Convolutional Neural Network (CNN)

Computer Vision Group, IIT Patna

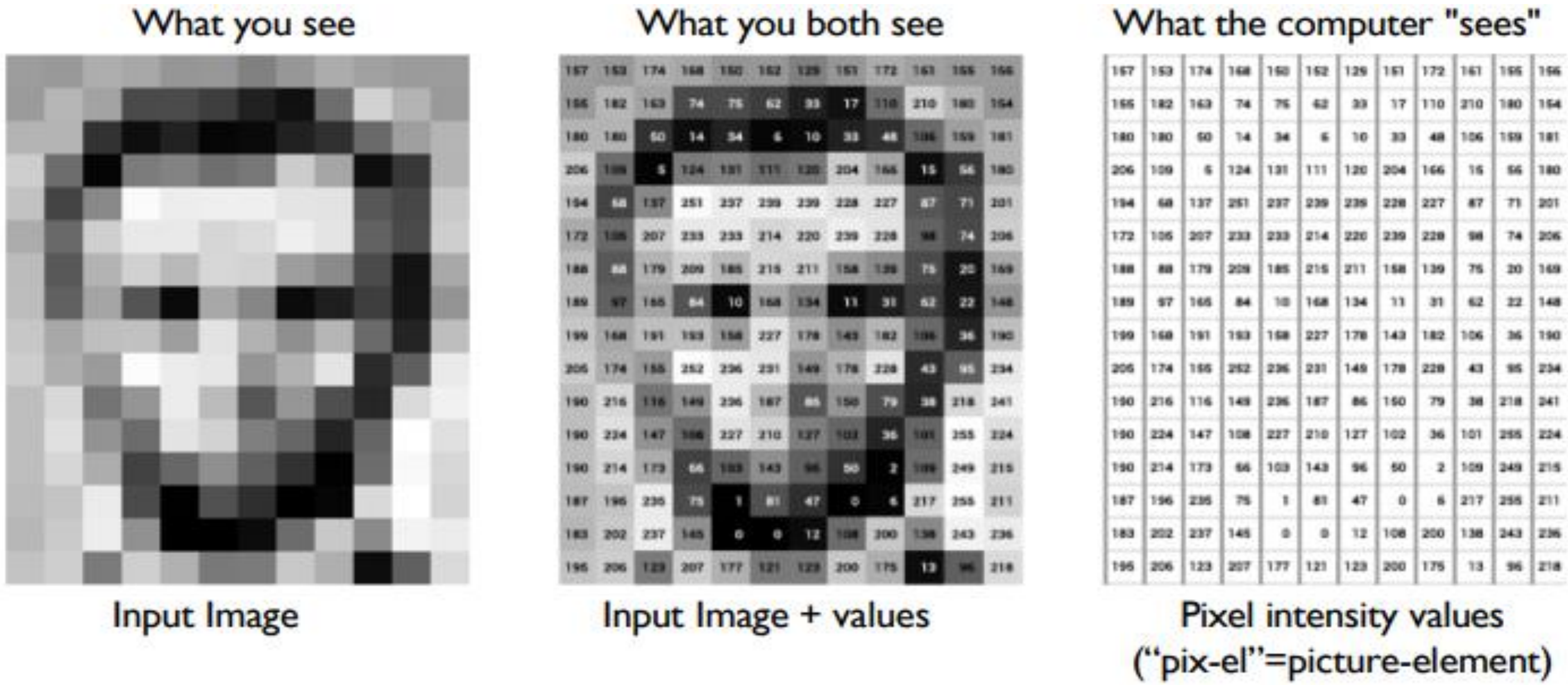
Slides prepared by: Ayindrila Dutta

Presented by: Dr. Chandranath Adak

Introduction

- Specialized neural network for processing data that has grid like topology
- Time series data (one dimensional)
- Image (two dimensional)
- Found to be reasonably suitable for certain class of problems eg. computer vision
- Instead of matrix multiplication, it uses **convolution** in at least one of the layers

What computers ‘see’: Images as Numbers

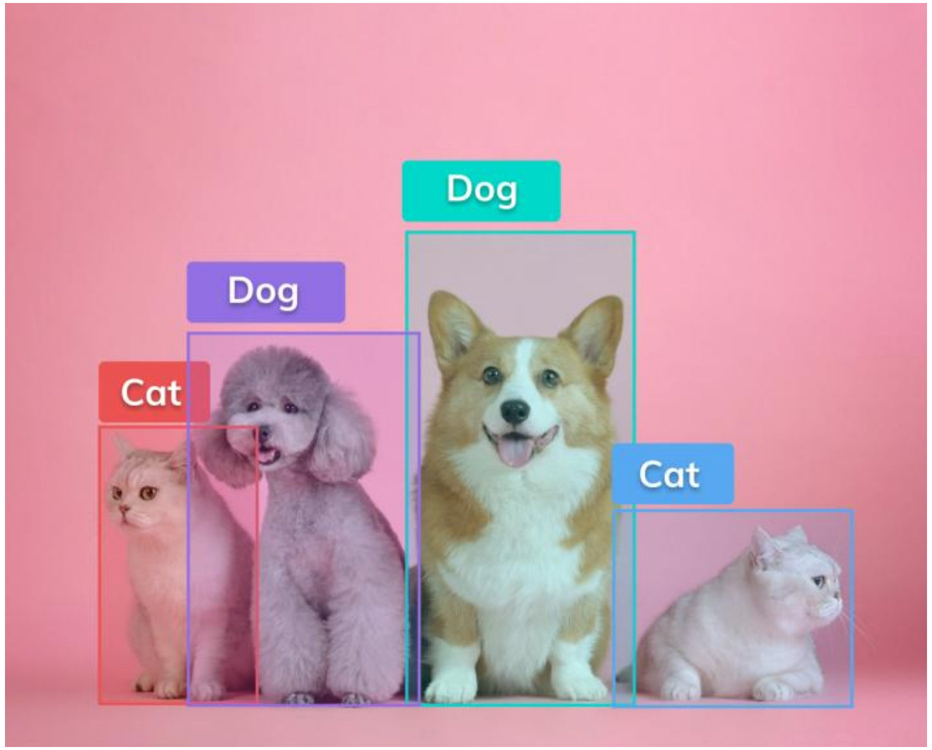


An image is just a matrix of numbers [0,255]. i.e., 1080x1080x3 for an RGB image.
Question: is this Lincoln? Washington? Jefferson? Obama?
How can the computer answer this question?

https://mit6874.github.io/assets/sp2020/slides/L03_CNNs_MK2.pdf

Computer Vision Problems

Classification
Cat? (0/1)



