Advanced Deep Learning

Lec 4: Convolutional Neural Networks

STAT4744/5744

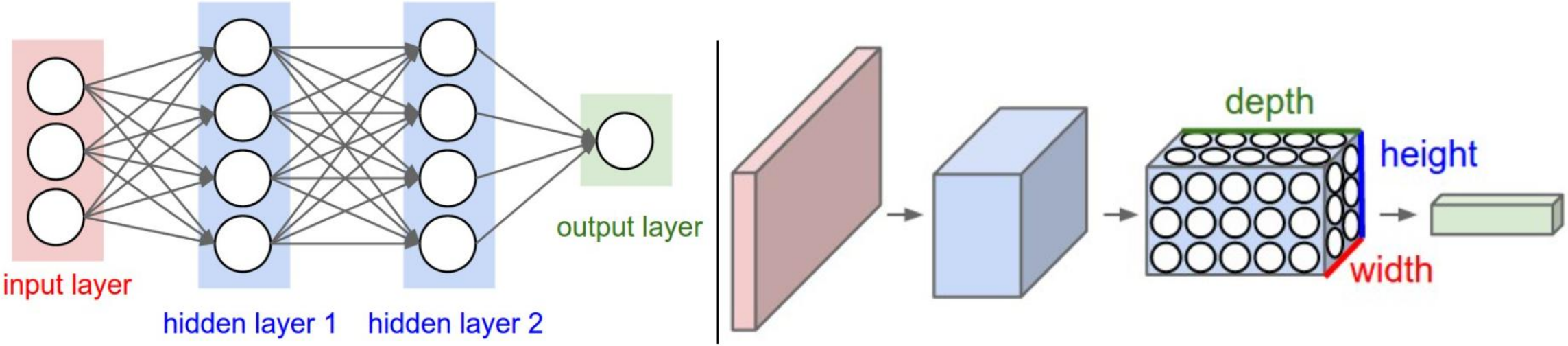
Xin (Shayne) Xing • Virginia Tech

Today’s Outlines

1. Overview
2. Convolutional Layer
3. Implementation using PyTorch

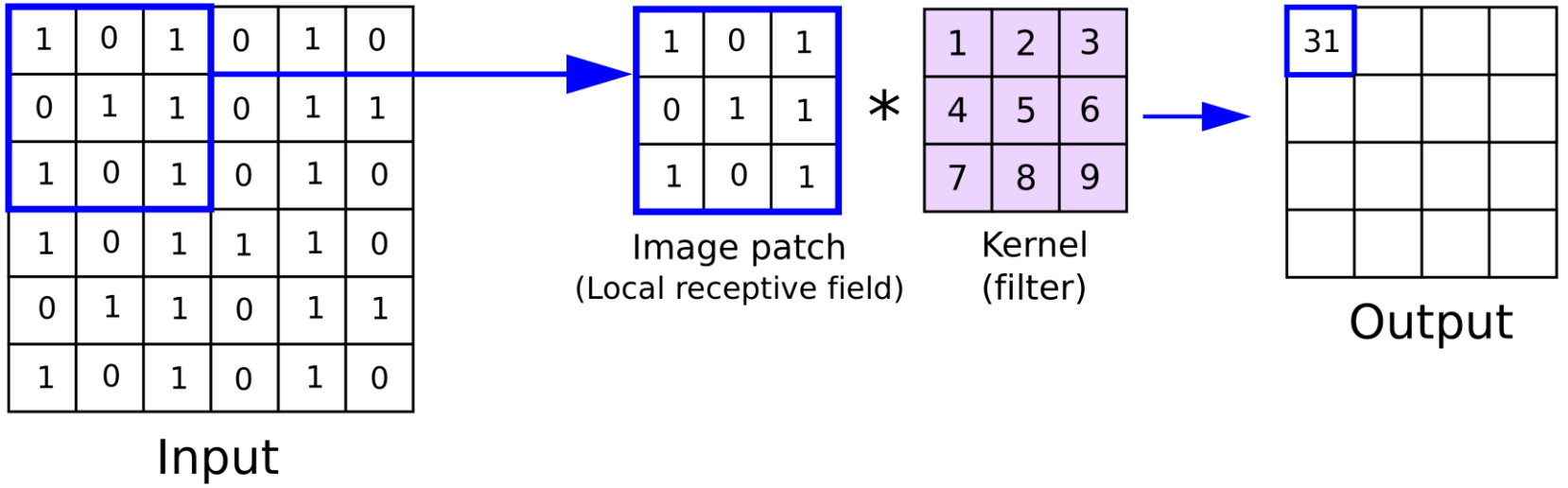
Architecture Overview

- Regular Neural Nets don’t scale well to full images.
- Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way.



Convolution operation

- A convolution operation is an element wise matrix multiplication operation. Where one of the matrices is the image, and the other is the 3D **filter** or 2D **kernel** that turns the image into something else. The output of this is the final convoluted image.



Examples

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Smooth

0	-1	0
-1	5	-1
0	-1	0

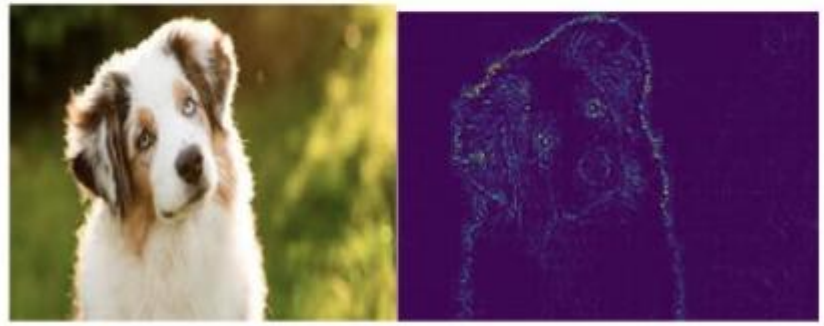
Sharpen

-1	-2	-1
0	0	0
1	2	1

Horizontal

-1	0	1
-2	0	2
-1	0	1

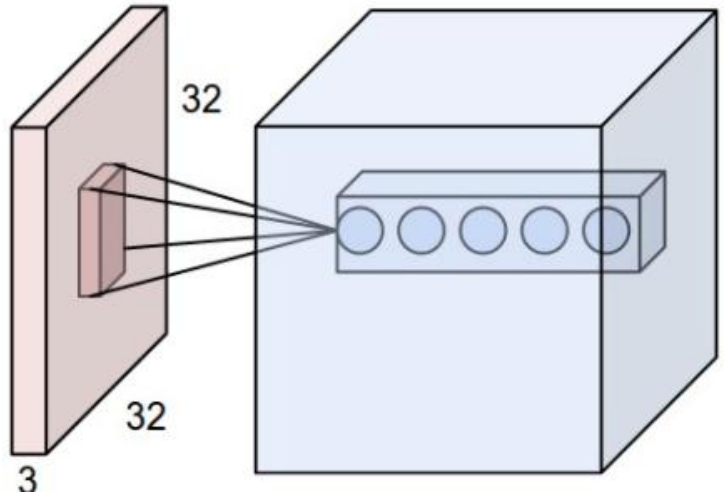
Vertical



Edge detection

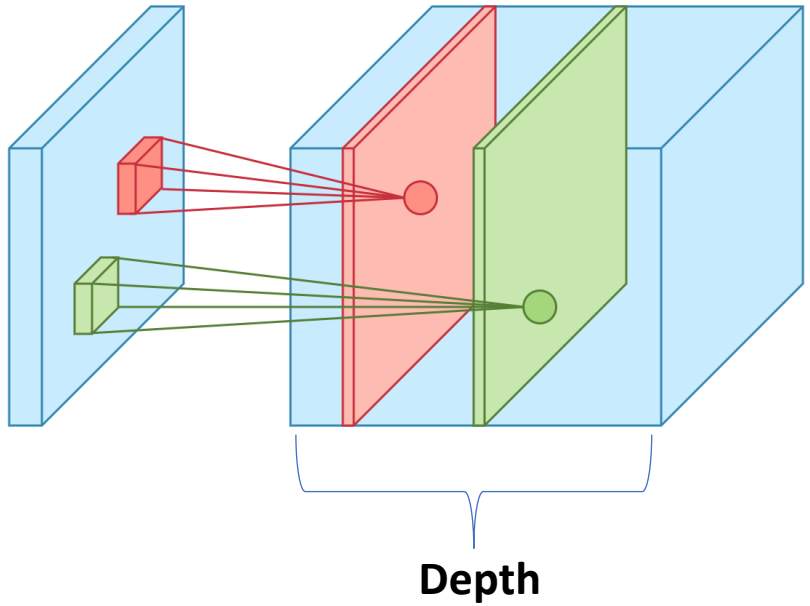
Convolution layers

- **Convolution layer is created by multiple convolutional 3D filters (2D kernels).**
- **Spatial arrangement**. We have explained the connectivity of each neuron in the Conv Layer to the input volume, but we haven't yet discussed how many neurons there are in the output volume or how they are arranged. Three hyperparameters control the size of the output volume: the **depth, stride** and **zero-padding**. We discuss these next:



