NAME: KANDURI RAMA NARASIMHA

REG NO: 22BCE7449

VIT-AP University

MNIST Digit Classification: Deep Learning Approach for Efficient Handwritten Digit Recognition

ABSTRACT: The MNIST Digit Classification project addresses a core problem in computer vision and machine learning: the accurate recognition of handwritten digits. The MNIST dataset, a widely accepted benchmark, contains 70,000 grayscale images of digits (0–9), each sized at 28×28 pixels. Despite its simplicity, the dataset reflects diverse handwriting styles, making it suitable for real-world applications such as automated form entry, postal code reading, and financial document processing, this project explores and compares multiple classification techniques, integrating both traditional machine learning and modern deep learning approaches. Classical models—including k-Nearest Neighbours (k-NN), Support Vector Machines (SVM), Decision Trees, and Random Forests—are implemented using Scikit-learn, while a Convolutional Neural Network (CNN) is developed using PyTorch to leverage deep learning's feature extraction capabilities. Tools such as NumPy, Pandas, Matplotlib, VSCode, and Jupyter Notebooks support the workflow.

The process begins with data preprocessing, including normalization and reshaping for model compatibility. Traditional models receive flattened pixel vectors and are evaluated using cross-validation and hyperparameter tuning methods like grid search and random search. For CNNs, the architecture includes convolutional, pooling, dropout, and dense layers, culminating in a softmax output layer. The model is trained using the Adam optimizer and categorical cross-entropy loss, with data augmentation techniques applied to enhance generalization and prevent overfitting.A thorough Exploratory Data Analysis (EDA) confirms a balanced class distribution and validates the dataset's integrity. Evaluation across models uses metrics such as accuracy, precision, recall, F1-score, and confusion matrices. Results indicate that while traditional models perform well—with Random Forest achieving 97.07% and SVM 96.61%—the CNN outperforms all others with 98.97% accuracy, thanks to its ability to automatically learn hierarchical image features.

This comparative study not only highlights the strengths and limitations of different classification strategies but also emphasizes the superiority of deep learning in image-based tasks. The insights and techniques developed are broadly applicable to industries like finance, logistics, and digital document processing, where accurate digit recognition is critical.

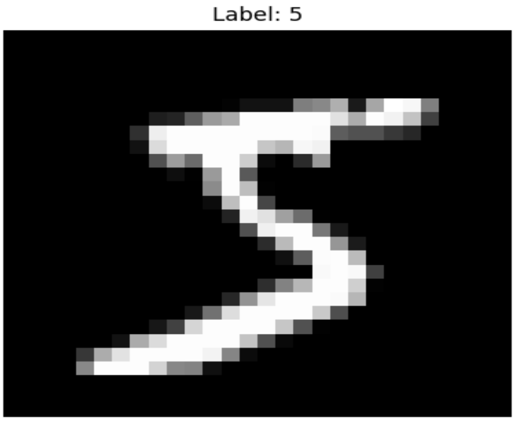
Table of contents

Handwritten digit recognition is a classic problem in computer vision that involves identifying numeric digits (0–9) from images. As digitized documentation and automation become increasingly common in industries such as banking, logistics, and education, accurate digit classification plays a crucial role in reducing manual errors and processing time. This project explores various machine learning and deep learning models to classify handwritten digits using the MNIST dataset—one of the most widely used benchmark datasets in the field.

Problem Statement

Despite the simplicity of digit recognition, variations in handwriting, image noise, and digit similarity make accurate classification a non-trivial task. The main challenge lies in developing a model that not only achieves high accuracy but is also robust, scalable, and capable of generalizing well across diverse handwriting styles. This project aims to investigate and compare traditional machine learning techniques with advanced deep learning architectures to determine the most effective approach for the MNIST digit classification problem.

Given below is example of handwritten digits in pixels



Objectives

- To perform exploratory data analysis (EDA) on the MNIST dataset to understand data distribution, structure, and challenges.

- To implement and evaluate traditional machine learning models such as k-Nearest Neighbors (k-NN), Support Vector Machine (SVM), Decision Tree, and Random Forest.
- To design and train a Convolutional Neural Network (CNN) using PyTorch for digit classification.
- To compare the performance of classical and deep learning models using metrics like accuracy, precision, recall, F1-score, and confusion matrices.
- To draw insights into the suitability of different classification approaches for real-world applications.

Scope of the Project

This project is limited to classification of the 10-digit classes (0–9) using the MNIST dataset. It includes:

- Data preprocessing (normalization, reshaping)
- EDA and visualization
- Implementation of traditional ML models using Scikit-learn
- Design and training of a CNN model using PyTorch and CUDA
- Evaluation and comparison of all models using consistent metrics
- No real-time digit recognition or deployment is included in the current scope, though the trained models can be extended for such applications.
-

Background / Context

The MNIST dataset, created by Yann LeCun et al., is derived from the NIST Special Database and contains 70,000 28x28 pixel grayscale images of handwritten digits. It has become a standard testbed for machine learning algorithms due to its manageable size, well-balanced class distribution, and relevance to real-world tasks like postal code recognition and check processing.

Over the years, MNIST has evolved from being a challenge to becoming a teaching and benchmarking tool for supervised learning and neural networks. Classic models like SVMs and Random Forests have demonstrated strong performance on MNIST, but the advent of deep learning—especially CNNs—has set new benchmarks in accuracy and feature extraction. This project uses MNIST to bridge foundational machine learning concepts with modern deep learning practices.

Literature Review

The MNIST dataset has been extensively used as a benchmark for handwritten digit recognition tasks in machine learning research. It consists of 60,000 training images and 10,000 testing images, each depicting digits from 0 to 9 in grayscale. Early research applied traditional machine learning techniques such as Support Vector Machines (SVM) (Cortes & Vapnik, 1995), K-Nearest Neighbors (KNN), and Decision Trees to MNIST, achieving moderate accuracy but often relying on handcrafted features and preprocessing (Duda et al., 2000). The introduction of Convolutional Neural Networks (CNNs), particularly the LeNet-5 architecture by LeCun et al. (1998), marked a significant milestone by enabling end-to-end learning directly from pixel data, which substantially improved classification accuracy on MNIST. Subsequent work has focused on improving CNN architectures and training strategies, including deeper networks, dropout regularization (Srivastava et al., 2014), and batch normalization (Ioffe & Szegedy, 2015), resulting in near-human or superhuman performance on this dataset. Additionally, data augmentation techniques have been employed to increase the robustness and generalization capabilities of models (Simard et al., 2003). Despite the MNIST dataset's simplicity compared to more complex image recognition challenges, it remains a crucial resource for benchmarking and exploring new methodologies in the field.

Dataset description

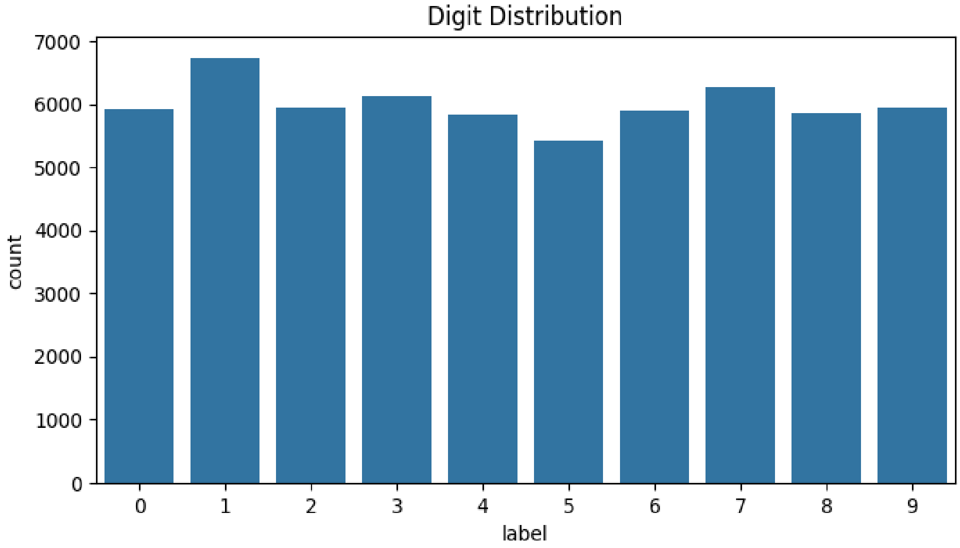
The dataset used in this project is the MNIST (Modified National Institute of Standards and Technology) database. It is a widely used benchmark dataset in machine learning and computer vision, particularly for image classification tasks.

