained. |
| Success scenario | The model is successfully saved |
| Failure scenario | The model cannot be saved for further processes |

*Table 3.7: Use Case for Predict Class*

|  |  |
| --- | --- |
| Use-Case Identifier | UC7 – Predict class |
| Primary Actor | User |
| Secondary Actor | None |
| Description | It predicts the data after the model is loaded. |
| Pre-condition | The model must be loaded. |
| Success scenario | The system predicts the test data successfully |
| Failure scenario | The system cannot predict the test data |

### **3.1.3 Non-functional Requirements**

Non-functional requirements are focused on how the system goes about delivering a specific function. At first glance they might be seen as less important than functional requirements, but both have a part to play in a good system. Non-functional requirements do not have an impact on the functionality of the system, but they do impact on how it will perform.

**Reliability:** The system should classify handwritten digits precisely despite the stroke’s thickness or the position, orientation, or intensity of the digit.

**Performance:** The classification time should be as minimum as possible making the system more robust and responsive.

**Flexibility:** The system should be flexible for the addition of any other algorithm or model architecture.

**Usability:** The system should be easy to use for any users with the necessary abstraction of the model implementation.

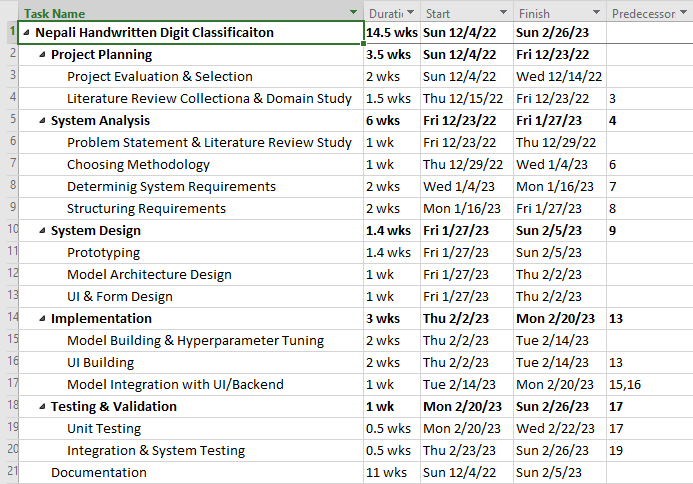
### **3.1.4 Feasibility Analysis**

Feasibility study is a short, focused study that takes place early in the requirement engineering process. The aim of feasibility study is to find out whether the system can be implemented or not and whether the system is worth implementing. Following are the feasibility study processes that were conducted during the system analysis phase.

#### **3.1.4.1 Schedule Feasibility**

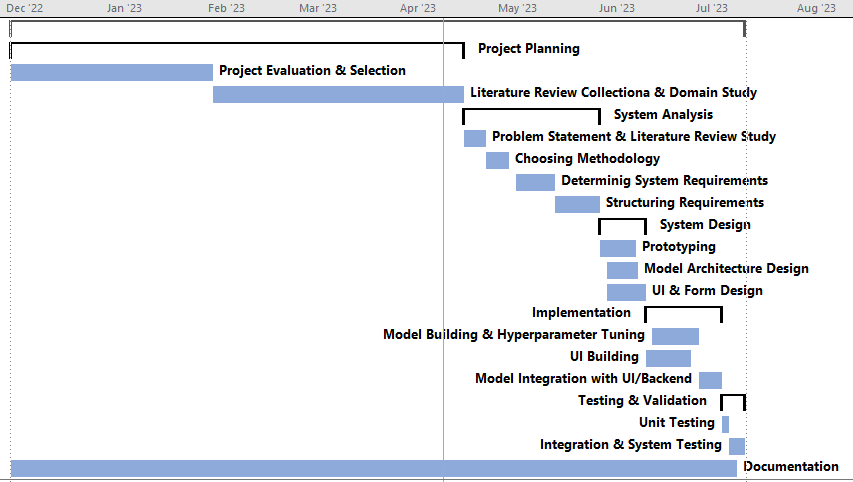
Schedule feasibility looks after the potential tie frame for the competition of the project. It also looks after the major activities and their time period or constraints involved. By thorough analysis, this project is feasible in schedule and can be completed in the time frame.