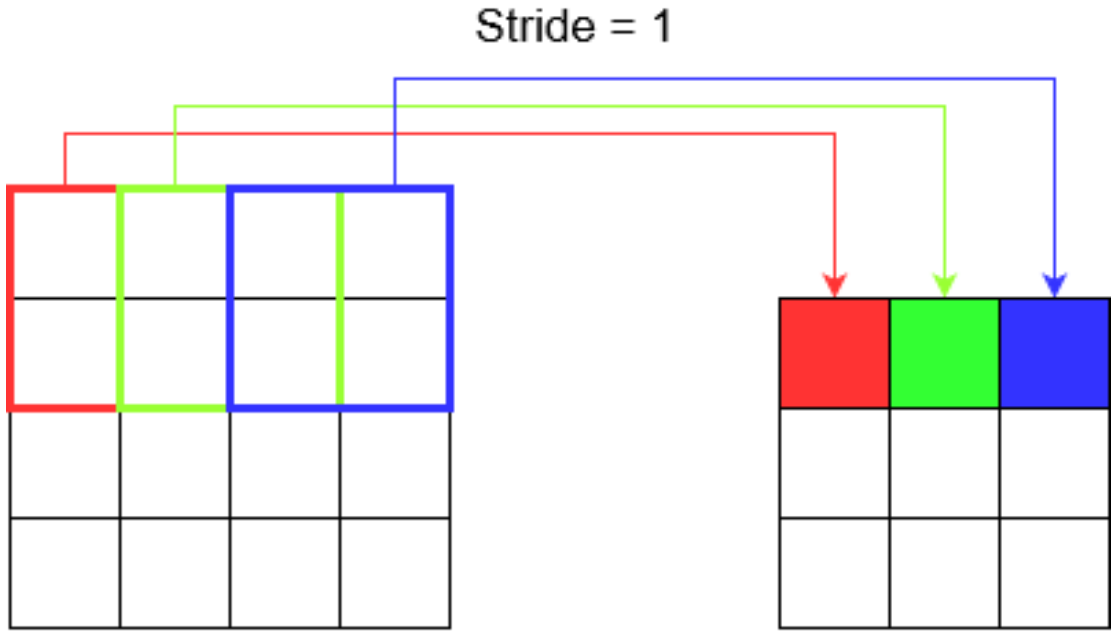
Depth

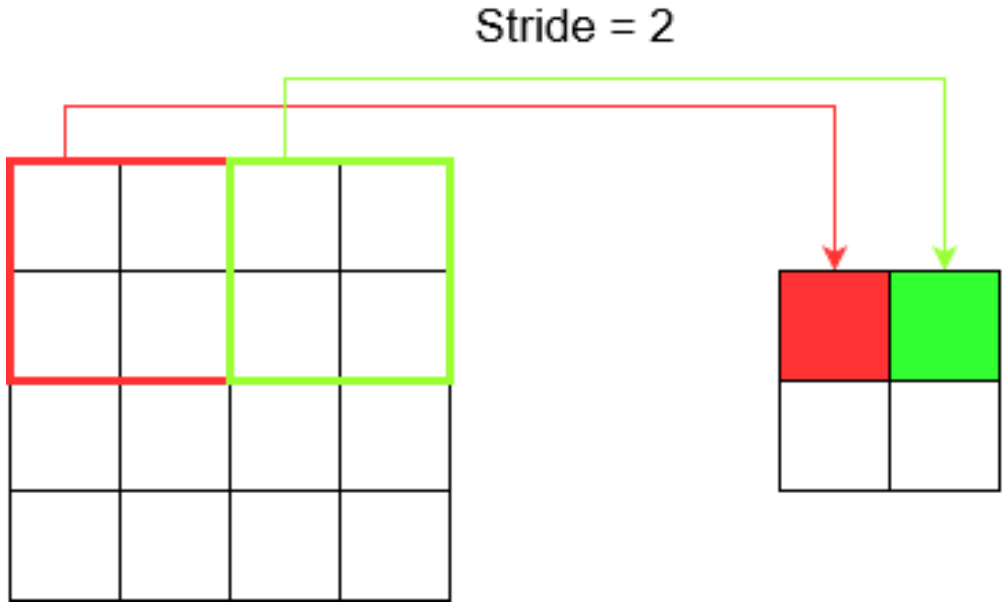
- **Depth**: corresponds to the **number of 3D filters** we would like to use. For example, if the first Convolutional Layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color. We will refer to a set of neurons that are all looking at the same region of the input as a **depth column**.



Stride

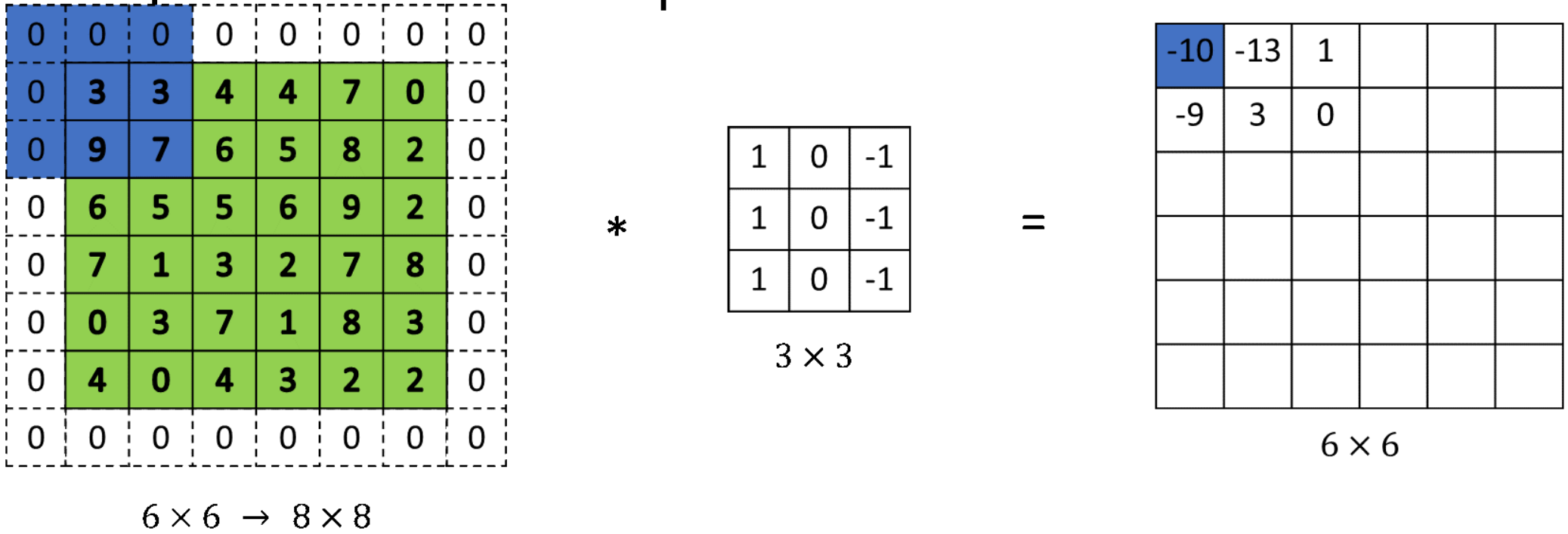
- **Stride：** When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes spatially.





