Fundamentals of Convolutional Neural Networks

Idant Srivastava

December 2025

Contents

Introduction

Convolutional Neural Network (CNN) is an advanced version of artificial neural networks (ANNs), primarily designed to extract features from grid-like matrix datasets. This is particularly useful for visual datasets such as images or videos, where data patterns play a crucial role. CNNs are widely used in computer vision applications due to their effectiveness in processing visual data. The key innovation lies in the convolution operation, which allows the network to learn hierarchical feature representations while maintaining spatial relationships and dramatically reducing the number of parameters compared to fully connected architectures.

Chapter 1

The Convolution Operation

1.1 Mathematical Foundation

The convolution operation is the cornerstone of CNNs. In the discrete 2D case (as used for images), convolution between an input matrix I and a kernel K is defined as:

$$S(i,j) = (I * K)(i,j) = \sum_{m} \sum_{n} I(i+m, j+n) \cdot K(m,n) \tag{1.1}$$

where S is the output feature map, I is the input, and K is the kernel or filter. The indices m and n iterate over the dimensions of the kernel.

1.2 Intuitive Understanding

Rather than viewing convolution as a complex mathematical operation, we can think of it as a sliding window that examines local regions of the input. The kernel slides across the input, and at each position, it performs element-wise multiplication with the overlapping input values, then sums the results to produce a single output value.

Consider a simple example: detecting vertical edges in an image. A vertical edge detector kernel might look like:

$$K = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{1.2}$$

When this kernel slides over an image region with a strong vertical transition (dark to light from left to right), the result will be a large positive value, effectively detecting the edge.

Chapter 2

Filters and Kernels

2.1 Definition and Purpose

The terms ”filter” and ”kernel” are often used interchangeably in the CNN. A kernel is a small matrix of learnable weights that is convolved with the input. During training, the network learns optimal kernel values for the task at hand, rather than using hand-crafted filters like traditional computer vision methods.

2.2 Multiple Filters

A convolutional layer typically uses multiple filters, each learning to detect different features. For instance, in early layers of a CNN processing images, different filters might learn to detect:

- Horizontal edges
- Vertical edges
- Diagonal patterns
- Color gradients
- Textures

Each filter produces its own feature map, and these are stacked together to form the layer's output volume.

2.3 Parameter Sharing

One of the key advantages of CNNs is parameter sharing. The same filter is applied across the entire input, meaning a filter that learns to detect a vertical edge at one location can detect it anywhere in the image. This reduces the number of parameters compared to fully connected layers and provides translation invariance.

Chapter 3

Feature Maps

3.1 Concept

A feature map is the output produced when a filter is convolved with the input (or the output of a previous layer). If we have n filters in a convolutional layer, we produce n feature maps, creating a 3D output volume with dimensions: height $\times$ width $\times$ number of filters.

3.2 Hierarchical Feature Learning

CNNs learn hierarchical representations through multiple layers:

- **Early layers**: Detect low-level features like edges, corners, and simple textures
- **Middle layers**: Combine low-level features into more complex patterns like shapes and object parts
- **Deep layers**: Recognize high-level concepts like complete objects, faces, or scenes

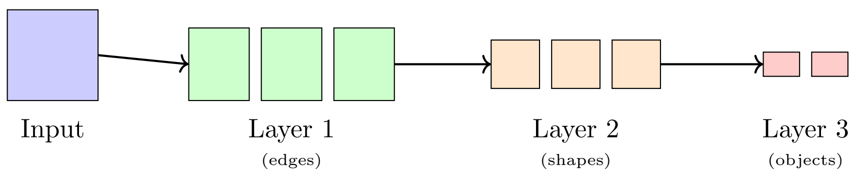

Figure 3.1: Hierarchical feature learning in CNNs: from simple to complex features

Chapter 4

Padding

4.1 The Problem

When we apply convolution without any modifications, the dimensions of the output are smaller than the input. For an input of size $n \times n$ and a kernel of size $k \times k$, the output size is $(n-k+1) \times (n-k+1)$. This becomes problematic in deep networks and loses information at the borders.

4.2 Valid Padding

Valid padding means no padding at all. The kernel only slides over positions where it completely overlaps with the input. Output size formula:

$$\text{Output size} = \left\lfloor \frac{n-k}{s} + 1 \right\rfloor \tag{4.1}$$

where n is input size, k is kernel size, and s is stride.

4.3 Same Padding

Same padding adds zeros around the input border so that the output has the same spatial dimensions as the input (when stride = 1). The padding amount p is calculated as:

$$p = \left\lfloor \frac{k-1}{2} \right\rfloor \tag{4.2}$$

4.4 Benefits of Padding

Padding provides several advantages: it prevents the dimensions from shrinking too quickly, preserves information at the borders of the image (which would otherwise be underutilized), and allows the construction of deeper networks without losing spatial resolution early.

Chapter 5

Strides

5.1 Definition

Stride is the number of pixels the filter (kernel) jumps over the input image (or feature map) at each step, determining how far it moves horizontally and vertically. A stride of 1 moves one pixel at a time, while a stride of 2 moves two pixels, and so on, effectively controlling the output's spatial dimensions (downsampling) and computational load, making larger strides useful for reducing size and increasing efficiency.

