**Handwritten Digit Recognition with LeNet5**

**By:**

Name:Navdeep. S

**TABLE OF CONTENT**

**INTRODUCTION\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3**

**MNIST DATASET \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_4**

**PREPROCESSING \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_6**

**VISUALIZATION \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7**

**MLP \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_8**

**CNN\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_11**

**LENET-5\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 15**

**HYPERPARAMETER TUNING \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_19**

**COMPARISON OF MODELS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_22**

**CONCLUSION\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_23**

**Handwritten Digit Recognition with LeNet-5 Model in PyTorch**

**Introduction**

Handwritten digit recognition is a classic problem in the field of machine learning and computer vision. The project uses the MNIST dataset, which contains grayscale images of handwritten digits (0-9). The primary objective is to classify these digits using machine learning models like Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN), and LeNet-5. The project is implemented in PyTorch, a popular deep-learning framework.

**Objectives**

1. To preprocess the MNIST dataset and prepare it for training machine learning models by normalizing pixel values and converting images into a format suitable for neural networks.

2. To implement and train multiple models for digit recognition, including:

* A baseline Multilayer Perceptron (MLP) model.
* A simple Convolutional Neural Network (CNN) model. The LeNet-5 model, a pioneering CNN architecture designed for handwritten character recognition.

3. To evaluate and compare the performance of these models using metrics such as accuracy, F1-score, and confusion matrices.

**Significance**

Handwritten digit recognition serves as a foundational problem in the field of machine learning, providing insights into image processing, feature extraction, and model optimization. The outcomes of this project have implications for real-world applications requiring reliable and efficient handwritten digit recognition systems.

By addressing these challenges and systematically evaluating the three models, this project seeks to contribute to the development of robust solutions for handwritten digit recognition, ensuring both high accuracy and computational efficiency.

**MNIST Dataset:**

The MNIST (Modified National Institute of Standards and Technology) dataset is one of the most widely used benchmarks in the field of machine learning and computer vision. Introduced by Yann LeCun and colleagues in 1998, it provides a standardized dataset for evaluating algorithms in handwritten digit recognition. Its simplicity, accessibility, and utility have made it a cornerstone in deep learning research and education.

**Description of the Dataset**

The MNIST dataset consists of 70,000 grayscale images of handwritten digits (0-9), each labeled with its corresponding digit. The dataset is divided into:

* **Training Set:** 60,000 images used to train models.
* **Test Set:** 10,000 images used to evaluate model performance.

Each image is:

* **Size:** 28 × 28 pixels, totaling 784 pixels per image.
* **Grayscale Intensity:** Pixel values range from 0 (black) to 255 (white).
* **Format:** Images are normalized and centered within a fixed-size frame to ensure consistency.

**Key Features**

1. **Preprocessed and Cleaned Data:** The dataset is preprocessed to remove noise and align digits, simplifying model training and evaluation.
2. **Diverse Handwriting Styles:** Collected from over 250 writers, the dataset represents significant variability, enabling robust model testing.
3. **Ease of Access:** Available in standard machine learning libraries like TensorFlow, PyTorch, and scikit-learn, MNIST is easy to integrate into projects.

**Why MNIST is Important**

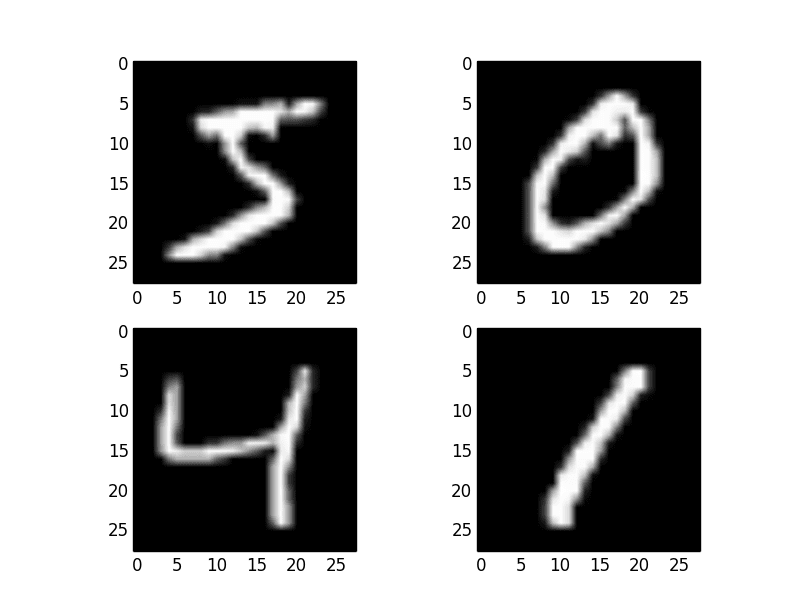
1. **Benchmark for Model Comparison:** MNIST provides a common ground for comparing algorithms, making it a valuable tool for research and development.
2. **Educational Value:** Its simplicity makes it an ideal dataset for beginners to understand image processing and machine learning fundamentals.
3. **Historical Significance:** MNIST paved the way for the development of advanced neural network architectures, including Convolutional Neural Networks (CNNs) and deep learning frameworks.

