

**Tribhuvan University**

**Orchid International College**

**A Final Year Project Report**

**On**

**HANDWRITTEN DIGIT RECOGNITION USING DEEP LEARNING**

**Under the Supervision of**

**Er. Dhiraj Kumar Jha**

**Lecturer**

**Orchid International College**

**Submitted To:**

**Department of Computer Science and Information Technology**

**Orchid International College**

**In partial fulfillment of the requirement for the Bachelor Degree in Computer Science and Information Technology**

**Submitted By:**

**Biswas Poudyal (10526/073)**

**Krishna Khanal (10691/073)**

**Sameer Kattel (10713/073)**

**Subina Lama (10720/073)**

**December, 2020**



# SUPERVISOR’S RECOMMENDATION

I hereby recommend that the report prepared under my supervision by Biswas Poudyal (TU Exam Roll No. 10526/073), Krishna Khanal (TU Exam Roll No. 10691/073), Sameer Kattel (TU Exam Roll No. 10713/073), Subina Lama (TU Exam Roll No. 10720/073) entitled “**HANDWRITTEN DIGIT RECOGNITION USING DEEP LEARNING**” in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Information Technology be processed for evaluation.

…………………..…….

**Er. Dhiraj Kumar Jha**

Project Coordinator, Department of CSIT

Orchid International College

Bijayachowk, Gaushala

******

# CERTIFICATE OF APPROVAL

This is to certify that this project prepared by Biswas Poudyal (TU Exam Roll No. 10526/073), Krishna Khanal (TU Exam Roll No. 10691/073), Sameer Kattel (TU Exam Roll No. 10713/073), Subina Lama (TU Exam Roll No. 10720/073) entitled “**HANDWRITTEN DIGIT RECOGNITION USING DEEP LEARNING**” in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Information Technology has been well studied. In our opinion, it is satisfactory in the scope and quality as a project for the required degree.

|  |  |
| --- | --- |
| ------------------------------------------------  **Er. Dhiraj Kumar Jha**  Project Coordinator,  Department of Computer Science and IT  Orchid International College,  Bijayachowk, Gaushala | ------------------------------------------------  **Bijay Mishra**  Program Coordinator,  Department of Computer Science and IT  Orchid International College,  Bijayachowk, Gaushala |
| -------------------------------------  **External Examiner**  Central Department of Computer Science and IT  Tribhuvan University  Kirtipur, Nepal | |

# ACKNOWLEDGEMENT

In the process of studying for and preparing this project report, we were fortunate to obtain the assistance of respected personalities, whom we would like to present our genuine thanks. Our appreciation goes first to our supervisor, Er. Dhiraj Kumar Jha. He is an extremely knowledgeable, friendly and patient person who provided us with sufficient guidance and reminders to inspire us to stay focused and reflect deeply on the purpose of this study. His recommendations and inputs have always improved this project, and finally producing a report of excellent quality. His help in stand-up meetings during difficulties has helped gain success in the project as represented today.

We want to express our sincere gratitude towards the Department of Computer Science and Information Technology, Orchid International College, for providing us with the wonderful opportunity, encouragement and environment to explore our skills and gain learning experience through this major project.

We want to express our thanks to Mr Bijay Mishra, program coordinator, Orchid International College, for his constant motivation, support and for providing us with a suitable working environment. His feedbacks help us regress and refine after each revision.

We want to thank all the faculty members, friends, parents, well-wishers, seniors, and juniors who directly and indirectly contribute to our project. Their suggestions, criticisms, and feedback have always proved to be in our favours and take small yet valuable steps toward the milestones.

Sincerely,

Biswas Poudyal (10526/073)

Krishna Khanal (10691/073)

Sameer Kattel (10713/073)

Subina Lama (10720/073)

# ABSTRACT

Handwritten Digit Recognition has been a topic of research in the field of OCR since 1998, first performed by LeCun using MNIST dataset with a 12% error rate. Since then, many researchers have worked on the problem using different approaches, being able to achieve an astonishing state-of-the-art performance. This project implements a simple 3- layered feedforward network to classify the handwritten digits. Some modern but less sophisticated algorithms and techniques are selected to train the model in an experimental approach. The best combination of hyperparameters was selected using a Coarse-to-Fine search. The MNIST dataset was then augmented using various affine transformations and used for training the final tuned model. The model was able to achieve an accuracy of 99.16% as a result. Furthermore, the project also implements a simple GUI interface with the capability to draw a digit on the interface or upload an image of a digit from local storage for prediction.

Keywords: ***MLP, handwritten digit recognition, dropout, MGD, Adam, hyperparameter tuning, data augmentation***

**TABLE OF CONTENTS**

[SUPERVISOR’S RECOMMENDATION i](#_Toc60002985)

[CERTIFICATE OF APPROVAL ii](#_Toc60002986)

[ACKNOWLEDGEMENT iii](#_Toc60002987)

[ABSTRACT iv](#_Toc60002988)

[LISTS OF FIGURES viii](#_Toc60002989)

[LISTS OF TABLES x](#_Toc60002990)

[LIST OFABBREVIATIONS xii](#_Toc60002991)

[Chapter 1 : INTRODUCTION 1](#_Toc60002992)

[1.1 Project Introduction 1](#_Toc60002993)

[1.2 Problem Statement 2](#_Toc60002994)

[1.3 Objectives 2](#_Toc60002995)

[1.4 Scope and Limitations 2](#_Toc60002996)

[1.5 Report Organization 3](#_Toc60002997)

[Chapter 2 : LITERATURE REVIEW 5](#_Toc60002998)

[Chapter 3 : SYSTEM ANALYSIS 7](#_Toc60002999)

[3.1 Requirement Analysis 7](#_Toc60003000)

[3.1.1 Functional Requirements 7](#_Toc60003001)

[3.1.2 Non-functional Requirements 12](#_Toc60003002)

[3.2 Feasibility Analysis 12](#_Toc60003003)

[3.2.1 Data Availability 12](#_Toc60003004)

[3.2.2 Literature Availability 12](#_Toc60003005)

[3.2.3 Availability and Cost of Computational Resources 13](#_Toc60003006)

[3.2.4 Operational Feasibility 13](#_Toc60003007)

[3.3 Data Description 13](#_Toc60003008)

[3.4 Process Modeling 15](#_Toc60003009)

[3.4.1 Context Diagram 15](#_Toc60003010)

[3.4.2 Level 0 DFD 16](#_Toc60003011)

[3.4.3 Level 1 DFD 17](#_Toc60003012)

[Chapter 4 : SYSTEM DESIGN 18](#_Toc60003013)

[4.1 Architecture Design 18](#_Toc60003014)

[4.1.1 Layers 18](#_Toc60003015)

[4.1.2 Activation function 19](#_Toc60003016)

[4.1.3 Cost function 19](#_Toc60003017)

[4.2 Data Design 21](#_Toc60003018)

[4.3 Process Design (Methodology Flow Chart / Process model) 22](#_Toc60003019)

[4.3.1 Data Collection 22](#_Toc60003020)

[4.3.2 Data Preparation 22](#_Toc60003021)

[4.3.3 Baseline Model 23](#_Toc60003022)

[4.3.4 Model Optimization 24](#_Toc60003023)

[4.3.5 Model Tuning 28](#_Toc60003024)

[4.3.6 Model Deployment: 28](#_Toc60003025)

[4.4 Experiments Design 29](#_Toc60003026)

[4.5 Interface Design 29](#_Toc60003027)

[Chapter 5 : IMPLEMENTATION 31](#_Toc60003028)

[5.1 Tools Used 31](#_Toc60003029)

[5.1.1 Development Tools: 31](#_Toc60003030)

[5.1.2 Design Tools: 31](#_Toc60003031)

[5.1.3 Dependencies 32](#_Toc60003032)

[5.2 System Configuration 33](#_Toc60003033)

[5.3 Methodology 34](#_Toc60003034)

[5.3.1 Data Collection 34](#_Toc60003035)

[5.3.2 Data Preparation 35](#_Toc60003036)

[5.3.3 Baseline Model 40](#_Toc60003037)

[5.3.4 Model Optimization 41](#_Toc60003038)

[5.3.5 Model Tuning 41](#_Toc60003039)

[5.3.6 Model Deployment 41](#_Toc60003040)

[5.4 Experiments and Results 42](#_Toc60003041)

[5.4.1 Experiment 1: Initialization Selection 43](#_Toc60003042)

[5.4.2 Experiment 2: Optimizer Selection 43](#_Toc60003043)

[5.4.3 Experiment 3: Regularizer Selection 44](#_Toc60003044)

[5.4.4 Experiment 4: Hyperparameter Tuning 45](#_Toc60003045)

[5.5 Final model 50](#_Toc60003046)

[Chapter 6 : TESTING 54](#_Toc60003047)

[6.1 Unit Testing 54](#_Toc60003048)

[6.2 Integration Testing 61](#_Toc60003049)

[6.3 System Testing 62](#_Toc60003050)

[Chapter 7 : CONCLUSION 65](#_Toc60003051)

[7.1 Project Conclusion 65](#_Toc60003052)

[7.2 Future Work 65](#_Toc60003053)

[REFERENCES 67](#_Toc60003054)

[APPENDIX 70](#_Toc60003055)

[Notebook Screenshots: 70](#_Toc60003056)

[UI Screenshots: 74](#_Toc60003057)

[Source Code: 76](#_Toc60003058)

# LISTS OF FIGURES

[Figure 3.1: Use Case diagram for Handwritten Digit Recognition System 8](#_Toc60001953)

[Figure 3.2: Digit 8 in the MNIST Dataset 14](#_Toc60001954)

[Figure 3.3: Data distribution in Training set 14](#_Toc60001955)

[Figure 3.4: Data distribution in Test set 15](#_Toc60001956)

[Figure 3.5: Context Diagram of Handwritten Digit Recognition System 15](#_Toc60001957)

[Figure 3.6: Level 0 DFD of Handwritten Digit Recognition System 16](#_Toc60001958)

[Figure 3.7: Level 1 DFD of Handwritten Digit Recognition System 17](#_Toc60001959)

[Figure 4.1: Types of Neural Network based on number of hidden layers 18](#_Toc60001960)

[Figure 4.2: Block Diagram of MLP architecture used in th project 20](#_Toc60001961)

[Figure 4.3: Process model used in the project 22](file:///Y:\Sameer%20Kattel\Education\Course\CSIT\7th%20Semester\Project%20Work\HandWritten-Digit-Recognition-using-Deep-Learning\8.%20Final%20Defence\Final%20Report\CSIT%20FINAL%20REPORT.docx#_Toc60001962)

[Figure 4.4: Neural Network showing Dropout of individual neural unit 27](#_Toc60001963)

[Figure 4.5: Wireframe of the Handwritten Digit Recognition System 30](#_Toc60001964)

[Figure 5.1: Sample MNIST images with their corresponding labels 35](#_Toc60001965)

[Figure 5.2: Data distribution of the split training set 36](#_Toc60001966)

[Figure 5.3: Data distribution of the split development set 36](#_Toc60001967)

[Figure 5.4: Data distribution of the split test set 37](#_Toc60001968)

[Figure 5.5: Images before applying any transformation 38](#_Toc60001969)

[Figure 5.6: Images after Rotation 39](#_Toc60001970)

[Figure 5.7: Images after Shifting 39](#_Toc60001971)

[Figure 5.8: Images after Cropping and Padding 39](#_Toc60001972)

[Figure 5.9: Images after applying random blurring filters 39](#_Toc60001973)

[Figure 5.10: Images after Zooming 40](#_Toc60001974)

[Figure 5.11: Loss and Accuracy of the Baseline Model 40](file:///Y:\Sameer%20Kattel\Education\Course\CSIT\7th%20Semester\Project%20Work\HandWritten-Digit-Recognition-using-Deep-Learning\8.%20Final%20Defence\Final%20Report\CSIT%20FINAL%20REPORT.docx#_Toc60001975)

[Figure 5.12: Prediction made after deploying final model in GUI interface. 42](#_Toc60001976)

[Figure 5.13: Accuracy Graph for experiment with random initialization (left) and he-initialization (right) 43](file:///Y:\Sameer%20Kattel\Education\Course\CSIT\7th%20Semester\Project%20Work\HandWritten-Digit-Recognition-using-Deep-Learning\8.%20Final%20Defence\Final%20Report\CSIT%20FINAL%20REPORT.docx#_Toc60001977)

[Figure 5.14: Accuracy Graphs for experiments with BGD (top-left), MGD (top-right) and Adam (bottom-left) 44](file:///Y:\Sameer%20Kattel\Education\Course\CSIT\7th%20Semester\Project%20Work\HandWritten-Digit-Recognition-using-Deep-Learning\8.%20Final%20Defence\Final%20Report\CSIT%20FINAL%20REPORT.docx#_Toc60001978)

[Figure 5.15: Accuracy Graphs for experiment with L2 (left) and dropout (right) regularization 45](file:///Y:\Sameer%20Kattel\Education\Course\CSIT\7th%20Semester\Project%20Work\HandWritten-Digit-Recognition-using-Deep-Learning\8.%20Final%20Defence\Final%20Report\CSIT%20FINAL%20REPORT.docx#_Toc60001979)

[Figure 5.16: New Search Space for minibatch size and learning rate after Coarse Search 47](#_Toc60001980)

[Figure 5.17: New Search Space for minibatch size and learning rate after Fine Search 49](#_Toc60001981)

[Figure 5.18: Loss and Accuracy of the Final Model 51](file:///Y:\Sameer%20Kattel\Education\Course\CSIT\7th%20Semester\Project%20Work\HandWritten-Digit-Recognition-using-Deep-Learning\8.%20Final%20Defence\Final%20Report\CSIT%20FINAL%20REPORT.docx#_Toc60001982)

[Figure 5.19: Confusion matrix for the predictions of the development set 52](#_Toc60001983)

[Figure 5.20: Sample mislabeled images from the development set 53](#_Toc60001984)

# LISTS OF TABLES

[Table 2.1: Benchmarking on MNIST Dataset using various techniques 6](#_Toc59991146)

[Table 3.1: Use Case description for Load Dataset 8](#_Toc59991147)

[Table 3.2: Use Case description for Preprocess Datasets 9](#_Toc59991148)

[Table 3.3: Use Case description for Train Model 9](#_Toc59991149)

[Table 3.4: Use Case Description for Save Model 10](#_Toc59991150)

[Table 3.5: Use Case Description for Load Model 10](#_Toc59991151)

[Table 3.6: Use Case description for Upload or Draw a Digit 11](#_Toc59991152)

[Table 3.7: Use Case description for Classify the digit 11](#_Toc59991153)

[Table 5.1: System Configuration for the project development 33](#_Toc59991154)

[Table 5.2: Loaded datasets and their size 34](#_Toc59991155)

[Table 5.3: Split dataset and their size 35](#_Toc59991156)

[Table 5.4: Size comparison of images in the dataset before and after preprocessing 37](#_Toc59991157)

[Table 5.5: Size comparison of labels in the dataset before and after preprocessing 37](#_Toc59991158)

[Table 5.6: Baseline model metrics for training, dev and test set 40](#_Toc59991159)

[Table 5.7: Top 10 best result obtained from Coarse Search 46](#_Toc59991160)

[Table 5.8: The best result and the new search ranges obtained from Coarse Search 46](#_Toc59991161)

[Table 5.9: Top 10 best result obtained from Fine Search 48](#_Toc59991162)

[Table 5.10: The best result and the new search ranges obtained form Fine Search 48](#_Toc59991163)

[Table 5.11: Top 10 best results obtained from the Detailed Search 49](#_Toc59991164)

[Table 5.12: The Best result obtained from entire Hyperparameter tuning process 50](#_Toc59991165)

[Table 5.13: Precision, Recall, and F1-Score of the final model 51](#_Toc59991166)

[Table 6.1: Test Case 1 - Dataset Preparation Module Test 54](#_Toc59991167)

[Table 6.2: Test Case 2 - Dataset Augmentation Module Test 56](#_Toc59991168)

[Table 6.3: Test Case 3 - Utility Module Test 57](#_Toc59991169)

[Table 6.4: Test Case 4 - Neural Network Module Test 59](#_Toc59991170)

[Table 6.5: Test Case 5 - GUI Module Test 60](#_Toc59991171)

[Table 6.6: Test Case 6 - Training Module Test 61](#_Toc59991172)

[Table 6.7: Precision, Recall and F1-Score for Test set 62](#_Toc59991173)

[Table 6.8: Test Case 7 - System Test 63](#_Toc59991174)

# LIST OFABBREVIATIONS

AWS: Amazon Web Service

BGD: Batch Gradient Descent

CNN: Convolution Neural Network

CPU: Central Processing Unit

CVPR: Conference on Computer Vision and Pattern Recognition

DFD: Data Flow Diagram

GB: Gigabyte

GPU: Graphical Processing Unit

GUI: Graphical User Interface

HU: Hidden Unit

ICML: International Conference on Machine Learning

IDE: Integrated Development Environment

KNN: K-Nearest Neighbor

MB: Megabyte

MGD: Minibatch Gradient Descent

MLP: MultiLayer Perceptron

MNIST: Modified National Institute of Standards and Technology

MSE: Mean Square Error

NIST: National Institute of Standards and Technology

NN: Neural Network

OCR: Optical Character Recognition

PIL: Python Imaging Library

ReLU: Rectified Linear Unit

RMSProp: Root Mean Square Propagation

SGD: Stochastic Gradient Descent

SVM: Support Vector Machine

UC : Use Case

UI: User Interface

URL: Uniform Resource Locator

VM: Virtual Machine

# : INTRODUCTION

## 1.1 Project Introduction

Different people have different writing styles. So, people’s writings are often asymmetric with the varying size, stroke, thickness, orientation and deformation of the characters, digits being no exception. And when it comes to recognizing these characters, unlike human beings, it becomes a great deal for machines. Handwritten Digit Recognition is a machine learning approach used for training machines to perform this difficult task. Machines trained well with this approach can recognize the digits from 0 to 9 with high precision.

Handwritten Digit Recognition has been widely used in different sectors like digitizing handwritten documents, recognizing the vehicle's number plate, recognizing zip codes, reading cheque-books and many more. Having such a vivid application, it has been actively pursued by many researchers and engineers since the beginning of the Machine Learning era. A large number of techniques are used to recognize the handwritten digits with varying levels of performance. Neural Network models (Deep Learning approach) in general, have been showing state-of-the-art performance [10] in this particular task.

This project implements one such neural network model, Multilayer Perceptron (MLP), to perform multiclass classification of handwritten digits. The model is trained and validated using the famous MNIST dataset. The loaded dataset is split into training, dev and test set and further processed. The preprocessed training set and dev set are fed to the model for training and evaluation. The evaluation is made based on the validation accuracy of the models. In the process of evaluation, the model is optimized and tuned using various techniques and algorithms. The final model is then trained in the large volume of the augmented dataset and tested using the test set. Thus the obtained model is ready to classify the real-world images of handwritten digits. The real-world image can be any custom image of a digit or a digit drawn into the UI of the system. However, this project does not perform multilabel classification of the image, i.e. it does not recognize the two or more digits in the same image. Nevertheless, the project can be easily extended to perform the task.

## 1.2 Problem Statement

The asymmetric nature of the digits written by different humans makes it a challenge to recognize those digits. The well-written digits are a bit easier to identify. However, some of the digits are small, poorly written and rotated in different orientations, while some are large, bold and translated to different directions. Some of those digits may also lose some of their features during the extraction process. Furthermore, some of them can have many similarities between them like 1 and 7, 5 and 6, and 3 and 8. This makes the recognition tasks even troublesome.

In order to solve the problem, an MLP model is implemented and trained using the MNIST dataset. Different optimization and tuning algorithms are implemented to get higher accuracy during validation and testing. Data augmentation techniques are used to further improve the performance of the model.

## 1.3 Objectives

The primary objectives of this project are:

* To implement an MLP model that recognizes the handwritten digits.
* To optimize and tune the model to achieve higher performance with the test accuracy over 99%.
* To implement data augmentation techniques so that the model can recognize more varied images of the handwritten digits.

## 1.4 Scope and Limitations

Handwritten Digit recognition systems can be widely used to advance the automation process and improve the interaction between man and machine in many ways, including office automation, bank cheque verification, computer vision, data entry applications and many more. This project can also be used to recognize other English alphabets, Devanagari numerals and characters using transfer learning.

The limitation of this project are:

* This system cannot recognize multiple digits in the same image or any Devanagari numerals.
* Despite high accuracy, the system may not recognize some of the digits as the system still has some error rate.
* Training the model using a larger dataset could give a better model. However, due to the hardware resources’ limitations for training the model, it could not be achieved in this project.
* The data augmentation technique used in this project did not include elastic distortion. So images with elastically distorted digits are still hard for the system to recognize with high precision.

## 1.5 Report Organization

The report is organized into seven chapters as follows:

**Chapter 1: Introduction**

The project is introduced in detail along with its objectives, scope and limitations.

**Chapter 2: Literature Review**

This chapter consists of a review of the past literature regarding the specific problem. It also consists of the various models showing the start-of-art performance in the same field.

**Chapter 3: System Analysis**

This chapter consists of an analysis of different functional and non-functional requirements of the system. It also consists of a detailed description of the dataset used in the project. Besides, feasibility analysis and process modeling are done to analyze the system’s feasibility and working mechanism.

**Chapter 4: System Design**

This chapter consists of the elaborated design of model architecture, data used, implementation process, experiments and UI used throughout the project.

**Chapter 5: Implementation**

This chapter includes the software tools, dependencies and hardware tools used to implement the system. It describes how all the steps in the process model were implemented. It describes how the experiments were carried out and what their results are. Finally, it shows the best model obtained with its performance metrics.

**Chapter 6: Testing**

This chapter consists of different levels of tests carried out to test the model built for handwritten digit recognition.

**Chapter 7: Conclusion**

This chapter includes an analysis of the results obtained during the experiments. It also includes the report’s conclusion and future works that can be carried out for expanding the project.

