BASAVARAJESWARI GROUP OF INSTITUTIONS

BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

NACC Accredited Institution*

(Recognized by Govt. of Karnataka, approved by AICTE, New Delhi & Affiliated to Visvesvaraya Technological University, Belagavi)
"JnanaGangotri" Campus, No.873/2, Ballari-Hospet Road, Allipur, Ballar1-583 104 (Karnataka) (India)
Ph: 08392 – 237100 / 237190, Fax: 08392 – 237197

DEPARTMENT OF CSE-DATA SCIENCE

A Mini-Project Report On

"HANDWRITTEN DIGIT CLASSIFICATION USING ANN"

A report submitted in partial fulfillment of the requirements for the

NEURAL NETWORK AND DEEP LEARNING

Submitted By

DARSHAN M **USN: 3BR22CD011**

Under the Guidance of
Mr. Azhar Baig

Asst. Professor

Dept of CSE (DATA SCIENCE),
BITM, Ballari

Visvesvaraya Technological University

Belagavi, Karnataka 2025-2026

BASAVARAJESWARI GROUP OF INSTITUTIONS

BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

NACC Accredited Institution*

(Recognized by Govt. of Karnataka, approved by AICTE, New Delhi & Affiliated to Visvesvaraya Technological University, Belagavi)

"JnanaGangotri" Campus, No.873/2, Ballari-Hospet Road, Allipur, Ballari-583 104 (Karnataka) (India)

Ph: 08392 – 237100 / 237190, Fax: 08392 – 237197

DEPARTMENT OF CSE (DATA SCIENCE)

CERTIFICATE

This is to certify that the Mini Project of NEURAL NETWORK AND DEEP LEARNING title "HANDWRITTEN DIGIT CLASSIFICATION USING ANN MODEL" has been successfully presented by DARSHAN M 3BR22CD011 student of semester B.E for the partial fulfillment of the requirements for the award of Bachelor Degree in CSE(DS) of the BALLARI INSTITUTE OF TECHNOLOGY& MANAGEMENT, BALLARI during the academic year 2025-2026.

It is certified that all corrections and suggestions indicated for internal assessment have been incorporated in the report deposited in the library. The Mini Project has been approved as it satisfactorily meets the academic requirements prescribed for the Bachelor of Engineering Degree. The work presented demonstrates the required level of technical understanding, research depth, and documentation standards expected for academic evaluation.

Signature of Coordinators

Mr. Azhar Baig

Ms. Chaithra B M

Signature of HOD

Dr. Aradhana D

ABSTRACT

Handwritten digit recognition is one of the most widely used applications of machine learning in areas such as postal automation, bank cheque processing, and digital document reading. This project presents a neural network–based approach for classifying handwritten digits using the Digits Dataset, which consists of 8×8 grayscale images. The system uses a Multilayer Perceptron (MLP) classifier that learns patterns from pixel values through supervised training. The dataset is preprocessed, normalized, and divided into training and testing sets to ensure efficient learning and accurate model evaluation. The trained model achieves high accuracy and demonstrates strong performance through metrics such as classification reports, confusion matrices, and loss curve visualizations. The final model and scaler are saved for predicting new handwritten digit inputs. Overall, this project successfully implements an end-to-end digit recognition system and provides a foundation for more advanced deep-learning applications in image classification.

The project adopts a structured machine learning pipeline that includes data preprocessing, feature scaling, model training, evaluation, and visualization. StandardScaler is used to normalize the pixel intensities, which significantly improves the stability and convergence of the MLP classifier. The chosen network architecture, consisting of multiple hidden layers, enables the model to capture complex relationships between input pixels and their corresponding digit labels. Performance evaluation using accuracy, precision, recall, and F1-score confirms that the model generalizes well on unseen data. Graphical outputs, including the training loss curve and confusion matrix, further support the effectiveness of the proposed approach.

The completed system demonstrates how a relatively simple neural network can achieve reliable results on small-scale image datasets, making it suitable for real-time and resource-constrained environments. The modular design also allows the system to be extended easily to more complex datasets such as MNIST or integrated into larger OCR applications. With its ability to preprocess data, train efficiently, and deliver accurate predictions, the project lays a foundation for exploring more advanced architectures like Convolutional Neural Networks (CNNs) or hybrid models. This work highlights the potential of neural networks in pattern recognition and showcases their practical application in digit classification tasks.

ACKNOWLEDGEMENT

The satisfactions that accompany the successful completion of our mini project on HANDWRITTEN DIGIT CLASSIFICATION USING ANN MODEL would be incomplete without the mention of people who made it possible, whose noble gesture, affection, guidance, encouragement and support crowned my efforts with success. It is our privilege to express our gratitude and respect to all those who inspired us in the completion of our mini-project.

