DSCI 565: CONVOLUTIONAL NEURAL NETWORKS

Ke-Thia Yao

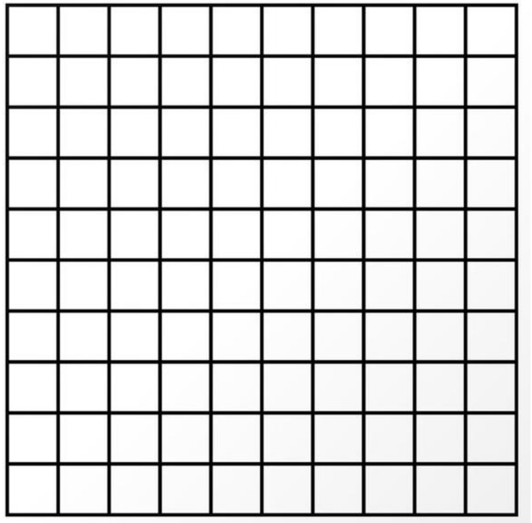
Lecture 8: 2025-09-22

Convolutional Neural Networks

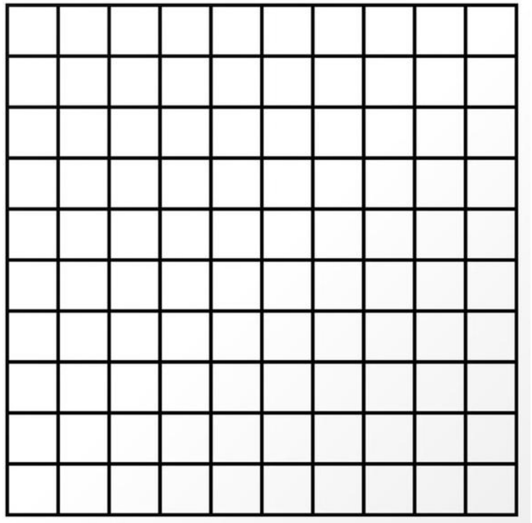
- Image data has a grid structure of pixels
- The neural networks that we have seen so far requires flattening of images in into vectors, which destroys the grid structure
- Can we some how preserve this grid structure?

Initial Attempt

- Give an input image $\mathbf{X}$, to preserve the grid structure we require the hidden layer $\mathbf{H}$ to have same shape as $\mathbf{X}$
- Let $[\mathbf{X}]_{i,j}$ and $[\mathbf{H}]_{i,j}$ denote the pixel at (i, j)
- $[\mathbf{H}]_{i,j}$ captures some feature related to (i, j)



$\mathbf{X}$



$\mathbf{H}$

Initial Attempt

- Then, if we use a fully connect architecture as before we need
  - A second-order tensor for the bias $[\mathbf{U}]_{i,j}$
  - A fourth-order tensor for the weights $[\mathbf{W}]_{i,j,kl}$
- Then, to compute $[\mathbf{H}]_{i,j}$

$(10^3)^4 = 10^{12}$

$$[\mathbf{H}]_{i,j} = [\mathbf{U}]_{i,j} + \sum_k \sum_l [\mathsf{W}]_{i,j,k,l} [\mathbf{X}]_{k,l}$$

- For a megapixel image (1000x1000), the weight matrix would have 10^{12} (a trillion parameters)
- The largest Large Language Models (LLMs) is about 200 billion parameters
- What else is wrong with this initial attempt?

Translational Invariance

- Detecting Waldo at one part of the image should be no different than any other part

$$[\mathbf{H}]_{i,j} = [\mathbf{U}]_{i,j} + \sum_k \sum_l [\mathsf{W}]_{\cancel{i,j},k,l} [\mathbf{X}]_{k,l}$$

Locality

- To capture the feature at location (i, j) we should not have to look at the entire image
- Only look at nearby pixels $(i + a, j + b)$ and $a, b \in [-\Delta, +\Delta]$

$$[\mathbf{H}]_{i,j} = [\mathbf{U}]_{i,j} + \sum_k \sum_l [\mathsf{W}]_{i,j,k,l} [\mathbf{X}]_{k,l}$$
$$= [\mathbf{U}]_{i,j} + \sum_a \sum_b [\mathsf{V}]_{i,j,a,b} [\mathbf{X}]_{i+a,j+b}$$

Translational Invariance

$$[\mathbf{H}]_{i,j} = u + \sum_a \sum_b [\mathbf{V}]_{a,b} [\mathbf{X}]_{i+a,j+b}$$

Locality

$$[\mathbf{H}]_{i,j} = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} [\mathbf{V}]_{a,b} [\mathbf{X}]_{i+a,j+b}$$

