Deep Learning Project

Université Gustave Eiffel

Digit Recognizer

Prepared By:

Laatar Mohamed

Loukil Mohamed Aziz

Tlili Mohamed Amine

Academic year:

2024-2025

Table of Contents

I. Introduction & Problem Understanding 4
1. Problem Definition and Objectives 4
2. Data-Specific Challenges 4
II. Model Architecture Design 4
1. Convolutional Neural Network (CNN) 4
2. K-Nearest Neighbors (KNN) 5
3. Multi-Layer Perceptron (MLP) 6
III. Training Optimization Strategies 6
1. Training Algorithms and Optimization Techniques 7
2. Impact of Optimization Strategies on Model Performance 8
IV. Model Evaluation & Validation 9
1. Validation Framework and Approach 9
2. Performance Metrics 9
V. Conclusion & Lessons Learned 13
1. Key Insights Gained 13
2. Challenges Encountered 13
3. Lessons for Future Projects 13

Acknowledgments

We, as a group, would like to sincerely thank Mr. Jawad Alaoui for his exceptional teaching and clear explanations during the course on Machine Learning & Deep Learning. His ability to present complex concepts in an accessible and engaging manner greatly enhanced our understanding and empowered us to successfully complete this project. This report reflects the knowledge and skills we gained under his invaluable guidance.

I. Introduction & Problem Understanding

1. Problem Definition and Objectives

The primary objective of this project is to develop a deep learning model capable of accurately identifying handwritten digits from images. The dataset used for this task is the well-known **MNIST** dataset, which consists of tens of thousands of labeled images of digits (0–9). This project aims to explore and benchmark different classification algorithms to determine which approach performs best in digit recognition.

2. Data-Specific Challenges

Image Preprocessing: The quality and consistency of the input images can impact model performance. Variations in handwriting styles, digit sizes, and pixel noise require effective preprocessing techniques to standardize the dataset for better learning outcomes.

Model Overfitting: Given the relatively small size of the MNIST dataset (compared to modern deep learning datasets), there is a risk of overfitting, especially when employing complex models. Proper regularization and data augmentation techniques are essential to mitigate this issue.

Generalization: Ensuring the model generalizes well to unseen handwritten digits is a critical challenge. Factors such as variations in writing styles, blurriness, and overlapping digits must be addressed to build a robust model.

II. Model Architecture Design

1. Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are widely regarded as one of the most effective architectures for image classification tasks. They leverage spatial hierarchies in images to extract and process features effectively.

This CNN architecture is specifically designed for processing grayscale images of digits with a size of 28x28 pixels. The input layer takes in data with a shape of (28, 28, 1), representing the image dimensions and a single grayscale channel.

The convolutional layers are organized into two distinct blocks, each responsible for extracting features at increasing levels of complexity. The first convolutional block consists of two Conv2D layers, each applying 32 filters of size (3, 3) with ReLU activation and 'same' padding to preserve the spatial dimensions of the input. These layers are followed by batch normalization to stabilize the

training process and accelerate convergence. A max pooling layer, with a (2, 2) pooling window and a stride of 2, reduces the spatial dimensions by half, effectively down-sampling the data. To prevent overfitting, a dropout layer with a rate of 0.25 is added.

The second convolutional block follows a similar structure but increases the number of filters in the Conv2D layers to 64, allowing the network to capture more complex patterns. Like the first block, batch normalization is applied after each convolutional layer. Another max pooling layer further reduces the spatial dimensions, and a dropout layer with the same rate of 0.25 is included to continue mitigating overfitting.

After the convolutional blocks, the Flatten layer converts the 2D feature maps into a 1D vector, making the data suitable for fully connected layers. The network then includes two fully connected layers. The first dense layer contains 512 neurons with ReLU activation, followed by batch normalization and a dropout layer with a rate of 0.25. The second dense layer is larger, with 1024 neurons and ReLU activation, emphasizing the learning of more complex patterns. This layer also includes batch normalization and a higher dropout rate of 0.5 for further regularization.