**Limitations**

While MNIST is a powerful learning tool, it has limitations:

* **Overuse:** It may no longer represent real-world complexity, as many modern applications involve more diverse and challenging datasets.
* **Simplistic Task:** The dataset’s limited variability and small size may not adequately test cutting-edge models designed for complex tasks.

**Sample Images**



The MNIST dataset remains a foundational dataset for developing and testing models in handwritten digit recognition. Its simplicity and utility have helped drive significant advancements in machine learning and deep learning, making it an indispensable tool for students, researchers, and practitioners alike.

**Data Preprocessing and Visualization**

Data preprocessing and visualization are critical steps in any machine learning project, ensuring that the dataset is clean, structured, and ready for model training. For the MNIST dataset, these steps help enhance the accuracy and reliability of machine learning models by addressing noise, scaling, and variability in the input data.

**Data Preprocessing**

The preprocessing phase involves preparing raw data for analysis and model training. For MNIST, key preprocessing steps include:

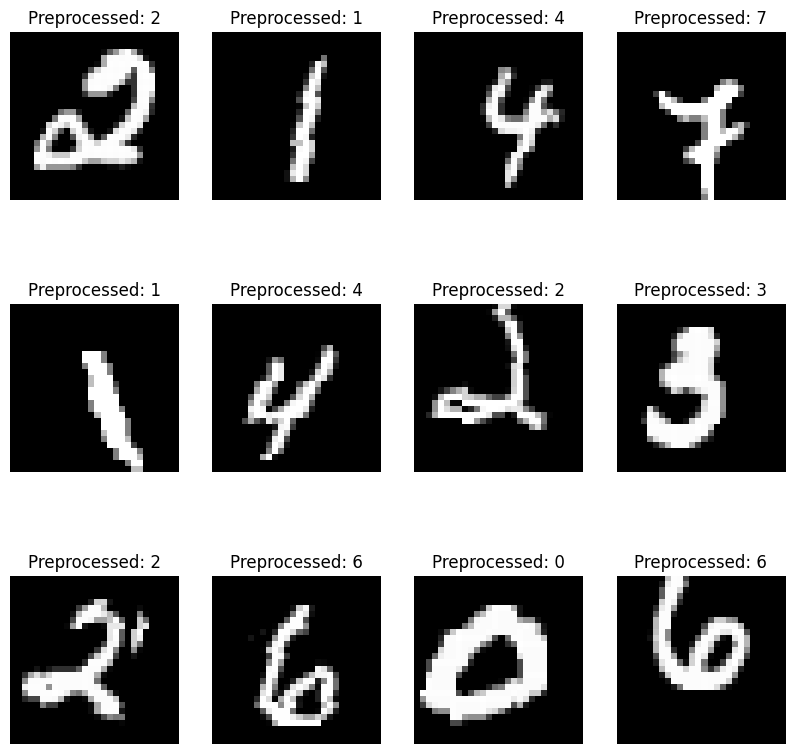
1. **Normalization**:  
   Each pixel's intensity value in the MNIST images ranges from 0 to 255. To make the dataset suitable for training, these values are normalized to a range of 0 to 1 by dividing by 255. This ensures faster convergence during training and reduces the risk of numerical instability.
2. **Flattening :**  
   The 28×28 images can be flattened into 1D arrays of 784 features for models like fully connected neural networks. However, CNNs work better with 2D image structures and thus retain the original format.
3. **Splitting the Data**:  
   The dataset is divided into training (60,000 images) and testing sets (10,000 images). Cross-validation can also be applied to evaluate model robustness.
4. **Data Augmentation**: While MNIST is a relatively simple dataset, augmenting the data can improve model generalization. Data augmentation involves transforming the images (e.g., rotations, shifts, or adding noise). For MNIST, common augmentation techniques include:
   * + **Random rotation**: Slightly rotate the images.
     + **Random translation**: Shift the images by a few pixels.

