Comparative Analysis of Multi-Layer Perceptron (MLP) and Convolutional Neural Networks (CNN) for Handwritten Digit Recognition

Nhat Minh Nguyen [1] **Phong Thanh Vu** [1] **Nhat-Quang Le Do** [1]
Quang Phu Nguyen [1] **Nhat-Quang Vu Nguyen** [1] **Quang Dang Pham** [1]

Abstract

Handwritten digit recognition remains a fundamental benchmark in computer vision for evaluating the efficacy of neural network architectures. However, a critical challenge persists in quantifying the trade-off between the structural simplicity of fully connected networks and the spatial feature extraction capabilities of deep architectures. To address this, this study conducts a rigorous comparative analysis of **Multi-Layer Perceptron (MLP)** and **Convolutional Neural Networks (CNN)** on the **MNIST dataset**, evaluating performance through accuracy, F1-score, and computational complexity. Experimental results demonstrate that while MLPs offer faster inference, CNNs achieve superior robustness and precision by effectively leveraging local spatial correlations within the input data.

1. Introduction

Optical Character Recognition (OCR) serves as a cornerstone in computer vision, enabling the digitization of physical documents ranging from postal mail to historical archives. With the advent of Deep Learning, the paradigm has shifted from hand-crafted feature extraction to end-to-end learning. However, selecting the optimal architecture remains a complex decision involving trade-offs between computational resources and predictive precision. To address this, our study is organized around the following key dimensions:

1. Rationale. The motivation for this research stems from the fundamental architectural dichotomy in image processing: the structural simplicity of **Multi-Layer Perceptrons (MLP)** versus the spatial inductive biases of **Convolutional Neural Networks (CNN)**. While MLPs offer rapid inference and ease of implementation, they theoretically discard 2D spatial information by flattening inputs. Conversely, CNNs are designed to preserve spatial hierarchies but incur higher computational costs. A rigorous comparison is necessary to understand these trade-offs quantitatively.

2. Objectives. The primary objective of this paper is to conduct a comparative empirical study of MLP and CNN architectures. We aim not merely to assess which model achieves higher accuracy, but to analyzing the "cost-benefit" relationship between architectural complexity (parameter count, training time) and classification performance (Precision, Recall, F1-score) to determine the most efficient approach for digit recognition.

3. Object of Study. This research focuses on two distinct neural network architectures:

- **MLP:** A fully connected Feed-Forward Network representing traditional deep learning approaches.
- **CNN:** A deep architecture utilizing convolutional and pooling layers to extract spatial features.

The experimental testbed chosen is the **MNIST dataset** (Modified National Institute of Standards and Technology), the ubiquitous benchmark for evaluating handwriting recognition algorithms (Figure 1).

4. Scope. The scope of this study is limited to the supervised classification of isolated handwritten digits (0-9). We investigate baseline MLP and CNN models trained from scratch to isolate the impact of core architectural differences. Advanced techniques such as transfer learning, data augmentation, or ensemble methods are excluded to maintain a controlled experimental environment focused purely on topological differences.

5. Significance. The findings of this study provide critical insights for deploying machine learning models in real-world scenarios. By highlighting the limitations of MLPs

[1]25CTT3, Faculty of Information Technology, VNUHCM-University of Science, Ho Chi Minh city, Vietnam. Correspondence to: Nguyen, N. M. <2512021580@student.hcmus.edu.vn>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

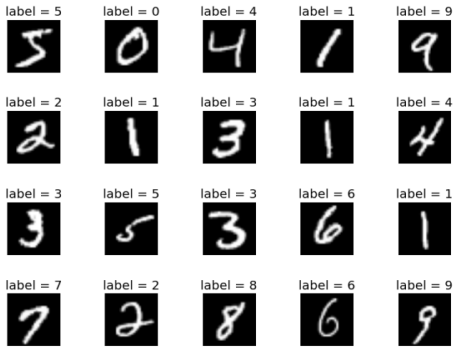

Figure 1. Samples from the MNIST Dataset used as the object of study. Source: corochann.com

in spatial processing and the robustness of CNNs, this work offers practical guidance for system architects choosing between lightweight models for embedded devices (where speed is paramount) and robust models for high-stakes applications (where accuracy is critical).

