CONVOLUTIONAL NEURAL NETWORK

Sourav Karmakar

souravkarmakar29@gmail.com

What Shall We Learn?

- ❑ Definition of Convolution Operation
- ❑ 1D convolution and its example
- ❑ 2D convolution (spatial convolution)
- ❑ 2D convolution operation on images
- ❑ Effect of applying different filters on images
- ❑ 2D convolution operation – Spatial dimension
- ❑ Pooling operation
- ❑ Convolutional Neural Network – Complete architecture
- ❑ Application of CNN on image classification (hands on)

Convolution Operation : Definition

Let,

- $x(t)$: Signal (a time varying function)
- $w(t)$: Weighing function (kernel or filter)
- $s(t)$: Smoothed signal / filtered signal obtained by convoluting $x(t)$ with $w(t)$

- **Continuous time Convolution Operation:**

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

- **Discrete time Convolution Operation:**

$$s(t) = x(t) * w(t) = \sum_{\tau} x(\tau)\, w(t-\tau) = \sum_{\tau} x(t-\tau)\, w(\tau)$$

- Note that the sign $*$ stands for convolution and not multiplication
- This is also known as 1-D convolution as there is only one variable here (i.e. time to be specific)

1D Convolution Operation : Example

- **Example: 1** Let us consider the following time varying signal and the kernel / filter:

	t:	0	1	2	3	4
Signal	$x(t)$:	2	3	10	1	3
Kernel	$w(t)$:	1	2	3		
Filtered Signal	$s(t)$:	2	7	22	30	35

Now we know that for discrete time, $s(t) = \sum_\tau x(\tau)\, w(t-\tau)$

- Thus, for $t = 0$, $s(0) = \sum_\tau x(\tau)\, w(0-\tau) = x(0)w(0) + x(1)w(-1) + \cdots = x(0)w(0) = 2$
  We have assumed $x(\tau) = 0\ for\ \tau < 0\ \&\ \tau > 4\ \ and\ \ w(\tau) = 0\ for\ \tau < 0\ \&\ \tau > 2$
- for $t = 1$, $s(1) = \sum_\tau x(\tau)\, w(1-\tau) = x(0)w(1) + x(1)w(0) = 4 + 3 = 7$
- for $t = 2$, $s(2) = \sum_\tau x(\tau)\, w(2-\tau) = x(0)w(2) + x(1)w(1) + x(2)w(0) = 6 + 6 + 10 = 22$
- for $t = 3$, $s(3) = \sum_\tau x(\tau)\, w(3-\tau) = x(0)w(3) + x(1)w(2) + x(2)w(1) + x(3)w(0) = 30$
- for $t = 4$, $s(4) = \sum_\tau x(\tau)\, w(4-\tau) = x(2)w(2) + x(3)w(1) + x(4)w(0) = 35$

1D Convolution Operation : Examples

- **Example: 2**

t:	0	1	2	3	4
$x(t)$:	2	3	10	1	3
$w(t)$:	0	2	0		
$s(t)$:	0	4	6	20	2

Observation: $s(t)$ is time shifted and enhanced version of $x(t)$

- **Example: 3**

t:	0	1	2	3	4
$x(t)$:	2	3	10	1	3
$w(t)$:	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$		
$s(t)$:	$\frac{2}{3}$	$\frac{5}{3}$	5	$\frac{14}{3}$	$\frac{14}{3}$

Observation: $s(t)$ is a blurred / averaged version of $x(t)$

- **Example: 4**

t:	0	1	2	3	4
$x(t)$:	2	3	10	1	3
$w(t)$:	-1	1	-1		
$s(t)$:	-2	-1	-9	6	-24

Observation: If we apply ReLU on the obtained $s(t)$ then the obtained signal after ReLU is:

$$\tilde{s}(t) = 0 \quad 0 \quad 0 \quad 6 \quad 0$$

This one acts like a ***peak-detector***

2D Convolution Operation

For 2D cases, most common example being the Images, the input signal is a function of 2D space.
Let,

- $f(x, y)$ is the input signal which is the function of space
- $w(x, y)$ is the kernel / weight function also called spatial / 2D filter
- $s(x, y)$ is the smoothed / filtered version of input signal then

- **Continuous 2D Convolution operation:**

$$s(x, y) = \int_u \int_v f(u, v)\, w(x - u, y - v)\, du\ dv = \int_u \int_v f(x - u, y - v)\, w(u, v)\, du\, dv$$

- **Discrete 2D Convolution operation:**

$$s(x, y) = \sum_u \sum_v f(u, v)\, w(x - u, y - v) = \sum_u \sum_v f(x - u, y - v)\, w(u, v)$$

Note that the kernel has been flipped with respect to the input signal in order to obtain commutative property

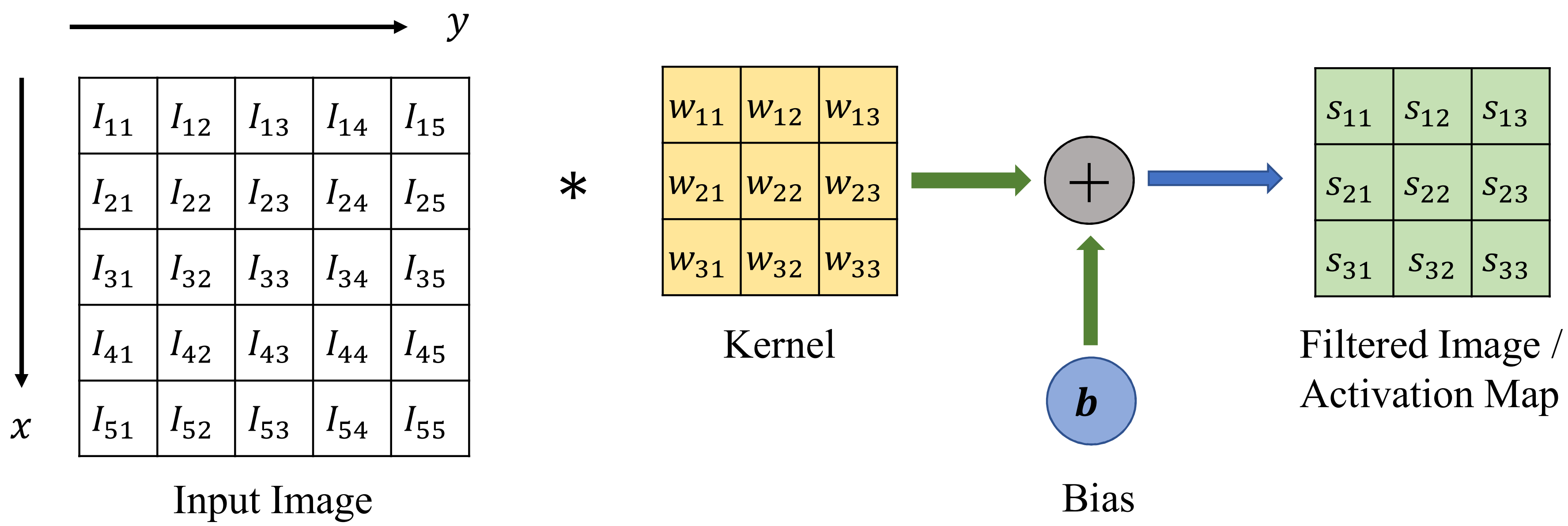
2D Convolution Operation

- Kernel flipping is not so important operation in terms of building convolutional neural networks.
- Hence, many Machine Learning libraries implement a similar kind of operation without kernel flipping and call it convolution. Mathematically for discrete case this looks like:

$$s(x,y) = \sum_{u}\sum_{v} f(x+u, y+v)\, w(u,v)$$

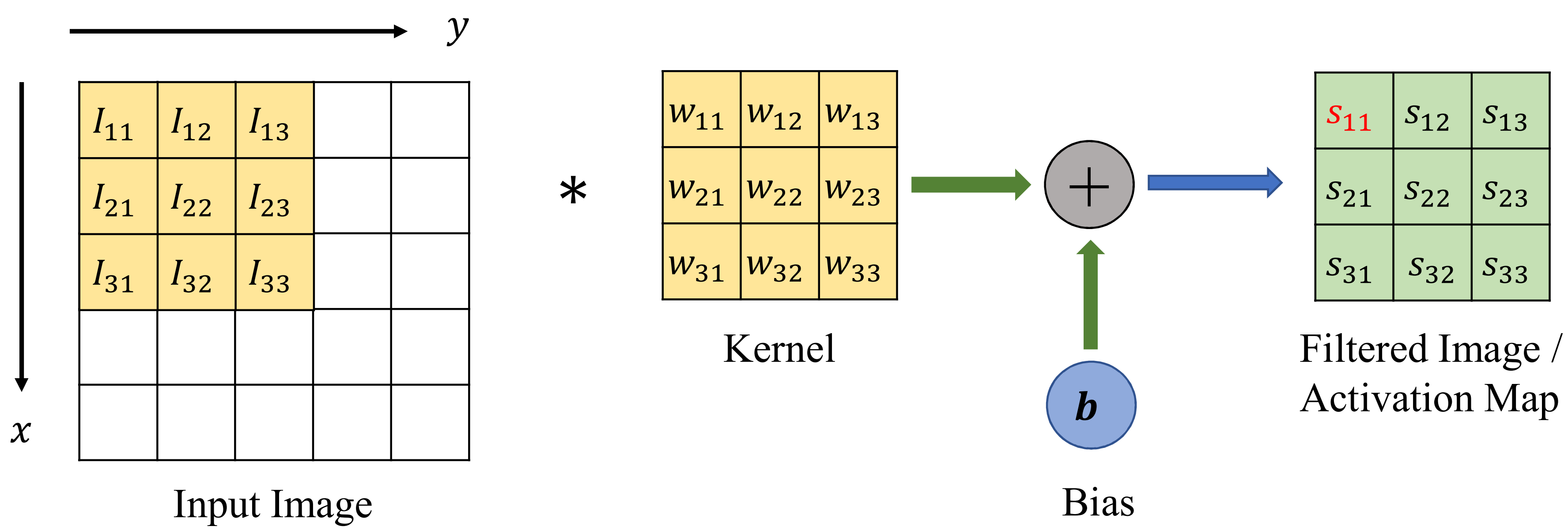
- Strictly speaking this operation is known as 2D **cross-correlation**. However many literature as well as ML libraries consider this as 2D convolution operation.
- From now on we shall consider this as convolution operation, noting that it has striking resemblance with the original convolution operation without kernel flipping.
- Hence, convolution operation can be considered as repetitive dot product (element wise product) between the filter / kernel with a small portion of the input signal.
- We shall now see how this manifests in case of images, first grayscale and then coloured (RGB) images .

