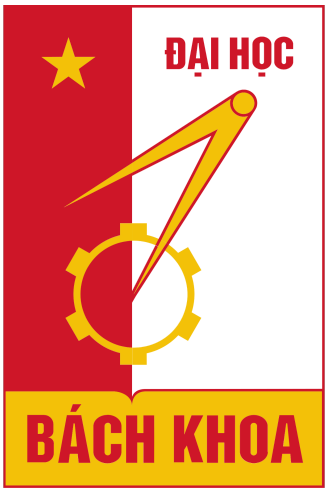
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

INTRODUCTION TO ARTIFICIAL INTELLIGENCE



MINI-PROJECT REPORT

Project Topic:

HANDWRITTEN DIGIT REGCONITION

LIST OF MEMBERS

Full Name	Class	Student ID
Trinh Thanh Cong	DSAI-03-K69	202416668
Doan Hai Dang	DSAI-03-K69	202416669
Trinh Binh An	DSAI-03-K69	202416652
Le Gia Bao	DSAI-03-K69	202416667

Hanoi, January 13, 2026

Contents

1. INTRODUCTION

1.1. Problem Definition

1.1.1 What is Digit Recognition?

- Digit recognition is the task of automatically identifying numerical digits (0–9) from images or handwritten input.
- It is a fundamental problem in pattern recognition and machine learning.
- Manual processing of handwritten digits is slow, costly, and prone to errors.
- Digit recognition enables computers to convert visual digit data into machine-readable numerical values.

1.1.2 Applications

Optical Character Recognition (OCR):

- Converts scanned documents and handwritten text into digital format.
- Enables automatic extraction of numerical data such as dates, IDs, and codes.

a) Banking and Financial Systems:

- Automatic cheque processing and verification.
- Recognition of account numbers, transaction amounts, and forms.
- Reduces manual labor and processing errors.

b) Postal and Logistics Systems:

- Automatic reading of handwritten postal and ZIP codes
- Improves mail sorting speed and delivery accuracy.

c) Document Automation:

- Enables large-scale processing of invoices, bills, and reports.
- Supports automation in government and enterprise systems.

1.1.3 Background and History of the Problem:

- **Early Rule-Based Methods (1960s–1980s):**

  Handcrafted rules based on digit shape; effective only for clean and simple digits.

- **Statistical Feature-Based Methods (1990s):**

  Manual feature extraction combined with statistical classifiers such as k-NN and Bayesian models.

- **Machine Learning Era (Late 1990s–2000s):**

  Learning-based models like SVM and neural networks improved robustness; MNIST became a benchmark dataset.

- **Deep Learning Revolution (2010s–Present):**

  CNNs automatically learn features and achieve state-of-the-art accuracy in digit recognition systems.

2. ALGORITHMS

2.1. K-Nearest Neighbours

2.1.1 Introduction:

`K-Nearest Neighbor (KNN)` is a non-parametric supervised learning method, while it can be used for regressions, mainly used for classification. Unlike parametric models (such as Linear Regression), KNN makes no underlying assumptions about the probability distribution of the data. This allows it to model complex, non-linear decision boundaries effectively.

An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors, in which k is a hyperparameter determined by users. KNN is essentially a "lazy learning" algorithm (or instance-based learning). It does not learn a fixed discriminative function during a training phase; instead, it memorizes the entire training dataset and performs the computation only when a prediction is requested.

2.1.2 Key Concepts:

1. Distance

`KNN` relies on a distance metric to determine the closeness of data points. Common distance metrics include **Euclidean distance**, **Manhattan distance**, and **Cosine similarity**. The choice of distance metric can significantly affect the model's performance, especially in high-dimensional spaces.

However, the most common distance-computing method is **Euclidean Distance**. The straight-line distance between two points x, y in a n-

dimensional space:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

2. The Hyperparameter K (Number of Neighbors):

The choice of K is the most critical factor in the design of a KNN model. It controls the balance between the model's flexibility and its stability (often referred to as the bias-variance tradeoff):

- **Small K (Low Bias, High Variance):** When K is small (e.g., $K = 1$), the algorithm becomes extremely sensitive to local data points. While this allows it to capture fine-grained patterns, it makes the model susceptible to noise and outliers, often leading to *overfitting*. The decision boundaries tend to be jagged and complex.

