Convolutional Neural Networks

Convolution, Pooling, and Learning

Yasser Hifny

Faculty of Computers and Artificial Intelligence
Capital University

Contents

1. Motivation
2. Convolution
3. 1D Cross-Correlation
4. 2D Cross-Correlation
5. Convolution over Volumes
6. Resolution Control
7. CNN Model and Learning
8. Pooling Layers
9. LeNet-5 Example
10. Assignment
11. Summary

Motivation

Convolutional Neural Networks (CNNs) are deep learning models designed for structured grid data such as images and videos.

- Feed-forward networks treat image pixels as separate input features.
- CNNs exploit local spatial structure.
- The same filters are reused across different image locations.
- CNNs learn hierarchical representations directly from raw data.
- Pooling can reduce spatial dimensions and computational cost.

Central idea

CNNs replace a fully connected treatment of every pixel with local connectivity and parameter sharing.

Why Move from Feed-Forward Networks to CNNs?

Feed-Forward Networks

- Very high-dimensional input.
- Large number of parameters.
- No inherent spatial locality.
- Separate parameters for input features.
- Difficult to learn hierarchical image structure.

CNNs

- Local receptive fields.
- Parameter sharing.
- Recognition across spatial locations.
- Hierarchical feature learning.
- Pooling for dimensionality reduction.

For example, a 64×64 RGB image contains

$$64 \times 64 \times 3 = 12,288$$

input values.

Five Motivations for CNNs

1. **Handling high dimensionality:** filters process small local regions.
2. **Parameter sharing:** the same filter is applied across different locations.
3. **Translation invariance:** learned features can be detected at different positions.
4. **Hierarchical feature learning:** early layers learn edges and textures; deeper layers learn shapes and objects.
5. **Dimensionality reduction:** pooling reduces spatial resolution while retaining important information.

Convolution: Mathematical Definition

For continuous one-dimensional signals:

$$(x * w)(t) = \int_{-\infty}^{\infty} x(\tau) w(t - \tau)\, d\tau.$$

For discrete sequences:

$$(x * w)[n] = \sum_{k=-\infty}^{\infty} x[k] w[n - k].$$

Interpretation

At every position, values from the input and kernel are multiplied and summed to produce the output.

In CNNs, kernels are learned during training.

Two-Dimensional Convolution

For two-dimensional data:

$$s_{ij} = (x * w)[i, j] = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} x_{k,l} \, w_{i-k,j-l}.$$

The chapter also notes the commutative form:

$$s_{ij} = (w * x)[i, j].$$

In practical deep-learning frameworks, the operation called “convolution” is generally implemented as cross-correlation, without flipping the kernel.

Cross-Correlation in CNNs

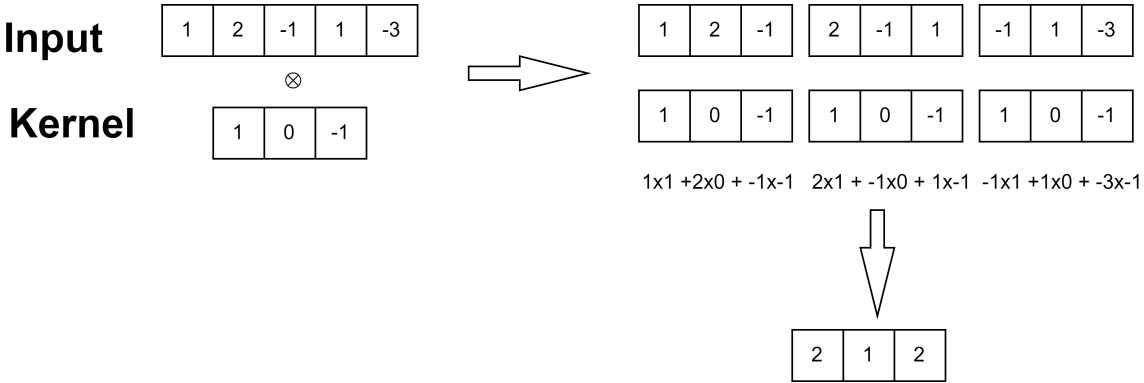
The practical operation is written as

$$s_{ij} = (x \otimes w)[i, j] = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} x_{i+k,j+l} w_{k,l}$$

- Slide the kernel over the input.
- Multiply corresponding elements.
- Sum the products.
- Store the result at the current output position.

