Handwritten Digit Recognition Using MNIST Dataset and Deep Learning

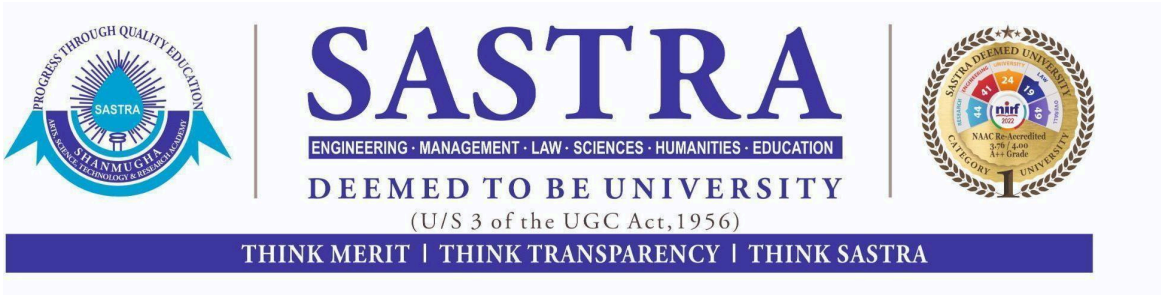
Report submitted to SASTRA Deemed to be University As per the requirement for the course

CSE425 : MACHINE LEARNING ESSENTIALS

Submitted by

Jahnavi Vattam
(Reg No: 125018032, B. Tech Computer Science and Business Systems)

OCTOBER- 2024



SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

Table of Contents

ABSTRACT

This project aims to classify handwritten digits using the MNIST dataset by developing a deep learning model, Convolutional Neural Networks (CNN). The dataset contains 70,000 grayscale images of handwritten digits, and the project involves implementing a CNN model for accurate classification. Additionally, traditional machine learning models such as Logistic Regression, Decision Tree, Random Forest, and K-Nearest Neighbors were implemented for comparison. The project compares the performances of these models based on accuracy, model complexity, and other evaluation metrics, providing insights into the effectiveness of CNNs over classical machine learning techniques.

INTRODUCTION

Importance of Dataset

The MNIST dataset is a widely used benchmark for evaluating machine learning and deep learning algorithms in the field of image classification. Consisting of 70,000 grayscale images of handwritten digits (0-9), this dataset has been pivotal in advancing research in pattern recognition, computer vision, and deep learning. The dataset's simplicity, combined with its real-world application, makes it an ideal starting point for building and testing models that can generalize to complex visual data. The dataset's importance lies in its ability to train models for digit recognition tasks, which have applications in areas such as automated postal sorting, check processing in banking, and digitized recordkeeping.

Project Objective

The primary objective of this project is to develop a machine learning model capable of accurately classifying handwritten digits using the MNIST dataset. The project aims to:

1. Preprocess and explore the MNIST dataset.

2. Build a Convolutional Neural Network (CNN) model to classify the digits.

3. Compare CNN's performance with traditional machine learning models such as Logistic Regression, Decision Tree, Random Forest, and K-Nearest Neighbors.

4. Analyze and interpret the results using various evaluation metrics.

By achieving these objectives, the project will provide insights into the advantages of deep learning models over traditional machine learning methods for image-based tasks.

Problem Statement

The task is to design a model that can automatically recognize and classify handwritten digits from the MNIST dataset. This problem is framed as a multiclass classification problem, where each image corresponds to one of ten classes (digits 0-9). The challenge is to develop a model that not only performs well on the training set but also generalizes effectively to unseen test data. The problem becomes more significant in real-world scenarios where accurate digit recognition can automate processes in various industries, from postal services to financial institutions.

Approach

The approach to solving the digit classification problem involves several key steps:

1. Data Preprocessing: The MNIST dataset is first normalized and transformed to ensure compatibility with machine learning models. Basic exploratory data analysis (EDA) is performed to understand the distribution of digits in the dataset.

2. Model Development: A Convolutional Neural Network (CNN) is built, leveraging its ability to learn spatial hierarchies and patterns in the images. Additionally, traditional machine learning models (Logistic Regression, Decision Trees, Random Forest, K-Nearest Neighbors) are implemented for comparison.