- **Large K (High Bias, Low Variance):** As K increases, the prediction becomes more stable because it averages over more neighbors. This smooths out the decision boundaries and reduces the impact of outliers. However, if K is too large, the model may become overly simplistic and fail to capture important regularities in the data, leading to *underfitting*.

- **Selection Strategy:** There is no universal optimal value for K. It is typically determined empirically through **Cross-Validation** (e.g., 5-fold or 10-fold), where different values of K are tested to minimize the validation error rate.

3. Weighted Voting:

Standard KNN employs a uniform voting scheme where every neighbor in the set of k neighbors contributes equally to the decision, regardless of their distance from the query point. This can be problematic,

particularly when the value of k is large, as distant neighbors (which may be less relevant) can outvote closer, more significant neighbors.

To address this, **Distance-Weighted KNN** assigns a weight w_i to each neighbor based on its distance d from the query point. A common weighting function is the inverse distance:

$$w_i = \frac{1}{d(x_{query}, x_i)^2}$$

- **Mechanism:** Instead of a simple count, the algorithm sums the weights of the neighbors belonging to each class. The class with the highest total weight is predicted.
- **Advantage:** This ensures that closer neighbors have a much stronger influence on the classification, reducing the risk of misclassification caused by distant points or outliers included within the k boundary.

2.1.3 Reasons for Selecting KNN:

We selected K-Nearest Neighbors (KNN) as the initial classifier for this project due to the following key advantages:

- **Effective for Pattern Recognition:** Handwritten digits rely on spatial similarity. KNN effectively classifies images by measuring pixel-wise proximity in feature space, making it naturally suited for character recognition without complex feature extraction.
- **Non-Parametric Nature:** As a non-parametric model, KNN makes no assumptions about the data distribution. This flexibility allows it to adapt to the irregular and non-linear decision boundaries often found in handwritten data.
- **Zero-Cost Training:** KNN is a lazy learner that requires no explicit training phase. This allows for rapid prototyping and immediate experimentation with the dataset.

- **Strong Baseline:** Due to its simplicity and deterministic nature, KNN serves as an excellent benchmark. It establishes a minimum performance threshold that more complex models (like SVMs or CNNs) must surpass to be considered effective.

2.2. Support Vector Machine (SVM)

2.2.1 Introduction:

Support Vector Machine (SVM) is a supervised machine learning algorithm used mainly for classification tasks. The main objective of SVM is to find a decision boundary, called a hyperplane, that separates data points of different classes as clearly as possible.

The key idea behind SVM is margin maximization. Instead of simply finding any boundary that separates the classes, SVM chooses the hyperplane that maximizes the distance between the boundary and the nearest data points from each class. A larger margin generally leads to better performance on unseen data

The data points that lie closest to the hyperplane are known as support vectors. These points are critical because they define the position and orientation of the hyperplane. Removing other points usually does not affect the final model, which makes SVM efficient and robust.

2.2.2 Key Concepts:

1. Hyperplane:

The decision boundary that separates different classes. In the linear case, the hyperplane is represented as:

$$\mathbf{w}^T\mathbf{x} + b = 0$$

where $\mathbf{w}$ is the normal vector to the hyperplane, b is the bias term representing the distance of the hyperplane from the origin along the

normal vector $\mathbf{w}$.

2. Support Vectors:

The closest data points to the hyperplane that determine the decision boundary. Only these points satisfy the constraint:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \leq 1$$

3. Margin:

The distance between the hyperplane and the support vectors, which SVM aims to maximize.

$$\text{Margin} = \frac{2}{\|\mathbf{w}\|}$$

4. Kernel:

A function that enables SVM to handle non-linear classification by transforming data into a higher-dimensional space:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x_i})^T\phi(\mathbf{x}_j)$$

5. Soft Margin:

A mechanism that allows some misclassification to improve generalization. This is achieved by introducing slack variables

$$\xi_i$$

and solving:

$$\min \frac{1}{2}\|w\|^2 + c\sum_{i=1}^{N}\xi_i$$

subject to:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

where is a regularization parameter that controls the trade-off between margin maximization and penalty for misclassifications and

$$\xi_i$$

are slack variables that represent the degree of violation of the margin by each data point.