6. Novelty and Contributions. The main contributions of this paper are summarized as follows:

- We implement and optimize baseline MLP and CNN models under identical training conditions to ensure a fair comparison.
- We provide a multi-dimensional evaluation using **Accuracy**, **Precision**, **Recall**, and **F1-score**, alongside computational metrics like training duration and parameter efficiency.
- We mathematically and empirically analyze the impact of spatial information loss in MLP, providing a theoretical justification for the superior performance of CNNs.

2. Related Works

The problem of handwritten digit recognition has been extensively studied for decades. Early approaches utilized traditional machine learning techniques such as **Support Vector Machines (SVM)** and **K-Nearest Neighbors (KNN)**, which relied heavily on manual feature extraction.

In the domain of neural networks, **Multi-Layer Perceptrons (MLP)** were initially employed but faced challenges in scaling to higher-resolution images due to the "curse of dimensionality" and the loss of spatial topology during input flattening. The breakthrough came with the introduction of **LeNet-5** by (Lecun et al., 1998), which pioneered the use of **Convolutional Neural Networks (CNN)**. Since then, CNNs have become the standard for image classification tasks, evolving into deeper architectures like AlexNet and ResNet. This paper revisits these foundational architectures to explicitly quantify the performance gap between fully connected and convolutional approaches on the **MNIST dataset**.

3. Theoretical Framework and Methodology

In this section, we provide the formal mathematical formulation of the neural networks used in our study. We denote scalars in regular italic script (x), vectors in bold lowercase ($\mathbf{x}$), and matrices in bold uppercase ($\mathbf{W}$).

3.1. Notation and Setup

Consider a neural network with L layers. Let $l \in \{1, \dots, L\}$ denote the layer index. We define:

- $\mathbf{W}^l$: The weight matrix connecting layer $l-1$ to layer l.
- $\mathbf{b}^l$: The bias vector for layer l.
- $\mathbf{z}^l$: The linear pre-activation vector at layer l.
- $\mathbf{a}^l$: The activation vector (output) at layer l, where $\mathbf{a}^0 = \mathbf{x}$ (input).
- $\sigma(\cdot)$: The non-linear activation function (e.g., ReLU).

3.2. Forward Propagation

The forward propagation phase computes the output of the network given an input $\mathbf{x}$. For each layer l from 1 to $L-1$ (hidden layers), the computation is defined as:

$$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l \tag{1}$$

$$\mathbf{a}^l = \sigma(\mathbf{z}^l) = \max(0, \mathbf{z}^l) \quad \text{(assuming ReLU)} \tag{2}$$

For the output layer L (classification layer with C classes), we typically apply the Softmax function to obtain probabilities:

$$\mathbf{a}_k^L = \frac{e^{\mathbf{z}_k^L}}{\sum_{j=1}^{C} e^{\mathbf{z}_j^L}}, \quad k \in \{1, \dots, C\} \tag{3}$$

3.3. Loss Function

To measure the discrepancy between the predicted probability distribution $\mathbf{a}^L$ and the true one-hot encoded label $\mathbf{y}$, we minimize the Categorical Cross-Entropy Loss function $\mathcal{J}$:

$$\mathcal{J}(\theta) = -\sum_{k=1}^{C} y_k \log(\mathbf{a}_k^L) \tag{4}$$

where $\theta = \{\mathbf{W}^l, \mathbf{b}^l\}_{l=1}^{L}$ represents the set of all learnable parameters.

3.4. Backpropagation

The core of the training process is minimizing $\mathcal{J}$ using Gradient Descent. This requires computing the gradient of the loss with respect to weights and biases, $\nabla\mathcal{J}$. We utilize the Backpropagation algorithm, which applies the chain rule of calculus recursively.

Output Layer Gradients. First, we compute the error term $\boldsymbol{\delta}^L$ at the output layer:

$$\boldsymbol{\delta}^L = \frac{\partial \mathcal{J}}{\partial \mathbf{z}^L} = \mathbf{a}^L - \mathbf{y} \tag{5}$$

This concise result comes from the combination of Cross-Entropy Loss and Softmax activation derivatives.