Channels

- So far, we have ignored the fact that some images have color channels
- With color
  - input becomes a third-order tensor $[\mathbf{X}]_{i,j,c}$ and
  - weight because a third-order tensor as well $[\mathbf{V}]_{a,b,c}$
- Also, instead of just one hidden feature map we want multiple feature maps $[\mathbf{H}]_{i,j,d}$, and weight becomes $[\mathbf{V}]_{a,b,c,d}$

$$[\mathbf{H}]_{i,j} = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} [\mathbf{V}]_{a,b} [\mathbf{X}]_{i+a,j+b}$$

$$[\mathrm{H}]_{i,j,d} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \sum_{c} [\mathrm{V}]_{a,b,c,d} [\mathrm{X}]_{i+a,j+b,c}$$

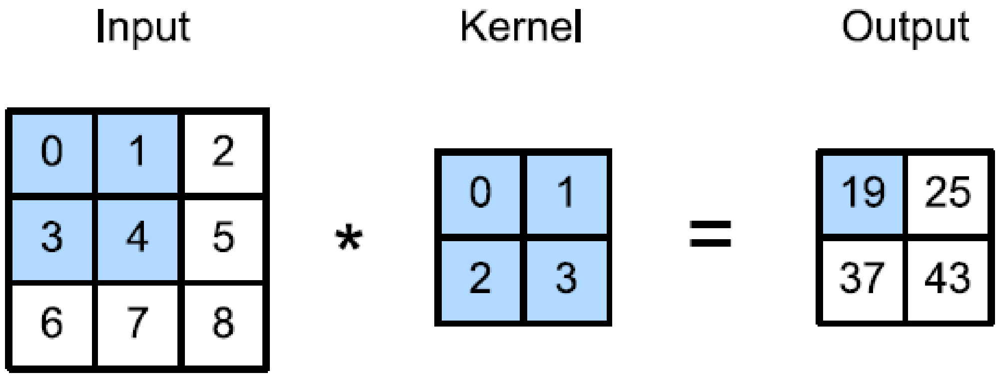
Convolution vs Cross-Correlation

- In machine learning, we call the previous formulas convolution
- But, in mathematics it is called cross-correlation
- In mathematics, convolution direction is “flipped”

$$(f * g)(\mathbf{x}) = \int f(\mathbf{z}) g(\mathbf{x} - \mathbf{z}) d\mathbf{z}.$$

Cross-Correlation Operation

- Example of a cross-correlation operation using a 2x2 kernel



$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$
$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$
$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$
$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$

- The resulting output shape reduce by 1 in each dimension
- In general, if the input size is $n_h \times n_w$ and kernel size is $k_h \times k_w$ output shape is

$$(n_h - k_h + 1, n_w - k_w + 1)$$

Inspiration for Mammal Visual Cortex

- Cats (and monkeys) have many neurons in the visual cortex of the brain have small local receptive fields
- They only activate when a limited region of the visual field is stimulated
- Moreover, some of these neurons respond to lines in certain directions