Finally, the output layer consists of 10 neurons, one for each digit class (0-9). It uses softmax activation function to output probabilities for each class, enabling the network to predict the digit in an image.

This architecture is designed to balance complexity and computational efficiency. By using convolutional layers to extract spatial features, pooling layers to reduce dimensions, and fully connected layers for classification, it is well-suited for digit recognition tasks. Regularization techniques like dropout and batch normalization ensure stable training and reduce the risk of overfitting.

2. K-Nearest Neighbors (KNN)

The K-Nearest Neighbors (KNN) model is a simple yet effective algorithm for image classification tasks, particularly suitable for this digit recognition problem.

The model works by identifying the K nearest neighbors to a given test data point in the feature space, and classifying it based on the majority label of those neighbors. In this case, the target variable is the label of the digit, ranging from 0 to 9. The KNN algorithm is non-parametric, meaning it makes no assumptions about the underlying data distribution, which is ideal for a flexible image classification task like this one.

In summary, this KNN model works by comparing new images with labeled training data and classifying them based on the majority vote from the nearest neighbors.

3. Multi-Layer Perceptron (MLP)

The Multi-Layer Perceptron (MLP) model is a versatile and effective architecture for classification tasks, including digit recognition. It is composed of fully connected layers, each layer processing the data with increasing complexity.

The network begins with an input layer consisting of 512 neurons, each using ReLU (Rectified Linear Unit) activation. ReLU introduces non-linearity into the model, allowing it to learn complex patterns in the data. To prevent overfitting, a dropout layer with a rate of 0.2 is applied after the input layer, randomly deactivating a fraction of the input neurons during training.

Following the input layer, the model includes two hidden layers. The first hidden layer contains 256 neurons with ReLU activation, again introducing non-linearity and enabling the learning of more abstract features. A dropout layer with a rate of 0.2 follows this layer to reduce overfitting. The second hidden layer consists of 128 neurons, also using ReLU activation. A dropout layer with the same rate of 0.2 is included to continue regularizing the network.

The final layer of the network is the output layer, consisting of 10 neurons. Each neuron corresponds to one of the 10 possible digit classes (0-9). The output layer uses the softmax activation function, which converts the raw output into probabilities, allowing the model to predict the most likely digit based on the image input.

The model is compiled with the Adam optimizer, which adapts the learning rate during training for better performance. The categorical cross-entropy loss function is used, suitable for multi-class classification problems. Accuracy is used as the evaluation metric to track the model's performance during training.

To prevent overfitting, early stopping is applied during training. The training process is halted if the validation loss does not improve for five consecutive epochs, ensuring that the model does not train for too long and overfit to the training data.

This MLP architecture is designed to be simple yet powerful for digit classification tasks. The use of fully connected layers allows the model to learn complex relationships between pixel values, while dropout layers help regularize the model and prevent overfitting. With a softmax output layer, the model can classify input images into one of the 10 possible digit classes, making it an effective solution for digit recognition.

III. Training Optimization Strategies

Optimizing the training process is crucial for improving model performance and ensuring efficient convergence. In this project, a combination of advanced optimization algorithms, learning rate

schedules, and regularization techniques has been employed to enhance the training of the **CNN** model.

1. Training Algorithms and Optimization Techniques

Adam Optimizer

Overview: The Adam optimizer combines the strengths of two optimization methods, AdaGrad and RMSProp, offering adaptive learning rates with momentum. It is computationally efficient and requires minimal tuning, making it suitable for large datasets and complex models.

Key Parameters:

- **Learning Rate**: Set to 0.001 for controlled and stable convergence.
- **Beta 1**: 0.9, to ensure smooth momentum updates by averaging gradients.
- **Beta 2**: 0.999, to stabilize learning by considering gradient variance.

Learning Rate Decay

Overview: A constant learning rate may hinder convergence to a local minimum. Learning rate decay dynamically adjusts the learning rate as training progresses, allowing finer adjustments as the model approaches optimal performance.