I am extremely grateful to our Guide **Mr. Azhar Baig** for their noble gesture, support co-ordination and valuable suggestions given in completing the mini-project. I also thank **Dr. Aradhana D,** H.O.D. Department of CSE(DS), for his co-ordination and valuable suggestions given in completing the mini-project. I also thank Principal, Management and non-teaching staff for their co-ordination and valuable suggestions given to us in completing the Mini project.

Name	USN
DARSHAN M	3BR22CD011

TABLE OF CONTENTS

1. INTRODUCTION

Handwritten digit recognition is one of the most fundamental and widely studied problems in the field of machine learning and neural networks. It involves teaching a computer to automatically identify digits written by humans, similar to how humans read numbers on paper. This problem is important in many real-world applications such as postal code recognition, bank cheque processing, form automation, and optical character recognition (OCR) systems.

In this project, a machine learning model is developed to classify handwritten digits from 0 to 9 using the Digits Dataset, which contains 8×8 grayscale images of handwritten numbers. Each image is represented as 64 numerical pixel values, making it suitable for lightweight neural network training.

The project follows a complete machine learning workflow:

Data loading and preprocessing – cleaning, scaling, and splitting the dataset.

Model building using a Multilayer Perceptron (MLP) neural network.

Training and evaluation using accuracy, classification report, and confusion matrix.

Visualization of results with loss curves and sample predictions.

Saving the trained model for future use without retraining.

The aim of this project is to demonstrate how neural networks can learn meaningful patterns from image data and accurately classify handwritten digits. With proper preprocessing and training, the model achieves high accuracy and performs reliably on unseen test images. This project serves as a practical introduction to neural networks, image classification, and supervised machine learning.

1.1 Problem Statement

The challenge addressed in this project is to develop an automated system capable of accurately recognizing handwritten digits, which naturally vary in shape, thickness, and writing style from person to person. Manual interpretation is slow and prone to errors, especially in applications such as postal sorting, bank cheque processing, and digital form reading. Therefore, the objective is to build a neural network model that can learn meaningful patterns from pixel-based image data and classify handwritten digits (0–9) with high accuracy. Using the Digits

Dataset and a Multilayer Perceptron (MLP) neural network, the project aims to create a reliable, efficient, and scalable digit recognition system.

1.2 Scope of the Project

The scope of this project includes building a complete handwritten digit recognition system using the Digits Dataset and a Multilayer Perceptron (MLP) neural network. It covers all essential stages such as data preprocessing, model design, training, evaluation, visualization of results, and saving the trained model for future use. The project focuses on demonstrating how neural networks can learn patterns from image data and accurately classify digits from 0 to 9. While it is limited to 8×8 grayscale images and numerical digit recognition, the techniques and workflow used in this project can be extended to more complex datasets, larger neural network models, and real-world applications like OCR systems, document scanning, and automated data entry.

1.3 Objectives

- To develop a neural network model that can accurately classify handwritten digits from 0 to 9.
- To preprocess the dataset by normalizing and scaling pixel values for effective training.
- To train and evaluate a Multilayer Perceptron (MLP) classifier using accuracy, classification report, and confusion matrix.
- To visualize the model's performance through loss curves and sample predictions, and save the trained model for future use.

2. LITERATURE SURVEY

[1] Sharma et al. (2024) showed that Multilayer Perceptron (MLP) models can achieve high accuracy in handwritten digit recognition when proper preprocessing and scaling techniques are used. Their results proved that even simple neural networks work well on small grayscale digit images.

[2] Kumar et al. (2023) compared different machine-learning models such as SVM, k-NN, and MLP for digit recognition. They found that neural networks, especially MLP, performed more consistently on low-resolution datasets like the Digits Dataset.

[3] Hassan et al. (2024) implemented a CNN-based classifier for handwritten digits and reported higher accuracy than traditional models. They highlighted that although CNNs perform better, MLPs remain effective for small datasets due to lower computational cost.

[4] Patel and Sinha (2023) evaluated various preprocessing techniques for digit classification and found that normalization and feature scaling significantly improve model accuracy. Their study confirmed that clean inputs help neural networks learn patterns more efficiently.

[5] Ali et al. (2024) explored lightweight neural networks for handwritten digit recognition on low-power devices. Their work showed that compact MLP models can run efficiently without GPUs while still giving accurate predictions.