**Visualization**

Visualizing data helps understand patterns, detect anomalies, and validate preprocessing. Common visualization techniques for the MNIST dataset include:

1. **Sample Images**:  
   Displaying a grid of sample images gives an overview of the dataset. For instance, plotting random images with their corresponding labels confirms that data and labels are aligned correctly.
2. **Distribution of Labels**:  
   A bar plot showing the frequency of each digit helps verify class balance. Ideally, MNIST has roughly equal representation of each digit.

**Example visualization**



Data preprocessing and visualization are indispensable for leveraging the full potential of the MNIST dataset. Preprocessing ensures data quality and compatibility with machine learning models, while visualization provides critical insights into the dataset's structure and characteristics, enabling informed decision-making in subsequent steps.

**Multilayer Perceptron (MLP)**

A Multilayer Perceptron (MLP) is a class of feedforward artificial neural networks (ANNs). It consists of an input layer, one or more hidden layers, and an output layer, with fully connected nodes between adjacent layers. MLPs are designed to learn complex patterns and relationships in data by leveraging non-linear activation functions.

**MLP Architecture**

**Input Layer**:

* Accepts input features, such as pixel values in image datasets like MNIST.
* For MNIST, the input size is typically 784 (28×28 pixels flattened into a 1D vector).

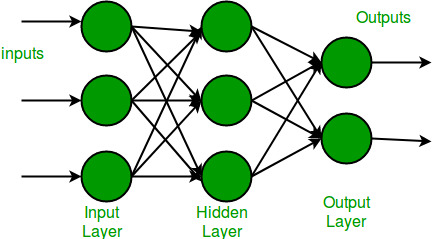
**Hidden Layers**:

* Comprise several neurons that process and transform the input data.
* Each neuron applies a weighted sum of inputs followed by a non-linear activation function.
* Common activation functions include ReLU (Rectified Linear Unit), sigmoid, and tanh.

**Output Layer**:

* Produces the final predictions.
* For MNIST, the output layer typically has 10 neurons (representing digits 0–9), with a softmax activation to output probabilities.

**Architecture Diagram**



**Key Components**

1. **Weights and Biases**:
   * Each connection between neurons has an associated weight, which adjusts during training.
   * Bias terms allow the model to shift activation functions for better learning.
2. **Activation Functions**:
   * Introduce non-linearity, enabling the model to learn more complex patterns.
   * ReLU is popular due to its computational efficiency and effectiveness in deeper networks.
3. **Loss Function**:
   * Measures the error between predicted and actual values.
   * For MNIST, categorical cross-entropy is commonly used.
4. **Optimization Algorithm**:
   * Adjusts weights and biases to minimize the loss function.
   * Popular optimizers include stochastic gradient descent (SGD) and Adam.

**Training Process**

1. **Forward Propagation**:
   * Inputs pass through the network, layer by layer, producing an output prediction.
2. **Loss Calculation**:
   * The difference between the predicted and actual values is calculated using the loss function.
3. **Backward Propagation**:
   * Gradients of the loss function with respect to weights and biases are computed using backpropagation.
   * These gradients are used to update the parameters through the optimizer.
4. **Iterations/Epochs**:
   * Training proceeds over multiple iterations (or epochs) until the model converges to an optimal state.

**Evaluation of an MLP**

After training, the MLP is evaluated on a separate test dataset to measure its generalization performance.

**Steps in Evaluation:**

1. **Forward Pass on Test Data**:
   * Input the test data through the trained model to generate predictions.
2. **Metrics Calculation**:
   * Compare predictions with true labels using evaluation metrics, such as:
     + **Accuracy**: For classification tasks.
     + **Precision, Recall, F1-Score**: For imbalanced classification problems.
     + **Mean Squared Error (MSE), Mean Absolute Error (MAE)**: For regression tasks.
3. **Generalization Check**:
   * Check for overfitting or underfitting by comparing training and test performance.