2D Convolution Operation on Images



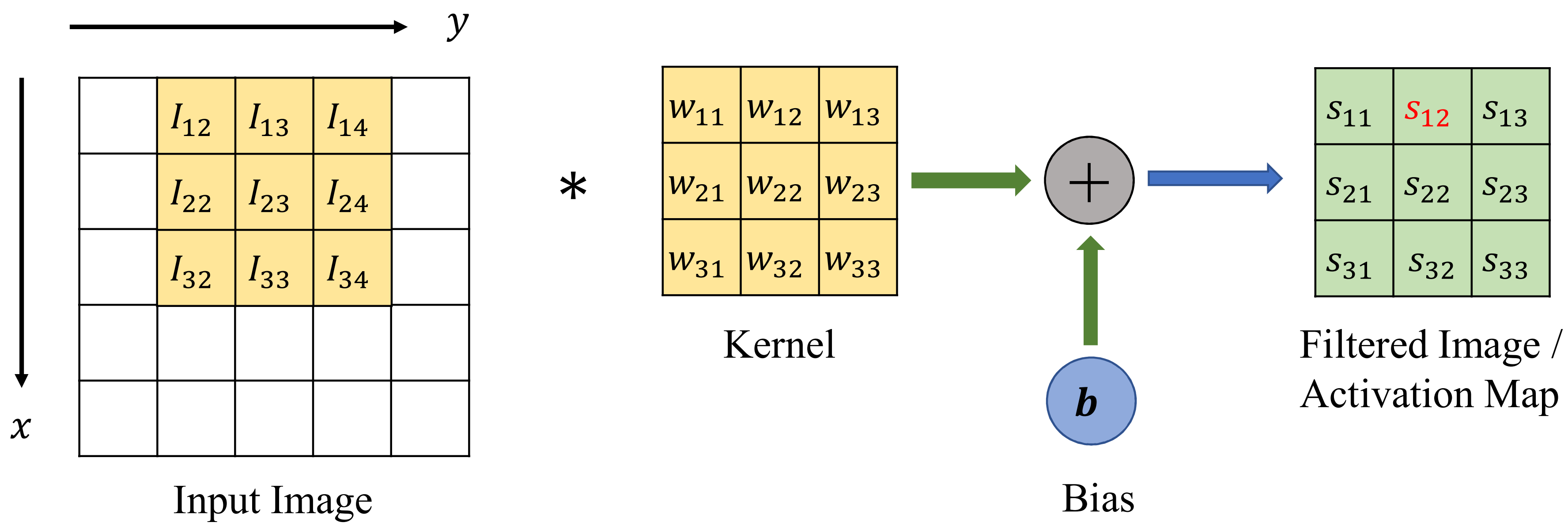
- Assume that the image is grayscale, which means it contains only one channel
- Usually the kernel dimensions are odd. For example: 3×3, 5×5, 7×7 etc.
- Usually the spatial dimensions of activation map is smaller than that of input image without padding. This shall be discussed with details in the later slides.

2D Convolution Operation on Images



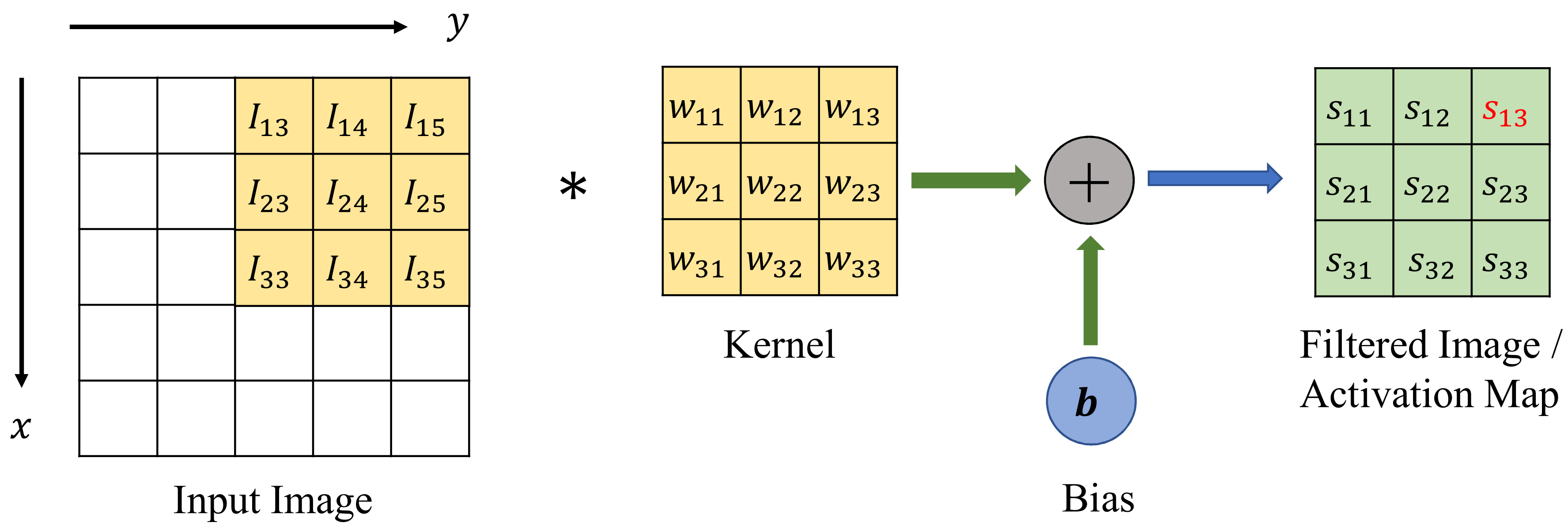
$$s_{11} = I_{11}w_{11} + I_{12}w_{12} + I_{13}w_{13} + I_{21}w_{21} + I_{22}w_{22} + I_{23}w_{23} + I_{31}w_{31} + I_{32}w_{32} + I_{33}w_{33} + b$$

2D Convolution Operation on Images



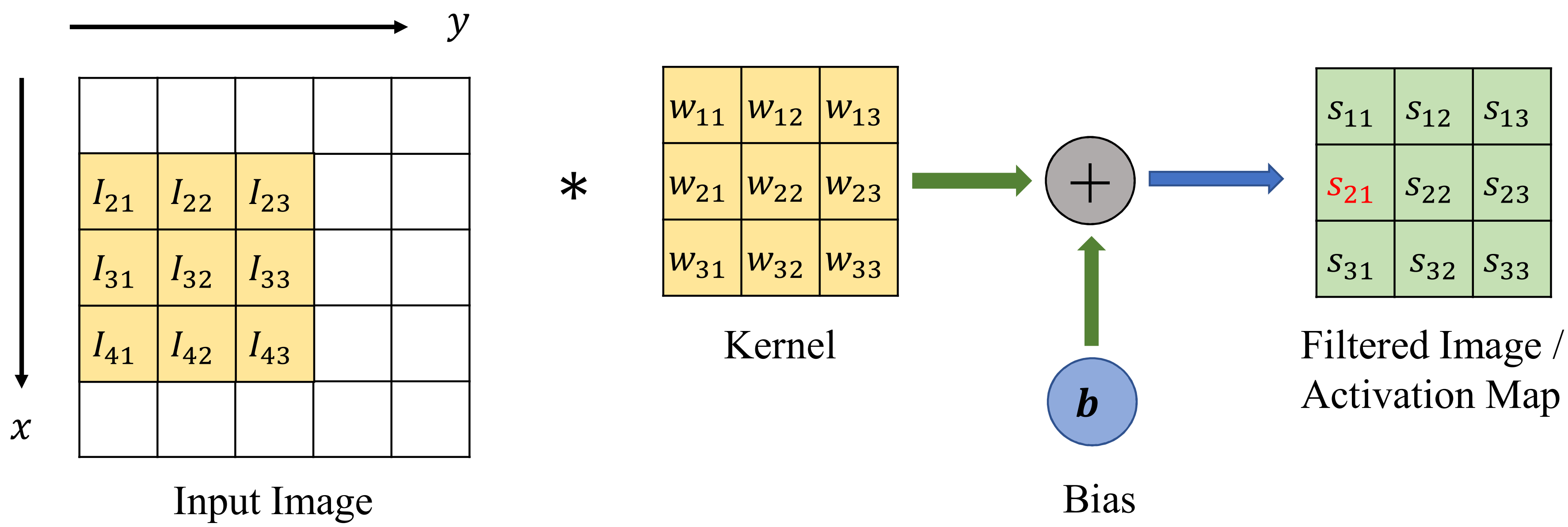
$$s_{12} = I_{12}w_{11} + I_{13}w_{12} + I_{14}w_{13} + I_{22}w_{21} + I_{23}w_{22} + I_{24}w_{23} + I_{32}w_{31} + I_{33}w_{32} + I_{34}w_{33} + b$$

2D Convolution Operation on Images



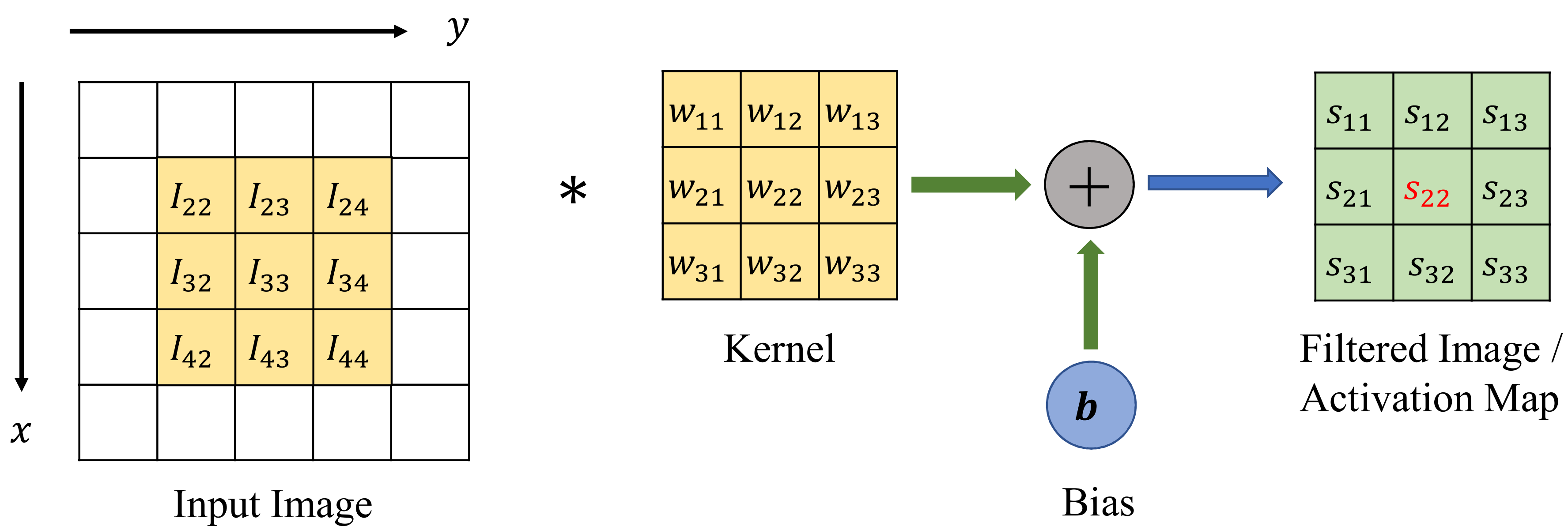
$$s_{13} = I_{13}w_{11} + I_{14}w_{12} + I_{15}w_{13} + I_{23}w_{21} + I_{24}w_{22} + I_{25}w_{23} + I_{33}w_{31} + I_{34}w_{32} + I_{35}w_{33} + b$$