Here is the work breakdown structure of the project:



*Figure 3.2: Work Breakdown Structure of Handwritten Nepali Digit Classification System*

Here is the Gantt Chart for the Handwritten Nepali Digit Classification System:



*Figure 3.3: Gantt Chart of the Project*

### **3.1.5 Analysis**

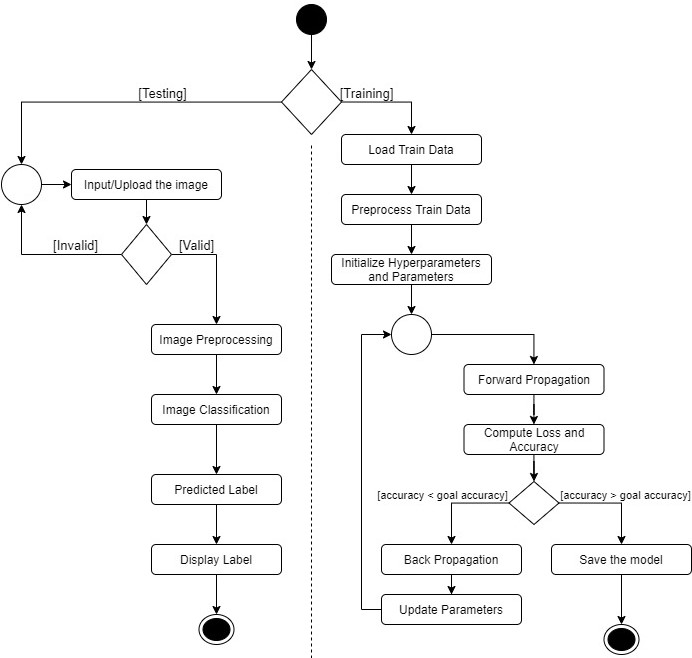
In analysis phase, we will be structuring the requirements of the project. Data Modeling and Process Modeling techniques were used for structuring the requirements.

**3.1.5.1 Process Modeling**

Process Modeling represents processes that capture, manipulate, store, and distribute data between a system and its environment and among the system components. Activity diagram, Sequence Diagram, etc. are the examples of process modeling techniques.

##### **Activity Diagram**

An activity diagram is a type of UML diagram that visually represents the flow of activities and actions within a system or process. It is a graphical representation of a workflow, showing the sequence of activities, decision points, and the flow of control between them. Activity diagrams are commonly used in software engineering, business process modeling, and other fields where complex processes need to be visualized and analyzed. They provide a high-level view of the system's behavior and help identify potential problems or areas for optimization. In an activity diagram, activities are represented by nodes, and transitions between activities are represented by arrows.



*Figure 3.4: Activity Diagram for Handwritten Nepali Digit Classification*

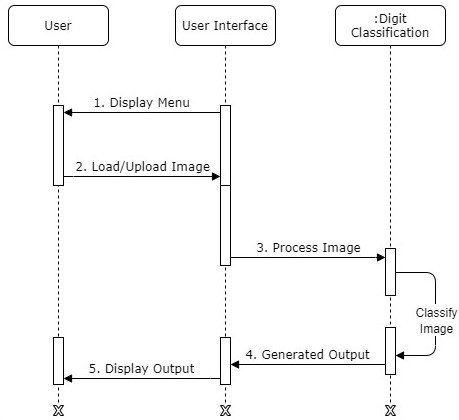
Here in the activity diagram, we can see in the testing phase, the system takes the input in the form of uploaded image from the user and validates it. It then preprocesses the image and performs the classification using the ANN and predicts the outcome and finally displays the output in the UI.

In the training phase, the train data is loaded and preprocessed performing the hyperparameter tuning process parallelly. After that forward propagation is implemented using ANN and then the accuracy is determined. If the accuracy is less than the desired accuracy then the process of backward propagation is done again and again until the desired accuracy is obtained. If the accuracy is obtained, then the model is saved and ready.

**Sequence Diagram**

A sequence diagram is a type of UML (Unified Modeling Language) diagram that illustrates how objects in a system interact with each other over time. It depicts the interactions between the objects in a chronological order, emphasizing the time ordering of messages exchanged between the objects.

In a sequence diagram, objects are represented as vertical lifelines, and messages are represented as horizontal arrows between the lifelines. The messages are labeled to indicate the information being transmitted, and the sequence in which they occur is shown by the order of the arrows on the diagram.