**Applications of MLP**

* Classification tasks (e.g., handwritten digit recognition, spam detection).
* Regression tasks (e.g., house price prediction).
* Feature extraction in larger neural network architectures.

**Convolutional Neural Network (CNN)**

A Convolutional Neural Network (CNN) is a specialized deep learning architecture designed to process data with grid-like topology, such as images. CNNs are highly effective for image recognition and classification tasks due to their ability to automatically and adaptively learn spatial hierarchies of features from data.

**Key Components of a CNN**

**1. Input Layer**

* The input to a CNN is typically an image represented as a multidimensional array. For example, grayscale images from the MNIST dataset are represented as 28x28 matrices of pixel intensities.

**2. Convolutional Layers**

* **Purpose:** Extract features from the input image using learnable filters (kernels).
* **Mechanism:** A kernel slides over the image, performing an element-wise multiplication and summation to produce feature maps.
* **Advantages:**
  + Reduces the need for manual feature engineering.
  + Preserves spatial relationships between pixels.

**3. Pooling Layers**

* **Purpose:** Down sample feature maps, reducing dimensionality and computational cost.
* **Types:**
  + **Max Pooling:** Takes the maximum value in a patch (e.g., 2x2).
  + **Average Pooling:** Computes the average of values in a patch.
* **Effect:** Retains important features while reducing noise.

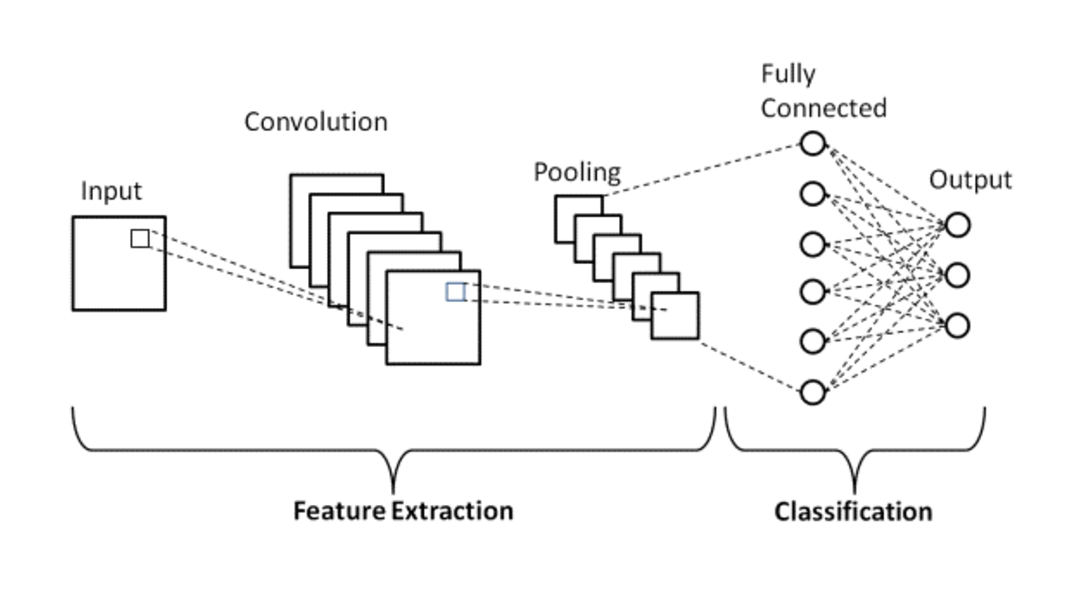
**4. Fully Connected Layers**

* After feature extraction, fully connected layers process the flattened feature maps into high-level representations.
* These layers form the final decision-making part of the network, mapping features to output classes.

**5**. **Output Layer**

* The output layer provides predictions for the given input.
* **Activation:** Softmax activation is commonly used for multi-class classification tasks, as it converts logits into probabilities.

**Architecture Diagram**



**Training a CNN**

The training process for a CNN is similar to other neural networks but leverages the convolutional structure to extract spatial hierarchies.