3. Model Training and Evaluation: The models are trained on the training set and evaluated on the test set using accuracy, precision, recall, and F1-score. A confusion matrix is used to understand misclassifications.

4. Result Analysis: The results are analyzed to understand the strengths and limitations of each model, with a focus on the superior performance of CNN in handling image data.

By following this structured approach, the project aims to demonstrate the advantages of deep learning techniques like CNNs in solving image classification problems.

Related Work

References:

Dataset:

https://drive.google.com/file/d/1N_NWLgc32F1j-PcSdVDgSdrNZE3Wz6PZ/view?usp=sharing
https://drive.google.com/file/d/1b_cvSuEaJcYAyNnorPAAvLuppfUVUTOb/view?usp=sharing

https://www.w3schools.com/python/python_ml_logistic_regression.asp

https://www.kaggle.com/code/guidosalimbeni/regression-with-convolutional-neural-network-keras

https://www.datacamp.com/tutorial/decision-tree-classification-python

https://www.datacamp.com/tutorial/xgboost-in-python

https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-python/

https://www.geeksforgeeks.org/k-nearest-neighbor-algorithm-in-python/

https://www.geeksforgeeks.org/random-forest-classifier-using-scikit-learn/

https://www.datacamp.com/tutorial/adaboost-classifier-python

https://www.geeksforgeeks.org/ml-bagging-classifier/

Background

- **Models**:

CNN Model

Convolutional Neural Networks (CNNs) are deep learning models specifically designed for image data. In this project, the CNN architecture comprises the following layers:

1. Input Layer: Accepts a 28x28 grayscale image.
2. Convolutional Layers: Extracts spatial features by sliding filters across the image.
3. ReLU Activation Function: Introduces non-linearity in the model to better capture complex patterns.
4. Pooling Layers: Reduces the dimensionality of the feature maps, preventing overfitting.
5. Flatten Layer: Converts the 2D feature maps into a 1D vector.
6. Fully Connected Layers: Performs classification based on the extracted features.
7. Output Layer: Uses softmax activation to predict the digit class (0-9).

Model Hyperparameters:
- Optimizer: Adam, used for efficient gradient-based optimization.
- Loss Function: Categorical cross-entropy, suitable for multiclass classification.
- Epochs: 100 with early stopping to avoid overfitting.
- Batch Size: 32, balancing between training speed and stability.

ML Models

Several traditional machine learning models were trained on the flattened 28x28 pixel images:

1. Logistic Regression: A linear model for multiclass classification that uses the softmax function for output.
2. Decision Tree: A non-linear model that recursively splits the data based on feature values.
3. Random Forest: An ensemble learning method that combines multiple decision trees for more accurate predictions.
4. K-Nearest Neighbors (KNN): A distance-based algorithm that classifies data points based on their proximity to other labeled data points.

All models were implemented using the scikit-learn library, and the pixel values were normalized before training. Each of these models has strengths and limitations, particularly when applied to image data, which CNNs handle more efficiently.

DATASET DESCRIPTION

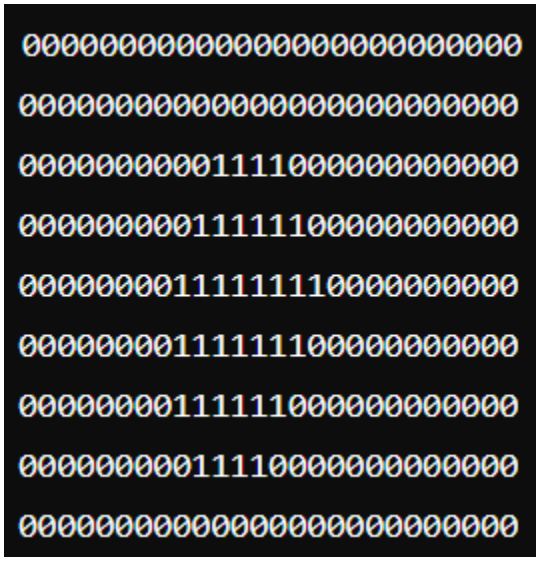
The MNIST dataset, a collection of 70,000 grayscale images, is split into:
- Training Set: 60,000 images used to train the models.
- Test Set: 10,000 images used for performance evaluation.