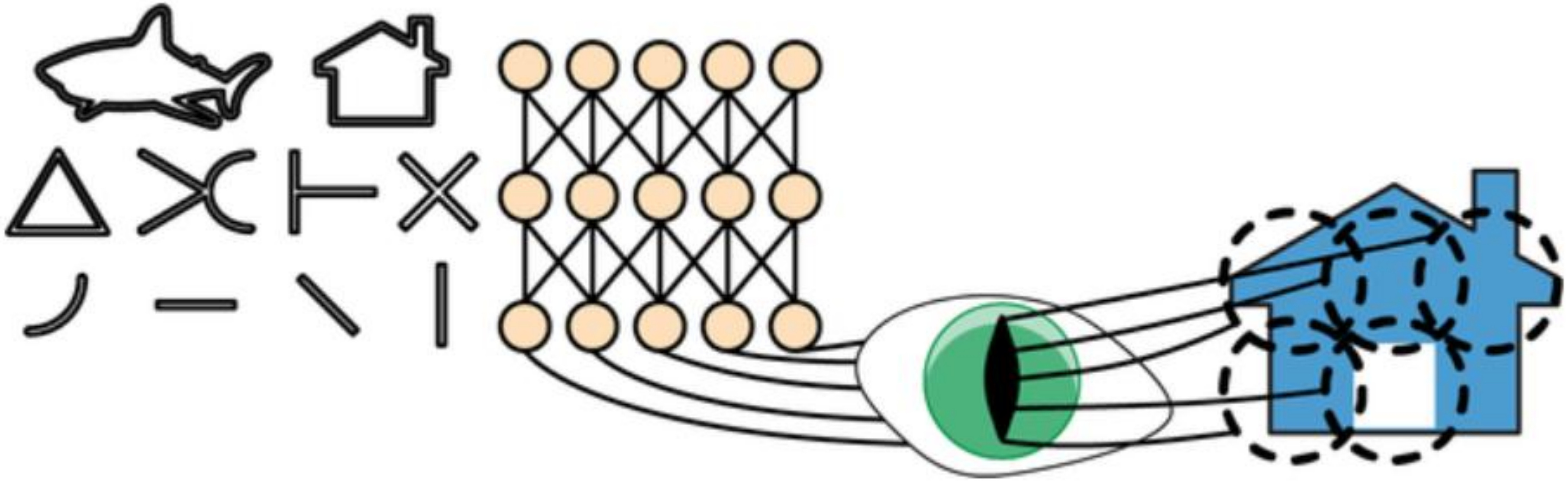
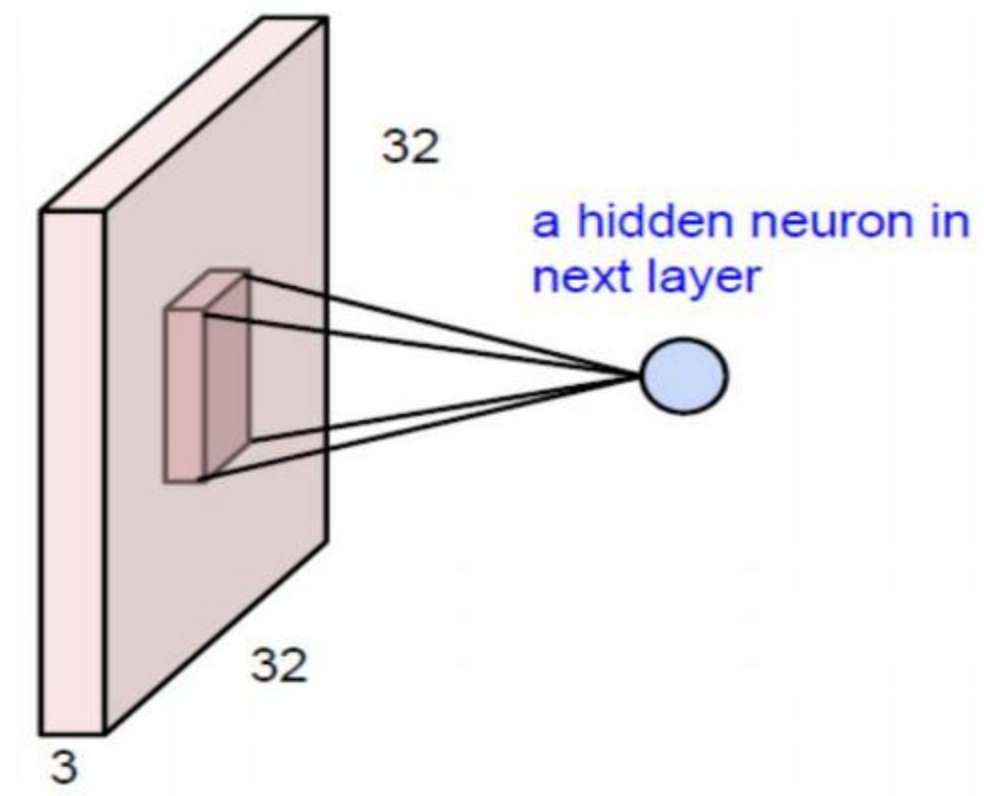


Figure 14-1. Biological neurons in the visual cortex respond to specific patterns in small regions of the visual field called receptive fields; as the visual signal makes its way through consecutive brain modules, neurons respond to more complex patterns in larger receptive fields