Zero-padding

- Zero-padding: pad the input volume with zeros around the border
- The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes



How to calculate the out volume

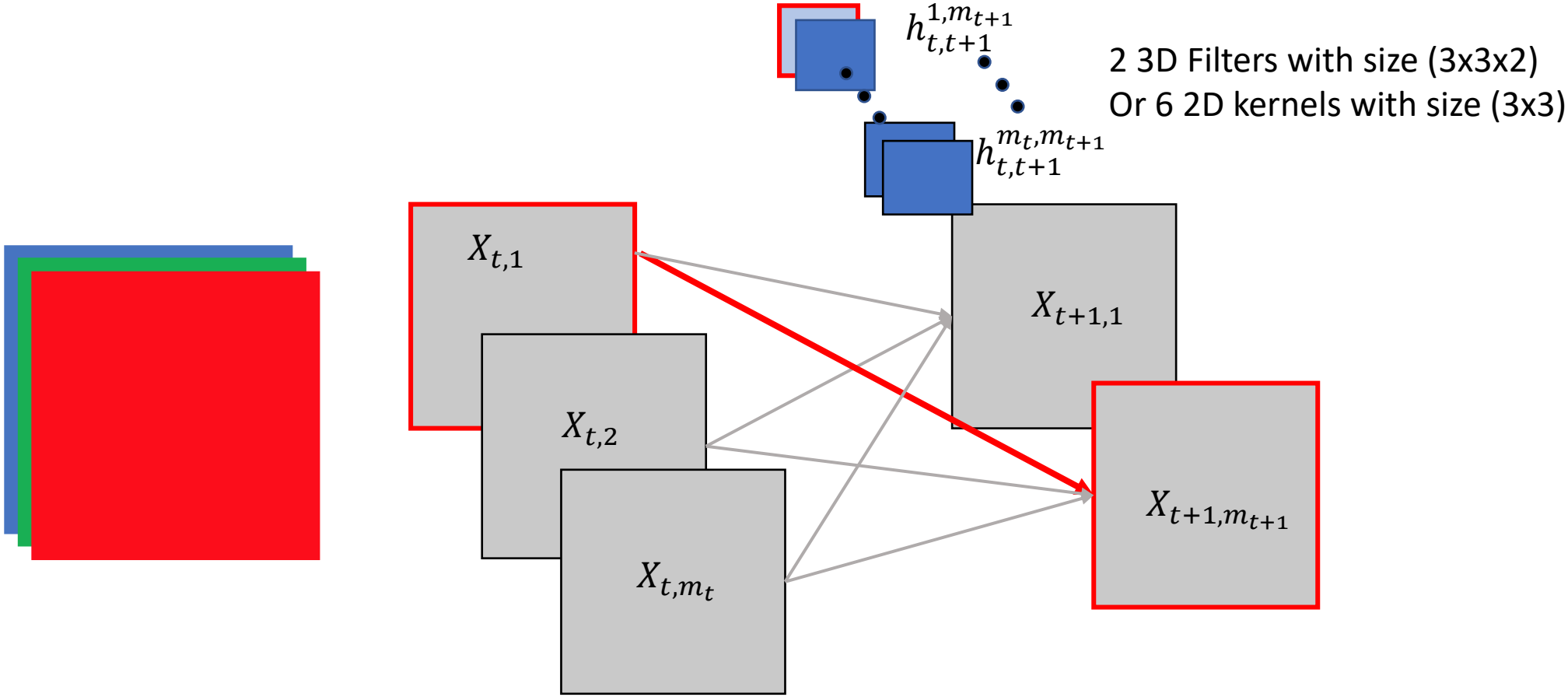
- input volume size (W*H), the receptive field size of the Conv Layer neurons (K*K), the stride with which they are applied (S), and the amount of zero padding used (P)