**Steps in Training:**

1. **Input Data**:
   * Feed raw input (e.g., images) into the network.
2. **Forward Propagation**:
   * Data flows through the convolutional, pooling, and fully connected layers.
   * Predictions are made at the output layer.
3. **Loss Function**:
   * Compute the error between predicted and actual labels.
     + Common loss functions:
       - **Cross-Entropy Loss**: For classification tasks.
       - **Mean Squared Error (MSE)**: For regression tasks.
4. **Backpropagation**:
   * Calculate the gradients of the loss with respect to the model's parameters using the **chain rule**.
5. **Optimization**:
   * Update parameters (weights and biases) using algorithms like **SGD**, **Adam**, or **RMSProp**.
6. **Repeat**:
   * Iterate through multiple epochs until the model converges to a minimum loss.

**Evaluation of a CNN**

After training, the CNN is evaluated on unseen data to measure its performance and generalization capability.

**Steps in Evaluation:**

1. **Forward Pass on Test Data**:
   * Input the test dataset into the trained model to generate predictions.
2. **Metrics Calculation**:
   * Compare predictions with true labels using evaluation metrics:
     + **Accuracy**: Percentage of correctly classified instances.
     + **Precision, Recall, F1-Score**: For imbalanced classification problems.
     + **Confusion Matrix**: For detailed classification performance.
     + **Mean Squared Error (MSE)**: For regression tasks.
3. **Generalization Assessment**:
   * Check for overfitting or underfitting by comparing training and test metrics.

**Advantages of CNNs:**

* Automatically extracts features from raw data.
* Reduces the need for manual feature engineering.
* Handles spatial relationships in data (e.g., image structure).

**Applications of CNNs:**

* **Image Classification**: (e.g., detecting cats vs. dogs).
* **Object Detection**: (e.g., identifying cars in a video).
* **Segmentation**: (e.g., medical image analysis).
* **Natural Language Processing**: (e.g., text classification).

**LeNet-5**

LeNet-5, developed by Yann LeCun and his colleagues in 1998, is a pioneering convolutional neural network (CNN) architecture. It was specifically designed for handwritten character recognition and was instrumental in the development of modern deep learning. LeNet-5 is particularly well-suited for datasets like MNIST due to its ability to capture spatial hierarchies in image data.

**Architecture Overview**

LeNet-5 is composed of **7 layers**, including learnable layers and non-learnable operations such as pooling. These layers are designed to process and reduce the dimensionality of input images while extracting relevant features. The architecture can be summarized as follows:

**a. Input Layer**

* **Input size:** 32×3232 \times 3232×32 grayscale images. (The MNIST dataset’s 28×2828 \times 2828×28 images are typically zero-padded to fit this size.)
* **Preprocessing:** Normalization to improve learning efficiency.

**b. Convolutional Layers**

1. **Layer 1: Convolution (C1)**
   * **Filters:** 6 filters of size 5×55 \times 55×5.
   * **Output size:** 28×28×628 \times 28 \times 628×28×6.
   * **Purpose:** Extract low-level features such as edges, lines, and blobs.
2. **Layer 2: Subsampling (S2)**
   * **Type:** Average pooling with a stride of 2.
   * **Output size:** 14×14×614 \times 14 \times 614×14×6.
   * **Purpose:** Reduce spatial dimensions while retaining critical features.
3. **Layer 3: Convolution (C3)**
   * **Filters:** 16 filters of size 5×55 \times 55×5.
   * **Output size:** 10×10×1610 \times 10 \times 1610×10×16.
   * **Purpose:** Extract more complex features by increasing filter depth.
4. **Layer 4: Subsampling (S4)**
   * **Type:** Average pooling with a stride of 2.
   * **Output size:** 5×5×165 \times 5 \times 165×5×16.
   * **Purpose:** Further dimensionality reduction and feature abstraction.

**c. Fully Connected Layers**

1. **Layer 5: Fully Connected (F5)**
   * **Nodes:** 120 neurons.
   * **Purpose:** Learn high-level representations from the extracted features.
2. **Layer 6: Fully Connected (F6)**
   * **Nodes:** 84 neurons.
   * **Purpose:** Map the high-level features to specific digit classes.

