Fundamentals of Convolutional Neural Networks

Vaibhav Chourasia

January 6, 2026

1 Introduction and Motivation

Convolutional Neural Networks (CNNs) are designed specifically for processing image data. Images possess a strong spatial structure: neighboring pixels are highly correlated and together form meaningful visual patterns such as edges, textures, and shapes.

Traditional fully connected neural networks fail to exploit this structure because images must be flattened into vectors, destroying spatial relationships and leading to an explosion in the number of parameters. CNNs overcome these limitations by introducing convolution, which enables local connectivity and parameter sharing.

2 The Convolution Operation

2.1 Definition

Convolution is the core operation of a CNN. It involves applying a small matrix of learnable weights, called a **filter** or **kernel**, across an input image to extract local features.

The same filter is applied at every spatial location, allowing the network to detect the same pattern regardless of its position in the image.

2.2 Intuition Behind Convolution

To detect patterns in images, we need a way to compare a small region of the image with a predefined pattern. This comparison is achieved using the dot product.

If a local region closely matches the pattern encoded by the filter, the output value will be large; otherwise, it will be small. Sliding this comparison across the image allows detection of the pattern at different locations.

2.3 Derivation of the Convolution Formula

Consider a grayscale image represented as a matrix:

$$I \in \mathbb{R}^{H \times W}$$

Let the filter be a smaller matrix:

$$K \in \mathbb{R}^{f \times f}$$

At a given spatial position (i, j) in the image, the filter overlaps a local region (patch) of the image of size $f \times f$. To measure similarity between this image patch and the filter, we compute the dot product between corresponding elements.

First, we multiply each element of the image patch with the corresponding filter weight:

$$I(i + m, j + n) \cdot K(m, n)$$

for all $0 \leq m, n < f$.

Next, we sum all these products to obtain a single scalar value representing the strength of the match at position (i, j).

Thus, the convolution output at position (i, j) is derived as:

$$S(i, j) = \sum_{m=0}^{f-1} \sum_{n=0}^{f-1} I(i + m, j + n)\, K(m, n)$$

2.4 Final Convolution Formula

$$\boxed{S(i, j) = \sum_{m=0}^{f-1} \sum_{n=0}^{f-1} I(i + m, j + n)\, K(m, n)}$$

2.5 Explanation of the Formula

- $I(i + m, j + n)$ selects a pixel from the input image
- $K(m, n)$ selects the corresponding filter weight
- Multiplication measures local similarity
- Summation aggregates similarity across the entire patch

Repeating this operation across all valid spatial locations produces a two-dimensional output called a **feature map**.

2.6 Convolution vs Cross-Correlation

In strict mathematical terms, convolution requires the filter to be flipped before multiplication. In CNNs, this flip is omitted, and the operation performed is cross-correlation.

Since filters are learned during training, this distinction does not affect the representational power of the network, and the operation is commonly referred to as convolution.

2.7 Illustration of the Convolution Operation

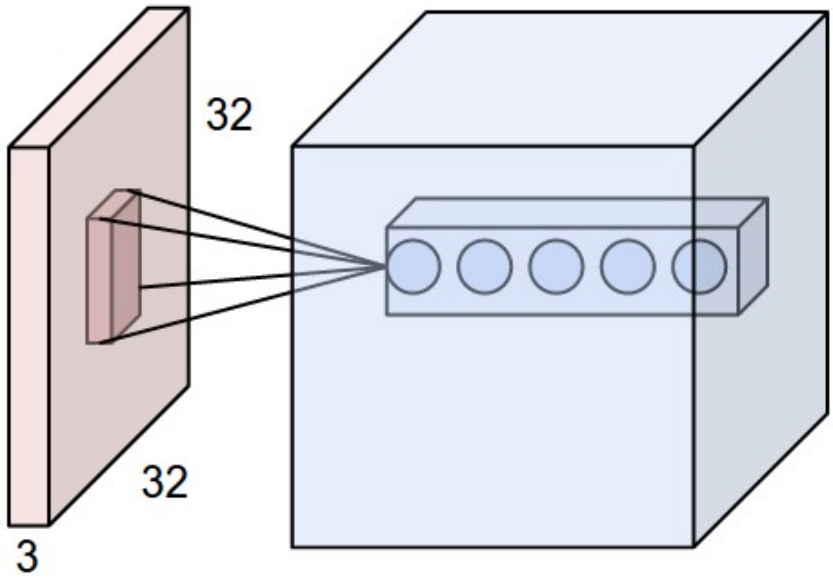

Figure 1: Illustration of the convolution operation using a sliding filter over an input image