- Train file: mnist_train.csv
- Test file: mnist_test.csv
- Download Source: Kaggle

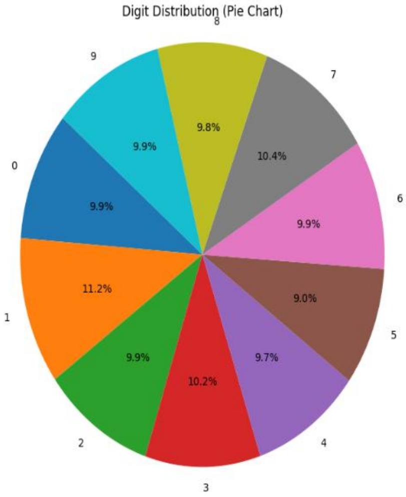
Exploratory data analysis

The dataset has no empty data and data is clean almost
Digits are distributed almost equally

Data distribution in bar graph



- Data distribution in pie graph



Size, Number of Features, and Data Types

Dataset	Shape	Description
Train	(60,000, 785)*4	60,000 rows, 784 pixel features + 1 label
Test	(10,000, 785)	10,000 rows, 784 pixel features + 1 label

Feature Breakdown:

- **Label**:
  - **Type**: Categorical
  - **Values**: Integer digits from 0 to 9
  - **Purpose**: Target variable (what digit the image represents)
- **Pixels (pixel1, pixel2, ..., pixel784)**:
  - **Type**: Numerical (Integer values from 0 to 255)
  - **Meaning**: Grayscale intensity of each pixel in a 28x28 image
  - **Shape**: Each row represents a 28×28 (flattened to 784) grayscale image

Data Augmentation and Generalisation

Data augmentation is a critical technique in the field of machine learning, particularly in image classification tasks. It involves the systematic application of various transformations to the training data in order to artificially expand its size and variability. This process enables the model to encounter a broader range of scenarios during training without the need to collect additional data. For example, in the case of handwritten digit recognition, slight modifications to digit images such as rotation or shifting can significantly improve model robustness.

The primary objective of data augmentation is to enhance the **generalisation** ability of a model. Generalisation refers to the model's capacity to perform well on unseen data, rather than merely memorising the training examples. Without adequate generalisation, a model may overfit, meaning it learns the noise or specific patterns in the training data rather than the underlying distribution. By introducing augmented examples that reflect plausible variations of the input data, the model is encouraged to focus on more general and meaningful features, such as the structural characteristics of digits, rather than their specific positions or sizes.

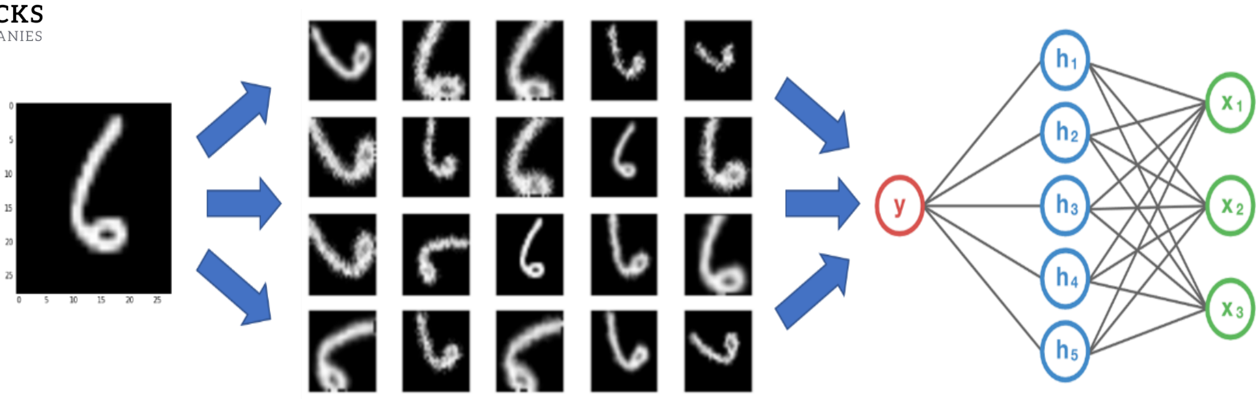
In the context of digit classification tasks such as MNIST, data augmentation plays a crucial role in enhancing the model's ability to generalize by artificially increasing the diversity of the training dataset. This is especially important given the limited size and variability of the original dataset. The following data augmentation techniques are commonly applied and particularly effective for handwritten digit recognition:

1. **Rotation:** Applying small random rotations, typically within a range of plus or minus ten degrees, allows the model to become invariant to the orientation of the digits. This

simulates the natural variability in handwriting where digits may be slightly tilted or written at an angle.

2. **Translation:** Slightly shifting the digit images horizontally or vertically ensures that the model is robust to positional variations. This technique helps the model correctly classify digits that are not perfectly centred within the image frame, mimicking real-world scenarios where handwritten digits may appear off-centre.
3. **Scaling or Zooming:** Modifying the size of the digits by zooming in or out simulates differences in writing styles and font sizes. This enables the model to handle digits written larger or smaller than average, thereby improving its flexibility in recognizing digits across various scales.
4. **Shearing:** Introducing shearing transformations distorts the images by slanting the digits in one direction. This augmentation mimics slanted or cursive handwriting styles and improves the model's ability to recognize digits despite such distortions.
5. **Adding Gaussian Noise:** Incorporating Gaussian noise into images helps the model learn to focus on the essential features of the digits rather than being distracted by irrelevant pixel-level variations. This technique enhances the model's robustness against noisy or low-quality images that may be encountered in practical applications.
6. **Horizontal and Vertical Flipping:** Although less common in digit recognition because some digits are asymmetric, controlled flipping can sometimes help the model generalize better, especially for datasets with specific variations.
7. **Elastic Distortions:** This advanced augmentation method applies smooth, random deformations to the image, mimicking the elastic nature of handwriting. Elastic distortions have been shown to significantly improve performance in handwritten character recognition tasks by providing realistic variability.
8. **Brightness and Contrast Adjustments:** Altering the brightness and contrast of images simulates different lighting conditions during image capture, helping the model become invariant to such environmental factors.

It is important to note that these transformations should preserve the semantic content of the images. For instance, applying horizontal flipping would not be appropriate for digit classification, as it may alter the digit in a way that changes its label (e.g., flipping a "6" may resemble a "9" or an unrecognisable symbol).



In summary, data augmentation plays a vital role in improving the model's performance by supporting generalisation, reducing overfitting, and enabling more effective learning from limited data. The choice of augmentation techniques must be carefully considered to maintain the integrity of the original data while promoting variability that reflects realistic conditions.

We use data augmentation for training data to make model more generalise

Models used in Project

K-Nearest Neighbors (KNN)

K-Nearest Neighbors is a straightforward, instance-based learning algorithm that classifies a data point based on the majority class among its nearest neighbors in the feature space. It does not require an explicit training phase but relies on calculating distances between samples during prediction. KNN is well-suited for datasets where similar data points tend to belong to the same class.

KNN serves as a simple baseline model for image classification on MNIST. Since the dataset contains images of handwritten digits with clear visual similarities within classes, KNN’s proximity-based approach can effectively differentiate digits based on pixel-level similarity.

Support Vector Machine (SVM)

SVM is a powerful supervised learning algorithm that constructs an optimal hyperplane to separate classes by maximizing the margin between them. It can efficiently handle linear and non-linear classification through kernel functions, making it adaptable to various data distributions.

SVM is selected for MNIST because of its proven high performance in digit recognition tasks, especially when using non-linear kernels such as the radial basis function (RBF). It excels at handling high-dimensional data and provides robust classification boundaries, which are essential for distinguishing subtle differences between handwritten digits.

Decision Tree

Decision Trees are supervised learning models that recursively split the dataset based on feature values to create a tree structure. Each node represents a decision rule, making the model highly interpretable and capable of capturing non-linear relationships.

Decision Trees are included to offer an interpretable model for MNIST classification. Despite not always matching the accuracy of other methods, they provide insight into which features (pixels) contribute most to classification decisions and serve as a useful baseline for comparison.

Random Forest

Random Forest is an ensemble learning method that builds multiple decision trees using random subsets of data and features, aggregating their outputs to improve accuracy and control overfitting. It combines the benefits of multiple trees to yield more stable and generalizable predictions.

Random Forest is chosen to address the limitations of single decision trees by reducing variance and improving classification performance on MNIST. Its ensemble nature helps capture diverse feature interactions, enhancing robustness and accuracy compared to individual trees.

Convolutional Neural Network (CNN)

CNNs are deep learning models specifically designed for image data. They use convolutional layers to automatically learn hierarchical spatial features and patterns directly from pixel data, effectively capturing edges, shapes, and textures relevant to classification.

CNNs are the state-of-the-art approach for image classification tasks like MNIST due to their ability to learn complex representations from raw images without manual feature engineering. Their architecture is well-suited to handle the spatial structure of handwritten digits, resulting in superior accuracy compared to traditional models.

