Convolutional Neural Networks (CNN): Mathematical Notes

Convolutional Neural Networks (CNNs) are a specialized kind of neural network for processing data that has a known grid-like topology, such as time-series data (1D grid) or image data (2D grid). They employ a mathematical operation called convolution in place of general matrix multiplication in at least one of their layers.

I. Core Mathematical Formulations

Unlike standard Multilayer Perceptrons (MLPs), CNNs use specialized operations to extract spatial hierarchies of features.

1. The Convolution Operation (2D Cross-Correlation)

In machine learning, what is often called "convolution" is technically cross-correlation. For a 2D image $\mathbf{X}$ and a 2D kernel/filter $\mathbf{K}$ of size $F \times F$, the operation is defined as:

$$S(i,j) = (\mathbf{X} * \mathbf{K})(i,j) = \sum_{m=0}^{F-1} \sum_{n=0}^{F-1} \mathbf{X}(i+m, j+n)\mathbf{K}(m,n)$$

where $S(i,j)$ is the value of the output feature map at position (i,j).

2. Common Activation Functions

Activations introduce non-linearity into the network, allowing it to learn complex patterns.

- **ReLU (Rectified Linear Unit):** The standard activation for hidden CNN layers.

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

- **Sigmoid:** Often used in binary classification output layers.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- **Softmax:** Used in multi-class classification output layers to produce probabilities.

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

3. Loss Functions

To update the weights via backpropagation, a loss function L measures the prediction error.

- **Mean Squared Error (MSE):** Commonly used for regression tasks.

$$L = \frac{1}{2} \sum_i (\hat{y}_i - t_i)^2$$

- **Categorical Cross-Entropy:** Standard for multi-class classification.

$$L = -\sum_i t_i \log(\hat{y}_i)$$

II. Step-by-Step Numerical Calculation (4×4 Input)

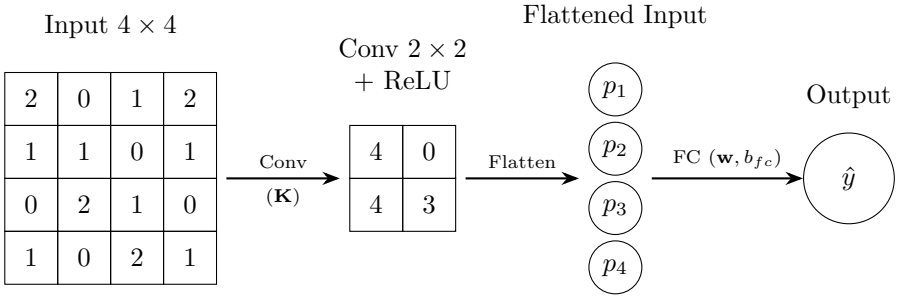
We define a minimal CNN problem to trace the forward and backward passes. We are given an Input matrix $\mathbf{X}$, a Convolutional Filter $\mathbf{K}$, Fully Connected (FC) weights $\mathbf{w}$, a bias b_{fc}, a target t, and a learning rate α.

$$\mathbf{X} = \begin{pmatrix} 2 & 0 & 1 & 2 \\ 1 & 1 & 0 & 1 \\ 0 & 2 & 1 & 0 \\ 1 & 0 & 2 & 1 \end{pmatrix} \qquad \mathbf{K} = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 1 & 0 & 1 \end{pmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 0.2 \\ -0.5 \\ 0.4 \\ 0.1 \end{bmatrix}$$

$b_{fc} = 0.2$
Target $t = 0$
Rate $\alpha = 0.1$

Network Architecture Diagram



Step 1: Convolution Operation & ReLU

We apply the 3×3 filter $\mathbf{K}$ to the 4×4 input $\mathbf{X}$ with a stride of 1, omitting padding.