2.2.3 Reasons for choosing SVM for the project

- **High-dimensional input space**: Each MNIST image is represented as a flattened vector of dimension: d=28×28=784.
- **SVM's suitability for high-dimensional data** :it constructs a decision boundary by maximizing the margin rather than relying on distance-based heuristics. This property allows SVM to maintain good generalization performance even when the feature dimension is large.
- **Limited number of critical samples**: In SVM, only samples that satisfy:

  $$y_i(\mathbf{w}^T\mathbf{x}_i + b) \leq 1$$

  become support vectors. These samples lie on or inside the margin and directly influence the decision boundary. As a result, the final model depends only on a subset of training data, reducing model complexity and improving generalization.
- **Nonlinear class boundaries**: Handwritten digits often exhibit nonlinear separability. Digits with similar shapes, such as (4, 9) or (3, 5), cannot be separated by a linear hyperplane in the original feature space. This motivates the use of kernel-based SVM, which enables nonlinear decision boundaries through implicit feature space transformation.

- **Multi-class Classification Using One-vs-One (OvO) and One-vs-Many(OvM)**:

  The MNIST dataset consists of K=10 digit classes. Since SVM is inherently a binary classifier, a multi-class strategy is required.

  This project compare the One-vs-One (OvO) strategy with 45 binary classifiers(0 or 1; 0 or 2; ...) and One-vs-Many(OvM) strategy with 9 classifiers (0 or not 0; 1 or not 1;...)

  Each classifier is trained to distinguish between two classes i and j by learning the decision function:

$$f_{ij}(x) = \mathbf{w}_{ij}^T \Phi(x) + b_{ij}$$

- **How prediction works**: When a new image is given for prediction, it is passed through all binary classifiers. Each classifier produces a simple decision:

$$\hat{y}_{ij} = \begin{cases} i, & f_{ij}(x) > 0 \\ j, & otherwise \end{cases}$$

  This means each classifier votes for one of the two result it was trained on.

  After all classifiers have voted, the final predicted digit is determined by majority voting:

$$\hat{y} = \arg\max_k \sum_{i \neq j} I(\hat{y}_{ij} = k)$$

  where `I()` is an indicator function that counts how many votes each digit receives. The digit with the highest number of votes is selected as the final prediction. This voting process improves robustness because the final decision is based on multiple independent classifiers instead of a single model.

2.3. Multi-Layer Perceptron

2.3.1 Introduction

Architecture: A Multi-Layer Perceptron (MLP) is a class of feed-forward artificial neural network (ANN). Structurally, it consists of at least three layers of nodes: an *input layer*, one or more *hidden layers*, and an *output layer*. A defining characteristic of MLPs is their **fully connected** nature, where every neuron in one layer is connected to every neuron in the subsequent layer.

Learning Mechanism: MLPs distinguish themselves from simpler linear models by their ability to learn complex, non-linear relationships. This is achieved through the use of non-linear **activation functions** (such as ReLU or Sigmoid) and the **backpropagation** algorithm. Backpropagation allows the network to adjust its internal weights iteratively to minimize prediction error.

Application to Digit Recognition: In the context of this project, the MLP learns to map raw pixel intensities to digit labels (0-9). By employing gradient-based optimization, the network adjusts its weights to identify spatial patterns within the flattened pixel vectors, effectively functioning as a robust non-linear classifier.

2.3.2 Key concepts

1. The Artificial Neuron Model

The fundamental unit of the MLP is the neuron (or perceptron). Each neuron performs a two-step computation:

- **Linear Aggregation:** First, it computes the weighted sum of its inputs (x_i) plus a bias term (b).
- **Non-Linear Activation:** It then passes this sum through a differentiable non-linear function $f(\cdot)$.