# : LITERATURE REVIEW

Handwritten Digit Recognition has been a topic of research in OCR since 1998, first performed by LeCun using the MNIST dataset with a 12% error rate. Since then, many researchers have worked on the problem using different approaches, being able to achieve an astonishing state-of-the-art performance. [1]

The first classifier, designed by LeCun, was a single-layer linear model with no preprocessing of the dataset. It was then followed by a large number of classifiers built using different machine learning approaches such as KNN, SVM etc. Along with that, the Neural network-based model kept on rising, outperforming all other earlier models. Over the years, LeCun himself experimented with several models with an increased number of hidden layers and hidden units, beating his previous records. In 2003, a team of researchers led by Patrice Y. Simard was the first to reduce the error rate below 1% using the Neural Network(NN) based model. They used 2-layer NN and 800 Hidden Units(HU) with elastic distortion. They used two simple cost functions, MSE and Cross entropy, each resulting in the error rate of 0.9% and 0.7%. [2] Hinton followed shortly using a 3-layer NN with 500+300 HU, Softmax function, cross-entropy loss function and weight decay to achieve an error rate of 1.53%. [3]

The next record-breaking performance using MLP was achieved by a team of researchers led by Dan Claudiu Ciresan in 2010. His team used a 6-layer NN with elastic distortion and trained the model over GPU to achieve an error rate as low as 0.35%. [4]

Besides the use of MLP, various other Neural Network architectures and different optimization techniques have been trained using the MNIST dataset to classify the handwritten digits. They, too, have shown a significant level of performance as that achieved by the MLP. Among them, CNN seems to be the most promising architecture that has been able to stay at the top of the MNIST handwritten digits classification benchmarks. [9]

The table below gives a brief preview of the work done to classify the Handwritten digits-based MNIST dataset using various algorithms and techniques. [10] [24]

Table 2.1: Benchmarking on MNIST Dataset using various techniques

|  |  |  |  |
| --- | --- | --- | --- |
| **Classifier** | **Error Rate (%)** | **Accuracy (%)** | **Reference** |
| linear classifier (1-layer NN) | 12 | 88 | LeCun et al. 1998 [1] |
| ProjectionNet: Learning Efficient On-Device Deep Networks Using Neural Projections (NN) | 5 | 95 | S. Ravi 2019 [13] |
| BinaryConnect: Training Deep Neural Networks with binary weights during propagations | 1 | 99 | Y. Bengio et al. 2015 [14] |
| 2-layer NN, 800 HU, cross-entropy [elastic distortions] | 0.7 | 99.3 | Simard et al. 2003[2] |
| Convolutional net Boosted LeNet-4, [distortions] | 0.7 | 99.3 | LeCun et al. 1998 [1] |
| Virtual SVM, deg-9 poly, 2-pixel jittered | 0.56 | 99.44 | DeCoste and Scholkopf, 2002 [6] |
| K-NN with non-linear deformation | 0.52 | 99.48 | Keysers et al. 2007 [5] |
| 6-layer NN 784-2500-2000-1500-1000-500-10 (on GPU) [elastic distortions] | 0.35 | 99.65 | Ciresan et al. 2010[4] |
| Committee of 35 Conv. net, 1-20-P-40-P-150-10 [elastic distortions] | 0.23 | 99.77 | Ciresan et al. CVPR 2012 [7] |
| Convolutional net With DropConnect | 0.21 | 99.79 | LeCun et al. ICML 2013 [8] |
| RMDL: Random Multimodel Deep Learning for Classification | 0.18 | 99.82 | K. Kowsari et al. 2018 [11] |
| Branching/Merging CNN + Homogeneous Filter Capsules | 0.16 | 99.84 | A. Byerly et al. 2020 [12] |

# : SYSTEM ANALYSIS

## 3.1 Requirement Analysis

Requirement Analysis is the process of identifying the needs of a particular system. It encompasses all the seen and unseen conditions to be met by the system under development. It eliminates any conflict in the requirements and sets the scope of the project. This process is carried out by analyzing two types of requirements:

### 3.1.1 Functional Requirements

Functional requirements help capture the system’s intended behavior by describing what the system should and should not do.

This system’s primary requirement is that it should be able to classify a handwritten digit with high precision. For this to happen, first, the model should be appropriately trained using the MNIST dataset. Before that, the dataset should be preprocessed to meet the model’s needs. Once the well-trained model is obtained, it can be saved and loaded into a UI that allows the user to upload or draw a digit. Thus, uploaded or drawn images of a digit can be classified using the loaded model. The system allows the user to perform all these tasks meeting its primary objective.

Thus the functional requirements of this system are as follows:

* Users should be able to load the dataset.
* Users should be able to preprocess the dataset.
* Users should be able to train the model.
* Users should be able to save the model.
* Users should be able to load the model.
* Users should be able to upload or draw a digit.
* Users should be able to classify the digit.

The following Use Case Diagram describes the system’s significant actions and interaction between actors (users) and the system.

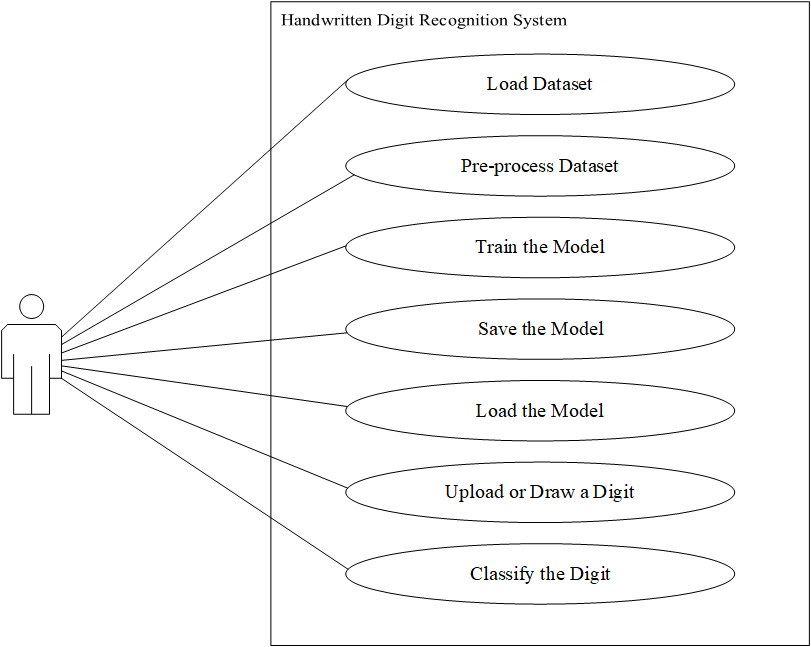


Figure 3.1: Use Case diagram for Handwritten Digit Recognition System

The following use case description tables elaborates the use case diagram above.

Table 3.1: Use Case description for Load Dataset

|  |  |
| --- | --- |
| Use-Case Identifier | UC1- Load Datasets |
| Primary Actor | User |
| Secondary Actor | None |
| Description | It loads the required size of the dataset from the local directory. |
| Pre-condition | All the required dependencies should have been imported. Data files should be in the local directory or the computer should be connected to the Internet. |
| Post-condition | Shuffled training set and test set data of required size loaded. |
| Failure | Failed to load data due to failure to find the data file or failure to connect to the Internet |

Table 3.2: Use Case description for Preprocess Datasets

|  |  |
| --- | --- |
| Use-Case Identifier | UC2- Preprocess Datasets |
| Primary Actor | User |
| Secondary Actor | None |
| Description | It splits the Training set to train and dev sets, flatten and normalize the train, dev and test set images, and one-hot encode the train, dev and test set labels. |
| Pre-condition | Training and Test dataset should have been loaded. |
| Post-condition | Train, dev and test dataset prepared for training. |
| Failure | Failed to prepare the dataset for training |

Table 3.3: Use Case description for Train Model

|  |  |
| --- | --- |
| Use-Case Identifier | UC3- Train Model |
| Primary Actor | User |
| Secondary Actor | None |
| Description | It trains the MLP model with the dataset using various algorithms. |
| Pre-condition | The dataset should have been prepared to fit the model architecture. Parameters and Hyperparameters should have been initialized. |
| Post-condition | Returns trained model with other training information |
| Failure | Memory size exceeded. |

Table 3.4: Use Case Description for Save Model

|  |  |
| --- | --- |
| Use-Case Identifier | UC4- Save Model |
| Primary Actor | User |
| Secondary Actor | None |
| Description | Saves the model obtained after the training. |
| Pre-condition | The model should have been trained. |
| Post-condition | Saves the model to a binary file |
| Failure | The file path does not exist. |

Table 3.5: Use Case Description for Load Model

|  |  |
| --- | --- |
| Use-Case Identifier | UC5- Load Model |
| Primary Actor | User |
| Secondary Actor | None |
| Description | It loads the saved model from a binary file. |
| Pre-condition | The model should have been saved in a binary file. |
| Post-condition | Model parameters and other training information loaded. |
| Failure | File not found. |

Table 3.6: Use Case description for Upload or Draw a Digit

|  |  |
| --- | --- |
| Use-Case Identifier | UC6- Upload or Draw a Digit |
| Primary Actor | User |
| Secondary Actor | None |
| Description | It uploads an image of a digit from a local directory or draws the digit in the UI panel. |
| Pre-condition | The image should have been placed in the local directory.  The UI for drawing should have been created. |
| Post-condition | Image of a digit obtained. |
| Failure | File not found/failed to capture the drawn image. |

Table 3.7: Use Case description for Classify the digit

|  |  |
| --- | --- |
| Use-Case Identifier | UC7- Classify the digit |
| Primary Actor | User |
| Secondary Actor | None |
| Description | Classify the digit in the image into one of 10 classes |
| Pre-condition | The image should have been uploaded or drawn. The model should have been loaded. Furthermore, the image should have been processed to fit the model’s needs. |
| Post-condition | Class of the image |
| Failure | Misclassification of the image. |

### 3.1.2 Non-functional Requirements

Non-functional requirements elaborate on the system’s e performance characteristic and define the constraints on how the system will do so. These include reliability, performance, service availability, responsiveness, throughput and security.

**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.

## 3.2 Feasibility Analysis

Feasibility analysis is the process of identifying the viability of the project through different perspectives. It is the analysis of different peripheral components of the project that may directly hamper the project’s execution. [24] Following are the feasibility concerned with this project.

### 3.2.1 Data Availability

The dataset can be abundantly found across the web, among which Lecun’s website is the primary source. The dataset available is 70000 images and their corresponding labels. For better results, the model will require as much data as possible. However, no massive volume of the dataset is available on the web. So a larger volume of the dataset can be obtained by augmenting the available images.

### 3.2.2 Literature Availability

A large number of researchers have studied this problem. So, a vast repository of literature regarding different approaches to solving this problem can be found in different web archives and databases. Researchers have addressed this problem with a huge range of approaches. So, these works of literature have lots of insightful information regarding the problem.

### 3.2.3 Availability and Cost of Computational Resources

Personal computers these days have grown really powerful with high-performance CPUs and massive memories. So they are sufficient to process the original dataset during the training. However, when data augmentation is added to generate the images in large volumes, the memory still seems insufficient in the machine available. Google Colab Instance and Kaggle Kernel seemed to be great alternatives to overcome this issue with no cost. Nevertheless, they too have their limitations based on memory and offline usage. So not enough resources are available for free if the training has to be done in a larger dataset volume. Paid cloud resources are available in AWS and Azure platforms. Despite being paid, they provide students access to the platform with some credits sufficient for the project completion.

### 3.2.4 Operational Feasibility

Though implemented from scratch, the abstraction of complex functionalities and automation of some tasks has made the system very easy and flexible to use.

## 3.3 Data Description

The MNIST dataset is a large dataset of handwritten digits’ images that is extensively used as a toy database for various image processing systems and machine learning projects in academics and industries. It was initially adopted from a larger dataset called NIST by Chris Burges and Corinna Cortes using bounding-box normalization and centering. Yann Lecun’s website is the primary source of this dataset. His version of the dataset is size-normalized and centered by the center of mass in a larger window.

The data is stored in the idx file format which has to be extracted using a custom program.

* train-images-idx3-ubyte.gz: consists of training set images (9.453 MB)
* train-labels-idx1-ubyte.gz: consists of training set labels (0.028 MB)
* t10k-images-idx3-ubyte.gz: consists of test set images (1.572 MB)
* t10k-labels-idx1-ubyte.gz: consists of test set labels (0.004 MB)

Once extracted, the training set consists of 60000 examples, and the test set consists of 10000 examples. Each set has gray-scale images of size 28x28 and their corresponding labels with 10 classes for digits from 0 to 9. The background of the image is black, and the digit itself is white in color. [10]

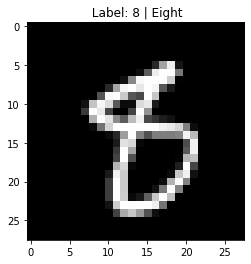


Figure 3.2: Digit 8 in the MNIST Dataset

The dataset contains a slightly varied number of images in each class. This variation can be seen in the visualization below:

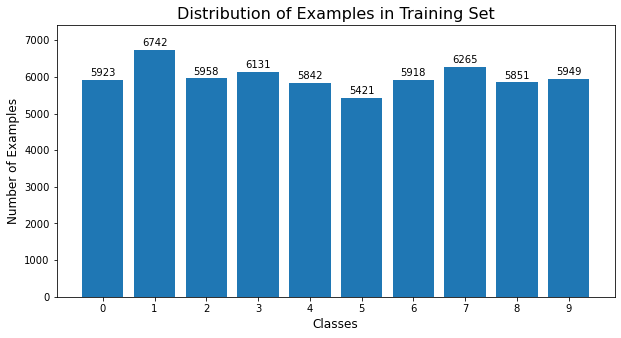


Figure 3.3: Data distribution in Training set

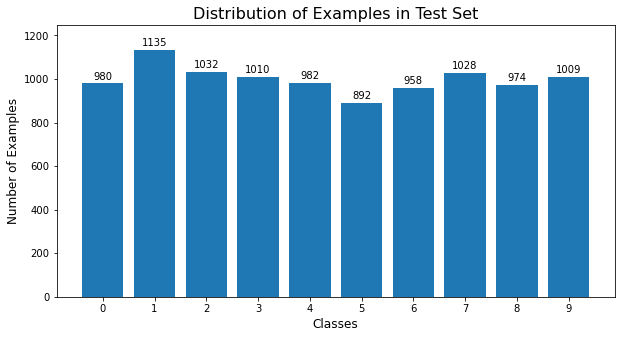


Figure 3.4: Data distribution in Test set

The dataset seems slightly imbalanced; however, it is not a significant issue as the imbalance is not too severe. [26]

## 3.4 Process Modeling

Process modeling is a technique of graphically representing the flow and structure of data through a system's processes that capture and transform inputs into outputs. It is generally represented by different levels of Data Flow Diagrams (DFDs).

### 3.4.1 Context Diagram

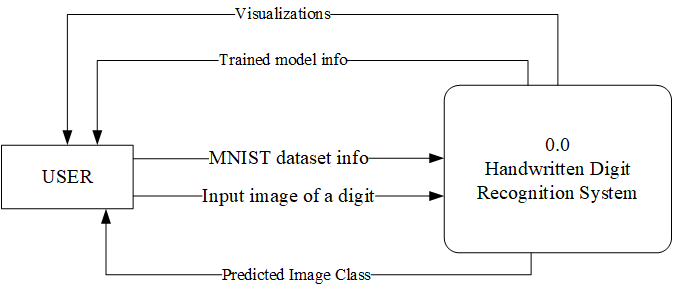


Figure 3.5: Context Diagram of Handwritten Digit Recognition System

The context diagram consists of one external entity, the user, along with the process block 0.0 representing the Handwritten Digit Recognition System. The external entity interacts with the system in multiple different ways as shown in the diagram above.

### 3.4.2 Level 0 DFD

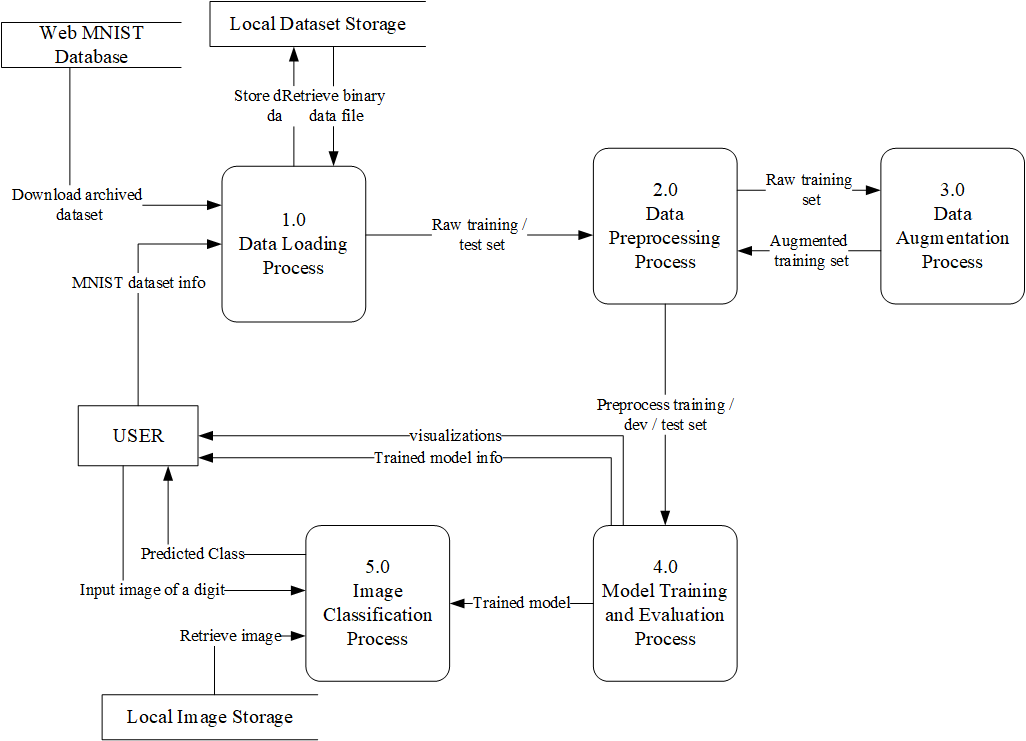


Figure 3.6: Level 0 DFD of Handwritten Digit Recognition System

The level 0 DFD consists of an external entity, the user, and the 5 process blocks representing the handwritten digit recognition system’s internal working and 3 data stores for the MNIST dataset and the input image. Process 1.0 loads the dataset from the local storage or the web database. Process 2.0 preprocesses the loaded dataset making it ready for training. Process 3.0 augments the raw training set obtained from process 2.0 and sends it back to process 2.0 for further processing. Process 4.0 trains and evaluates the model using the processed dataset obtained from process 2.0 and gives out the trained model. It also gives out some visualizations and trained model information. Process 5.0 predicts the class of the input digit using the model obtained from process 4.0. The input image may be uploaded from the local storage or drawn by the user in the UI.

### 3.4.3 Level 1 DFD

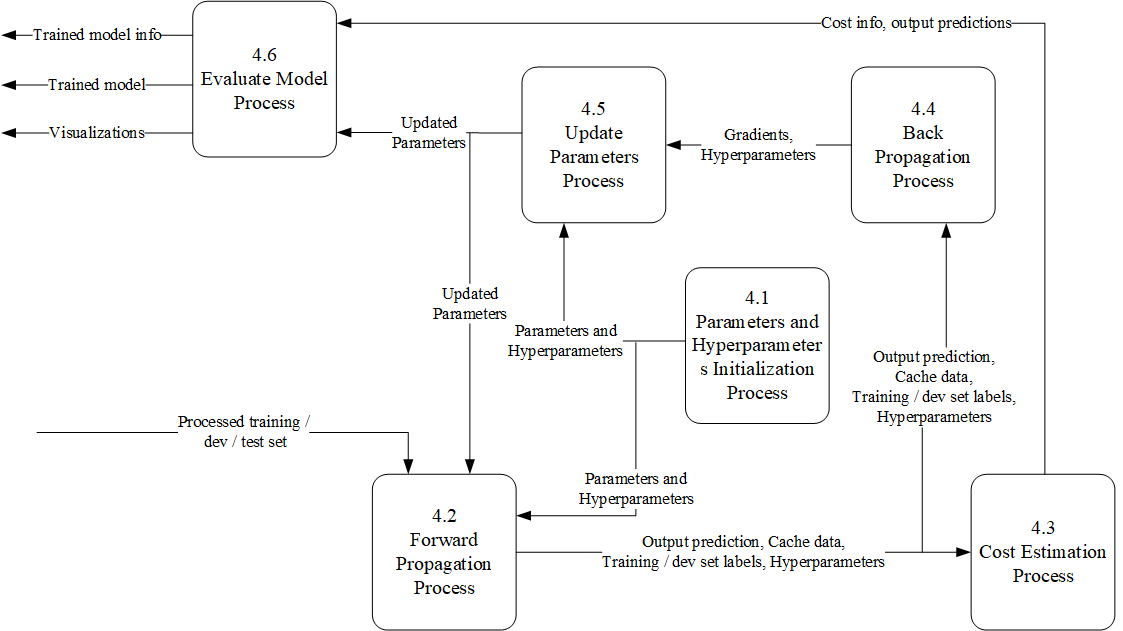


Figure 3.7: Level 1 DFD of Handwritten Digit Recognition System

The level 1 DFD for process 4.0 consists of 6 process blocks representing the internal working of process 4.0. Process 4.1 initializes the parameters and hyperparameters needed to train the model. Process 4.2 performs the forward propagation using the training set and gives out the predicted output of the training set with some cache data. Process 4.3 computes the cost of the prediction. Process 4.4 computes the gradients of the model. Process 4.5 updates the parameters using the gradients and the previous set of parameters. Process 4.6 evaluates the trained model using the dev set and test set. It gives out useful visualization, trained model and model-related information.

# : SYSTEM DESIGN

## 4.1 Architecture Design

The architecture used in this project is a Multilayer Perceptron (MLP). It is a kind of Feedforward Artificial Neural Network composed of multiple layers of neurons.

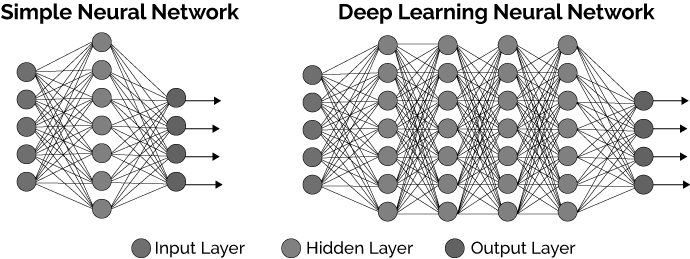


Figure 4.1: Types of Neural Network based on number of hidden layers

Src: [xenonstack.com](https://www.xenonstack.com/blog/log-analytics-deep-machine-learning/)

Neurons are the simple computational units that build up the entire network. Each neuron consists of some input and outputs that have trainable parameters called weights and biases. Weights are often initialized randomly and biases with zeros. The entire training process is to adjust these parameters.

### 4.1.1 Layers

The MLP consists of at least 3 layers - input, hidden and output. The input layer is often not taken into account. The hidden layer can be of any number, and the output layer is always one. So an MLP with 2 layers means having 1 input, 1 hidden and 1 output layer. Each layer can have one or more neurons/ units. Such a simple architecture is called Shallow Neural Network. Deep Neural Networks have more than 2 hidden layers.

The model in this project consists of 784 units in the input layer and 10 units in the output layer. The 784 input units represent the 784 pixels of the input image, and the 10 output units represent 10 output classes. The number of hidden layers and their units may vary during the tuning process.

### 4.1.2 Activation function

An activation function is a simple mathematical function that maps the summed weighted input to the output of a neuron. It provides a threshold at which the neuron is activated.

Two different activation functions would be used in the model in this project. They are:

**ReLU activation function**: ReLU is implemented in all the hidden layers of the network. It is one of the most used non-linear activation functions. The function and its derivative both are monotonic. Its output is equal to its input if the input is greater than 0. ReLU is defined as

And its derivative as

**Softmax activation function**: The Softmax function is implemented in the output layer of the network. It calculates the probabilities of each class over all possible classes. The calculated probabilities can be used to determine the target class for the given inputs. The total sum of all the outputs is always equal to 1. [15]. It is defined as

### 4.1.3 Cost function

It is the function that is used to measure the performance of a model. The cost function used in this model is the Softmax Cross-Entropy Loss Function.

Softmax Cross-Entropy Loss function: Also known as categorical cross-entropy function, it is used to measure the performance of multiclass classification models. It is a form of maximum likelihood estimation in statistics [16]. It is defined as:

The figure below shows the layer-based block diagram of an MLP architecture using all these above discussed components.