- **Top-Left (z_{11}):** $z_{11} = 2(1) + 0(-1) + 1(0) + 1(0) + 1(1) + 0(-1) + 0(1) + 2(0) + 1(1) = 4$
- **Top-Right (z_{12}):** $z_{12} = 0(1) + 1(-1) + 2(0) + 1(0) + 0(1) + 1(-1) + 2(1) + 1(0) + 0(1) = 0$
- **Bottom-Left (z_{21}):** $z_{21} = 1(1) + 1(-1) + 0(0) + 0(0) + 2(1) + 1(-1) + 1(1) + 0(0) + 2(1) = 4$
- **Bottom-Right (z_{22}):** $z_{22} = 1(1) + 0(-1) + 1(0) + 2(0) + 1(1) + 0(-1) + 0(1) + 2(0) + 1(1) = 3$

Pre-activation output: $\mathbf{Z} = \begin{pmatrix} 4 & 0 \\ 4 & 3 \end{pmatrix}$. Applying ReLU ($f(z) = \max(0, z)$), the matrix remains unchanged.

Step 2: Flatten & Fully Connected Layer

The 2×2 output is flattened into $\mathbf{p} = [4, 0, 4, 3]^T$. The final prediction $\hat{y}$ is:

$$\hat{y} = \mathbf{w}^T \mathbf{p} + b_{fc}$$
$$\hat{y} = (0.2)(4) + (-0.5)(0) + (0.4)(4) + (0.1)(3) + 0.2$$
$$\hat{y} = 0.8 + 0 + 1.6 + 0.3 + 0.2 = 2.9$$

Step 3: Backward Pass (Weight Updating)

Using MSE loss: $L = \frac{1}{2}(\hat{y} - t)^2$. The gradient with respect to $\hat{y}$ is $\frac{\partial L}{\partial \hat{y}} = \hat{y} - t = 2.9 - 0 = 2.9$.

Gradients for weights via chain rule ($\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \cdot p_i$):

$$\frac{\partial L}{\partial w_1} = 2.9 \cdot 4 = 11.6, \quad \frac{\partial L}{\partial w_2} = 2.9 \cdot 0 = 0$$
$$\frac{\partial L}{\partial w_3} = 2.9 \cdot 4 = 11.6, \quad \frac{\partial L}{\partial w_4} = 2.9 \cdot 3 = 8.7, \quad \frac{\partial L}{\partial b_{fc}} = 2.9 \cdot 1 = 2.9$$

Applying gradient descent ($w^{(new)} = w^{(old)} - \alpha \frac{\partial L}{\partial w}$):

$$\mathbf{w}^{(new)} = \begin{bmatrix} 0.2 - 1.16 \\ -0.5 - 0 \\ 0.4 - 1.16 \\ 0.1 - 0.87 \end{bmatrix} = \begin{bmatrix} -0.96 \\ -0.50 \\ -0.76 \\ -0.77 \end{bmatrix}, \quad b_{fc}^{(new)} = 0.2 - 0.29 = -0.09$$

Final Updated Parameters

$$\mathbf{w}^{(new)} = \begin{bmatrix} -0.96 \\ -0.50 \\ -0.76 \\ -0.77 \end{bmatrix}, \quad b_{fc}^{(new)} = -0.09$$

III. Deep Architecture & Parameter Flow

To understand how CNNs scale, let us trace a $16 \times 16 \times 3$ (RGB) input image through a multi-layer architecture.