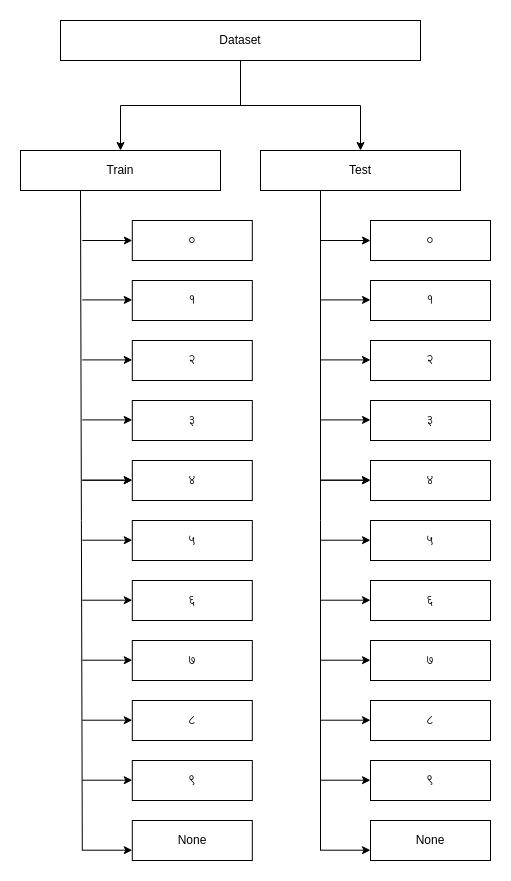


*Figure 3.5: Sequence Diagram for Handwritten Nepali Digit Classification*

Here in the sequence diagram, the user uploads the image to be classified to the system via the displayed UI. Then the system processes the image by running the image through the classification algorithm using ANN. And once the image is classified, the generated output is sent back to the UI where the user is able to view the predicted result of the image they uploaded.

### **3.1.6 Dataset Analysis**

The dataset the project intends to use consists of 2000 data for each label initially among which 85% is divided into train data and remaining 15% as test data The data for each label are stored in the same folder and the name of the folder is same as the label for the data. The folder structure for the dataset is visualized below:



*Figure 3.6: Folder Structure for dataset*

Each image is of a fixed size i.e., 32 \* 32 pixels. This makes it easy for the data preprocessing as it removes the overhead of resizing the images into a same size. The images are black and white hence have only one-color channel.

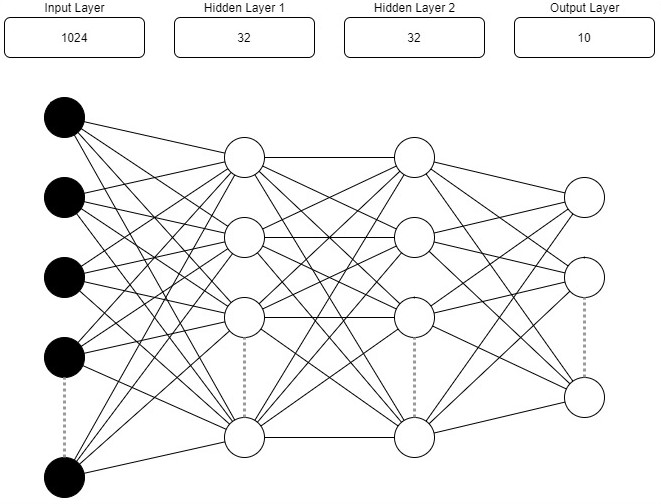
# **CHAPTER 4: SYSTEM DESIGN**

## **4.1 Design**

### **4.1.1 Model Architecture**

The project is based upon the Multilayer Perceptron architecture. The input layer is of size 1024, this is due to the fact that each image size is 32 \* 32 pixels and they are stacked into a single dimension resulting in 1024 inputs.

A simple representation of the model architecture is shown below:



*Figure 4.1: Multilayer Perceptron*

**The model consists of the 3 types of layers:**

1. **Input Layer**

The input layer of Artificial Neural Network is the layer responsible for bringing in the data from outside to the network. The shape of the input layer depends on the size of the input. In the case of this project, the input layer will consist of 32 \* 32 image stacked into a single vector.

1. **Hidden layer**

The hidden layers are the layers that are stacked between input and the output layer of the network. These layers don’t interact with the external environments of the network. They take inputs from input layers or other hidden layers and the outputs from these layers are utilized as input in output layer or the hidden layers.

The role of the hidden layers is to process the input information and produce an output that can be used to make a prediction.

