Basics of Computer Vision

Kaiser Hamid

December 16, 2024

Contents

1 Introduction

Before delving into complex convolutional neural network (CNN) architectures in subsequent articles, it is essential to understand the fundamental operations that form the backbone of these networks. This article focuses on the basics of image operations such as **convolution**, **pooling**, **padding**, and **stride**.

While modern CNNs often work with RGB images (3-channel), here we will simplify to a single-channel (grayscale) image to illustrate core concepts. In later articles, we will extend these ideas to multi-channel inputs and more sophisticated architectures.

2 Representation of a Grayscale Image

A grayscale image can be represented as a two-dimensional array (matrix) of intensity values. For simplicity, consider an 8×8 grayscale image. Each element I_{ij} in this matrix can be an intensity value ranging from 0 (black) to 255 (white) for an 8-bit image, or a normalized value between 0 and 1.

For example, let our 8×8 image I be represented as:

$$I = \begin{bmatrix} i_{11} & i_{12} & i_{13} & i_{14} & i_{15} & i_{16} & i_{17} & i_{18} \\ i_{21} & i_{22} & i_{23} & i_{24} & i_{25} & i_{26} & i_{27} & i_{28} \\ i_{31} & i_{32} & i_{33} & i_{34} & i_{35} & i_{36} & i_{37} & i_{38} \\ i_{41} & i_{42} & i_{43} & i_{44} & i_{45} & i_{46} & i_{47} & i_{48} \\ i_{51} & i_{52} & i_{53} & i_{54} & i_{55} & i_{56} & i_{57} & i_{58} \\ i_{61} & i_{62} & i_{63} & i_{64} & i_{65} & i_{66} & i_{67} & i_{68} \\ i_{71} & i_{72} & i_{73} & i_{74} & i_{75} & i_{76} & i_{77} & i_{78} \\ i_{81} & i_{82} & i_{83} & i_{84} & i_{85} & i_{86} & i_{87} & i_{88} \end{bmatrix}$$

3 Convolution

3.1 Definition

Convolution in the context of image processing is an operation where a small matrix of weights, called a *kernel* or *filter*, is applied to the image to produce a transformed output. In CNNs, convolutional filters detect features such as edges, corners, or more complex patterns depending on the learned weights.

Mathematically, for an image I and a kernel K, the convolution $I * K$ at

a particular location (x, y) is given by:

$$(I * K)(x, y) = \sum_m \sum_n I(x + m, y + n) \cdot K(m, n)$$

where m and n iterate over the width and height of the kernel.

3.2 Example With an 8×8 Image

Consider a 3×3 kernel:

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

To compute the convolution at the top-left corner of the image (assuming no padding and unit stride), we overlay the kernel on the first 3×3 patch of I:

$$I_{top-left} = \begin{bmatrix} i_{11} & i_{12} & i_{13} \\ i_{21} & i_{22} & i_{23} \\ i_{31} & i_{32} & i_{33} \end{bmatrix}$$

Then the convolution result at $(1, 1)$ (top-left position of the output):

$$(I*K)(1,1) = i_{11}k_{11}+i_{12}k_{12}+i_{13}k_{13}+i_{21}k_{21}+i_{22}k_{22}+i_{23}k_{23}+i_{31}k_{31}+i_{32}k_{32}+i_{33}k_{33}.$$

We repeat this process by sliding the kernel across the image in both the horizontal and vertical directions to compute the full output of the convolution.

4 Padding

4.1 Definition

Padding refers to adding extra rows and/or columns around the image, typically filled with zeros (zero-padding). Padding helps to control the spatial size of the output after convolution. Without padding, every convolution reduces the image size. With sufficient padding, we can preserve the original spatial dimension of the image.

4.2 Example

If we have an 8×8 image and a 3×3 kernel, convolving without padding would produce a 6×6 output (since we lose one pixel on each border for every dimension). By adding one layer of zero-padding around the original image, we effectively have a 10×10 padded input, and convolving with a 3×3 kernel will yield an 8×8 output.

For example, with zero-padding of size 1:

$$Padded(I) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & i_{11} & i_{12} & i_{13} & i_{14} & i_{15} & i_{16} & i_{17} & i_{18} & 0 \\ 0 & i_{21} & i_{22} & i_{23} & i_{24} & i_{25} & i_{26} & i_{27} & i_{28} & 0 \\ 0 & i_{31} & i_{32} & i_{33} & i_{34} & i_{35} & i_{36} & i_{37} & i_{38} & 0 \\ 0 & i_{41} & i_{42} & i_{43} & i_{44} & i_{45} & i_{46} & i_{47} & i_{48} & 0 \\ 0 & i_{51} & i_{52} & i_{53} & i_{54} & i_{55} & i_{56} & i_{57} & i_{58} & 0 \\ 0 & i_{61} & i_{62} & i_{63} & i_{64} & i_{65} & i_{66} & i_{67} & i_{68} & 0 \\ 0 & i_{71} & i_{72} & i_{73} & i_{74} & i_{75} & i_{76} & i_{77} & i_{78} & 0 \\ 0 & i_{81} & i_{82} & i_{83} & i_{84} & i_{85} & i_{86} & i_{87} & i_{88} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

5 Stride

5.1 Definition

The **stride** is the amount by which we move the kernel across the image. A stride of 1 means the kernel moves one pixel at a time. A stride of 2 means the kernel moves two pixels at a time, effectively producing a smaller output.

5.2 Example

For the same 8×8 image and 3×3 kernel, if the stride is 2, then after placing the kernel at $(1, 1)$, the next position horizontally would be $(1, 3)$ instead of $(1, 2)$. This results in fewer total positions for convolution and a smaller output feature map.

6 Pooling

6.1 Definition

Pooling is a downsampling operation used to reduce the spatial dimensions of feature maps. Two common forms are *max pooling* and *average pooling*:

- **Max Pooling**: Takes the maximum value within a specified window.
- **Average Pooling**: Takes the average value within a specified window.

6.2 Example With a 2×2 Pooling Window

Suppose after some convolution operation, we have a feature map of size 8×8. Applying a 2×2 max pooling with stride 2 means we consider non-overlapping 2×2 blocks:

$$\begin{bmatrix} i_{11} & i_{12} \\ i_{21} & i_{22} \end{bmatrix}, \begin{bmatrix} i_{13} & i_{14} \\ i_{23} & i_{24} \end{bmatrix}, \ldots$$

For each 2×2 block, we pick the maximum (for max pooling) or compute the average (for average pooling). Thus, an 8×8 input would become a 4×4 output after 2×2 pooling with stride 2.

7 Summary

In this first article, we established the fundamental operations in convolutional neural networks:

- **Convolution** with a kernel detects features in an image.
- **Padding** allows us to control the spatial size of the output, often used to maintain image dimensions.
- **Stride** affects how much the kernel moves, influencing the output size.
- **Pooling** reduces spatial dimensions and helps in extracting abstracted features.

Future articles will build upon these concepts to discuss the construction of full CNN architectures and how multiple layers of convolution, pooling, and other operations come together to form powerful image recognition systems.