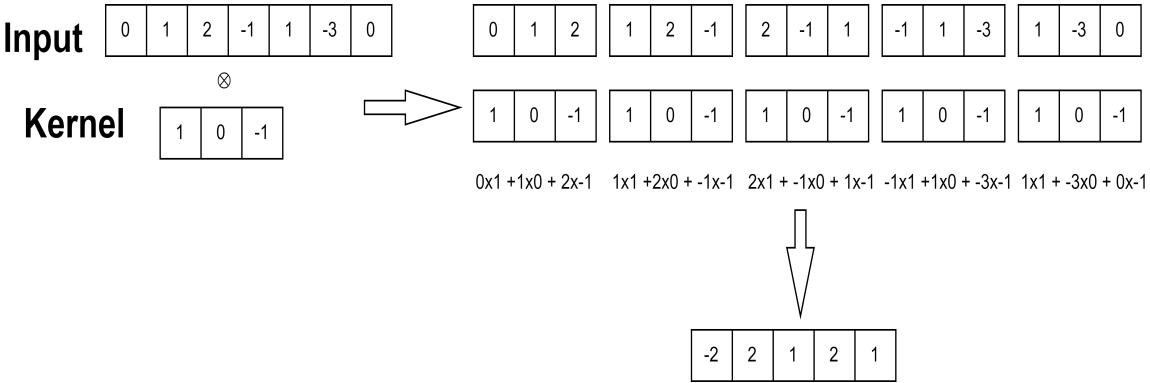
Because the kernel parameters are learned, the use of cross-correlation rather than a flipped mathematical convolution does not prevent the network from learning useful filters.

1D Cross-Correlation: Basic Example



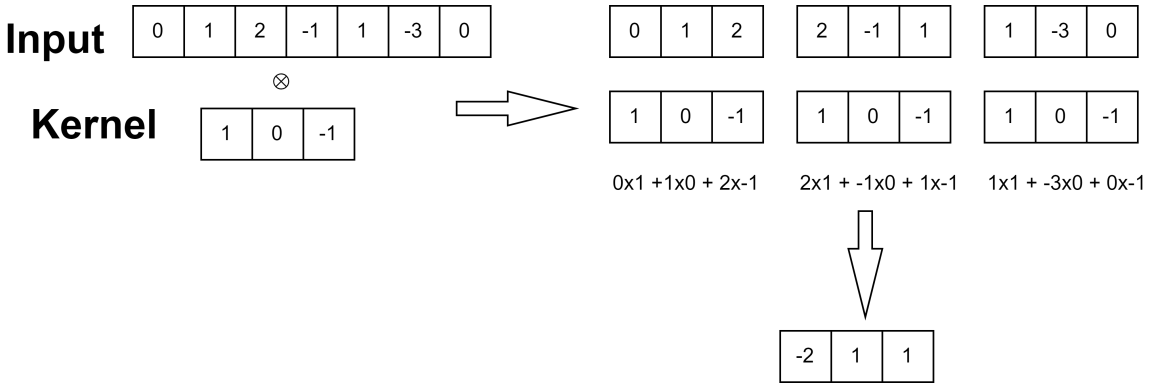
A one-dimensional input vector with a three-element kernel.

1D Cross-Correlation: Padding



Zero-padding is added at the borders so that the output can preserve the input size.

1D Cross-Correlation: Padding and Stride



Example with padding equal to 1 and stride equal to 2.

Padding and Stride

- **Padding** adds values around the input boundary.
- **Stride** determines how far the kernel moves at every step.
- Stride 1: the kernel moves one element at a time.
- Stride 2: the kernel moves two elements at a time.

For input size n, kernel size f, padding p, and stride s:

$$\boxed{o = \left\lfloor \frac{n + 2p - f}{s} \right\rfloor + 1}$$

2D Cross-Correlation

For image data, a kernel is slid over the two-dimensional input.

1. **Sliding window:** move the kernel over the image.
2. **Element-wise multiplication:** multiply corresponding input and kernel values.
3. **Summation:** add the products to produce one output value.

For an $n \times n$ input and $f \times f$ kernel:

$$o_1 \times o_2 = \left[\left\lfloor \frac{n + 2p - f}{s} \right\rfloor + 1 \right] \times \left[\left\lfloor \frac{n + 2p - f}{s} \right\rfloor + 1 \right].$$

2D Cross-Correlation: 6×6 Input

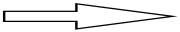
Input

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

$\otimes$

Kernel

1	0	-1
1	0	-1
1	0	-1



-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

A 6×6 input and a 3×3 kernel.

2D Cross-Correlation with Stride 2

Input

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

$\otimes$

Kernel

1	0	-1
1	0	-1
1	0	-1



-5	0
0	-4

The same input/kernel dimensions with stride 2.

2D Cross-Correlation Reduced to 1D

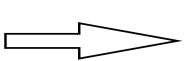
Input

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3

$\otimes$

Kernel

1	0	-1
1	0	-1
1	0	-1



-5	-4	0	8

When the kernel height equals the input height, the operation reduces to a 1D cross-correlation along the width dimension.