System Requirements

This project requires a basic computing setup capable of running both classical machine learning algorithms and deep learning models using PyTorch. The following are the hardware and software requirements.

Hardware Requirements

Component	Minimum Requirement	Recommended Requirement
Processor	Intel i3 / AMD Ryzen 3	Intel i5/i7 or AMD Ryzen 5+
RAM	4 GB	8 GB or more
Storage	5 GB free space	SSD recommended for speed
GPU (Optional)	Not required for basic ML models	NVIDIA GPU with CUDA support (e.g., GTX 1050 or higher) for faster CNN training

Note: CNN training on a CPU is possible but slower compared to using a GPU.

Software Requirements

Software	Version/Details
Operating System	Windows, Linux, or macOS
Python	Version 3.7 or higher
Jupyter Notebook or VSCode	For interactive development
PyTorch	Version 2.0 or compatible
Scikit-learn	Version 1.0 or higher
TensorFlow/Keras	Optional (not used in this PyTorch-based project)
Matplotlib, Seaborn	For data visualization
NumPy, Pandas	For data handling and processing

Dependencies and Libraries Used

- numpy
- pandas
- matplotlib
- seaborn
- scikit-learn
- torch
- torchvision
- vscode (as development environments)

Methodology / Design

Overview of System Architecture

The project follows a modular architecture combining data preprocessing, model training, evaluation, and comparison. It includes both classical machine learning and deep learning (CNN) approaches.

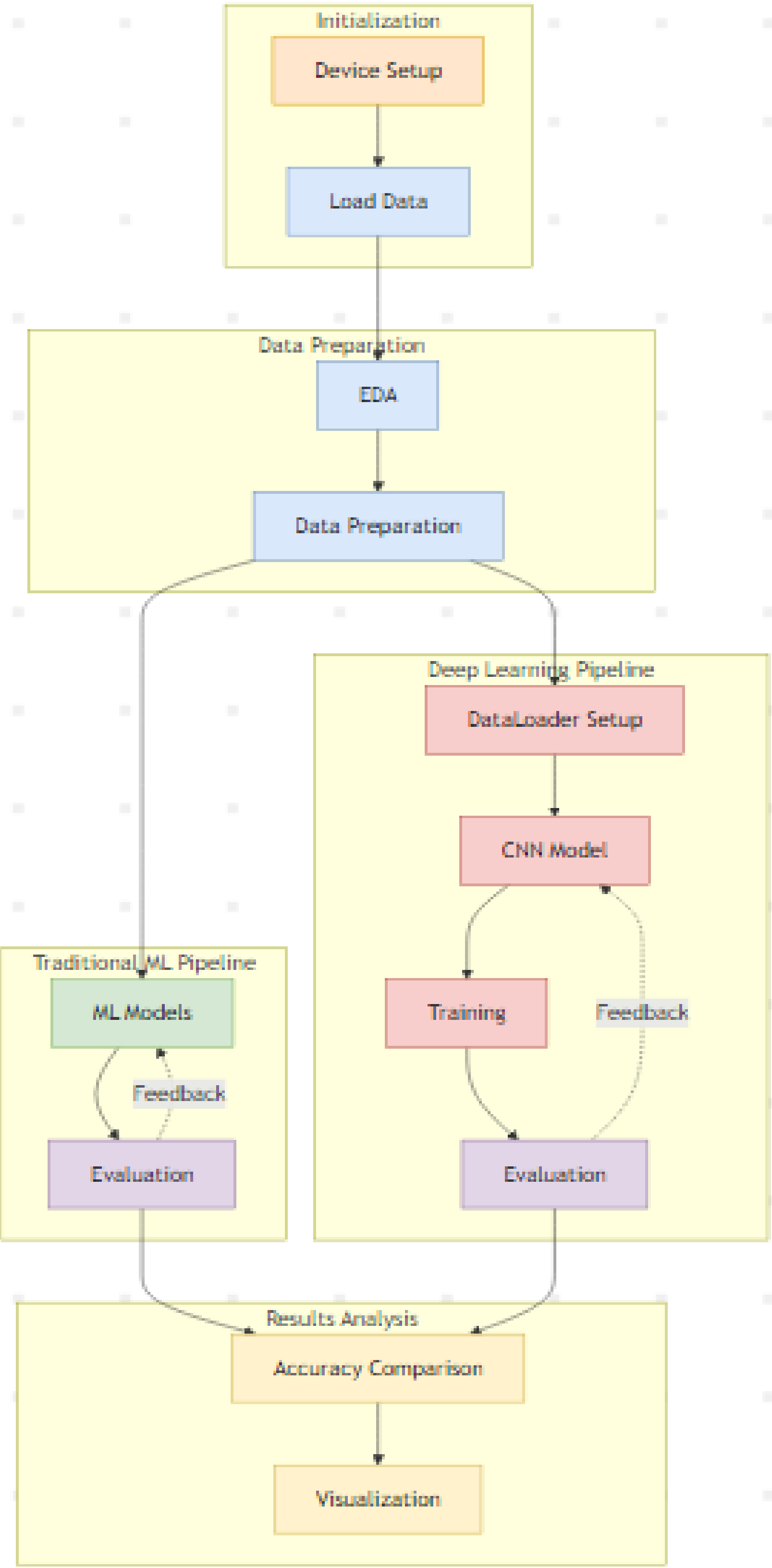
System Pipeline:

1. **Data Loading and Preprocessing**
   MNIST data is loaded, normalized, and reshaped for model compatibility.
2. **Exploratory Data Analysis (EDA)**
   Distribution and pixel intensity visualizations.
3. **Model Building**
   - Classical ML: k-NN, SVM, Decision Tree, Random Forest (with hyperparameter tuning)
   - Deep Learning: CNN with multiple convolutional and pooling layers
4. **Training and Evaluation**
   Models are trained on the training set and evaluated using the test set.
5. **Comparison and Reporting**
   Accuracy, precision, recall, and F1-score are compared across models.
6.

Tools and Technologies Used

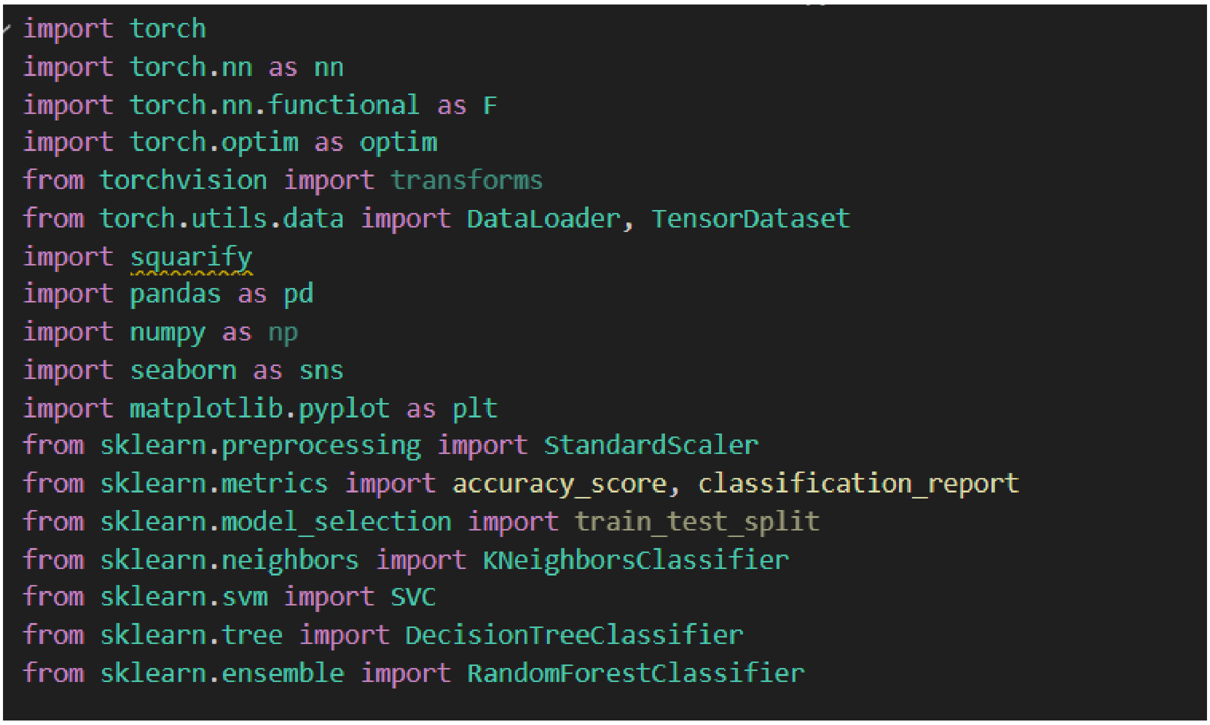
- **Python**: Main programming language
- **Jupyter Notebook / VSCode**: Development environment
- **PyTorch**: Deep learning framework for CNN
- **Scikit-learn**: Classical ML algorithms and evaluation tools
- **Matplotlib, Seaborn**: Visualization libraries
- **NumPy, Pandas**: Data handling and manipulation