Convolution Filter

Use convolution filter to mimic neurons with limited receptive fields



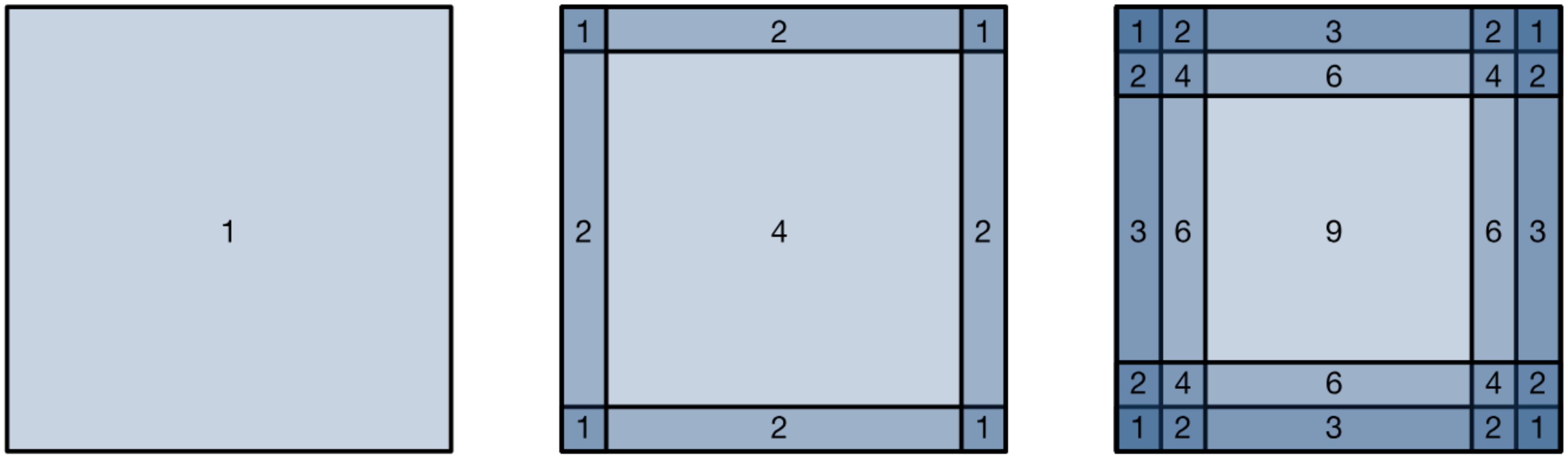
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Notebook

- chapter_convolutional-neural-networks/conv-layer.ipynb

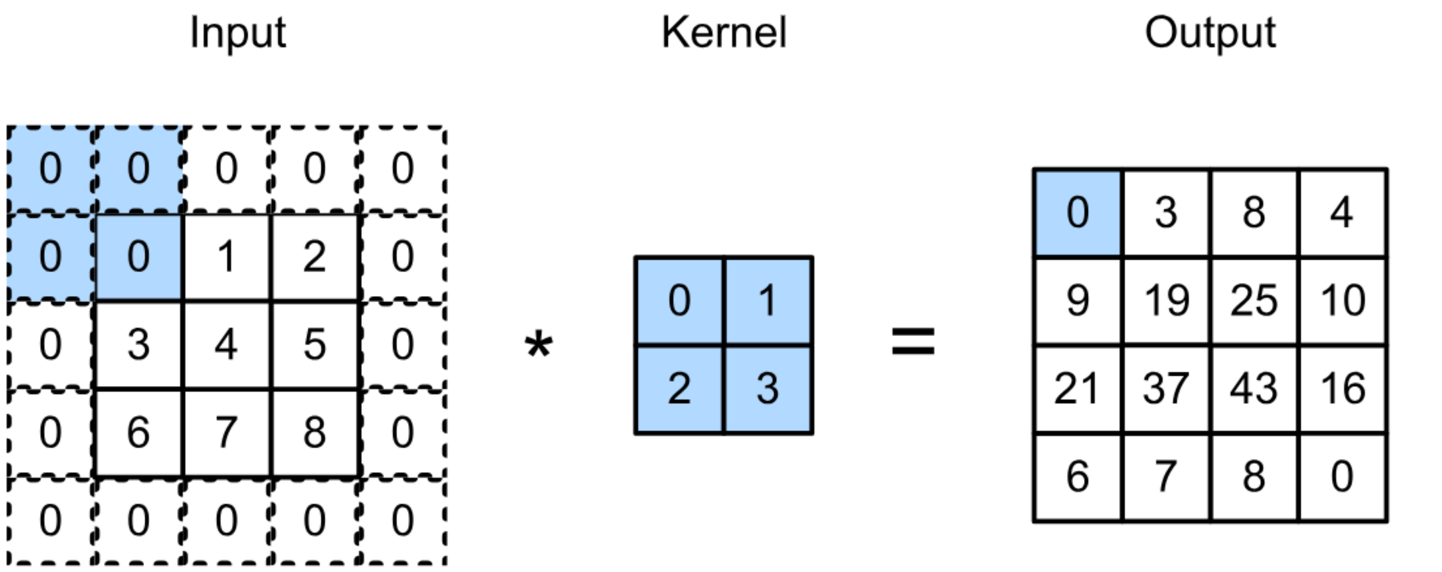
Padding

- For convolutions
  - Pixels on edges of images are used less often
  - Size of the image shrinks
- Shrinking becomes problematic if we have multiple convolution layers