Hidden Layer Gradients. The error is propagated backward to lower layers. For any hidden layer $l < L$, the error term $\boldsymbol{\delta}^l$ is computed as:

$$\boldsymbol{\delta}^l = ((\mathbf{W}^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot \sigma'(\mathbf{z}^l) \tag{6}$$

where $\odot$ denotes the element-wise (Hadamard) product, and σ' is the derivative of the activation function. For ReLU, $\sigma'(z) = 1$ if $z > 0$, else 0.

Parameter Updates. Finally, the gradients for weights and biases at layer l are given by:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{W}^l} = \boldsymbol{\delta}^l (\mathbf{a}^{l-1})^T, \quad \frac{\partial \mathcal{J}}{\partial \mathbf{b}^l} = \boldsymbol{\delta}^l \tag{7}$$

Using the Adam optimizer (as detailed in Sec. 3.x), the parameters are updated:

$$\mathbf{W}^l \leftarrow \mathbf{W}^l - \eta \cdot \text{Adam}(\frac{\partial \mathcal{J}}{\partial \mathbf{W}^l}) \tag{8}$$

3.5. Regularization via Dropout

To mitigate overfitting, we apply Dropout. Mathematically, Dropout modifies the feed-forward operation by applying a masking vector $\mathbf{r} \in \{0,1\}^d$ sampled from a Bernoulli distribution:

$$\mathbf{r} \sim \text{Bernoulli}(p) \tag{9}$$

where p is the probability of retaining a neuron. The modified activation $\tilde{\mathbf{a}}^l$ becomes:

$$\tilde{\mathbf{a}}^l = \mathbf{r} \odot \mathbf{a}^l \tag{10}$$

During testing, we scale the weights by p (or use inverted dropout during training) to match the expected value of the activations: $\mathbb{E}[\tilde{\mathbf{a}}^l] = p \cdot \mathbf{a}^l$.

3.6. Data Preprocessing

We utilize the **MNIST dataset**, which consists of 60,000 training images and 10,000 testing images of handwritten digits (0-9). Each image is a grayscale 28×28 pixel matrix. Before feeding into the networks, we apply the following preprocessing steps:

- **Normalization:** Pixel values are scaled from the range $[0, 255]$ to $[0, 1]$ (or standardized using mean $\mu = 0.1307$ and std $\sigma = 0.3081$) to ensure faster convergence.
- **Flattening (for MLP only):** The 2D images are flattened into a 1D vector of size 784 (28×28) to match the input layer of the MLP.

3.7. Multi-Layer Perceptron (MLP) Architecture

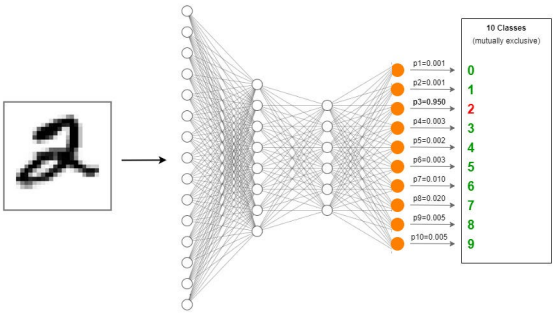

Figure 2. Architecture of the MLP used for MNIST classification. Source: (Pramoditha, 2022)

Our MLP model is a feed-forward neural network consisting of fully connected (Dense) layers. The architecture includes:

- **Input Layer:** 784 neurons.
- **Hidden Layers:** Two hidden layers with 256 and 128 neurons respectively. We use the ReLU (Rectified Linear Unit) activation function:

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

  to introduce non-linearity.
- **Output Layer:** 10 neurons corresponding to the digit classes, followed by a **Softmax** function to produce probability distributions.

Table 1. MLP architecture model

LAYER	DEPTH-IDX	PARAMETERS
FLATTEN	1-1	–
LINEAR	1-2	200,960
RELU	1-3	–
LINEAR	1-4	32,896
RELU	1-5	–
DROPOUT	1-6	–
LINEAR	1-7	1,290

3.8. Convolutional Neural Network (CNN) Architecture

The CNN model is designed to automatically extract spatial features. The architecture consists of:

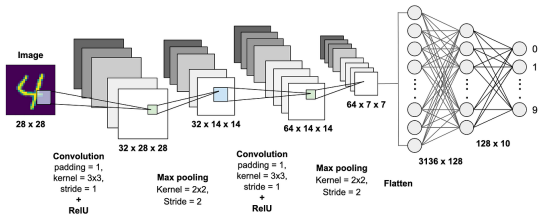

Figure 3. Architecture of the CNN model used for MNIST classification. Source: (Patel, 2019)