5.2 Impact on Output Size

The output size with stride s is calculated as:

$$\text{Output size} = \left\lfloor \frac{n + 2p - k}{s} + 1 \right\rfloor \tag{5.1}$$

where n is input size, p is padding, k is kernel size, and s is stride.

5.2.1 Computational Trade-offs

Larger strides reduce computation and memory requirements by producing smaller feature maps. However, they may lose fine-grained spatial information. Stride-2 convolutions are sometimes used as an alternative to pooling for downsampling.

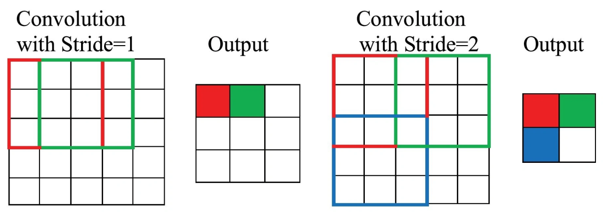

Figure 5.1: Strides

Chapter 6

Pooling Operations

6.1 Purpose

Pooling layers reduce the spatial dimensions of feature maps, providing several benefits: reducing computational complexity, decreasing the number of parameters (helping prevent overfitting), and introducing a degree of translation invariance. Pooling creates a more compact representation while retaining the most important information.

6.2 Max Pooling

Max pooling takes the maximum value from each region in the feature map. For a 2×2 pooling window with stride 2:

$$\text{Output}(i, j) = \max_{m,n \in \text{window}} \text{Input}(i \cdot s + m, j \cdot s + n) \tag{6.1}$$

Max pooling preserves the strongest activations, effectively identifying the presence of a feature in that region regardless of its exact position.

6.3 Average Pooling

Average pooling computes the average value in each region:

$$\text{Output}(i, j) = \frac{1}{k^2} \sum_{m,n \in \text{window}} \text{Input}(i \cdot s + m, j \cdot s + n) \tag{6.2}$$

where k is the pooling window size.

6.4 Comparison

Max pooling is more commonly used in CNNs because it better preserves important features and provides better performance in most computer vision tasks. Average pooling is sometimes used in the final layers of classification networks (global average pooling) to reduce spatial dimensions to a single value per channel.

Chapter 7

Convolutions Over Volumes

7.1 Multi-Channel Inputs

In the context of Convolutional Neural Networks (CNNs), input data typically possesses a volumetric structure, defined by dimensions $Height \times Width \times Depth$. For instance, processing an RGB image with depth $D = 3$ requires filters of dimensions $k \times k \times 3$. During the forward pass, the filter convolved spatially across the input computes the dot product across all channels simultaneously.
This operation aggregates the volumetric information at each spatial location into a single scalar value, resulting in a 2D activation map. When multiple filters are employed, their respective maps are stacked to produce a new output volume, where the depth corresponds to the number of filters.

7.2 Multiple Filters on Volumes

When we apply multiple filters to a volume, each filter produces one feature map, and these are stacked to create the output volume. For f filters:

$$\text{Input: } n_{\text{in}} \times n_{\text{in}} \times c_{\text{in}} \rightarrow \text{Output: } n_{\text{out}} \times n_{\text{out}} \times f \tag{7.1}$$

where the output height and width depend on padding, stride, and kernel size.

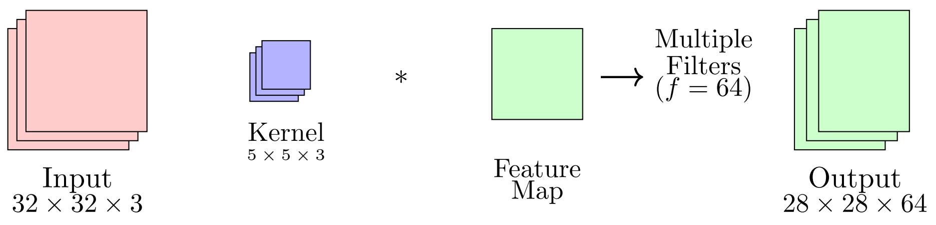

Figure 7.1: Convolution over volumes: from multi-channel input to multi-channel output

7.3 Parameter Count

For a convolutional layer with input depth c_{in}, kernel size $k \times k$, and f filters, the number of parameters is:

$$\text{Parameters} = (k \times k \times c_{\text{in}} + 1) \times f \tag{7.2}$$

The "+1" accounts for the bias term for each filter. This parameter sharing makes CNNs far more efficient than fully connected networks.

Conclusion

Convolutional Neural Networks leverage several key concepts to efficiently process spatial data. The convolution operation with learnable filters enables automatic feature extraction, while padding and stride control spatial dimensions. Pooling layers provide dimensionality reduction and translation invariance. The ability to process multi-channel volumes allows CNNs to handle complex real-world inputs like color images.

These fundamental components work together to create powerful architectures capable of learning hierarchical representations from raw pixels to high-level concepts. Understanding these building blocks is essential for designing, implementing, and debugging modern CNN architectures for various computer vision tasks.

The beauty of CNNs lies in their simplicity and effectiveness: through repeated application of these basic operations, stacked in deep architectures, they can learn to solve remarkably complex visual recognition tasks that were previously thought to require extensive hand-crafted feature engineering.