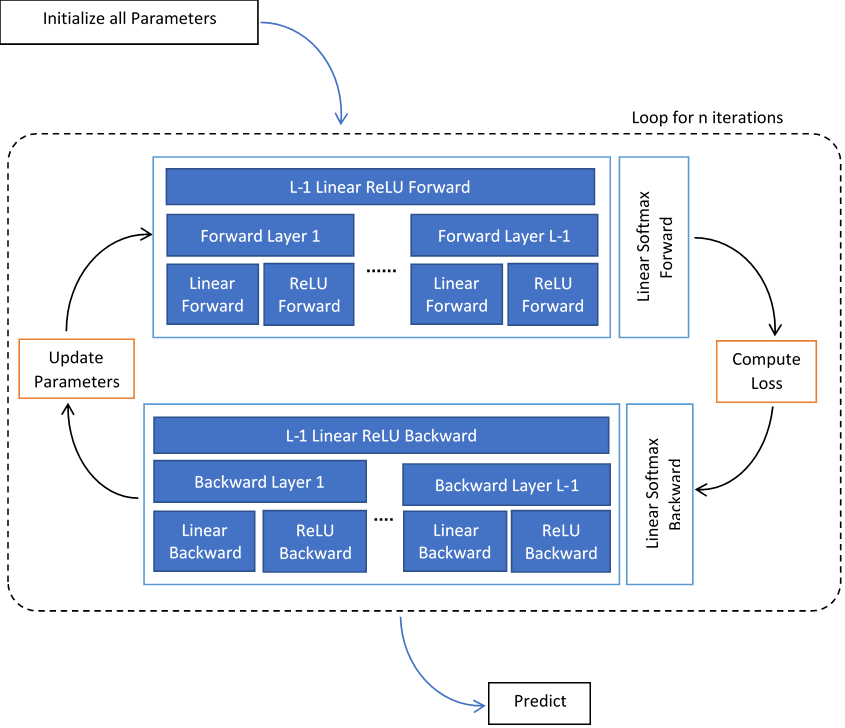


Figure 4.2: Block Diagram of MLP architecture used in the project

Each layer in the figure above contains several neural units. The parameters are initialized, and forward propagation of the network is carried out. The output of the forward propagation is used to calculate the loss. The loss thus computed is propagated backward to the neural units to adjust the parameters with a technique called Back Propagation. The parameters are then updated, and the process continues until the training completes. Once a trained model is obtained, the prediction of a new input image can be made.

## 4.2 Data Design

The loaded dataset consists of 3D NumPy arrays of training and test images and 2D NumPy arrays of training and test labels. The entire process needs 3 sets of data, namely training, development and test set. The training set is used to train the model, while the dev set is used to evaluate the model during the training. The test set is used to test the final model at the end. Thus the original dataset should be split into these three sets of data.

The network has 784 input nodes; however, the images are of shape 28x28. So each image should be preprocessed into a 1D vector of size 784. On the other hand, there are 10 classes to be represented by 10 output nodes. The output labels, however, are single scalar values. So each label has to be preprocessed into a 1D vector of size 10.

Thus the input should be an ndarray of the shape (784, m), and the output labels should be an ndarray of the shape (10, m), where m is the number of examples. The following table shows the data sizes before and after processing.

## 4.3 Process Design (Methodology Flow Chart / Process model)

The process model of this project consists of the following steps to achieve the project’s objective:

Model Tuning

Data Collection

Data Preprocessing

Baseline Model

Model Optimization

Model Deployment

Figure 4.3: Process model used in the project

### 4.3.1 Data Collection

The data will be manually or automatically collected from the source. The dataset will be loaded into the system in NumPy array format.

### 4.3.2 Data Preparation

The data obtained may not always be in the desired format. A little preprocessing needs to be done to the data before feeding it to the training network.

In the context of this project, the input image should be flattened and normalized. The output labels should be one-hot encoded. Data augmentation can also be used to increase the dataset volume.

### 4.3.3 Baseline Model

The baseline model is a fundamental model using a simple algorithm to accomplish the predefined task. This can be described as the simplest useful model development level and can be compared to a prototype in the software development process. Further complexity is added in the process of optimizing and tuning the baseline model.

In this project, the baseline model is created using **random initialization** and implements **Batch Gradient Descent** as the model optimizer. No regularization and data augmentation techniques are implemented.

The hyperparameters used are learning rate = 0.001 and epoch size = 25. These hyperparameters are taken arbitrarily during the training with no tuning. The number of hidden layers and the number of nodes in each hidden layer are chosen arbitrarily during the training.

**Random Initialization**: It is the process of randomly initializing the weights of the neural network with some small numbers. It is used to break symmetry and make sure different hidden units can learn different features. However, the biases can be zero-initialized.

**Batch Gradient Descent**: Batch Gradient Descent, also known as Gradient Descent, is a simple iterative optimization algorithm in Machine Learning. Its main goal is to minimize the loss function by computing its gradient and moving slowly towards the minimum. It uses the backpropagation algorithm to compute the backward gradient using the computed loss and update the parameter. Its major variation with other similar gradient descent algorithms is that the parameters are updated only when the entire training set is passed forward and backward through the network, i.e., the network should see the entire training set before updating its parameters. [17] The following algorithm describes the process of batch gradient descent.

X = input data

Y = output labels

parameters = initialized parameters based on network layers

for i in range(0, epoch):

a, caches = Forward propagation using X and parameters

cost = compute cost using a and Y

grads = back propagate using a, caches and parameters

parameters = update parameters using grads and old parameters

### 4.3.4 Model Optimization

Once the baseline model is up and running, it is continuously evaluated and optimized based on specific evaluation metrics. **He-initialization** will be implemented and evaluated. **Minibatch Gradient Descent** and **Adam** will be implemented and evaluated. Besides regularization techniques like **L2 regularization**, **dropout** and **data augmentation** will also be implemented and evaluated.

**He-Initialization**: He-Initialization is similar to random initialization, except that it uses a scaling factor for the weights 𝑊[𝑙]. The weight of the layer is scaled by the square root of 2 divided by the layer dimension of layer.

Weights = random small value \*

**Minibatch Gradient Descent**: Minibatch Gradient descent is the variant of the Stochastic Gradient Descent that adapts the benefits of both BGD and SGD. Creating batches of input MGD vectorizes the training processing while maintaining computational efficiency and memory consumption. So it is very suitable for large datasets.

In MGD, each epoch consists of several iterations. Each iteration processes a batch of input. So the number of iterations is based on the number of mini-batches. The size of the minibatch is fixed. It is a hyperparameter and has to be tuned to obtain better performance. [16] [18] The following algorithm describes the process of minibatch gradient descent.

X = input data

Y = output labels

parameters = initialized parameters based on network layers

for i in range(0, epoch):

obtain mini-batches with batch size S from X

for minibatch in mini-batches:

x, y = obtain input and output data of each minibatch

a, caches = Forward propagation using x and parameters

cost = compute cost using a and y

grads = backpropagate using a, caches and parameters

parameters = update parameters using grads and old parameters

**Adam**: Adam optimization is an extension to Stochastic gradient decent and can be used in place of MGD and SGD to update network weights more efficiently. It is one of the most effective and popular optimization algorithms used for training neural networks. It combines the concepts from RMSProp and Momentum.

Like momentum, it calculates an exponentially weighted average of past gradients and stores it in variables 𝑣 (before bias correction) and (with bias correction). And like RMSProp, it calculates an exponentially weighted average of the squares of the past gradients and stores it in variables 𝑠 (before bias correction) and (with bias correction). [16] [19] [20]

It updates parameters in a direction based on combining information from and above. The update rule is:

Where,

* t counts the number of steps taken by Adam
* L is the number of layers
* 𝛽1 and 𝛽2 are hyperparameters that control the two exponentially weighted averages.
* 𝛼 is the learning rate
* 𝜀 is a tiny number used to avoid division by zero

**L2 Regularization**: L2 regularization is a standard way to avoid overfitting in a neural network. It penalizes the square values of the weights and forces them towards zero. But, it never lets the weight be equal to zero. An additional hyperparameter regularization rate is added that has to be tuned to achieve better performance. [16] [21]

On using L2 regularization, a regularization term is added to the loss function and the backpropagation. The loss function along with L2 regularization is:

The gradient of the weight with L2 regularization during backpropagation is:

**Dropout**: Dropout is a popular regularization technique used mostly in deep learning. The simplest explanation for dropout is it randomly shuts down some neurons of particular hidden layers in each iteration. The same neurons are shut down during the forward and backward propagation within an iteration. At each iteration, each neuron of a layer is shut down with probability 1 − 𝑘𝑒𝑒𝑝\_𝑝𝑟𝑜𝑏 or kept with probability 𝑘𝑒𝑒𝑝\_𝑝𝑟𝑜𝑏. The dropped-out neurons do not contribute to the training in both the forward and backward propagations of the iteration. keep\_prob is another hyperparameter that needs to be tuned. [16] [22]

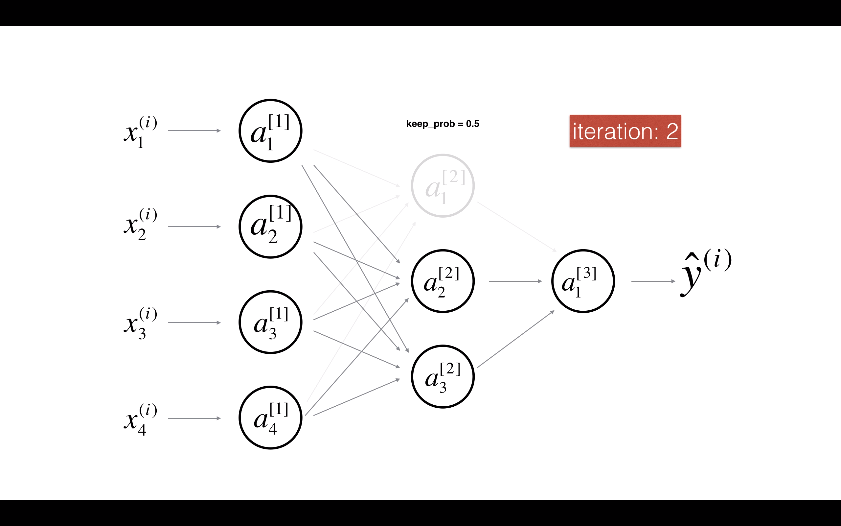


Figure 4.4: Neural Network showing Dropout of individual neural unit

When some neurons are shut down, the model is actually modified. The idea behind dropout is that a different model that uses only a subset of the actual neurons is trained at each iteration. With dropout, the neurons become less sensitive to the activation of other neurons because the other neuron might be shut down at any time. The dropout technique used in this project is known as the inverted dropout.

**Data Augmentation**: Data Augmentation is the process of transforming the existing data into slightly varied forms using techniques like geometric and elastic distortions. Augmentation can vastly increase the data volume that is similar yet different by different factors. Data Augmentation can be done in two different ways; Offline Augmentation and Online Augmentation. This project implements the Online Data Augmentation technique.

Offline Augmentation: In this process, the data is preprocessed, and the required volume of data is prepared before feeding them to the network. The network sees the same data for a certain number of epochs during the training.

Online Data Augmentation: In this process, the original data is fed to the network, and during the training, the augmented data are generated on the fly. In each epoch, the network sees slightly different data, or say, the network probably never sees the same data twice. So, the network generalizes better using this method than that of offline augmentation.

### 4.3.5 Model Tuning

The optimized model has various hyperparameters that directly and indirectly influence the result. These hyperparameters are:

* Learning rate
* Minibatch size
* Number of hidden layers
* Number of nodes in each hidden layers
* Number of epoch to be trained
* Lambda
* Keep\_probs (dropout parameter)
* Beta1
* Beta2
* epsilon

Once suitable algorithms and optimization techniques are evaluated and selected, the final model should be tuned for the best possible hyperparameter combinations.

Various techniques are used for tuning the hyperparameters. Among them, this project implements random search techniques to search for the best possible hypermeter combination.

Random Search: Random search is a search technique used when a range of possibly suitable values for the hyperparameters are known instead of specific values. So a search is done by selecting the hyperparameter values at random within the range.

This project implements Coarse-to-Fine search, a variant of random search. In Coarse-to-Fine search, the values are searched first in a broader range, and the range is gradually decreased in each search carried out. [16] [23]

### 4.3.6 Model Deployment:

Once the model is optimized and the hyperparameters tuned, the final model is trained with a large dataset and saved. A GUI interface is created where the model is loaded. The loaded model is used to classify the uploaded or drawn image of a handwritten digit.

## 4.4 Experiments Design

The project implements various algorithms and techniques during the model optimization process. Besides, it also has a number of hyperparameters to be selected in the process of model tuning. So, some experiments would be carried to facilitate the process and obtain the best possible combination of algorithms and hyperparameters to build an efficient model. All the experiments would be performed using only 50% of the original dataset to limit training time. Validation accuracy would be used as an evaluation metric to evaluate the algorithms and hyperparameters.

The project would have 4 experiments. Experiments 1, 2 and 3 will be used for model optimization, and 4 would be used for hyperparameter tuning.

During the model optimization process best initialization technique, optimization algorithm and regularization technique would be selected. A random search would be carried out during the tuning process to find the best combination of hyperparameters. Tuning all the hyperparameters would be carried out simultaneously, as tuning them separately might result in some inconsistencies. Thus obtained best combination would be used to train as a final model.

## 4.5 Interface Design

Once the best possible model is obtained, it is used to predict the real-world images using a user interface. The UI should contain the following features:

* Upload image button for uploading an image of a digit
* A canvas to draw a digit on the interface itself
* Predict button to classify the image
* Ability to predict the second guess
* Clear button to clear the canvas

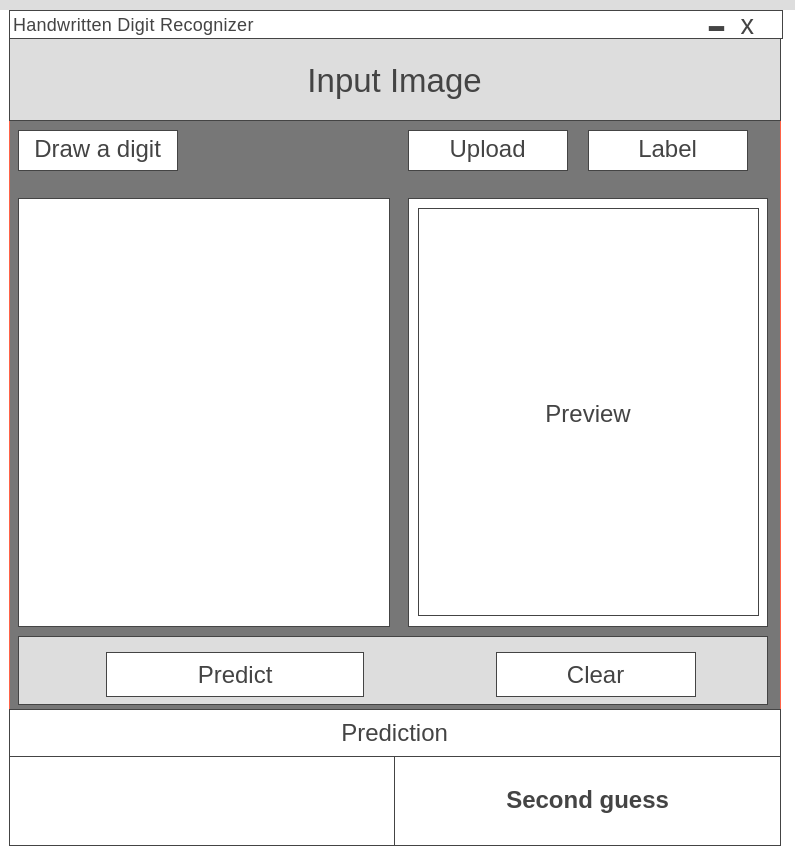


Figure 4.5: Wireframe of the Handwritten Digit Recognition System

# : 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 (v1.9.12)**: 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 (v6.1.5)**: Jupyter notebook is an interactive Ipython interface for programming. It was used for the majority of the development task and code experimentation.

**Spyder (v4.1.5)**: Spyder cross-platform IDE for developing Python programs. It was used as additional support to the Jupyter notebook for some development and code evaluation tasks.

**Python (v3.8.5)**: Python is a general-purpose language largely used for developing Machine Learning Projects. The entire project is written using this language.

**Git (v2.17.1) 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 Tools:

Design tools are the software packages used to design necessary diagrams and various other project artifacts such as reports, proposals and presentations. The following software packages were used for this purpose:

**MS Visio 16**: MS Visio is a diagramming tool. It was used to create DFDs, Use Case and interface design

**MS Word 16**: MS Word is a word processing tool. It was used for creating project artifacts such as abstract, proposal, and reports

**MS PowerPoint 16**: MS PowerPoint is a presentation tool. It was used for creating various powerpoint presentations throughout the project.

### 5.1.3 Dependencies

Dependencies are various python packages that were used as core components while developing the project. They come under two categories:

**Standard Modules**: These modules are inbuilt to the python package and come preloaded with the python software. Following standard modules were used in this project for the described purposes.

**os module**: Used to interact with the operating platform for file operations

**struct module**: Used to unpack the binary data file

**urllib module**: Used to download data files from the web

**gzip module**: Used to unzip gzip files

**time module**: Used to get system time

**Tkinter module**: Used to create a GUI interface

**gc module**: Used to free system memory

**Third-party packages:** These modules are developed by third-party vendors or communities and have to be downloaded separately in order to be used. Following third party packages were used in this project:

**NumPy package(v1.19.1)**: Used to carry out all mathematical operations

**matplotlib package(v3.3.1)**: Used to generate visualization

**scipy package(v1.5.2)**: Used in the image augmentation process for rotating, blurring and shifting images

**pickle package(v0.7.5)**: Used to save and load trained model and augmented datasets

**PIL package(v8.0.1)**: Used to load, save and process image during prediction

**pyscreenshot(v2.2)**: Used to capture the image drawn in UI

## 5.2 System Configuration

Three different systems were used for the development and evaluation of the system. The development and selection of the algorithms were carried out in the local machine. However, other time-consuming tasks like hyperparameter tuning and training the final model had to be performed in the cloud due to the memory limitation of the local machine and frequent electricity outages. Azure VMs were used as cloud machines

Table 5.1: System Configuration for the project development

|  |  |  |  |
| --- | --- | --- | --- |
| Machine Instance | Local Machine | Azure Virtual Machines | |
| System | Linux 64-bit | Linux 64-bit | Linux 64-bit |
| Distribution | Ubuntu 18.04 bionic | Ubuntu 18.04 bionic | Ubuntu 18.04 bionic |
| Processor | Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz | AMD EPYC 7452 32-Core Processor | Intel(R) Xeon(R) CPU E5-2673 v4 @ 2.30GHz |
| Physical / Total Cores | 4/8 | 2/4 | 2/4 |
| Main Memory | 16 GB | 32 GB | 32 GB |
| Disk Space | 256 GB | 128 GB | 128 GB |

## 5.3 Methodology

### 5.3.1 Data Collection

The project implements two modes of data collection; manual and auto.

In manual mode, a user needs to download the data files from Lecun’s website manually. If the source differs, the file format should be the same for further processing. The user should then unzip the downloaded files and store the final binary files to the local storage before loading the dataset into the system.

In auto mode, a python script performs all the manual tasks and loads the dataset into the system, making the data collection process consistent and straightforward. The program only downloads the files from the source if they are not found in the local storage. The loaded dataset consists of a training set and a test set, each containing input images and output labels in a NumPy array format.

Table 5.2: Loaded datasets and their size

|  |  |
| --- | --- |
| **Dataset** | **Dataset Size** |
| Training Set Images | (60000, 28, 28) |
| Training Set Labels | (60000, 1) |
| Test Set Images | (10000, 28, 28) |
| Test Set Labels | (10000, 1) |

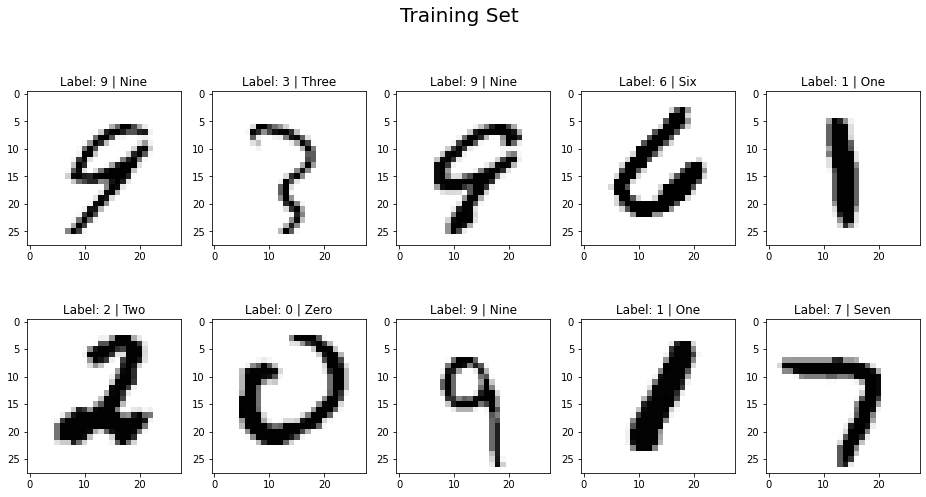
Below is the image of some sample images in the MNIST dataset.

Figure 5.1: Sample MNIST images with their corresponding labels

### 5.3.2 Data Preparation

The loaded data was converted into the desired format using the following approaches

**Train-dev split of the training set**: The training set with 60000 examples was split into a training set and a development(dev) set with 50000 and 10000 examples respectively, making 3 sets of data. The split training set is used to train the model, and the dev set is used to evaluate the model performance during the model development and evaluation process. The test set is used to evaluate the final model.

Table 5.3: Split dataset and their size

|  |  |
| --- | --- |
| **Dataset** | **Dataset Size** |
| Training Set Images | (50000, 28, 28) |
| Training Set Labels | (50000, 1) |
| Development Set Images | (10000, 28, 28) |
| Development Set Labels | (10000, 1) |

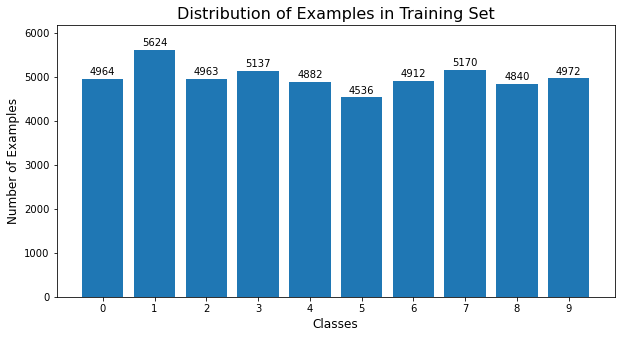
The distribution of the data in the three datasets is as follows:

Figure 5.2: Data distribution of the split training set

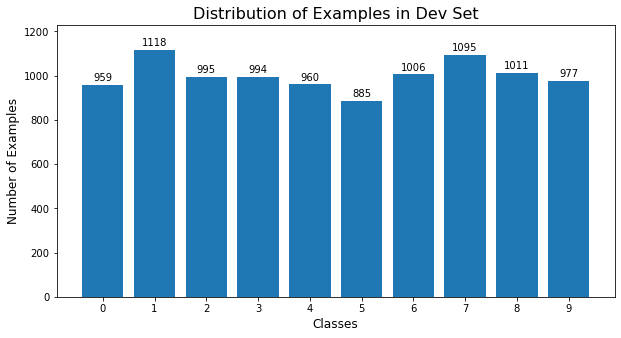


Figure 5.3: Data distribution of the split development set

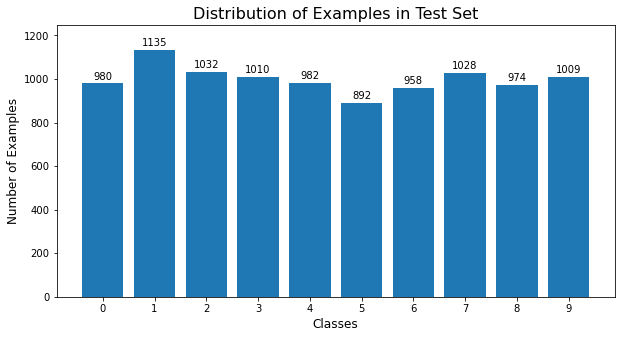


Figure 5.4: Data distribution of the split test set

**Flattening and Normalizing Input Image**: Each image in all 3 sets was first flattened into a 1D vector of shape (784, 1). Then the image pixels were normalized into the range of 0 and 1 to lower the computation cost.