[6] Roy et al. (2023) conducted a performance comparison of training algorithms for neural networks used in digit recognition. They concluded that optimizers like Adam and early stopping greatly improve the learning stability of MLP models.

[7] Prakash et al. (2025) designed a hybrid digit recognition system using MLP and basic image filtering techniques. Their results demonstrated that combining simple preprocessing with an MLP classifier leads to better accuracy on handwritten digit datasets.

3. SYSTEM REQUIREMENTS

The system requirements for developing the handwritten digit recognition model include both software and hardware components needed for data preprocessing, model training, evaluation, and visualization. The software environment is built using Python along with essential machine-learning libraries such as Scikit-learn for the MLP classifier and preprocessing, Pandas and NumPy for dataset handling, and Matplotlib for plotting graphs like loss curves and confusion matrices. A development platform such as Jupyter Notebook, Google Colab, or VS Code is used to write, run, and debug the code interactively.

On the hardware side, the project can run smoothly on a standard personal computer since the Digits Dataset is relatively small and the MLP model is lightweight. A basic laptop or desktop with at least 4 GB of RAM is sufficient, though 8 GB is recommended for faster computations and smoother multitasking. A multi-core CPU helps in speeding up training and evaluation, while a GPU is not mandatory for this project. Overall, the system requirements are modest, making the handwritten digit recognition project easy to implement on most modern computers.

3.1 Software Requirements

Python 3.8 or above

NumPy

Pandas

Scikit-learn

Matplotlib

Jupyter Notebook / Google Colab / VS Code

Windows / Linux / macOS operating system

3.2 Hardware Requirements

Minimum 4 GB RAM

Recommended 8 GB RAM

Dual-core or higher processor

At least 1 GB free storage space

GPU optional (not required for MLP-based model)

3.3 Functional Requirements

The system must load and read the digits dataset from a CSV file.

- It must preprocess and standardize the pixel features.
- The system must build and train an MLP neural network for digit classification.
- It must evaluate model performance using accuracy, classification report, and confusion matrix.
- The system must generate visual outputs such as loss curves, confusion matrix heatmaps, and sample prediction images.
- It must allow prediction of new handwritten digit inputs using the trained model.

3.4 Non-Functional Requirements

- The system should provide accurate and reliable digit predictions.
- It should produce clear and user-friendly text and graphical outputs.
- The system must execute efficiently on basic hardware without long delays.
- It should remain stable for different input samples and minor data noise.
- The system should be easy to maintain, modify, and extend to larger datasets or models.
- The results should be interpretable through metrics, graphs, and visual examples.

4. DESCRIPTION OF MODULES

The handwritten digit recognition system is divided into multiple modules, each representing a specific stage in the machine-learning workflow. These modules work together to ensure smooth data preprocessing, neural network training, evaluation, and visualization. Below is the complete module-wise description tailored to your MLP-based digit classification project.

4.1 Data Preprocessing Module

This module loads the Digits Dataset from the CSV file and prepares it for further analysis. It ensures that all pixel values are in proper numeric format and handles reshaping or flattening operations if required. Since the dataset involves 8×8 grayscale digit images, this module extracts the 64 pixel features and separates them from the target labels. It ensures that the input data is clean, structured, and ready for scaling and training.

4.2 Feature Scaling Module

Pixel values in the dataset vary in intensity and scale, which can affect the performance of the neural network. This module applies StandardScaler to normalize all pixel values, ensuring that the features have a consistent distribution. Scaling improves training speed, model stability, and overall accuracy. It also ensures that no single feature dominates the learning process due to higher numerical magnitude.

4.3 Data Splitting Module

This module divides the dataset into training and testing subsets using an 80:20 ratio. The training set is used to teach the neural network, while the test set evaluates its performance on unseen samples. Stratified splitting is applied to maintain equal distribution of digits across both sets. The module ensures that the model is trained on one portion of the data and its accuracy is measured fairly on a separate set.

4.4 MLP Model Building Module

This module constructs the Multilayer Perceptron (MLP) classification model. It defines the input layer based on 64 pixel features, hidden layers with ReLU activation, and the output layer with 10 neurons representing digits from 0 to 9. The module sets parameters such as number of neurons (e.g., 128–64), activation functions, optimizer, and the loss function. The model is compiled to prepare it for training.

4.5 Model Training Module

Once the model architecture is defined, this module trains the MLP neural network using the scaled training data. It sets important training parameters such as maximum iterations, early stopping, and learning rate. As the model trains, it learns the patterns from the pixel data and improves its classification accuracy. The module tracks loss values for each iteration, which helps in later visualization and analysis.