Object Detection

https://www.v7labs.com/blog/object-detection-guide

Computer Vision Problems

Neural Style Transfer

https://www.deeplearning.ai/

Deep Learning on large images

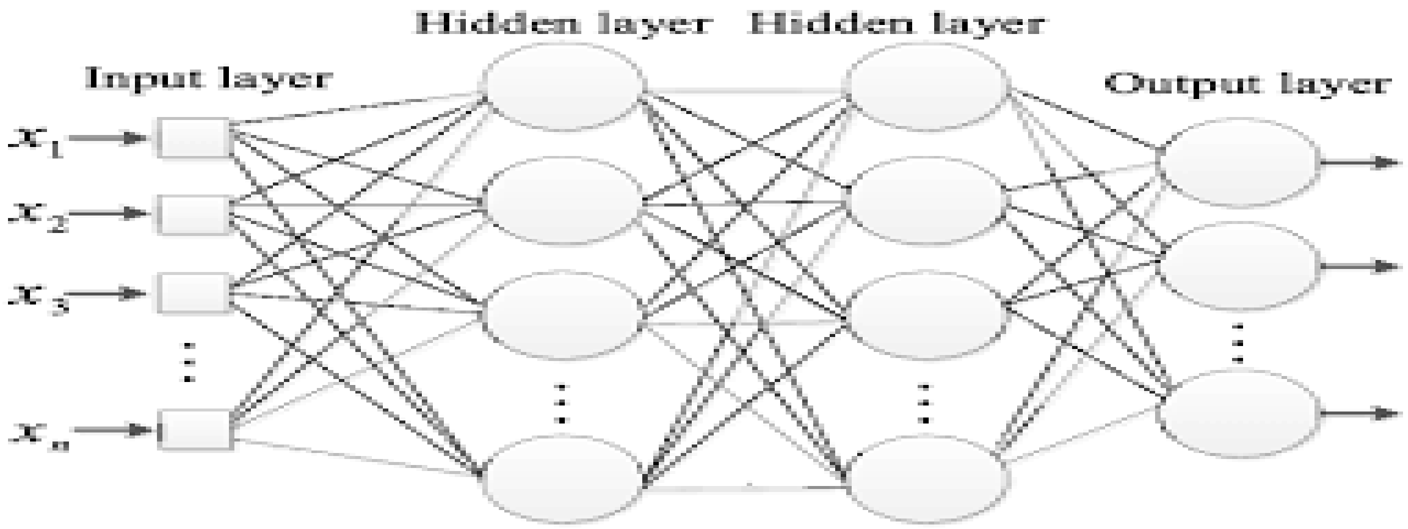
One of the challenges of computer vision is we need a quick and precise algorithm to handle them.

For example:

64x64x3
=12288

1000x1000x3
= 3 million

Deep Learning on large images (cont.)

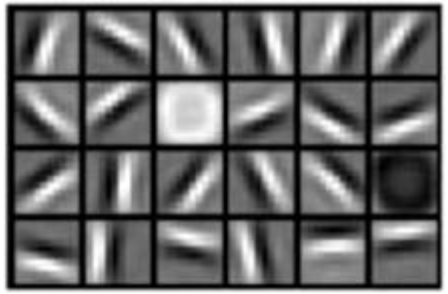


A 1000x1000x3 image will represent 3 million feature/input to the full connected neural network. If the following hidden layer contains 1000, then we will want to learn weights of the shape [1000, 3 million] which is 3 billion parameter only in the first layer and that so computationally expensive.

Solutions is to build this using **convolution layers** instead of the **fully connected layers**

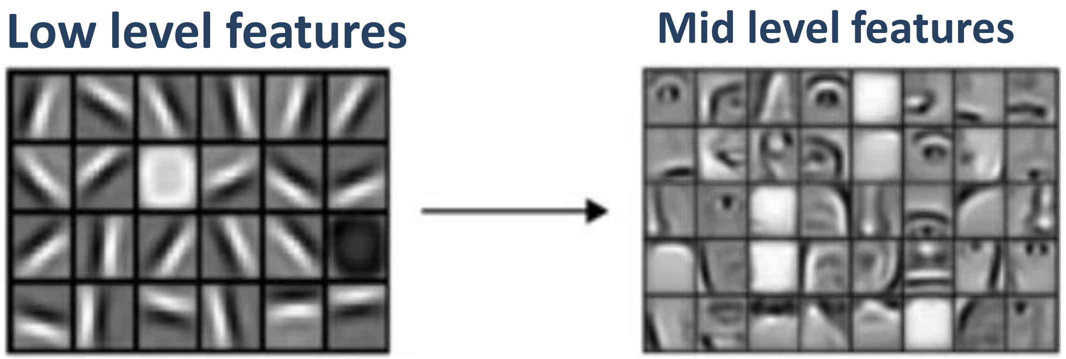
Edge detection example

Low level features



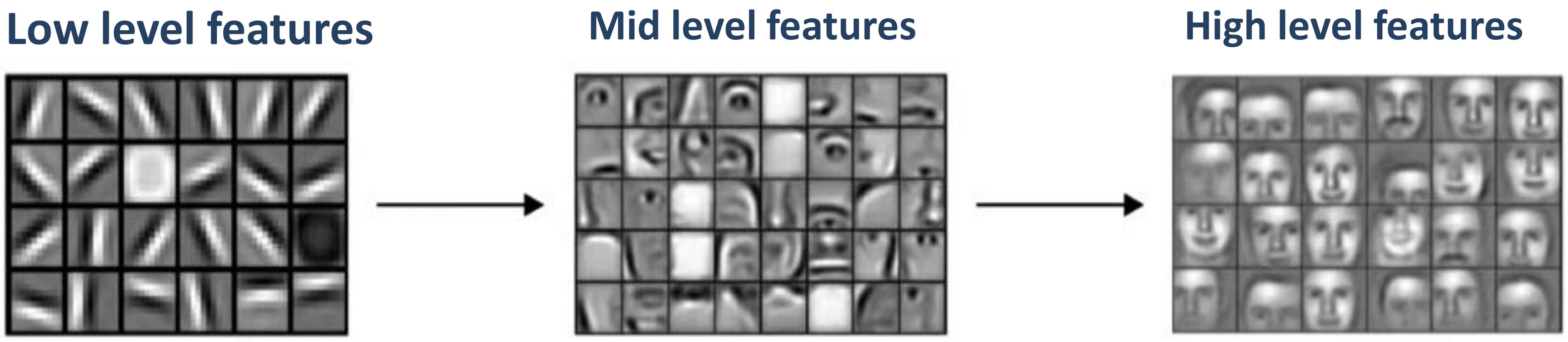
- Early layers of CNN might detect edges

Edge detection example



- Early layers of CNN might detect edges
- The middle layers will detect parts of objects

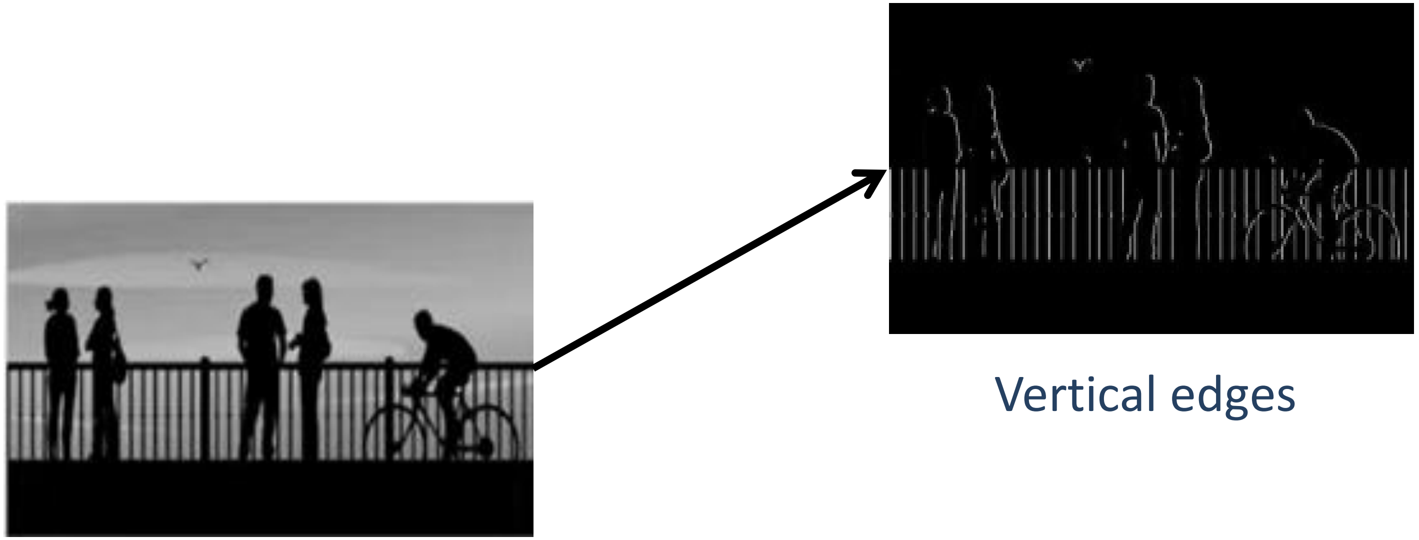
Edge detection example



- Early layers of CNN might detect edges
- The middle layers will detect parts of objects
- The later layers will put the these parts together to produce an output.