Flowchart of process



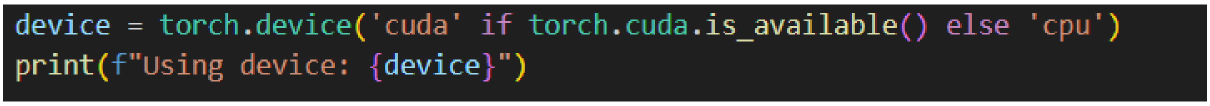
Key Code Snippets

1.importing requirements

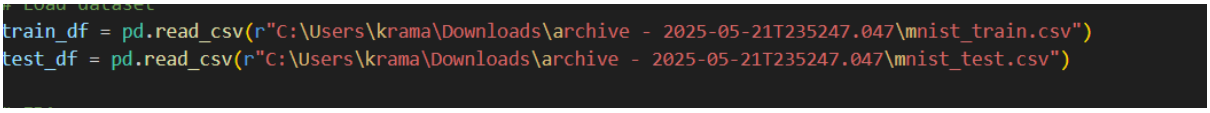


```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import transforms
from torch.utils.data import DataLoader, TensorDataset
import squarify
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

Device setup and Loading data

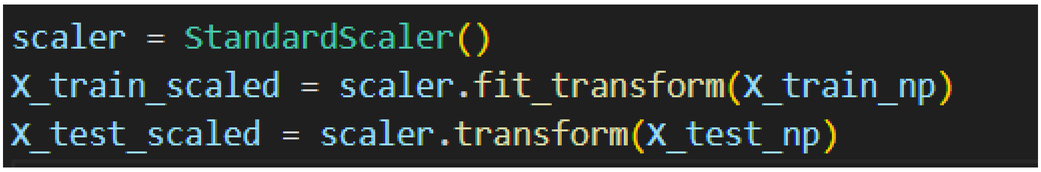


```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
```



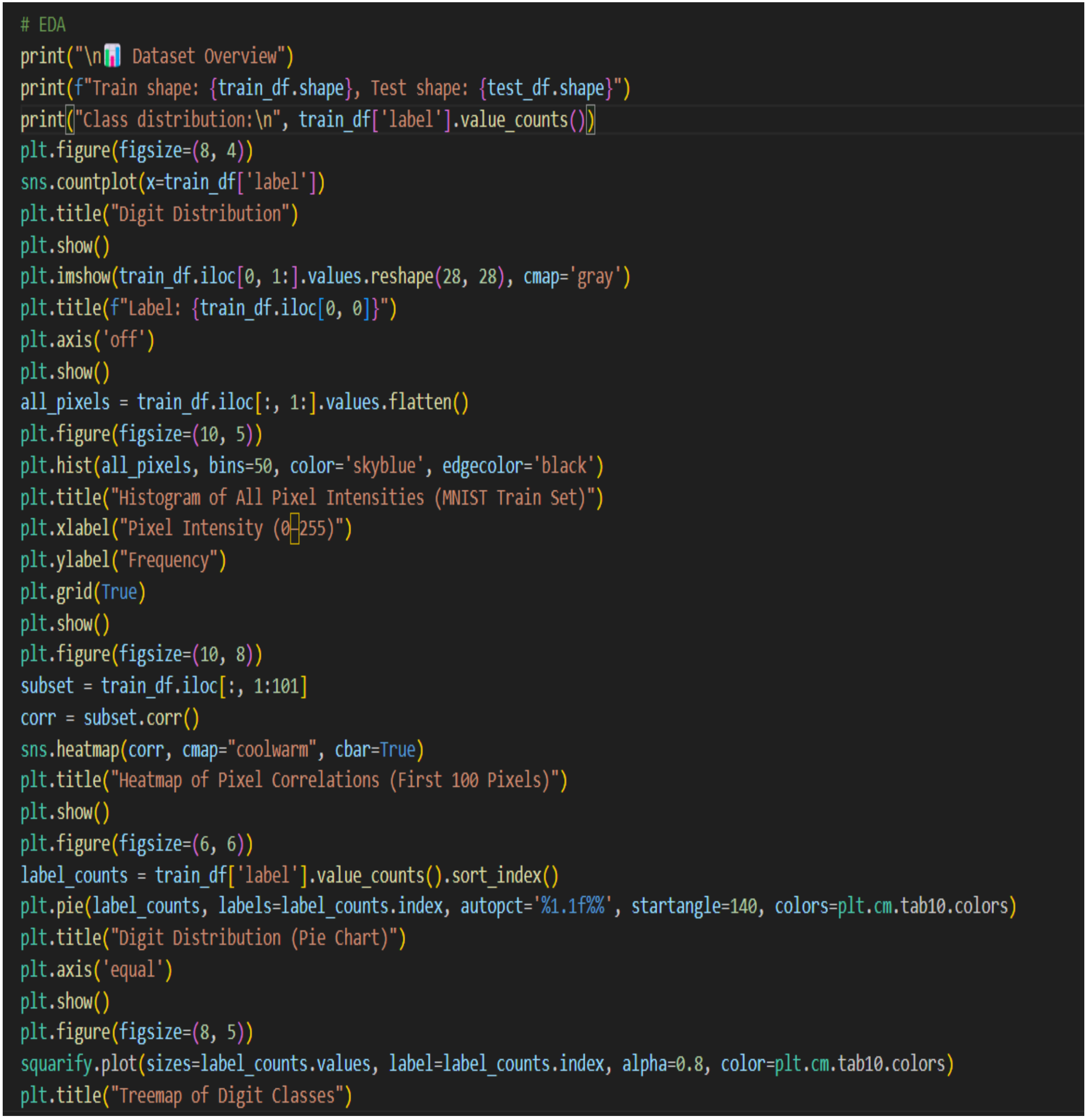
```
# Load dataset
train_df = pd.read_csv(r"C:\Users\krama\Downloads\archive - 2025-05-21T235247.047\mnist_train.csv")
test_df = pd.read_csv(r"C:\Users\krama\Downloads\archive - 2025-05-21T235247.047\mnist_test.csv")
```

Data Normalization and Reshaping



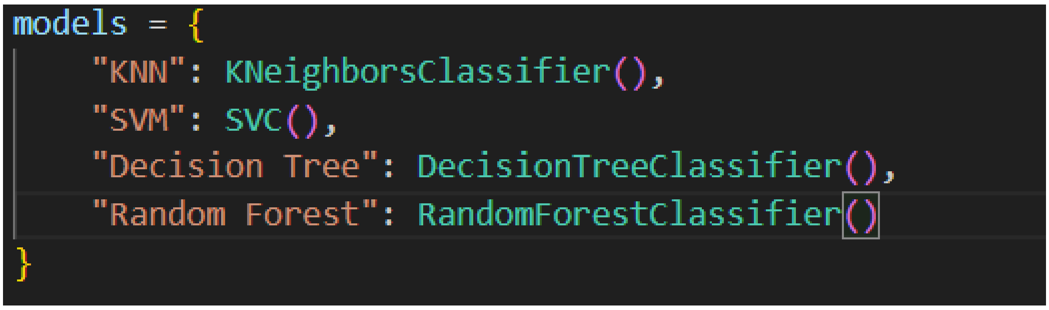
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_np)
X_test_scaled = scaler.transform(X_test_np)
```

