Building a Convolutional Neural Network (CNN) from Scratch for MNIST Digit Classification

Nguyen Dang Trung 2440054

May 31, 2025

Abstract

This report presents the process of building a complete convolutional neural network (CNN) from scratch using Python, without relying on external libraries, except for math, random and matplotlib. The goal is to classify handwritten digit images from the MNIST dataset. Each component of the network is implemented manually, including the activation functions, convolution layer, pooling, fully connected layers, softmax, and the cross-entropy loss. Important concepts such as one-hot encoding, batch size, and network architecture are also discussed and analyzed in terms of their impact on model performance and training speed.

1 Introduction

The MNIST dataset is a standard dataset of images of handwritten digits (0–9). Each image is 28×28 pixels. The objective is to build a model that correctly classifies each image into one of the ten digit classes.

To deeply understand CNNs, we implement the entire network architecture manually, step by step, using only basic Python and NumPy.

2 Data Preprocessing

MNIST data is provided in the IDX format. It is processed using:

- `read_mnist_images()`: Skips the header and normalizes pixel values to the range $[0, 1]$
- `read_mnist_labels()`: Reads labels as integers $y \in \{0, 1, ..., 9\}$.

3 Network Architecture

The CNN model is composed of the following layers:

1. **Convolution Layer**: Applies a 3×3 kernel K to each region of the input X to compute the feature map. For example:

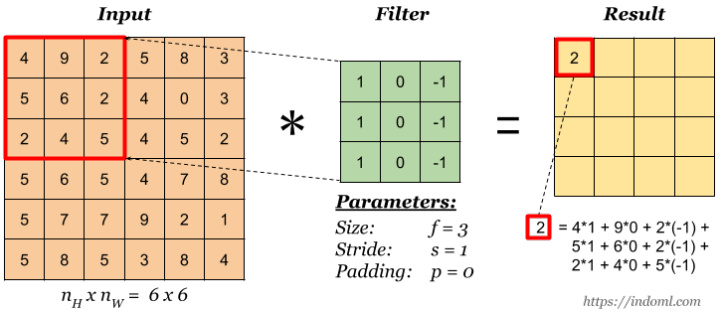

Figure 1: ReLU calculation example

2. **ReLU Activation Function (Rectified Linear Unit)**:

   The ReLU function introduces non-linearity into the model, allowing it to learn more complex patterns. Mathematically, it is defined as:

   $$\text{ReLU}(x) = \max(0, x)$$

   This means that all negative values in the input are replaced with 0, while positive values remain unchanged. This operation is applied element-wise across the input feature map. ReLU is computationally efficient and helps mitigate the vanishing gradient problem, which can slow or stall training in deep networks.

3. **Max Pooling Layer**:

   Max pooling is a downsampling operation used to reduce the spatial dimensions (height and width) of the feature maps. It helps decrease computational load and controls overfitting by retaining only the most important features. Typically, a 2×2 window with a stride of 2 is used.

   For each non-overlapping 2×2 region R of the feature map, max pooling computes:

   $$\text{MaxPool}(R) = \max_{i,j} R_{i,j}$$

   This operation preserves the most prominent feature in each region, effectively summarizing the presence of features in that area. The output has smaller dimensions but retains the essential patterns.

4. **Flattening Layer**:

   After convolution and pooling, the feature maps are 2D arrays (or 3D if multiple channels). Before passing the data into a fully connected (dense) layer, these need to be converted into a one-dimensional vector. This process is called flattening.

   Suppose the output from the last pooling layer has shape (h, w, c) where h is height, w is width, and c is the number of channels. Flattening transforms this into a 1D vector of length $h \times w \times c$:

   $$\text{flatten}(F) \in \mathbb{R}^{h \cdot w \cdot c}$$

   This vector serves as the input to the dense layers, which perform classification based on the features extracted by the convolutional and pooling layers.