Cross-Correlation over a Volume

For a color image:

$$\text{Input} = 6 \times 6 \times 3$$

A filter must have the same number of channels:

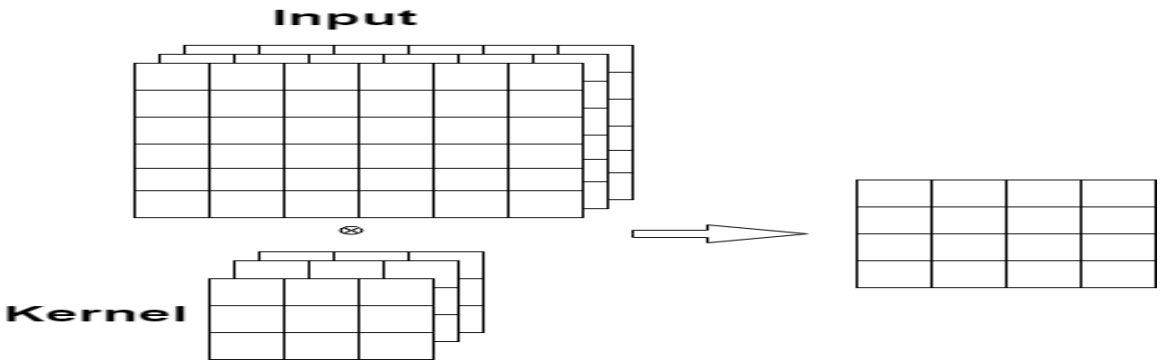
$$\text{Filter} = 3 \times 3 \times 3.$$

With stride 1 and no padding, the output is:

$$4 \times 4.$$

Each output value is obtained from the sum of the element-wise products over all three channels.

3D Cross-Correlation



A $6 \times 6 \times 3$ input processed by a $3 \times 3 \times 3$ filter.

Multiple Filters

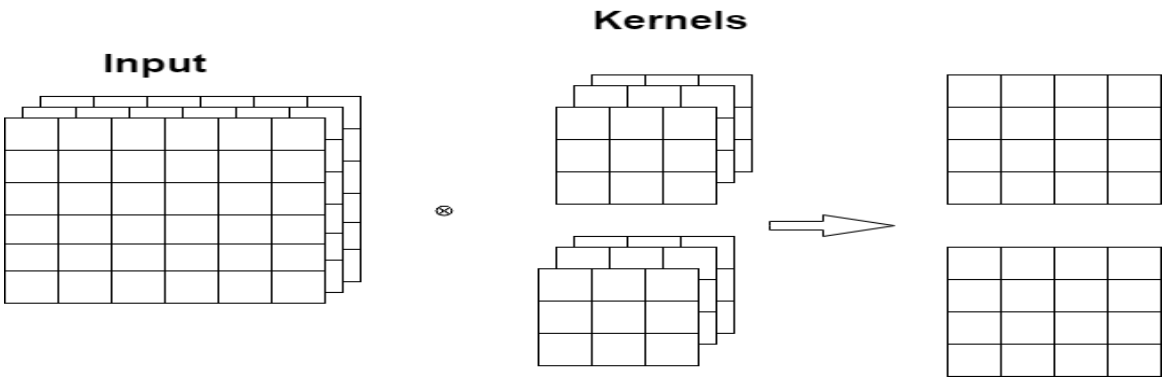
If multiple filters are applied simultaneously, each filter can learn a different feature.
For two filters:

$$6 \times 6 \times 3 \quad \longrightarrow \quad 4 \times 4 \times 2.$$

For example:

- one filter may learn vertical-edge patterns;
- another may learn horizontal-edge patterns.

Multiple Filters over a Volume



Multiple filters produce multiple output feature maps.

Resolution Control: Pooling

Max pooling and average pooling are down-sampling operations.

- Reduce spatial dimensions.
- Reduce computational cost.
- Reduce the number of values passed to subsequent layers.
- Provide some degree of robustness to spatial variation.

Max pooling

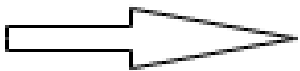
Selects the maximum value in each pooling window.

Average pooling

Computes the average value in each pooling window.

Max Pooling Example

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16



6	8
14	16

A 4×4 input with a 2×2 max-pooling window produces a 2×2 output.

Pooling Output Size

For input size n, pooling-window size f, padding p, and stride s:

$$\boxed{o = \left\lfloor \frac{n - f + 2p}{s} \right\rfloor + 1}$$

For $n = 4$, $f = 2$, $p = 0$, and $s = 2$:

$$o = \left\lfloor \frac{4 - 2}{2} \right\rfloor + 1 = 2.$$

Thus, the output is 2×2.