Padding

- Padding adds extra zero pixels around the border



- With padding p_h and p_w the output shape is

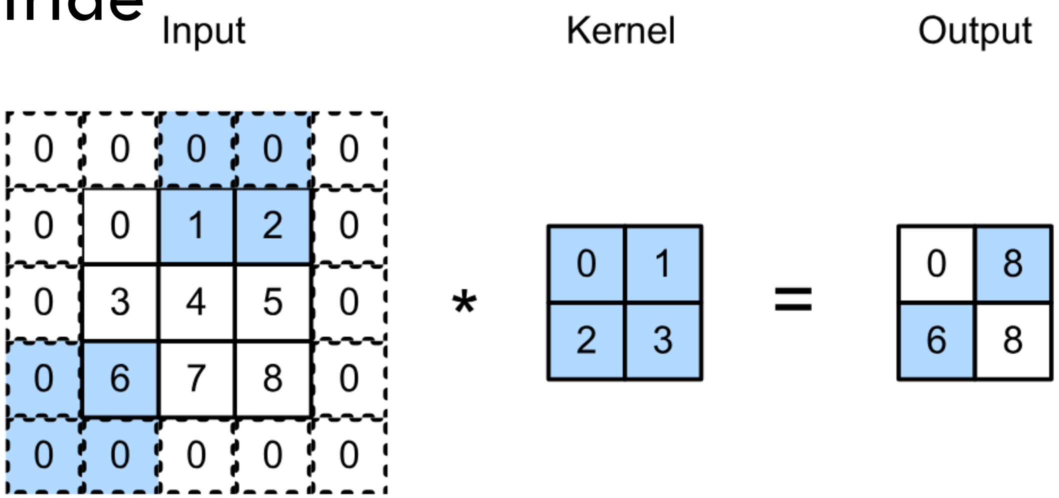
$$(n_h - k_h + p_h + 1, n_w - k_w + p_w + 1)$$
$$= (3 - 2 + 2 + 1, 3 - 2 + 2 + 1) = (4, 4)$$

Padding and Kernel Size

- To have the same number of pixel of each side of the border:
  p_h and p_w must be even
- To keep the same image size:
  $p_h = k_h - 1$ and $p_w = k_w - 1$
- Then kernel size must be odd
- Having the same image size makes the interpretation of the hidden layer simpler:
  $[\mathbf{H}]_{i,j}$ is calculated using pixels centered around $[\mathbf{X}]_{i,j}$
- Note: PyTorch padding `Conv2d(…, padding=p)`is $p_h = p_w = 2p$

Stride

- By default, the convolution is slides one pixel at a time (stride of 1)
- To decrease computational complexity and/or to downsample, we can increase the stride



- With stride s_h and s_w, the output shape is

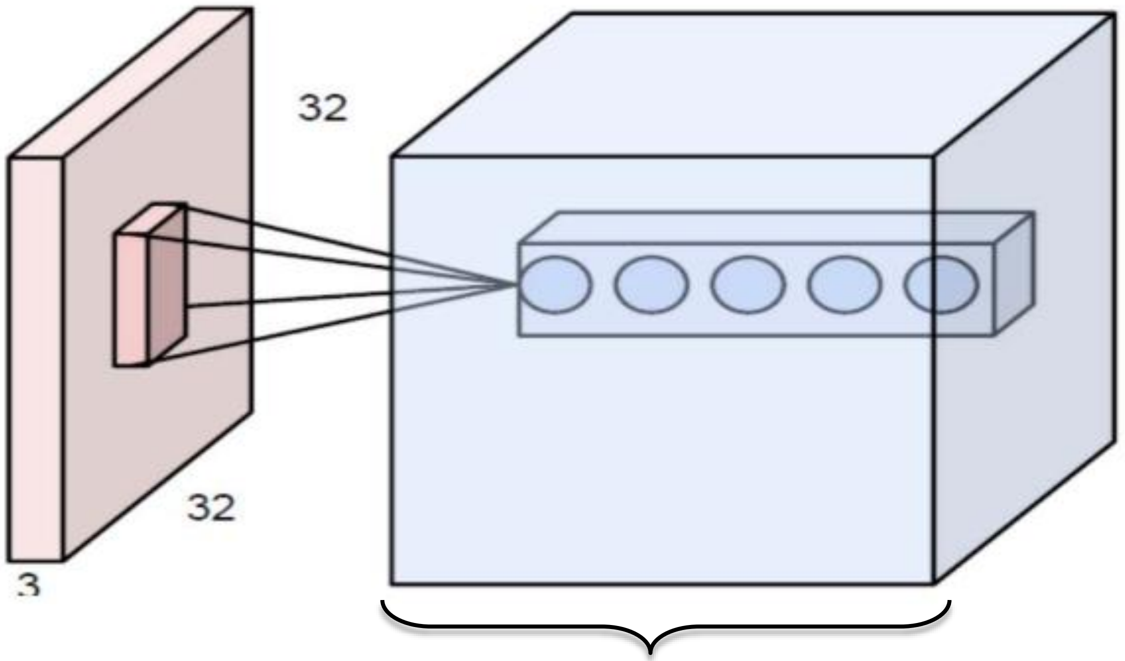
$$\lfloor(n_\mathrm{h}-k_\mathrm{h}+p_\mathrm{h}+s_\mathrm{h})/s_\mathrm{h}\rfloor \times \lfloor(n_\mathrm{w}-k_\mathrm{w}+p_\mathrm{w}+s_\mathrm{w})/s_\mathrm{w}\rfloor$$