Mathematically, for a single neuron:

$$z = \sum_{i=1}^{n} w_i x_i + b$$

$$a = f(z)$$

where w_i are the learnable weights, b is the bias, and a is the final output activation.

2. Activation Functions

Activation functions introduce non-linearity into the network, enabling it to learn complex patterns that a simple linear model cannot.

Rectified Linear Unit (ReLU): Used primarily in hidden layers, ReLU is computationally efficient and helps mitigate the "vanishing gradient" problem during training.

$$f(x) = \max(0, x)$$

Sigmoid Function: Historically used to map outputs to a range between 0 and 1. While less common in deep hidden layers today, it is essentially a smooth step function.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Softmax Function (Output Layer): For multi-class classification tasks (like digit recognition), the final layer uses Softmax to convert raw logits into a probability distribution. It ensures that all output values sum to 1.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

3. Learning Process

- **Loss Function (Categorical Cross-Entropy):** To measure the performance of the model, we calculate the difference between the predicted probability distribution ($\hat{y}$) and the actual true label (y). For classification, Cross-Entropy is the standard objective function:

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

- **Backpropagation:** This is the core training algorithm. It computes the gradient of the loss function L with respect to every weight w in the network using the chain rule:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$$

  These gradients are then used to update the weights via an optimizer (such as Stochastic Gradient Descent or Adam) to minimize the error.

Reasons for choosing MLP

The Multi-Layer Perceptron was selected as an intermediate classifier for this project due to the following strategic advantages:

- **Modeling Non-Linear Relationships:** Handwritten digits often contain irregularities, loops, and varying writing styles that cannot be separated linearly. Unlike simple linear classifiers (e.g., Perceptrons), MLPs utilize non-linear activation functions (such as ReLU) to learn complex, non-linear decision boundaries. This allows the model to capture the intricate topological differences between digits like '3' and '8'.

- **Global Pattern Recognition:** The *fully connected* nature of the MLP architecture ensures that every neuron in a hidden layer

receives input from every pixel in the input image. This allows the network to aggregate global information and learn holistic patterns across the entire digit, rather than focusing solely on local features.

- **Computational Efficiency vs. CNNs:** While Convolutional Neural Networks (CNNs) are the state-of-the-art for image recognition, they introduce significant architectural complexity (pooling layers, convolutional kernels). For relatively small datasets (like MNIST), an MLP offers a simpler, lightweight alternative that is computationally cheaper to train while still achieving respectable accuracy.

- **Benchmark for Learning Capability:** The MLP serves as a critical "trainable baseline." Unlike KNN (which memorizes data), the MLP actually learns a parametric function. Implementing an MLP allows us to verify that a neural network approach outperforms traditional statistical methods before moving to more advanced deep learning architectures.

2.3.3 Convolutional Neural Network (CNN)

1. Introduction

Convolutional Neural Networks (CNNs) represent a specialized class of deep learning models designed specifically for processing data with a grid-like topology, such as image pixel arrays. They are currently the state-of-the-art architecture for computer vision tasks.

- **Spatial Awareness:** Unlike Multi-Layer Perceptrons (MLPs) which flatten inputs and lose spatial context, CNNs preserve the spatial structure of the image. This allows the model to recognize local patterns (such as edges, curves, and corners) regardless of their position in the image.

- **Hierarchical Feature Learning:** CNNs learn in a hierarchy. Lower layers detect simple low-level features (e.g., vertical lines), while deeper layers combine these to detect high-level semantic shapes (e.g., loops in the digit '8' or strokes in '7').

2. Key Concepts

1. **Convolutional Layer:** This is the core building block. Instead of connecting every neuron to every pixel, we slide a small learnable filter (or kernel) over the input image to produce a **Feature Map**. This operation mathematically corresponds to a discrete convolution:

$$(F * X)(i, j) = \sum_m \sum_n X(i + m, j + n) \cdot F(m, n)$$

where X is the input image, F is the learnable filter/kernel, and (i, j) are the spatial coordinates.