Convolutional Layer Backpropagation 1: Forward Pass

To compute the gradients in the convolutional layer, assume a single channel; the equations then simplify slightly. Single-channel input $a^{[l-1]}$ (3×3), kernel $w^{[l]}$ (2×2), output $z^{[l]}$ (2×2):

$$\begin{bmatrix} z_{00}^{[l]} & z_{01}^{[l]} \\ z_{10}^{[l]} & z_{11}^{[l]} \end{bmatrix} = \begin{bmatrix} a_{00}^{[l-1]} & a_{01}^{[l-1]} & a_{02}^{[l-1]} \\ a_{10}^{[l-1]} & a_{11}^{[l-1]} & a_{12}^{[l-1]} \\ a_{20}^{[l-1]} & a_{21}^{[l-1]} & a_{22}^{[l-1]} \end{bmatrix} \otimes \begin{bmatrix} w_{00}^{[l]} & w_{01}^{[l]} \\ w_{10}^{[l]} & w_{11}^{[l]} \end{bmatrix} + b^{[l]}$$

Here $\otimes$ denotes cross-correlation: $z^{[l]} = a^{[l-1]} \otimes w^{[l]} + b^{[l]}$.

Convolutional Layer Backpropagation 2: Forward Pass Expanded

$$
\begin{aligned}
z_{00}^{[l]} &= w_{00}^{[l]} a_{00}^{[l-1]} + w_{01}^{[l]} a_{01}^{[l-1]} + w_{10}^{[l]} a_{10}^{[l-1]} + w_{11}^{[l]} a_{11}^{[l-1]} + b^{[l]} \\
z_{01}^{[l]} &= w_{00}^{[l]} a_{01}^{[l-1]} + w_{01}^{[l]} a_{02}^{[l-1]} + w_{10}^{[l]} a_{11}^{[l-1]} + w_{11}^{[l]} a_{12}^{[l-1]} + b^{[l]} \\
z_{10}^{[l]} &= w_{00}^{[l]} a_{10}^{[l-1]} + w_{01}^{[l]} a_{11}^{[l-1]} + w_{10}^{[l]} a_{20}^{[l-1]} + w_{11}^{[l]} a_{21}^{[l-1]} + b^{[l]} \\
z_{11}^{[l]} &= w_{00}^{[l]} a_{11}^{[l-1]} + w_{01}^{[l]} a_{12}^{[l-1]} + w_{10}^{[l]} a_{21}^{[l-1]} + w_{11}^{[l]} a_{22}^{[l-1]} + b^{[l]}
\end{aligned}
$$

Each weight $w_{pq}^{[l]}$ touches every output $z_{ij}^{[l]}$ exactly once, through the input $a_{p+i,\,q+j}^{[l-1]}$. This is what the gradient formulas on the next slides exploit.

Convolutional Layer Backpropagation 3: Gradient w.r.t. Weights

By the chain rule, the gradient of the loss E with respect to each weight is:

$$\frac{\partial E}{\partial w_{00}^{[l]}} = \frac{\partial E}{\partial z_{00}^{[l]}} a_{00}^{[l-1]} + \frac{\partial E}{\partial z_{01}^{[l]}} a_{01}^{[l-1]} + \frac{\partial E}{\partial z_{10}^{[l]}} a_{10}^{[l-1]} + \frac{\partial E}{\partial z_{11}^{[l]}} a_{11}^{[l-1]}$$

$$\frac{\partial E}{\partial w_{01}^{[l]}} = \frac{\partial E}{\partial z_{00}^{[l]}} a_{01}^{[l-1]} + \frac{\partial E}{\partial z_{01}^{[l]}} a_{02}^{[l-1]} + \frac{\partial E}{\partial z_{10}^{[l]}} a_{11}^{[l-1]} + \frac{\partial E}{\partial z_{11}^{[l]}} a_{12}^{[l-1]}$$

$$\frac{\partial E}{\partial w_{10}^{[l]}} = \frac{\partial E}{\partial z_{00}^{[l]}} a_{10}^{[l-1]} + \frac{\partial E}{\partial z_{01}^{[l]}} a_{11}^{[l-1]} + \frac{\partial E}{\partial z_{10}^{[l]}} a_{20}^{[l-1]} + \frac{\partial E}{\partial z_{11}^{[l]}} a_{21}^{[l-1]}$$

$$\frac{\partial E}{\partial w_{11}^{[l]}} = \frac{\partial E}{\partial z_{00}^{[l]}} a_{11}^{[l-1]} + \frac{\partial E}{\partial z_{01}^{[l]}} a_{12}^{[l-1]} + \frac{\partial E}{\partial z_{10}^{[l]}} a_{21}^{[l-1]} + \frac{\partial E}{\partial z_{11}^{[l]}} a_{22}^{[l-1]}$$