For this project the size and the number of the hidden layer will be determined based on the performance of the model but the model will consist 2 hidden layers

**First Hidden Layer:**

The first hidden layer is responsible for extracting features from the input data. Each node in this layer receives input from every pixel in the image and applies a set of weights and biases to produce a single output value. These values represent important features or patterns that can be used to classify the image. In this case, this layer may contain multiple nodes, and each node may use a different set of weights and biases to extract different features from the input image.

**Second Hidden Layer:**

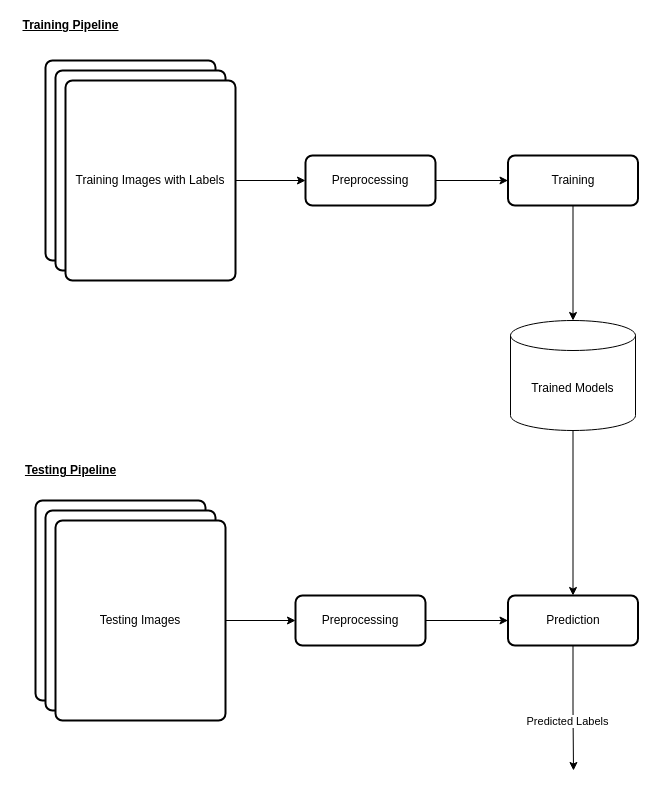
The second hidden layer takes the output from the first hidden layer as input and further processes it to produce a more refined set of features. This layer may have fewer nodes than the first hidden layer, but each node may be more complex and specialized, allowing it to identify more intricate patterns in the input data. The second hidden layer may use a different set of weights and biases than the first layer to perform this processing.

Overall, the combination of these two hidden layers allows the neural network to extract increasingly complex and abstract features from the input image, leading to more accurate predictions. The output of the second hidden layer is then passed to the output layer, which produces the final prediction for the digit class of the input image.

1. **Output Layer**

The output layer is the layer in which the network outputs its predictions. For this project the output layer will output 11 possible outcomes, 10 for every Nepali digit and 1 for non-digit character recognition.

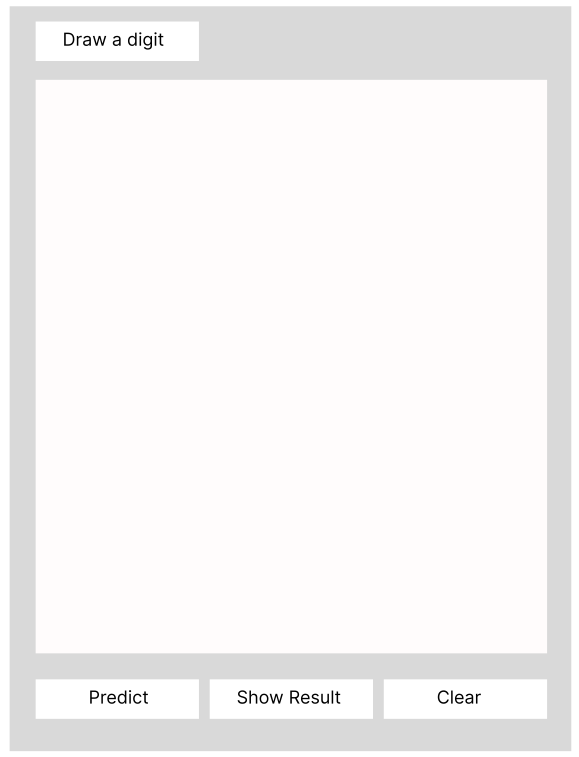
### **4.1.2 Model Building Pipeline**



*Figure 4.2: Model Building Pipeline*

### **4.1.3 UI Design**

User interface (UI) design is the process designers use to build interfaces in software or computerized devices, focusing on looks or style. Designers aim to create interfaces which users find easy to use and pleasurable. The project UI design is implemented using figma tools which is suitable for the project. The design helped the project for the user interface and user interaction which enables us to draw the section for the user interface.