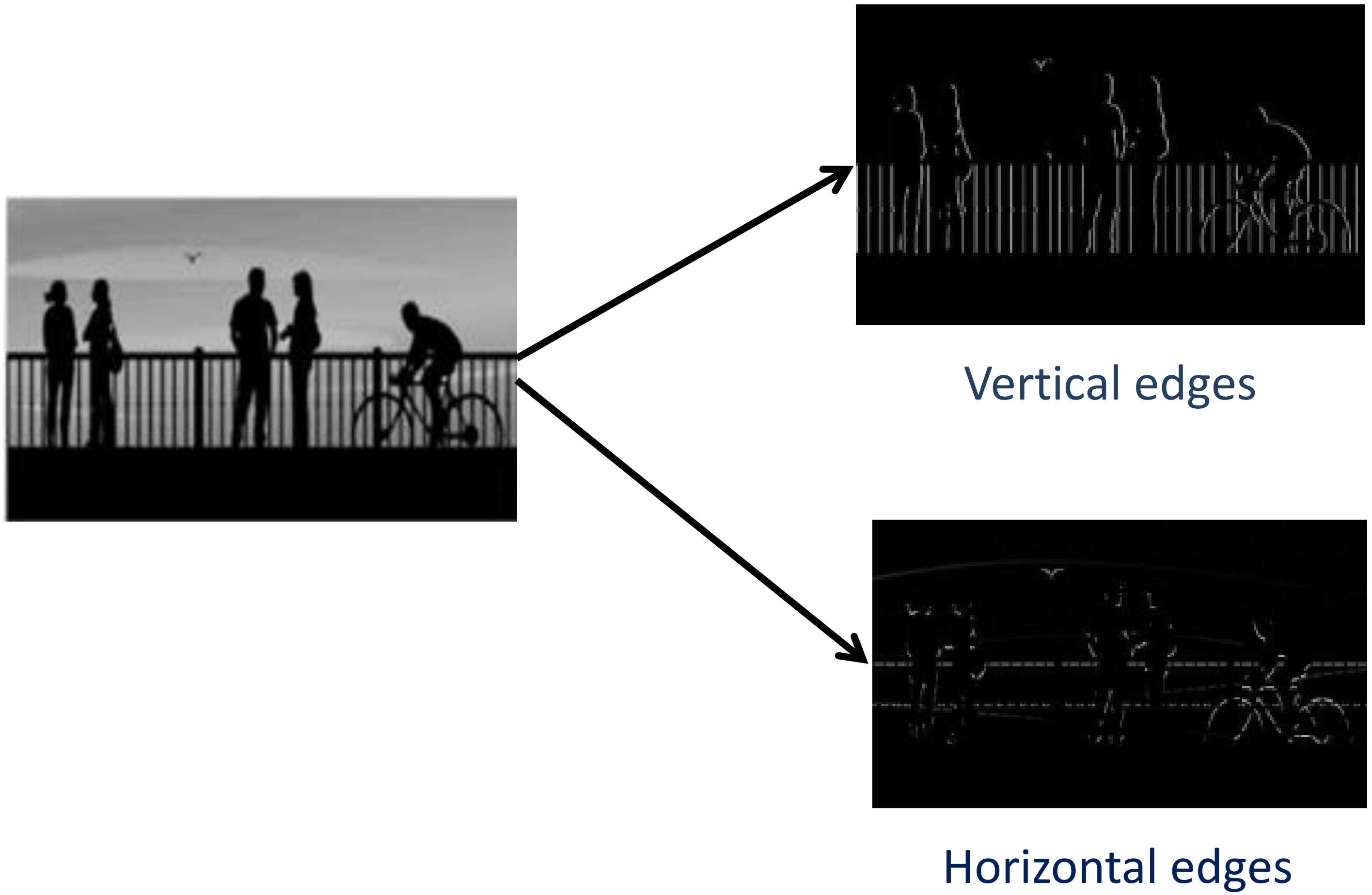
- In an image we can detect vertical edges, horizontal edges, or full edge detector.

Vertical edges, horizontal edges

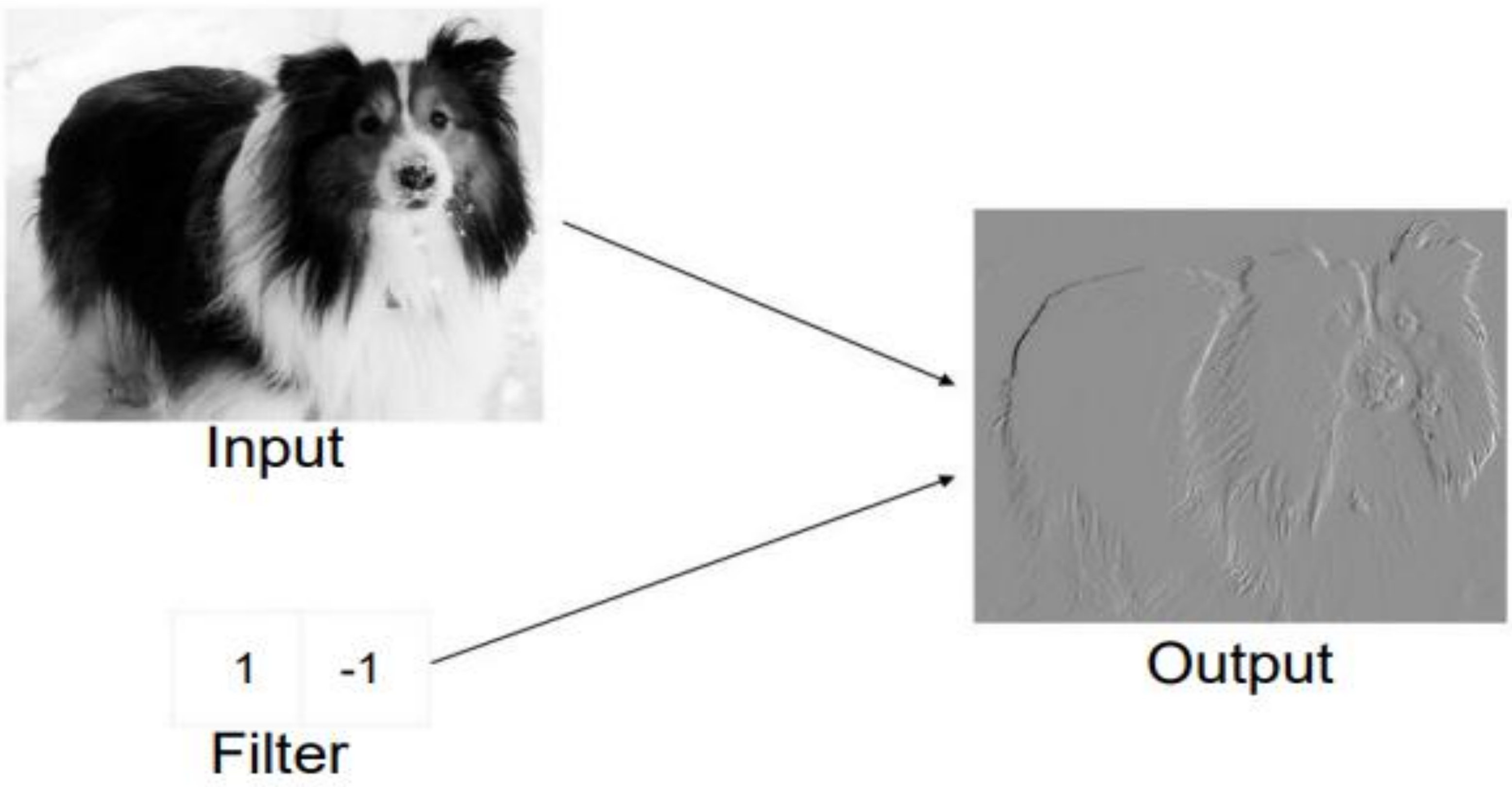


Vertical edges

Vertical edges, horizontal edges



How can we detect edges with a kernel?