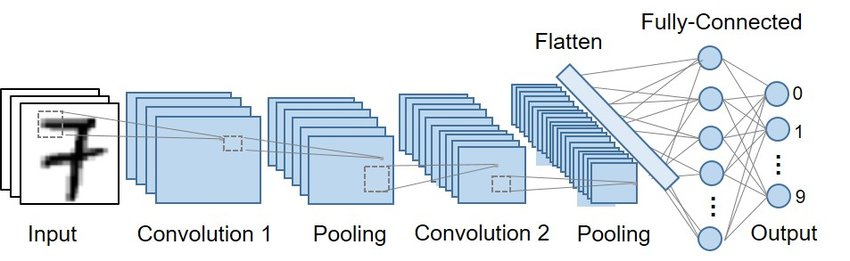
**d. Output Layer**

1. **Layer 7: Output**
   * **Nodes:** 10 neurons (one for each digit).
   * **Activation:** Softmax function to provide probabilities for classification.

**Key Features**

* **Parameter Sharing:** Reduces the number of parameters compared to fully connected layers, making the model computationally efficient.
* **Pooling Layers:** Prevent overfitting and improve robustness by downsampling feature maps.
* **Layered Abstraction:** Each layer captures increasingly complex patterns, from simple edges to full digit shapes.

**Architecture Diagram**



**Training LeNet-5**

LeNet-5 is trained using the **supervised learning approach** with labeled data.

**Steps in Training:**

1. **Input Preparation**:
   * Preprocess input images (resize to 32×32, normalize pixel values).
2. **Forward Propagation**:
   * Pass the input through the layers of the network.
   * Convolutions extract spatial features; pooling reduces dimensions; fully connected layers generate predictions.
3. **Loss Calculation**:
   * Compare predictions with ground truth labels using a loss function.
     + Commonly used: **Cross-Entropy Loss** for classification.
4. **Backpropagation**:
   * Compute gradients of the loss function with respect to weights and biases using the **chain rule**.
5. **Optimization**:
   * Update weights using an optimization algorithm (e.g., **Stochastic Gradient Descent (SGD)**).
6. **Repeat**:
   * Iterate through the dataset for multiple epochs, minimizing the loss function.

**Evaluation of LeNet-5**

After training, LeNet-5 is evaluated to measure its generalization and performance.

**Steps in Evaluation:**

1. **Forward Pass on Test Data**:
   * Input the test images into the trained model and generate predictions.
2. **Comparison with Ground Truth**:
   * Compare the predicted labels with the actual labels.
3. **Metrics Calculation**:
   * Compute performance metrics, such as:
     + **Accuracy**: Proportion of correct predictions.
     + **Precision, Recall, F1-Score**: To handle imbalanced datasets.
     + **Confusion Matrix**: For detailed analysis of classification results.
4. **Generalization Check**:
   * Compare training and test performance to identify overfitting or underfitting.

**Applications of LeNet-5**

* **Handwritten Digit Recognition**: (e.g., MNIST dataset).
* **Document Processing**: Automated recognition of characters in scanned documents.
* **Historical Significance**: A foundational architecture for modern CNNs.

**Hyperparameter Tuning**

Hyperparameter tuning is a crucial step in improving the performance of a neural network model. In this section, we will discuss various hyperparameters used in training the CNN model for handwritten digit recognition using the MNIST dataset, and how adjusting these parameters can influence the accuracy of the model.

**Multilayer Perceptron (MLP)**

1. **Learning Rate:**
   * Value: 0.001
   * Used in the Adam optimizer to control the step size during optimization.
   * Low value chosen for smooth convergence.
2. **Number of Epochs:**
   * Value: 20
   * Defines how many times the entire dataset is passed through the model during training.
3. **Batch Size:**
   * It specifies the number of samples processed before the model updates the weights.
4. **Hidden Layer Sizes:**
   * First layer: 256
   * Second layer: 128
   * Third layer: 64
   * Determines the number of neurons in each layer, balancing expressiveness and computation.
5. **Activation Function:**
   * **ReLU (Rectified Linear Unit)** applied after each hidden layer to introduce non-linearity.

**Tuning:**

* You might tune the learning rate (e.g., using values like 0.001, 0.0005) or adjust the number of hidden neurons based on training/validation performance.