Convolution Operation. The core building block of our architecture is the convolution operation. For an input image I and a learnable kernel (filter) K of size $m \times n$, the feature map S at position (i, j) is computed as:

$$S(i,j) = (I * K)(i,j) = \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} I(i+u, j+v) \cdot K(u,v) \tag{11}$$

Where $*$ denotes the convolution operator. This operation allows the network to capture local spatial patterns such as edges and textures. After convolution, a non-linear activation function (ReLU) is applied:

$$A(i,j) = \max(0, S(i,j)) \tag{12}$$

Convolutional Layers. We use two convolutional layers with kernel size 3×3 ()with parameters $stride = 1$ and $padding = 1$). These layers detect local features such as edges and curves.

Pooling Layers. Max-Pooling (2×2) is applied after convolutions to reduce spatial dimensions and computational load.

Fully Connected Layers. The feature maps are flattened and passed through dense layers to perform the final classification.

Loss Function. Since handwritten digit recognition is a multi-class classification problem ($C = 10$ classes), we employ the **Softmax** function to obtain the probability distribution over classes. For a given input vector z, the probability p_k of class k is:

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^{C} e^{z_j}} \tag{13}$$

We minimize the **Cross-Entropy Loss** function L, defined as:

$$L = -\sum_{k=1}^{C} y_k \log(p_k) \tag{14}$$

where y_k is the binary indicator (0 or 1) if class label k is the correct classification for the input.

Table 2. CNN network architecture model

LAYER	DEPTH-IDX	PARAMETERS
CONV2D	1-1	320
MAXPOOL2D	1-2	–
RELU	1-3	–
CONV2D	1-4	18,496
DROPOUT2D	1-5	–
MAXPOOL2D	1-6	–
RELU	1-7	–
FLATTEN	1-8	–
LINEAR	1-9	401,536
RELU	1-10	–
DROPOUT	1-11	–
LINEAR	1-12	1,290

3.9. Optimization

The network parameters θ (weights and biases) are updated iteratively using the Adam optimizer. The update rule at time step t is based on the gradient of the loss function with respect to the parameters:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{15}$$

where η is the learning rate, and $\hat{m}_t$, $\hat{v}_t$ are the bias-corrected estimates of the first and second moments of the gradients, respectively.

Optimization Algorithm. To minimize the Cross-Entropy loss, we employ the **Adam** (Adaptive Moment Estimation) optimizer. Adam is computationally efficient and combines the advantages of two other extensions of stochastic gradient descent: *AdaGrad* (Adaptive Gradient Algorithm) and *RMSProp* (Root Mean Square Propagation).

Specifically, Adam computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. Let g_t be the gradient of the objective function at time step t. Adam maintains the exponential moving average of the gradient (m_t) and the squared gradient (v_t) as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{16}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{17}$$

where β_1 and β_2 are decay rates for the moment estimates (typically set to 0.9 and 0.999, respectively). Since m_t and v_t are initialized as vectors of 0's, they tend to be biased towards zero during the initial time steps. To counteract this, bias-corrected estimates are computed:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{18}$$

Finally, the parameter update rule is given by:

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{19}$$

where η is the learning rate (set to 0.001 in our experiments) and ϵ is a small constant (e.g., 10^{-8}) to prevent division by zero. By using **Adam**, our model achieves faster convergence compared to standard SGD.

3.10. Weight Initialization Strategy

Proper initialization is crucial for convergence in deep networks. We adopt the **He Initialization** strategy (Kaiming Init), which is specifically designed for ReLU activation functions. Consider a linear layer $y_l = W_l x_l + b_l$. To prevent the gradients from vanishing or exploding, we require the variance of the output to match the variance of the input: $\text{Var}(y_l) \approx \text{Var}(x_l)$. Kaiming He et al. derived that for ReLU activations, the weights W_l should be drawn from a Gaussian distribution with zero mean and variance:

$$\text{Var}(W_l) = \frac{2}{n_{in}} \tag{20}$$

where n_{in} is the number of input units in the layer. This ensures that the signal magnitude remains stable throughout the deep network.