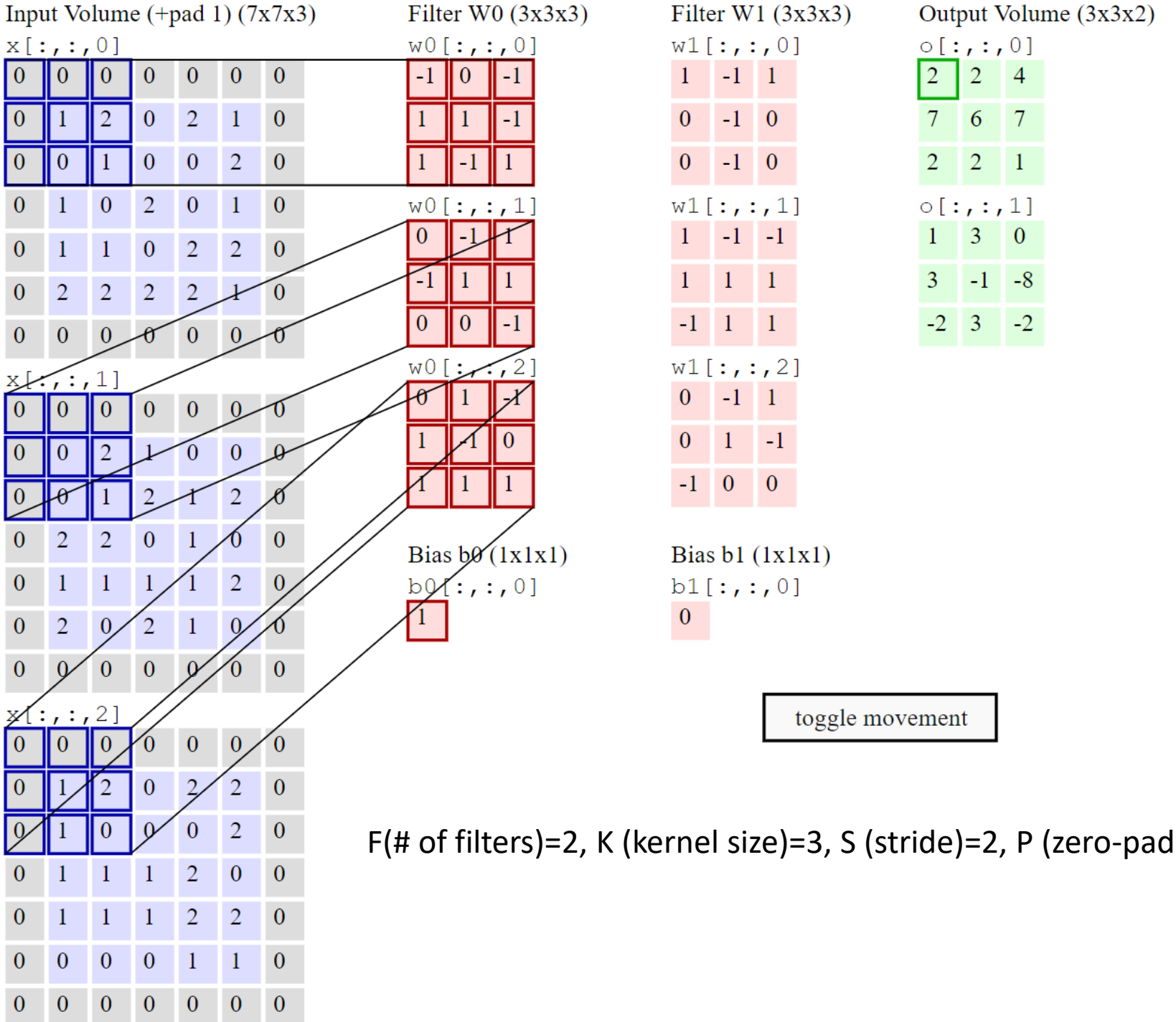
$$W_o = \frac{W - K + 2P}{S} + 1$$

$$H_0 = \frac{H - K + 2P}{S} + 1$$

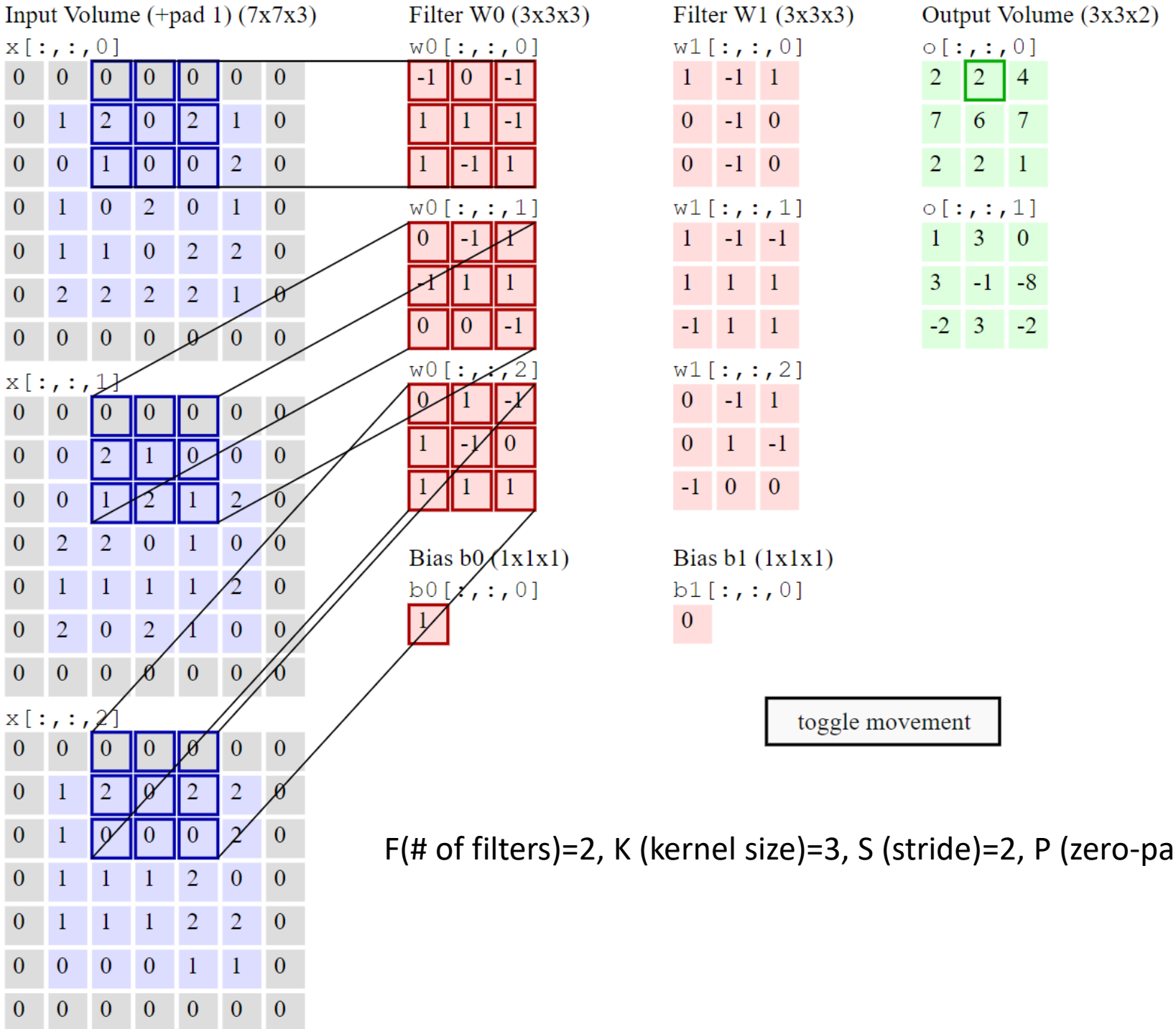
- For example for a 7x7 input and a 3x3 filter with stride 1 and pad 0 we would get a 5x5 output.