Table 5.4: Size comparison of images in the dataset before and after preprocessing

|  |  |  |
| --- | --- | --- |
| **Data** | **Before Processing** | **After Processing** |
| Training Set Images | (50000, 28, 28) | (784, 50000) |
| Dev Set Images | (10000, 28, 28) | (784, 10000) |
| Test Set Images | (10000, 28, 28) | (784, 10000) |

**Encoding Output labels**: The output labels were one-hot encoded. Each label was represented in the form of 10 classes.

Table 5.5: Size comparison of labels in the dataset before and after preprocessing

|  |  |  |
| --- | --- | --- |
| **Data** | **Before Processing** | **After Processing** |
| Training Set Labels | (50000, 1) | (10, 50000) |
| Dev Set Labels | (10000, 1) | (10, 10000) |
| Test Set Labels | (10000, 1) | (10, 10000) |

Encoded labels example:

[7 2 1 0 4 1 4 9 5 9] - original labels

[[0 0 0 1 0 0 0 0 0 0] - encoded labels

[0 0 1 0 0 1 0 0 0 0]

[0 1 0 0 0 0 0 0 0 0]

[0 0 0 0 0 0 0 0 0 0]

[0 0 0 0 1 0 1 0 0 0]

[0 0 0 0 0 0 0 0 1 0]

[0 0 0 0 0 0 0 0 0 0]

[1 0 0 0 0 0 0 0 0 0]

[0 0 0 0 0 0 0 0 0 0]

[0 0 0 0 0 0 0 1 0 1]]

**Data augmentation**: Both online and offline Data augmentation were implemented, but only the online data augmentation was used to train the final model. Following transformations were carried out using spline interpolation and fill mode parameter as “nearest”:

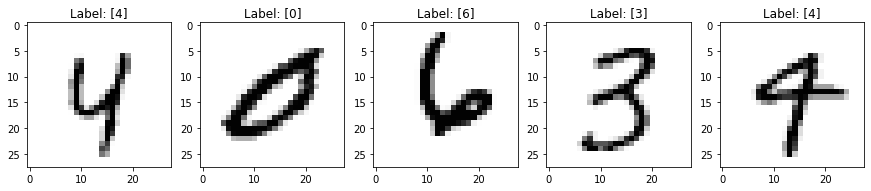


Figure 5.5: Images before applying any transformation

**Rotation**: Images were rotated randomly with the angle’s value in the range of +60 to -60 degrees.

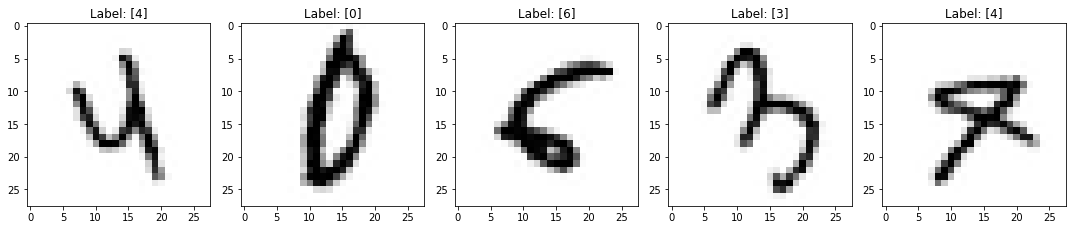


Figure 5.6: Images after Rotation

**Shifting**: Images were randomly shifted along x and y axes. The shift parameter was determined randomly within the range +7 to -7.

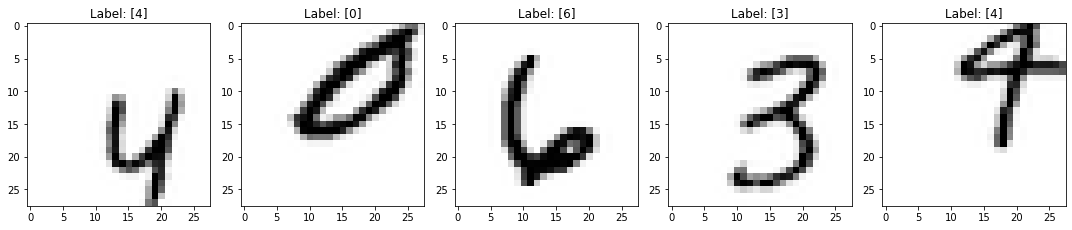


Figure 5.7: Images after Shifting

**Cropping and Padding**: The images were randomly cropped across x and y axes and padded with sufficient pixels to get the images of size 28x28 as required for training.

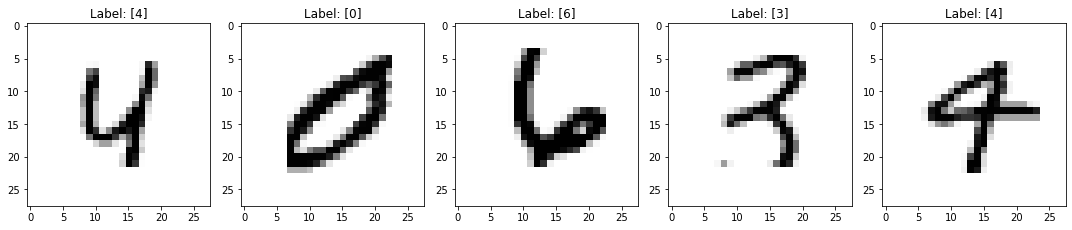


Figure 5.8: Images after Cropping and Padding

**Blurring**: The images were randomly blurred using 5 random filters. The filters used were Gaussian, Uniform, Maximum, Minimum and Median. The Gaussian filter used sigma in between the range 0-2.5. The Maximum and Minimum filters used the size parameter in the range 0-4 and the Median and Uniform filters used the size parameter in the range 1-6.

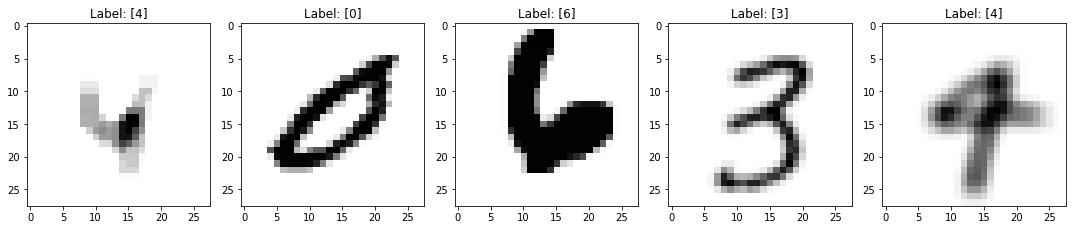


Figure 5.9: Images after applying random blurring filters

**Zooming**: The images were randomly cropped across x and y, and instead of padding, they were rescaled to 28x28 size to give zooming out effect.

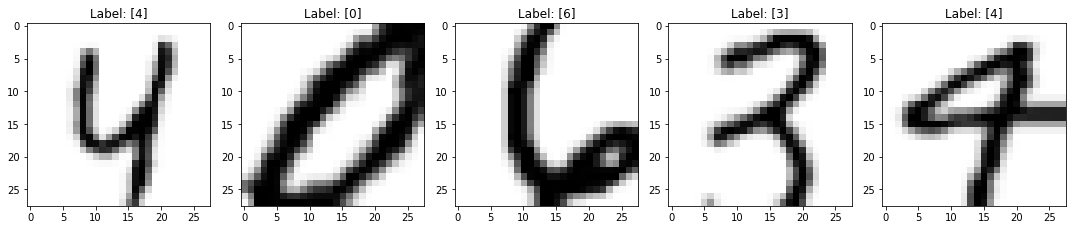


Figure 5.10: Images after Zooming

### 5.3.3 Baseline Model

A simple Softmax classifier, with 2 hidden layers having 32 and 16 hidden units respectively, was created implementing Batch Gradient Descent. The initial parameters were randomly initialized and fed into the network for training. The model was trained for 10,000 epochs with a learning rate of 0.1. Only 50% of the original dataset was preprocessed and fed into the model for training.

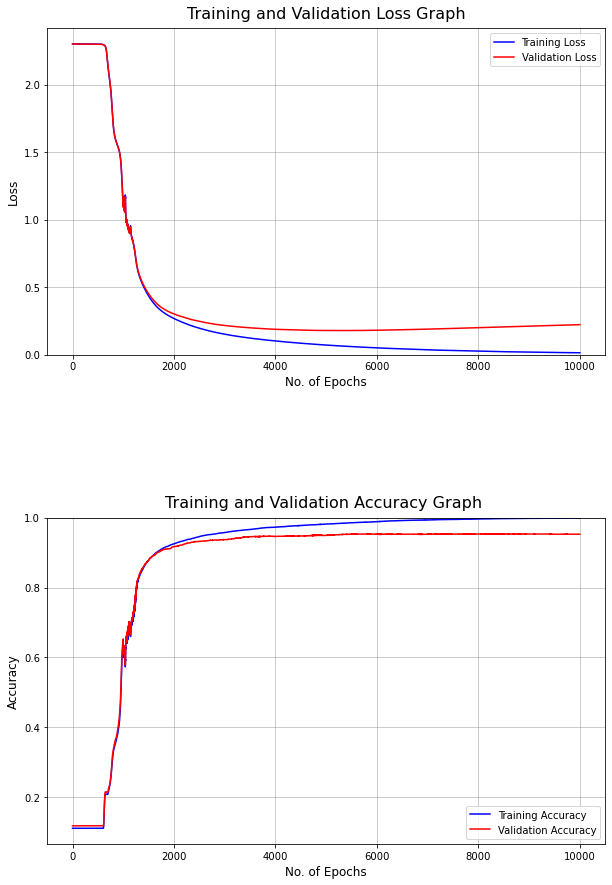
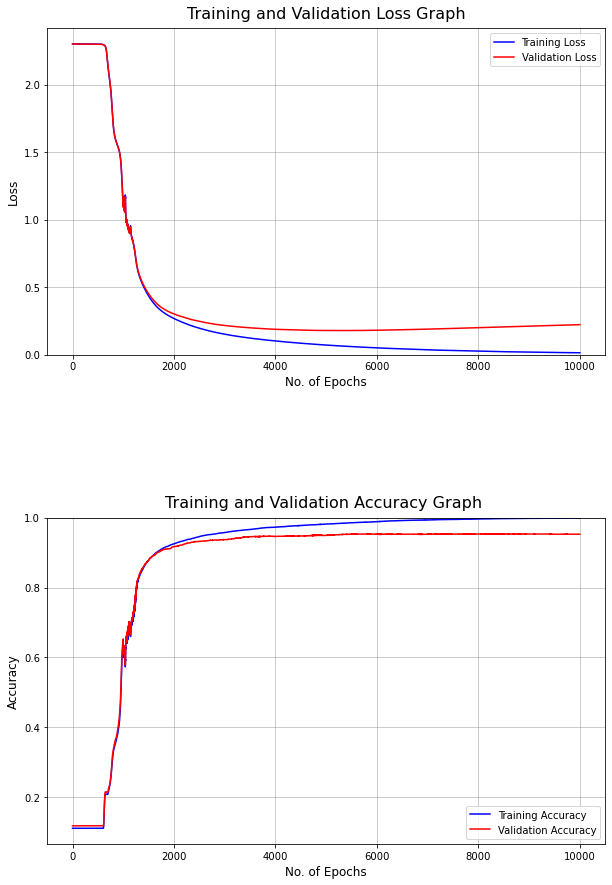


Figure 5.11: Loss and Accuracy of the Baseline Model

The baseline model achieved the validation accuracy of 95.26%, which would be a reference parameter for all the optimization and tuning processes carried out later in the project. Other details about the result of the baseline model are elaborated by the table below:

Table 5.6: Baseline model metrics for training, dev and test set

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset** | **Macro Avg. Precision** | **Macro Avg. Recall** | **Macro Avg. F1-Score** | **Accuracy** |
| Training set | 0.99896 | 0.99894 | 0.99895 | 0.99896 |
| Dev set | 0.95206 | 0.95190 | 0.95193 | 0.95260 |
| Test set | 0.95626 | 0.95573 | 0.95594 | 0.95660 |

### 5.3.4 Model Optimization

Complexity was slowly added to the baseline model using various other techniques and algorithms. Each addition to the model was evaluated with the other, and the best was selected. Few experiments were carried out in which the initialization technique was selected between the random and the He-initialization, the optimizer algorithm was selected among BDG, MDG and Adam, and the regularization technique was selected between dropout and L2 regularization. Different hyperparameters were used according to the need. Experiments 1, 2 and 3 elaborate this selection process.

### 5.3.5 Model Tuning

The optimized model was then tuned for the best possible hyperparameters. Random samples were drawn from a specific range for all the hyperparameters. And a search was carried out among various combinations of the sampled hyperparameters. Finally, with everything selected and tuned, the final model was trained using a large volume of data generated using online data augmentation. Early stopping with patience = 20 was used to train the model, and the best model was saved for further prediction. Experiment 4 and subtopic 5.5 elaborate this selection and training process.

### 5.3.6 Model Deployment

The final model was deployed using a desktop application developed using the Tkinter module. The parameters obtained after the final training of the selected model was loaded to the application for prediction. Users can draw a digit in the interface or upload an image of it, and the application will predict the class of the digit in the image. The interface also displays the second guess of the image along with the confidence of the prediction.

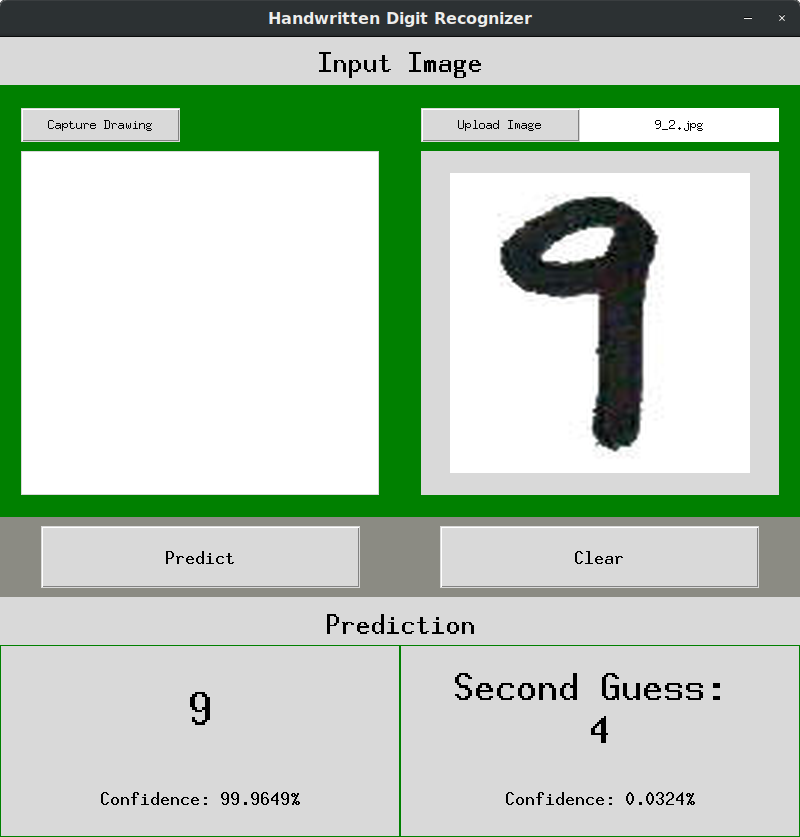


Figure 5.12: Prediction made after deploying final model in GUI interface.

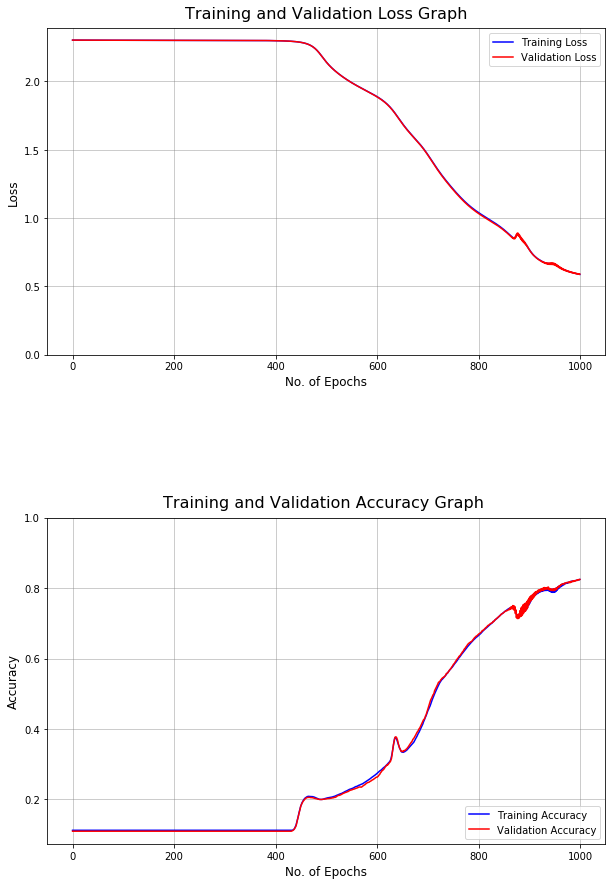
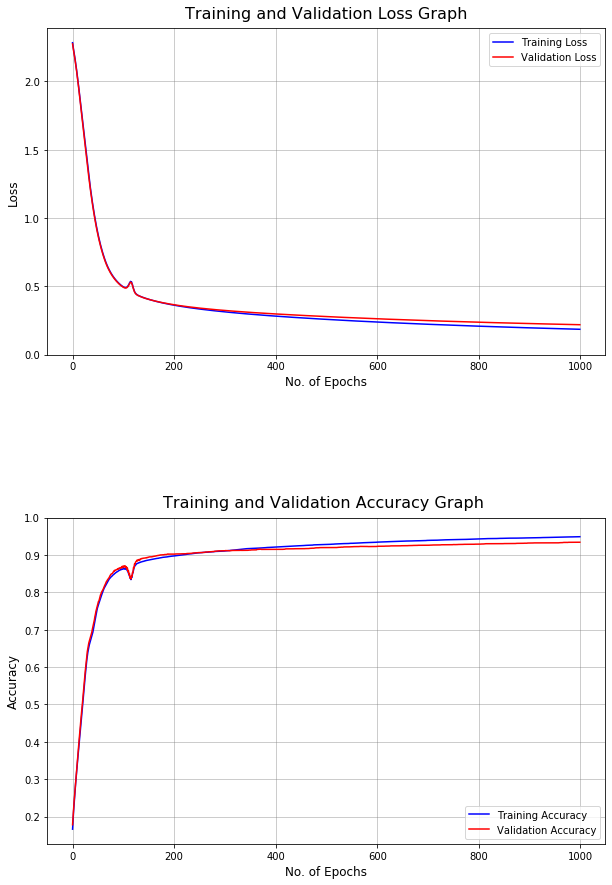
## 5.4 Experiments and Results

All the experiments were performed based on the experiment design the report has discussed in the earlier sections. These experiments were used to obtain the best possible model to recognize the handwritten digits. The algorithms and hyperparameter values that have shown better performance are particular to the problem this project is trying to solve. They may show different performances for different problems and activations used, but verifying that is beyond the scope of this project.

### 5.4.1 Experiment 1: Initialization Selection

The main purpose of this experiment was to select a better initialization technique between random initialization and He initialization. The experiment used the same model configuration as that of the baseline model except for the initialization technique to be evaluated and epoch size. The model was independently trained using each initialization technique for 1000 epochs.

Figure 5.13: Accuracy Graph for experiment with random initialization (left) and he-initialization (right)



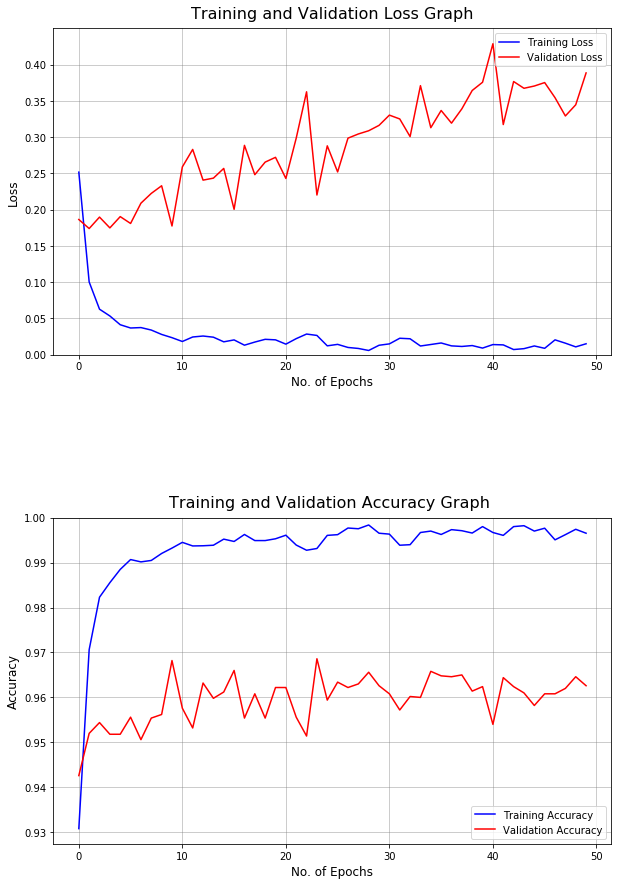
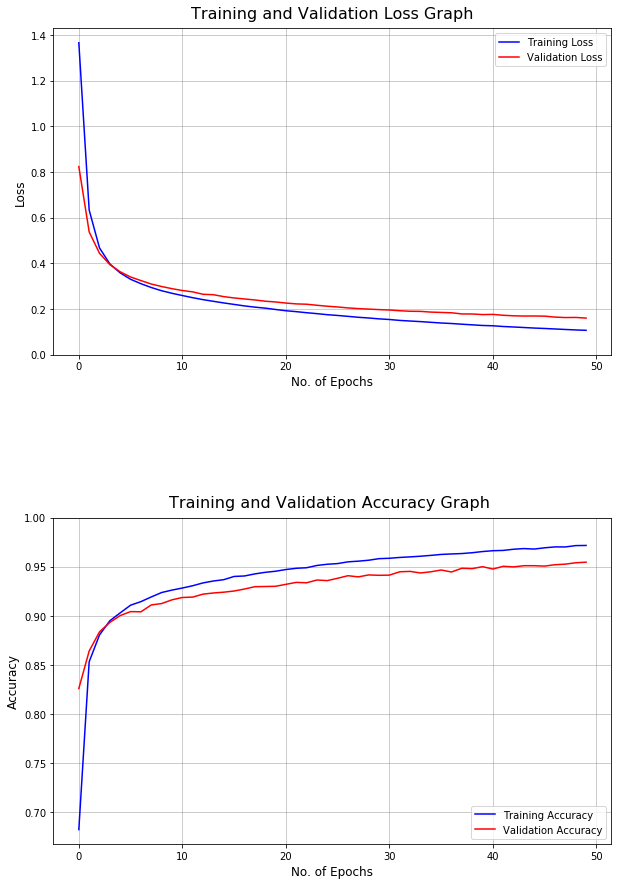
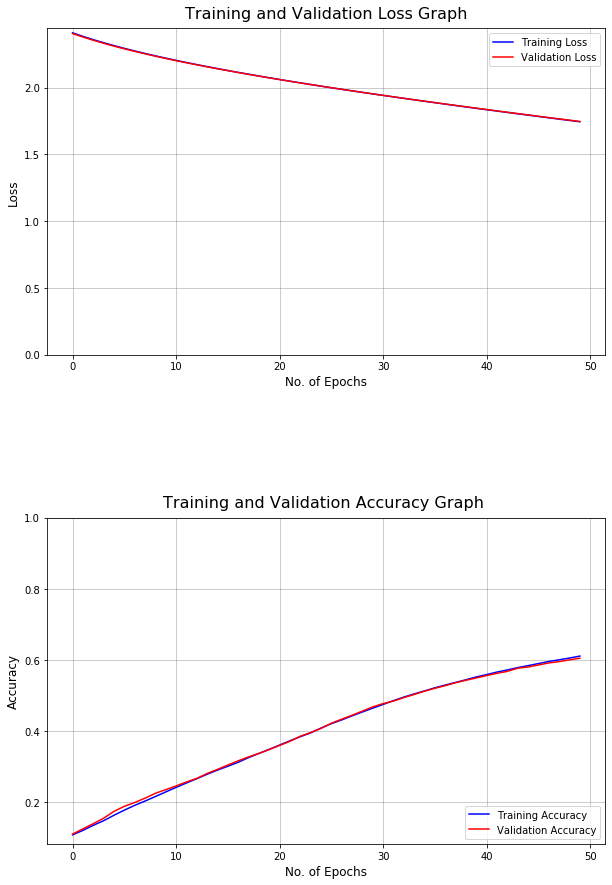
Given the constraints, the random initialization achieved the training accuracy of 82.46% and the validation accuracy of 82.44%. It also seems to be making almost no progress for about 450 epochs. On the other hand, progress in the He-initialization can be seen starting in the very early stage of training leading to the final training accuracy of 94.90% and validation accuracy of 93.42%. The quick convergence and better validation accuracy make He-initialization better than random initialization.