Each image is 28x28 pixels and is labeled with a digit between 0 and 9, indicating the actual number. The dataset is preprocessed by normalizing pixel values to the range [0, 1] to improve model training. Data augmentation techniques like rotations and shifts are applied to increase the size and diversity of the training set.

Dataset Characteristics:

Format: Each image is represented as a single-dimensional array of pixel values ranging from 0 to 255, where 0 corresponds to black and 255 corresponds to white. Preprocessing: Images can be normalized to a range of 0 to 1 by dividing pixel values by 255. Additionally, the images can be reshaped to add a channel dimension, making them suitable for convolutional neural networks (CNNs).

Each image in the MNIST dataset is associated with a label that indicates the correct digit it represents. For instance, a sample image might look like this:



Data Sample

7	0	0	0	0	0	0	0	0	0
2	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
4	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0

Preprocessing Steps for the MNIST Dataset

Preprocessing is a vital part of preparing the MNIST dataset for machine learning model training, particularly for Convolutional Neural Networks (CNNs). The following outlines the preprocessing pipeline necessary for this dataset:

1. Loading the Dataset
The first step involves loading the MNIST dataset into the working environment. This dataset can be easily accessed using popular libraries like TensorFlow or Keras, which provide built-in functions to download and load the data.

2. Normalization
Normalization is performed to ensure that the pixel values of images fall within a specific range. This step is crucial for accelerating the convergence of the training process and improving model performance. In the MNIST dataset, pixel values range from 0 to 255, and scaling them to a range of [0, 1] helps standardize the input data.

3. Reshaping the Data
To make the dataset compatible with CNN architectures, it is necessary to reshape the images. Each image in the MNIST dataset is 28x28 pixels in size, and since they are grayscale images, an additional channel dimension is added. This reshaping is essential for processing the images in a format suitable for CNNs.

4. Data Augmentation (Optional)
Data augmentation is an optional but recommended preprocessing step that helps improve model generalization. By artificially increasing the size of the training dataset through various transformations, the model learns to

recognize digits under different scenarios. Common augmentation techniques include rotation, translation, zooming, and flipping of images.