Convolutional Layer with Demo

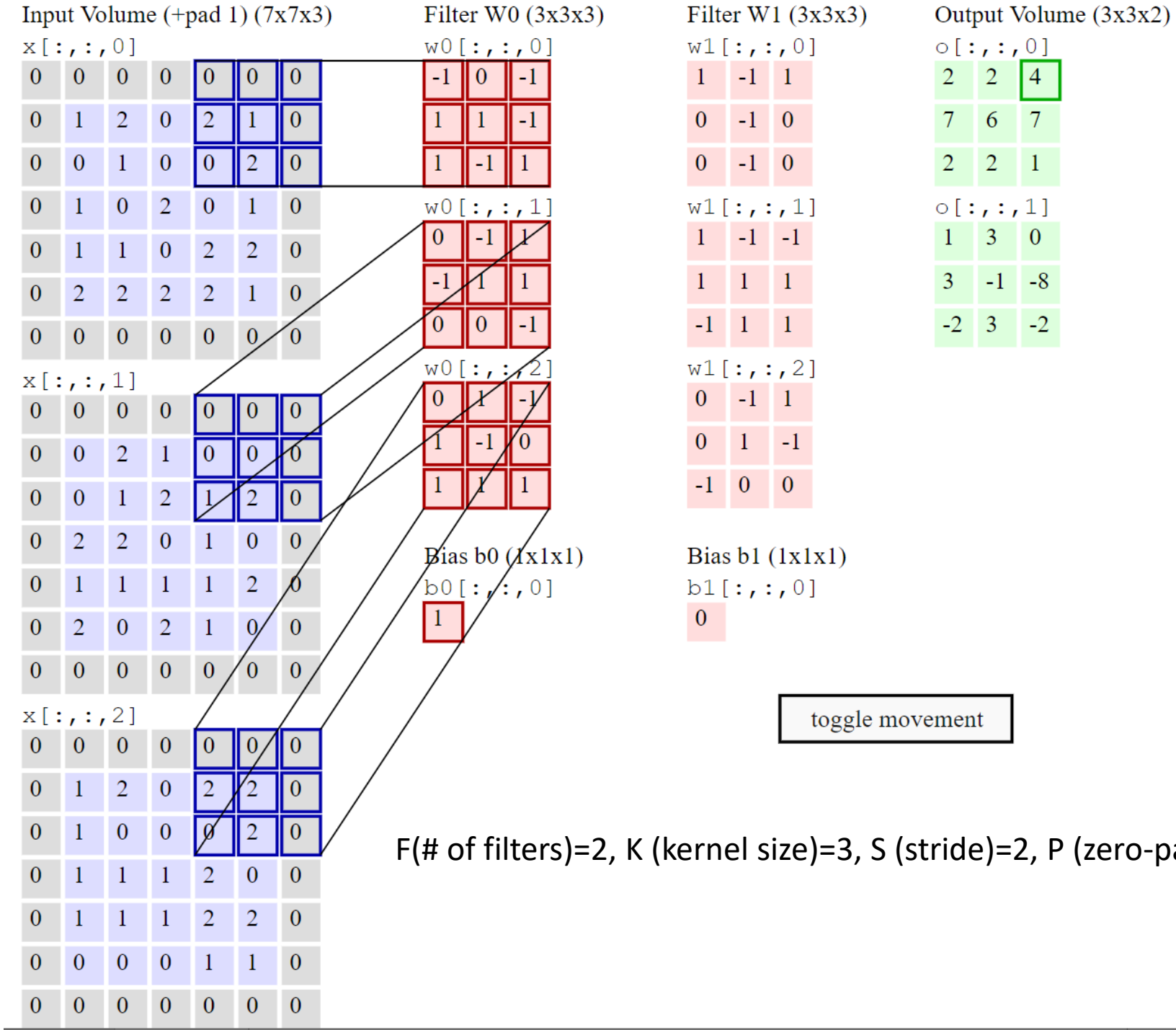




F(# of filters)=2, K (kernel size)=3, S (stride)=2, P (zero-padding)=1



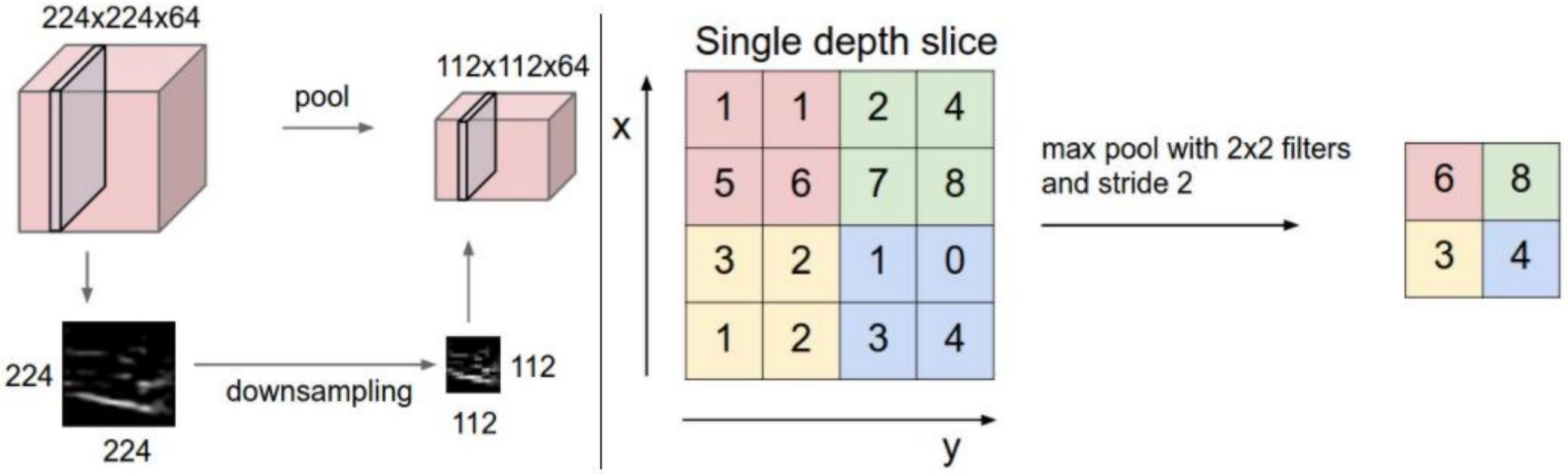
F(# of filters)=2, K (kernel size)=3, S (stride)=2, P (zero-padding)=1



F(# of filters)=2, K (kernel size)=3, S (stride)=2, P (zero-padding)=1

Pooling Layer

- It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.



Calculate output volume of a pooling layer

- input volume size (W*H), the receptive field size of the Conv Layer neurons (K*K), the stride with which they are applied (S)

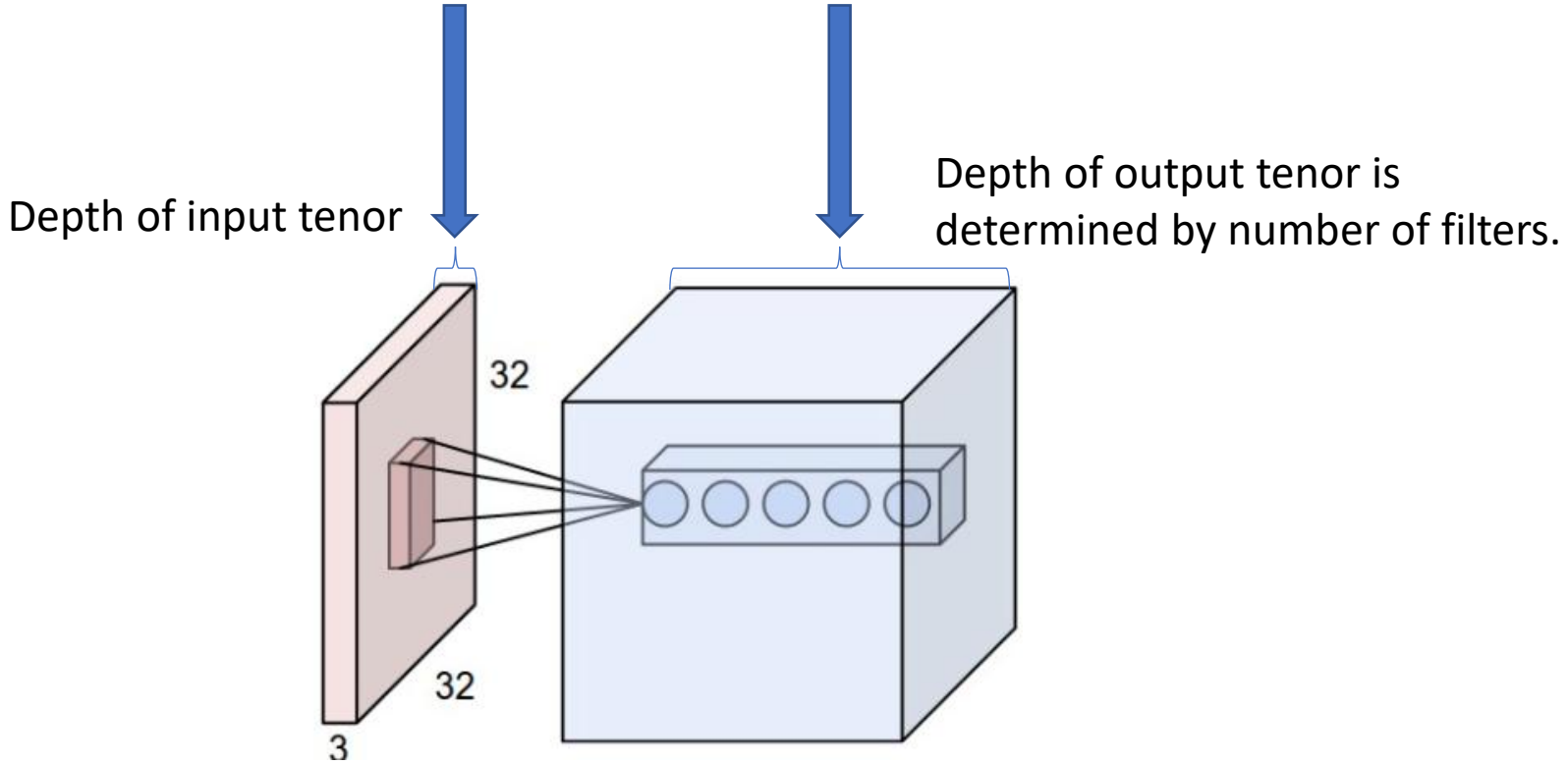
$$W_o = \frac{W - K}{S} + 1$$

$$H_0 = \frac{H - K}{S} + 1$$

- For example for a 7x7 input and a 3x3 filter with stride 2 we would get a 3x3 output.