Convolutional Layer Backpropagation 4: General Form and Bias

Assume $z^{[l]}$ is $m \times n$, and let $\delta_{i,j}^{[l]} = \frac{\partial E}{\partial z_{i,j}^{[l]}}$. Then

$$\frac{\partial E}{\partial w_{p,q}^{[l]}} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \delta_{i,j}^{[l]} \, a_{p+i,\, q+j}^{[l-1]}$$

In matrix form:

$$\underbrace{\begin{bmatrix} \frac{\partial E}{\partial w_{00}^{[l]}} & \frac{\partial E}{\partial w_{01}^{[l]}} \\ \frac{\partial E}{\partial w_{10}^{[l]}} & \frac{\partial E}{\partial w_{11}^{[l]}} \end{bmatrix}}_{\frac{\partial E}{\partial w^{[l]}}} = a^{[l-1]} \otimes \underbrace{\begin{bmatrix} \frac{\partial E}{\partial z_{00}^{[l]}} & \frac{\partial E}{\partial z_{01}^{[l]}} \\ \frac{\partial E}{\partial z_{10}^{[l]}} & \frac{\partial E}{\partial z_{11}^{[l]}} \end{bmatrix}}_{\delta^{[l]}}$$

The weight gradient is itself a cross-correlation of the input with the error term. For the bias:

$$\boxed{\frac{\partial E}{\partial b^{[l]}} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \delta_{i,j}^{[l]}}$$

Convolutional Layer Backpropagation 5: Gradient w.r.t. Input

To propagate the error back, we need $\frac{\partial E}{\partial a^{[l-1]}}$. Each input affects only the outputs whose window covers it:

$$\frac{\partial E}{\partial a_{00}^{[l-1]}} = \frac{\partial E}{\partial z_{00}^{[l]}} w_{00}^{[l]}$$

$$\frac{\partial E}{\partial a_{01}^{[l-1]}} = \frac{\partial E}{\partial z_{00}^{[l]}} w_{01}^{[l]} + \frac{\partial E}{\partial z_{01}^{[l]}} w_{00}^{[l]}$$

$$\frac{\partial E}{\partial a_{02}^{[l-1]}} = \frac{\partial E}{\partial z_{01}^{[l]}} w_{01}^{[l]}$$

$$\vdots$$

$$\frac{\partial E}{\partial a_{11}^{[l-1]}} = \frac{\partial E}{\partial z_{00}^{[l]}} w_{11}^{[l]} + \frac{\partial E}{\partial z_{01}^{[l]}} w_{10}^{[l]} + \frac{\partial E}{\partial z_{10}^{[l]}} w_{01}^{[l]} + \frac{\partial E}{\partial z_{11}^{[l]}} w_{00}^{[l]}$$

$$\vdots$$

$$\frac{\partial E}{\partial a_{22}^{[l-1]}} = \frac{\partial E}{\partial z_{11}^{[l]}} w_{11}^{[l]}$$

Convolutional Layer Backpropagation 6: Padding and Rotation

The equations on the previous slide are a cross-correlation of the zero-padded error with the kernel rotated by 180°:

$$\underbrace{\begin{bmatrix} \frac{\partial E}{\partial a_{00}^{[l-1]}} & \frac{\partial E}{\partial a_{01}^{[l-1]}} & \frac{\partial E}{\partial a_{02}^{[l-1]}} \\ \frac{\partial E}{\partial a_{10}^{[l-1]}} & \frac{\partial E}{\partial a_{11}^{[l-1]}} & \frac{\partial E}{\partial a_{12}^{[l-1]}} \\ \frac{\partial E}{\partial a_{20}^{[l-1]}} & \frac{\partial E}{\partial a_{21}^{[l-1]}} & \frac{\partial E}{\partial a_{22}^{[l-1]}} \end{bmatrix}}_{\frac{\partial E}{\partial a^{[l-1]}}} = \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \frac{\partial E}{\partial z_{00}^{l}} & \frac{\partial E}{\partial z_{01}^{l}} & 0 \\ 0 & \frac{\partial E}{\partial z_{10}^{l}} & \frac{\partial E}{\partial z_{11}^{l}} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\text{Padded } \frac{\partial E}{\partial z^{l}}} \otimes \underbrace{\begin{bmatrix} w_{11}^{[l]} & w_{10}^{[l]} \\ w_{01}^{[l]} & w_{00}^{[l]} \end{bmatrix}}_{\text{Rotated } w^{[l]}} \tag{1}$$