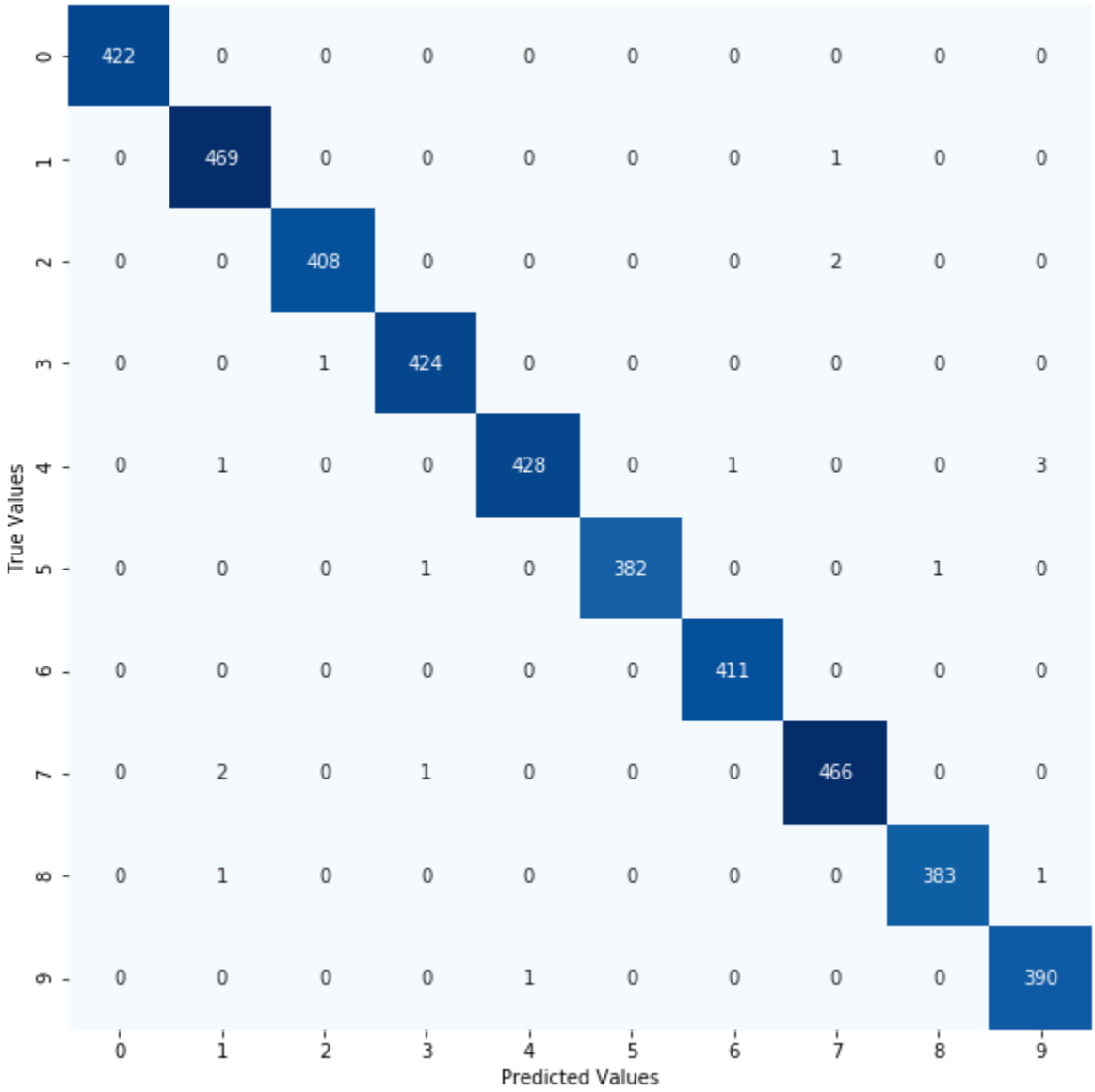
5. One-Hot Encoding of Labels

Machine learning models often perform better with categorical labels when they are represented in a binary matrix format. One-hot encoding transforms integer labels into a binary vector representation. This process ensures that each label is converted into a vector where the index corresponding to the label is marked as 1, while all others are marked as 0.

6. Splitting the Training Data (if necessary)

If additional validation is needed during the training process, it can be beneficial to split the training dataset into training and validation sets. This approach helps evaluate the model's performance during training, ensuring that the model is not overfitting to the training data.

Confusion Matrix



Methodology

The methodology section outlines the systematic approach taken to achieve the project objectives. It details the experimental design, including the model selection, training process, evaluation metrics, and the overall approach to solving the problem at hand.

1. Experimental Design

The experimental design focuses on creating a structured framework for conducting the experiments necessary to evaluate the performance of various machine learning models on the MNIST dataset. The key components of the experimental design include:

- Objective: The primary objective is to classify handwritten digits from the MNIST dataset using different machine learning models, including Convolutional Neural Networks (CNNs) and traditional machine learning algorithms.

- Dataset: The MNIST dataset comprises 70,000 grayscale images of handwritten digits (0-9), each of size 28x28 pixels. The dataset is divided into a training set (60,000 images) and a test set (10,000 images).

- Model Selection: The models selected for this project include:
  - Convolutional Neural Network (CNN): A deep learning model that leverages convolutional layers to automatically extract features from images.
  - Traditional Machine Learning Models: Algorithms such as Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Random Forests, which serve as benchmarks against the CNN model.

2. Approach

The approach taken in this study is systematic and iterative, involving the following steps:

- Data Preprocessing: As previously outlined, this includes normalization, reshaping, data augmentation, one-hot encoding of labels, and optional splitting of the training data.