2. **Activation Function (ReLU):** After every convolution operation, a non-linear activation function is applied element-wise. The Rectified Linear Unit (ReLU) is the standard choice because it accelerates convergence and solves the vanishing gradient problem:

$$f(x) = \max(0, x)$$

3. **Pooling Layer (Subsampling):** Pooling layers are inserted between convolutional layers to reduce the spatial dimensions (width and height) of the representation.

    - **Mechanism:** Common techniques include *Max Pooling* (taking the largest value in a window) or *Average Pooling*.
    - **Benefit:** This reduces the computational cost and introduces **translation invariance**, making the model robust to small

shifts or distortions in the handwritten digits.

4. **Fully Connected (Dense) Layer:** Once the spatial features have been extracted and down-sampled, the high-dimensional feature maps are flattened into a 1D vector. This vector is passed through one or more fully connected layers (identical to an MLP) to perform the final classification reasoning.

5. **Softmax Output Layer:** The final layer maps the network's output logits (z) into a probability distribution over the 10 digit classes ($i \in \{0, \ldots, 9\}$):

$$P(y = i) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

3. Reasons for choosing CNN

Convolutional Neural Networks were selected as the primary advanced classifier for this project due to their inherent design advantages for image analysis:

- **Preservation of Spatial Locality:** Handwritten digits are defined by the spatial arrangement of their pixels. Unlike Multilayer Perceptrons (MLPs) that flatten the input into a 1D vector (destroying spatial structure), CNNs process the image as a 2D grid. This allows the model to exploit local correlations between neighboring pixels, which is essential for recognizing continuous strokes and curves.

- **Hierarchical Feature Extraction:** CNNs automatically learn a hierarchy of features without manual engineering. Early layers detect low-level primitives (such as edges and gradients), while deeper layers combine these into high-level semantic patterns (such as loops in the number '8' or crossing lines in '4'). This mimics the human visual cortex's approach to object recognition.

- **Translation Invariance:** Handwriting is rarely perfectly centered or aligned. Through the use of **Pooling Layers** (subsampling), CNNs become robust to small translations and distortions. This means the network can correctly classify a digit even if it is shifted slightly to the left or right, a common occurrence in the MNIST dataset.

- **State-of-the-Art Performance:** Empirically, CNNs have established themselves as the benchmark architecture for computer vision. On standard datasets like MNIST, they consistently outperform "shallow" models (like SVM or KNN) and dense fully connected networks by a significant margin, often achieving accuracy rates exceeding 99%.

3. EXPERIMENTAL SETUP

The experiments were conducted using the MNIST dataset, which consists of 70,000 grayscale images of handwritten digits (0–9), divided into two parts: a 60,000 images training set and a 10,000 images testing set for evaluation. Each image has a resolution of 28×28 pixels.

3.1. Data Pre-processing

Before feeding data into the models, the following transformations were applied to ensure numerical stability and compatibility:

- **Normalization:** Pixel intensity values (originally 0–255) were scaled to the range $[0, 1]$ by dividing by 255. This step is critical for faster convergence in gradient-based models (MLP, CNN) and prevents distance-based models (KNN, SVM) from being biased by large values.

- **Flattening (Vectorization):** For the KNN, SVM, and MLP classifiers, each 28×28 image was flattened into a 1D vector of size 784.

- **Spatial Preservation:** For the CNN, the original 2D structure ($28 \times 28 \times 1$) was preserved to allow the convolutional layers to extract spatial features.

3.2. Training Strategy

- **K-Nearest Neighbors (KNN):** We optimized the hyperparameter K by evaluating a range of values. The model was trained using the brute-force search method with Euclidean distance.

- **Support Vector Machine (SVM):** Given the multi-class nature of the problem (10 digits), we implemented the **One-vs-One**

(OvO) strategy. Hyperparameters (such as the regularization parameter C and kernel coefficient γ) were tuned using 5-fold stratified cross-validation.

- **Multi-Layer Perceptron (MLP):** The network was trained using the `Backpropagation` algorithm with the `Categorical Cross-Entropy` loss function. We utilized an adaptive optimizer(Adam) with auto-adjusted learning rate to minimize the loss over multiple epochs.
- **Convolutional Neural Network (CNN):** The architecture was inspired by the classic **LeNet-5** design, consisting of alternating convolutional and pooling layers followed by fully connected layers.