### 5.4.2 Experiment 2: Optimizer Selection

The main purpose of this experiment is to select the best optimization algorithm among BGD, MGD and Adam. A two hidden layered architecture with 512 and 256 hidden units respectively were used for this experiment. The models were trained for 50 epochs each with a learning rate of 0.01. Since He-initialization proved to be better in experiment 1, it was used as the initialization technique. Minibatch gradient descent introduced a new hyperparameter called minibatch size, whose value was taken as 128. Adam introduced 3 new hyperparameters: Beta1, Beta2 and epsilon, whose values were taken as 0.9, 0.999, 1e-8 respectively. The values of the Adam hyperparameters were taken based on the literature and are not tuned in this project. [16]

BGD was able to achieve training accuracy of 61.12% and validation accuracy of 60.52%. MGD achieved a training accuracy of 97.03% and a validation accuracy of 95.46%. And Adam was able to achieve the training accuracy of 99.28% and validation accuracy of 96.26%. On the other hand, the figures above show that the convergence rate of MGD is better than that of BGD. However, Adam seems to be better than MGD, achieving accuracy above 95% in just 2 epochs that took MGD 24 epochs. Thus, Adam seems to be better than BGD and MGD in both the criteria.

Figure 5.14: Accuracy Graphs for experiments with BGD (top-left), MGD (top-right) and Adam (bottom-left)

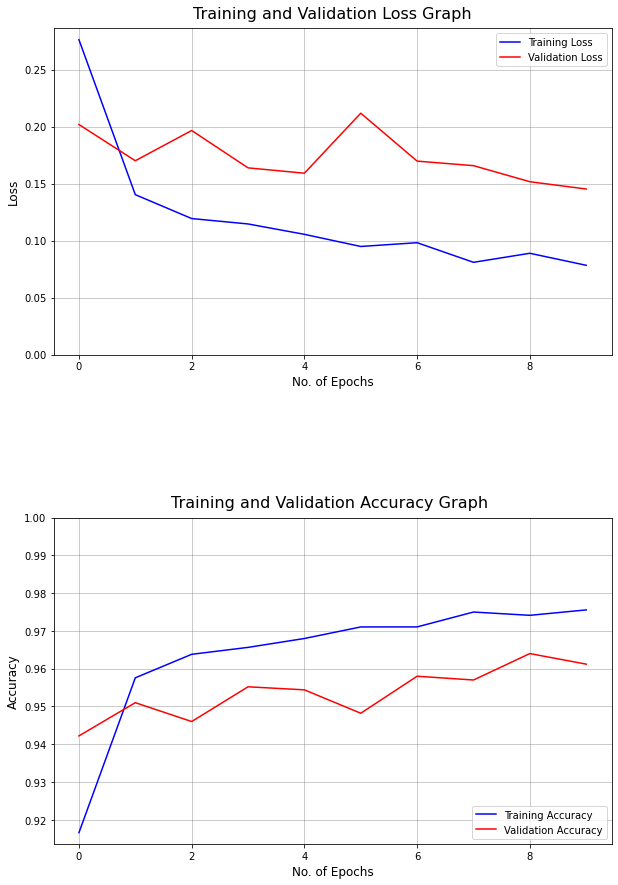
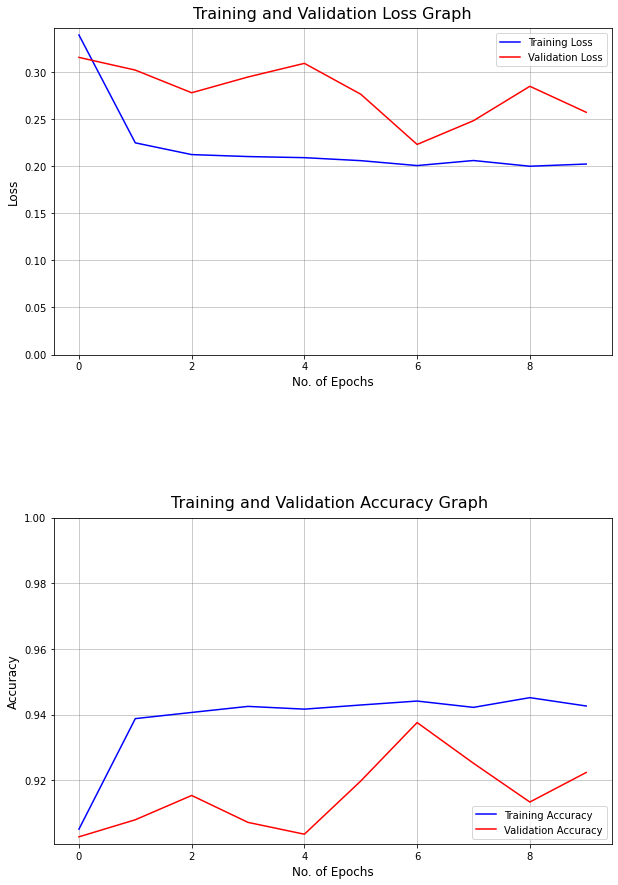


### 5.4.3 Experiment 3: Regularizer Selection

The main purpose of this experiment is to select a better regularization technique between L2 regularization and dropout regularization. The same model configuration as that of experiment 2 was used with few changes. Adam optimizer was used for training the models for ten epochs each. L2 regularization introduced a new hyperparameter called lambda, whose value was taken as 0.7. Dropout also introduced a new hyperparameter called keep\_probs, whose value was taken as 0.7

The model achieved training accuracy of 93.07% and validation accuracy of 92.24% using L2 regularization and training accuracy of 97.46% and validation accuracy of 96.12% using Dropout regularization. The variance using both the technique seems low, but the model’s bias that used L2 regularization is higher. So, dropout has outperformed L2 regularization based on higher validation accuracy.

Figure 5.15: Accuracy Graphs for experiment with L2 (left) and dropout (right) regularization



### 5.4.4 Experiment 4: Hyperparameter Tuning

The purpose of this experiment was to find the best possible hyperparameter combination using a random search. The parameters were initialized using He-initialization and optimized using Adam optimizer. Dropout was used as a regularization technique. The value of the hyperparameters was obtained from randomly generated values within a specific range.

The search was carried in three phases: *coarse search, fine search and detailed search*.

**Coarse Search**: In this phase, the number of hidden layers and the number of hidden units were randomly sampled from the range [1,5) and [400,4000) respectively. The number of hidden units in each layer could be both constant or varied randomly (ordered from larger to small) but are multiple of 4 for computational ease. The sampled pair formed an architecture.

Similarly, the mini-batch size was randomly sampled from the range [50,550), The learning rate was sampled from a log range of [1e-5,1e-1), and the keep\_probs values were selected from the range [0.5,0.9] with step 0.1.

1000 such unique samples were generated, and the search was carried out to find out which combination gave the best result. Training with each sample was carried out for 2 epochs.

The top 10 best results obtained after the search were

Table 5.7: Top 10 best result obtained from Coarse Search

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Validation Acc** | **Learning Rate** | **Minibatch Size** | **Keep\_probs** | **Hidden Layers** | **Hidden Units** |
| 0.9704 | 0.007938 | 434 | 0.9 | 1 | [3448] |
| 0.9684 | 0.002715 | 421 | 0.7 | 2 | [2552, 1444] |
| 0.9682 | 0.000658 | 354 | 0.9 | 2 | [3620, 3516] |
| 0.9668 | 0.002184 | 437 | 0.8 | 2 | [3560, 532] |
| 0.9666 | 0.000366 | 422 | 0.9 | 3 | [3488, 2496, 992] |
| 0.9664 | 0.001781 | 410 | 0.6 | 2 | [3672, 2372] |
| 0.9664 | 0.002413 | 417 | 0.9 | 2 | [2844, 720] |
| 0.9664 | 0.001154 | 399 | 0.7 | 3 | [2992, 1316, 1276] |
| 0.9664 | 0.000876 | 281 | 0.9 | 2 | [2180, 884] |
| 0.9662 | 0.005073 | 440 | 0.8 | 2 | [3768 1912] |

The search took 33 hrs. 28 mins. to complete yielding the best validation accuracy of 97.04% for the combination of the hyperparameters shown in the table below.

Table 5.8: The best result and the new search ranges obtained from Coarse Search

|  |  |  |
| --- | --- | --- |
| Hyperparameter | Best value | New search range |
| Hidden Layers | 1 | [1, 4] |
| Hidden Units | 3448 | [482, 3818] |
| Learning Rate | 0.007938 | [10e-3.74, 10e-1.92] |
| Minibatch Size | 434 | [168, 490] |
| Keep\_probs | 0.9 | [0.9, 0.7, 0.6, 0.8] |

The search also yielded a new range for each hyperparameter as seen in the table. These new ranges were computed based on the top 10 results and would be used for another phase of the random search.

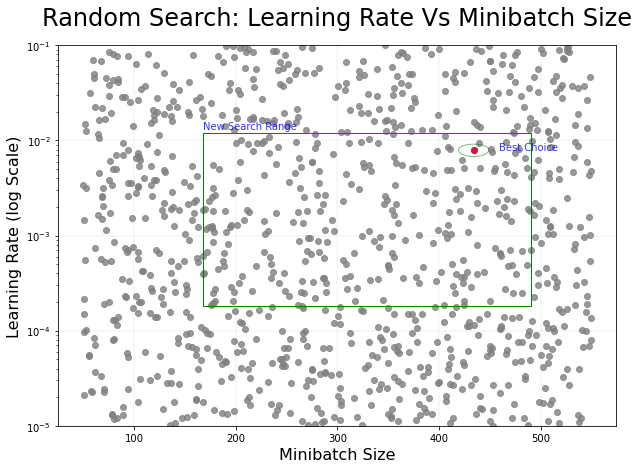


Figure 5.16: New Search Space for minibatch size and learning rate after Coarse Search

The figure above demonstrates a new search space for learning rate and minibatch size generated around the best possible value(circled) after the coarse search.

**Fine Search**: In this phase, the hyperparameters were randomly selected from the new range obtained in the coarse search. This search space is narrower than the initial search space. 1000 unique samples were again generated, and the search was carried out to find out which combination gave the best result. Training with each sample was carried out for 5 epochs.

The top 10 best results obtained after the search were

Table 5.9: Top 10 best result obtained from Fine Search

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Validation Acc** | **Learning Rate** | **Minibatch Size** | **Keep\_probs** | **Hidden Layers** | **Hidden Units** |
| 0.97780 | 0.001976 | 203 | 0.7 | 1 | [2724] |
| 0.97760 | 0.001009 | 335 | 0.7 | 2 | [3492, 2760] |
| 0.97740 | 0.000879 | 397 | 0.6 | 2 | [3624, 492] |
| 0.97740 | 0.000303 | 174 | 0.6 | 3 | [3768, 3152, 924] |
| 0.97680 | 0.000204 | 187 | 0.7 | 3 | [3544, 3164, 1968] |
| 0.97680 | 0.001696 | 173 | 0.9 | 1 | [1432] |
| 0.97680 | 0.000202 | 171 | 0.9 | 3 | [3732, 3440, 1308] |
| 0.97660 | 0.000971 | 255 | 0.6 | 2 | [2356, 2196] |
| 0.97660 | 0.000671 | 220 | 0.8 | 3 | [3604, 1300, 516] |
| 0.97660 | 0.000878 | 354 | 0.9 | 3 | [3556 ,2116, 1984] |

The search took 49 hr. 55 mins. to complete yielding the best validation accuracy of 97.78% for the combination of the hyperparameters shown in the table below.

Table 5.10: The best result and the new search ranges obtained form Fine Search

|  |  |  |
| --- | --- | --- |
| Hyperparameter | Best value | New search range |
| Hidden Layers | 1 | [1, 4] |
| Hidden Units | 2724 | [442, 3818] |
| Learning Rate | 0.001976 | [10e-4.00, 10e-2.53] |
| Minibatch Size | 203 | [102, 447] |
| Keep\_probs | 0.7 | [0.9, 0.7, 0.6, 0.8] |

The search also yielded a new range for each hyperparameter as seen in the table. These new ranges would be used for another phase of the random search.

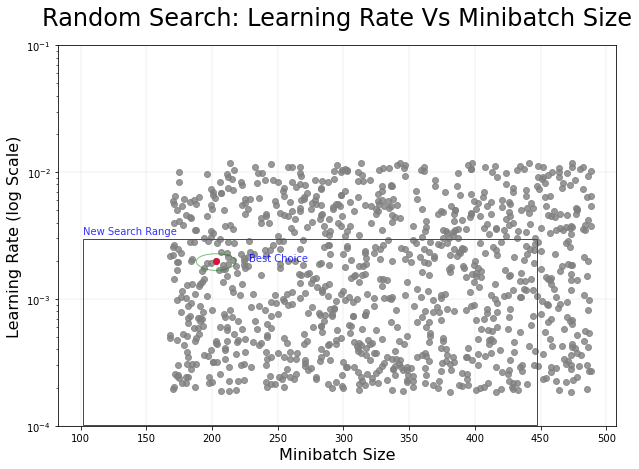


Figure 5.17: New Search Space for minibatch size and learning rate after Fine Search

The figure above demonstrates a new search space for learning rate and minibatch size generated around the best possible value(circled) after the fine search.

**Detailed Search**: In this phase the hyperparameters were randomly selected from the new range obtained in the fine search. The search space is much narrower, and no further search is carried out afterward. 1000 unique samples were again generated, and the search was carried out to find out which combination gave the best result. Training with each sample was carried out for 10 epochs.

The top 10 best results obtained after the search were

Table 5.11: Top 10 best results obtained from the Detailed Search

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Validation Acc** | **Learning Rate** | **Minibatch Size** | **Keep\_probs** | **Hidden Layers** | **Hidden Units** |
| 0.98180 | 0.000672 | 345 | 0.9 | 2 | [2916, 2884] |
| 0.98140 | 0.001595 | 198 | 0.9 | 1 | [2292] |
| 0.9806 | 0.000822 | 159 | 0.9 | 1 | [3400] |
| 0.98060 | 0.000153 | 215 | 0.8 | 3 | [2768, 1528, 1520] |
| 0.98020 | 0.001829 | 226 | 0.8 | 1 | [3772] |
| 0.98020 | 0.000898 | 180 | 0.9 | 1 | [3564] |
| 0.98020 | 0.000481 | 257 | 0.8 | 2 | [2556, 624] |
| 0.98000 | 0.002873 | 370 | 0.6 | 1 | [1944] |
| 0.97980 | 0.000759 | 432 | 0.8 | 2 | [2816, 468] |
| 0.97980 | 0.000818 | 135 | 0.8 | 1 | [2940] |

The search took 106 hrs. 41 mins. to complete yielding the best validation accuracy of 98.18% for the combination of the hyperparameters shown in the table below.

Table 5.12: The Best result obtained from entire Hyperparameter tuning process

|  |  |
| --- | --- |
| Hyperparameter | Best value |
| Hidden Layers | 2 |
| Hidden Units | [2916, 2884] |
| Learning Rate | 0.000672 |
| Minibatch Size | 345 |
| Keep\_probs | 0.9 |

The architecture that obtained the best validation accuracy was selected as a final model and trained using a larger volume of the dataset.

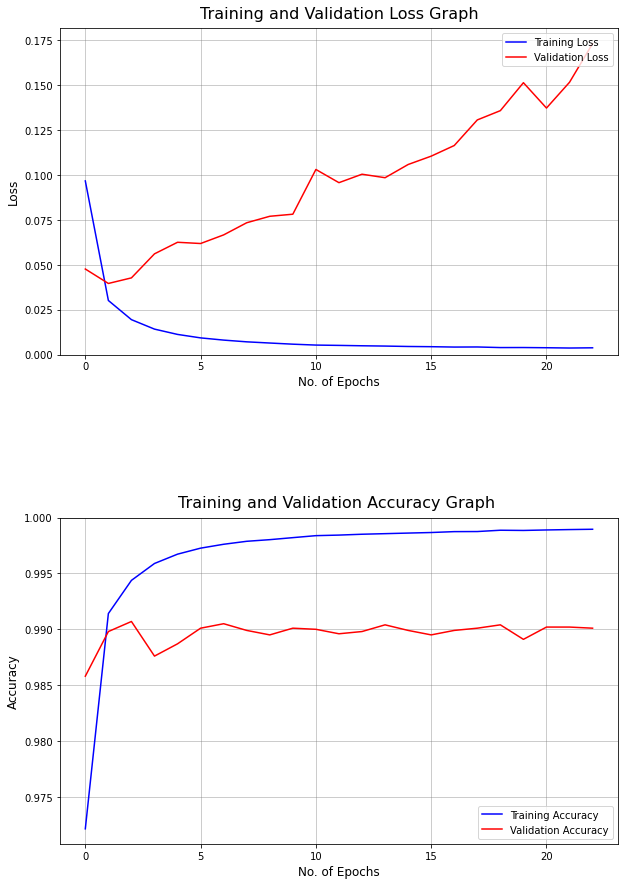
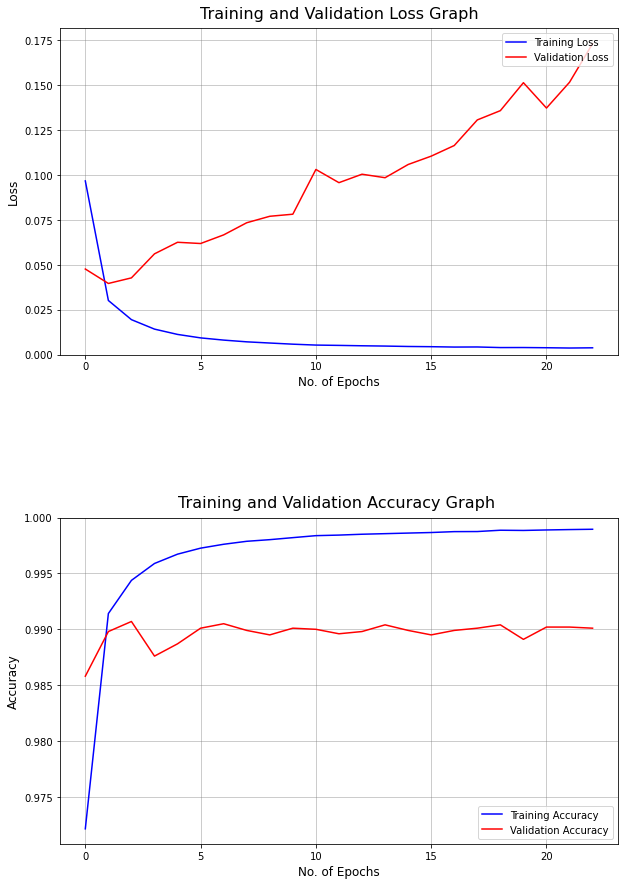
## 5.5 Final model

The final model is a fully connected feedforward network with 2 hidden layers having 2916,2884 neural units respectively. It uses He-initialization for initializing the initial parameters, Adam optimizer to optimize the learning process and dropout as a regularization technique. A learning rate of 0.000672, minibatch size of 345 and dropout parameter of 0.9 are used to train the model. The model was trained for 100 epochs with an early stopping parameter of value 20.

A considerable volume of data was used for the final training. 2 million augmented images were generated from the original training set with 50000 images during each epoch and fed to the training model. The validation set of 10000 images was used for validating the model. The entire training was completed in 29hr 15mins running for 23 epochs.

The best validation accuracy obtained was 99.07% at the 3rd epoch, after which the model started to overfit. The corresponding training accuracy of the model was 99.44%.

Figure 5.18: Loss and Accuracy of the Final Model



Various other evaluation metrics were also obtained. Precision, recall and f1-score for each class were computed. Macro averages of all three metrics were also computed. The macro-average precision was 99.067%, the macro-average recall was 99.053%, and the macro-average f1-score was 99.059% for the validation dataset.

Table 5.13: Precision, Recall, and F1-Score of the final model

|  |  |  |  |
| --- | --- | --- | --- |
| Label | Precision | Recall | F1 Score |
| 0 | 0.99297 | 0.99497 | 0.99397 |
| 1 | 0.99360 | 0.99725 | 0.99542 |
| 2 | 0.99607 | 0.99509 | 0.99558 |
| 3 | 0.99112 | 0.99308 | 0.99210 |
| 4 | 0.99160 | 0.98539 | 0.98848 |
| 5 | 0.99219 | 0.98888 | 0.99053 |
| 6 | 0.98700 | 0.99532 | 0.99114 |
| 7 | 0.99049 | 0.98674 | 0.98861 |
| 8 | 0.99047 | 0.98629 | 0.98837 |
| 9 | 0.98123 | 0.98225 | 0.98174 |
| Macro Avg | 0.99067 | 0.99053 | 0.99059 |

The confusion matrix, on the other hand, the confusion matrix gives some more insights on how the model performed prediction in the validation set and what were its missing points. It can be seen that the model has misclassified some of the images that have similar structures like loops, semi-loops and straight lines. The model seems to have confused 4 with 9, 5 with 6, 7 with 1 and 9, and many digits with similar structures. Overall the final model obtained has misclassified 93 out of 10,000 images in the validation set.