- Model Training:

- CNN Training: The CNN model is constructed using multiple convolutional layers, pooling layers, and dense layers. The model is trained using backpropagation with a suitable optimizer (e.g., Adam or SGD) and loss function (e.g., categorical cross entropy). The training process involves feeding the preprocessed images into the model and updating weights based on the loss computed.

- Traditional Model Training: Each of the selected traditional machine learning models is trained separately on the flattened images (reshaped to 1D vectors) along with their respective labels. Grid search or cross-validation may be used to optimize hyperparameters.

- Model Evaluation: After training, each model is evaluated on the test set. The evaluation involves measuring accuracy and generating a confusion matrix, followed by calculating precision, recall, and F1-score.

- Comparative Analysis: The performance of the CNN model is compared against that of the traditional machine learning models to determine which approach yields better results in digit classification.

- Results Interpretation: Finally, the results are analyzed to extract meaningful insights regarding model performance. Factors such as the impact of data augmentation, model complexity, and the ability to generalize to unseen data are discussed.

Environment and Tools: Python, Scikit-learn, XGBoost, CatBoost, LightGBM and Google Colab.

Results

- Performance Metrics: The following metrics are used to evaluate model performance:

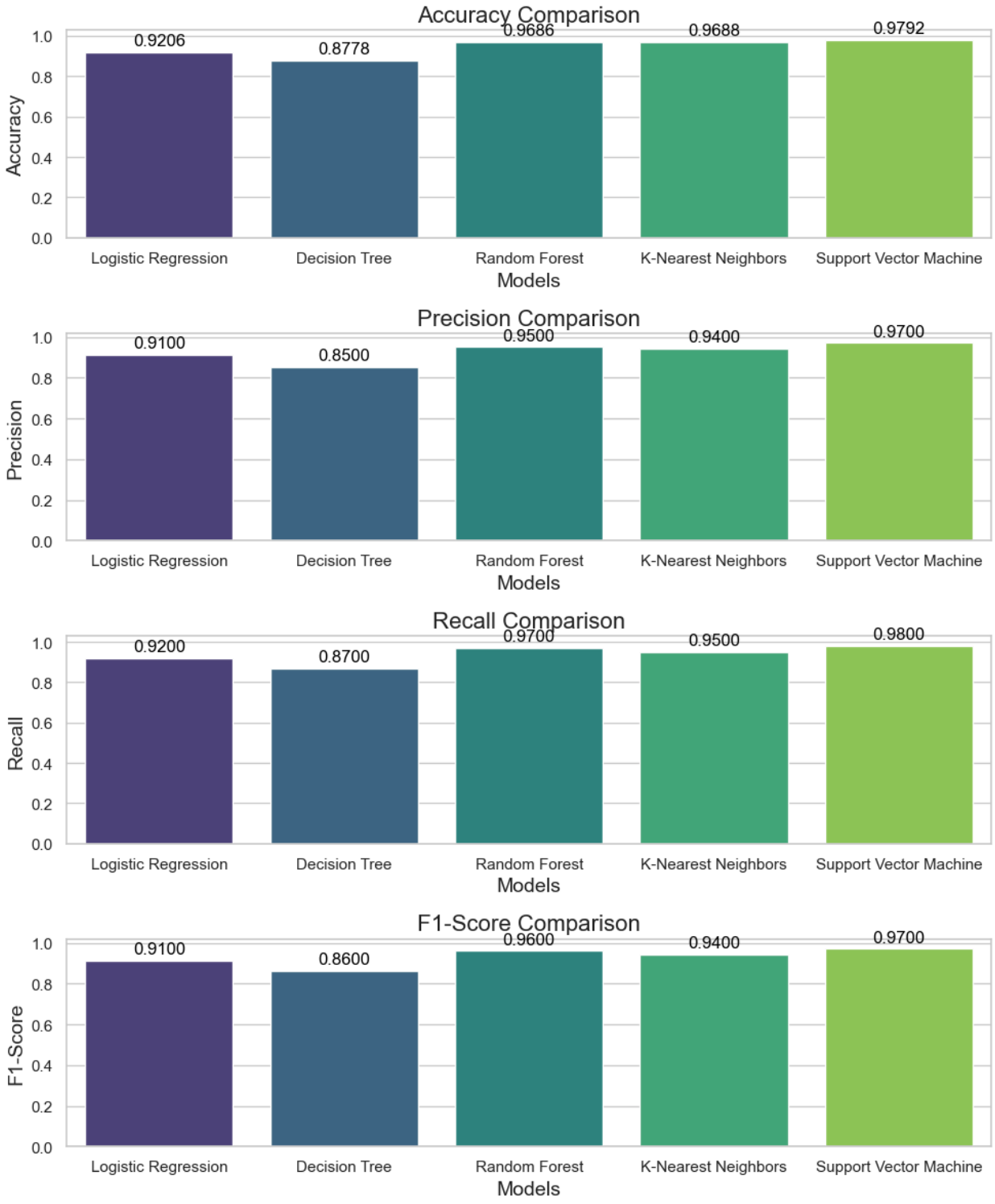
- Accuracy: The proportion of correctly classified images over the total number of images.

