

Module – 3

Convolution neural networks

Chapter 9

The Convolution Operation

Motivation

Pooling

Convolution and Pooling as an Infinitely Strong Prior

Variants of the Basic Convolution Function

Structured Outputs

Data Types

Efficient Convolution Algorithms

Convolutional Networks and the History of Deep Learning

Convolutional Neural Networks :

A Convolutional Neural Network (CNN) is a type of neural network that uses a mathematical operation called convolution instead of general matrix multiplication in at least one layer. It's designed to automatically and adaptively learn spatial hierarchies of features from input data.

Use of CNNs:

CNNs are ideal for data with grid-like topology:

1D: Time-series (e.g., ECG, speech)

2D: Images (e.g., handwritten digits, faces)

3D: Videos or volumetric scans

Convolutional Neural Networks exploit three key ideas:

Local connectivity: Neurons connect only to nearby inputs.

Parameter sharing: Same filter (kernel) slides across the input.

Translation invariance: Features are detected regardless of position.

9.1 The Convolution Operation

Convolution is a mathematical operation that blends two functions to produce a third. In deep learning, it's used to extract features from input data—especially images, audio, and time series.

Example: Tracking a Spaceship

Imagine you are tracking a spaceship's position over time using a noisy laser sensor:

$x(t)$: Position reading at time t

$w(a)$: Weighting function that gives more importance to recent readings

To smooth the noisy signal, compute a weighted average of past readings:

(i) Basic Convolution definition:

$$S(t) = \int x(a) w(t-a) da.$$

Where,

$S(t)$ is the input,

$X(a)$ is the signal(position measurement),

$W(t - a)$ is the kernel(weighting function)

(ii) Compact Notation:

$$S(t) = (x * w)(t).$$

convolution using the $*$ operator.

(iii) 1D Discrete Convolution

$$S(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a) w(t-a)$$

The **discrete version** of convolution where summation replaces the integral.

Formula slides one signal (kernel/filter) over the other.

(iv) 2D Convolution for Images

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) K(i-m, j-n).$$

Let:

- I be the **input image**
- K be the **kernel/filter**
- $s(i,j)$ be the output at position (i,j)
- Represents **convolution with a flipped kernel**.
- Traditional convolution flips the kernel both horizontally and vertically.

Variants of 2D Convolution

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n K(m,n) I(i-m, j-n)$$

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n)$$

The roles of I and K.

Uses kernel flipping.

The order of multiplication is swapped.

(v) Cross correlation, not convolution:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n).$$

9.2 Motivation

Convolution leverages three important ideas that can help improve a machine learning system: **sparse interactions**, **parameter sharing** and **equivariant representations**.

(i) Sparse Interactions (Sparse Connectivity)

A CNN achieves sparsity by making the **kernel smaller than the input**. Each output unit (a pixel in the feature map) only depends on a small, localized region of the input.

(ii) Parameter Sharing (Tied Weights)

The **same kernel (set of parameters)** is used to compute features across **every location** of the input feature map. The weights are **tied**.

(iii) Equivariance to Translation

Parameter sharing makes the network naturally **equivariant** to translation. the resulting feature map will contain the exact same feature map, but shifted by the same amount.