Figure 5.19: Confusion matrix for the predictions of the development set

Below are some samples of misclassified images from the validation set.

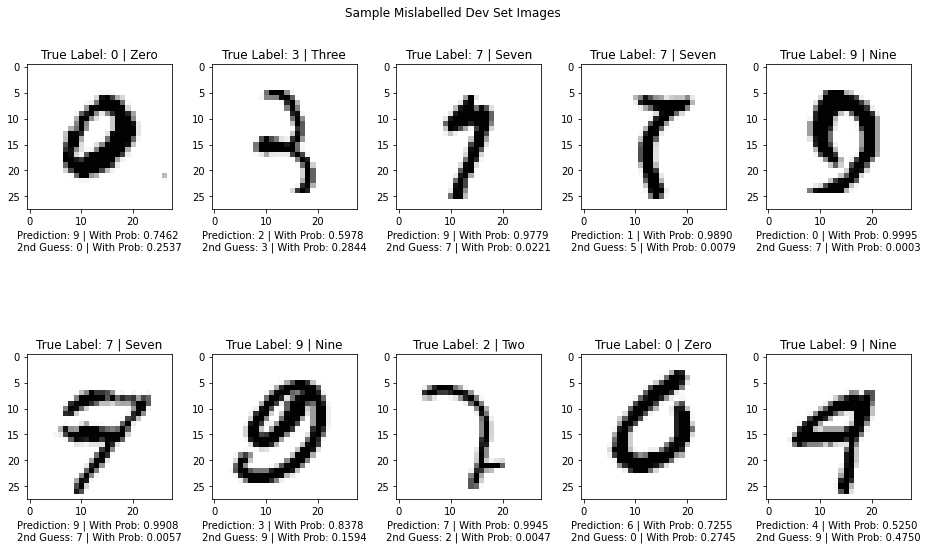


Figure 5.20: Sample mislabeled images from the development set

# : TESTING

Various testing strategies were used to test the developed system. Unit testing was used to test the behavior of the major individual functions. Integration testing was used to test the behavior of the entire training pipeline. Furthermore, System testing was carried out to test the actual performance of the trained model in the test set. GUI interface was also tested during the system testing phase under different scenarios. [25]

## 6.1 Unit Testing

The following test cases were evaluated during the Unit testing phase.

Table 6.1: Test Case 1 - Dataset Preparation Module Test

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario ID | | Dataset Preparation-1 | | Test Case ID | Dataset Preparation-1A | |
| Test Case Description | | Test Case for dataset preprocessing for model training | | Test Priority | High | |
| Prerequisite | | Dataset type and dataset source URL | | Post-Request | NA | |
| Test Execution Steps: | | | | | | |
| S.N. | Action | Input | Expected Output | Actual Output | | Test Result |
| 1 | Download Dataset | Dataset name, the path to which the dataset is to be downloaded | 4 Dataset gzip files downloaded to the specified path, including missing files (if any) | All dataset files downloaded to the predefined path in gzip format (including any missing files) | | Pass |
| 2 | Decompress Dataset | The path that consists of the dataset files | Extracted binary files of the dataset in the same path | 4 binary dataset files in the same path | | Pass |
| 3 | Retrieve Dataset | Path of the dataset, name of the dataset files | Numpy arrays: training images of size (60000,28,28),  Training labels of size (60000,1),  Test images of size (10000,28,28) and Test labels of size (10000,1) | Numpy arrays: training images of size (60000,28,28),  Training labels of size (60000,1),  Test images of size (10000,28,28) and Test labels of size (10000,1) | | Pass |
| 4 | Sample Dataset | Numpy array of images and labels, the sample size | Images and labels of the given sample size | Images and labels of the given sample size | | Pass |
| 5 | Split Training set | Training images and labels | Split training set into training and dev set based on the size of the dataset supplied | Training and Dev set of the intended size | | Pass |
| 6 | Flatten Input Images | Training images or Dev Images or Test Images | Images converted to 1D vectors for respective dataset {form size (m,28,28) to (784,m)} | Images converted to 1D vectors for respective dataset | | Pass |
| 7 | Normalize Input Images | Training images or Dev Images or Test Images | The normalized value of the images in between 0 and 1 | The normalized value of the images in between 0 and 1 | | Pass |
| 8 | Encode Output Labels | Training, Dev or Test set labels | One hot encoded labels for corresponding datasets | One hot encoded labels for corresponding datasets | | Pass |

Table 6.2: Test Case 2 - Dataset Augmentation Module Test

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario ID | | Dataset Augmentation-1 | | Test Case ID | Dataset Augmentation-1A | |
| Test Case Description | | Test Case for dataset augmentation | | Test Priority | High | |
| Prerequisite | | Loaded dataset | | Post-Request | NA | |
| Test Execution Steps: | | | | | | |
| S.N. | Action | Input | Expected Output | Actual Output | | Test Result |
| 1 | Generate mini-batches of the dataset | Input image, labels, minibatch size, a seed value | Suffered and randomly split mini-batches of the dataset | Suffered and randomly split mini-batches of the dataset | | Pass |
| 2 | Rotate Image | Image, label | Randomly rotated images between -60 to +60 degree | Randomly rotated images | | Pass |
| 3 | Blur Image | Image, label, filter mode | Randomly blurred Images using 5 different filters | Randomly blurred Images | | Pass |
| 4 | Shift Image | Image, label, shifting direction | Randomly shifted Images in a horizontal and vertical direction | Randomly shifted Images | | Pass |
| 5 | Crop and Pad Image | Image, label | Randomly cropped images that ate padded with white pixels to regain the original image size | Randomly cropped Images | | Pass |
| 6 | Zoom Image | Image, label | Randomly zoomed out images | Randomly zoomed Images | | Pass |
| 7 | Save generated data | The augmented batch of data, Path to save the data | Binary pickle dump files consisting of the augmented data in the given path | Binary pickle dump files consisting of the augmented data in the given path | | Pass |
| 8 | Load generated data | Path of the data file | The augmented batch of images with their corresponding labels | The augmented batch of images with their corresponding labels | | Pass |

Table 6.3: Test Case 3 - Utility Module Test

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario ID | | Utility-1 | | Test Case ID | Utility-1A | |
| Test Case Description | | Test Case for utility functions | | Test Priority | High | |
| Prerequisite | | A seed value for random value generation | | Post-Request | NA | |
| Test Execution Steps: | | | | | | |
| S.N. | Action | Input | Expected Output | Actual Output | | Test Result |
| 1 | Compute relu | Z = np.random.randn(1,6) | [[1.62434536 0. 0. 0. 0.86540763 0.] ] | [[1.62434536 0. 0. 0. 0.86540763 0.] ] | | Pass |
| 2 | Compute Gradient of Relu | dA = np.random.randn(1,6),  cache = np.random.randn(1,6) | [[1.62434536 0. -0.52817175 0. 0.86540763 0. ]] | [[1.62434536 0. -0.52817175 0. 0.86540763 0. ]] | | Pass |
| 3 | Compute Softmax | Z = np.random.rand(7,1) | [[0.15477477]  [0.10270926]  [0.17340649]  [0.15467071]  [0.15237489]  [0.13925557]  [0.12280831]] | [[0.15477477]  [0.10270926]  [0.17340649]  [0.15467071]  [0.15237489]  [0.13925557]  [0.12280831]] | | Pass |
| 4 | Generate random minibatch of data | X = np.random.randn(20,20),  Y = np.random.rand(1,20) | Minibatches[0][0].shape = (20, 4) ,  Minibatches[0][0].shape =(1, 4) | Minibatches[0][0].shape = (20, 4) ,  Minibatches[0][0].shape =(1, 4) | | Pass |
| 5 | Perform Time Conversion | millisec = 12450 | 0hr 0mins 12s 450ms | 0hr 0mins 12s 450ms | | Pass |
| 6 | Generate Confusion Matrix | y = np.array([[1,2,3,3,0,2,5,4,2,2]]).reshape(10,1),  pred = np.array([[2,2,1,3,0,2,5,4,2,2]]).reshape(1,10) | [[1 0 0 0 0 0]  [0 0 1 0 0 0]  [0 0 4 0 0 0]  [0 1 0 1 0 0]  [0 0 0 0 1 0]  [0 0 0 0 0 1]] | [[1 0 0 0 0 0]  [0 0 1 0 0 0]  [0 0 4 0 0 0]  [0 1 0 1 0 0]  [0 0 0 0 1 0]  [0 0 0 0 0 1]] | | Pass |
| 7 | Compute Precision | [[1 0 0 0 0 0]  [0 0 1 0 0 0]  [0 0 4 0 0 0]  [0 1 0 1 0 0]  [0 0 0 0 1 0]  [0 0 0 0 0 1]],  Label = 2 | 0.8 | 0.8 | | Pass |
| 8 | Compute Recall | [[1 0 0 0 0 0]  [0 0 1 0 0 0]  [0 0 4 0 0 0]  [0 1 0 1 0 0]  [0 0 0 0 1 0]  [0 0 0 0 0 1]],  Label = 2 | 1.0 | 1.0 | | Pass |
|  | Compute F1-Score | prec = 0.8  rec = 1.0 | 0.888888888888889 | 0.888888888888889 | | Pass |
|  | Compute Accuracy | [[1 0 0 0 0 0]  [0 0 1 0 0 0]  [0 0 4 0 0 0]  [0 1 0 1 0 0]  [0 0 0 0 1 0]  [0 0 0 0 0 1]] | 0.8 | 0.8 | | Pass |
|  | Save Model | Trained model, filename | Model saved with the given filename. | Model saved with the given filename. | | Pass |
|  | Load Model | filename | A model with the given filename loaded | A model with the given filename loaded | | Pass |

Table 6.4: Test Case 4 - Neural Network Module Test

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario ID | | Neural Network-1 | | Test Case ID | Neural Network-1A | |
| Test Case Description | | Test Case for Neural Network functions | | Test Priority | High | |
| Prerequisite | | Loaded and processed dataset | | Post-Request | NA | |
| Test Execution Steps: | | | | | | |
| S.N. | Action | Input | Expected Output | Actual Output | | Test Result |
| 1 | Initialize hidden layers | Input shape, output shape, other hidden units 784, 10, hidden\_layers = [32,16] | [784, 32, 16, 10] | [784, 32, 16, 10] | | Pass |
| 2 | Initialize parameters | Layer dimensions,  Initialization type | Initialized weights and biases based on the layer dimension | Initialized weights and biases based on the layer dimension | | Pass |
| 3 | Initialize hyperparameters | Hyperparameter values | Dictionary of hyperparameters with values | Dictionary of hyperparameters with values | | Pass |
| 4 | Initialize Adam Parameters | Parameters | Initialized v and s | Initialized v and s | | Pass |
| 5 | Compute Forward propagation | Input images, parameters, keep\_probs | Outputs values of the last layer and caches | Outputs values of the last layer and caches | | Pass |
| 6 | Compute Softmax Cross Entropy | The output of the final layer, caches | Scalar cross-entropy cost | Scalar cross-entropy cost | | Pass |
| 7 | Compute Backward Propagation | The output of the final layer, caches, Output labels | Gradients of each layer | Gradients of each layer | | Pass |
| 8 | Update Parameters | Parameters, gradients, hyperparameters | Updated parameters | Updated parameters | | Pass |
| 9 | Evaluate Model | Input images and output labels | Accuracy and loss of the dataset | Accuracy and loss of the dataset | | Pass |
| 10 | Predict Output | Input images, parameters | First and second predictions | First and second predictions | | Pass |

Table 6.5: Test Case 5 - GUI Module Test

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario ID | | GUI-1 | | Test Case ID | GUI-1A | |
| Test Case Description | | Test Case for GUI Interface | | Test Priority | High | |
| Prerequisite | | Trained Models, scanned handwritten digits. | | Post-Request | NA | |
| Test Execution Steps: | | | | | | |
| S.N. | Action | Input | Expected Output | Actual Output | | Test Result |
| 1 | Load parameters | Saved model path | Loaded parameters of the save model | Loaded parameters of the save model | | Pass |
| 2 | Upload image | Click the ‘Upload Image’ button,  Uploaded Image | Image preview and image name preview seen in the interface | Image preview and image name preview seen in the interface | | Pass |
| 3 | Draw on canvas | Event (mouse drag) | A drawing of a digit on the GUI interface | A drawing of a digit on the GUI interface | | Pass |
| 4 | Capture canvas screen | Click the ‘Capture Drawing’ button | A preview of the drawn image in the interface | A preview of the drawn image in the interface | | Pass |
| 5 | Clear canvas | Click the ‘Clear’ button | Clear canvas, previews and predictions | Clear canvas, previews and predictions | | Pass |
| 6 | Process Image | Uploaded or drawn image | Resize, Flattened and normalized image | Resize, Flattened and normalized image | | Pass |
| 7 | Classify image | Click the ‘Predict’ button | First and second predictions in the interface | First and second predictions in the interface | | Pass |

## 6.2 Integration Testing

The following test cases were carried out during the integration testing phase.

Table 6.6: Test Case 6 - Training Module Test

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario ID | | Training-1 | | Test Case ID | Training-1A | |
| Test Case Description | | Test Case for integrating all the modules and training the model | | Test Priority | High | |
| Prerequisite | | Dataset, data augmentation and utility modules | | Post-Request | NA | |
| Test Execution Steps: | | | | | | |
| S.N. | Action | Input | Expected Output | Actual Output | | Test Result |
| 1 | Load Dataset | Dataset type, a sample size | Numpy array of training and dev set images and labels | Numpy array of training and dev set images and labels | | Pass |
| 2 | Prepare Dataset | Train, dev and test datasets | Flattened and normalized image and one-hot encoded labels | Flattened and normalized image and one-hot encoded labels | | Pass |
| 3 | Dataset generator | Training dataset, augmentation count, batch size | Generate and saves augmented training set | Number of augmented dataset files equal to the augmentation count | | Pass |
| 4 | Train model | Train and dev datasets, parameters, hyperparameters, optimizer, initialization, regularizer, and patience | A trained model with the best validation accuracy.  Decreasing training loss and increasing training accuracy | A trained model with the best validation accuracy,  Decreasing training loss and increasing training accuracy | | Pass |

## 6.3 System Testing

The trained model was tested for its performance in the test set. The process yielded a test accuracy of 99.16% along with other metrics as shown in the table below:

Table 6.7: Precision, Recall and F1-Score for Test set

|  |  |  |  |
| --- | --- | --- | --- |
| **Label** | **Precision** | **Recall** | **F1 Score** |
| 0 | 0.99592 | 0.99592 | 0.99592 |
| 1 | 0.99385 | 0.99648 | 0.99516 |
| 2 | 0.98843 | 0.99322 | 0.99082 |
| 3 | 0.99211 | 0.99604 | 0.99407 |
| 4 | 0.98981 | 0.98880 | 0.98930 |
| 5 | 0.99549 | 0.98879 | 0.99213 |
| 6 | 0.98858 | 0.99374 | 0.99115 |
| 7 | 0.98735 | 0.98735 | 0.98735 |
| 8 | 0.99483 | 0.98871 | 0.99176 |
| 9 | 0.99005 | 0.98612 | 0.98808 |
| **Macro Avg** | **0.99164** | **0.99152** | **0.99157** |

Some other tests on the GUI interface were carried out during this testing phase. They are briefed below.

Table 6.8: Test Case 7 - System Test

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Scenario ID | | System-1 | | Test Case ID | System-1A | |
| Test Case Description | | Test Case for Complete system functioning | | Test Priority | High | |
| Prerequisite | | Pretrained model, scanned handwritten digit | | Post-Request | NA | |
| Test Execution Steps: | | | | | | |
| S.N. | Action | Input | Expected Output | Actual Output | | Test Result |
| 1 | Predict Uploaded image | Image of a handwritten digit (3), Click the ‘Predict’ button | The first prediction value as 3 | The first prediction value as 3 | | Pass |
| 2 | Predict drawn image | Image of a digit captured from the canvas (2), click ‘Predict’ button | The first prediction value as 2 | The first prediction value as 2 | | Pass |
| 3 | No image uploaded or drawn | Click the ‘Predict’ button | Message to upload or draw an image | Message to upload or draw an image | | Pass |
| 4 | Clear | Click the ‘Clear’ button | All preview and messages erased, canvas empty | All preview and messages erased, canvas empty | | Pass |

The completion of all the above test cases successfully indicates that the system was able to meet all of its primary objectives. The results clearly demonstrate that the prediction of the system can be relied upon.

# : CONCLUSION

## 7.1 Project Conclusion

The project demonstrates the use of MLP with two hidden layers with 2916, 2884 neural units combined with some modern optimization algorithms like dropout and Adam to classify Handwritten digits with high precision. Online Data augmentation technique (Affine Transformations only) was used in the MNIST training set to generate 2 million randomly transformed images during each training epoch, and the training lasted for 23 such epochs. This can be extrapolated as the model has been fed with 46 million slightly varied images from the MNIST dataset. The model thus obtained was able to achieve a test accuracy of 99.16%, being unable to classify 84 out of 10000 images in the test set. Even so, the second guess for 64 out of 84 misclassified images was correct, which could mean that there is still room for improvement. The rest misclassified images are challenging even for human eyes to correctly identify due to their disfigured nature, which might have occurred during the image extraction process. So, if compared to the models yielding better results shown in the benchmarks, the result of this project seems to be pretty good considering the use of less sophisticated techniques and simpler architecture.

## 7.2 Future Work

Better results could have been obtained using more advanced algorithms, activation functions, tuning mechanisms and optimization processes as seen in more advanced researches. However, lack of adequate knowledge and resources limited the scope of the project. Nevertheless, the project can be easily enhanced in the future. More advanced techniques like Batch Normalization and Model Ensemble could be used. Elastic distortion of images seems to yield a better result as well. Regularization techniques like Drop Connect and activation functions like maxout could also be used to improve the results. Moreover, implementing different architecture like CNN could yield a way better result as demonstrated by a large number of researchers. The other way that could be used to extend the project is by implementing multilabel classification. This could enable the system to identify two or more digits in the same image.

Thus the project can be taken in multiple directions, whether it be an improvement on its application or improvement in the model performance itself.

# REFERENCES

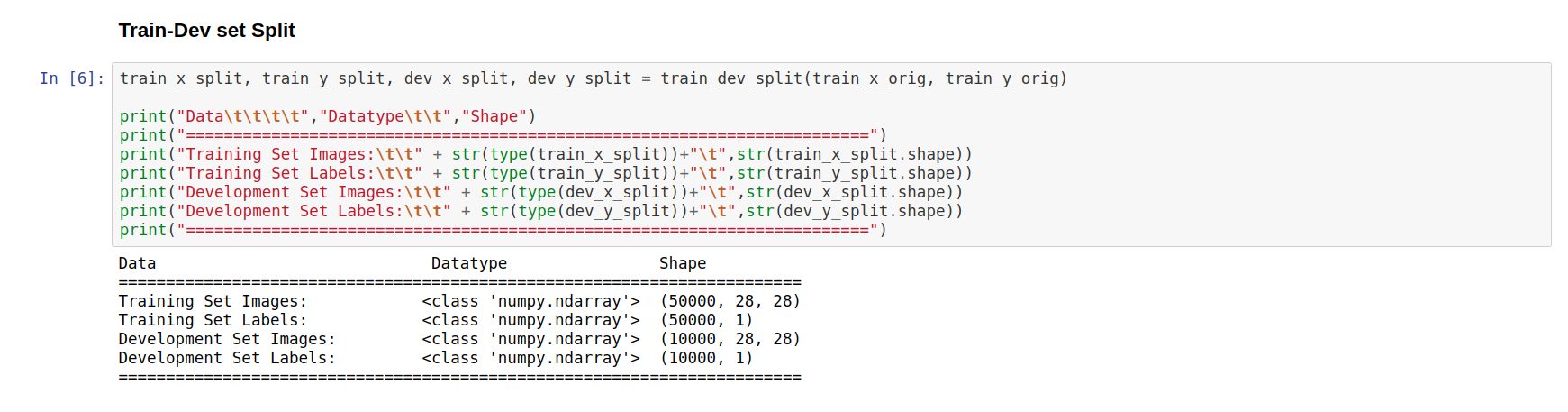
1. Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.
2. P. Y. Simard, D. Steinkraus and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings., Edinburgh, UK, 2003, pp. 958-963
3. G. E. Hinton, S. Osindero and Y. Teh, "A Fast Learning Algorithm for Deep Belief Nets," in Neural Computation, vol. 18, no. 7, pp. 1527-1554, July 2006.
4. D. C. Ciresan, U. Meier, L. M. Gambardella, J. Schmidhuber, “Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition.” in Neural Computation, vol. 22, no. 12, pp. 3207–3220, Dec 2010.
5. D. Keysers, T. Deselaers, C. Gollan and H. Ney, "Deformation Models for Image Recognition," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 29, no. 8, pp. 1422-1435, Aug. 2007.
6. D. Decoste, B. Schölkopf, “Training Invariant Support Vector Machines,” Machine Learning, 46, 161–190, 2002.
7. D. Ciresan, U. Meier and J. Schmidhuber, "Multi-column deep neural networks for image classification," 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, 2012, pp. 3642-3649.
8. L. Wan, M. D. Zeiler, S. Zhang, Y. LeCun, R. Fergus, “Regularization of Neural Networks using DropConnect,” ICML, 2013.
9. “Papers with Code - MNIST Benchmark (Image Classification),” The latest in machine learning. [Online]. Available: https://paperswithcode.com/sota/image-classification-on-mnist.
10. “THE MNIST DATABASE,” MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. [Online]. Available: http://yann.lecun.com/exdb/mnist/.
11. K. Kowsari, M. Heidarysafa, D. E. Brown, K. J. Meimandi, and L. E. Barnes, “RMDL: Random Multimodel Deep Learning for Classification,” arXiv.org, 31-May-2018. [Online]. Available: https://arxiv.org/abs/1805.01890v2.
12. Byerly, T. Kalganova, and I. Dear, “A Branching and Merging Convolutional Network with Homogeneous Filter Capsules,” arXiv.org, 13-Jul-2020. [Online]. Available: https://arxiv.org/abs/2001.09136v4 .
13. S. Ravi, “ProjectionNet: Learning Efficient On-Device Deep Networks Using Neural Projections,” arXiv.org, 09-Aug-2017. [Online]. Available: https://arxiv.org/abs/1708.00630v2.
14. M. Courbariaux, Y. Bengio, and J.-P. David, “BinaryConnect: Training Deep Neural Networks with binary weights during propagations,” arXiv.org, 18-Apr-2016. [Online]. Available: https://arxiv.org/abs/1511.00363v3.
15. Kulbear, “Kulbear/deep-learning-nano-foundation,” GitHub. [Online]. Available: https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions.
16. “Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization,” Coursera. [Online]. Available: http://www.coursera.org/learn/deep-neural-network.
17. “Neural Networks and Deep Learning,” Coursera. [Online]. Available: http://www.coursera.org/learn/neural-networks-deep-learning.
18. Kapil, Divakar. “Stochastic vs Batch Gradient Descent.” Medium, Medium, 21 June 2019. [Online]. Available: https://medium.com/@divakar\_239/stochastic-vs-batch-gradient-descent-8820568eada1
19. D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” arXiv.org, 30-Jan-2017. [Online]. Available: https://arxiv.org/abs/1412.6980.
20. “Papers with Code - Adam Explained,” The latest in machine learning. [Online]. Available: https://paperswithcode.com/method/adam.
21. S. Yıldırım, “L1 and L2 Regularization - Explained,” Medium, 08-May-2020. [Online]. Available: https://towardsdatascience.com/l1-and-l2-regularization-explained-874c3b03f668.
22. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” Journal of Machine Learning Research, 01-Jan-1970. [Online]. Available: https://jmlr.org/papers/v15/srivastava14a.html.
23. CS231n Convolutional Neural Networks for Visual Recognition. [Online]. Available: https://cs231n.github.io/neural-networks-3/.
24. MNIST on Benchmarks.AI. [Online]. Available: https://benchmarks.ai/mnist.
25. Jeremy Jordan, “Organizing machine learning projects: project management guidelines.,” Jeremy Jordan, 22-Dec-2020. [Online]. Available: https://www.jeremyjordan.me/ml-projects-guide/.
26. J. Brownlee, “A Gentle Introduction to Imbalanced Classification,” Machine Learning Mastery, 14-Jan-2020. [Online]. Available: https://machinelearningmastery.com/what-is-imbalanced-classification/.