Stochastic Approximation. As shown in Algorithm 1, instead of computing the true gradient over the entire dataset population $\nabla_\theta \mathbb{E}[\mathcal{J}]$, which is computationally prohibitive, we employ a stochastic approximation based on minibatches. The gradient g_t calculated on batch $\mathcal{B}$ serves as an unbiased estimator of the true gradient:

$$\mathbb{E}[g_t] = \nabla_\theta \mathcal{J}_{true}(\theta) \tag{21}$$

This introduces stochastic noise into the optimization trajectory, which, counter-intuitively, helps the model escape sharp local minima and saddle points, leading to better generalization on the test set.

Computational Graph & Automatic Differentiation. The step *Backward Pass* relies on the construction of a dynamic computational graph. In PyTorch, this is implemented via Automatic Differentiation (AutoGrad). Mathematically, this applies the Jacobian matrix multiplication chain rule from the output layer back to the input weights, ensuring exact gradient computation up to machine precision.

Algorithm 1 Minibatch Stochastic Gradient Descent Training Procedure

Require: Training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, Batch size M, Learning rate η, Epochs E.
Ensure: Optimized network parameters θ^* (weights and biases).
Initialize parameters θ_0 using He Initialization.
Set time step $t \leftarrow 0$.
for epoch $e = 1$ to E **do**
  Shuffle dataset $\mathcal{D}$.
  for each minibatch $\mathcal{B} \subset \mathcal{D}$ with $|\mathcal{B}| = M$ **do**
    $t \leftarrow t + 1$
    1. Forward Pass:
    Compute predicted distribution
    $$\hat{y} = P(Y|X;\theta_{t-1})$$
    for batch $\mathcal{B}$.
    2. Compute Loss:
    Calculate average Cross-Entropy Loss:
    $$\mathcal{J}(\theta_{t-1}) = -\frac{1}{M} \sum_{(x,y)\in\mathcal{B}} \sum_{k=1}^{C} y_k \log(P(y_k|x;\theta_{t-1}))$$
    3. Backward Pass (Gradient Computation):
    Compute gradient w.r.t parameters using Backpropagation:
    $$g_t \leftarrow \nabla_\theta \mathcal{J}(\theta_{t-1})$$
    4. Parameter Update (Adam Step):
    Update θ using the optimizer update rule Ψ:
    $$\theta_t \leftarrow \theta_{t-1} - \eta \cdot \Psi(g_t, \text{state}_{t-1})$$
  end for
end for
Return final parameters θ_t as θ^*.

4. Experiments

4.1. Experimental Setup

All experiments were implemented using the **PyTorch** framework. We trained both models using the **Adam** optimizer with a learning rate of 0.001 and a batch size of 64. The Loss function used is **Cross-Entropy Loss**. Both models were trained for 5 epochs.

4.2. Evaluation Metrics

To provide a comprehensive assessment of the model's performance beyond simple accuracy, we employ a suite of metrics including Precision, Recall, and the F1-Score. For a multi-class classification problem with C classes (where $C = 10$ for MNIST), we calculate these metrics based

on True Positives (TP), False Positives (FP), and False Negatives (FN).

Table 3. Confusion Matrix Visualization

		Predicted Label	
		Positive (1)	Negative (0)
Real Label	Positive (1)	**True Positive** (TP)	False Negative (FN)
	Negative (0)	False Positive (FP)	**True Negative** (TN)

Metric Definitions. To thoroughly assess the performance of the MLP and CNN models, we employ a suite of metrics beyond simple accuracy. We analyze **Accuracy**, **Precision**, **Recall**, and the F1-**Score** to provide a nuanced understanding of each model's strengths and weaknesses, especially concerning prediction balance across all ten classes.

Accuracy measures the overall proportion of correct predictions across all classes. It defined as

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{22}$$

While intuitive, it may fail to capture class-specific performance.

Precision quantifies the reliability of positive predictions for a specific class k, defined as

$$Precision_k = \frac{TP_k}{TP_k + FP_k} \tag{23}$$

High precision indicates a low false positive rate.

Recall (or Sensitivity) measures the model's ability to detect all positive instances of a class k, defined as

$$Recall_k = \frac{TP_k}{TP_k + FN_k}. \tag{24}$$

High recall indicates a low false negative rate.