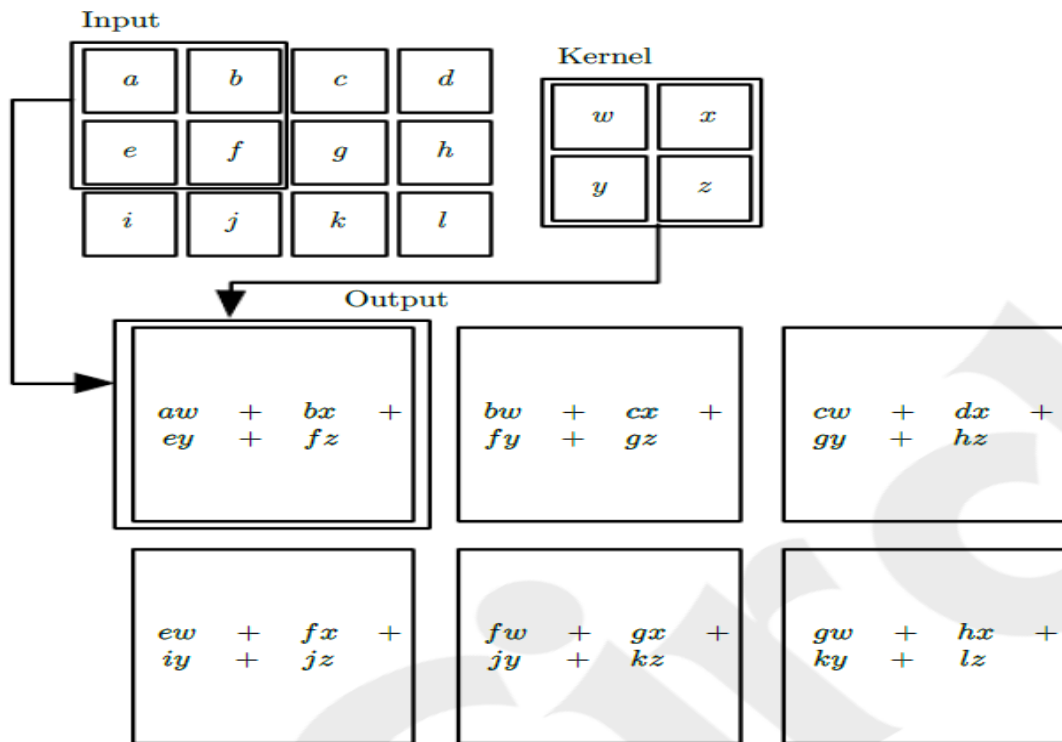


Figure 9.1: An example of 2-D convolution without kernel flipping. We restrict the output to only positions where the kernel lies entirely within the image, called “valid” convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.

Input

$$17 \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

Kernel

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

i) Shift kernel one column to the right

Output

$$\begin{bmatrix} a_{12}w_{11} + a_{13}w_{12} + a_{14}w_{13} \\ a_{22}w_{21} + a_{23}w_{22} + a_{24}w_{23} \\ a_{32}w_{31} + a_{33}w_{32} + a_{34}w_{33} \end{bmatrix}$$

ii) Shift kernel one row down

Output

$$\begin{bmatrix} a_{21}w_{11} + a_{22}w_{12} + a_{23}w_{13} \\ a_{31}w_{21} + a_{32}w_{22} + a_{33}w_{23} \\ a_{41}w_{31} + a_{42}w_{32} + a_{43}w_{33} \end{bmatrix}$$

iii) Shift kernel one column to the left

Output

$$\begin{bmatrix} a_{11}w_{11} + a_{12}w_{12} + a_{13}w_{13} \\ a_{21}w_{21} + a_{22}w_{22} + a_{23}w_{23} \\ a_{31}w_{31} + a_{32}w_{32} + a_{33}w_{33} \end{bmatrix}$$

iv) Shift kernel one row and one column

output

$$\begin{bmatrix} a_{22}w_{11} + a_{23}w_{12} + a_{24}w_{13} \\ a_{32}w_{21} + a_{33}w_{22} + a_{34}w_{23} \\ a_{42}w_{31} + a_{43}w_{32} + a_{44}w_{33} \end{bmatrix}$$

9.3 Pooling

A pooling function is applied to the output of the detector stage. It replaces a small spatial neighbourhood of the output with a summary statistic (e.g., the maximum or average value).