3.3. Evaluation Metrics

To assess the performance of the classifiers, we utilized two primary metrics on the unseen test set (10,000 samples):

- **Classification Accuracy:** The ratio of correctly predicted digits to the total number of samples:

$$\text{Accuracy} = \frac{\textit{Number of Correct Predictions}}{\textit{Total Number of Samples}}$$

4. Results and Comparison

4.1. K-Nearest Neighbors

The performance of the KNN classifier was evaluated using different values of K (number of neighbors). The results are summarized in Table 4.1.

K (number of neighbors)	Accuracy
1	96.91%
3	97.05%
5	96.88%
7	96.94%
9	96.59%
11	96.68%
13	96.53%

Table 4.1: KNN Accuracy for different values of K

Performance Summary for Optimal K (K=3):

- **Training Time:** None (Lazy Learning)
- **Inference Time:** 6.4s
- **Accuracy:** 97.05%

4.2. SVM (Cross-validation)

We evaluated Support Vector Machines using both One-vs-One (OvO) and One-vs-Many (OvM) strategies with 5-fold cross-validation.

One-vs-One:

Training Time: 162.0s (Average: 32.4s per fold)
Inference Time: 60.6s
Accuracy: 97.92%

One-vs-Many:

Training Time: 160.8s (Average: 32.16s per fold)

Inference Time: 63.9s

Accuracy: 97.92%

4.3. Multi-Layer Perceptron (MLP)

The MLP model offered a significant improvement in inference speed compared to KNN and SVM.