3 Filters and Kernels

3.1 Definition

A **filter** (also called a **kernel**) is a small matrix of learnable parameters used in the convolution operation. Filters are designed to detect specific local patterns in images such as edges, corners, and textures.

Typical filter sizes are 3×3, 5×5, or 7×7.

3.2 Intuition Behind Filters

Each filter can be thought of as a template representing a particular visual pattern. When a filter slides over the image, it produces a strong response at locations where the local image structure closely resembles this template.

For example:

- Horizontal edge filters respond strongly to horizontal edges
- Vertical edge filters respond strongly to vertical edges

Unlike traditional image processing, CNNs do not manually define these filters; instead, the network learns them automatically during training.

3.3 Derivation: Why Filters Are Small

Consider an input image of size $H \times W$. If a filter were as large as the image, it would behave like a fully connected layer and lose spatial locality.

By restricting the filter size to $f \times f$ where $f \ll H, W$, each convolution operation focuses only on local neighborhoods. This enforces the assumption that meaningful visual patterns are local.

Thus, small filters allow:

- Local feature extraction
- Fewer parameters
- Better generalization

3.4 Parameter Sharing

A crucial property of filters is **parameter sharing**. The same filter weights are used at every spatial location.

If a filter has $f \times f$ parameters, the total number of parameters remains f^2, regardless of image size.

This is a major advantage over fully connected layers and dramatically reduces the risk of overfitting.

4 Feature Maps

4.1 Definition

A **feature map** is the output produced when a filter is convolved across the entire input image. It is a two-dimensional matrix where each value corresponds to the response of the filter at a particular spatial location.

4.2 Derivation of Feature Map Formation

At each spatial location (i, j), the convolution operation produces a scalar value:

$$S(i, j)$$

Repeating this operation at all valid positions produces a grid of values:

$$S = \begin{bmatrix} S(1,1) & S(1,2) & \dots \\ S(2,1) & S(2,2) & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

This grid is called a feature map.

4.3 Interpretation of Feature Maps

Each feature map highlights the spatial locations where a specific feature is present in the image.

- High values indicate strong feature presence
- Low values indicate weak or no feature presence

Using multiple filters results in multiple feature maps, allowing the CNN to capture diverse patterns simultaneously.

4.4 Why Feature Maps Are Important

Feature maps transform raw pixel data into higher-level representations while preserving spatial information. As the network deepens, feature maps evolve from simple to complex features.

4.5 Illustration of Feature Maps

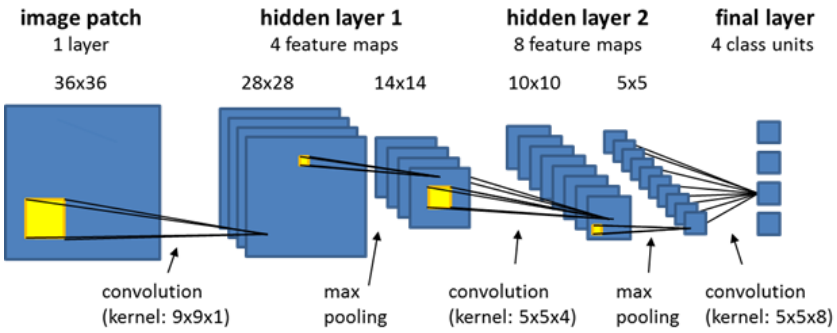

Figure 2: Feature maps generated in successive convolutional layers of a convolutional neural network, where different filters extract increasingly abstract spatial features

5 Padding

5.1 Why Padding Is Required

When a filter is applied to an image without padding, it can only be placed where it fully fits inside the image. As a result, the spatial dimensions of the output feature map become smaller than the input.