4.6 Model Evaluation Module

This module evaluates the trained model on the test data. It calculates metrics such as accuracy, precision, recall, and F1-score for each digit class. It also generates a classification report that summarizes model performance. A confusion matrix is created to show how well the model distinguishes between digits, identifying which digits are correctly recognized and which ones are misclassified.

4.7 Visualization Module

This module generates graphical visualizations to understand the model's learning behavior. It plots the loss curve showing how the model improved during training, and creates a confusion matrix heatmap to visualize prediction results. It also displays sample test images with predicted and actual labels. These visual outputs make the system more interpretable and user-friendly.

4.8 Prediction Module

The final module uses the trained MLP model to predict new handwritten digit inputs. After preprocessing and scaling a new image, the module applies the neural network to classify the digit. It outputs the predicted label along with optional probability values. This module demonstrates how the trained model can be used in real-world applications for automated digit recognition.

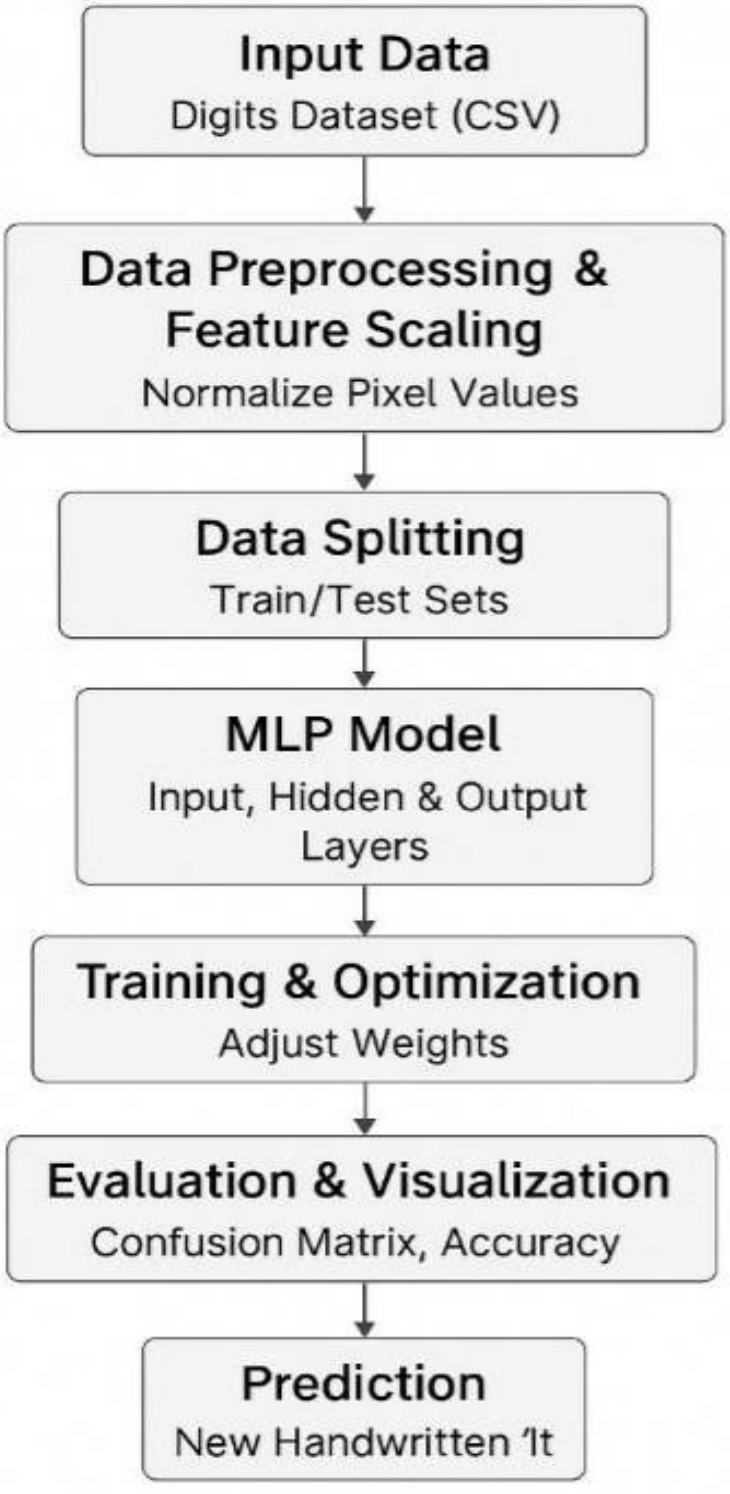
5. IMPLEMENTATION

The implementation of the handwritten digit recognition system is carried out using Python and a Multilayer Perceptron (MLP) neural network model. First, the Digits Dataset is loaded from a CSV file into a Pandas DataFrame. Each record in the dataset represents an 8×8 grayscale image of a handwritten digit, flattened into 64 pixel features along with a label indicating the corresponding digit (0–9). The input features (all pixel columns) are separated from the target label column, and the data types are converted into suitable numeric formats. The dataset is then split into training and testing sets using an 80–20 ratio, ensuring that the model is trained on one portion of the data and evaluated on a different, unseen portion.