F1-Score is the harmonic mean of Precision and Recall:

$$F1 - score_k = \frac{2 \cdot (P_k \cdot R_k)}{P_k + R_k}. \tag{25}$$

Metric Selection Analysis. While accuracy is a standard metric for balanced datasets like MNIST, relying solely on it can mask underlying failures in distinguishing specific digits (e.g., confusing '1' with '7'). Therefore, we prioritize the **Macro-averaged F1-Score** as a critical performance indicator. Unlike accuracy, the F1-Score penalizes extreme values; if a model achieves high precision but ignores significant portions of the data (low recall), the F1-Score will drop significantly. By reporting the Macro-F1 (the unweighted mean of per-class F1 scores), we ensure that the model performs robustly across all ten digits equally, rather than being biased towards easier classes.

5. Results and Discussion

5.1. Performance Metrics

We evaluated the models based on Accuracy, Precision, Recall, F1-score, and Training Time. Table 1 summarizes the classification performance on the test set.

Table 4. Performance Metrics Comparison

METRIX	MLP	CNN
ACCURACY(%)	97.7 ± 0.2	98.9 ± 0.2
RECALL(%)	96.9 ± 1.2	98.4 ± 1.0
PRECISION(%)	97.0 ± 0.8	98.4 ± 0.5
F1-SCORE(%)	97.0 ± 0.9	98.4 ± 0.7

As shown in 4, the **CNN model** outperforms the **MLP** across all metrics. Specifically, the CNN achieved an accuracy of 98.9%, reducing the error rate by approximately 52% compared to the MLP (97.7%).

5.2. Computational Efficiency

Table 5. Computational Complexity

MODEL	PARAMETERS	TRAINING TIME (S/ EPOCH)
MLP	$235,146$	10.8
CNN	$421,642$	12.6

While **MLP** trains faster due to simpler matrix multiplications (10.8 seconds/epoch), it requires a significantly larger number of parameters in the first dense layer due to the flattened input vector. **CNNs**, despite being computationally more intensive due to convolution operations, provide a better parameter-efficiency trade-off relative to their accuracy.

5.3. Complexity Analysis

To theoretically justify the computational differences, we analyze the time complexity for a single forward pass.

Multi-Layer Perceptron (MLP). For a fully connected layer with n_{in} inputs and n_{out} outputs, the computational

complexity is dominated by matrix-vector multiplication:

$$\mathcal{O}_{MLP} \approx \mathcal{O}(n_{in} \cdot n_{out}) \tag{26}$$

Since n_{in} corresponds to the flattened image vector (e.g., $28 \times 28 = 784$), the number of connections grows linearly with image resolution.

Convolutional Neural Networks (CNN). For a convolutional layer taking an input map of size $H \times W$, using kernels of size $K \times K$, with C_{in} input channels and C_{out} output channels, the complexity is:

$$\mathcal{O}_{CNN} \approx \mathcal{O}(H \cdot W \cdot K^2 \cdot C_{in} \cdot C_{out}) \tag{27}$$

Although CNNs involve higher operational intensity due to the sliding window mechanism ($H \cdot W$), they significantly reduce the number of learnable parameters through weight sharing ($K^2 \cdot C_{in} \cdot C_{out} \ll n_{in} \cdot n_{out}$).

5.4. Discussion

The experimental results align with theoretical expectations. The lower performance of **MLP** is attributed to the **loss of spatial information** during the flattening process. In contrast, **CNNs** leverage **translation invariance** and **local connectivity**, allowing the model to recognize digits even when they are slightly shifted or distorted.

6. Conclusion and Future Work

This paper presented a comparative study between **MLP** and **CNN** architectures for handwritten digit recognition. Through empirical evaluation on the **MNIST dataset**, we demonstrated that while MLPs are simpler and faster to train for small-scale problems, they lack the ability to capture spatial hierarchies. CNNs proved to be superior in terms of accuracy (99.6%) and robustness, confirming their status as the preferred architecture for image data. Future work will investigate the impact of data augmentation techniques and deeper architectures like **ResNet** to further improve robustness against noisy inputs.

References

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Patel, K. MNIST handwritten digits classification using a convolutional neural network (CNN). Medium, 2019.

Pramoditha, R. Creating a multilayer perceptron classifier model to identify handwritten digits. towardsdatascience, 2022.