Repeated convolution operations cause rapid shrinking of feature maps and lead to loss of information near image boundaries. Padding is introduced to prevent this issue.

5.2 Derivation of Output Size Without Padding (VALID)

Consider an input image of height H and a filter of size f.

The filter can be placed starting from the top-most valid position and shifted downward until it no longer fits completely inside the image.

The number of valid vertical positions is:

$$H - f + 1$$

Similarly, the number of valid horizontal positions is:

$$W - f + 1$$

Thus, without padding and with stride 1, the output feature map has dimensions:

$$(H - f + 1) \times (W - f + 1)$$

This configuration is called **VALID padding**.

5.3 Zero Padding

To preserve spatial dimensions, zeros are added around the border of the image.

If P rows of zeros are added to the top and bottom, and P columns to the left and right, the padded image size becomes:

$$(H + 2P) \times (W + 2P)$$

This allows the filter to be applied at more spatial locations.

5.4 Derivation of SAME Padding

For SAME padding, the goal is to keep the output size equal to the input size.

Starting from the output size formula:

$$H_{\text{out}} = H + 2P - f + 1$$

We want:

$$H_{\text{out}} = H$$

Substituting:

$$H = H + 2P - f + 1$$

Solving for P:

$$2P = f - 1 \quad \Rightarrow \quad P = \frac{f-1}{2}$$

This derivation shows why SAME padding typically requires an odd-sized filter.

5.5 Illustration of Padding

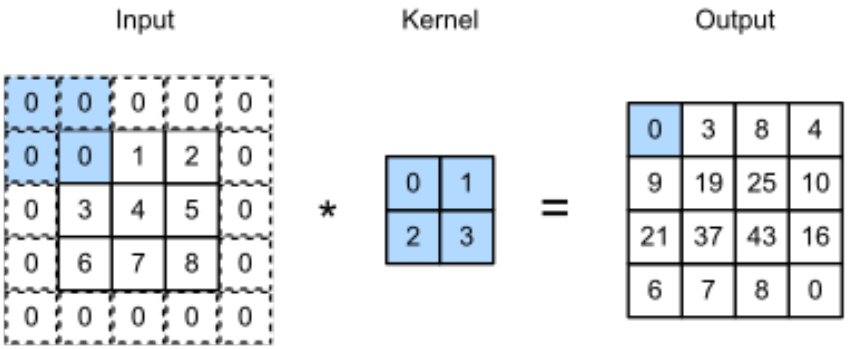

Figure 3: Illustration of zero padding applied to an input image, showing how padding preserves spatial dimensions and enables convolution at image boundaries

6 Strides

6.1 Definition

The stride determines how many pixels the filter shifts at each step during convolution. A stride of 1 moves the filter one pixel at a time, while larger strides skip pixels.

6.2 Derivation of Output Size with Stride

When stride S is introduced, the filter moves S pixels at a time instead of one.

The number of positions the filter can take vertically is:

$$\left\lfloor \frac{H + 2P - f}{S} \right\rfloor + 1$$

The same applies horizontally.

Thus, the general output size formula becomes:

$$H_{\text{out}} = \left\lfloor \frac{H + 2P - f}{S} \right\rfloor + 1$$

6.3 Explanation of the Formula

- H is the input height
- P is the padding size
- f is the filter size
- S is the stride

Larger strides reduce the number of convolution positions, resulting in smaller feature maps and reduced computational cost.