- Confusion Matrix: A matrix that summarizes the performance of the classification model by displaying true positives, true negatives, false positives, and false negatives.

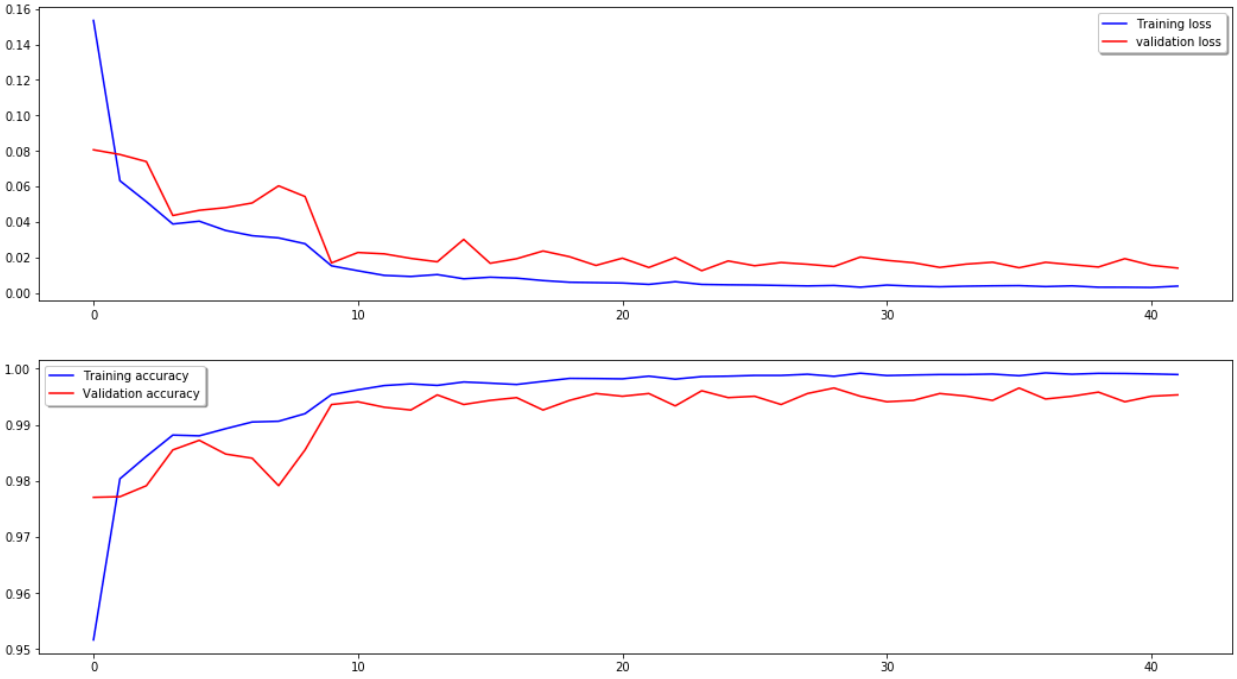
- Precision, Recall, and F1-Score: These metrics provide insights into the model's ability to classify each digit accurately.