EDA



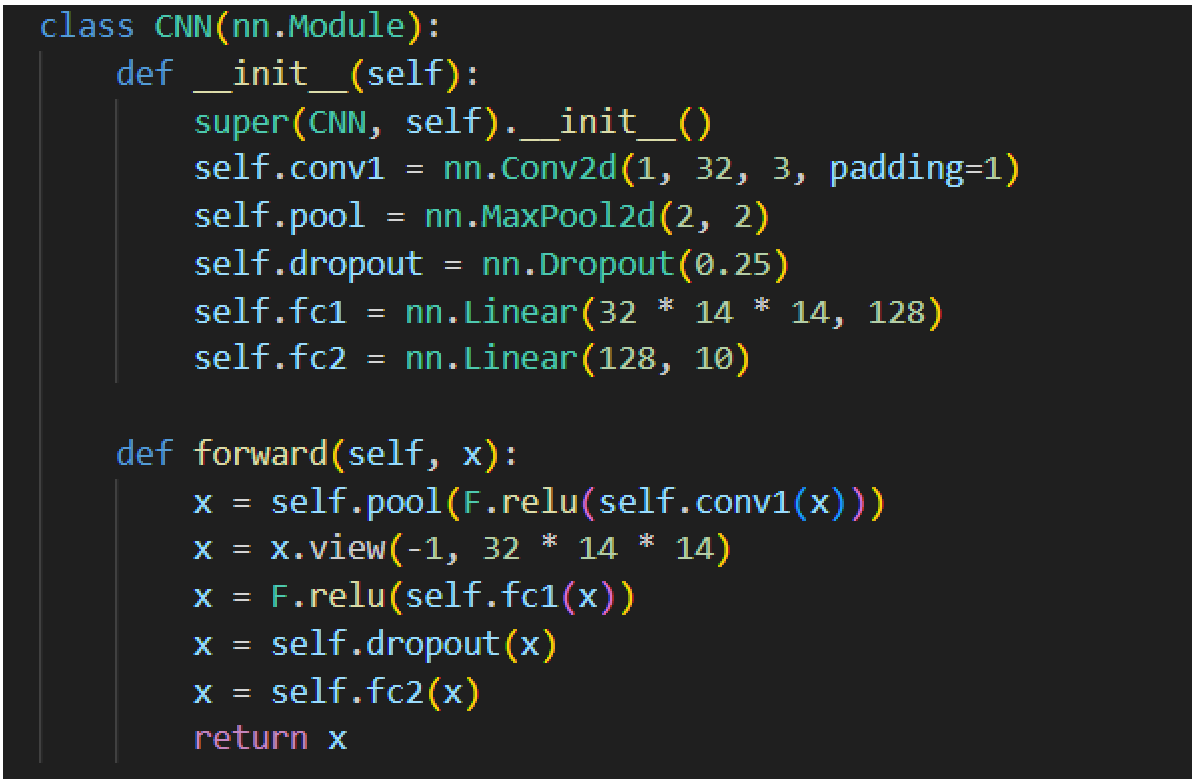
```
# EDA
print("\n📊 Dataset Overview")
print(f"Train shape: {train_df.shape}, Test shape: {test_df.shape}")
print("Class distribution:\n", train_df['label'].value_counts())
plt.figure(figsize=(8, 4))
sns.countplot(x=train_df['label'])
plt.title("Digit Distribution")
plt.show()
plt.imshow(train_df.iloc[0, 1:].values.reshape(28, 28), cmap='gray')
plt.title(f"Label: {train_df.iloc[0, 0]}")
plt.axis('off')
plt.show()
all_pixels = train_df.iloc[:, 1:].values.flatten()
plt.figure(figsize=(10, 5))
plt.hist(all_pixels, bins=50, color='skyblue', edgecolor='black')
plt.title("Histogram of All Pixel Intensities (MNIST Train Set)")
plt.xlabel("Pixel Intensity (0–255)")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()
plt.figure(figsize=(10, 8))
subset = train_df.iloc[:, 1:101]
corr = subset.corr()
sns.heatmap(corr, cmap="coolwarm", cbar=True)
plt.title("Heatmap of Pixel Correlations (First 100 Pixels)")
plt.show()
plt.figure(figsize=(6, 6))
label_counts = train_df['label'].value_counts().sort_index()
plt.pie(label_counts, labels=label_counts.index, autopct='%1.1f%%', startangle=140, colors=plt.cm.tab10.colors)
plt.title("Digit Distribution (Pie Chart)")
plt.axis('equal')
plt.show()
plt.figure(figsize=(8, 5))
squarify.plot(sizes=label_counts.values, label=label_counts.index, alpha=0.8, color=plt.cm.tab10.colors)
plt.title("Treemap of Digit Classes")
```

CLASSIC MODELS



```
models = {
    "KNN": KNeighborsClassifier(),
    "SVM": SVC(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier()
}
```

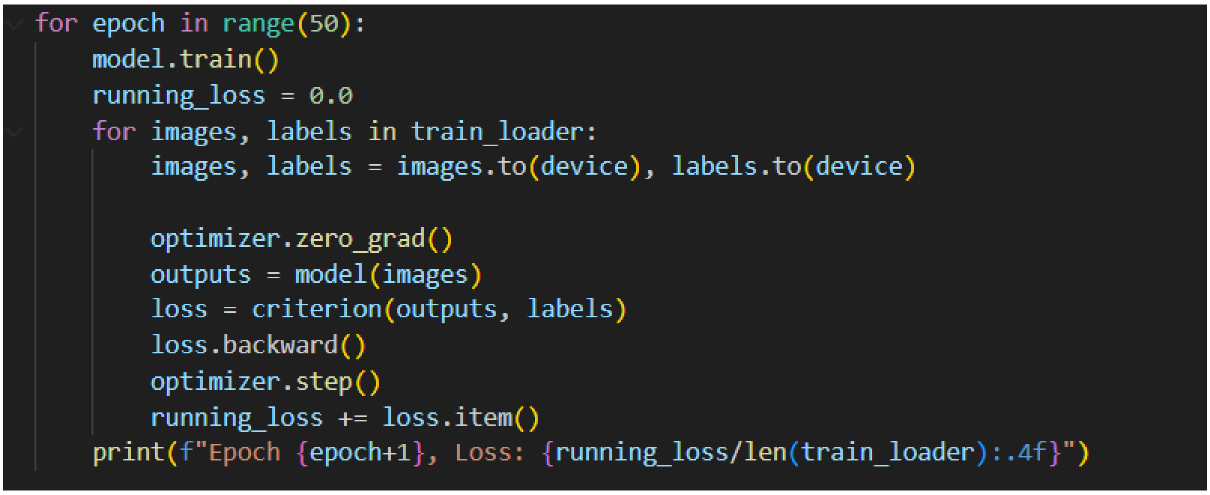
CNN Model Definition in PyTorch



```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.25)
        self.fc1 = nn.Linear(32 * 14 * 14, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = x.view(-1, 32 * 14 * 14)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

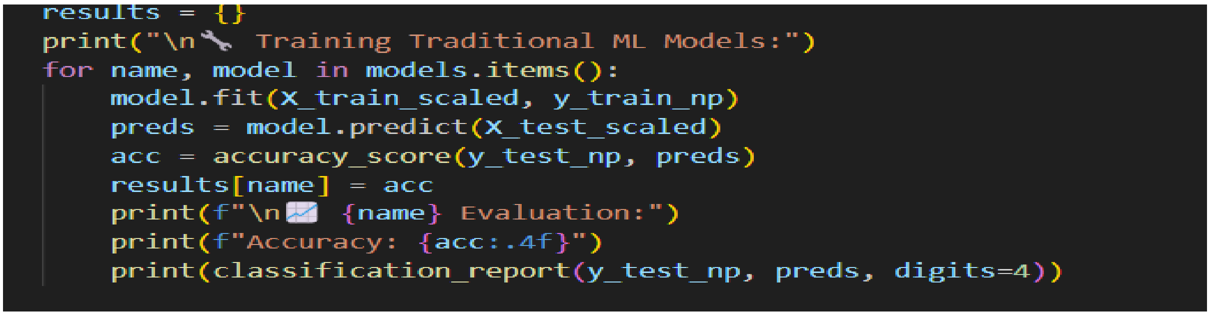
Model Training Loop



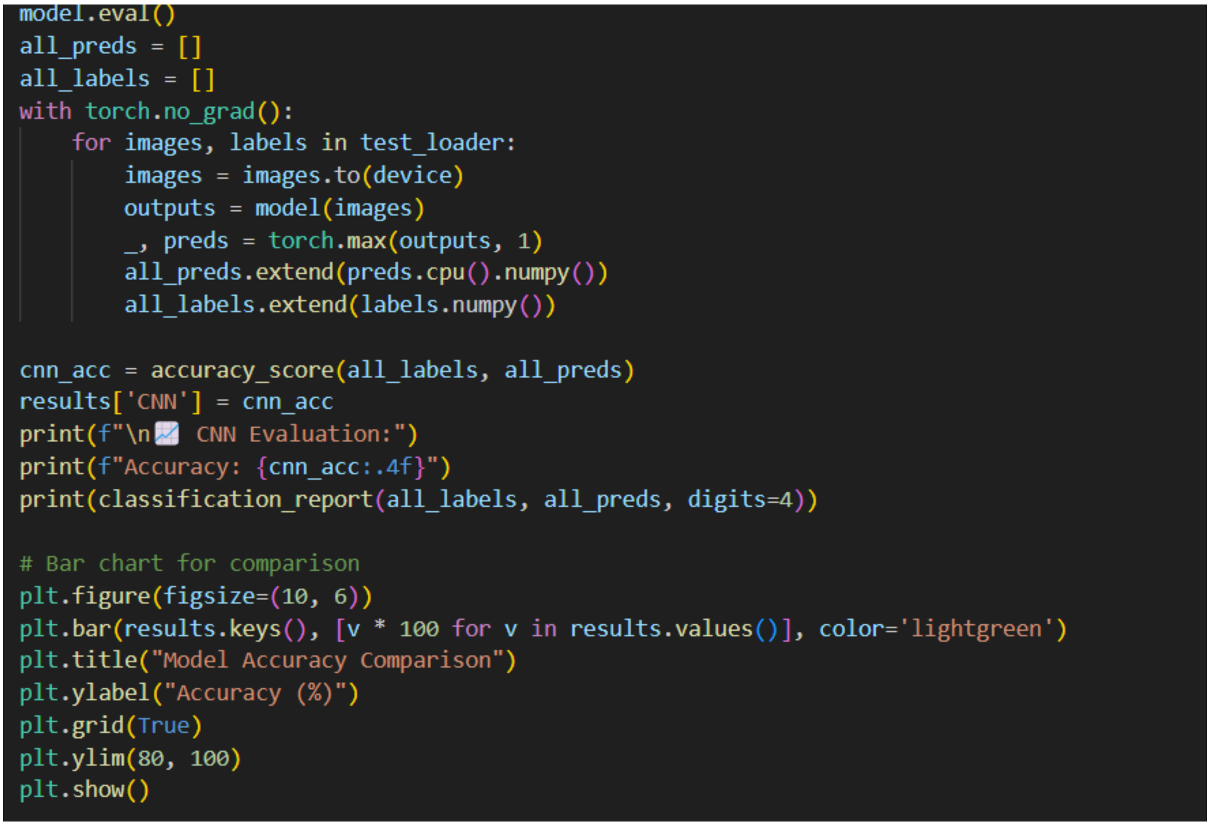
```
for epoch in range(50):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f"Epoch {epoch+1}, Loss: {running_loss/len(train_loader):.4f}")
```

EVALUATION