Goodfellow 2016

Convolution operation

- Consider the scenario of locating a spaceship with a laser sensor
- Suppose, the sensor is noisy
  - Accurate estimation is not possible
- Weighted average of location can provide a good estimate $s(t) = \int x(a)w(t - a)da$
  - $x(a)$: Location at age a by the sensor, t : current time, w : weight
  - This is known as convolution
  - Usually denoted as $s(t) = (x * w)(t)$
- In neural network terminology x is input, w is kernel and output is referred as **feature map**

Convolution operation (contd)

- Discrete convolution can be represented as

$$s(t) = (x * w)(t) = \sum_{a=\infty}^{\infty} x(a)w(t - a)$$

- In neural network input is multidimensional and so is kernel
  - These will be referred as tensor
- Two-dimensional convolution can be defined as

$$s(i, j) = (I * \mathcal{K})(i, j) = \sum_{m,n} I(m, n)k(i - m, j - n) = \sum_{m,n} I(i - m, j - n)k(m, n)$$

  - Commutative
- In many neural network, it implements as cross-correlation

$$s(i, j) = (I * \mathcal{K})(i, j) = \sum_{m} \sum_{n} I(i + m, j + n)k(m, n)$$

  - No kernel flip is possible

Vertical edge detection

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

“convolution”

*

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

filter/kernel

=

-2			

3x1 + 1x1 + 3x1 + 0x0 + 4x0 + 6x0 + 1x-1 + 7x-1 + 1x-1 = -2

Vertical edge detection

"convolution"

$$\begin{bmatrix}3&0&1&1&7&4\\1&4&7&8&3&3\\3&6&1&5&2&3\\0&1&4&1&7&8\\5&2&2&5&4&8\\3&4&5&2&3&7\end{bmatrix} * \begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix} = \begin{bmatrix}-5&-4&&\\ & & & \\ & & & \\ & & & \end{bmatrix}$$

3x1 + 1x1 + 2x1 + 0x0 + 5x0 + 7x0 + 1x-1 + 8x-1 + 2x-1 = -5

0x1 + 4x1 + 6x1 + 1x0 + 7x0 + 1x0 + 1x-1 + 8x-1 + 5x-1 = -4

Vertical edge detection

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

"convolution"

$*$

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

$=$

-5	-4	-3	4
-9	-3	0	0
1	-2	-5	-8
-3	-1	-3	-15

$3 \times 1 + 1 \times 1 + 2 \times 1 + 0 \times 0 + 5 \times 0 + 7 \times 0 + 1 \times -1 + 8 \times -1 + 2 \times -1 = -5$

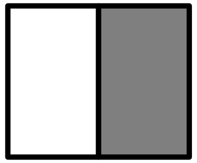
$0 \times 1 + 4 \times 1 + 6 \times 1 + 1 \times 0 + 7 \times 0 + 1 \times 0 + 1 \times -1 + 8 \times -1 + 5 \times -1 = -4$

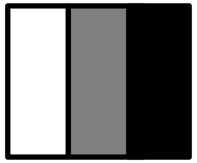
Vertical edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

$*$

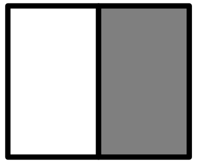
1	0	-1
1	0	-1
1	0	-1

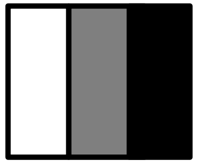




Vertical edge detection

$$
\begin{array}{|c|c|c|c|c|c|}
\hline
10 & 10 & 10 & 0 & 0 & 0 \\ \hline
10 & 10 & 10 & 0 & 0 & 0 \\ \hline
10 & 10 & 10 & 0 & 0 & 0 \\ \hline
10 & 10 & 10 & 0 & 0 & 0 \\ \hline
10 & 10 & 10 & 0 & 0 & 0 \\ \hline
10 & 10 & 10 & 0 & 0 & 0 \\ \hline
\end{array}
\;*\;
\begin{array}{|c|c|c|}
\hline
1 & 0 & -1 \\ \hline
1 & 0 & -1 \\ \hline
1 & 0 & -1 \\ \hline
\end{array}
\;=\;
\begin{array}{|c|c|c|c|}
\hline
0 & & & \\ \hline
 & & & \\ \hline
 & & & \\ \hline
 & & & \\ \hline
\end{array}
$$





Vertical edge detection

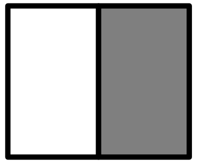
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

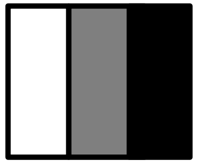
*

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

=

0	30	30	





Vertical edge detection

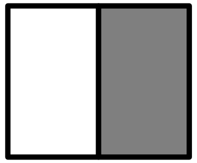
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

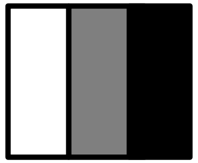
*

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

=

0	30	30	0
0	30	30	0
0	30	30	0
0			





Vertical edge detection

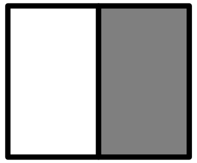
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

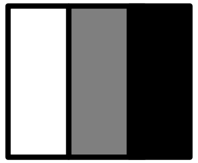
*

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

=

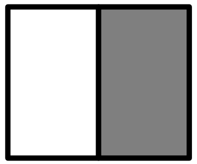
0	30	30	0
0	30	30	0
0	30	30	0
0	30		

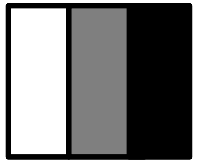


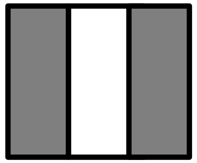


Vertical edge detection

$$
\begin{bmatrix}
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1
\end{bmatrix}
=
\begin{bmatrix}
0 & 30 & 30 & 0 \\
0 & 30 & 30 & 0 \\
0 & 30 & 30 & 0 \\
0 & 30 & 30 & 0
\end{bmatrix}
$$





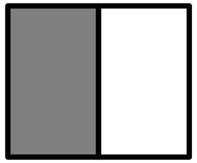


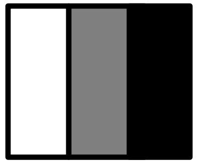
Vertical edge detection example

0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

$*$

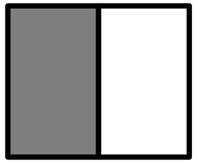
1	0	-1
1	0	-1
1	0	-1

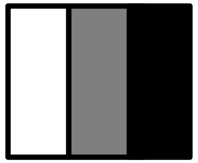




Vertical edge detection example

$$
\begin{bmatrix}
0 & 0 & 0 & 10 & 10 & 10 \\
0 & 0 & 0 & 10 & 10 & 10 \\
0 & 0 & 0 & 10 & 10 & 10 \\
0 & 0 & 0 & 10 & 10 & 10 \\
0 & 0 & 0 & 10 & 10 & 10 \\
0 & 0 & 0 & 10 & 10 & 10
\end{bmatrix}
*
\begin{bmatrix}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1
\end{bmatrix}
=
\begin{bmatrix}
0 & -30 & -30 & 0 \\
0 & -30 & -30 & 0 \\
0 & -30 & -30 & 0 \\
0 & -30 & -30 & 0
\end{bmatrix}
$$