Techniques Used:

- **Learning Rate Scheduler:** Adjusts the learning rate at each epoch using a predefined decay function. For instance, the learning rate was reduced by 10% at each epoch as follows:

```
reduce_lr = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)
```

Early Stopping

Early stopping is a regularization strategy that halts training when the model shows no significant improvement in validation performance over a specified number of epochs.

- This technique prevents overfitting by ensuring that the model stops training at the optimal point.
- Additionally, it reduces unnecessary computation by avoiding further training when performance gains plateau.

2. Impact of Optimization Strategies on Model Performance

The combination of advanced optimization strategies had a significant impact on the CNN model's performance, resulting in improved accuracy and convergence behavior.

Progressive Accuracy Improvement

Through the use of learning rate decay and the Adam optimizer, the model demonstrated steady improvement in accuracy across epochs.

Key observations:

- **Initial Learning Stages**: The model achieved a training accuracy of **78.6%** and validation accuracy of **92.3%** by the end of the first epoch, showcasing effective initial weight updates.
- **Consistent Gains**: With each epoch, validation accuracy improved further, reaching **99.5%** by the 50th epoch, illustrating the optimizer's ability to refine the model.

Role of Learning Rate Decay

The gradual reduction in learning rate allowed the optimizer to take smaller steps as the model approached the optimal solution.

- **Initial Learning Rate**: A learning rate of **0.001** ensured efficient early training.
- **Decay Effect**: By the final epochs, the learning rate decreased to as low as $\mathbf{5.7 \times 10^{-6}}$, enabling fine-tuned adjustments without overshooting.

Overfitting Prevention

Validation loss consistently decreased alongside training loss, reaching a minimal value of **0.0134**. This indicates the effectiveness of:

- **Learning Rate Decay**: Reduced overfitting by ensuring controlled updates.
- **Regularization Strategies**: Dropout layers and early stopping thresholds helped prevent excessive model complexity.

Efficient Convergence

The combination of strategies resulted in quick convergence within **50 epochs**. The model achieved near-perfect training accuracy (100%) while maintaining high validation accuracy (99.55%), reflecting robust generalization to unseen data.

IV. Model Evaluation & Validation

1. Validation Framework and Approach

CNN's performance was evaluated using training and testing datasets to measure its generalization capability. The training set was used for optimizing the model's parameters, while the testing set served as an independent dataset to validate the model's performance. The evaluation metrics were monitored over 50 epochs, focusing on the loss and accuracy curves for both training and testing phases.

KNN model involves using a training and testing split to evaluate the model's generalization ability. The training set is used to train the model, while the testing set serves as an independent dataset to assess the model's performance on unseen data. To optimize the model's hyperparameters, a grid search with 5-fold cross-validation is employed. This process helps identify the best combination of hyperparameters (such as the number of neighbors, weight function, and distance metric).

The **Multi-Layer Perceptron** (**MLP**) model was trained using a structured approach, with training and validation datasets employed to ensure effective learning and generalization. Early stopping was incorporated to prevent overfitting, with the model's performance monitored using loss and accuracy metrics over 50 epochs. Additionally, the model's classification performance on the training data was analyzed using a confusion matrix to evaluate its ability to correctly predict class labels.

2. Performance Metrics

CNN

The evaluation is based on two primary metrics: loss and accuracy.

Loss Comparison (Model Loss Plot)

- The loss curve for the training data demonstrates a fluctuating yet decreasing trend, converging towards lower values as the epochs progress. This indicates that the model is learning effectively during training.
- The testing loss remains relatively stable and consistently low, suggesting that the model generalizes well without significant overfitting. The slight divergence between training and testing loss is normal and acceptable.