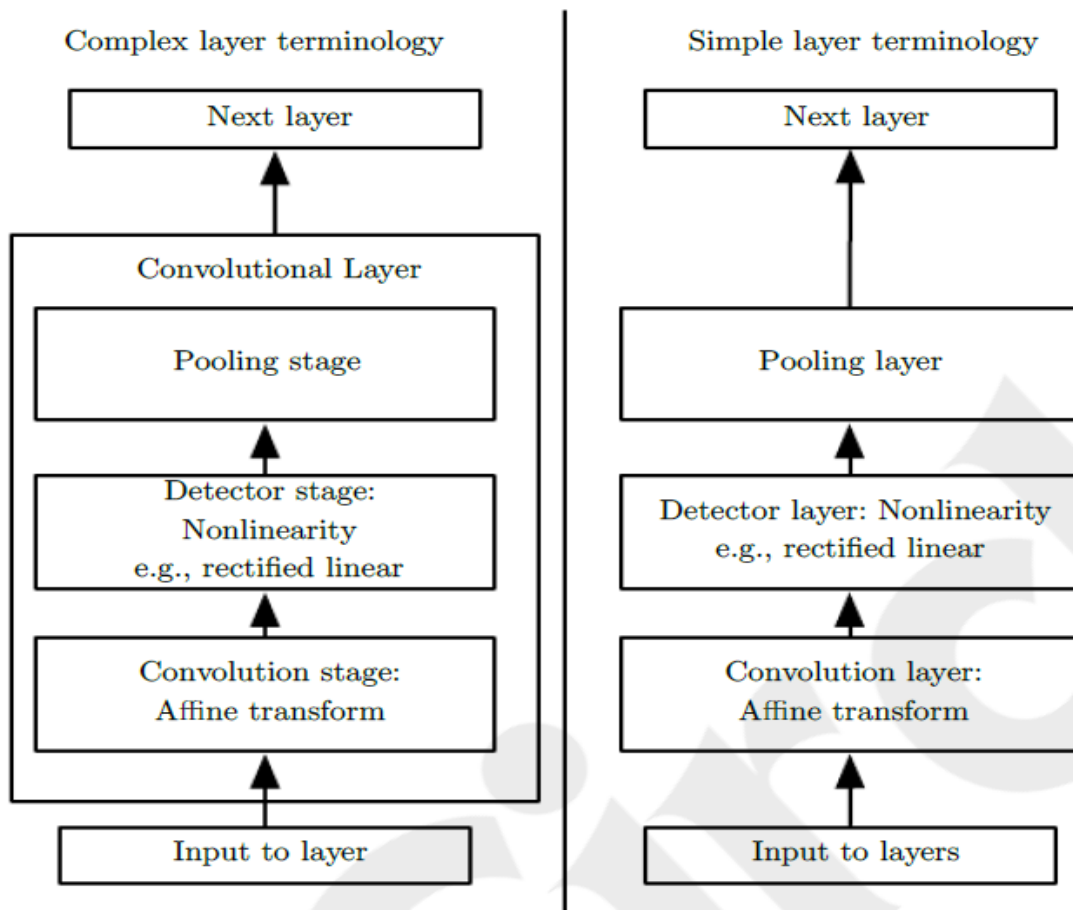


Figure 9.7: The components of a typical convolutional neural network layer. There are two commonly used sets of terminology for describing these layers. *(Left)* In this terminology, the convolutional net is viewed as a small number of relatively complex layers, with each layer having many “stages.” In this terminology, there is a one-to-one mapping between kernel tensors and network layers. In this book we generally use this terminology. *(Right)* In this terminology, the convolutional net is viewed as a larger number of simple layers; every step of processing is regarded as a layer in its own right. This means that not every “layer” has parameters.

The Convolution Neural Network Layer – Three Essential Stages

The three key stages of a Convolutional Neural Network (ConvNet) layer. The stages are:

1. **Convolution** – to extract feature maps from input data.
2. **Detection (ReLU)** – to apply non-linearity and enable complex pattern learning.
3. **Pooling** – to reduce spatial dimensions and focus on important features.

The emphasis is on understanding the sequence: Convolution → Detector (ReLU) → Pooling.

Stages 1 & 2 – Convolution and Detection

This slide explains the first two stages in more detail:

- **Convolution** involves applying filters to the input data to create linear feature maps.
- **Detection** uses a non-linear activation function, typically ReLU (Rectified Linear Unit), to allow the model to capture complex patterns beyond just linear ones.

Stage 3 – The Pooling Function

This slide covers the final stage of the ConvNet layer:

- **Pooling** replaces small regions of the feature map with summary statistics, like maximum or average values.
- The **goal** of pooling is to achieve **translation invariance**, meaning the model can recognize patterns regardless of their position, and **dimensionality reduction**, which simplifies computations by reducing the size of the data.

Max Pooling – The Primary Method

- Max Pooling works by selecting the **maximum value** from a group of values within a defined window (like 2x2 or 3x3).
- The main idea is to retain the most prominent feature detected in that region, while **discarding the exact location information**. This helps in reducing the spatial dimensions of the data and makes the network more robust to small shifts or distortions in the input.

Alternative Pooling Methods

Besides Max Pooling, other pooling techniques include:

- **Average Pooling**, which calculates the average value within the window instead of the maximum.
- **L² Norm Pooling**, which uses the square root of the sum of squares of the values in the window as the output.