Other Edge Filters

Horizontal edge detection Filter

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

Other Edge Filters

Sobel filter

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

Scharr filter

3	0	-3
10	0	-10
3	0	-3

Simple Kernels / Filters

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)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

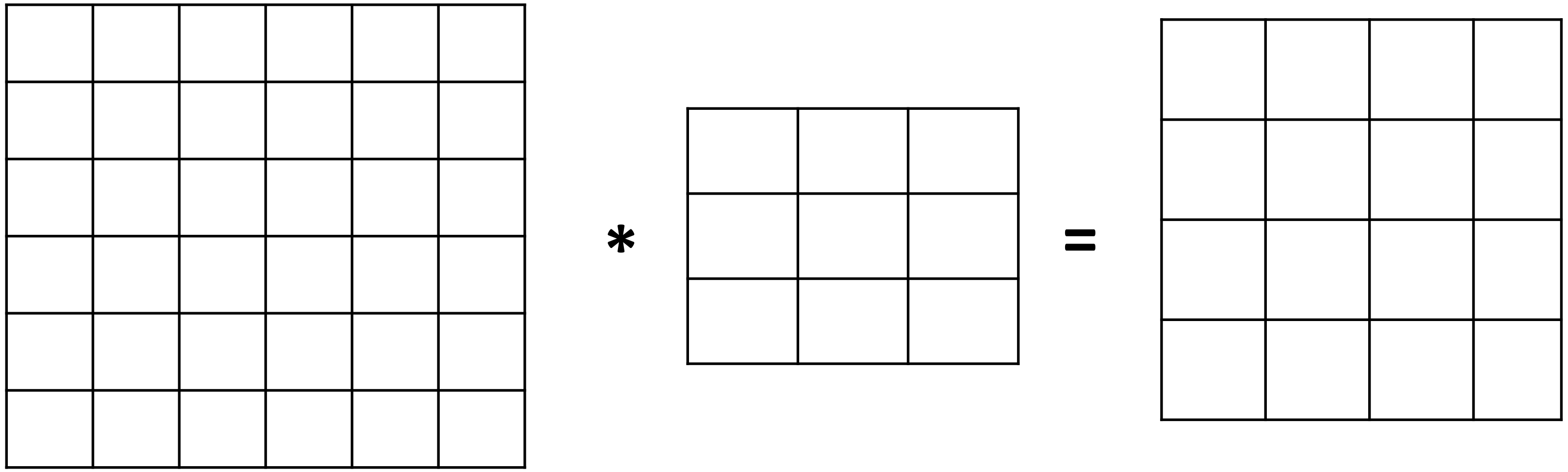
https://mit6874.github.io/assets/sp2020/slides/L03_CNNs_MK2.pdf

Learning to detect edges

$$
\begin{array}{|c|c|c|c|c|c|}
\hline
3 & 0 & 1 & 2 & 7 & 4 \\ \hline
1 & 5 & 8 & 9 & 3 & 1 \\ \hline
2 & 7 & 2 & 5 & 1 & 3 \\ \hline
0 & 1 & 3 & 1 & 7 & 8 \\ \hline
4 & 2 & 1 & 6 & 2 & 8 \\ \hline
2 & 4 & 5 & 2 & 3 & 9 \\ \hline
\end{array}
\;*\;
\begin{array}{|c|c|c|}
\hline
w1 & w2 & w3 \\ \hline
w4 & w5 & w6 \\ \hline
w7 & w8 & w9 \\ \hline
\end{array}
\;=\;
\begin{array}{|c|c|c|c|}
\hline
 & & & \\ \hline
 & & & \\ \hline
 & & & \\ \hline
 & & & \\ \hline
\end{array}
$$

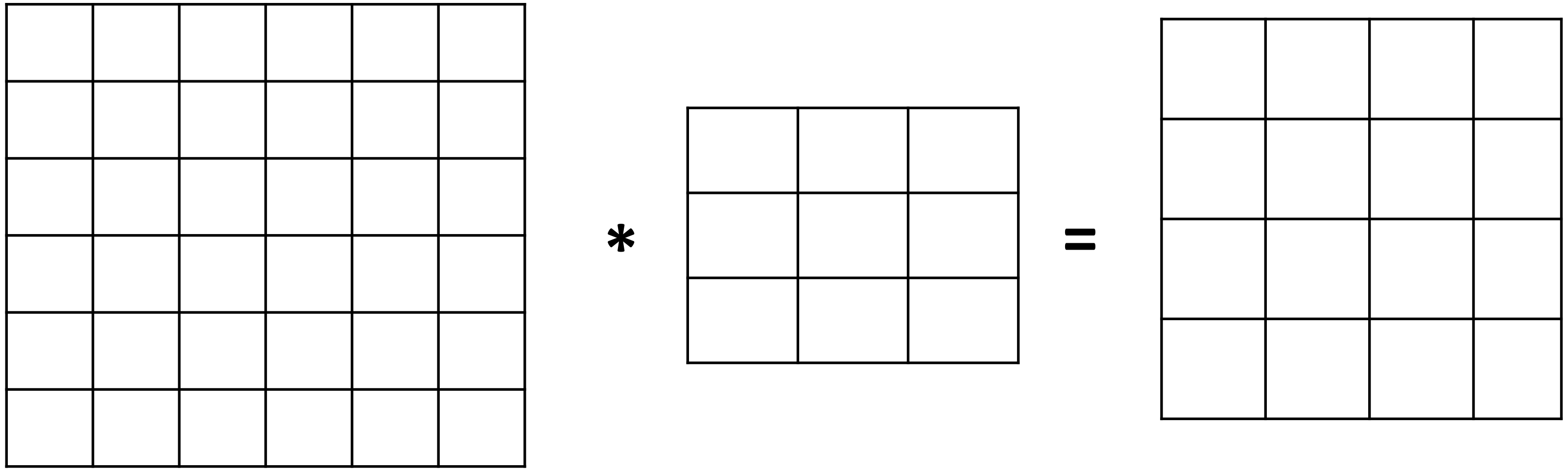
It can learn horizontal, vertical, angled, or any edge type automatically