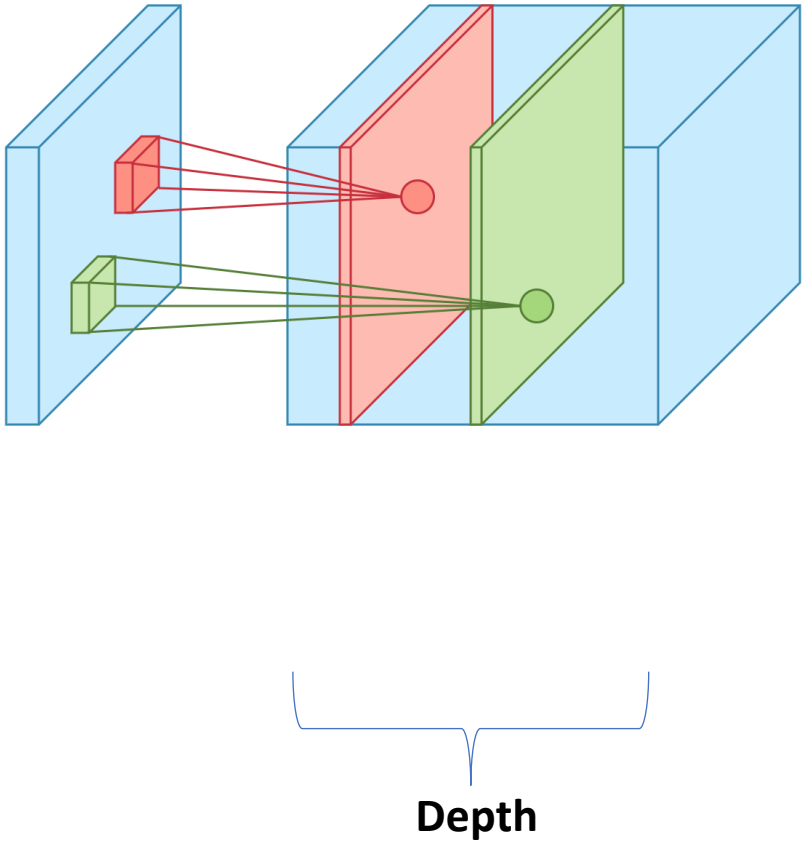
Convolutional layers in Pytorch

- nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)



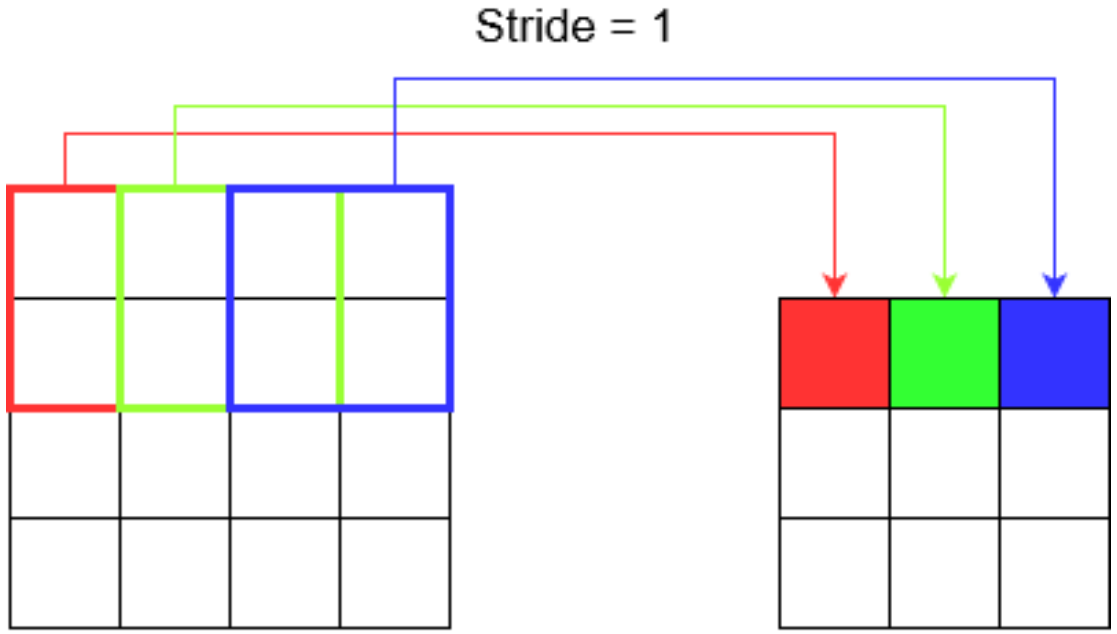
Kernel Size and Filters

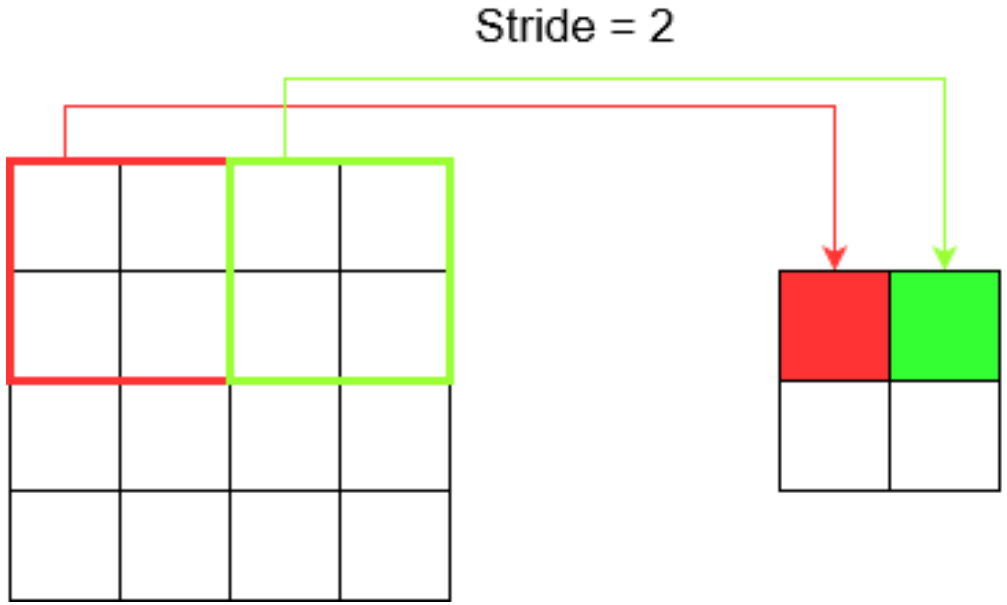
- `kernel_size:` An integer or list of 2 integers, specifying the width and height of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- `out_channels:` Integer, the dimensionality of the output space (i.e. the **depth** of the output).



Stride

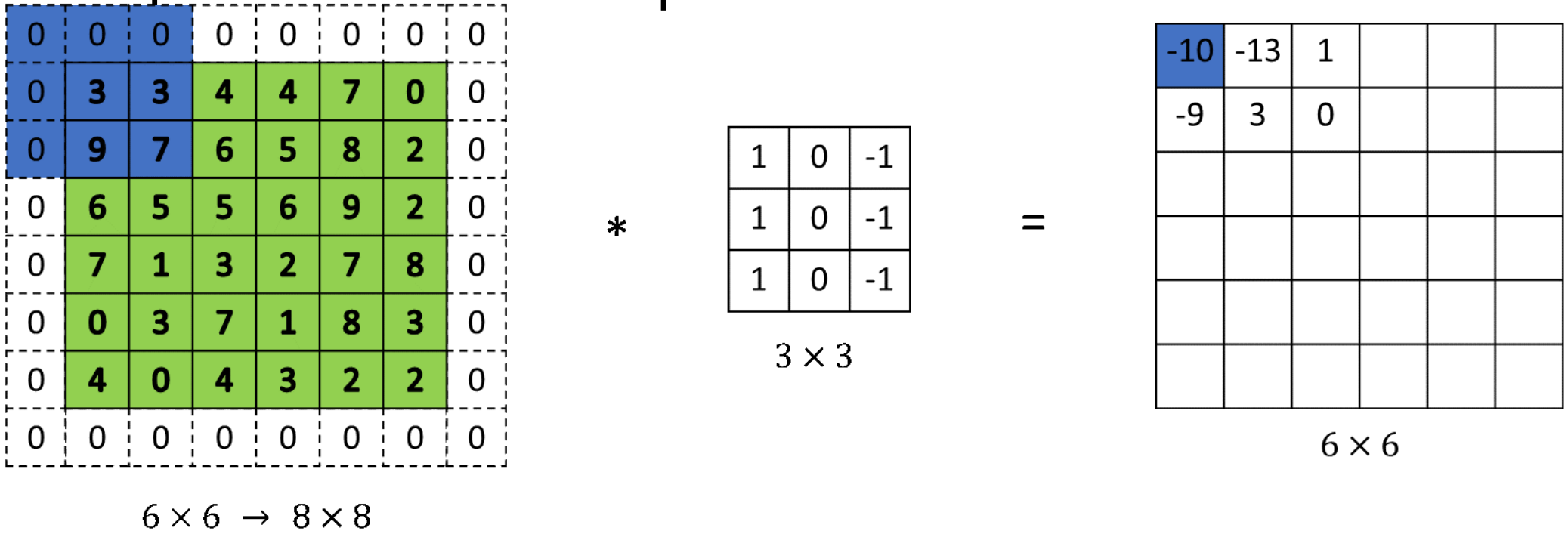
- `Strides:` An integer or list of 2 integers, specifying the strides of the convolution along the width and height. Can be a single integer to specify the same value for all spatial dimensions.