Spatial Dimension Output Formula

For both Convolutional and Pooling layers, the spatial size of the output feature map is calculated as:

$$W_{out} = \left\lfloor \frac{W_{in} - F + 2P}{S} \right\rfloor + 1$$
$$H_{out} = \left\lfloor \frac{H_{in} - F + 2P}{S} \right\rfloor + 1$$

Parameter Calculation Formulas

For a Convolutional Layer:
Each filter has $F \times F \times C_{in}$ weights, plus 1 bias term. If we have C_{out} filters, the total parameters are:

$$\text{Parameters}_{conv} = (F \cdot F \cdot C_{in} + 1) \cdot C_{out}$$

For a Fully Connected (Dense) Layer:
If the layer has N_{in} input neurons and N_{out} output neurons:

$$\text{Parameters}_{fc} = (N_{in} + 1) \cdot N_{out}$$

For a Pooling Layer:
Pooling layers compute a fixed mathematical operation (like max or average) and have no weights or biases to learn.

$$\text{Parameters}_{pool} = 0$$

Where:

- W_{in}, H_{in} = Width and Height of the input feature map
- C_{in} = Number of input channels (depth)
- W_{out}, H_{out} = Width and Height of the output feature map
- C_{out} = Number of output channels (number of filters applied)
- F = Filter/Kernel spatial size (assuming a square filter of $F \times F$)
- S = Stride (step size of the filter)
- P = Padding (number of zero-pixels added to the spatial borders)

IV. Step-by-Step Network Example

Let's trace the full architecture classifying the image into 10 categories.

Layer 0: Input Layer

- **Description:** The raw image data. Since it is an RGB image, it has 3 color channels.
- **Dimensions:** $16 \times 16 \times 3$ ($W_{in} = 16$, $H_{in} = 16$, $C_{in} = 3$)
- **Parameters:** 0

Layer 1: Convolutional Layer (Conv1)

- **Hyperparameters:** Number of Filters (C_{out}) = 8, Filter Size (F) = 3×3, Stride (S) = 1, Padding (P) = 1
- **Output Dimension Calculation:**

$$W_{out} = \left\lfloor \frac{16 - 3 + 2(1)}{1} \right\rfloor + 1 = 16$$

- **Output Shape:** $16 \times 16 \times 8$
- **Parameter Calculation:**

$$\text{Params} = (3 \cdot 3 \cdot 3 + 1) \cdot 8 = (27 + 1) \cdot 8 = 28 \cdot 8 = 224$$

Layer 2: Max Pooling Layer (Pool1)

- **Hyperparameters:** Filter Size (F) = 2×2, Stride (S) = 2, Padding (P) = 0
- **Output Dimension Calculation:**

$$W_{out} = \left\lfloor \frac{16 - 2 + 2(0)}{2} \right\rfloor + 1 = \lfloor 7 \rfloor + 1 = 8$$

- **Output Shape:** $8 \times 8 \times 8$
- **Parameter Calculation:** 0

Layer 3: Convolutional Layer (Conv2)

- **Hyperparameters:** Number of Filters (C_{out}) = 16, Filter Size (F) = 3×3, Stride (S) = 1, Padding (P) = 0
- **Output Dimension Calculation:**

$$W_{out} = \left\lfloor \frac{8 - 3 + 2(0)}{1} \right\rfloor + 1 = \lfloor 5 \rfloor + 1 = 6$$

- **Output Shape:** $6 \times 6 \times 16$
- **Parameter Calculation:**

$$\text{Params} = (3 \cdot 3 \cdot 8 + 1) \cdot 16 = (72 + 1) \cdot 16 = 73 \cdot 16 = 1168$$

Layer 4: Max Pooling Layer (Pool2)

- **Hyperparameters:** Filter Size (F) = 2×2, Stride (S) = 2, Padding (P) = 0
- **Output Dimension Calculation:**

$$W_{out} = \left\lfloor \frac{6 - 2 + 2(0)}{2} \right\rfloor + 1 = \lfloor 2 \rfloor + 1 = 3$$

- **Output Shape:** $3 \times 3 \times 16$
- **Parameter Calculation:** 0

Layer 5: Flattening

- **Description:** Flattens the 3D feature map to a 1D vector.
- **Output Calculation:** $N_{in} = 3 \cdot 3 \cdot 16 = 144$
- **Output Shape:** 144×1
- **Parameter Calculation:** 0

Layer 6: Fully Connected Layer (Output/Dense Layer)