# **CHAPTER 5: IMPLEMENTATION**

## **5.1 Tools Used**

The following tools were used for the development of the project and design of the project

documents:

### **5.1.1 Development Tools**

Development tools are the software packages that were used for the development and maintenance of the project. The following software packages were used for this purpose:

**Anaconda**

Anaconda is a package management and deployment tool. It was used to create and manage the conda environment for the development of the project.

**Jupyter Notebook**

Jupyter notebook is an interactive python interface for programming. It was used for the majority of the development task and code experimentation.

**Visual Studio Code**

Visual Studio Code is a code editor redefined and optimized for building and debugging modern web and cloud applications. Visual Studio code is used as a code editor for developing the backend and frontend of the system.

**Python**

Python is a general-purpose language largely used for developing Machine Learning Projects. Python is used for designing the machine learning model and serve as a backend for the hosted model

**HTML/CSS**

The Hypertext Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. CSS is the styling sheet used to style the HTML documents. The HTML & CSS are used to create the User Interface in this project.

**JavaScript**

JavaScript is a general-purpose language generally used in writing the logic in the Web pages. JavaScript was used to handle the front-end logic of the system.

**Git and GitHub**

Git is a version control system. And GitHub is an online tool that uses git for version control and team collaboration. The combination of these tools was used for project versioning and code collaboration within the project team.

### **5.1.2 Design and Documentation Tools**

The design tools were used for designing the system architecture, User Interfaces and project mockups. The documentation tools were used to prepare the necessary documents for the project.

**Diagrams.net**

Diagrams.net is a free and open-source cross-platform graph drawing software that can be used to create diagrams such as flowcharts, wireframes, UML diagrams, organizational charts, and network diagrams. The project uses diagrams.net to prepare the diagrams such as Use Case diagram, Sequence Diagram, Activity Diagram, Model Architecture, etc.

**MS Word**

MS Word is a word processing tool that is used in preparing the documents necessary for the project.

**MS PowerPoint & Google Slides**

MS PowerPoint and Google slides were used extensively for preparing the presentation slides for the project.

**Figma**

Figma is a tool for designing the Prototypes and mockups. Figma was used to design the prototype of the frontend of the system.

**5.1.3 Dependencies**

Dependencies are the modules or the packages that the project needs during its operation. Following are the dependencies for the project:

**NumPy**: NumPy is a python package used to carry out all the numeric computation

**Matplotlib**: matplotlib is a python package used to generate graphs and visualizations of the system

**Pickle**: Pickle is a python package used to save and load trained models

**Pillow**: Pillow is a python package used to load, save and process image

**React**: React is a JavaScript library used to handle the frontend logic of the system

## **5.2 Methodology**

### **5.2.1 Data Preparation**

The data need to be prepared in order to fit to the model. The data for training collected were in the image (\*.jpg) format and were later converted into the dataframe for simplistic view. The 32 \* 32 images were converted into a 1-D array and appended as a row in the dataframe. The last column contains the label of the images.

### **5.2.2. Model Introduction**

The model used in the project is Artificial Neural Network or simply a Multilayer Perceptron (MLP). The Model has various components these components include:

* Model Representation
* Parameters and Hyperparameters
* Activation Functions
* Cost Function
* Weight Initialization
* Forward and Backward Propagation, etc.

#### **5.2.2.1 Model Representation**

The model representation refers to how the Neural Networks is designed. The model has multiple layers categorized into 3 types:

**Input Layer:** The Input layer is the layer where the inputs to the model are provided. For this project the input layer has a shape of (1,1024). This is because the images that we provide as input to the model has a shape (32,32) and the image is passed as a 1-D vector to the model.

**Hidden Layer(s):** A hidden layer is the intermediate layer between the input and output layer. A neural network can have multiple hidden layers. For this project there are 2 hidden layers of 32 neurons each with shape of (1024,32) and (32,32) respectively.