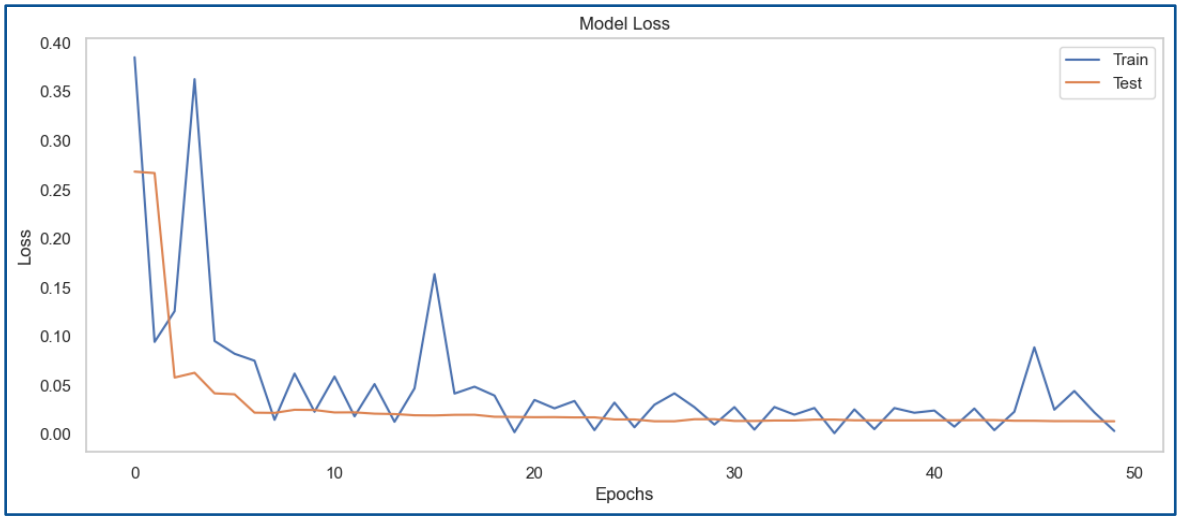

Figure 1: Loss plot CNN model

Accuracy Comparison (Model Accuracy Plot)

- The training accuracy increases over epochs, achieving near-perfect performance with minor fluctuations.
- The testing accuracy plateaus at a high value early on, indicating that the model reaches optimal performance quickly and maintains stability throughout the training process. The close alignment between the training and testing accuracy curves further reflects good generalization with minimal overfitting.

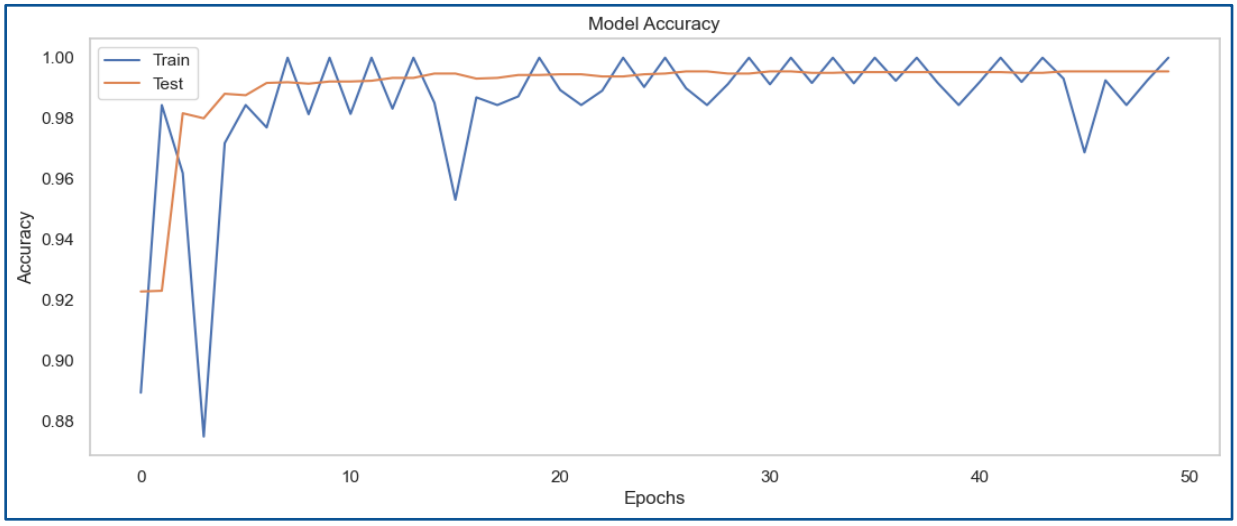

Figure 2: accuracy plot CNN model

After comparing the performance of various models, we found that the Convolutional

Neural Network (CNN) provided the best accuracy. Therefore, we have chosen to make the final predictions using the CNN model.