Multiplying by the derivative of the previous layer's activation $a^{[l-1]} = g(z^{[l-1]})$ gives

$$\boxed{\delta^{[l-1]} = \frac{\partial E}{\partial z^{[l-1]}} = \frac{\partial E}{\partial a^{[l-1]}} \cdot \frac{\partial a^{[l-1]}}{\partial z^{[l-1]}} = \left(\text{pad}(\delta^{[l]}) \otimes \text{rot180}(w^{[l]})\right) \odot g'(z^{[l-1]})}$$

- $\otimes$: cross-correlation; $\odot$: element-wise (Hadamard) product.
- $\text{pad}(\delta^{[l]})$: the error term from the next layer, zero-padded.
- $w^{[l]}$: the kernel of the convolutional layer.

Why the Kernel Gradient Is a Cross-Correlation

For the complete output feature map, every kernel element is reused at many spatial locations.
For kernel element $w_{p,q}$:

$$\boxed{\frac{\partial E}{\partial w_{p,q}} = \sum_i \sum_j a_{i+p,j+q}^{[l-1]} \, \delta_{ij}^{[l]}}$$

For all kernel elements together:

$$\boxed{\frac{\partial E}{\partial w^{[l]}} = a^{[l-1]} \otimes \delta^{[l]}}$$

Thus the same local input values that were used in the forward pass are multiplied by the corresponding error signal in the backward pass.

Interpretation

The kernel gradient accumulates information from **all locations where the kernel was used**.

Numerical Example: Kernel Gradient

Suppose the 2×2 error map is

$$\delta = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

and the 3×3 input is

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

For the kernel element w_{00}:

$$\frac{\partial E}{\partial w_{00}} = 1(1) + 2(2) + 3(4) + 4(5).$$

Hence:

$$\boxed{\frac{\partial E}{\partial w_{00}} = 1 + 4 + 12 + 20 = 37}$$

Important

Each occurrence of a kernel weight contributes to the gradient. The contributions are summed over all output positions.

Gradient with Respect to the Bias

The same bias b is added to every output $z_{ij} = \sum_{p,q} w_{pq} a_{i+p,j+q} + b$:
Therefore,

$$\frac{\partial z_{ij}}{\partial b} = 1.$$

Applying the chain rule:

$$\boxed{\frac{\partial E}{\partial b} = \sum_i \sum_j \delta_{ij}}$$

For example, if

$$\delta = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix},$$

then

$$\boxed{\frac{\partial E}{\partial b} = 1 + 2 + 3 + 4 = 10.}$$

Reason

A shared bias receives a gradient contribution from every output location.

Input Gradient: Numerical Setup

Consider

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \qquad w = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \qquad b = 0.$$

The forward cross-correlation gives

$$z = \begin{bmatrix} 37 & 47 \\ 67 & 77 \end{bmatrix}.$$

Assume the upstream gradient is

$$\boxed{\delta = \frac{\partial E}{\partial z} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}}$$

where each δ_{ij} is the gradient arriving from the subsequent layer.

Input Gradient: Corner Element

Consider a_{00}. It occurs only in z_{00}:

$$z_{00} = w_{00}a_{00} + w_{01}a_{01} + w_{10}a_{10} + w_{11}a_{11}.$$

Thus

$$\boxed{\frac{\partial E}{\partial a_{00}} = \delta_{00}w_{00}}$$

and numerically

$$\frac{\partial E}{\partial a_{00}} = 1(1) = \boxed{1}.$$

Boundary effect

A corner element belongs to only one receptive field in this valid, stride-1 example.

Input Gradient: Edge Element

Consider a_{01}. It occurs in both z_{00} and z_{01}:

$$\frac{\partial z_{00}}{\partial a_{01}} = w_{01}, \qquad \frac{\partial z_{01}}{\partial a_{01}} = w_{00}.$$

Therefore,

$$\boxed{\frac{\partial E}{\partial a_{01}} = \delta_{00} w_{01} + \delta_{01} w_{00}}$$

and

$$\frac{\partial E}{\partial a_{01}} = 1(2) + 2(1) = \boxed{4}.$$

The two terms arise from the two overlapping receptive fields.

Input Gradient: Interior Element

Consider the center element a_{11}. It occurs in all four outputs:

$$z_{00},\ z_{01},\ z_{10},\ z_{11}.$$

Hence

$$\boxed{\frac{\partial E}{\partial a_{11}} = \delta_{00} w_{11} + \delta_{01} w_{10} + \delta_{10} w_{01} + \delta_{11} w_{00}}$$

Numerically,

$$\frac{\partial E}{\partial a_{11}} = 1(4) + 2(3) + 3(2) + 4(1) = \boxed{20}.$$

Interpretation

The interior element receives four contributions because all four receptive fields overlap at this location.