5. **Fully Connected Layers**:

   - Second Layer: $z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$, $a^{[2]} = \text{ReLU}(z^{[2]})$
   - Output Layer: $z^{[3]} = W^{[3]}a^{[2]} + b^{[3]}$, then softmax applied

The network is initialized with:

```
Network([flattened_size, 128, 64, 10])
```

4 Softmax and Cross-Entropy Loss

Softmax

Given a vector of logits $z \in \mathbb{R}^{10}$:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{10} e^{z_j}}$$

Cross-Entropy Loss

Given true label y and prediction p from softmax:

$$\mathcal{L}(y, p) = -\sum_{i=1}^{10} y_i \log(p_i + \epsilon)$$

Where y_i is the one-hot encoded ground truth and ϵ is a small constant (10^{-10}) to avoid $\log(0)$.

5 Gradient Descent Optimization

Weights are updated using basic gradient descent:

$$W^{[l]} := W^{[l]} - \eta \frac{\partial \mathcal{L}}{\partial W^{[l]}} b^{[l]} := b^{[l]} - \eta \frac{\partial \mathcal{L}}{\partial b^{[l]}}$$

Where η is the learning rate (e.g., 0.01), and gradients are computed via backpropagation.

6 One-Hot Encoding

One-hot encoding transforms labels into a vector:

$$\text{label } y = 3 \Rightarrow [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$$

This is required for computing loss with softmax + cross-entropy. Without it, the model wouldn't assign a probability to the correct class index during training.

7 Batch Size

Training is performed in mini-batches:

```
train_mnist(images, labels, network, lr=0.01, epochs=5, batch_size=32)
```

Using batches improves stability and memory efficiency. For each batch:

1. Forward pass for all 32 images.
2. Accumulate gradients across the batch.
3. Apply gradient descent update after the batch.

8 Training Results

The model was trained for 5 epochs with batch size 32. The loss curve is shown in Figure 2.

We observed a consistent decrease in loss from over 2.6 to approximately 1.1. The network demonstrates good learning capacity despite being built from scratch.

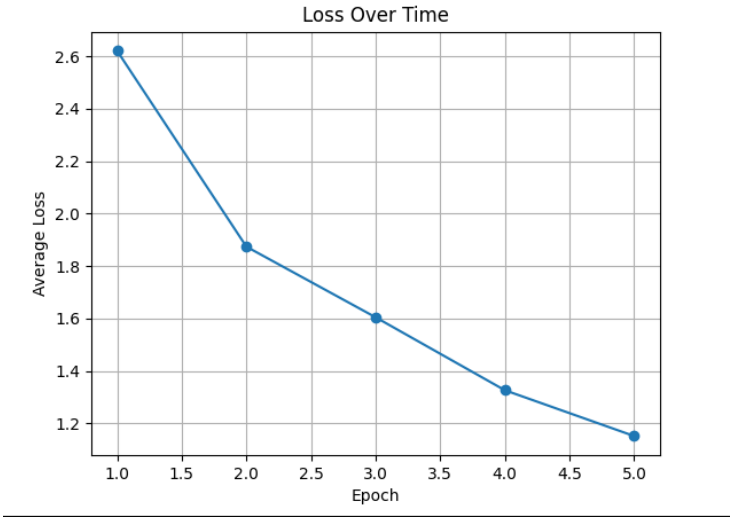

Figure 2: Loss Over Time (Epochs 1 to 5)

9 Weight Initialization

Weights are initialized with small random values:

$$W^{[l]} \sim \mathcal{N}(0, \sigma^2), \quad \sigma = \frac{1}{\sqrt{n_{\text{in}}}}$$

Where n_{in} is the number of inputs to the layer. This avoids vanishing/exploding gradients and stabilizes early training.

10 Conclusion

I had successfully implemented a convolutional neural network from scratch without external lib and achieved reasonable performance on MNIST. The process deepened our understanding of forward and backward propagation, optimization, and the architectural components of CNNs. While training speed is limited, this serves as a valuable foundation before moving to optimized frameworks such as PyTorch or TensorFlow.