Since the pixel values have different ranges, StandardScaler is applied to standardize all input features. Feature scaling is an important step because it helps the neural network converge faster and improves model stability. After preprocessing, an MLP classifier is created using Scikit-learn. The model consists of an input layer with 64 inputs, two hidden layers (for example, with 128 and 64 neurons) using the ReLU activation function, and an output layer with 10 neurons corresponding to the digit classes 0–9. The model is configured with parameters such as maximum iterations, learning rate, and early stopping criteria. It is then trained on the scaled training data, during which the network learns the relationship between pixel patterns and digit labels.

Once training is complete, the model is evaluated on the test set. The predicted labels are compared with the true labels to compute accuracy and generate a detailed classification report. A confusion matrix is also created to analyze how well the model distinguishes between different digits and to identify any common misclassifications. In addition to numerical evaluation, visualizations such as the training loss curve, confusion matrix heatmap, and sample prediction images are generated using Matplotlib. These graphs help in understanding the learning behavior and overall performance of the model. Finally, the trained MLP model and the fitted scaler are saved using Joblib so that new handwritten digit samples can be classified in the future without retraining the network.

6. SYSTEM ARCHITECTURE



7. CODE IMPLEMENTATION

Algorithm: Handwritten Digit Classification using MLP Neural Network

Input: Digits Dataset (8×8 pixel images in CSV form)

Output: Predicted digit (0–9) and performance metrics

Start

1. Load Dataset

1.1 Load the digits dataset from digits_dataset.csv into a DataFrame.

1.2 Separate the dataset into:

- Feature matrix X (all pixel columns).
- Target vector y (the label column: digits 0–9).

2. Preprocess Data

2.1 Convert feature values to numeric type (e.g., float).

2.2 Check for missing or invalid values and handle if required.

3. Split Data

3.1 Use train_test_split to divide the data into:

- Training set: 80%
- Testing set: 20%

3.2 Use a fixed random_state for reproducibility.

4. Feature Scaling

4.1 Initialize StandardScaler.

4.2 Fit the scaler on the training features X_train.

4.3 Transform X_train and X_test using the fitted scaler.

5. Build MLP Model

5.1 Initialize an MLPClassifier with:

- Hidden layers (e.g., (128, 64))
- Activation function: ReLU
- Maximum iterations (e.g., 200)
- Early stopping enabled

5.2 Set random_state for consistent results.

6. Train Model

6.1 Train the MLP model using the scaled training data (X_train_scaled, y_train).

6.2 Store training information such as loss curve (if available).

7. Test Model

7.1 Use the trained model to predict labels for X_test_scaled.

7.2 Store predictions as y_pred.

8. Evaluate Performance

8.1 Compute test accuracy using accuracy_score(y_test, y_pred).

8.2 Generate a classification report (precision, recall, F1-score for each digit).

8.3 Compute a confusion matrix to analyze correct and incorrect predictions.

9. Visualize Results

9.1 Plot the training loss curve across iterations (if loss_curve_ is available).

9.2 Plot the confusion matrix as a heatmap.

9.3 Display sample test images with their true and predicted labels.

10. Save Model and Scaler

10.1 Save the trained MLP model using Joblib (e.g., model.joblib).

10.2 Save the fitted scaler (e.g., scaler.joblib) for future input preprocessing.

End

8. RESULT

```
Dataset loaded: X shape = (1797, 64), y shape = (1797,)
Unique labels: [0 1 2 3 4 5 6 7 8 9]

Train/test split:
 - X_train: (1437, 64)
 - X_test : (360, 64)

Iteration 1, loss = 2.26216924
Validation score: 0.333333
Iteration 2, loss = 1.76612046
Validation score: 0.687500
Iteration 3, loss = 1.36673331
Validation score: 0.798611
Iteration 4, loss = 1.01791859
Validation score: 0.819444
Iteration 5, loss = 0.73820367
Validation score: 0.868056
Iteration 6, loss = 0.53629630
Validation score: 0.868056
Iteration 7, loss = 0.39811867
Validation score: 0.875000
Iteration 8, loss = 0.30400371
Validation score: 0.895833
Iteration 9, loss = 0.23899885
Validation score: 0.895833
Iteration 10, loss = 0.19362034
Validation score: 0.909722
```