```
results = {}
print("\n🔧 Training Traditional ML Models:")
for name, model in models.items():
    model.fit(X_train_scaled, y_train_np)
    preds = model.predict(X_test_scaled)
    acc = accuracy_score(y_test_np, preds)
    results[name] = acc
    print(f"\n📈 {name} Evaluation:")
    print(f"Accuracy: {acc:.4f}")
    print(classification_report(y_test_np, preds, digits=4))
```



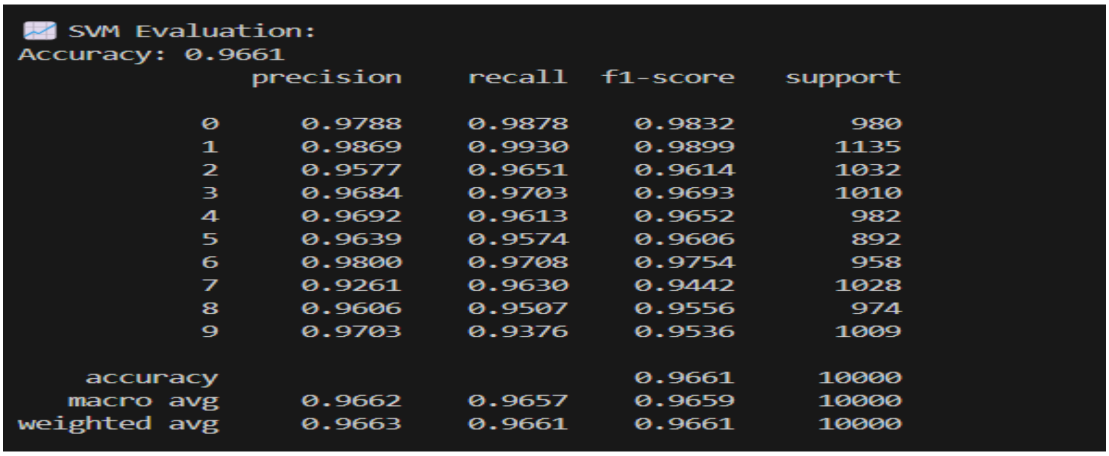
```
model.eval()
all_preds = []
all_labels = []
with torch.no_grad():
    for images, labels in test_loader:
        images = images.to(device)
        outputs = model(images)
        _, preds = torch.max(outputs, 1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.numpy())

cnn_acc = accuracy_score(all_labels, all_preds)
results['CNN'] = cnn_acc
print(f"\n📈 CNN Evaluation:")
print(f"Accuracy: {cnn_acc:.4f}")
print(classification_report(all_labels, all_preds, digits=4))

# Bar chart for comparison
plt.figure(figsize=(10, 6))
plt.bar(results.keys(), [v * 100 for v in results.values()], color='lightgreen')
plt.title("Model Accuracy Comparison")
plt.ylabel("Accuracy (%)")
plt.grid(True)
plt.ylim(80, 100)
plt.show()
```

Observations:

In this experiment, a range of machine learning models were evaluated on a handwritten digit classification task, likely based on the MNIST dataset. Among the traditional models, the **Support Vector Machine (SVM)** performed the best with an accuracy of **96.61%**, demonstrating high precision and recall across all digit classes. This confirms SVM's robustness in handling high-dimensional data and separating classes with clear margins

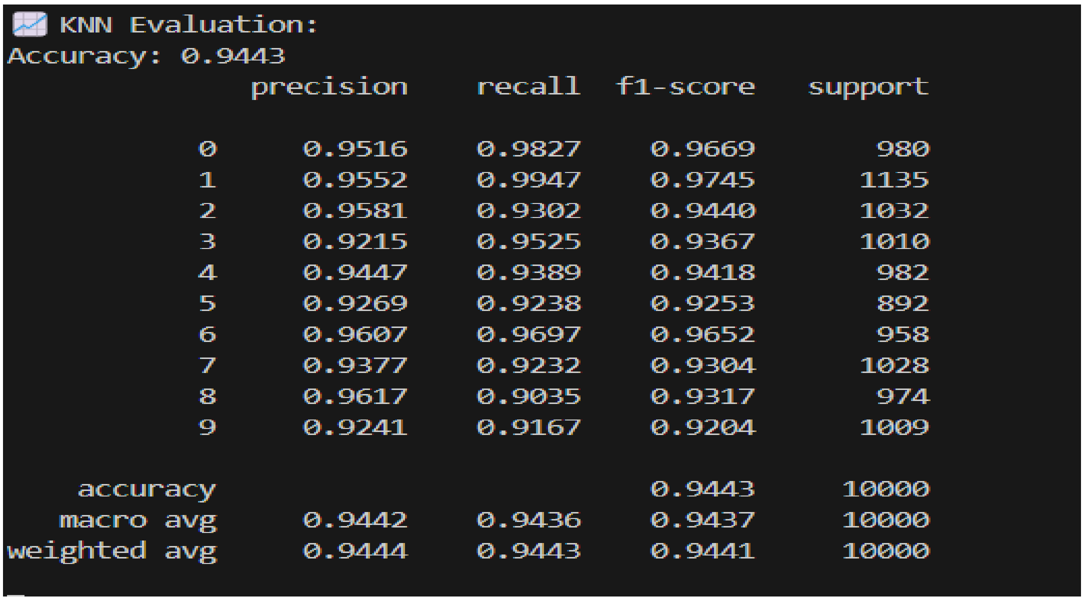


```
📈 SVM Evaluation:
Accuracy: 0.9661
              precision    recall  f1-score   support