6.4 Illustration of Stride

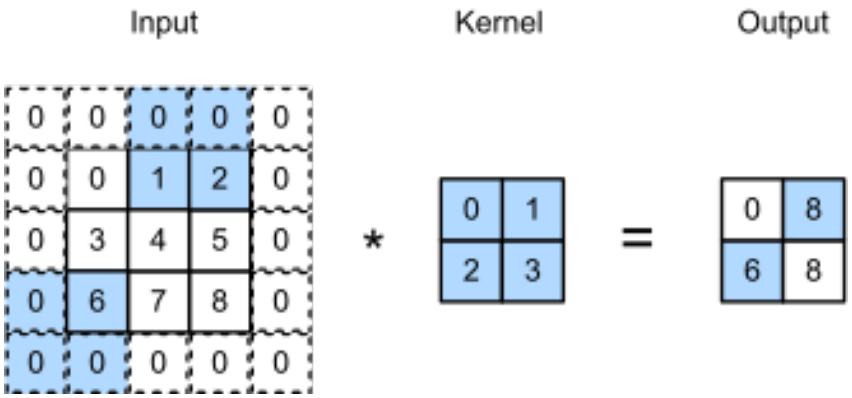

Figure 4: Effect of stride on convolution, where increasing the stride causes the filter to skip input locations and reduces the spatial size of the output feature map

7 Pooling

7.1 Why Pooling Is Introduced

After several convolution layers, feature maps may become large and sensitive to small spatial changes in the input image. Pooling is introduced to reduce spatial dimensions while retaining the most important information.

Pooling also helps the network become more robust to small translations and distortions.

7.2 Derivation of Pooling Operation

Consider a local pooling window of size $p \times p$ applied to a feature map. Let the values inside this window be:

$$\{x_1, x_2, \ldots, x_{p^2}\}$$

Pooling replaces this window with a single representative value.

7.3 Max Pooling

In max pooling, the representative value is chosen as the maximum value in the pooling window:

$$y = \max(x_1, x_2, \ldots, x_{p^2})$$

Explanation: Max pooling preserves the strongest activation, which usually corresponds to the most prominent feature in that region.

7.4 Average Pooling

In average pooling, the representative value is the mean of the pooling window:

$$y = \frac{1}{p^2} \sum_{i=1}^{p^2} x_i$$

Explanation: Average pooling captures the average presence of a feature rather than its strongest response.

7.5 Effect of Pooling on Dimensions

If pooling is applied with window size $p \times p$ and stride p, the spatial dimensions of the feature map are reduced by a factor of p in each direction.

7.6 Illustration of Pooling

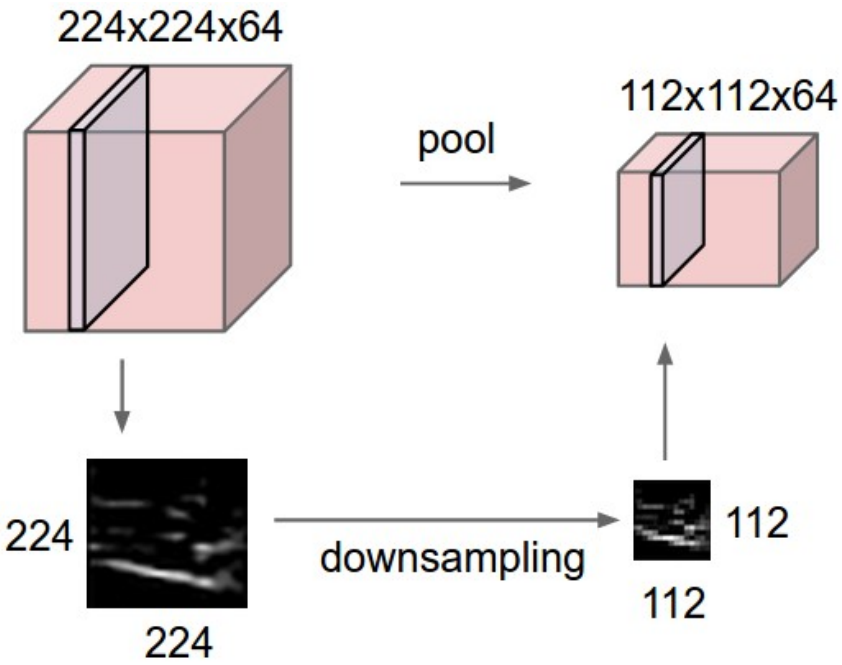

Figure 5: Illustration of pooling operation showing spatial downsampling while preserving the number of feature maps

8 Convolutions Over Volumes

8.1 Why Convolution Must Operate Over Volumes

Real-world images are color images with multiple channels (e.g., RGB). Such images are represented as 3D volumes:

$$I \in \mathbb{R}^{H \times W \times C}$$

where C is the number of channels.