2D Convolution Operation on Images



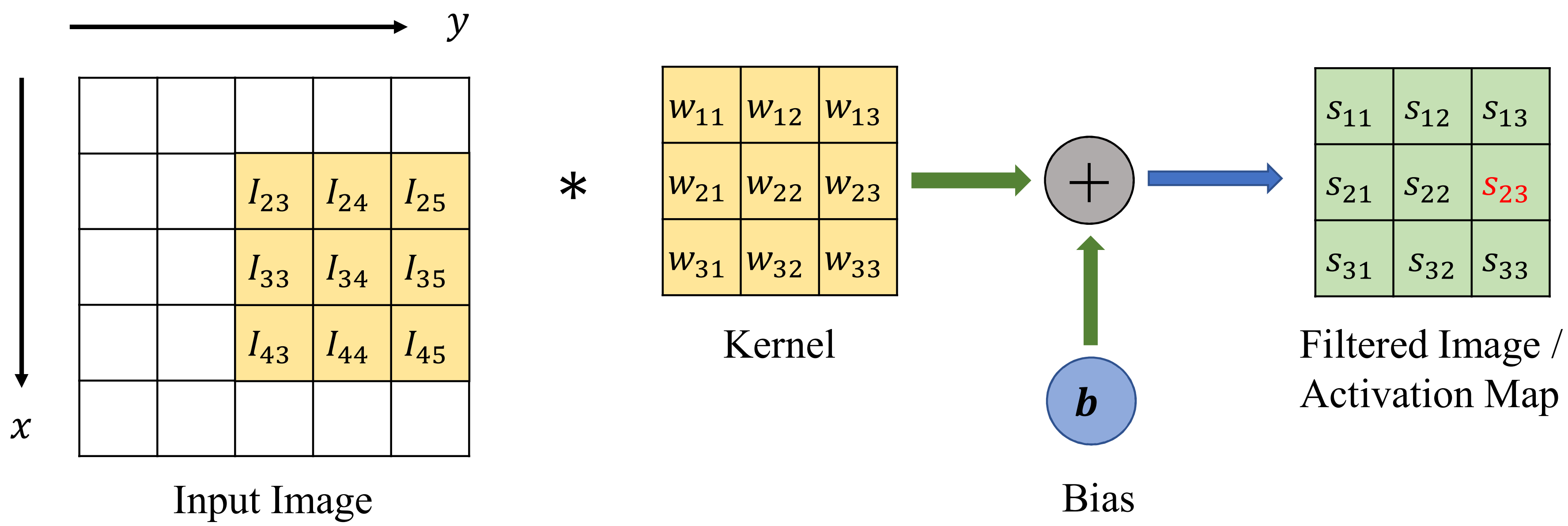
$$s_{21} = I_{21}w_{11} + I_{22}w_{12} + I_{23}w_{13} + I_{31}w_{21} + I_{32}w_{22} + I_{33}w_{23} + I_{41}w_{31} + I_{42}w_{32} + I_{43}w_{33} + b$$

2D Convolution Operation on Images



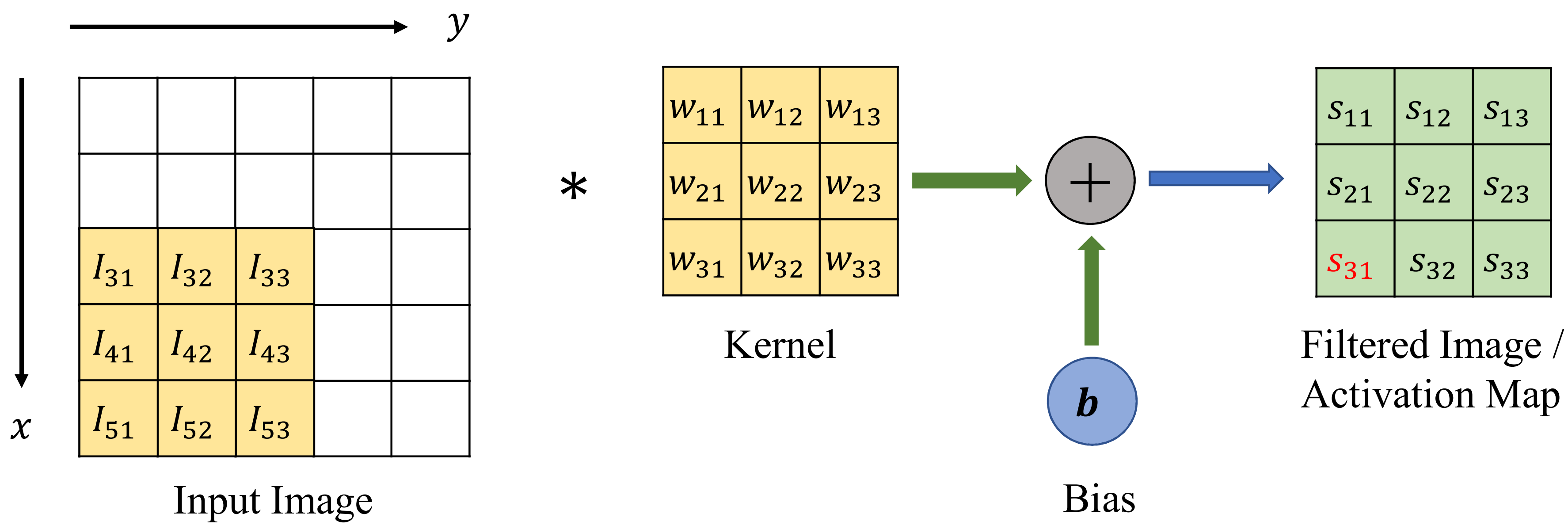
$$s_{22} = I_{22}w_{11} + I_{23}w_{12} + I_{24}w_{13} + I_{32}w_{21} + I_{33}w_{22} + I_{34}w_{23} + I_{42}w_{31} + I_{43}w_{32} + I_{44}w_{33} + b$$

2D Convolution Operation on Images



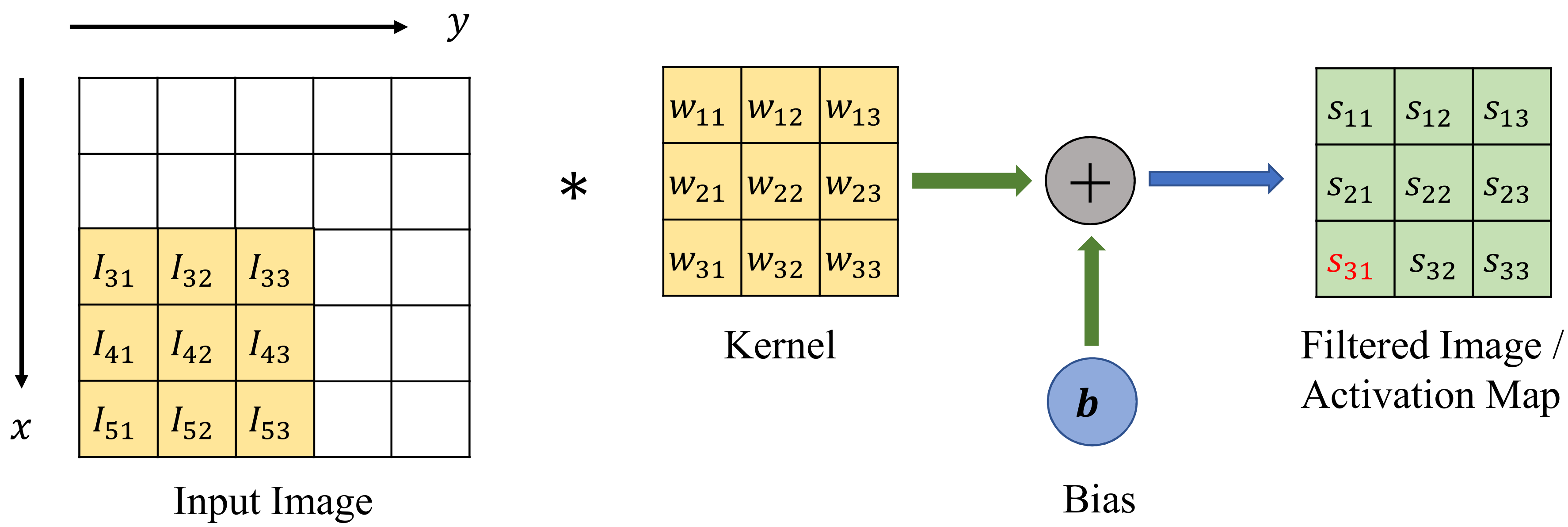
$$s_{23} = I_{23}w_{11} + I_{24}w_{12} + I_{25}w_{13} + I_{33}w_{21} + I_{34}w_{22} + I_{35}w_{23} + I_{43}w_{31} + I_{44}w_{32} + I_{45}w_{33} + b$$

2D Convolution Operation on Images



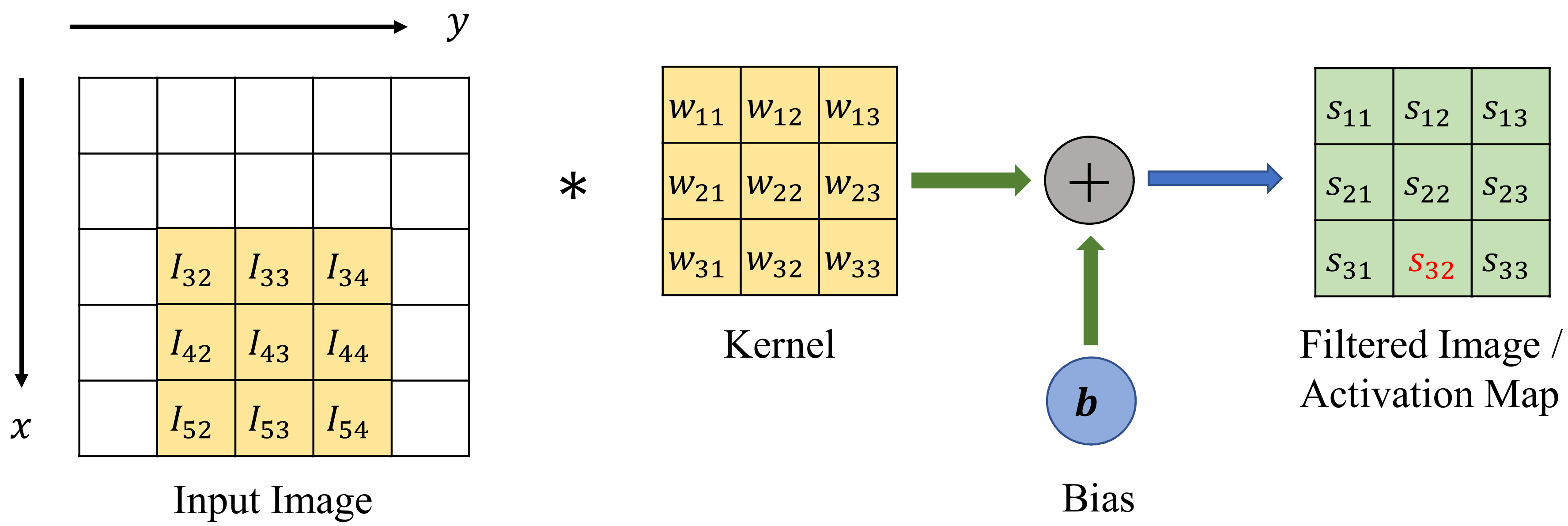
$$s_{31} = I_{31}w_{11} + I_{32}w_{12} + I_{33}w_{13} + I_{41}w_{21} + I_{42}w_{22} + I_{43}w_{23} + I_{51}w_{31} + I_{52}w_{32} + I_{53}w_{33} + b$$