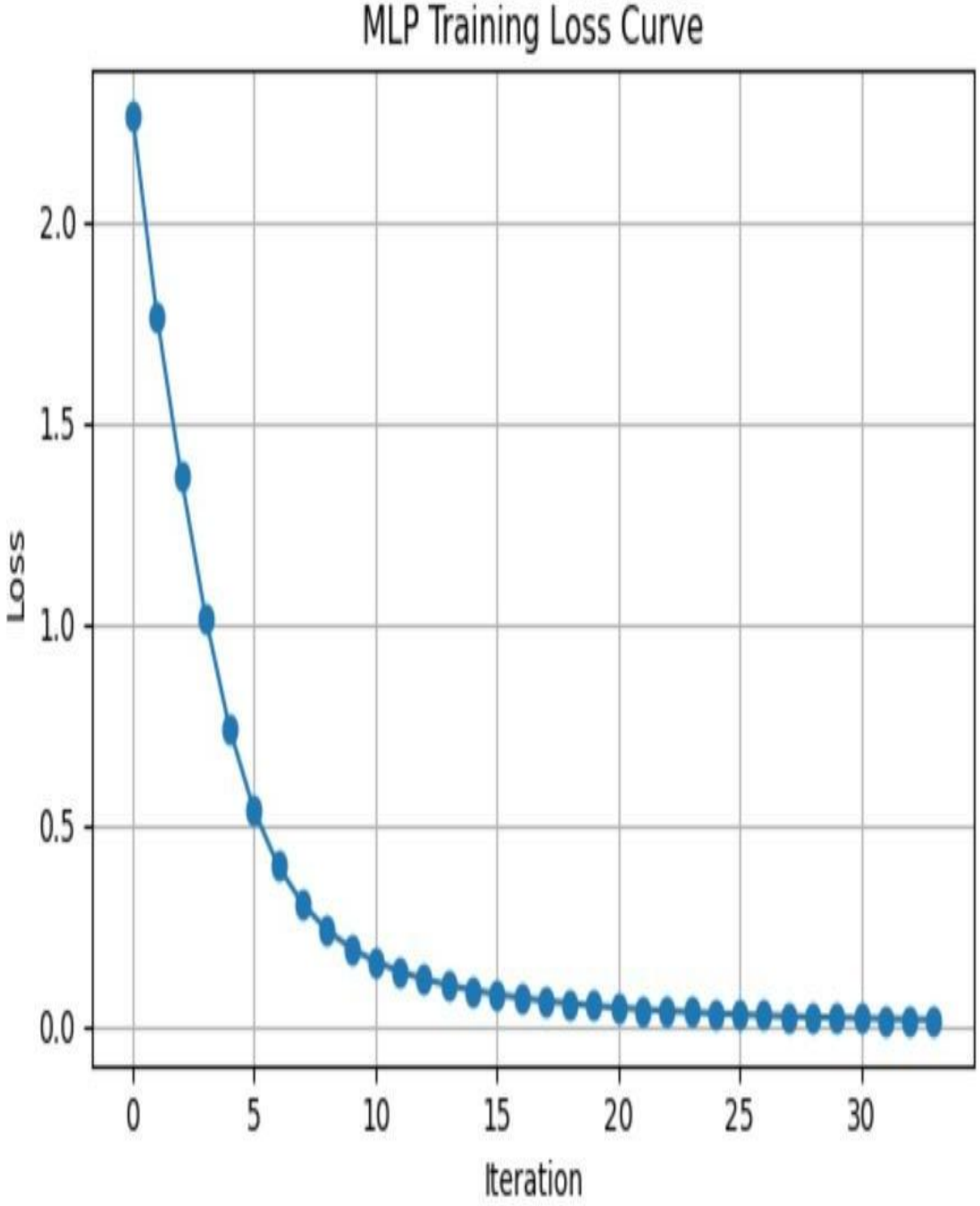
```
Iteration 11, loss = 0.16048227
Validation score: 0.916667
Iteration 12, loss = 0.13614299
Validation score: 0.923611
Iteration 13, loss = 0.11697519
Validation score: 0.930556
Iteration 14, loss = 0.10222775
Validation score: 0.937500
Iteration 15, loss = 0.08947527
Validation score: 0.930556
Iteration 16, loss = 0.07901544
Validation score: 0.930556
Iteration 17, loss = 0.07073748
Validation score: 0.937500
Iteration 18, loss = 0.06278272
Validation score: 0.958333
Iteration 19, loss = 0.05665559
Validation score: 0.958333
Iteration 20, loss = 0.05080870
Validation score: 0.951389
Iteration 21, loss = 0.04585733
Validation score: 0.937500
Iteration 22, loss = 0.04196748
Validation score: 0.958333
Iteration 23, loss = 0.03795126
Validation score: 0.951389
Iteration 24, loss = 0.03472122
Validation score: 0.951389
```

```
Iteration 25, loss = 0.03188970
Validation score: 0.951389
Iteration 26, loss = 0.02897307
Validation score: 0.951389
Iteration 27, loss = 0.02670225
Validation score: 0.958333
Iteration 28, loss = 0.02472246
Validation score: 0.958333
Iteration 29, loss = 0.02290394
Validation score: 0.951389
Iteration 30, loss = 0.02114496
Validation score: 0.951389
Iteration 31, loss = 0.01960103
Validation score: 0.958333
Iteration 32, loss = 0.01825616
Validation score: 0.958333
Iteration 33, loss = 0.01702211
Validation score: 0.958333
Iteration 34, loss = 0.01604532
Validation score: 0.958333
Validation score did not improve more than tol=0.000100 for 15 consecutive epochs. Stopping.
```