- **Description:** Computes the final classification scores for 10 target classes.
- **Hyperparameters:** Input Neurons (N_{in}) = 144, Output Neurons (N_{out}) = 10
- **Output Shape:** 10×1
- **Parameter Calculation:**

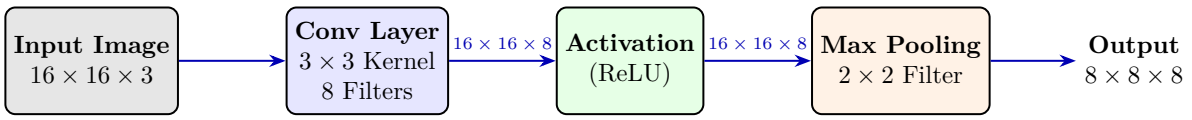
$$\text{Params} = (144 + 1) \cdot 10 = 145 \cdot 10 = 1450$$

Architecture Summary Table

Block	Layer	Type	Hyperparameters	Output Dimensions	Parameters
0	**0**	Input	RGB Image	$16 \times 16 \times 3$	0
1	**1**	Conv1	$F = 3, S = 1, P = 1, K = 8$	$16 \times 16 \times 8$	$(3 \times 3 \times 3 + 1) \times 8 = 224$
	2	Pool1	$F = 2, S = 2, P = 0$	$8 \times 8 \times 8$	0
2	**3**	Conv2	$F = 3, S = 1, P = 0, K = 16$	$6 \times 6 \times 16$	$(3 \times 3 \times 8 + 1) \times 16 = 1,168$
	4	Pool2	$F = 2, S = 2, P = 0$	$3 \times 3 \times 16$	0
3	**5**	Flatten	N/A	144×1	0
4	**6**	Dense	10 Output Neurons	10×1	$(144 + 1) \times 10 = 1,450$
				Total Trainable Parameters	**2,842**

V. Graphical Representation of the First Block

Below is a visual representation of the first set of operations applying a 3×3 kernel, ReLU activation, and a 2×2 Max Pooling layer to our $16 \times 16 \times 3$ image.



VI. Discussion

- **Parameter Sharing (Weight Sharing):** In a fully connected network, every output neuron requires a unique weight mapping to every input neuron. In a CNN, the exact same filter weights are moved (convolved) across the entire input feature map. This drastically reduces the number of trainable parameters, mitigating the risk of overfitting and vastly improving memory efficiency.
- **Local Connectivity (Receptive Fields):** Unlike Dense layers where all neurons interact, CNN neurons only connect to a small, localized region of the input volume. This mimics the human visual cortex, forcing the network to learn low-level spatial features (edges, corners) in early layers, which are combined into high-level features (faces, objects) in deeper layers.
- **Translation Invariance and Equivariance:**

  - *Equivariance:* Because filters slide over the input, shifting a feature in the input image shifts the corresponding activation in the output feature map by the same amount.
  - *Invariance:* Pooling layers (like Max Pooling) provide local translation invariance. Small shifts in the input image do not alter the pooled output, making the network robust to exact spatial positioning of objects.

- **Computational Complexity:** While CNNs have fewer parameters than MLPs, their forward pass can be computationally heavy. The number of Floating-Point Operations (FLOPs) for a single convolutional layer is approximately:

$$O(H_{out} \cdot W_{out} \cdot C_{out} \cdot C_{in} \cdot F^2)$$

  This highlights why deep CNNs require significant GPU acceleration during training.

- **Memory Requirements:** While weights are minimal, storing the *activations* (the intermediate feature maps outputted by each layer) requires immense memory. These activations must be retained in memory during the forward pass so they can be utilized for computing gradients during the backward pass via the chain rule.

Prepared By:

Md. Atikuzzaman

Lecturer

Department of Computer Science and Engineering

Green University of Bangladesh

Email: atik@cse.green.edu.bd