2D Convolution Operation on Images



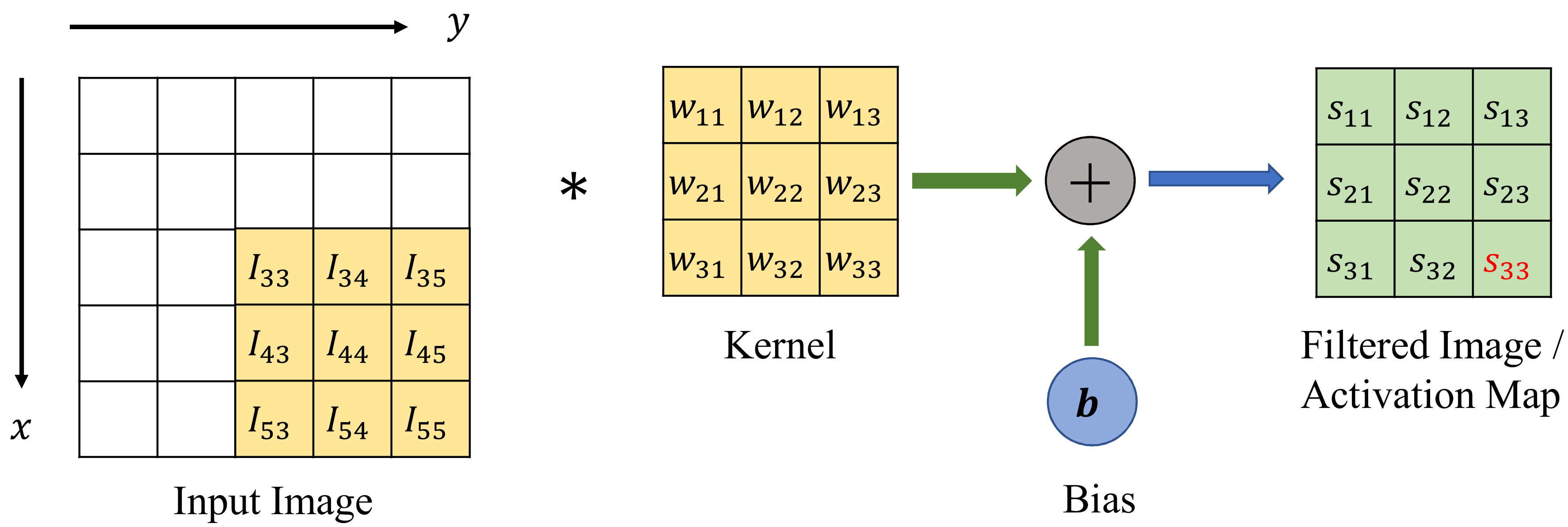
$$s_{31} = I_{31}w_{11} + I_{32}w_{12} + I_{33}w_{13} + I_{41}w_{21} + I_{42}w_{22} + I_{43}w_{23} + I_{51}w_{31} + I_{52}w_{32} + I_{53}w_{33} + b$$

2D Convolution Operation on Images



$$s_{32} = I_{32}w_{11} + I_{33}w_{12} + I_{34}w_{13} + I_{42}w_{21} + I_{43}w_{22} + I_{44}w_{23} + I_{52}w_{31} + I_{53}w_{32} + I_{54}w_{33} + b$$

2D Convolution Operation on Images



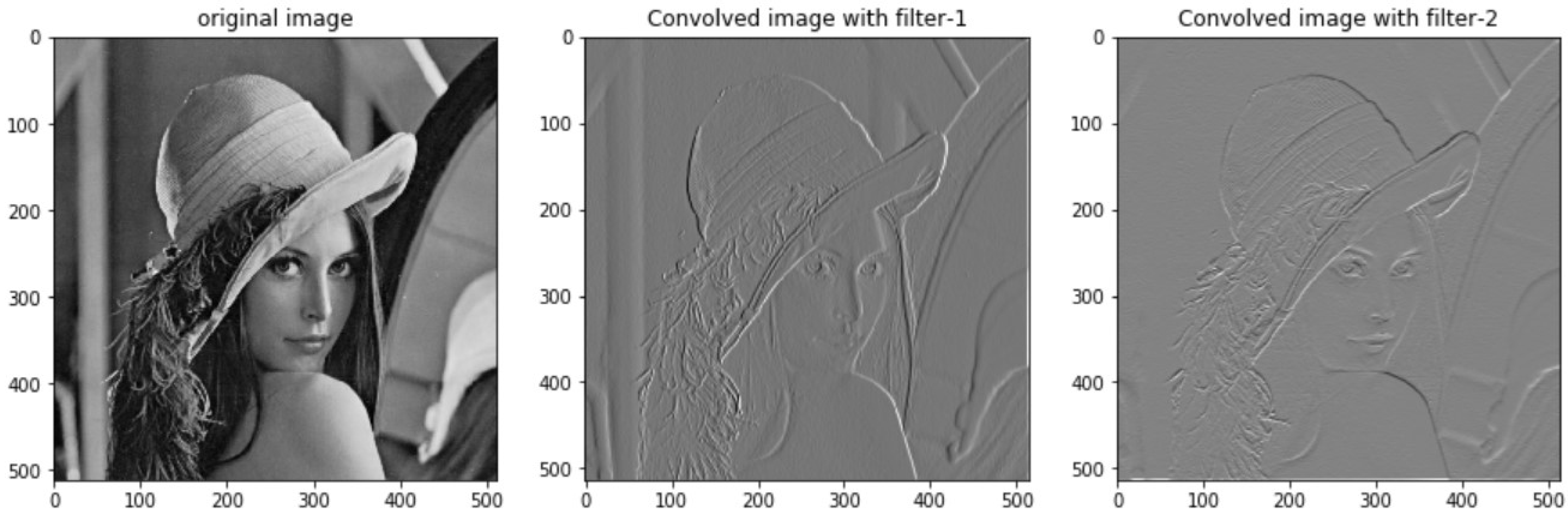
$$s_{33} = I_{33}w_{11} + I_{34}w_{12} + I_{35}w_{13} + I_{43}w_{21} + I_{44}w_{22} + I_{45}w_{23} + I_{53}w_{31} + I_{54}w_{32} + I_{55}w_{33} + b$$

In general for a 3×3 filter:

$$s_{ij} = I_{ij}w_{11} + I_{i(j+1)}w_{12} + I_{i(j+2)}w_{13} + I_{(i+1)j}w_{21} + I_{(i+1)(j+1)}w_{22} + I_{(i+1)(j+2)}w_{23} + I_{(i+2)j}w_{31} + I_{(i+2)(j+1)}w_{32} + I_{(i+2)(j+2)}w_{33}$$

Effect of applying Filters on Image

- The idea of applying filter to extract information from images is a very popular and well known technique in Image Processing. For example consider the following image:



Original Image

Convolved with kernel

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Convolved with kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

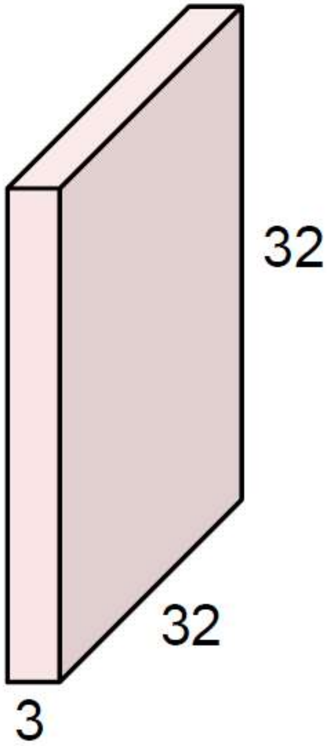
Effect of applying Filters on Image

- The idea of applying filter to extract information from images is a very popular and well known technique in Image Processing.
- Different filters captures different aspects of the image. Some captures vertical lines, some horizontal lines, some peak points etc.
- The only difference in the idea of filters in convolution from traditional image processing perspective to that of in the perspective of Convolutional Neural Network is, here in CNN the filter/kernel parameters are **learned from the dataset**, instead of being user specified.
- During the training of CNN, all the weights/parameters are randomly initialized. The weight parameters are learned from given dataset using back-propagation learning algorithm, optimizing suitable loss function.
- The kernels thus learned from the given dataset will obtain the feature maps from the input image and these feature maps in turn are helpful for certain task (like classification)

Now we shall see how we extend the idea of 2D convolution for coloured image (multi-channel)

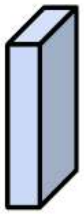
2D Convolution Operation on Images

32x32x3 image



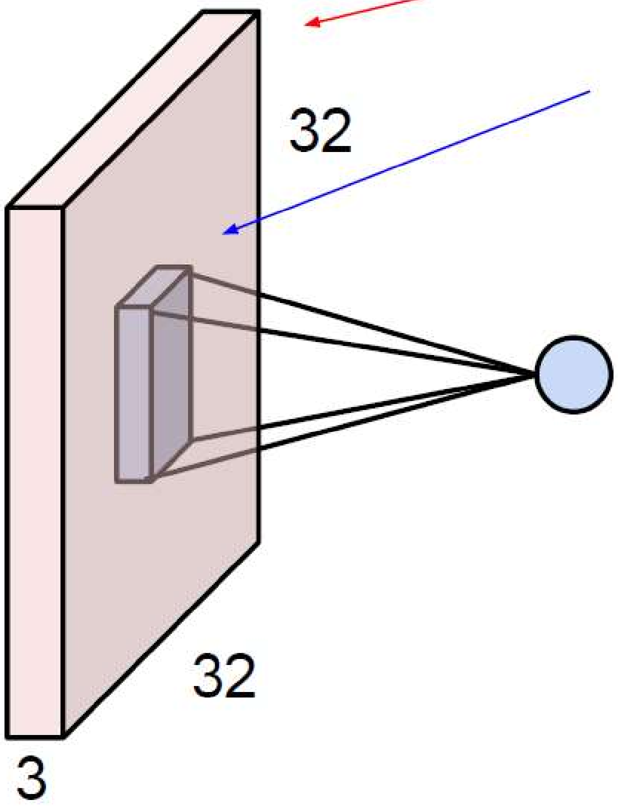
Filters always extend the full depth of the input volume