KNN

Confusion Matrix was used to assess the classification performance across all 10 classes.

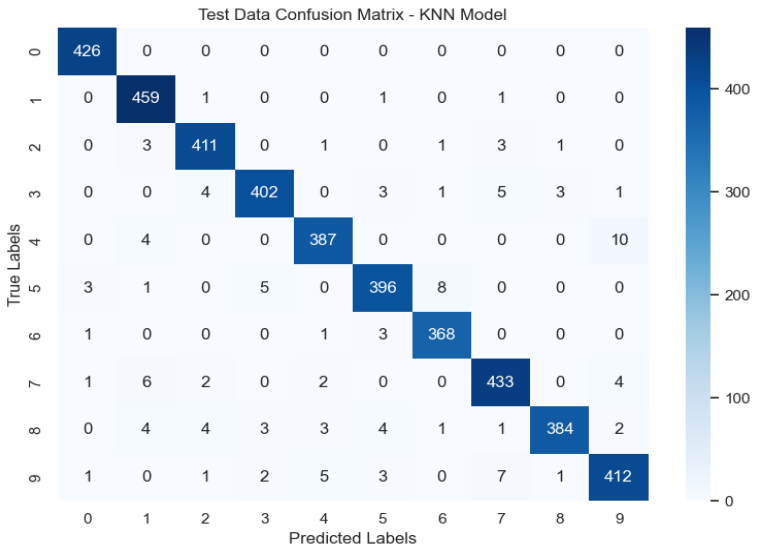

Figure 3: Confusion matrix KNN

MLP

Loss Comparison (Model Loss Plot)

- The training loss demonstrates a steep decline in the early epochs, stabilizing as the epochs progress. This reflects the model's ability to minimize error during the training phase effectively.
- The validation loss mirrors this trend, with a rapid decrease followed by stabilization at a low value. The alignment between training and validation loss curves indicates that the model generalizes well to unseen data, with minimal overfitting.

Accuracy Comparison (Model Accuracy Plot)

- Training accuracy steadily increases, achieving near-perfect accuracy in later epochs, demonstrating the model's ability to learn complex patterns in the training data.
- Validation accuracy rapidly improves in the early epochs and stabilizes at a high

level, highlighting the model's robustness and consistent performance on unseen data.