# APPENDIX

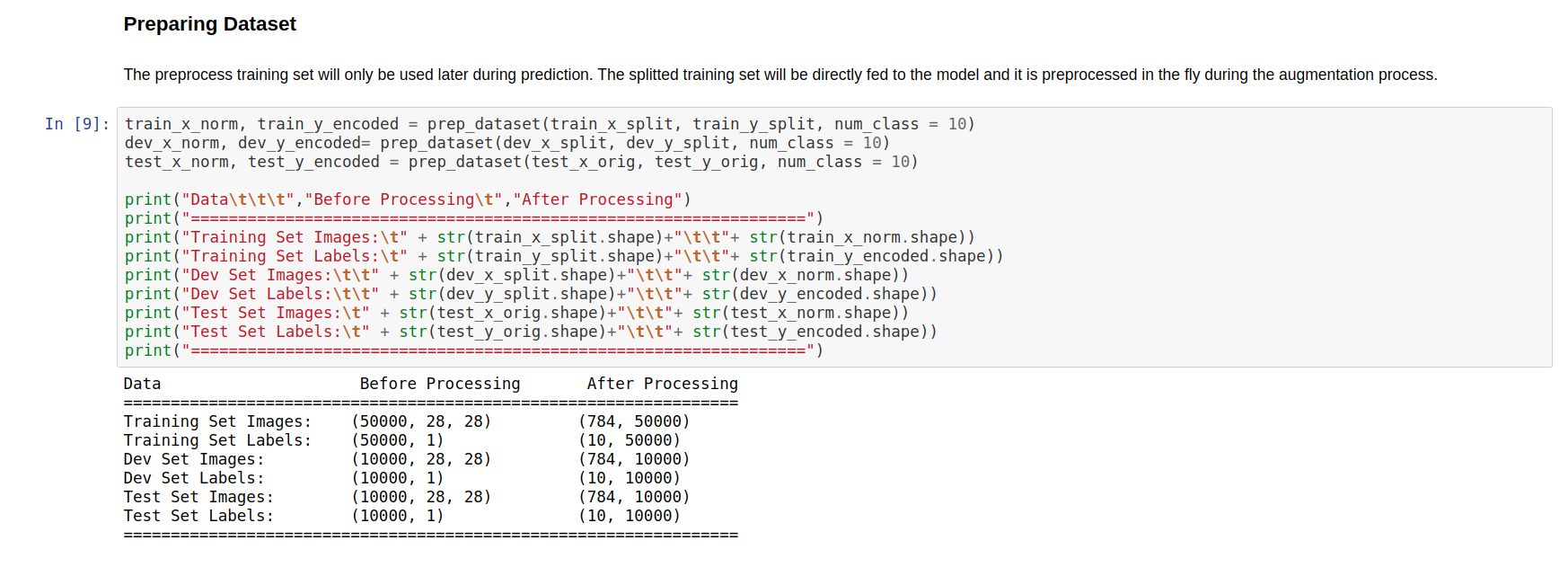
## Notebook Screenshots:



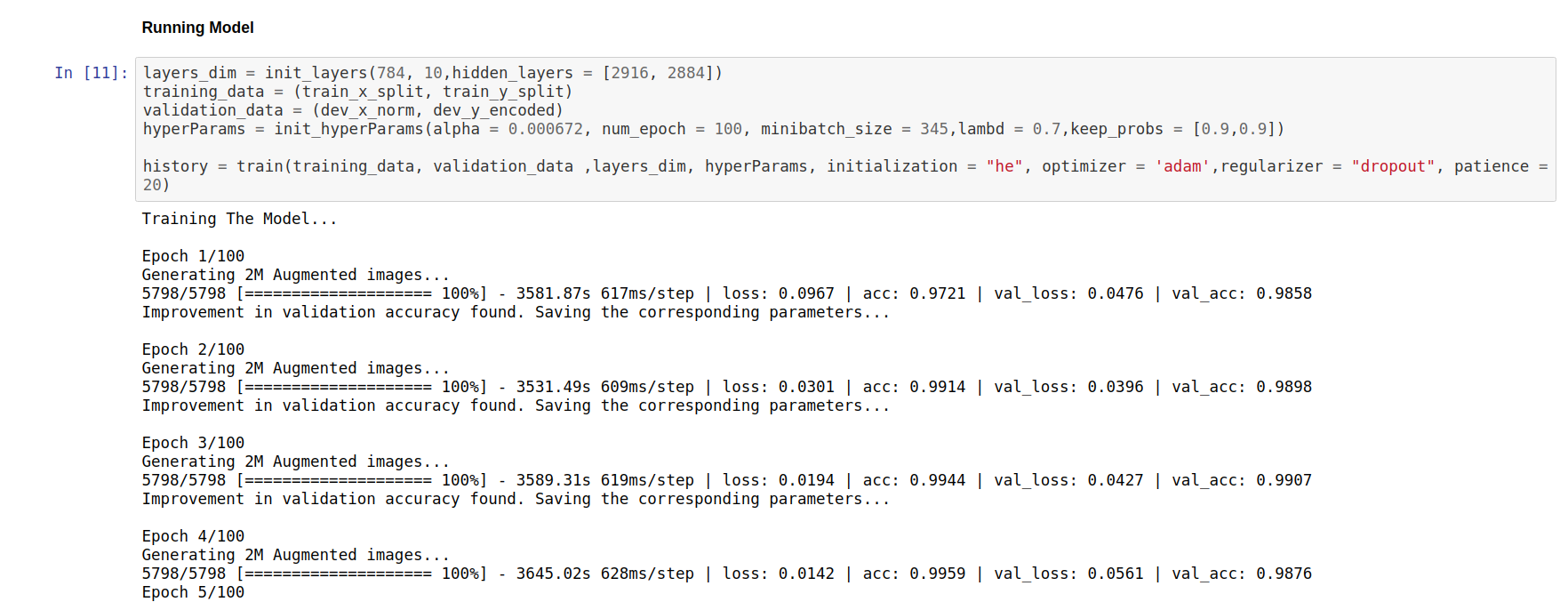
Appendix A: Notebook Screenshot of Loading Dataset



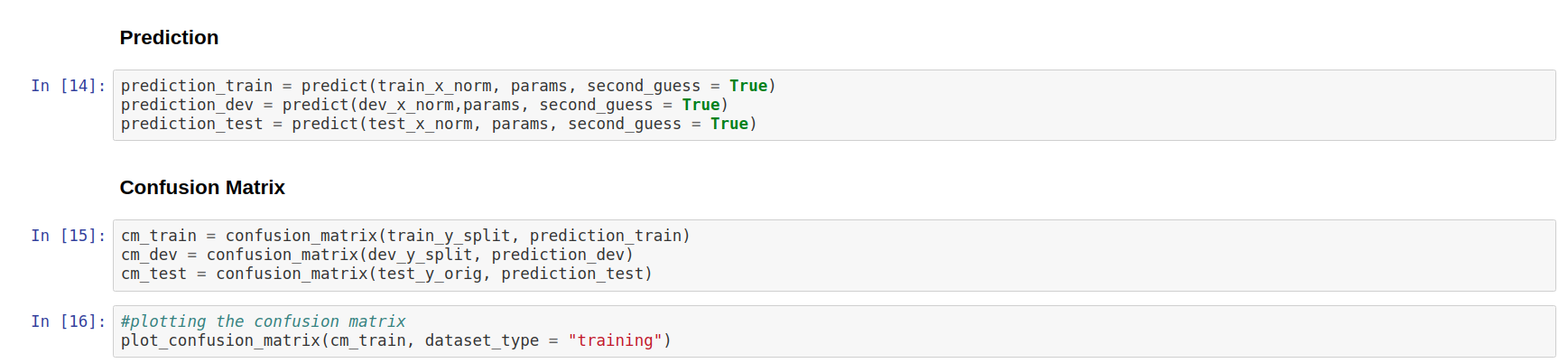
Appendix B: Notebook Screenshot of Splitting Training Dataset



Appendix C: Notebook Screenshot of Preparing Dataset



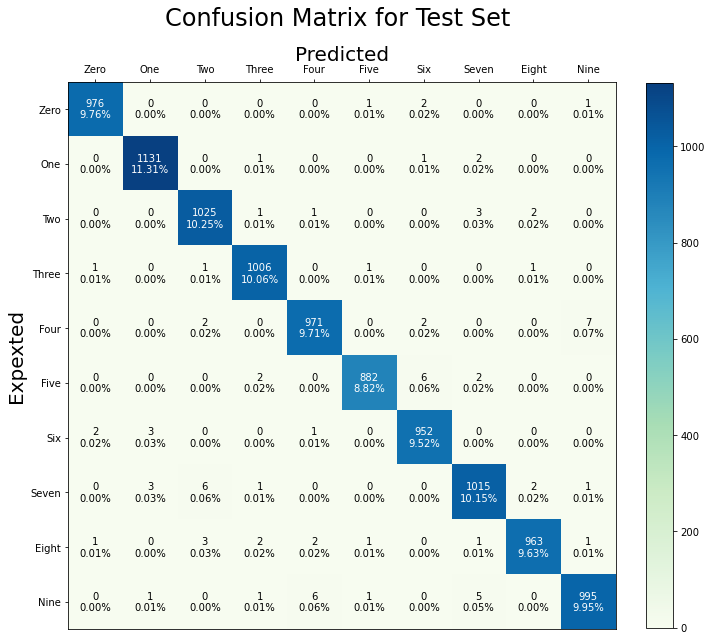
Appendix D: : Notebook Screenshot of Training the Final Model



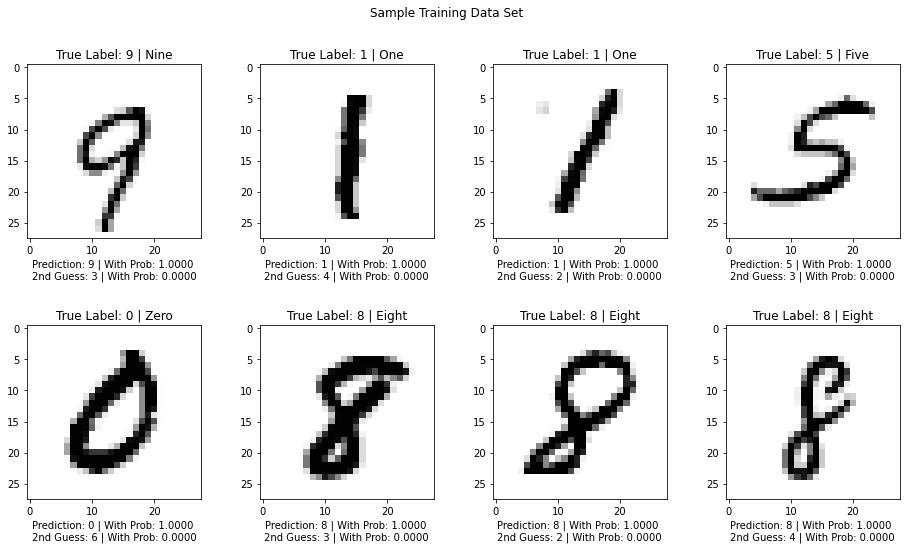
Appendix E: Notebook Screenshot of making Prediction in all three datasets and generating Confusion Matrix



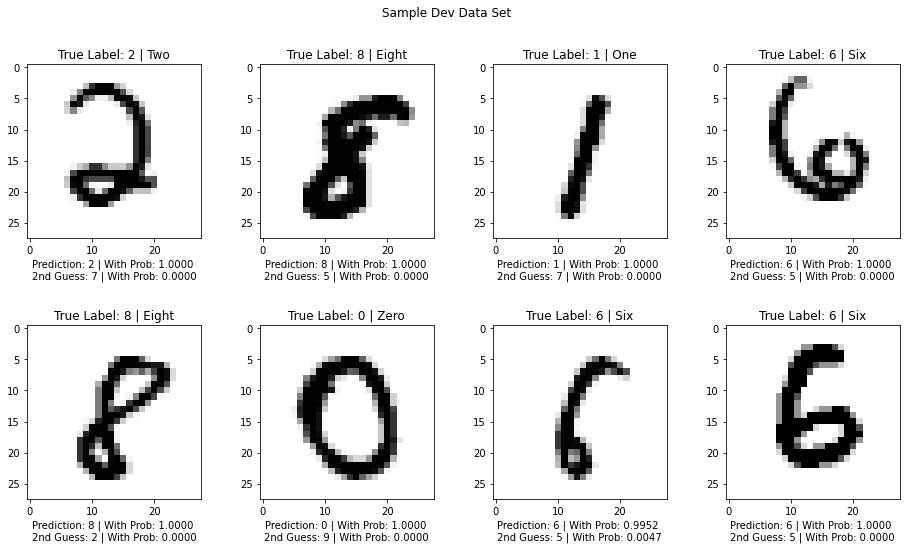
Appendix F: Confusion Matrix of Training Set Prediction



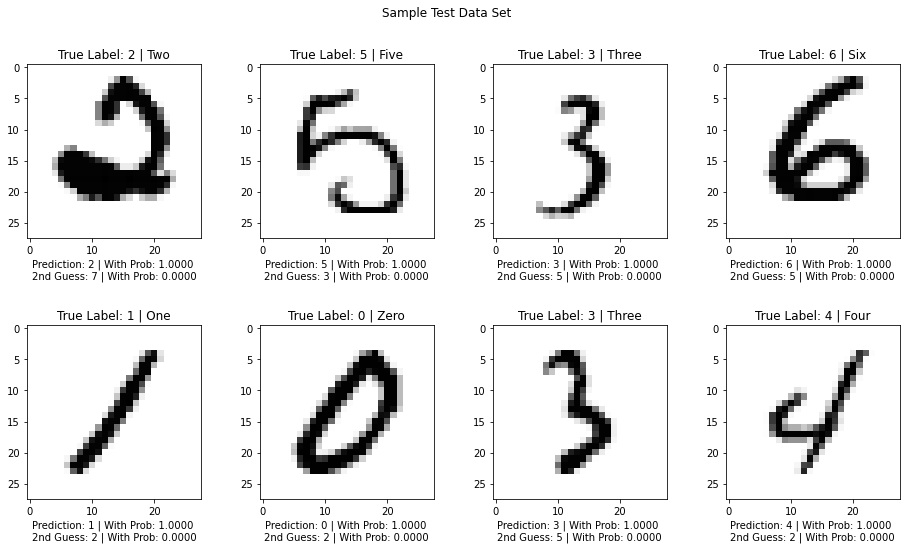
Appendix G: Confusion Matrix of Test Set Prediction



Appendix H: Samples of images from Training set along with their Predictions

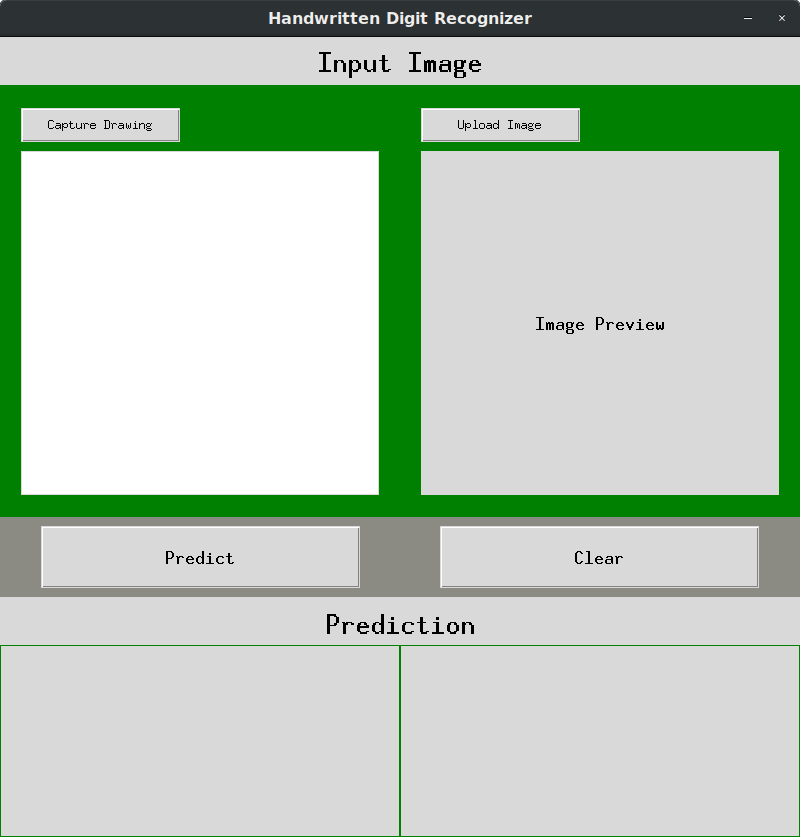


Appendix I: Samples of images from Dev set along with their Predictions

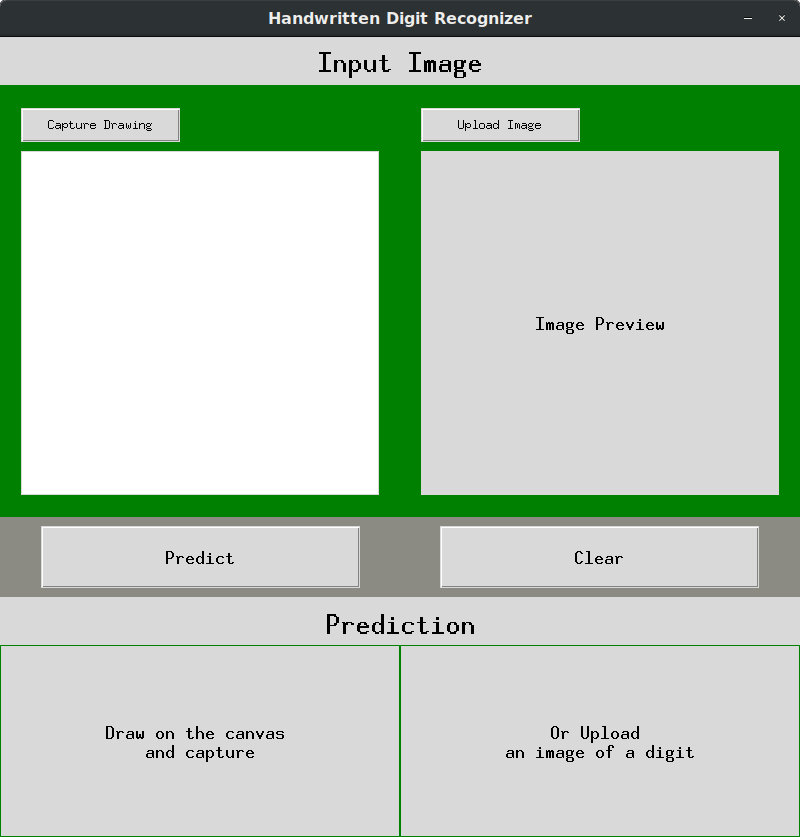


Appendix J: Samples of images from Test set along with their Predictions

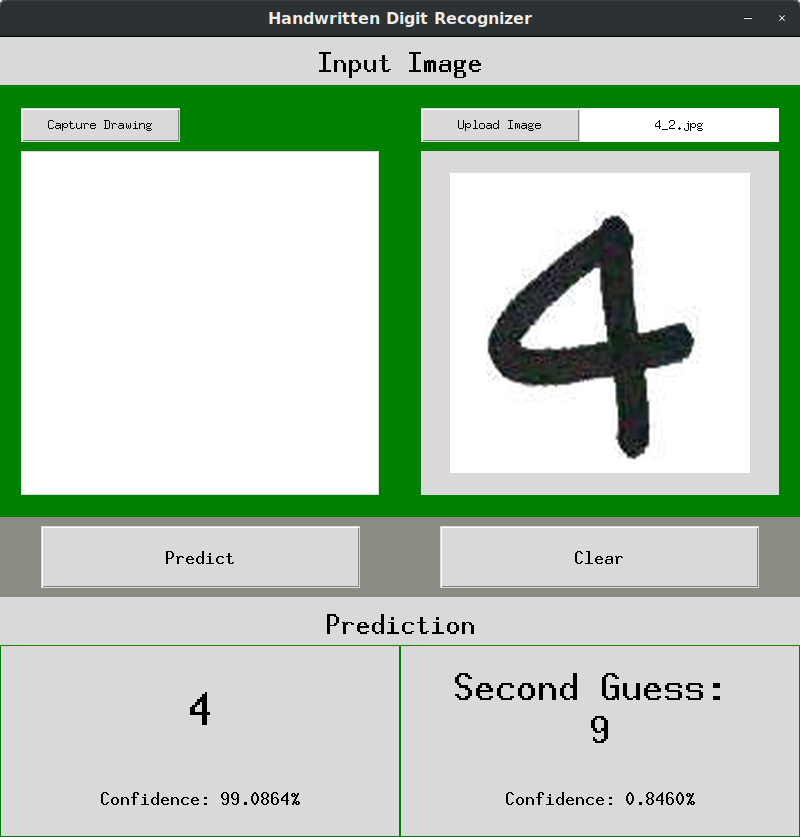
## UI Screenshots:



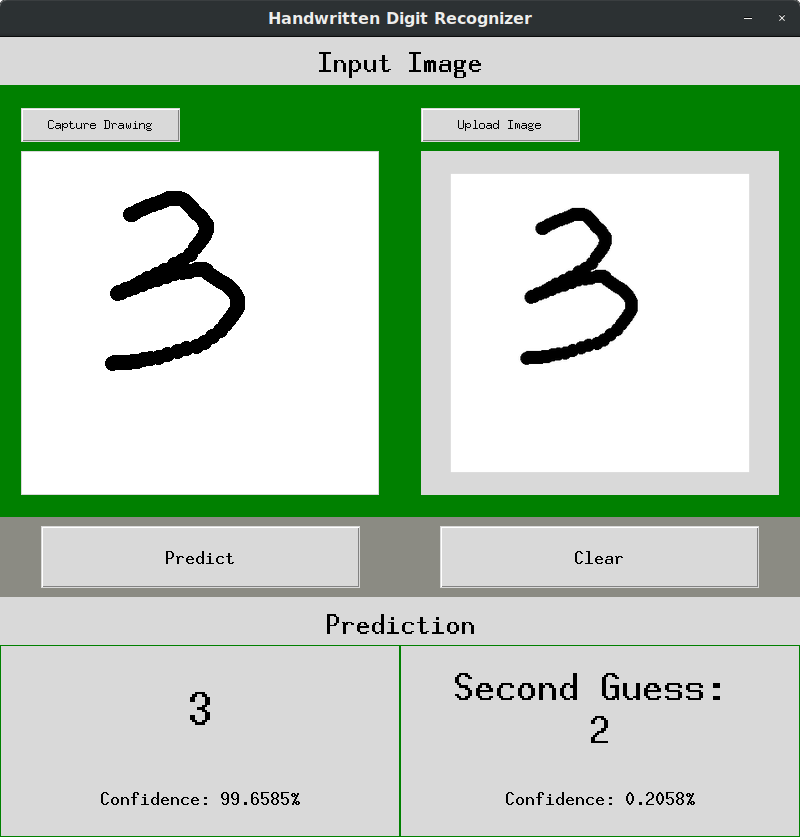
Appendix K: Screenshot of UI in its initial phase



Appendix L: Screenshot of UI when Prediction Button is clicked without uploading or capturing the drawn image



Appendix M: Screenshot of UI Prediction the uploaded image



Appendix N: : Screenshot of UI Predicting the drawn image

## Source Code:

Below is the source code implementation of MLP in the project.