5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

2D Convolution Operation on Images



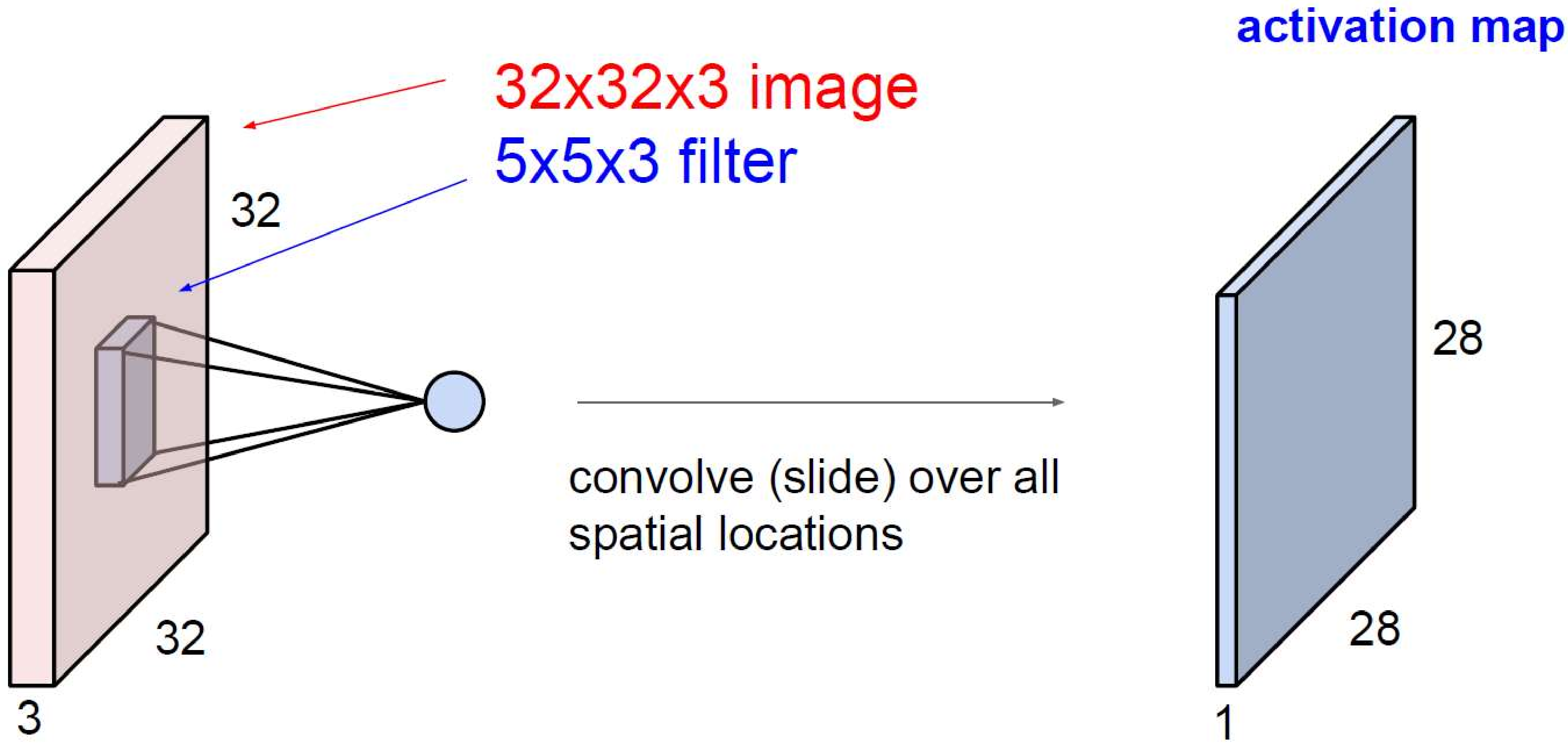
32x32x3 image

5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

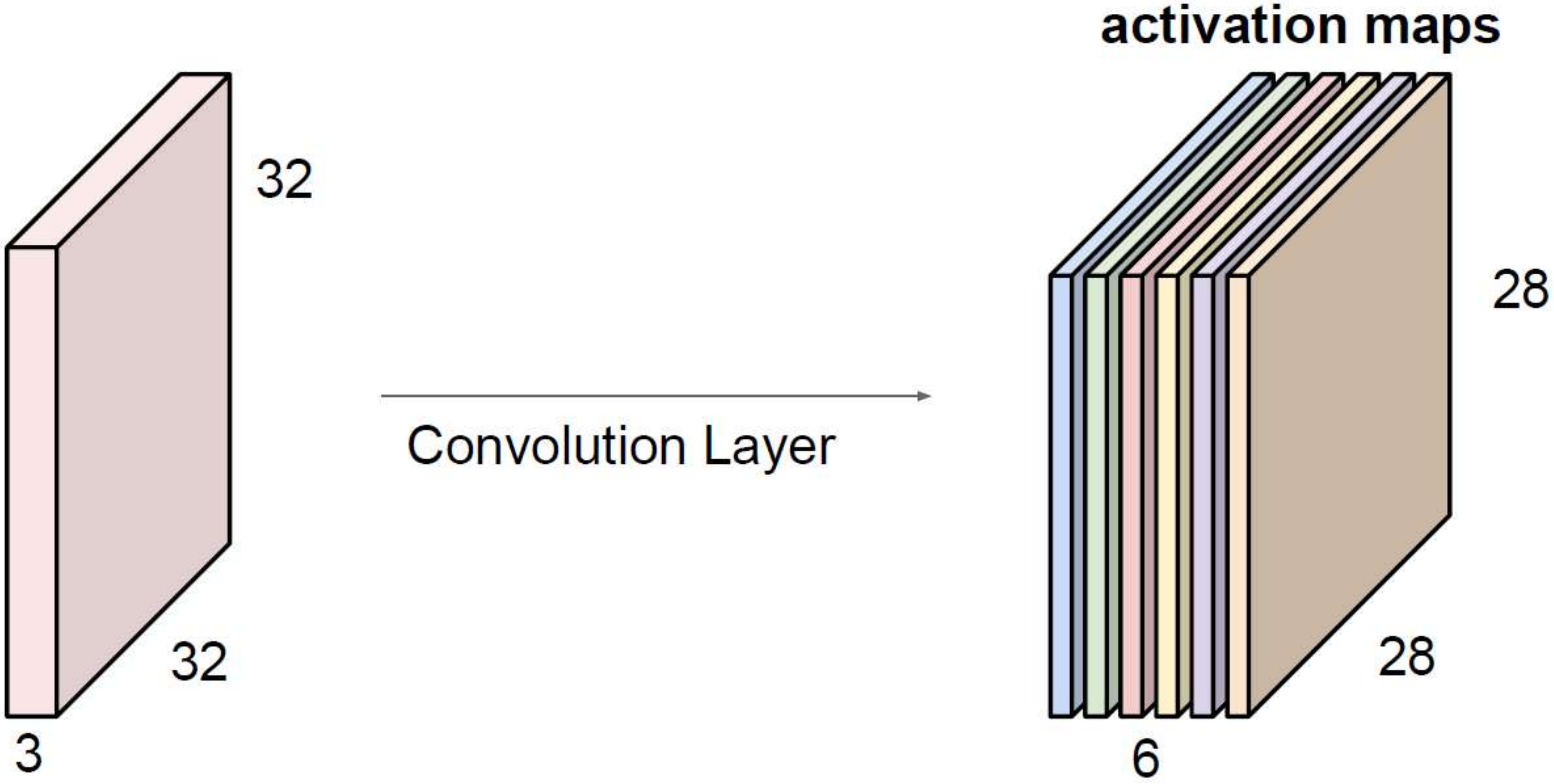
$$w^T x + b$$

2D Convolution Operation on Images



2D Convolution Operation on Images

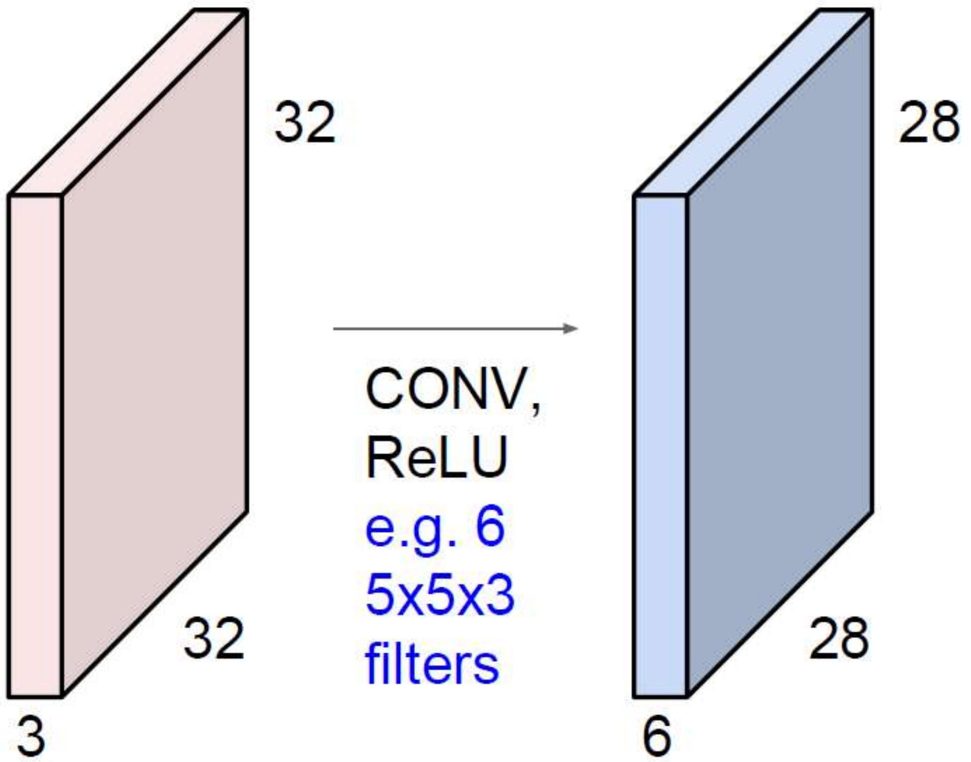
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