Drawback of convolution operation



6x6 **3x3** **4x4**

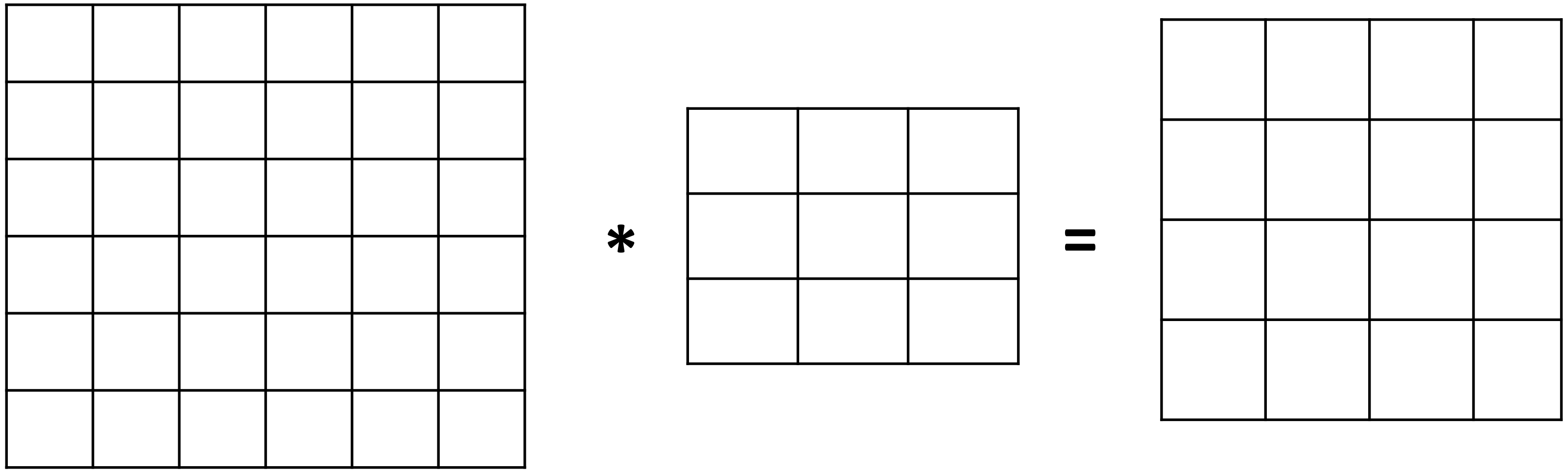
Drawback of convolution operation



6x6   3x3   4x4

$n \times n$ $f \times f$ $n-f+1 \times n-f+1$

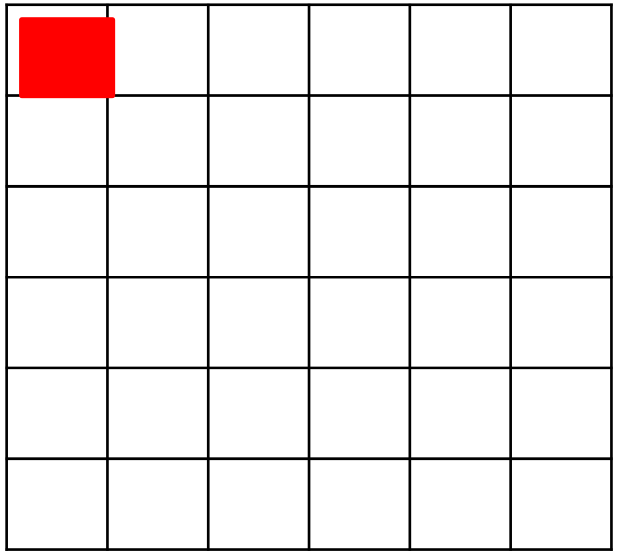
Drawback of convolution operation



6x6 $n \mathrm{x} n$ **3x3** $f \mathrm{x} f$ **4x4** $n\text{-}f{+}1 \ \mathrm{x}\ n\text{-}f{+}1$

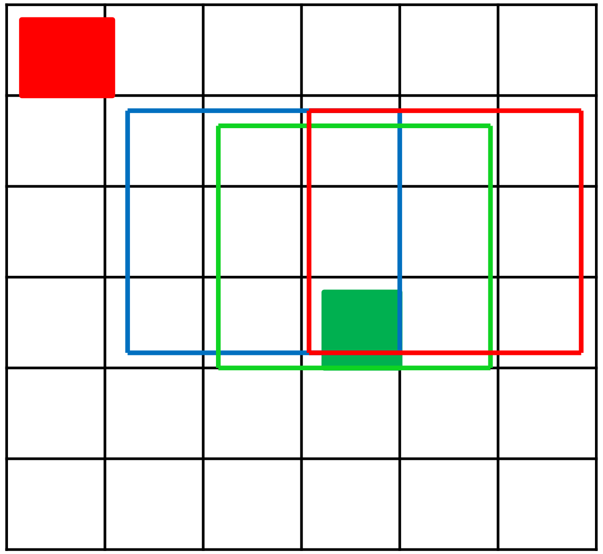
- The convolution operation shrinks the matrix if $f > 1$

Drawback of convolution operation



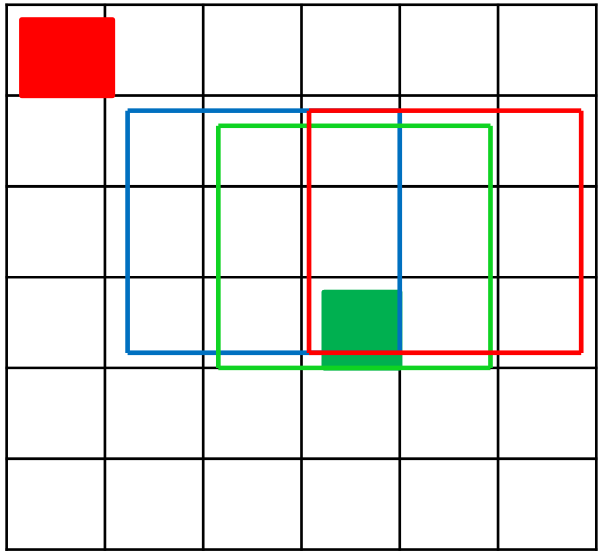
- 1x1 information is used only once

Drawback of convolution operation



- 1x1 information is used only once
- 4x4 information is used multiple times

Drawback of convolution operation



- 1x1 information is used only once
- 4x4 information is used multiple times

Problems with convolutions are:

- Shrinks output.
- throwing away a lot of information that are in the edges.

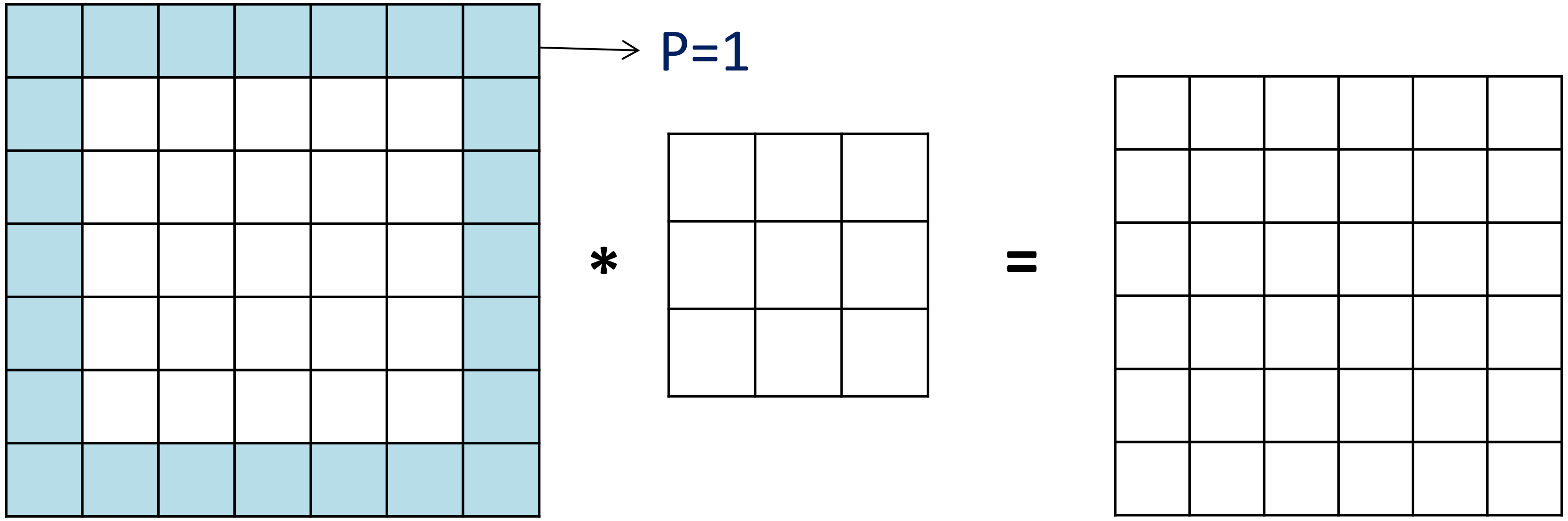
Padding

To solve these problems we pad the input image before convolution by adding some rows and columns to it, this is called padding amount $\mathcal{P}$.