Notebook

- chapter_convolutional-neural-networks/padding-and-strides.ipynb

Multiple Input and Multiple Output Channels

- Three input channels (RGB colors) with five output channels



# of filters/channels/feature maps

- With input channel size c_i, the kernel shape becomes $c_i \times k_h \times k_w$

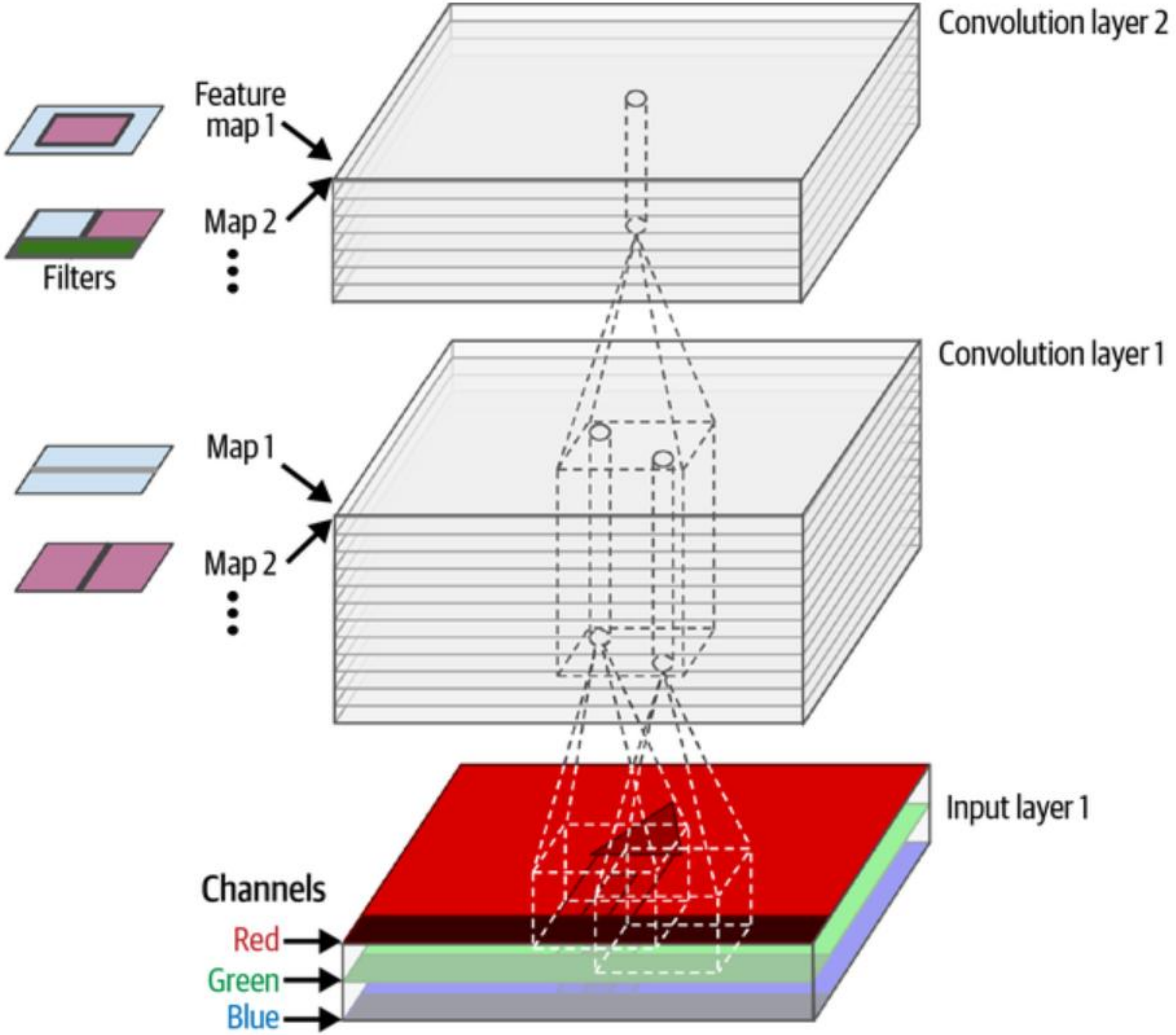
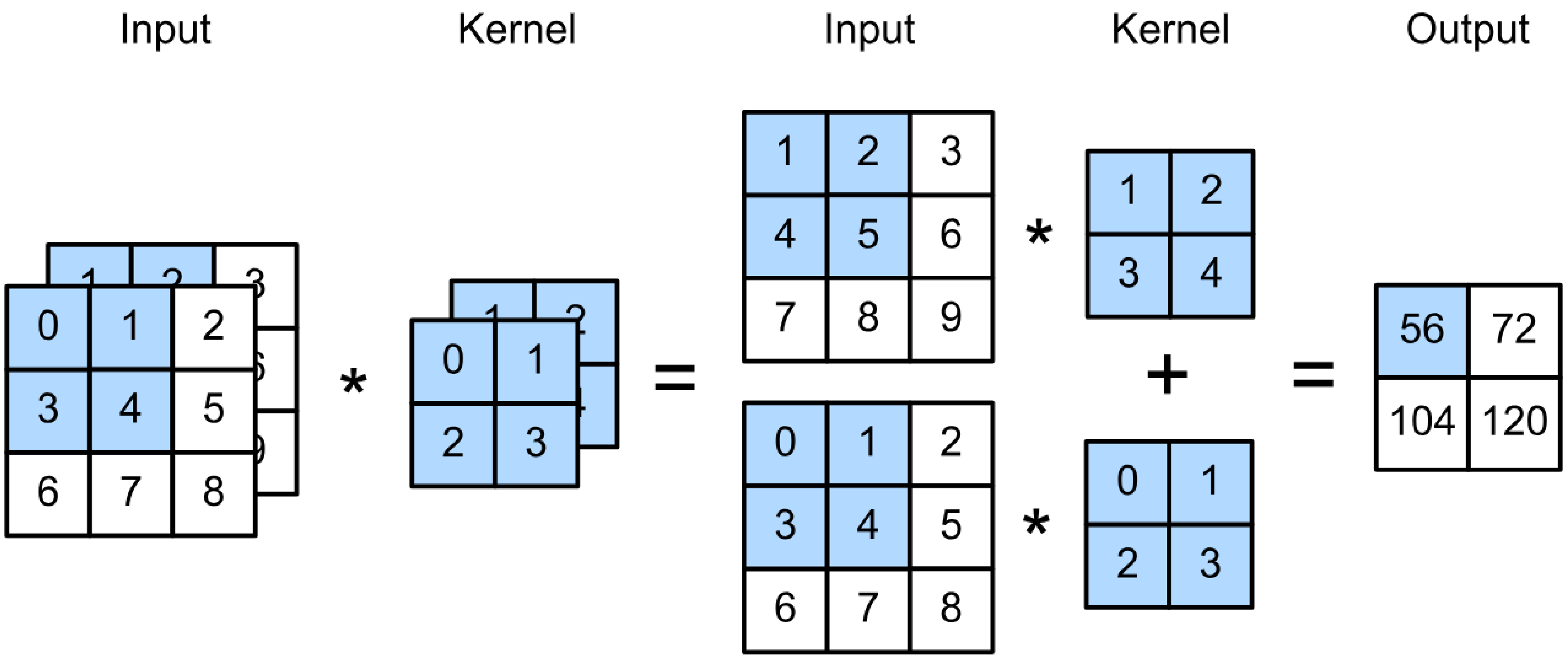