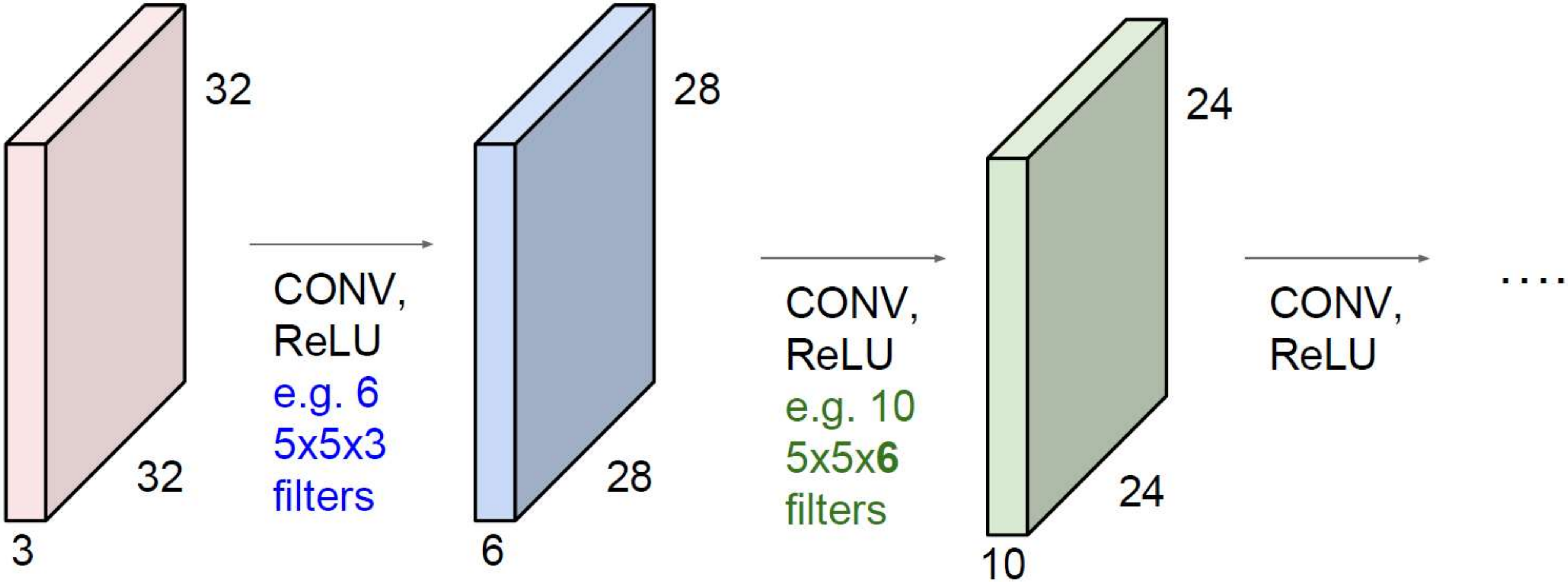
Convolutional Neural Network

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



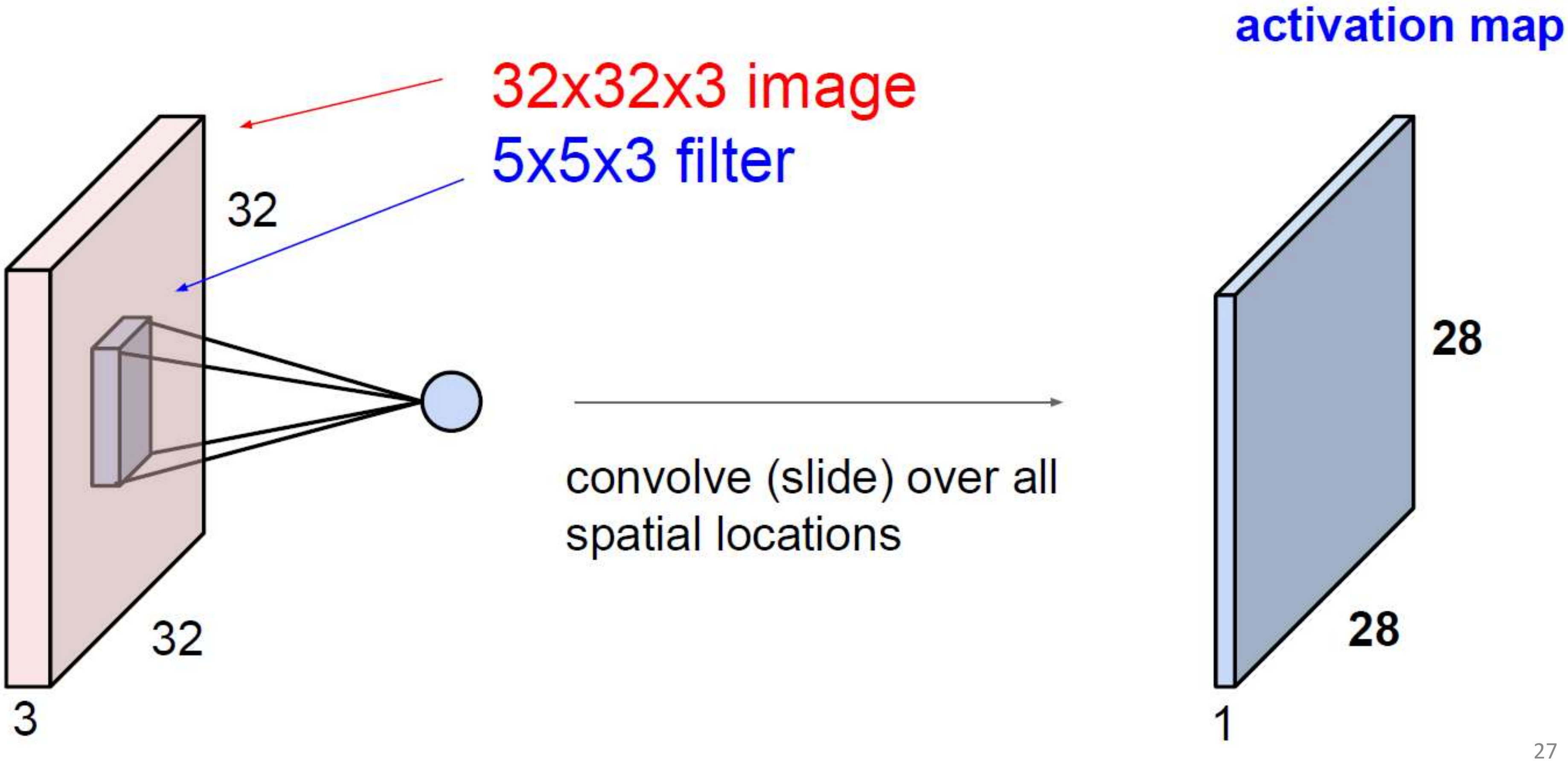
Convolutional Neural Network

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



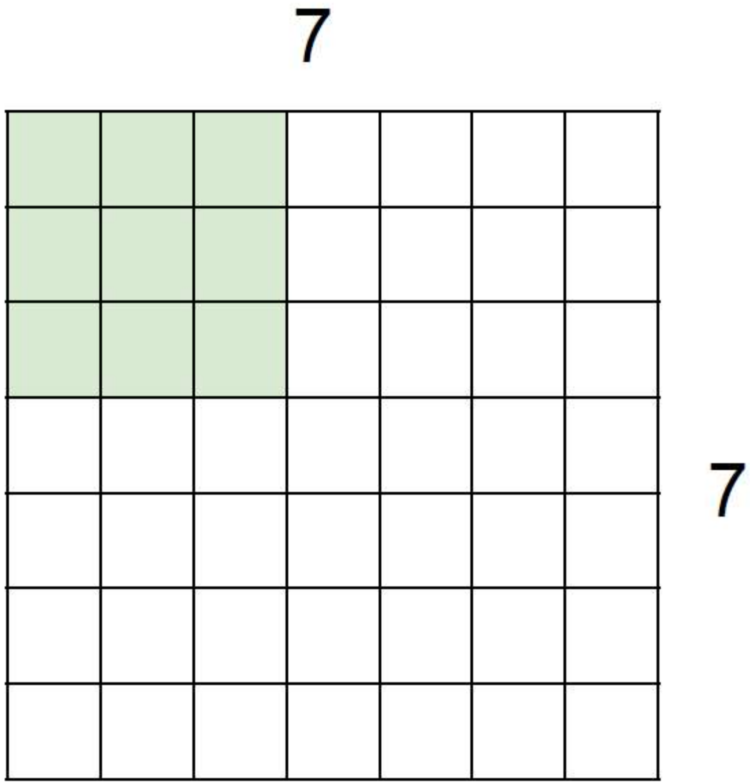
Convolution Operation

A closer look at spatial dimensions:



2D Convolution Operation – Spatial Dimensions

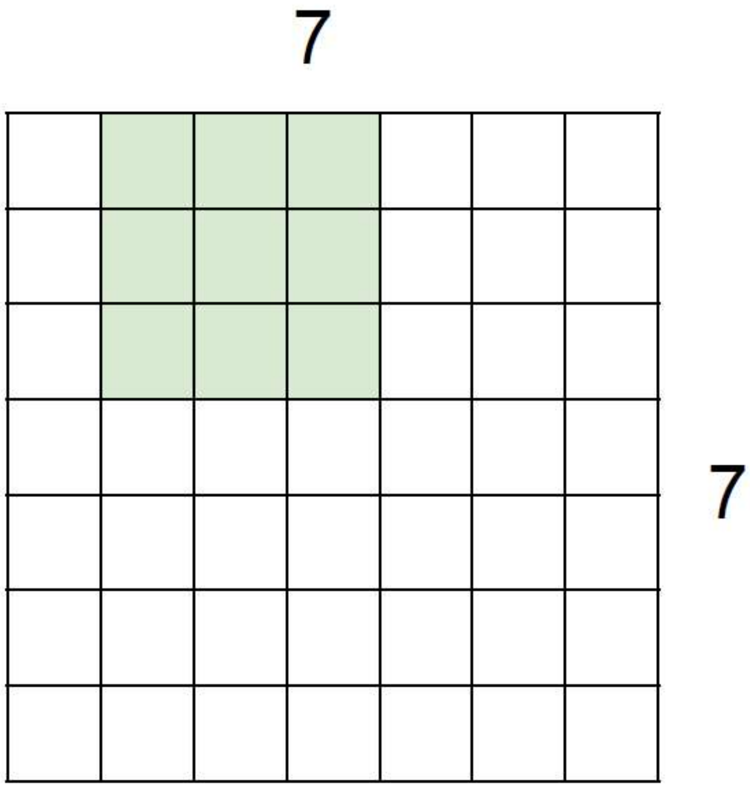
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

2D Convolution Operation – Spatial Dimensions

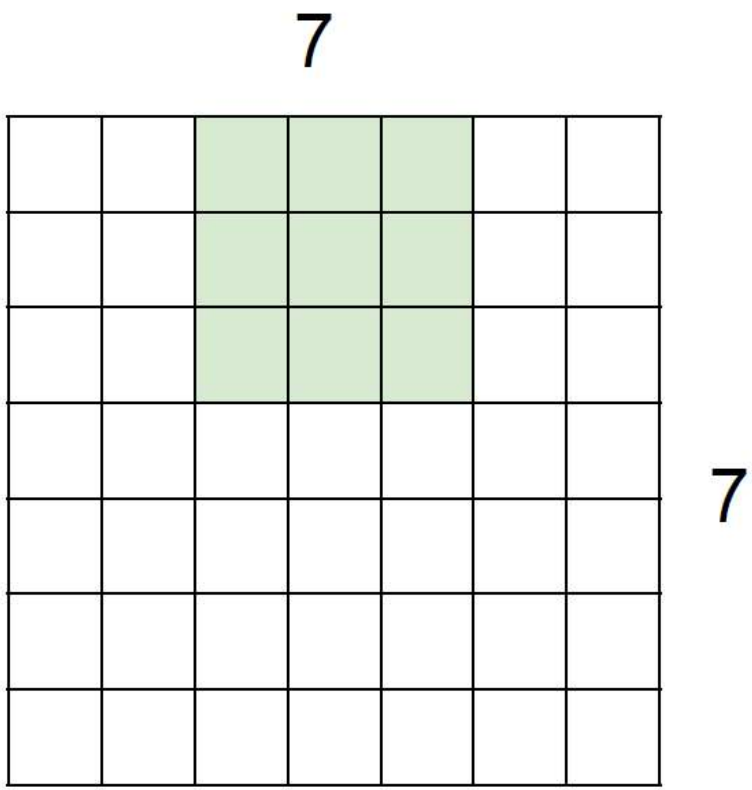
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

2D Convolution Operation – Spatial Dimensions

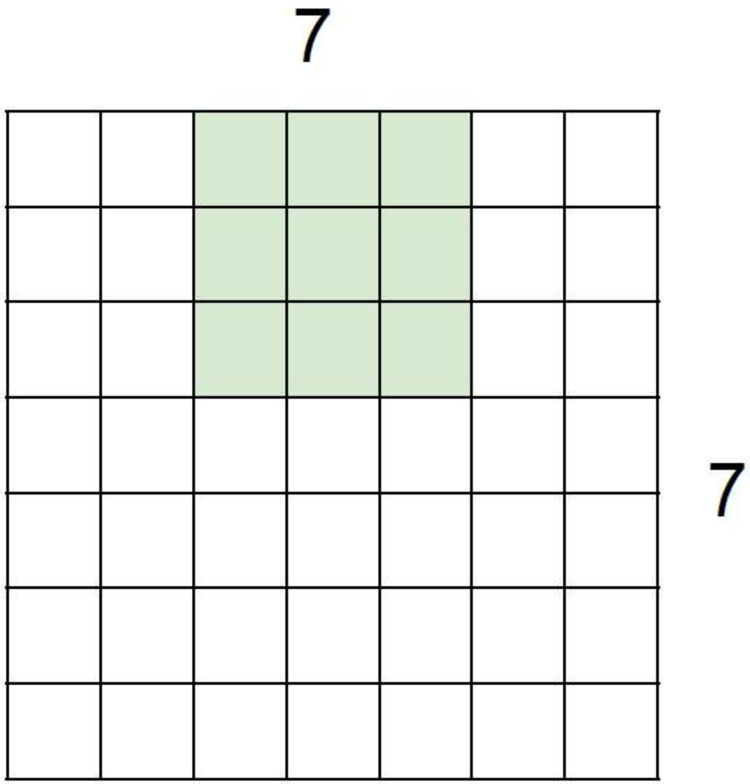
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

2D Convolution Operation – Spatial Dimensions

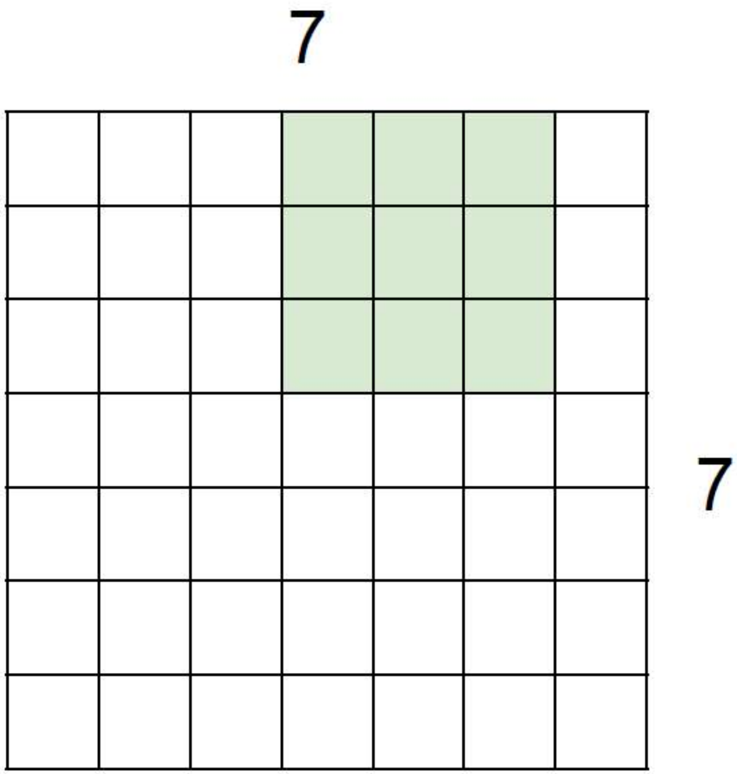
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