           0     0.9788    0.9878    0.9832       980
           1     0.9869    0.9930    0.9899      1135
           2     0.9577    0.9651    0.9614      1032
           3     0.9684    0.9703    0.9693      1010
           4     0.9692    0.9613    0.9652       982
           5     0.9639    0.9574    0.9606       892
           6     0.9800    0.9708    0.9754       958
           7     0.9261    0.9630    0.9442      1028
           8     0.9606    0.9507    0.9556       974
           9     0.9703    0.9376    0.9536      1009

    accuracy                         0.9661     10000
   macro avg     0.9662    0.9657    0.9659     10000
weighted avg     0.9663    0.9661    0.9661     10000
```

The **K-Nearest Neighbors (KNN)** algorithm achieved an accuracy of **94.43%**, performing well but slightly under traditional model leaders. KNN displayed solid precision for most digits but had slight dips in recall for digits like 8 and 5, indicating confusion in distinguishing similar patterns

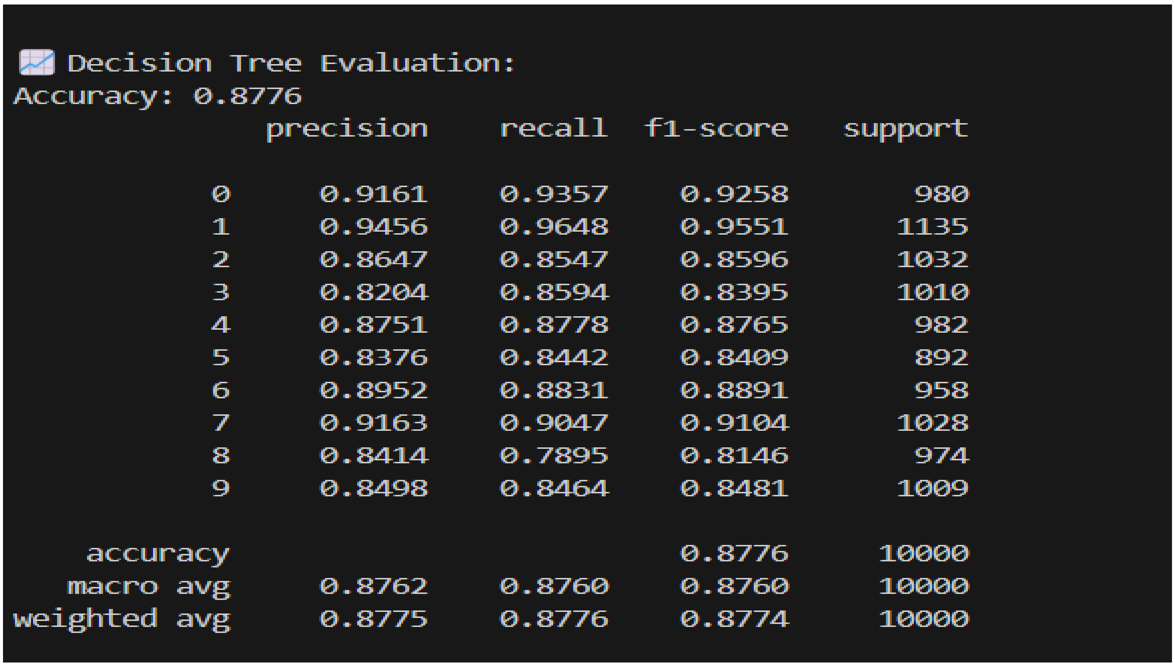


```
📈 KNN Evaluation:
Accuracy: 0.9443
              precision    recall  f1-score   support

           0     0.9516    0.9827    0.9669       980
           1     0.9552    0.9947    0.9745      1135
           2     0.9581    0.9302    0.9440      1032
           3     0.9215    0.9525    0.9367      1010
           4     0.9447    0.9389    0.9418       982
           5     0.9269    0.9238    0.9253       892
           6     0.9607    0.9697    0.9652       958
           7     0.9377    0.9232    0.9304      1028
           8     0.9617    0.9035    0.9317       974
           9     0.9241    0.9167    0.9204      1009

    accuracy                         0.9443     10000
   macro avg     0.9442    0.9436    0.9437     10000
weighted avg     0.9444    0.9443    0.9441     10000
```

Decision Trees, while simple and interpretable, achieved the lowest performance among traditional methods, with an accuracy of **87.76%**. This lower accuracy can be attributed to overfitting and its inability to generalize complex patterns in image data.

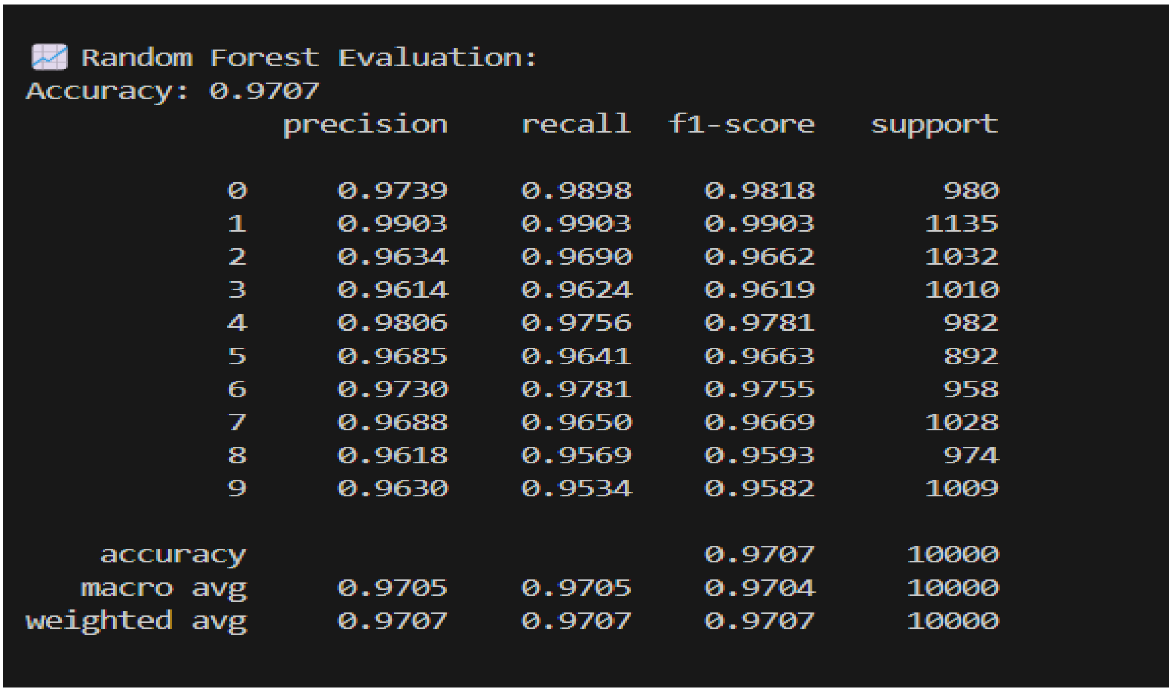


```
📈 Decision Tree Evaluation:
Accuracy: 0.8776
              precision    recall  f1-score   support

           0     0.9161    0.9357    0.9258       980
           1     0.9456    0.9648    0.9551      1135
           2     0.8647    0.8547    0.8596      1032
           3     0.8204    0.8594    0.8395      1010
           4     0.8751    0.8778    0.8765       982
           5     0.8376    0.8442    0.8409       892
           6     0.8952    0.8831    0.8891       958
           7     0.9163    0.9047    0.9104      1028
           8     0.8414    0.7895    0.8146       974
           9     0.8498    0.8464    0.8481      1009

    accuracy                         0.8776     10000
   macro avg     0.8762    0.8760    0.8760     10000
weighted avg     0.8775    0.8776    0.8774     10000
```

The **Random Forest** model followed closely with an impressive **97.07%** accuracy, benefiting from ensemble learning to correct individual decision tree weaknesses, and showed balanced precision and recall, particularly excelling with digits like 1, 4, and 6.

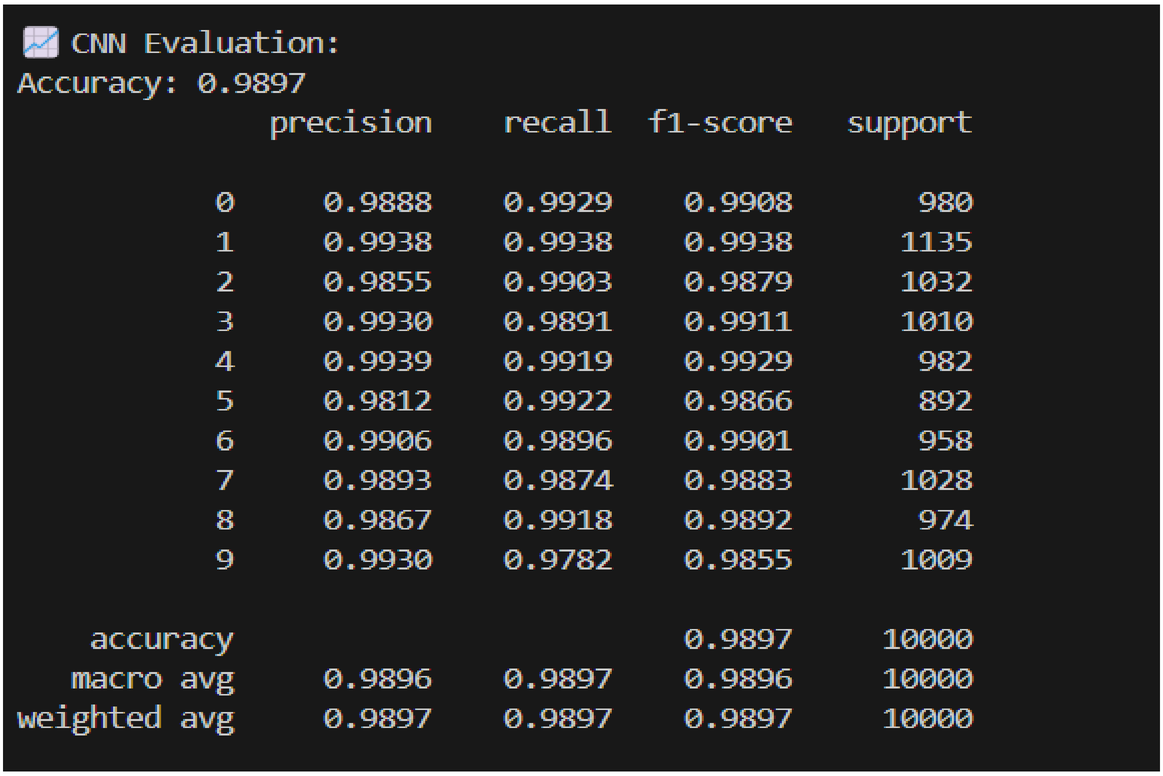