**# Activation Functions**

def relu(Z):

A = np.maximum(0.0, Z)

cache = Z # storing Z for later use during back propagation

assert (A.shape == Z.shape)

return A, cache

def relu\_grad(dA, cache):

Z = cache

dZ = np.array(dA, copy=True)

dZ[Z <= 0] = 0 # implementing integrated form of (gradiant of ReLU function \* gradient of the loss function)

assert (dZ.shape == Z.shape)

return dZ

def softmax(Z):

shift = Z - np.max(Z, axis = 0)# Avoiding underflow or overflow errors due to floating point instability in softmax

t = np.exp(shift)

A = np.divide(t, np.sum(t, axis=0))

cache = Z

assert (A.shape == Z.shape)

return A, cache

**# initializing the layers**

def init\_layers(input\_shape, output\_shape, hidden\_layers):

input\_nodes = input\_shape

output\_nodes = output\_shape

layers\_dim = [input\_nodes]

for i in hidden\_layers:

layers\_dim.append(i)

layers\_dim.append(output\_nodes)

return layers\_dim

**# initializing parameters**

def init\_parameters(layers\_dim, initialization="random"):

L = len(layers\_dim)

params = {}

for l in range(1, L):

# initializing Weights

if initialization == "he":

# he-initialization

params['W' + str(l)] = np.random.randn(layers\_dim[l], layers\_dim[l - 1]) \* np.sqrt(

np.divide(2, layers\_dim[l - 1]))

elif initialization == "random":

# random initialization scaled by 0.01

params['W' + str(l)] = np.random.randn(layers\_dim[l], layers\_dim[l - 1]) \* 0.01

else:

raise ValueError("Initialization must be 'random' or 'he'")

# initializing biases

params['b' + str(l)] = np.zeros((layers\_dim[l], 1))

assert (params['W' + str(l)].shape == (

layers\_dim[l], layers\_dim[l - 1])), "Dimention of W mismatched in init\_params function"

assert (params['b' + str(l)].shape == (layers\_dim[l], 1)), "Dimention of b mismatched in init\_params function"

return params

**# initializing hyper parameters**

def init\_hyperParams(alpha, num\_epoch, minibatch\_size=64, lambd=0, keep\_probs=[], beta1=0.9, beta2=0.999, epsilon=1e-8):

hyperParams = {'learning\_rate' : alpha,

'num\_epoch' : num\_epoch,

'mini\_batch\_size': minibatch\_size,

'lambda' : lambd,

'keep\_probs' : keep\_probs,

'beta1' : beta1,

'beta2' : beta2,

'epsilon' : epsilon

}

return hyperParams

**# Forward Propagation**

**## forward sum**

def forward\_sum(A\_prev, W, b):

m = A\_prev.shape[1]

Z = np.dot(W, A\_prev) + b

cache = (A\_prev, W, b)

assert (Z.shape == (W.shape[0], m)), "Dimention of Z mismatched in forward\_prop function"

return Z, cache

**## forward Activation**

def forward\_activation(A\_prev, W, b, activation):

if activation == 'relu':

Z, sum\_cache = forward\_sum(A\_prev, W, b)

A, activation\_cache = relu(Z)

elif activation == 'softmax':

Z, sum\_cache = forward\_sum(A\_prev, W, b)

A, activation\_cache = softmax(Z)

elif activation == "tanh":

# Z, sum\_cache = forward\_sum(A\_prev,W,b)

# A, activation\_cache = tanh(Z)

pass

cache = (sum\_cache, activation\_cache)

assert (A.shape == Z.shape), "Dimention of A mismatched in forward\_activation function"

return A, cache

**# dropout for individual layer**

def forward\_dropout(A, keep\_probs):

# implementing dropout

D = np.random.rand(A.shape[0], A.shape[1])

D = (D < keep\_probs).astype(int)

A = np.multiply(A, D)

A = np.divide(A, keep\_probs)

dropout\_mask = D

assert (dropout\_mask.shape == A.shape), "Dimention of dropout\_mask mismatched in forward\_dropout function"

return A, dropout\_mask

**## forward prop for L layers**

def forward\_prop(X, parameters, keep\_probs=[], regularizer=None):

caches = []

A = X

L = len(parameters) // 2

num\_class = parameters["W" + str(L)].shape[0]

dropout\_masks = []

# len(keep\_probs) == L-1: no dropouts in the Output layer, no dropout at all for prediction

if regularizer == "dropout":

assert (len(keep\_probs) == L - 1)

for l in range(1, L):

A\_prev = A

A, cache = forward\_activation(A\_prev, parameters['W' + str(l)], parameters['b' + str(l)], activation='relu')

caches.append(cache)

if regularizer == "dropout":

A, dropout\_mask = forward\_dropout(A, keep\_probs[l - 1])

dropout\_masks.append(dropout\_mask)

else:

pass

AL, cache = forward\_activation(A, parameters['W' + str(L)], parameters['b' + str(L)], activation='softmax')

caches.append(cache)

assert (AL.shape == (num\_class, X.shape[1])), "Dimention of AL mismatched in forward\_prop function"

return AL, caches, dropout\_masks

**# compute Cross entropy cost**

def softmax\_cross\_entropy\_cost(AL, Y, caches, lambd=0, regularizer=None, from\_logits=False):

L = len(caches)

m = Y.shape[1]

# cost computation from logit

# ref link : https://www.d2l.ai/chapter\_linear-networks/softmax-regression-concise.html

if from\_logits == True:

z = caches[-1][-1] # retriving the logit(activation cache) of the last layer from the caches

z = z - np.max(z, axis=0) # calculating negative z for avoiding numerical overflow in exp computation

logit\_log = z - np.log(np.sum(np.exp(z), axis=0)) # calculating the log of the softmax to feed into cost

cost = -(1. / m) \* np.sum(np.sum(np.multiply(Y, logit\_log), axis=0, keepdims=True))

else:

cost = -(1. / m) \* np.sum(np.sum(np.multiply(Y, np.log(AL + 1e-8)), axis=0,

keepdims=True)) # add very small number 1e-8 to avoid log(0)

if regularizer == "l2":

norm = 0

for l in range(L):

current\_cache = caches[l]

sum\_cache, \_ = current\_cache

\_, W, \_ = sum\_cache

norm += np.sum(np.square(W))

L2\_cost = (lambd / (2 \* m)) \* norm

cost = cost + L2\_cost

else:

pass

cost = np.squeeze(cost) # Making sure your cost's shape is not returned as ndarray

assert (cost.shape == ()), "Dimention of cost mismatched in softmax\_cross\_entropy\_cost function"

return cost

**# Back Propagation**

**## calculating backward gradient**

def backward\_grad(dZ, cache, lambd, regularizer):

A\_prev, W, b = cache

m = A\_prev.shape[1]

if regularizer == "l2":

dW = (1 / m) \* np.dot(dZ, A\_prev.T) + np.multiply(np.divide(lambd, m), W)

else:

dW = (1 / m) \* np.dot(dZ, A\_prev.T)

db = (1 / m) \* np.sum(dZ, axis=1, keepdims=True)

dA\_prev = np.dot(W.T, dZ)

assert (dW.shape == W.shape), "Dimention of dW mismatched in backward\_grad function"

assert (db.shape == b.shape), "Dimention of db mismatched in backward\_grad function"

assert (dA\_prev.shape == A\_prev.shape), "Dimention of dA\_prev mismatched in backward\_grad function"

return dA\_prev, dW, db

**## calculating backward activation**

def backward\_activation(dA, cache, lambd, regularizer, activation):

sum\_cache, activation\_cache = cache

if activation == "relu":

dZ = relu\_grad(dA, activation\_cache)

dA\_prev, dW, db = backward\_grad(dZ, sum\_cache, lambd, regularizer=regularizer)

elif activation == "softmax":

dZ = dA

dA\_prev, dW, db = backward\_grad(dZ, sum\_cache, lambd, regularizer=regularizer)

elif activation == "tanh":

pass

# dZ = tanh\_grad(dA,activation\_cache)

# dA\_prev, dW, db = backward\_grad(dZ, sum\_cache, lambd, regularizer = regularizer)

return dA\_prev, dW, db

**# implementing backward dropout**

def backward\_dropout(dA\_prev\_temp, D, keep\_prob):

dA\_prev = np.multiply(dA\_prev\_temp, D)

dA\_prev = np.divide(dA\_prev, keep\_prob)

return dA\_prev

**# back prop foL layers**

def backward\_prop(AL, Y, caches, dropout\_masks=[], keep\_probs=[], lambd=0, regularizer=None):

grads = {}

L = len(caches) # the number of layers

Y = Y.reshape(AL.shape) # after this line, Y is the same shape as AL

dA = np.subtract(AL, Y)

current\_cache = caches[L - 1]

grads["dA" + str(L - 1)], grads["dW" + str(L)], grads["db" + str(L)] = backward\_activation(dA, current\_cache, lambd=lambd, regularizer=regularizer,

activation='softmax')

for l in reversed(range(L - 1)):

current\_cache = caches[l]

if regularizer == "dropout":

# implementing dropout

D = dropout\_masks[l]

dA\_prev\_temp = backward\_dropout(grads["dA" + str(l + 1)], D, keep\_probs[l])

dA\_prev, dW\_temp, db\_temp = backward\_activation(dA\_prev\_temp, current\_cache, lambd=lambd,

regularizer=regularizer, activation='relu')

else:

dA\_prev, dW\_temp, db\_temp = backward\_activation(grads["dA" + str(l + 1)], current\_cache, lambd=lambd,

regularizer=regularizer, activation='relu')

grads["dA" + str(l)] = dA\_prev

grads["dW" + str(l + 1)] = dW\_temp

grads["db" + str(l + 1)] = db\_temp

return grads

**# Update Parameters**

**## Initializing Adam**

def initialize\_adam(parameters):

L = len(parameters) // 2

v = {}

s = {}

for l in range(L):

v["dW" + str(l + 1)] = np.zeros(parameters["W" + str(l + 1)].shape)

v["db" + str(l + 1)] = np.zeros(parameters["b" + str(l + 1)].shape)

s["dW" + str(l + 1)] = np.zeros(parameters["W" + str(l + 1)].shape)

s["db" + str(l + 1)] = np.zeros(parameters["b" + str(l + 1)].shape)

return v, s

**## update Parameters**

def update\_parameters(parameters, grads, learning\_rate, optimizer="bgd", beta1=0, beta2=0, epsilon=0, v={}, s={}, t=0):

L = len(parameters) // 2

v\_corrected = {}

s\_corrected = {}

for l in range(L):

if optimizer == 'adam':

# Moving average of the gradients.

v["dW" + str(l + 1)] = np.add(beta1 \* v["dW" + str(l + 1)], (1 - beta1) \* grads["dW" + str(l + 1)])

v["db" + str(l + 1)] = np.add(beta1 \* v["db" + str(l + 1)], (1 - beta1) \* grads["db" + str(l + 1)])

# Compute bias-corrected first moment estimate.

v\_corrected["dW" + str(l + 1)] = np.divide(v["dW" + str(l + 1)], (1 - np.power(beta1, t)))

v\_corrected["db" + str(l + 1)] = np.divide(v["db" + str(l + 1)], (1 - np.power(beta1, t)))

# Moving average of the squared gradients.

s["dW" + str(l + 1)] = np.add(beta2 \* s["dW" + str(l + 1)],

(1 - beta2) \* np.square(grads["dW" + str(l + 1)]))

s["db" + str(l + 1)] = np.add(beta2 \* s["db" + str(l + 1)],

(1 - beta2) \* np.square(grads["db" + str(l + 1)]))

# Compute bias-corrected second raw moment estimate.

s\_corrected["dW" + str(l + 1)] = np.divide(s["dW" + str(l + 1)], (1 - np.power(beta2, t)))

s\_corrected["db" + str(l + 1)] = np.divide(s["db" + str(l + 1)], (1 - np.power(beta2, t)))

# Update parameters.

parameters["W" + str(l + 1)] = np.subtract(parameters["W" + str(l + 1)],

learning\_rate \* np.divide(v\_corrected["dW" + str(l + 1)], np.sqrt(s\_corrected["dW" + str(l + 1)]) + epsilon))

parameters["b" + str(l + 1)] = np.subtract(parameters["b" + str(l + 1)], learning\_rate \* np.divide(v\_corrected["db" + str(l + 1)], np.sqrt(s\_corrected["db" + str(l + 1)]) + epsilon))

else:

parameters["W" + str(l + 1)] = parameters["W" + str(l + 1)] - (learning\_rate \* grads["dW" + str(l + 1)])

parameters["b" + str(l + 1)] = parameters["b" + str(l + 1)] - (learning\_rate \* grads["db" + str(l + 1)])

return parameters, v, s

**# Evaluating the model using acc and loss**

def evaluate(X, Y, parameters):

m = Y.shape[1]

# predicting output using fordward propogation

probas, caches, \_ = forward\_prop(X, parameters)

# computing loss

loss = softmax\_cross\_entropy\_cost(probas, Y, caches, from\_logits=True)

# deriving the predictrueted labels

true\_labels = np.argmax(Y, axis=0).reshape(1, m)

# deriving the predicted labels

predicted\_labels = np.argmax(probas, axis=0).reshape(1, m)

# identifing correctly predicted labels

correct\_prediction = np.equal(predicted\_labels, true\_labels)

# computing accuracy

num\_correct\_prediction = np.sum(correct\_prediction)

accuracy = (num\_correct\_prediction / m)

return accuracy, loss

**# Final Model Training**

def train(training\_data, validation\_data, layers\_dim, hyperParams, initialization="random", optimizer='bgd',

regularizer=None, verbose=3, patience=None, step\_decay=None):

# unpacking the hyperparameters

learning\_rate = hyperParams['learning\_rate']

num\_epoch = hyperParams['num\_epoch']

b1 = hyperParams['beta1']

b2 = hyperParams['beta2']

ep = hyperParams['epsilon']

lambd = hyperParams['lambda']

keep\_probs = hyperParams['keep\_probs']

# unpacking the data

X\_train, Y\_train = training\_data

X\_dev, Y\_dev = validation\_data

# setting up necessary variables for early stopping

if patience != None and patience != 0:

# configuring path to save the intermediate best parameters

path = "temp/"

if not os.path.exists(path):

os.makedirs(path)

filename = "best\_param\_intermediate"

early\_stop\_count = 0 # count variable for counting consucative the epochs without progress

max\_val\_acc = 0 # for keeping track of maximum validation accuracy

# initializing the training variables

seed = 1

m = Y\_train.shape[1]

train\_accs = [] # for keeping track of training accuracy

val\_accs = [] # for keeping track of Validation accuracy

train\_losses = [] # for keeping track of training loss

val\_losses = [] # for keeping track of Validation loss

# selecting the minibatch size for each optimizer

if optimizer == 'sgd':

mini\_batch\_size = 1

elif optimizer == 'bgd':

mini\_batch\_size = m

elif optimizer == 'mgd' or optimizer == 'adam':

mini\_batch\_size = hyperParams['mini\_batch\_size']

else:

raise ValueError("Optimizer value out of scope")

# initializing the model parameters

parameters = init\_parameters(layers\_dim, initialization)

# initializing adam parameters, used only when optimizer = 'adam'

t = 0

v, s = initialize\_adam(parameters)

train\_tic = time.time() # for calculating entire training time

print("Training The Model...")

# Gradient Descent begins

for i in range(num\_epoch):

seed += 1

time\_trained = 0 # for computing training time of each epoch

batch\_times = [] # for accumulating the training time of each batch

accs = [] # for tracking batch training accuracy

losses = [] # for tracking batch training loss

# learning rate scheduling

if step\_decay != None and step\_decay != 0:

if i % step\_decay == 0:

decay\_rate = learning\_rate / ((i + 1) / num\_epoch)

learning\_rate = learning\_rate\_scledule(learning\_rate, i, decay\_rate)

if learning\_rate <= 0.0001: learning\_rate = 0.0001

if verbose > 0:

if step\_decay != None and step\_decay != 0:

print("\nEpoch %d/%d: learning rate - %.6f" % (i + 1, num\_epoch, learning\_rate))

else:

print("\nEpoch %d/%d" % (i + 1, num\_epoch))

# augmenting the dataset online

data\_generator(X\_train, Y\_train, batch\_size=2048, aug\_count=8, pre\_process\_data=True)

aug\_images, aug\_labels = load\_augmented\_data()

# generating minibatches of the augmented dataset

minibatches = rand\_mini\_batches(aug\_images, aug\_labels, mini\_batch\_size, seed)

total\_minibatches = len(minibatches)

del aug\_images, aug\_labels

for ind, minibatch in enumerate(minibatches):

batch\_tic = time.time() # for calculating time of an epoch cycle

# retriving minibatch of X and Y from training set

(minibatch\_X, minibatch\_Y) = minibatch

# forward Propagation

AL, caches, dropout\_masks = forward\_prop(minibatch\_X, parameters, keep\_probs=keep\_probs,

regularizer=regularizer)

# Computing cross entropy cost

cross\_entropy\_cost = softmax\_cross\_entropy\_cost(AL, minibatch\_Y, caches, lambd=lambd,

regularizer=regularizer,

from\_logits=True) # accumulating the batch costs

# Backward Propagation

grads = backward\_prop(AL, minibatch\_Y, caches, dropout\_masks=dropout\_masks, keep\_probs=keep\_probs,

lambd=lambd, regularizer=regularizer)

# Updating parameters

t += 1

parameters, v, s = update\_parameters(parameters, grads, learning\_rate, optimizer=optimizer, beta1=b1,

beta2=b2, epsilon=ep, v=v, s=s, t=t)

# Calculating training time for each batch

batch\_times.append(time.time() - batch\_tic)

time\_trained = np.sum(batch\_times)

# calculating training progress

per = ((ind + 1) / total\_minibatches) \* 100

inc = int(per // 10) \* 2

# calculating accuracy and loss of the training batch

acc, loss = evaluate(minibatch\_X, minibatch\_Y, parameters)

accs.append(acc)

losses.append(loss)

# Verbosity 0: Silent mode

# Verbosity 1: Epoch mode

# Verbosity 2: Progress bar mode

# Verbosity 3 or greater: Metric mode

if verbose == 2:

print("%d/%d [%s>%s %.0f%%] - %.2fs" % (

ind + 1, total\_minibatches, '=' \* inc, '.' \* (20 - inc), per, time\_trained), end='\r')

elif verbose > 2:

print("%d/%d [%s>%s %.0f%%] - %.2fs | loss: %.4f | acc: %.4f" % (

ind + 1, total\_minibatches, '=' \* inc, '.' \* (20 - inc), per, time\_trained, np.mean(losses),

np.mean(accs)), end='\r')

del caches, minibatch\_X, minibatch\_Y, minibatch, grads # clearing caches

# ----------------------------------------------batch ends-----------------------------------------

# accumulating the acc and loss of the last iteration of each epoch

train\_accs.append(np.mean(accs))

train\_losses.append(np.mean(losses))

# evaluating the model using validation accuracy and loss

val\_acc, val\_loss = evaluate(X\_dev, Y\_dev, parameters)

val\_accs.append(val\_acc)

val\_losses.append(val\_loss)

time\_per\_batch = int(np.mean(batch\_times) \* 1000)

if verbose == 2:

print("%d/%d [%s 100%%] - %.2fs %dms/step" % (

total\_minibatches, total\_minibatches, '=' \* 20, time\_trained, time\_per\_batch), end='\r')

elif verbose > 2:

print("%d/%d [%s 100%%] - %.2fs %dms/step | loss: %.4f | acc: %.4f | val\_loss: %.4f | val\_acc: %.4f" % (

total\_minibatches, total\_minibatches, '=' \* 20, time\_trained, time\_per\_batch, np.mean(losses),

np.mean(accs), val\_loss, val\_acc), end='\r')

# early stopping implementation

if patience != None and patience != 0:

# getting the best val accuracy

if val\_acc >= max\_val\_acc:

max\_val\_acc = val\_acc

print("\nImprovement in validation accuracy found. Saving the corresponding parameters...")

save\_model(path + filename, parameters)

early\_stop\_count = 0

else:

early\_stop\_count += 1

if early\_stop\_count == patience:

print(

"\n\nSince the Val Acc didn't increase for last %d epochs, Training is halted returning the best parameters obtained." % patience)

break;

del minibatches

gc.collect()

# -------------------------------------------Gradient Descent ends--------------------------------

hrs, mins, secs, ms = convert\_time((time.time() - train\_tic) \* 1000)

print("\n\nTotal Training Time = %dhr %dmins %dsecs %.2fms" % (hrs, mins, secs, ms))

# loading the best parameters

if patience != None and patience != 0:

parameters = load\_model(path + filename)

os.remove(path + filename) # removing temporary file

# saving weights to wandb

np.save(path + "weights", parameters)

wandb.save(path + "weights.npy")

history = {"parameters" : parameters,

"accuracy" : train\_accs,

"loss" : train\_losses,

"val\_accuracy": val\_accs,

"val\_loss" : val\_losses

}

return history

**# Making Predictions**

def predict(X, parameters, second\_guess=False):

prediction = {}

# Computing the Output predictions.

# no keep\_probs : no dropout during prediction

probas, caches, \_ = forward\_prop(X, parameters)

# getting the number of examples

m = probas.shape[1]

# deriving the predicted labels with their probabilities

predicted\_labels = np.argmax(probas, axis=0).reshape(1, m)

predicted\_prob = np.max(probas, axis=0).reshape(1, m)

# Computing the second guess

if second\_guess == True:

second\_max = np.array(probas, copy=True)

second\_max[predicted\_labels, np.arange(m)] = 0 # zeroing out the first max prediction

sec\_predicted\_labels = np.argmax(second\_max, axis=0).reshape(1, m) # selecting the second max predicted label

sec\_predicted\_prob = np.max(second\_max, axis=0).reshape(1, m) # selecting the second max prediction

prediction["Second Prediction"] = [sec\_predicted\_labels, sec\_predicted\_prob]

prediction["First Prediction"] = [predicted\_labels, predicted\_prob]

return prediction