Review: Zero-padding

- Zero-padding: pad the input volume with zeros around the border
- The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes



Add zero-padding

- padding:
  - int, or list of 2 ints, or list of 2 lists of 2 ints.
  - If int: the same symmetric padding is applied to width and height.
  - If list of 2 ints: interpreted as two different symmetric padding values for height and width: (symmetric_height_pad, symmetric_width_pad).
  - If list of 2 lists of 2 ints: interpreted as ((top_pad, bottom_pad), (left_pad, right_pad))

Pooling layers in Pytorch

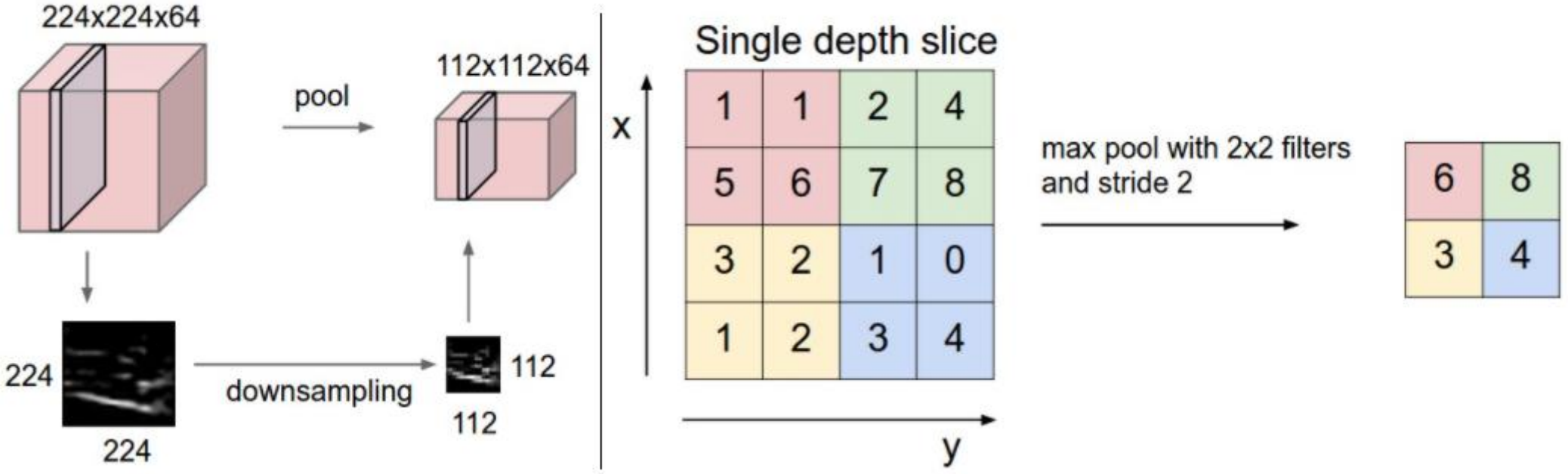
- `nn.MaxPool2d(kernel_size, stride=None, padding=0)`

- `nn. AvgPool2d(kernel_size, stride=None, padding=0)`

Review: Pooling Layer

- It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.

padding



Layers used to build ConvNets

- We use three main types of layers to build ConvNet architectures:
- **Convolutional Layer**,
- **Pooling Layer**,
- **Fully-Connected Layer** (exactly as seen in regular Neural Networks).

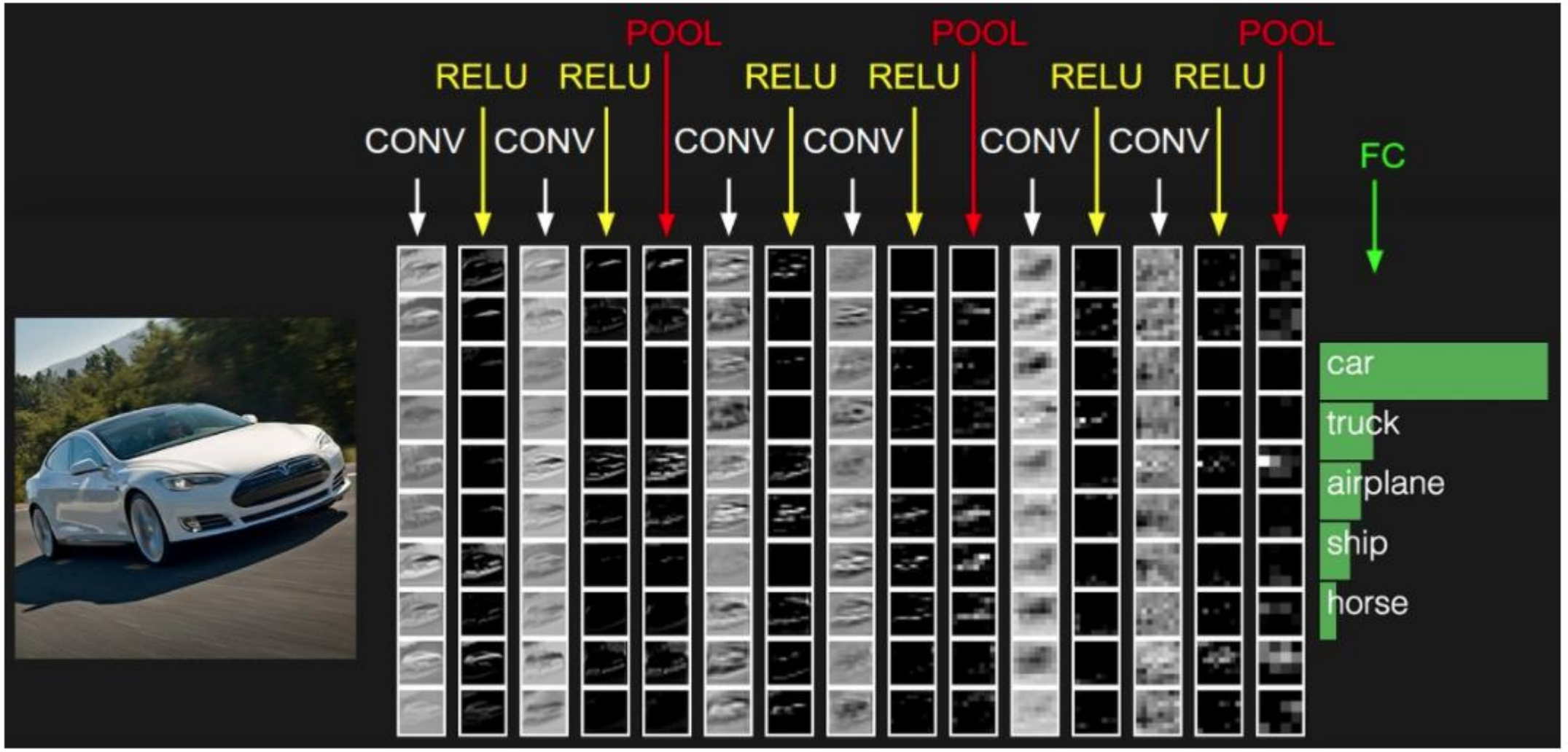
Example

- 1. INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- 2. CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
- 3. RELU layer will apply an elementwise activation function, such as the max(0,x) thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).