**Output Layer:** The output layer of the system is the layer that outputs the predictions of the neural network. For this project, there are 10 neurons in the output layer.

#### **5.2.2.2 Parameters and Hyperparameters**

Parameters are the values that will change during the training process. While the hyperparameters are the values that are set during the model initialization and remain after that. Following are the hyperparameters in the model designed in the project:

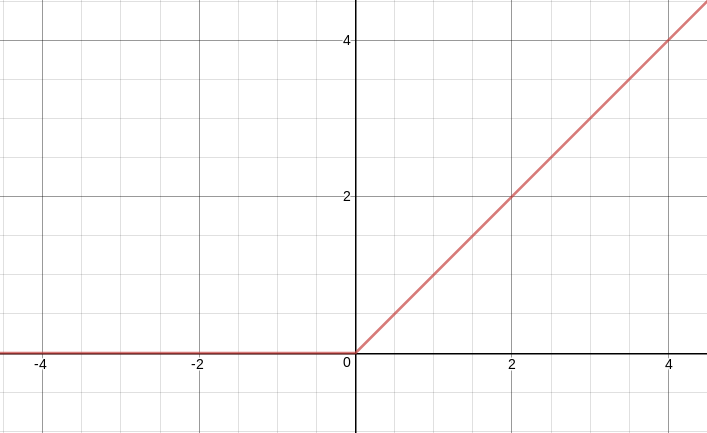
* Learning rate
* Hidden layers
* Neurons in the hidden layers
* Optimizers
* Number of iterations/epochs

#### **5.2.2.3 Activation Functions**

Activation Functions bring the non-linearity in the model. They are applied to the linear output of the system. There are various types of the activation functions that can be used in the neural networks. We have used two types of the activation functions in the model

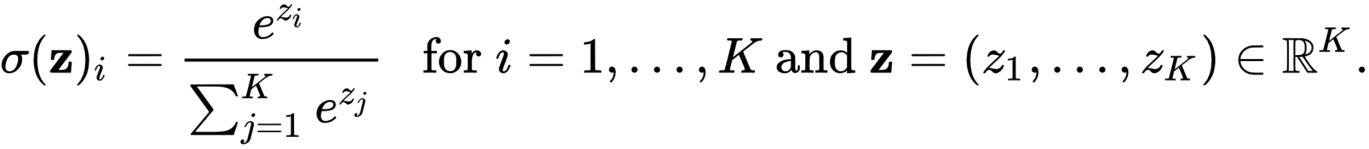
**Rectified Linear Unit (ReLU): T**he rectifier or ReLU activation function is an activation function defined as the positive part of its argument. This can be represented as:

ReLU(x) = maximum (0, x)



*Figure 5.1: ReLU Activation Function Graph*

**SoftMax:** The SoftMax function, also known as softargmax or normalized exponential function, converts a vector of K real numbers into a probability distribution of K possible outcomes. The formula for softmax is shown below.



### **5.2.3 Baseline Modelling**

A baseline model is essentially a simple model that acts as a reference in a machine learning project. The baseline model for this project had the following configuration:

*Table 5.1: Baseline Modelling*

|  |  |  |
| --- | --- | --- |
| **Properties** | **Values** | **Remarks** |
| Hidden Layers | 2 |  |
| Layer Sizes   1. Input Layer 2. Hidden Layer(1) 3. Hidden Layer(2) 4. Output Layer | 1024  32  32  10 | With ReLU Activation  With ReLU Activation  With Softmax Activation |
|
|
|
|
| Learning Rate | 0.1 |  |
| Optimizer | Batch Gradient Descent |  |
| Iterations | 4000 |  |
| Accuracy:   1. Train 2. Test | 89.9%  87.5% |  |
| Precision   1. Train 2. Test | 90.2%  87.5% |  |
| Recall   1. Train 2. Test | 90.2%  87.5% |  |

### **5.2.4 Model Optimization**

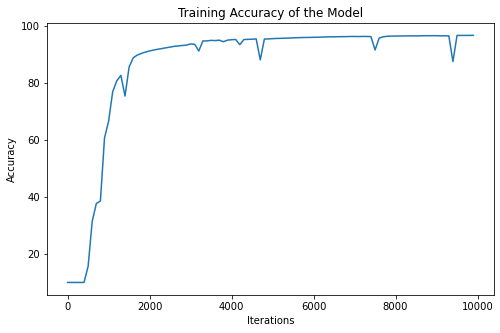
Model optimization is the process of achieving the optimal outcome from the model. Various methods were used for model optimization in this project. Hyperparameters were tuned to obtain the optimal performance in this project, this included the tuning the parameters such as regularization parameter, normalization of dataset and number of iterations/epochs.