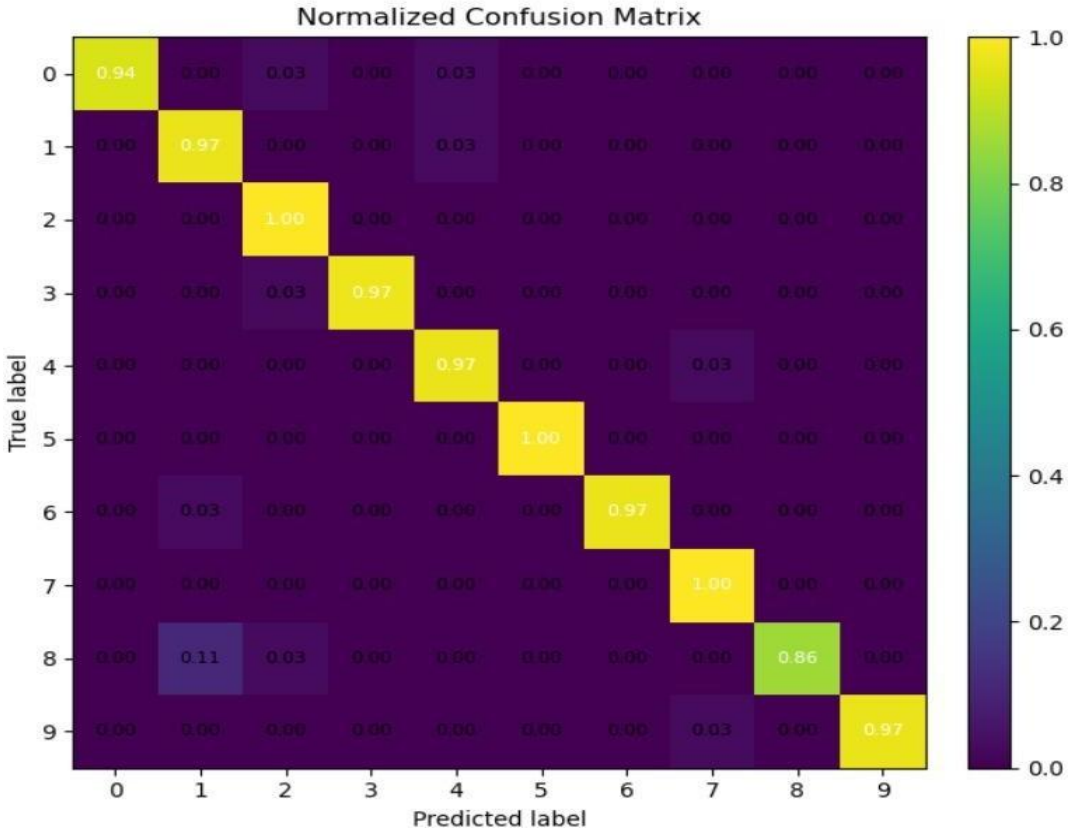
```
Training completed in 0.97 seconds.
Training iterations (loss_curve_) length: 34

Test Accuracy: 0.9667

Classification Report:
```

	precision	recall	f1-score	support
0	1.0000	0.9444	0.9714	36
1	0.8750	0.9722	0.9211	36
2	0.9211	1.0000	0.9589	35
3	1.0000	0.9730	0.9863	37
4	0.9459	0.9722	0.9589	36
5	1.0000	1.0000	1.0000	37
6	1.0000	0.9722	0.9859	36
7	0.9474	1.0000	0.9730	36
8	1.0000	0.8571	0.9231	35
9	1.0000	0.9722	0.9859	36
accuracy			0.9667	360
macro avg	0.9689	0.9663	0.9664	360
weighted avg	0.9692	0.9667	0.9667	360