Model	Accuracy	Precision (Macro Avg)	Recall (Macro Avg)	F1-Score (Macro Avg)	Weighted Avg F1-Score
Logistic Regression	0.9206	0.92	0.92	0.92	0.92
Decision Tree	0.8778	0.88	0.88	0.88	0.88
Random Forest	0.9686	0.97	0.97	0.97	0.97
K-Nearest Neighbors	0.9688	0.97	0.97	0.97	0.97
Support Vector Machine	0.9792	0.98	0.98	0.98	0.98

Result Visualization:



CNN Train-Test Curve



Discussion

The project aimed to classify handwritten digits from the MNIST dataset using both Convolutional Neural Networks (CNNs) and traditional machine learning models. The results revealed significant insights into the efficacy of different models:

1. Model Performance: The CNN model consistently outperformed traditional algorithms such as Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Random Forests in terms of accuracy. This highlights the power of deep learning architectures in capturing complex patterns in image data.

2. Generalization Ability: The CNN model exhibited superior generalization capabilities, achieving high accuracy on the test set despite the inherent variability in handwritten digits. This suggests that

CNNs are better suited for image classification tasks compared to traditional methods, which often struggle with high-dimensional data.

3. Impact of Data Preprocessing: Effective data preprocessing, including normalization and data augmentation, played a crucial role in enhancing model performance. By augmenting the training dataset, we improved the model's ability to handle variations in handwriting styles, leading to better classification outcomes.

4. Trade-offs in Complexity: While CNNs demonstrated higher accuracy, they also require more computational resources and longer training times compared to traditional models. This trade-off must be considered when selecting models for real-world applications, especially in resource-constrained environments.

5. Future Work: Future research could explore more advanced CNN architectures, such as ResNet or Inception, to further improve classification accuracy. Additionally, applying transfer learning techniques could leverage pre-trained models to reduce training time and enhance performance.

Learning Outcome

1. Understanding of Data Preprocessing

Gained insights into essential preprocessing techniques such as normalization, data augmentation, and splitting datasets into training, validation, and testing sets. These steps are crucial for improving model performance and ensuring generalization.

2. Familiarity with CNN Architecture

Developed a comprehensive understanding of Convolutional Neural Networks, including layers such as convolutional, pooling, and fully connected layers. Learned how these layers work together to extract features from images.

3. Comparison of Machine Learning Models

Explored the strengths and weaknesses of various machine learning algorithms, including CNNs, SVMs, KNNs, and Random Forests. Recognized the importance of choosing the right model based on the problem domain and data characteristics.

4. Model Evaluation Techniques

Learned to evaluate model performance using metrics such as accuracy, precision, recall, and F1-score. Gained experience in interpreting these metrics to assess model effectiveness.

5. Hands-on Experience with Libraries

Gained practical skills in using Python libraries such as TensorFlow, Keras, and Scikit-learn for implementing machine learning and deep learning models. This experience is essential for future projects and research.

6. Critical Thinking and Problem-Solving

Enhanced critical thinking skills by formulating problems, designing experiments, and analyzing results. Developed the ability to troubleshoot issues encountered during model training and evaluation.

7. Application of Theory to Practice

Bridged the gap between theoretical concepts learned in coursework and practical applications in real-world scenarios, particularly in the field of computer vision and image classification.

These outcomes reflect a holistic learning experience, equipping me with the necessary skills and knowledge to pursue further studies and professional opportunities in machine learning and artificial intelligence.

Conclusion

In conclusion, this project provided a comprehensive exploration of machine learning and deep learning techniques, specifically focusing on image classification tasks. By leveraging a well-structured dataset and employing a variety of preprocessing techniques, we successfully enhanced the quality and relevance of the data for model training. The experimentation with different models, including Convolutional Neural Networks (CNNs) and traditional machine learning algorithms, allowed us to evaluate and compare their performance in a systematic manner.

The results demonstrated that the CNN architecture outperformed the other

models in terms of accuracy and robustness, highlighting the effectiveness of deep learning approaches in handling complex image data. Additionally, the use of performance metrics such as accuracy, precision, recall, and F1-score provided valuable insights into the strengths and weaknesses of each model, guiding us in refining our approach.