Complete Numerical Input Gradient

For

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad w = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad \delta = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix},$$

the nine input gradients are

$$\begin{aligned} \frac{\partial E}{\partial a_{00}} &= 1, & \frac{\partial E}{\partial a_{01}} &= 4, & \frac{\partial E}{\partial a_{02}} &= 4, \\ \frac{\partial E}{\partial a_{10}} &= 6, & \frac{\partial E}{\partial a_{11}} &= 20, & \frac{\partial E}{\partial a_{12}} &= 20, \\ \frac{\partial E}{\partial a_{20}} &= 9, & \frac{\partial E}{\partial a_{21}} &= 24, & \frac{\partial E}{\partial a_{22}} &= 16. \end{aligned}$$

Thus

$$\boxed{\frac{\partial E}{\partial a} = \begin{bmatrix} 1 & 4 & 4 \\ 6 & 20 & 16 \\ 9 & 24 & 16 \end{bmatrix}}$$

The gradient is larger at positions receiving contributions from several output units.

Numerical Check: Padding and Rotation

For the example,

$$\text{rot180}(w) = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix}.$$

Pad the 2×2 error map by one cell:

$$\text{pad}(\delta) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Then

$$\boxed{\text{pad}(\delta) \otimes \text{rot180}(w) = \begin{bmatrix} 1 & 4 & 4 \\ 6 & 20 & 16 \\ 9 & 24 & 16 \end{bmatrix}}$$

which is exactly the element-by-element result.

Compact form

The padding-and-rotation formula is simply a matrix form of the chain-rule summation.

Input Gradient: Boundary versus Interior

Input position	Contributing outputs	Gradient
a_{00} (corner)	1	1
a_{01} (edge)	2	4
a_{11} (interior)	4	20

In general:

- Boundary elements participate in fewer receptive fields.
- Interior elements participate in more receptive fields.
- The contributions are summed, not averaged.

This follows directly from local connectivity and overlapping receptive fields.

Input Gradient with Stride

Stride changes which output positions depend on a particular input. For a 1D stride-2 example:

$$z_0 = w_0 x_0 + w_1 x_1 + w_2 x_2, \qquad z_1 = w_0 x_2 + w_1 x_3 + w_2 x_4.$$

Let

$$\delta_0 = \frac{\partial E}{\partial z_0}, \qquad \delta_1 = \frac{\partial E}{\partial z_1}.$$

Then

$$\boxed{\begin{aligned} \frac{\partial E}{\partial x_0} &= \delta_0 w_0, \\ \frac{\partial E}{\partial x_1} &= \delta_0 w_1, \\ \frac{\partial E}{\partial x_2} &= \delta_0 w_2 + \delta_1 w_0, \\ \frac{\partial E}{\partial x_3} &= \delta_1 w_1, \\ \frac{\partial E}{\partial x_4} &= \delta_1 w_2. \end{aligned}}$$

Only x_2 belongs to both receptive fields, so it receives two contributions.

Input Gradient: General Formula

For stride 1 and no padding:

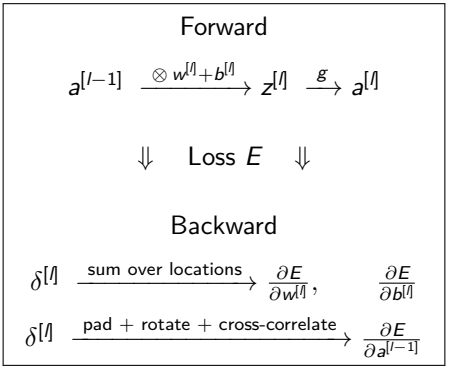
$$\frac{\partial E}{\partial a^{[l-1]}} = \text{pad}\left(\frac{\partial E}{\partial z^{[l]}}\right) \otimes \text{rot180}(w^{[l]})$$

For an individual input element:

$$\frac{\partial E}{\partial a_{r,s}^{[l-1]}} = \sum_{\substack{i,j \\ 0 \le r-i < k_h \\ 0 \le s-j < k_w}} \delta_{ij}^{[l]} w_{r-i,s-j}^{[l]}$$

where k_h and k_w are the kernel height and width. For non-unit stride, only output positions whose receptive fields contain the input element contribute.

Convolutional Backpropagation: Complete Picture



Main message

The convolutional layer is trained by three gradients:

$$\frac{\partial E}{\partial w}, \quad \frac{\partial E}{\partial b}, \quad \frac{\partial E}{\partial a}.$$

Max Pooling: Forward Pass

The output at position (i, j) is

$$O_{i,j} = \max_{(m,n)\in \text{window}} X_{i\cdot s+m,\, j\cdot s+n}.$$

The pooling window moves according to the stride s.