Convolution must therefore operate across both spatial dimensions and depth.

8.2 Derivation of Volume Convolution

To process a multi-channel input, the filter must also extend across the full depth of the input:

$$K \in \mathbb{R}^{f \times f \times C}$$

At each spatial position (i, j), convolution computes the sum of dot products across all channels:

$$S(i,j) = \sum_{c=1}^{C} \sum_{m=0}^{f-1} \sum_{n=0}^{f-1} I(i+m, j+n, c)\, K(m, n, c)$$

This produces a single scalar output at position (i, j).

8.3 Output Volume

Applying one filter produces one feature map. Using N different filters produces an output volume of size:

$$\mathbb{R}^{H' \times W' \times N}$$

Key insight: The number of output channels equals the number of filters.

8.4 Illustration of Convolution Over Volumes

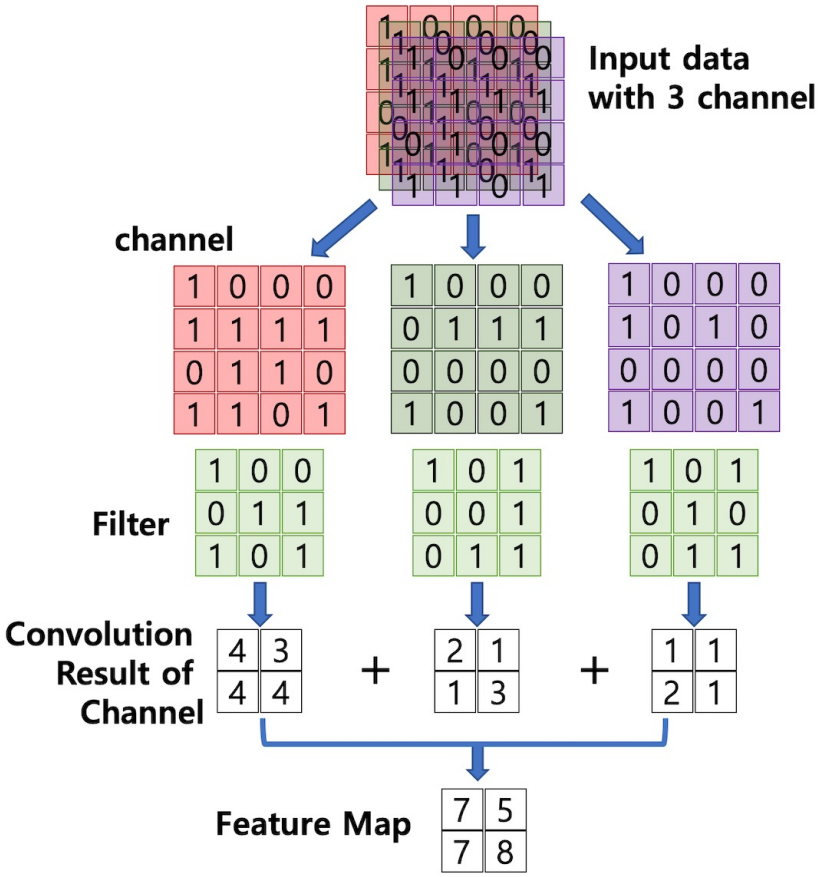

Figure 6: Convolution over volumes illustrating multi-channel input, depth-wise convolution, and summation across channels to produce a single feature map

9 Architectural Intuition of CNNs

9.1 Hierarchical Feature Learning

CNNs learn visual features hierarchically:

- Early layers detect simple patterns such as edges
- Middle layers detect shapes and textures
- Deeper layers detect object parts and complete objects

Each layer builds upon the representations learned by the previous layer.

9.2 Why CNNs Are Effective

CNNs are effective because they:

- Exploit spatial locality
- Share parameters across the image
- Learn increasingly abstract representations

These properties make CNNs computationally efficient and well-suited for image recognition tasks.

10 Conclusion

This report presented a detailedexplanation of the fundamental components of Convolutional Neural Networks. By deriving each operation from basic principles and explaining their intuition, CNNs emerge as a natural and powerful architecture for visual data processing.