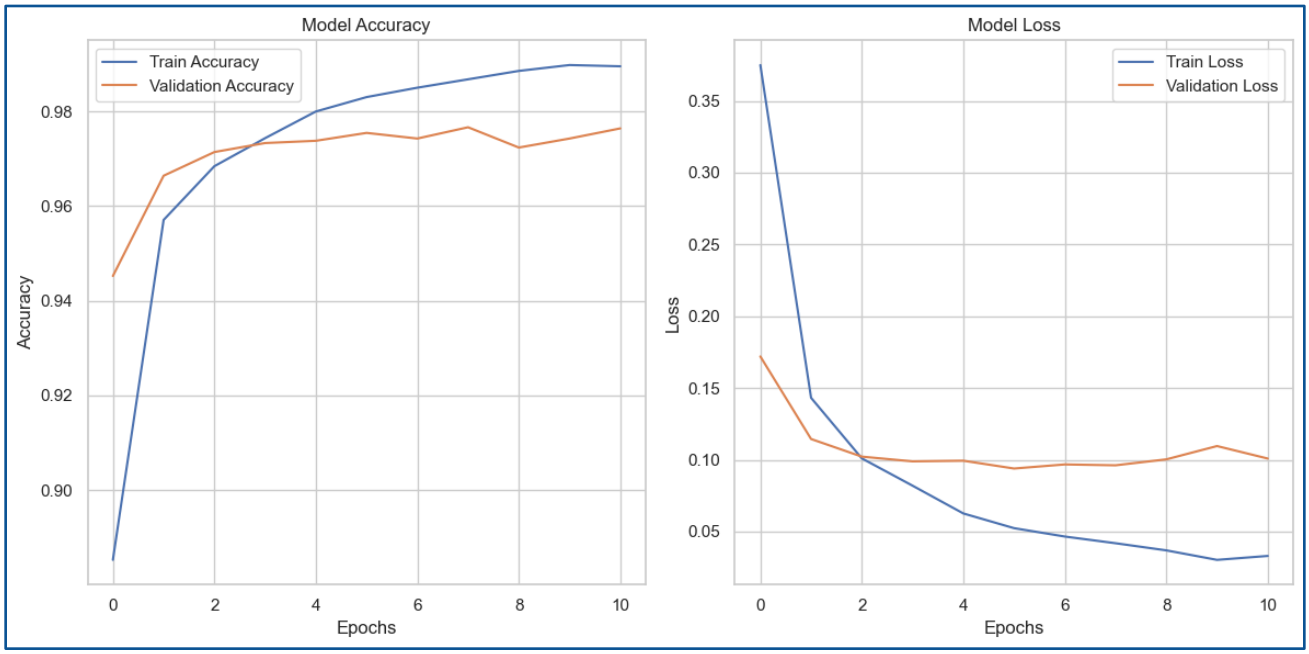

Figure 4: Accuracy and Loss MLP

Confusion Matrix for Training Data

- The confusion matrix generated for the training dataset provides a detailed view of the model's classification accuracy across all 10 classes.
- The heatmap visualization reveals high diagonal values, indicating that the majority of the true labels are correctly predicted by the model. Off-diagonal elements are minimal, signifying few misclassifications.

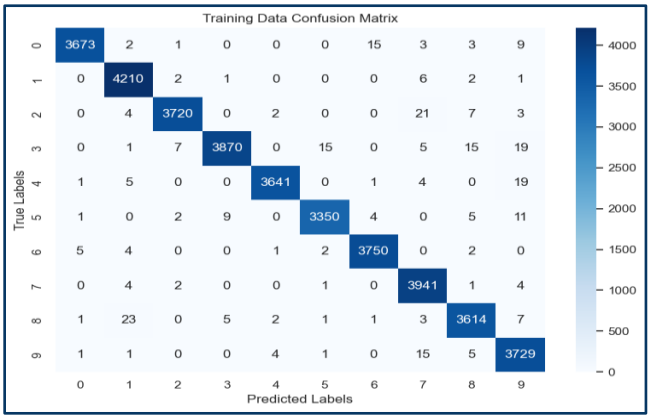

Figure 5: Confusion Matrix MLP

V. Conclusion & Lessons Learned

1. Key Insights Gained

CNN Effectiveness: Convolutional Neural Networks (CNNs) were highly effective for handwritten digit recognition, capturing spatial features efficiently.

Regularization: Dropout, batch normalization, and early stopping were essential to prevent overfitting and ensure better generalization.

Optimization Impact: Adam optimizer, learning rate decay, and early stopping improved convergence and model accuracy.

2. Challenges Encountered

Training Time: Training complex models like CNNs and SVMs was time-consuming.

Model Selection: Choosing the best-performing model required experimentation with different algorithms.

Data Preprocessing: Variations in handwriting and pixel noise posed challenges during preprocessing.

3. Lessons for Future Projects

Experimentation: Trying different models and optimization techniques is crucial for finding the best solution.

Regularization: Early use of regularization techniques helps prevent overfitting.

Hyperparameter Tuning: Automated hyperparameter optimization should be utilized for better performance.

Data Augmentation: Using data augmentation can improve generalization, especially with more complex datasets.

Computational Efficiency: Plan for the computational cost of training complex models to optimize time and resources.

In summary, this project emphasized the importance of regularization, optimization, and experimentation to build effective machine learning models. These lessons will be beneficial for future projects, especially in image classification tasks.