9.6 Structured Outputs

In the traditional use of Convolutional Neural Networks (CNNs), you input an image and get a single class label as output, such as “dog” or “cat.” The output in such cases is either a vector (for classification tasks) or a scalar (for regression tasks).

However, in the case of structured outputs, instead of assigning one label to the whole image, you want the CNN to label each pixel in the image individually. This means the network produces a tensor (a multi-dimensional array) as the output.

For each pixel at position (i, j) , the network computes the probability that this pixel belongs to a particular class i . Mathematically, this can be represented as:

S_{ij} = Probability that pixel (i, j) belongs to class i

This approach allows the CNN to segment an image pixel by pixel, enabling it to draw detailed masks or identify object boundaries precisely.

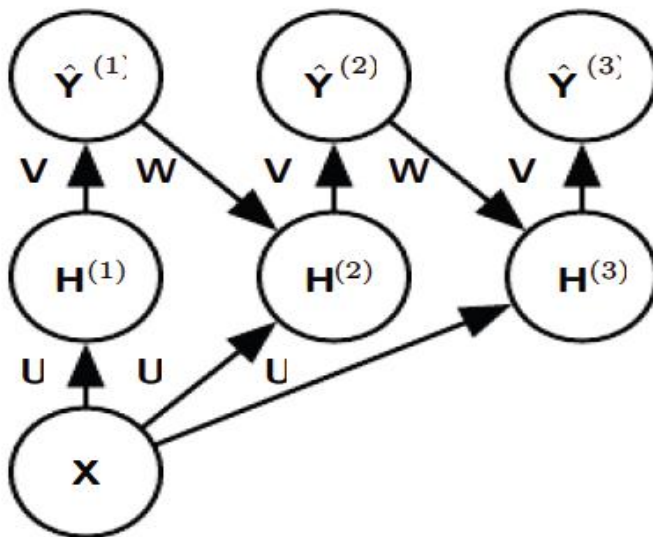


Figure 9.17: An example of a recurrent convolutional network for pixel labeling.

In a Recurrent Convolutional Network (RCNN), the structure combines the principles of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). It processes image data iteratively, refining its predictions over multiple steps.

The input image (for example, an RGB image) is represented as X .

The network generates pixel-wise label predictions at each step, denoted as $Y(t)$, and produces hidden feature maps $H(t)$ at each time step.

The convolution kernels U, V, and W are used as follows:

U extracts features from the input image X.

V generates label predictions from the hidden features.

W applies convolution to the previous label predictions to refine them.

Properties:

- I. Shared Weights: The same kernels U, V, and W are reused at each step.
- II. Recurrent Behaviour: Predictions from the previous step ($Y(t-1)$) are used as inputs for the next step to improve accuracy.
- III. First Step: Only the input image is used for the initial prediction.
- IV. Subsequent Steps: Both the input image and the previous prediction are used to refine results further.

9.7 Data Types

The data used with a convolutional network usually consist of several channels, each channel being the observation of a different quantity at some point in space or time.

Type	Single Channel	Multichannel
1-D Data	Example: Audio waveform. The input is a 1D signal that varies over time. Each point represents the amplitude of sound at a specific time step. CNNs convolve (slide) filters over time to learn patterns like pitch or rhythm.	Example: Skeleton animation data. Used in 3D animations where the pose of a skeleton changes over time. Each joint's angle (or axis of motion) forms a separate channel. So, each channel represents motion data for one joint over time.
2-D Data	Example: Audio spectrogram (Fourier-transformed audio). Audio is converted to a 2D tensor — one axis for frequency, one for time.	Example: Colour image data. Each image has three channels — red, green, and blue.

	Convolution along time and frequency helps detect patterns invariant to pitch shifts or timing differences.	Convolution kernels slide over both height and width of the image. CNNs learn spatial patterns (edges, textures, objects) across all channels.
3-D Data	Example: Volumetric data (e.g., medical CT or MRI scans). Data has three spatial dimensions — height, width, and depth. Used for analyzing 3D objects or body scans in medicine.	Example: Colour video data. Video frames are 3D (time $\times$ height $\times$ width). Each frame can also have multiple colour channels (RGB). Convolutions can capture spatial and temporal patterns, such as motion or object tracking.