### **5.2.5 Final Model**

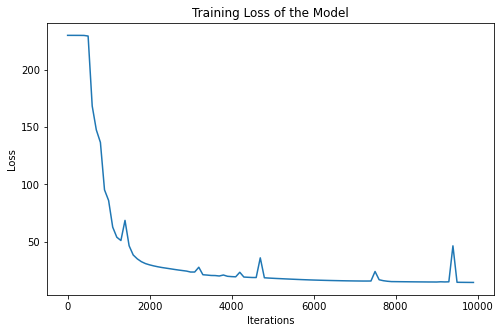
The final model has changes compared to the baseline model. The Final Model is a fully connected neural network with 2 hidden layers. The final model uses the He-initialization for weight initialization and L2 regularization for regularization. The more information about the final model and its performance are shown below:

*Table 5.2: Final Model*

|  |  |  |
| --- | --- | --- |
| **Properties** | **Values** | **Remarks** |
| Hidden Layers | 2 |  |
| Layer Sizes   1. Input Layer 2. Hidden Layer (1) 3. Hidden Layer (2) 4. Output Layer | 1024  32  32  10 | With ReLU Activation  With ReLU Activation  With Softmax Activation |
|
|
|
|
| Learning Rate | 0.01 |  |
| Optimizer | Batch Gradient Descent |  |
| Iterations | 10000 |  |
| Accuracy:   1. Train 2. Test | 95.6%  94.5% |  |
| Precision   1. Train 2. Test | 95.2%  94.8% |  |
| Recall   1. Train 2. Test | 95.2%  96.5% |  |



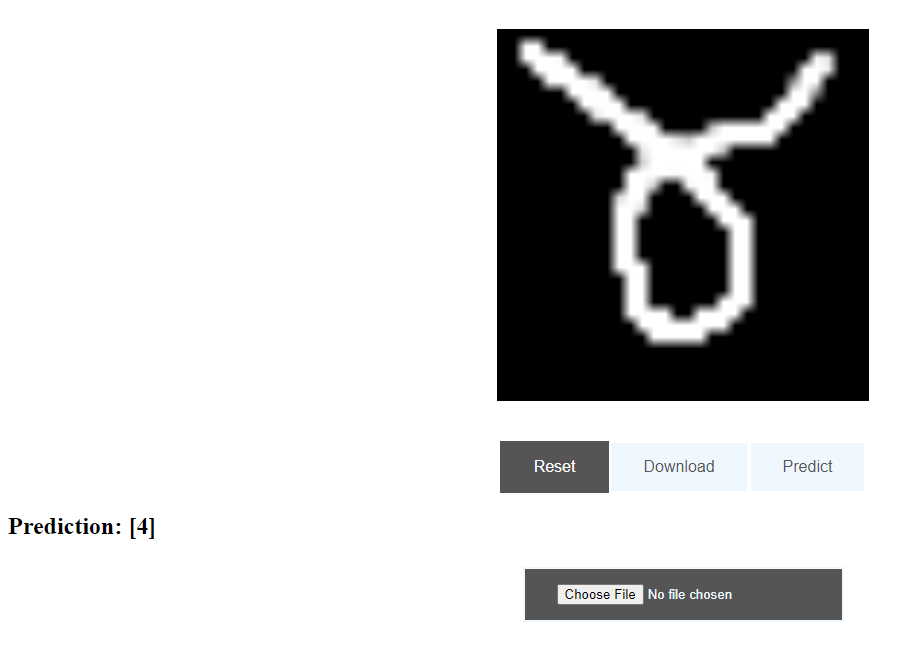
*Figure 5.2: Training accuracy of the final model*



*Figure 5.3: Training loss of the final model*

### **5.2.6 Model Deployment**

The model was deployed on the API using FastAPI and the UI for the prediction was built on HTML/CSS, JavaScript and React Library. Following image shows the UI of the system where the model was deployed.