Example

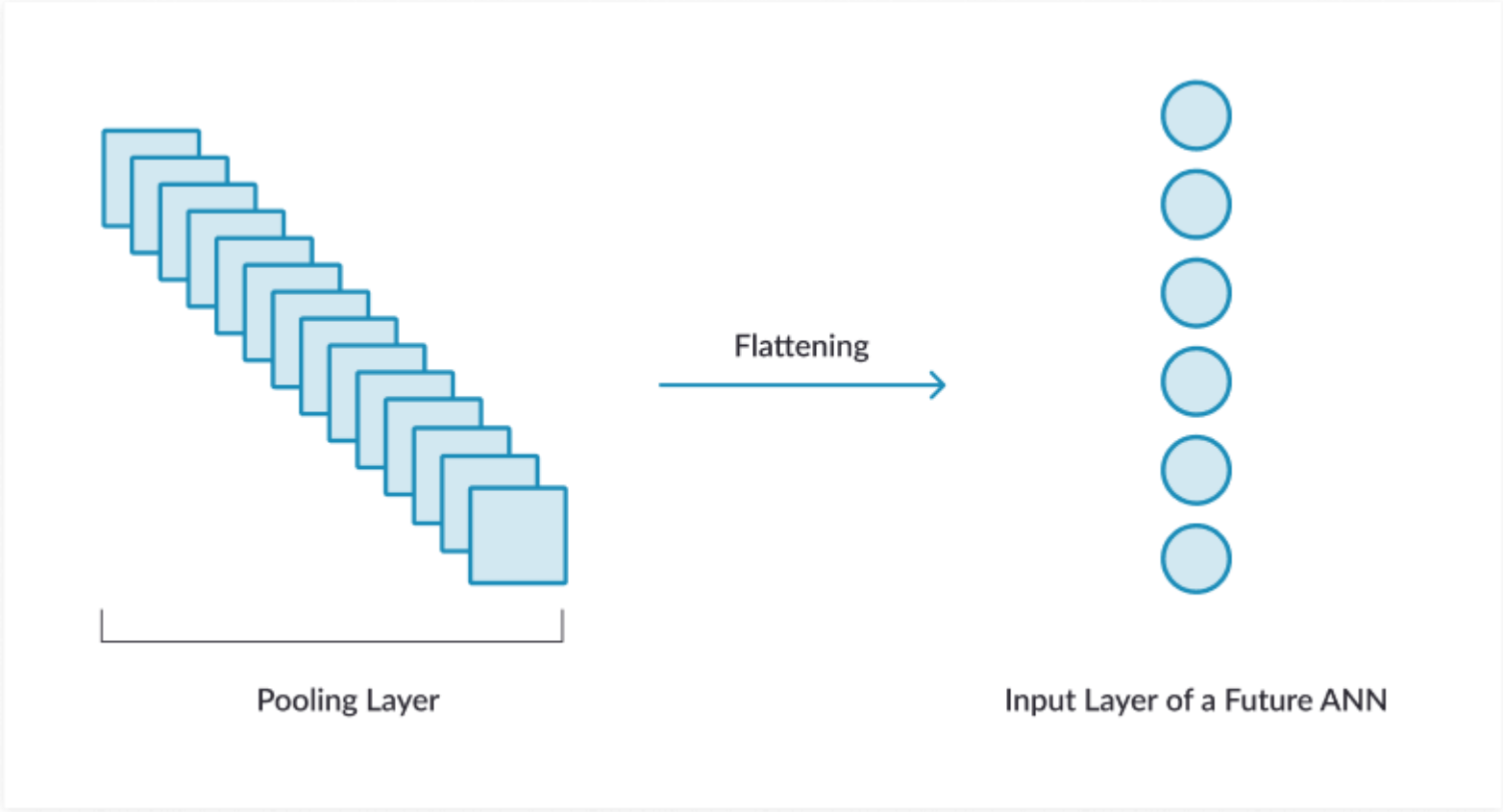
- 4. POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

Overview Summary



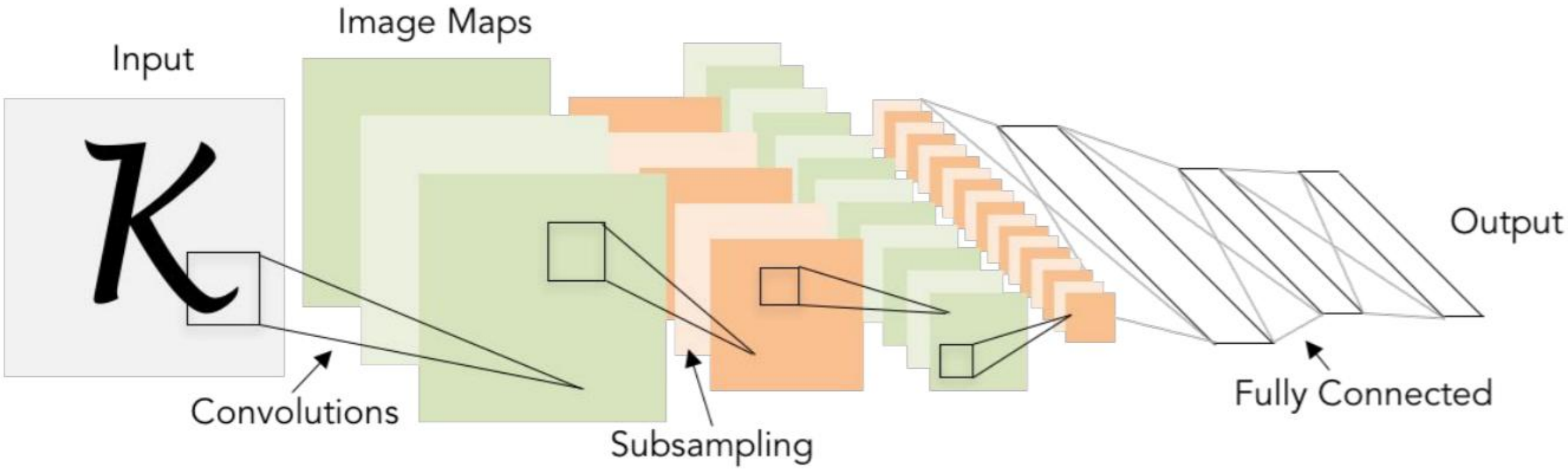
Flatten a layer

- `x.view(x.shape[0], -1)`



Review: LeNet-5

[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

History

- LeNet5 was one of the earliest convolutional neural networks and promoted the development of deep learning. Since 1988, after years of research and many successful iterations, the pioneering work has been named LeNet5

- In 1989, Yann LeCun et al. at Bell Labs first applied the backpropagation algorithm to practical applications, and believed that the ability to learn network generalization could be greatly enhanced by providing constraints from the task's domain.

Timeline

1989	Yann LeCun et al. proposed the original form of LeNet
1989	Yann LeCun proves that minimizing the number of free parameters in neural networks can enhance the generalization ability of neural networks.
1990	Their paper describes the application of backpropagation networks in handwritten digit recognition once again
1998	They reviewed various methods applied to handwritten character recognition and compared them with standard handwritten digit recognition benchmarks. The results show that convolutional neural networks outperform all other models.

Turing Award Won by 3 Pioneers in Artificial Intelligence

- 2018 Turing Award Won by Yann LeCun, Geoffrey Hinton and Yoshua Bengio for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing.

LeNet-5 layers:

- 1. Convolution #1. Input = 28x28x1. Output = 24x24x6.
- 2. Pooling #1. Input = 24x24x6. Output = 12x12x6.
- 3. Convolution #2. Input = 12x12x6. Output = 8x8x16.
- 4. Pooling #2. Input = 8x8x16. Output = 4x4x16.

LeNet-5 layers:

- 5. Fully Connected #1. Input = 4x4x16. Output = 120
- 6. Fully Connected #2. Input = 120. Output = 84
- 7. Output 10