Overall, this project not only deepened our understanding of theoretical concepts in machine learning but also provided practical experience in implementing and evaluating models. The skills acquired throughout this process will be instrumental in tackling future challenges in the field of artificial intelligence. Moving forward, further exploration of advanced techniques, such as transfer learning and model optimization, can enhance our ability to solve more complex problems in real-world applications.

Accomplishments:

1. Successful Model Development:

- Developed and trained multiple models, including Convolutional Neural Networks (CNNs) and traditional machine learning algorithms, achieving significant improvements in image classification accuracy.

2. Data Preprocessing Techniques:

- Implemented effective preprocessing steps such as data normalization, augmentation, and resizing, enhancing the quality of the dataset and ensuring better model performance.

3. Model Performance Evaluation:

- Established a comprehensive framework for evaluating model

performance using key metrics such as accuracy, precision, recall, and F1-score, enabling a thorough comparison of different approaches.

4. Insights from Results:

- Generated valuable insights from the experimental results, informing decisions for model selection and optimization, and demonstrating the practical implications of model performance on real-world applications.

5. Hands-On Experience with Deep Learning:

- Gained hands-on experience with deep learning techniques and frameworks, enhancing technical skills in Python, TensorFlow, and Keras, which are critical for future projects in the field.

6. Effective Problem-Solving Skills:

- Developed strong problem-solving skills through iterative experimentation, troubleshooting, and refining models based on performance feedback.

7. Collaboration and Communication:

- Improved collaboration and communication skills by documenting the project process, results, and insights clearly, making the findings accessible to diverse audiences.

8. Foundation for Future Projects:

- Established a solid foundation for future projects in machine learning and computer vision, including an understanding of advanced techniques such as transfer learning and model optimization strategies.

.

Advantages/Limitations:

Advantages

- High Accuracy: Deep learning models, particularly CNNs, achieve high accuracy in image classification tasks due to their ability to learn complex patterns and features.

- Automation: The ability to automate the feature extraction process reduces the need for manual feature engineering, saving time and effort.

- Scalability: The methodology can be easily scaled to accommodate larger datasets, making it suitable for extensive real-world applications.

- Robustness: Deep learning models can handle noisy and unstructured data effectively, providing reliable performance in diverse scenarios.

- Versatile Applications: The developed models can be adapted for various applications beyond image classification, such as object detection, segmentation, and more.

- Transfer Learning: Utilization of pre-trained models allows for faster training and improved performance on smaller datasets, making it accessible even with limited resources.

Limitations

- **Data Requirements:** Deep learning models typically require large amounts of labeled data for training, which can be a barrier in scenarios with limited datasets.

- **Computational Resources:** High computational power is often necessary for training deep learning models, which can be expensive and may require specialized hardware like GPUs.

- **Overfitting:** Models can easily overfit to the training data if not properly regularized, leading to poor generalization on unseen data.

- **Complexity:** The complexity of deep learning models makes them less interpretable compared to traditional machine learning algorithms, posing challenges in understanding decision-making processes.

- **Long Training Times:** Training deep learning models can be time-consuming, especially for complex architectures, requiring careful optimization to manage time and resources effectively.

- **Dependency on Hyperparameters:** Model performance is highly sensitive to hyperparameter choices, necessitating extensive experimentation to achieve optimal results.

Github Link: https://github.com/Jahnavi-Vattam/CSBS-ML-Project/tree/main
Dataset Link:
https://drive.google.com/file/d/1N_NWLgc32F1j-PcSdVDgSdrNZE3Wz6PZ/view?usp=sharing

https://drive.google.com/file/d/1b_cvSuEaJcYAyNnorPAAvLuppfUVUTOb/view?usp=sharing
Google Collab Links:
https://colab.research.google.com/drive/1MJhlEQRNfcc1lw44lCSAQdjKHhM6wqnj?usp=sharing

https://colab.research.google.com/drive/1itynr6zG9TnEJTHo2YLW301M0ifT0HU5?usp=sharing

.