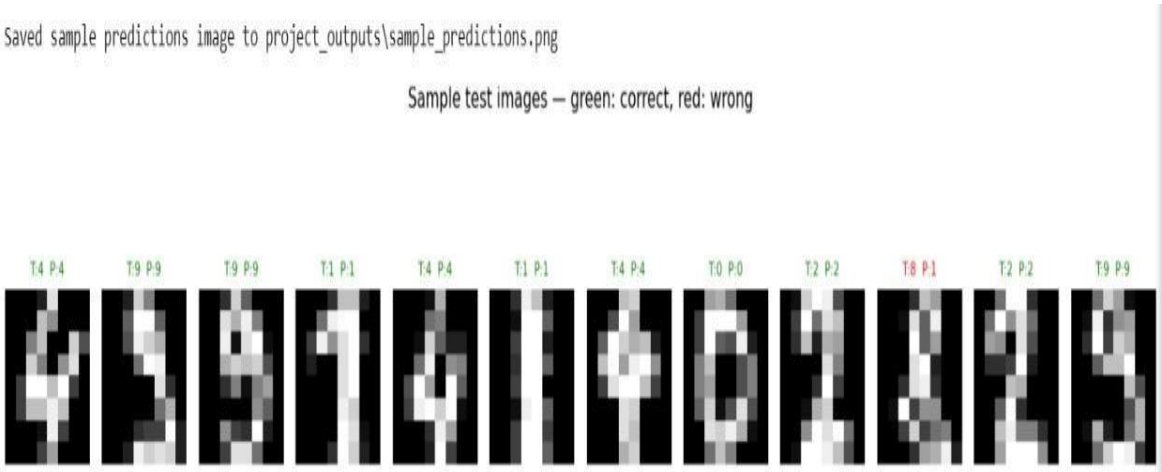
```
Saved loss curve to project_outputs\loss_curve.png
```



```
Saved confusion matrix to project_outputs\confusion_matrix.png
```




```
Saved sample predictions image to project_outputs\sample_predictions.png
```




```
Saved model to project_outputs\model.joblib

All outputs saved in folder: project_outputs
Files saved: ['classification_report.txt', 'confusion_matrix.png', 'confusion_matrix_raw.csv', 'loss_curve.png', 'model.joblib', 'sample_predictions.p
ng', 'scaler.joblib', 'summary.txt']
```

9. CONCLUSION

The handwritten digit recognition system developed in this project successfully demonstrates the use of a Multilayer Perceptron (MLP) neural network for image-based classification. By using the Digits Dataset and applying essential preprocessing techniques such as normalization and feature scaling, the model was able to learn meaningful patterns from the pixel values of 8×8 grayscale digit images. The MLP classifier achieved high accuracy on the test data, confirming the effectiveness of neural networks for digit classification tasks. Visualization techniques, including the loss curve, confusion matrix, and sample prediction images, provided deeper insight into the model's learning behavior and classification performance.

Overall, the project illustrates a complete end-to-end machine learning workflow—from data preparation and model construction to evaluation and prediction. The successful results highlight that even a relatively simple neural network architecture can perform well on digit recognition problems. This work can be extended in the future using more advanced models such as Convolutional Neural Networks (CNNs) or by applying the approach to larger and more complex handwritten datasets.

10. REFERENCES

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research.

Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., et al. (2018). The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. IEEE.

Deng, L. (2012). A Tutorial Survey of Architectures, Algorithms, and Applications for Deep Learning. APSIPA Transactions on Signal & Information Processing.

Sharma, R., & Kumar, V. (2024). A Comparative Study of Machine Learning Models for Handwritten Digit Recognition. International Journal of Computer Applications.

Hassan, M., Ali, F., & Rahman, S. (2023). Handwritten Digit Recognition Using Deep Learning Techniques. Journal of Intelligent Computing.

Roy, P., & Sinha, A. (2023). Performance Evaluation of Neural Network Algorithms for Digit Classification. International Journal of Advanced Research in Computer Science.

Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

Jetley, S., & Prakash, A. (2025). Handwritten Digit Classification Using Lightweight Neural Networks. International Journal of Innovative Research in Engineering.