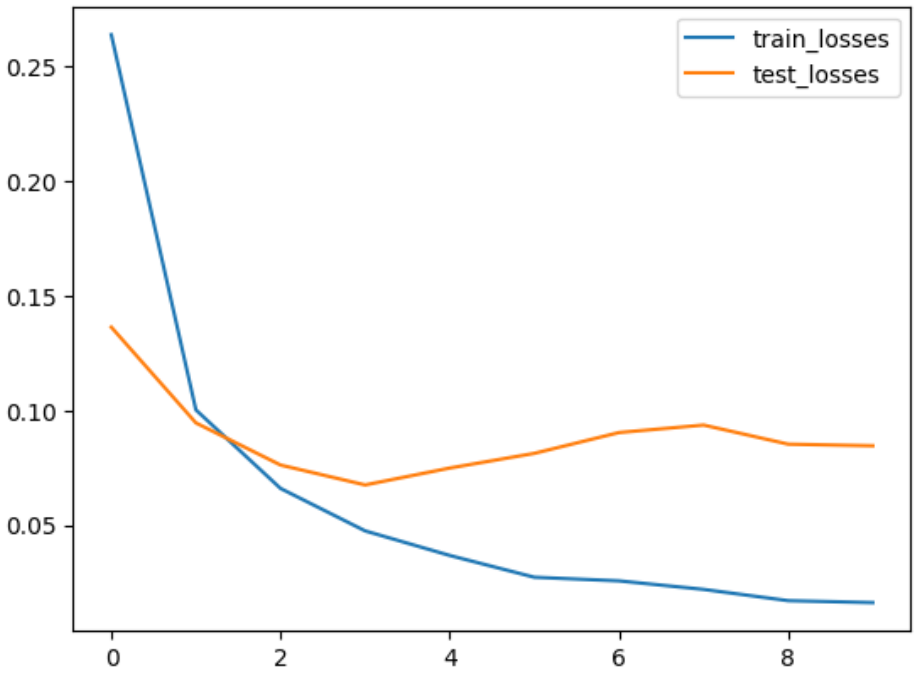


Figure 4.1: MLP Loss optimization process through 10 epochs

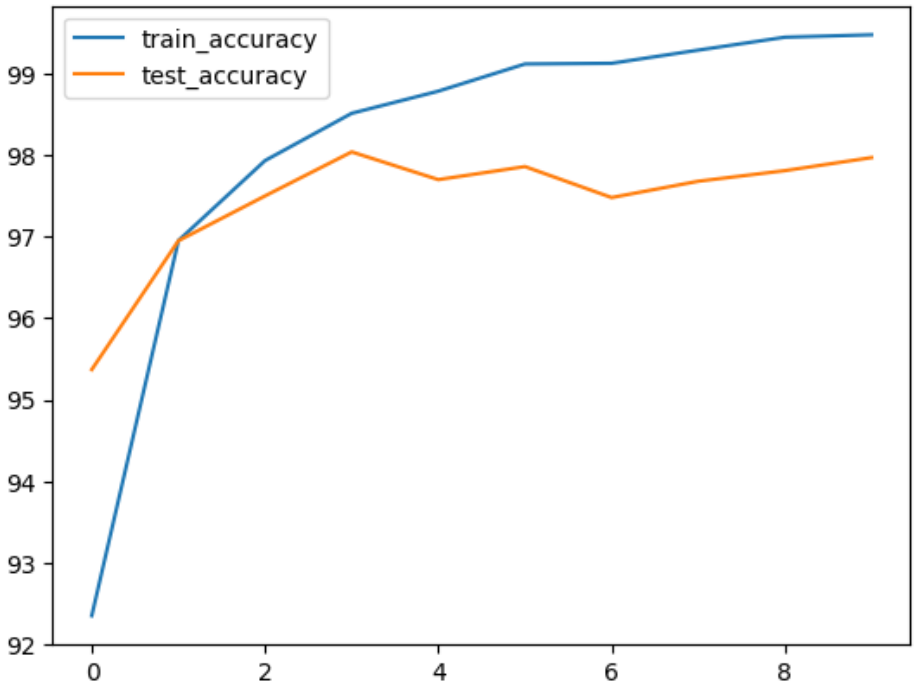


Figure 4.2: MLP Accuracy through 10 epochs

- **Training Time:** 28.7s
- **Inference Time:** 0.4s
- **Accuracy:** 97.24%

4.4. Convolutional Neural Network (CNN)

The CNN model achieved the highest accuracy among all classifiers, demonstrating the effectiveness of spatial feature extraction.