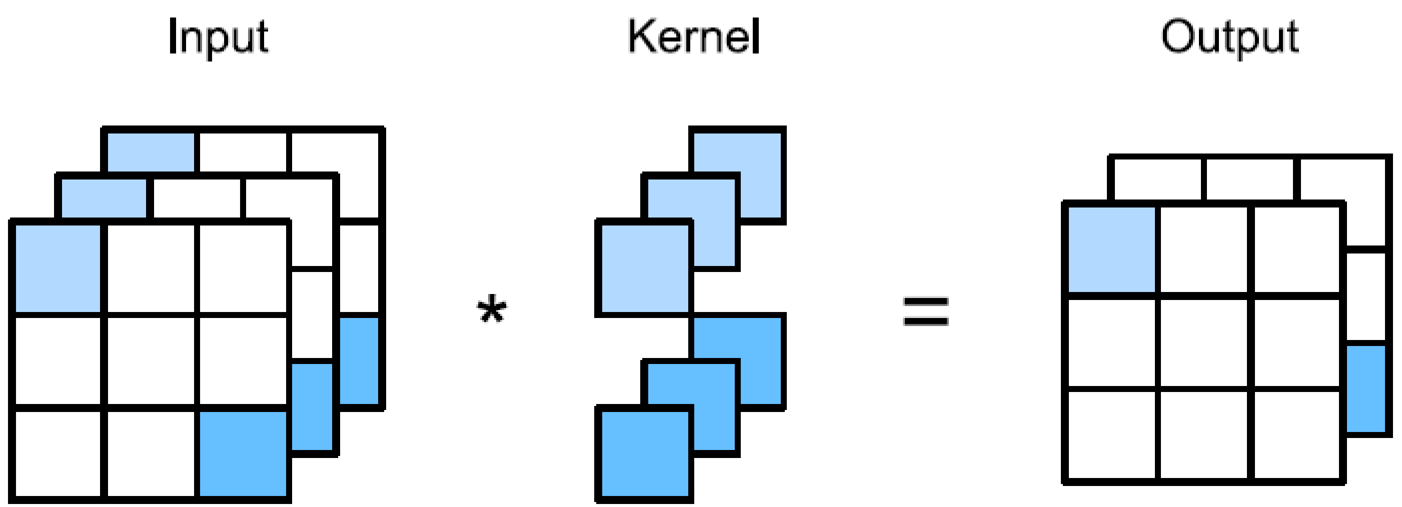
Figure 14-6. Two convolutional layers with multiple filters each (kernels), processing a color image with three color channels; each convolutional layer outputs one feature map per filter

Two Input Channel Examples



1×1 Convolutional Layer

- With a 1×1 convolution, the convolution is only accessing one pixel, and no adjacent pixels
- But this one pixel consists of multiple channels
- $\Rightarrow$ like performing a dot-product of channel vector with kernel's vector

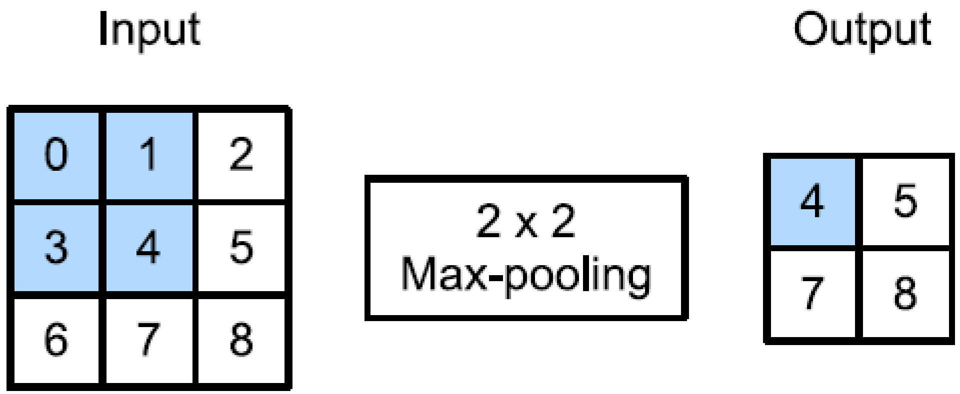


Notebook

- chapter_convolutional-neural-networks/channels.ipynb

Pooling

- Pooling combines multiple pixels into one pixel, while keeping the number channels the same size
- Rational for pooling
  - Down sample to reduce size
  - Mitigating sensitivity of convolution to specific location
- Operations to combine pixels: **max**, average, min



Pooling and Stride

- The default stride for a pooling is the same shape as the pooling window shape
- E.g., if the pooling window shape is 2x2, then the stride is 2x2

See chapter_convolutional-neural-networks/pooling.ipynb

LeNet

- Developed by Yann LeCun, LeNet is one of the first CNNs
- Designed for digit recognition, it reached error rate of 1%
- Error rate comparable to support vector machines
- LeNet is in use in ATM machine

LeNet-5