Effect

Max pooling retains the strongest activation within each local region.

Max Pooling: Backward Pass

The gradient is propagated only to the input position that produced the maximum:

$$\frac{\partial E}{\partial X_{i\cdot s+m, j\cdot s+n}} = \begin{cases} \frac{\partial E}{\partial O_{i,j}}, & X_{i\cdot s+m, j\cdot s+n} = O_{i,j}, \\ 0, & \text{otherwise.} \end{cases}$$

Thus, during backpropagation, the maximum location receives the upstream gradient.

Average Pooling: Forward and Backward Pass

Forward:

$$O_{i,j} = \frac{1}{f^2} \sum_{(m,n)\in \text{window}} X_{i\cdot s+m, j\cdot s+n}.$$

Backward:

$$\boxed{\frac{\partial E}{\partial X_{i\cdot s+m, j\cdot s+n}} = \frac{1}{f^2} \frac{\partial E}{\partial O_{i,j}}}$$

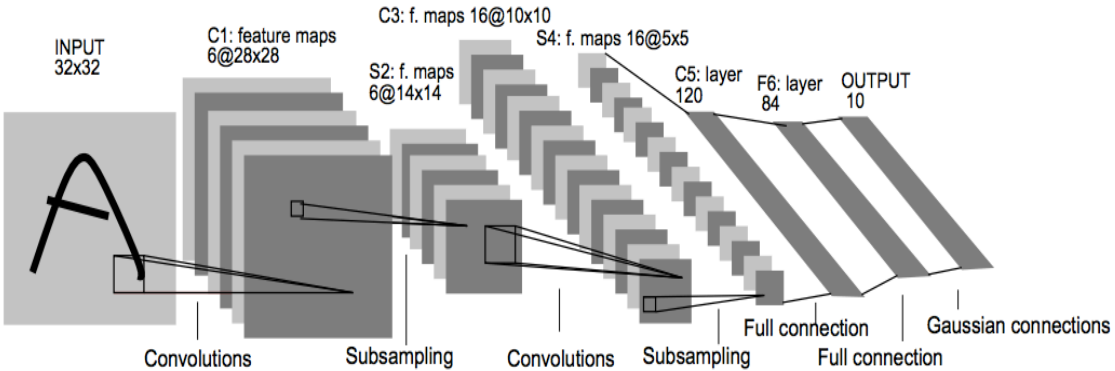
The upstream gradient is distributed equally among all positions in the pooling window.

LeNet-5

LeNet-5 is a classic convolutional neural network proposed by Yann LeCun and colleagues in 1998. It was designed primarily for handwritten digit recognition.

- Input: grayscale 32×32 image.
- First convolution: 6 filters of size 5×5.
- Subsampling follows the first convolution.
- Second convolution: 16 filters of size 5×5.
- Additional subsampling.
- Fully connected layers with 120, 84, and 10 neurons.

LeNet-5 Architecture



LeNet-5 architecture for handwritten character recognition.

LeNet-5: Main Components

1. 32×32 grayscale input.
2. Six 5×5 convolutional filters followed by 2×2 subsampling.
3. Sixteen 5×5 convolutional filters followed by subsampling.
4. Flattening followed by fully connected layers of 120 and 84 neurons.
5. Ten output neurons corresponding to the ten MNIST classes.

The source states that LeNet-5 uses sigmoid activation in its layers except the output layer, which uses softmax.

Training LeNet-5

The source describes the training procedure as:

$$\boxed{\text{Forward propagation} \rightarrow \text{Loss} \rightarrow \text{Backpropagation} \rightarrow \text{Gradient descent}}$$

- Weights are updated iteratively.
- The objective is to minimize the training loss.
- Cross-entropy is used as the loss function.
- The architecture demonstrated the effectiveness of CNNs for image classification.

Assignment

Implement the LeNet-5 network using:

- Keras
- Google Colab
- MNIST dataset

Load the dataset using:

```
tf.keras.datasets.mnist.load_data()
```

Goal

Build, train, and evaluate a LeNet-5-style network for handwritten-digit recognition.

Summary

- CNNs exploit local spatial structure and parameter sharing.
- Convolution in deep-learning frameworks is implemented as cross-correlation.
- Padding and stride control the spatial dimensions of feature maps.
- Multiple filters produce multiple feature maps.
- Pooling performs spatial down-sampling.
- Convolutional-layer gradients are themselves related to cross-correlation operations.
- Pooling layers have simple, structured backward passes.
- LeNet-5 illustrates the combination of convolution, subsampling, and fully connected layers.

Thank You

Questions?