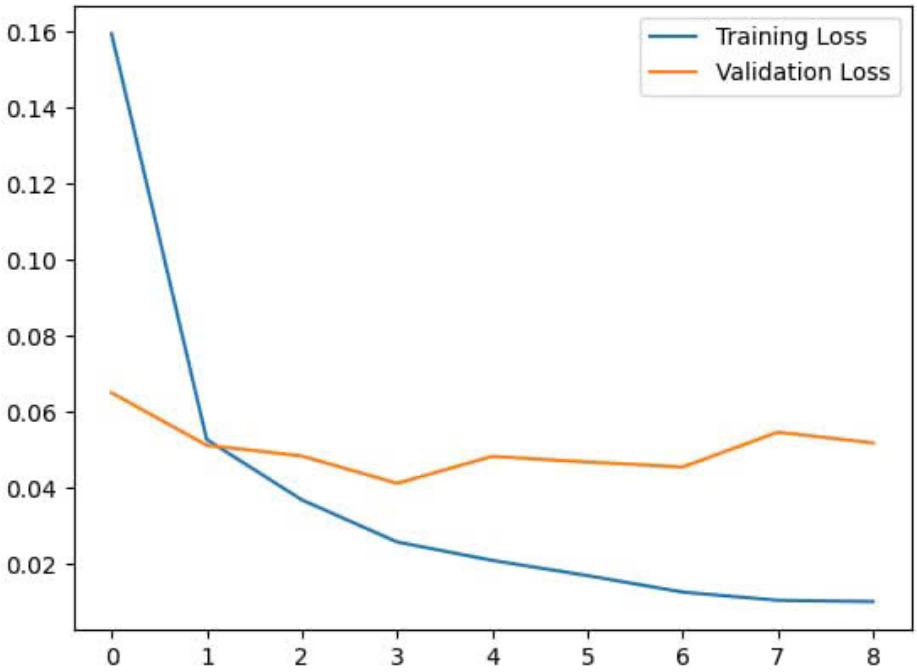


Figure 4.3: CNN Loss optimization process through 10 epochs

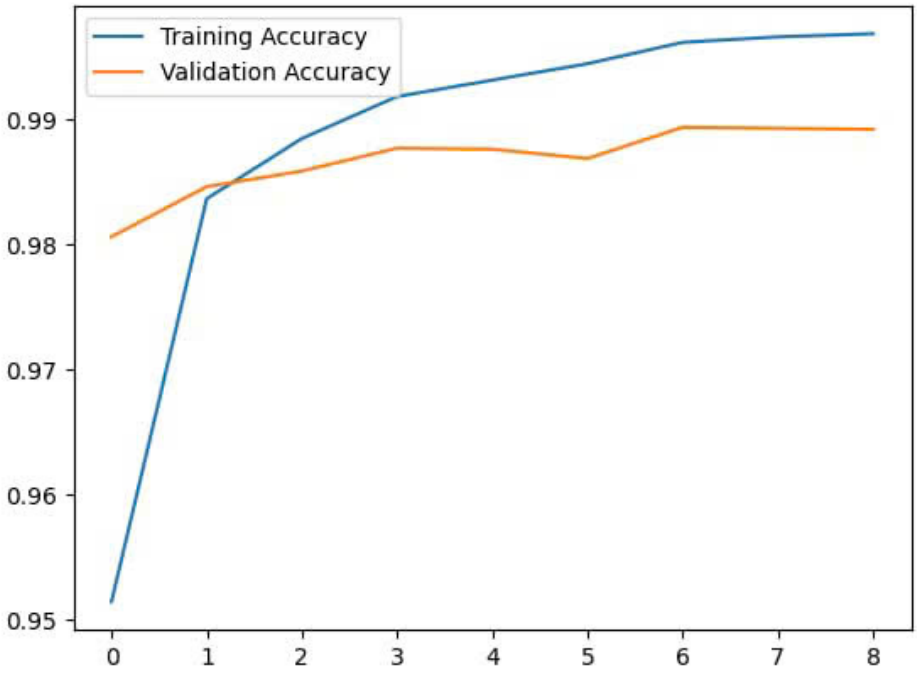


Figure 4.4: CNN Accuracy through 10 epochs

- **Training Time:** 108.4s
- **Inference Time:** 1.0s

- **Accuracy:** 98.92%

4.5. Conclusion

The experimental results demonstrate that deep learning models outperform traditional machine learning methods in digit recognition tasks.

- **K-Nearest Neighbors (KNN):** While simple and intuitive, KNN struggles with high-dimensional data and becomes computationally expensive during inference due to its lazy learning nature.
- **Support Vector Machine (SVM):** SVM showed moderate performance due to its margin maximization principle and robustness to overfitting. However, the computational cost at runtime is significantly higher compared to other models (10 to 60 times). This can be explained by the prediction mechanism, which requires computing the similarity between the input and all support vectors. This leads to a time complexity of $O(N_{sv} \cdot d)$, where N_{sv} is the number of support vectors and d is the number of feature dimensions.
- **Multi-Layer Perceptron (MLP):** With the lowest inference time, moderate training time, and accuracy only slightly behind CNN, the MLP proved its capability to model complex non-linear relationships. However, it fails to fully utilize the spatial structure of images due to the flattening of input vectors.
- **Convolutional Neural Network (CNN):** CNNs significantly outperformed all other models in accuracy because they preserve spatial information and automatically extract hierarchical features. This makes CNNs the most suitable choice for image-based tasks like handwritten digit recognition.

5. CONCLUSION

This project explored multiple machine learning and deep learning techniques for handwritten digit recognition, including KNN, SVM, MLP, and CNN.

Traditional approaches such as KNN and SVM provide strong baselines, with SVM achieving high accuracy through margin optimization and kernel methods. MLPs demonstrated the effectiveness of neural networks in learning complex patterns.

However, CNNs achieved the best performance due to their ability to learn spatially meaningful features directly from image data. Their hierarchical feature extraction makes them the most suitable model for digit recognition tasks.

Overall, this highlights the importance of choosing architectures that align with the structure of the data. For image-based problems, convolutional neural networks remain the optimal solution.