```
📈 Random Forest Evaluation:
Accuracy: 0.9707
              precision    recall  f1-score   support

           0     0.9739    0.9898    0.9818       980
           1     0.9903    0.9903    0.9903      1135
           2     0.9634    0.9690    0.9662      1032
           3     0.9614    0.9624    0.9619      1010
           4     0.9806    0.9756    0.9781       982
           5     0.9685    0.9641    0.9663       892
           6     0.9730    0.9781    0.9755       958
           7     0.9688    0.9650    0.9669      1028
           8     0.9618    0.9569    0.9593       974
           9     0.9630    0.9534    0.9582      1009

    accuracy                         0.9707     10000
   macro avg     0.9705    0.9705    0.9704     10000
weighted avg     0.9707    0.9707    0.9707     10000
```

On the other hand, the **Convolutional Neural Network (CNN)** significantly outperformed all traditional models with a final test accuracy of **98.97%** after 50 epochs. The training loss consistently decreased over epochs, stabilizing at a very low value, indicating a well-trained network. Precision and recall values were near-perfect across all digit classes, confirming the CNN's ability to learn hierarchical image features and capture spatial relationships. This highlights the superiority of deep learning approaches in image classification tasks, particularly when ample data and training time are available.

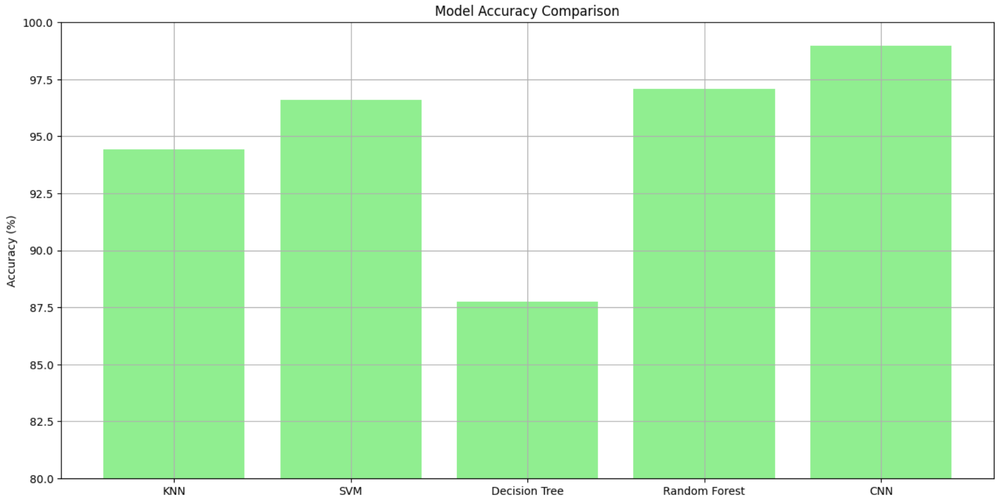


```
📈 CNN Evaluation:
Accuracy: 0.9897
              precision    recall  f1-score   support

           0     0.9888    0.9929    0.9908       980
           1     0.9938    0.9938    0.9938      1135
           2     0.9855    0.9903    0.9879      1032
           3     0.9930    0.9891    0.9911      1010
           4     0.9939    0.9919    0.9929       982
           5     0.9812    0.9922    0.9866       892
           6     0.9906    0.9896    0.9901       958
           7     0.9893    0.9874    0.9883      1028
           8     0.9867    0.9918    0.9892       974
           9     0.9930    0.9782    0.9855      1009

    accuracy                         0.9897     10000
   macro avg     0.9896    0.9897    0.9896     10000
weighted avg     0.9897    0.9897    0.9897     10000
```

The given image have comparision of accuracy of models together as we can see CNN performs best among all those



In conclusion, while traditional machine learning models offer strong performance and fast interpretability, the CNN's deep learning approach yielded the highest accuracy and reliability, making it the optimal choice for this digit classification task.

Drawbacks of the Convolutional Neural Network

Despite its high accuracy and strong generalisation performance, the Convolutional Neural Network (CNN) has certain drawbacks that must be acknowledged. Firstly, CNNs are computationally intensive and require significantly more processing power and memory compared to traditional machine learning models. This can make them less suitable for deployment in resource-constrained environments or applications requiring real-time predictions on edge devices. Secondly, CNNs require large volumes of labelled data for effective training, which may not always be readily available in practical scenarios. Additionally, the training process involves a large number of parameters and hyperparameters, which can lead to longer training times and increased risk of overfitting if not properly managed. The interpretability of CNNs is also limited, as the learned features are not easily understandable by humans, making it difficult to explain decision-making processes. Lastly, model tuning and architecture design often require deep expertise and iterative experimentation, which can be a barrier for practitioners with limited experience in deep learning.

Another notable limitation of Convolutional Neural Networks is their sensitivity to variations in input data. While CNNs can perform well on data distributions similar to their training sets, even minor changes such as lighting conditions, orientation, or background noise can lead to a decline in performance if the model has not been sufficiently regularised or augmented. This highlights the importance of robust data preprocessing and augmentation strategies.

Moreover, CNNs are largely domain-specific and may not generalise well across vastly different tasks without significant retraining or transfer learning. For instance, a CNN trained on handwritten digits may not perform effectively on natural images without substantial modifications and retraining on a new dataset.

Another drawback is the **black-box nature** of deep learning models like CNNs. Unlike traditional machine learning algorithms where decision boundaries can often be interpreted, the internal representations and layer activations of CNNs are abstract and difficult to analyse, which can pose challenges in sensitive applications where model transparency and accountability are critical.

Additionally, CNNs are prone to **adversarial attacks**—small, imperceptible perturbations in input data can drastically alter the output predictions. This vulnerability can be exploited maliciously in security-critical applications, making robustness and model verification areas of growing concern in CNN deployment.

Finally, the development and fine-tuning of CNNs typically demand a high level of technical expertise and access to advanced computational resources. This may limit their accessibility for smaller teams or organisations without such infrastructure.

Conclusion

Following a comprehensive evaluation of multiple machine learning algorithms and a deep learning approach, it is evident that the **Convolutional Neural Network (CNN)** outperforms all traditional models in terms of accuracy and generalisation. The CNN achieved an accuracy of **98.97 percent** on the test dataset, significantly surpassing traditional methods such as K-Nearest Neighbors (94.43 percent), Support Vector Machine (96.61 percent), Decision Tree (87.76 percent), and even the highly competitive Random Forest model (97.07 percent).

In addition to superior accuracy, the CNN demonstrated excellent precision, recall, and f1-scores across all digit classes. This indicates strong generalisation capabilities, meaning the model performs well not only on the training data but also on unseen samples. The deep learning model also showed continuous improvement in training loss over multiple epochs, stabilising at a very low value, which suggests effective learning and convergence.

While traditional models like SVM and Random Forest showed competitive performance and are generally easier to train and interpret, they lack the same level of feature extraction capabilities as the CNN. The CNN automatically learns hierarchical spatial features directly from the pixel data, making it far more suitable for complex visual tasks such as handwritten digit recognition.

Therefore, **the Convolutional Neural Network is the most appropriate model to retain for deployment or further development**. It offers the best trade-off between accuracy and generalisation and is well-aligned with the requirements of image-based classification tasks. It is also highly adaptable to enhancements such as data augmentation and model regularisation, which can further boost its robustness in real-world applications.

In conclusion, based on empirical results and practical considerations, the CNN model is selected as the final model for this project.

References

- Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273-297.
- Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern Classification*. Wiley-Interscience.
- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1), 1929-1958.
- Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of the 32nd International Conference on Machine Learning*, 448-456.
- Simard, P. Y., Steinkraus, D., & Platt, J. C. (2003). Best practices for convolutional neural networks applied to visual document analysis. *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, 958-963.