# **CHAPTER 6: TESTING**

## **6.1 Unit Testing**

Following are the test cases for the image digit classification:

*Table 6.1 : Test scenarios for the image digit classification*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S.**  **N.** | **Test Scenario** | **Actions** | **Input Data** | **Expected outcome** | **Observed**  **Outcome** | **Assertion** |
| **1.** | Input valid the  image for testing | 1.0 Open the  webpage  2.0 Input test  data | Input the image of label 3 | 1. Image labelled as 3 | As expected | Pass |
| **2.** | Input Invalid Data | 1.0 Open the  webpage  2.0 Input test  data | Input of a non-image file | 1. Invalid  Datatype | As expected | Pass |
| **3.** | Input a non nepali digit | 1.0 Open the  webpage  2.0 Input test  data | Input a non-nepali digits | 1. Image  Labelled as non-digit | As expected | Pass |

Following are the test scenarios for Neural Network Initialization:

*Table 6.2: Test scenarios for Neural Network Initialization*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S.**  **N.** | **Test Scenario** | **Actions** | **Input Data** | **Expected outcome** | **Outcome** | **Pass/**  **Fail)** |
| **1.** | Input valid the  Values for Neural Network Initialization | 1.0 Input both input and hidden layer information | (1024, [64,64,10]) | 1. Initialize the Model | As expected | Pass |
| **2.** | Input valid the  Values for Neural Network Initialization | 1.0 Input only the input layer information | (1024 , ) | 1. Initializes the model with 1024 input layers | As expected | Pass |
| **3.** | Input invalid values for Neural Network Initialization | 1. Empty initialization of Class | None | 1. Throws an error with message “Please Input the input layer information” | As expected | Pass |

# **CHAPTER 7: CONCLUSION AND FUTURE RECOMMENDATIONS**

## **7.1 Conclusion**

Use of Neural Networks in the Character Recognition has been a topic of interest for a long time now. The capability of them to learn anything provided to them makes it easy to make it happen. The model can efficiently be used for Nepali Digits Recognition as show in the project. Using the neural networks, the project was able to obtain an accuracy more than 95%. Although being a simple model with only two hidden layers and 32 neurons in each layer we were able to improve the performance of the model from 89% in baseline model to 95% in the final model using techniques such as regularization, normalization and other optimization techniques. Although there are rooms for improvement the project serves its purpose that too with a simple architecture.

## **7.2 Future Recommendations**

Although the project has a good performance there are various aspects the project could improve upon. There are better and faster model architectures developed for the image related classification such as Convolutional Neural Networks and Vision Transformers. This has been proven by the various experiments conducted by the researchers in the field. The Project could improve on that aspect in future. The project is also only able to recognize the single characters not the character streams which would have a bigger implication and usage in the real-world practice.

# **REFERENCES**

|  |  |
| --- | --- |
| [1] | N. P. a. B. Bal, *Improving nepali OCR performance by using hybrid recognition approaches,” 2016 7th International Conference on Information, Intelligence, Systems & Applications (IISA),* 2016. |
| [2] | G. H. a. R. W. David Rumelhart, *Learning internal representations by error propagation,* 1985. |
| [3] | L. B. Y. B. a. P. H. Y. Lecun, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324,* 1998. |
| [4] | S.-C. M. Yadav, *Optical character recognition for Hindi language using a neural-network approach,” Journal of Information Processing Systems, vol. 9, no. 1, pp. 117–140,* 2013. |
| [5] | M. K. S. Bhattarai, *Optical character recognition system for Nepali language using ConvNet,” Proceedings of the 9th International Conference on Machine Learning and Computing,* 2017. |
| [6] | O. M. K. a. D. S. Dadvandipour, *Analysis of machine learning algorithms for character recognition: A case study on handwritten digit recognition,” Indonesian Journal of Electrical Engineering and Computer Science, vol. 21, no. 1, p. 574,* 2021. |
| [7] | N. Singh, *An efficient approach for handwritten Devanagari character recognition based on Artificial Neural Network,” 2018 5th International Conference on Signal Processing and Integrated Networks (SPIN),* 2018. |
| [8] | DeepAI.org, " Multilayer Perceptron," 2023. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron. |
| [9] | á. Abonyi, "The CRISP-DM methodology," [Online]. Available: https://www.researchgate.net/figure/The-CRISP-DM-methodology-as-continuous-data-driven-improvement-process-CRISP-DM-2000\_fig1\_284887315. |
| [10] | C. M. Bishop., "Chapter 5 - Neural Networks," in *Pattern recognition and machine learning*, New York, NY: Springer New York, 2016. |

# APPENDIX