**Convolutional Neural Network (CNN)**

1. **Learning Rate:**
   * Value: 0.001
   * Used in the Adam optimizer for updating weights.
   * Default value commonly used for Adam optimizations.
2. **Number of Epochs:**
   * Value: 20
   * Chosen for a balance between training time and performance.
3. **Batch Size:**
   * Influences the stability and speed of training.
4. **Filter Size in Convolution:**
   * 3x3 kernel size for the convolutional layer with 16 filters in the first layer.
5. **Pooling:**
   * **Max Pooling** with a kernel size of 2x2 reduces spatial dimensions by half.
6. **Activation Function:**
   * **ReLU** after the convolutional layer.

**Tuning:**

* You could adjust:
  + Number of filters (e.g., increase from 16 to 32).
  + Kernel size (e.g., 5x5 instead of 3x3).
  + Learning rate for better convergence.

**LeNet-5**

1. **Learning Rate:**
   * Value: 0.001
   * Adam optimizer for adjusting weights, same as CNN.
2. **Number of Epochs:**
   * Value: 20
   * Similar to CNN for a balanced trade-off between training time and performance.
3. **Batch Size:**
   * Controls the gradient estimation stability during training.
4. **Kernel Size in Convolution:**
   * First convolutional layer: 5x5, padding=2.
   * Second convolutional layer: 5x5, no padding.
5. **Pooling:**
   * **Average Pooling** with 2x2 kernel size after each convolution layer.
6. **Fully Connected Layers:**
   * Three layers:
     + First FC: 120 units
     + Second FC: 84 units
     + Third FC: 10 units (output layer)
7. **Activation Function:**
   * **ReLU** used after all layers except the output layer.

**Tuning:**

* You could experiment with:
  + Learning rates (e.g., try 0.0005 or 0.01).
  + Increase/decrease kernel size in the convolutional layers (e.g., 3x3 or 7x7).
  + Add dropout for regularization.

**Comparison of MLP, CNN, and LeNet-5**

**MLP:**

* Advantages:
  + Simple and quick to train.
  + Works well for small datasets.
* Disadvantages:
  + Requires flattening images, losing spatial relationships.
  + Struggles to generalize on complex image data.

**CNN:**

* Advantages:
  + Leverages spatial relationships in data.
  + More robust feature extraction, leading to better performance.
* Disadvantages:
  + More computationally expensive than MLP.
  + Requires more advanced setup and tuning.

**LeNet-5:**

* Advantages:
  + Specifically designed for digit recognition and similar tasks.
  + Efficient training and inference due to optimized architecture.
  + Proven high accuracy and low computational cost for MNIST.
* Disadvantages:
  + Limited flexibility for larger and more complex datasets.

**Conclusion:**

In the comparison of three models—Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN), and LeNet-5—for handwritten digit recognition using the MNIST dataset, LeNet-5 stands out as the most suitable choice. Each model has strengths and weaknesses, but LeNet-5's design and performance make it particularly well-suited for this task.

LeNet-5’s design is highly tailored for recognizing handwritten digits. Its balanced architecture—comprising convolutional, pooling, and fully connected layers—makes it a compact yet powerful model. Unlike the MLP, which struggles to exploit spatial information, and the general CNN, which may be more complex than necessary, LeNet-5 offers an efficient solution optimized for MNIST’s specific characteristics.

Compared to the general CNN, LeNet-5 achieves similar or better accuracy with fewer trainable parameters, making it computationally efficient. This efficiency reduces the risk of overfitting while maintaining high accuracy, a critical advantage for tasks with relatively small datasets like MNIST. The simplicity of the architecture also enables faster training and lower resource requirements, further enhancing its suitability for this

While the MLP performed well with a 97% accuracy, it lacks the ability to learn spatial hierarchies in image data, a key requirement for image classification tasks. The general CNN improved on this by leveraging convolutional layers, achieving around 98% accuracy. However, it lacks the task-specific optimizations present in LeNet-5, which allow the latter to strike a balance between complexity and performance.

LeNet-5 emerges as the optimal model among the three for MNIST digit recognition due to its superior accuracy, task-specific architecture, and computational efficiency. Its design aligns perfectly with the dataset's requirements, enabling robust and reliable performance. While newer architectures may outperform LeNet-5 in more complex or large-scale datasets, it remains a benchmark for smaller-scale, grayscale image classification tasks, showcasing its enduring relevance and effectiveness.