2D Convolution Operation – Spatial Dimensions

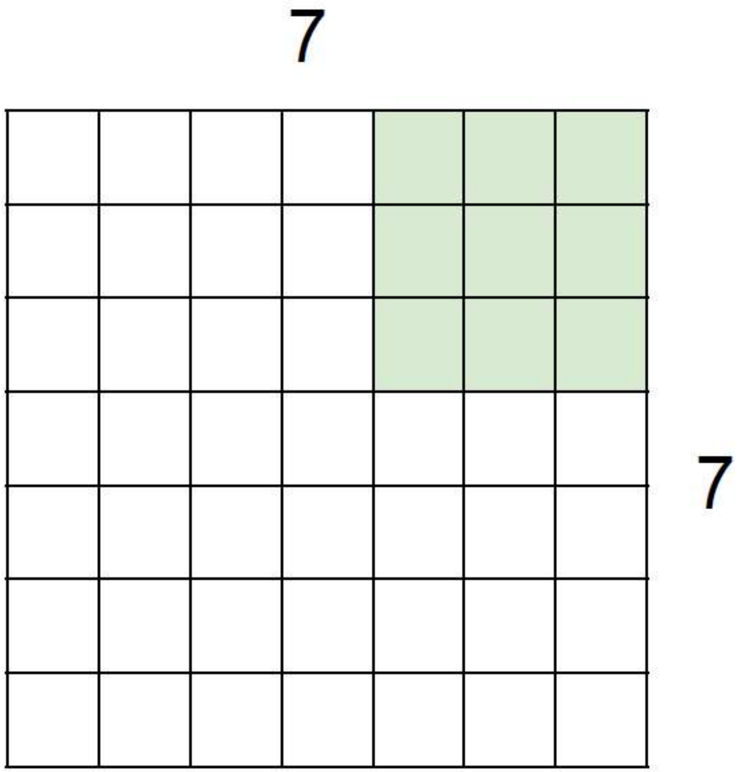
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

2D Convolution Operation – Spatial Dimensions

A closer look at spatial dimensions:

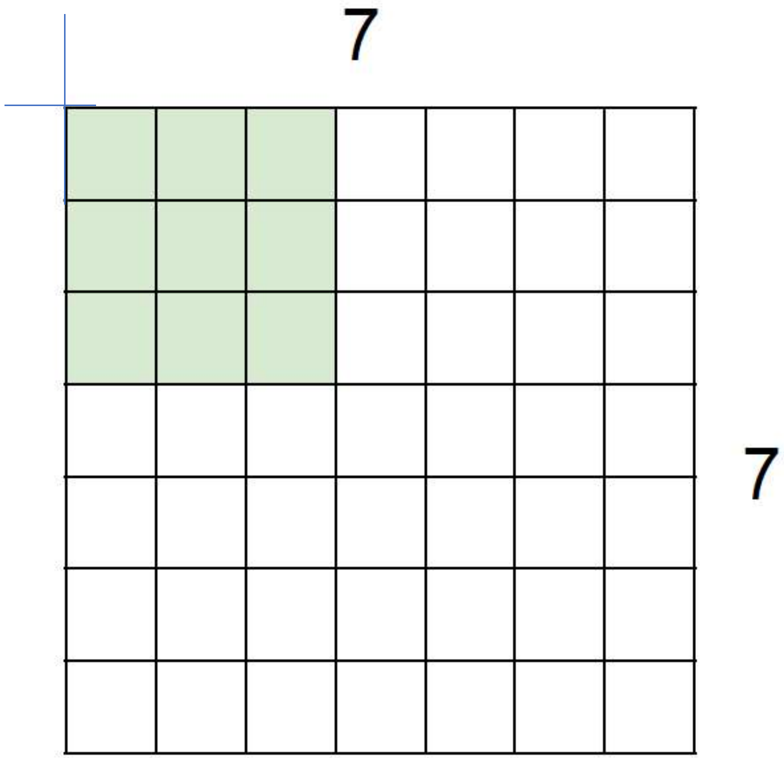


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

2D Convolution Operation – Spatial Dimensions

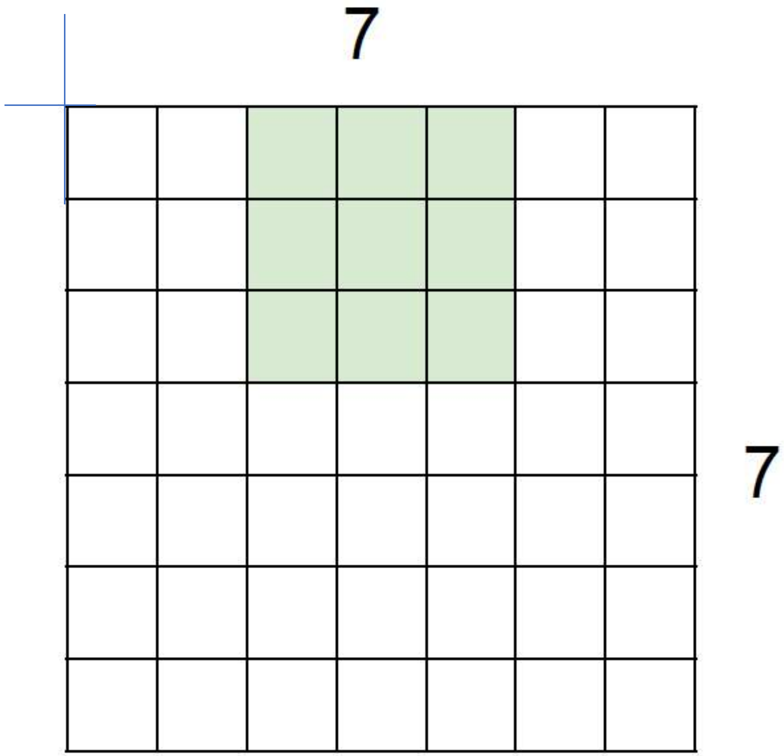
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

2D Convolution Operation – Spatial Dimensions

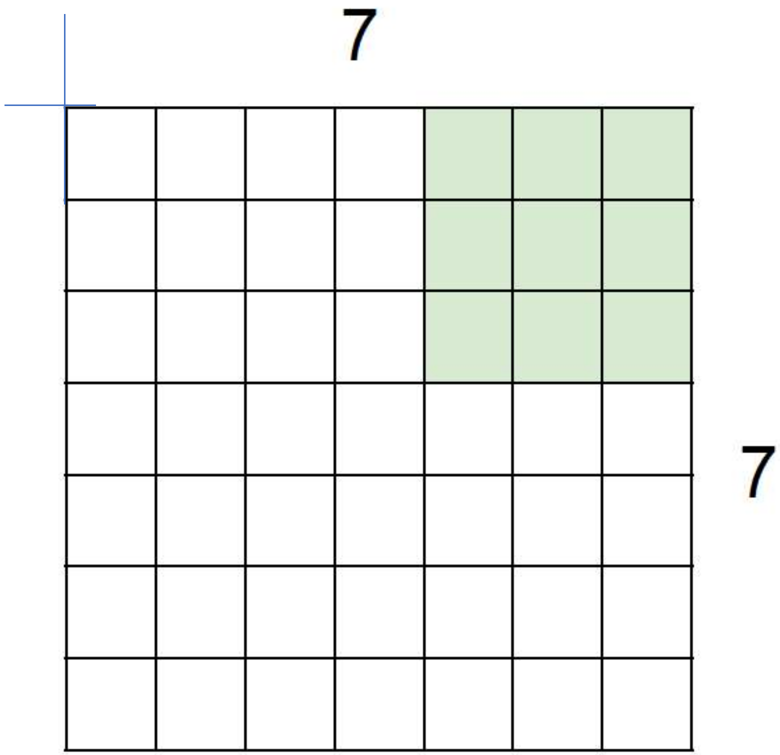
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

2D Convolution Operation – Spatial Dimensions

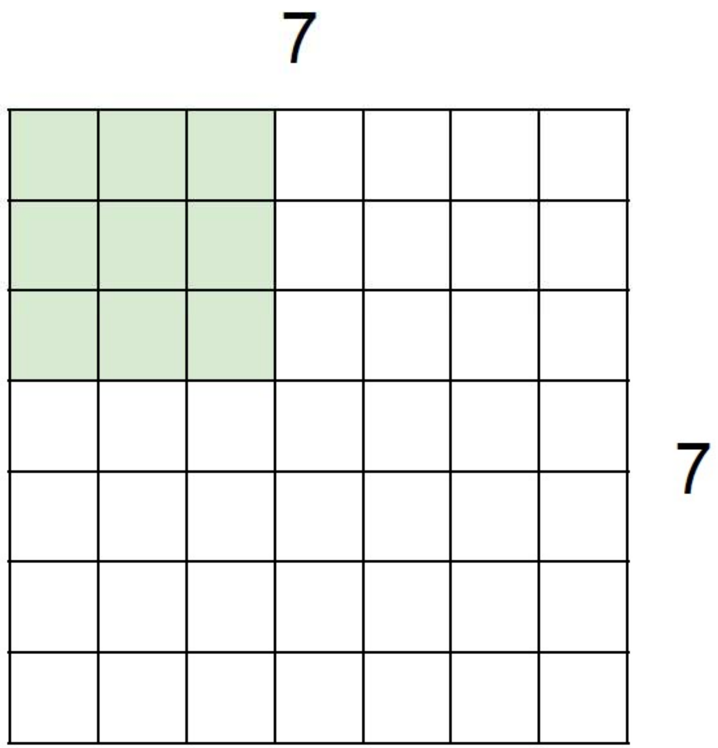
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

2D Convolution Operation – Spatial Dimensions

A closer look at spatial dimensions:

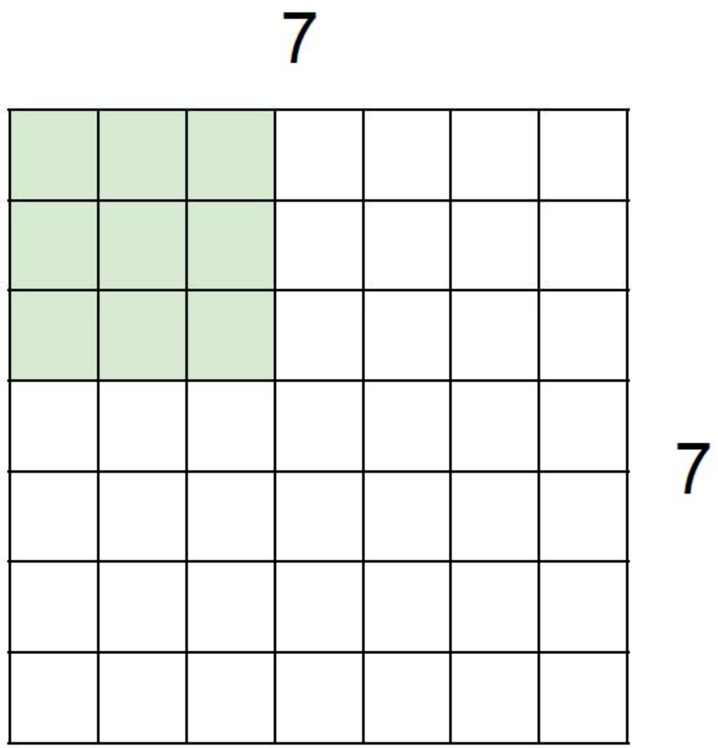


7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

2D Convolution Operation – Spatial Dimensions

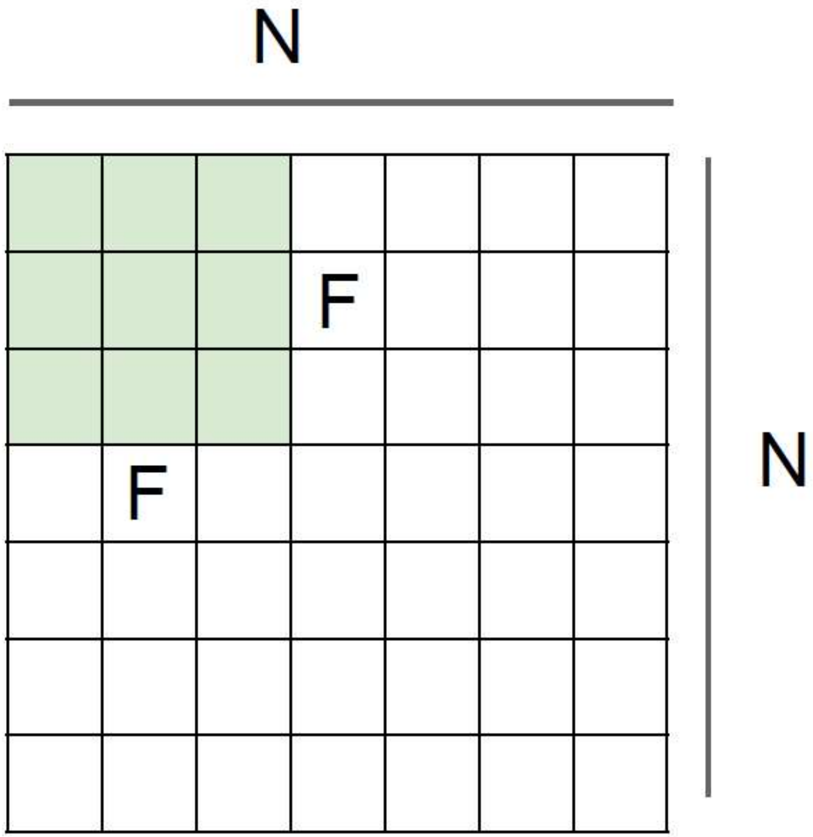
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

2D Convolution Operation – Spatial Dimensions



Output size:

(N - F) / stride + 1

e.g. N = 7, F = 3:

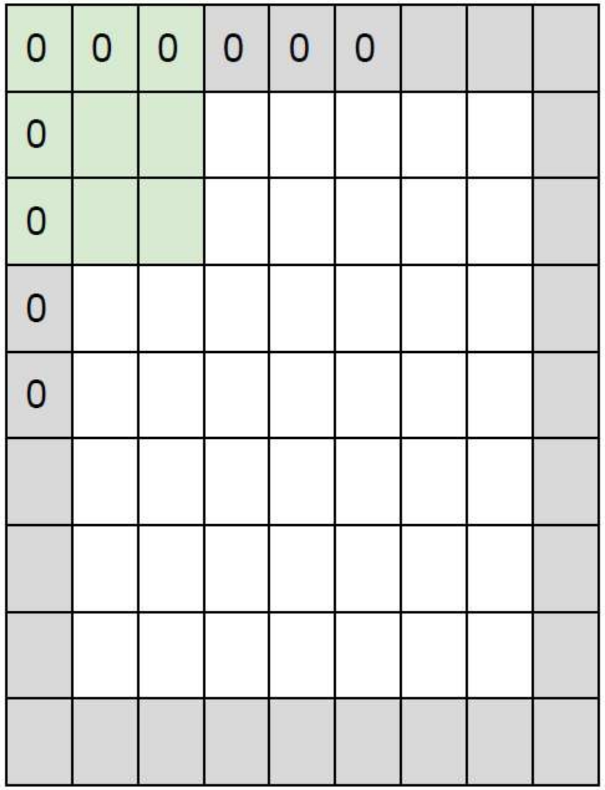
stride 1 => (7 - 3)/1 + 1 = 5

stride 2 => (7 - 3)/2 + 1 = 3

stride 3 => (7 - 3)/3 + 1 = 2.33

2D Convolution Operation – Spatial Dimensions

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

2D Convolution Operation – Spatial Dimensions

Summary

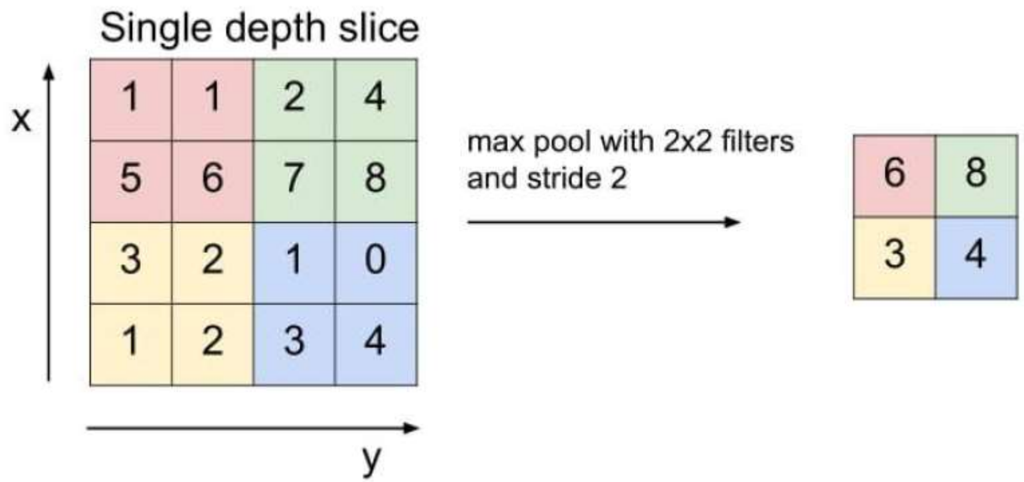
- Accepts a volume of size $\mathbf{W_1} \times \mathbf{H_1} \times \mathbf{D_1}$.
- Requires four hyperparameters:
  - number of filters $\mathbf{K}$.
  - their spatial extent $\mathbf{F}$.
  - the stride $\mathbf{S}$.
  - the amount of zero padding $\mathbf{P}$.
- Produces a volume of size $\mathbf{W2} \times \mathbf{H2} \times \mathbf{D2}$ where:
  - $\mathbf{W_2} = (\mathbf{W_1} - \mathbf{F} + \mathbf{2P})/\mathbf{S} + \mathbf{1}$.
  - $\mathbf{H2} = (\mathbf{H1} - \mathbf{F} + \mathbf{2P})/\mathbf{S} + \mathbf{1}$ (*i.e.* width and height are computed equally by symmetry).
  - $\mathbf{D2} = \mathbf{K}$.
- With parameter sharing, it introduces $\mathbf{F} \cdot \mathbf{F} \cdot \mathbf{D1}$ weights per filter, for a total of $(\mathbf{F} \cdot \mathbf{F} \cdot \mathbf{D1}) \cdot \mathbf{K}$ weights and $\mathbf{K}$ biases.

Pooling Operation

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.

Max Pooling

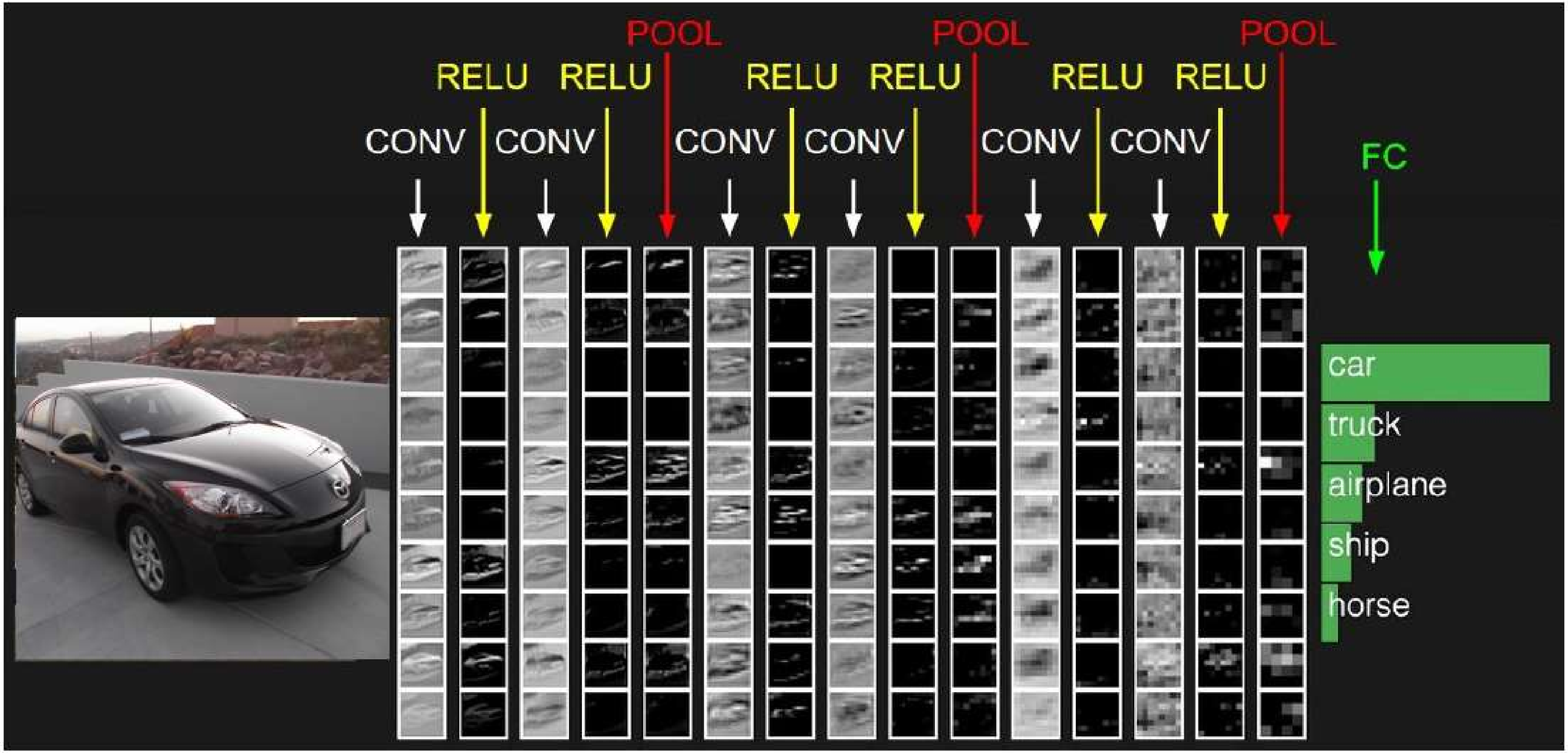


There are other kinds of pooling for example:

- Average Pooling
- Median Pooling
- Min Pooling etc.

But Maxpooling is more commonly used.

Convolutional Neural Network – Overall Architecture



Question and Answer