The padding values are ***zeros***.

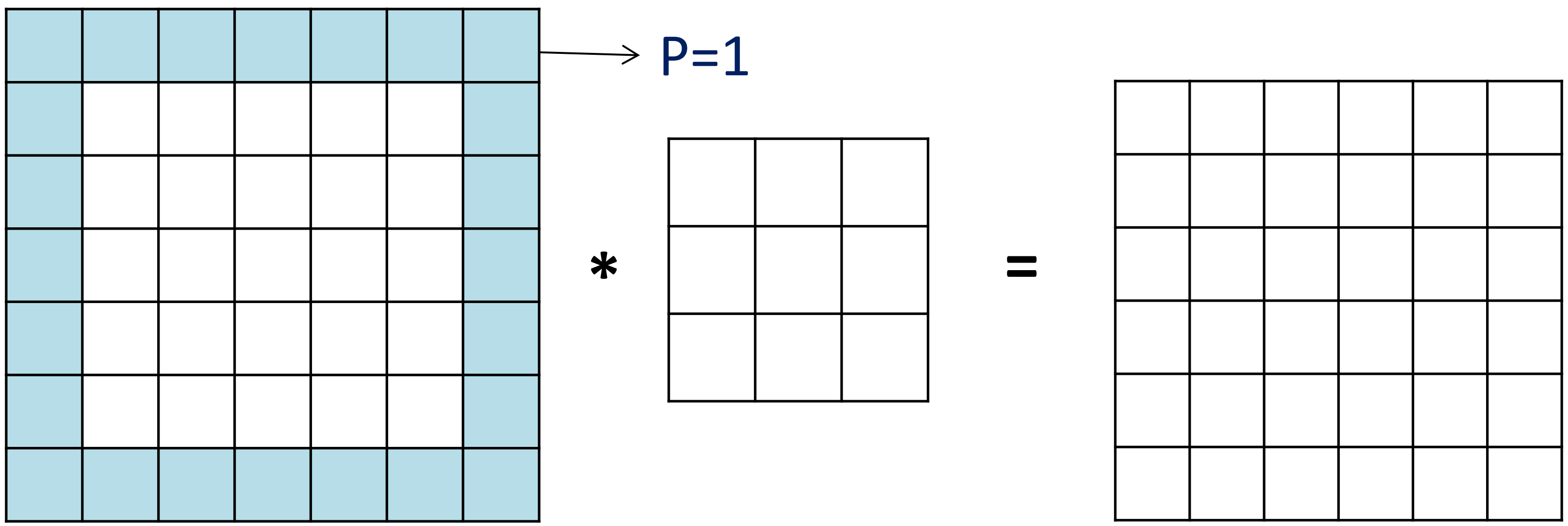
0	0	0	0	0	0	0
0						0
0						0
0						0
0						0
0						0
0	0	0	0	0	0	0

Padding



$n \times n$ $\quad * \quad$ $f \times f$ $\quad = \quad$ $n+2p-f+1 \times n+2p-f+1$

Padding



$n x n$	$f x f$	$n+2p-f+1$ x $n+2p-f+1$
6x6	**3x3**	**6+2*1-3+1 x 6+2*1-3+1**
		6 x 6

P = (f-1) / 2, where f is usually odd

Strided convolution

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

*

3	4	4
1	0	2
-1	0	3

=

85		

Stride = 2

Strided convolution

$$
\begin{bmatrix}
2 & 3 & 7 & 4 & 1 & 3 & 9 \\
5 & 7 & 8 & 5 & 7 & 4 & 2 \\
6 & 4 & 8 & 3 & 6 & 9 & 6 \\
7 & 8 & 9 & 6 & 5 & 3 & 3 \\
3 & 2 & 5 & 8 & 2 & 4 & 1 \\
3 & 2 & 6 & 1 & 7 & 8 & 3 \\
5 & 1 & 3 & 9 & 2 & 1 & 6
\end{bmatrix}
*
\begin{bmatrix}
3 & 4 & 4 \\
1 & 0 & 2 \\
-1 & 0 & 3
\end{bmatrix}
=
\begin{bmatrix}
91 & 73 & 74 \\
 & & \\
 & &
\end{bmatrix}
$$

Stride = 2

Strided convolution

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

*

3	4	4
1	0	2
-1	0	3

=

91	73	74
103		

Stride = 2

Strided convolution

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

*

3	4	4
1	0	2
-1	0	3

=

91	73	74
103	80	90
53	78	55

Stride = 2

Strided convolution

$n \times n$ image $\qquad f \times f$ filter

padding $p \qquad$ stride s

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

Strided convolution

Convolution with a padding so that output size is the same as the input size.

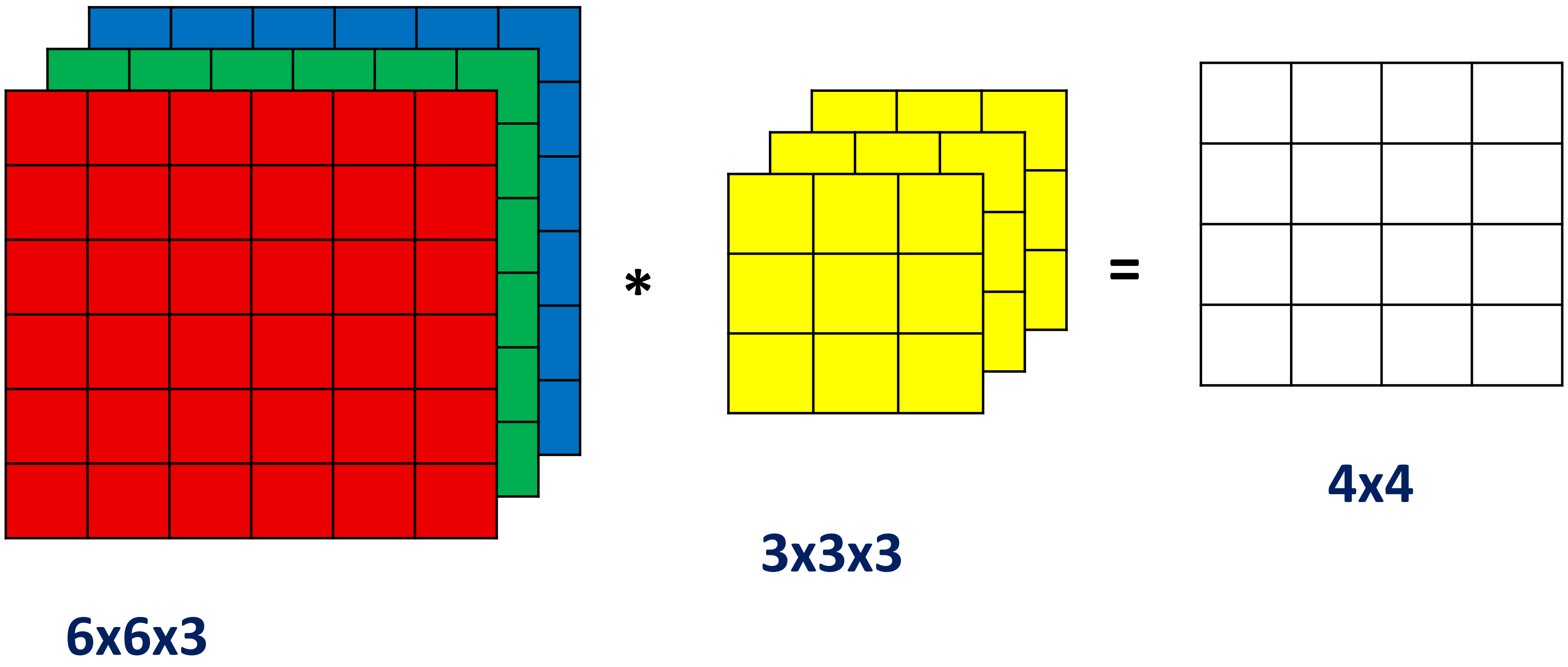
by the equation:

$p = (n*s - n + f - s) / 2$

When $s = 1$ ==> $P = (f-1) / 2$

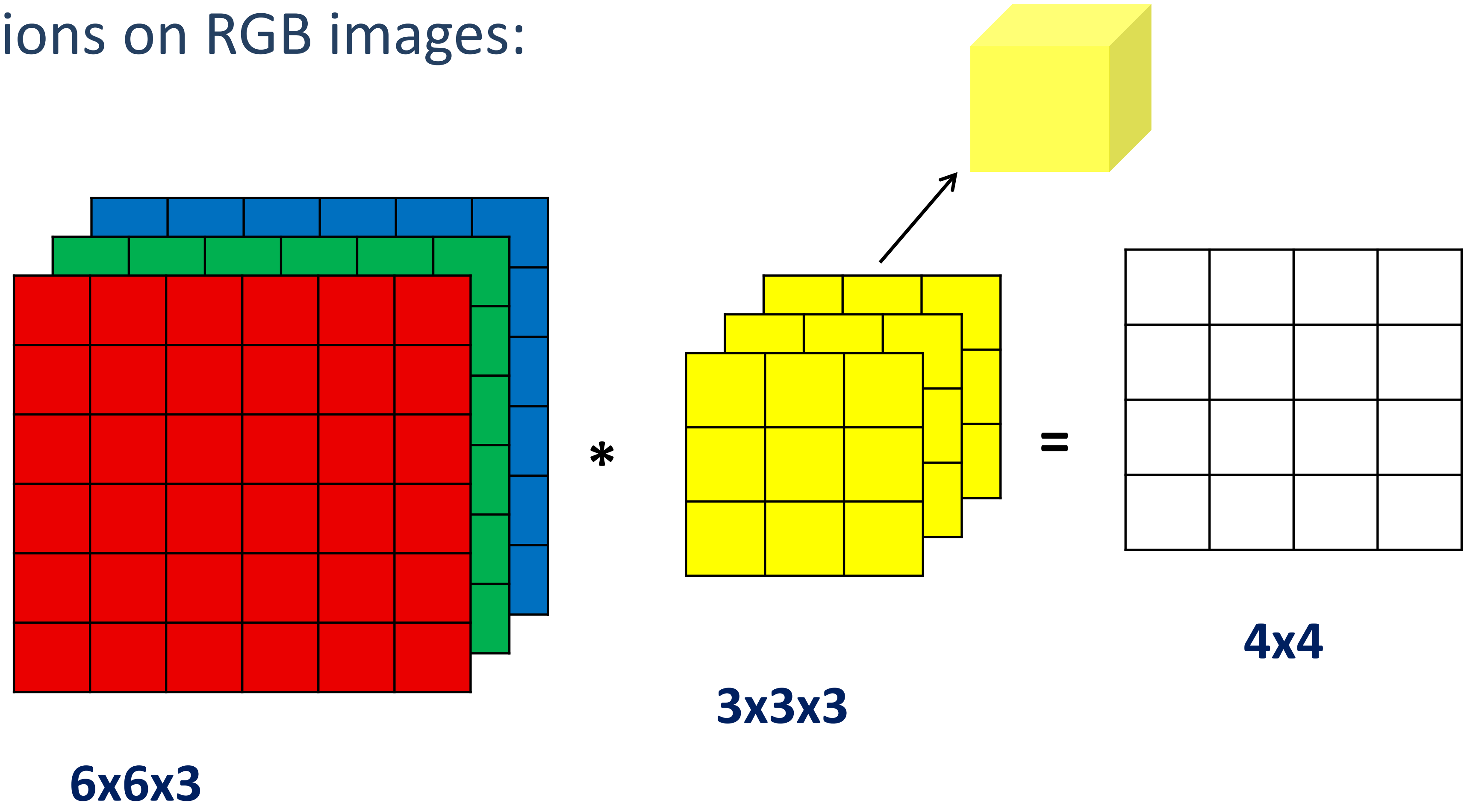
Convolutions over volumes

Convolutions on RGB images:



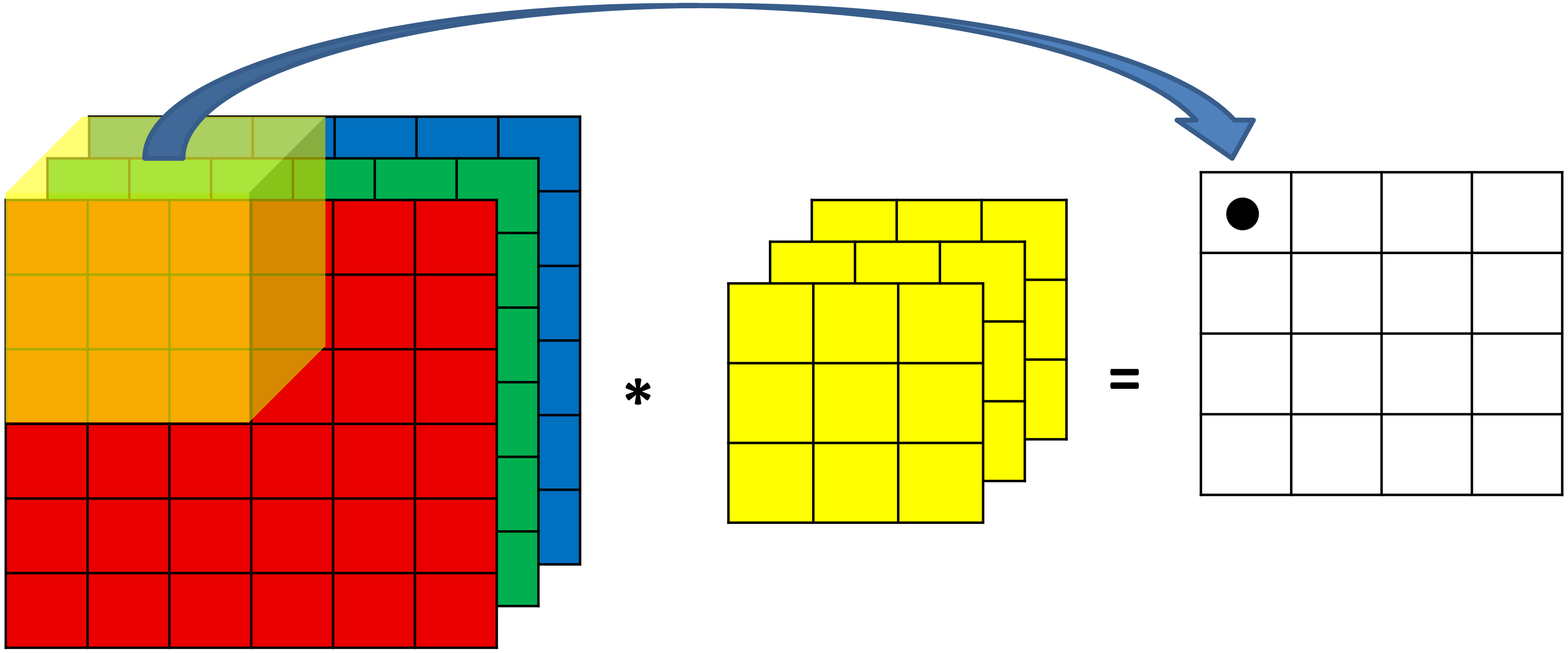
Convolutions over volumes

Convolutions on RGB images:



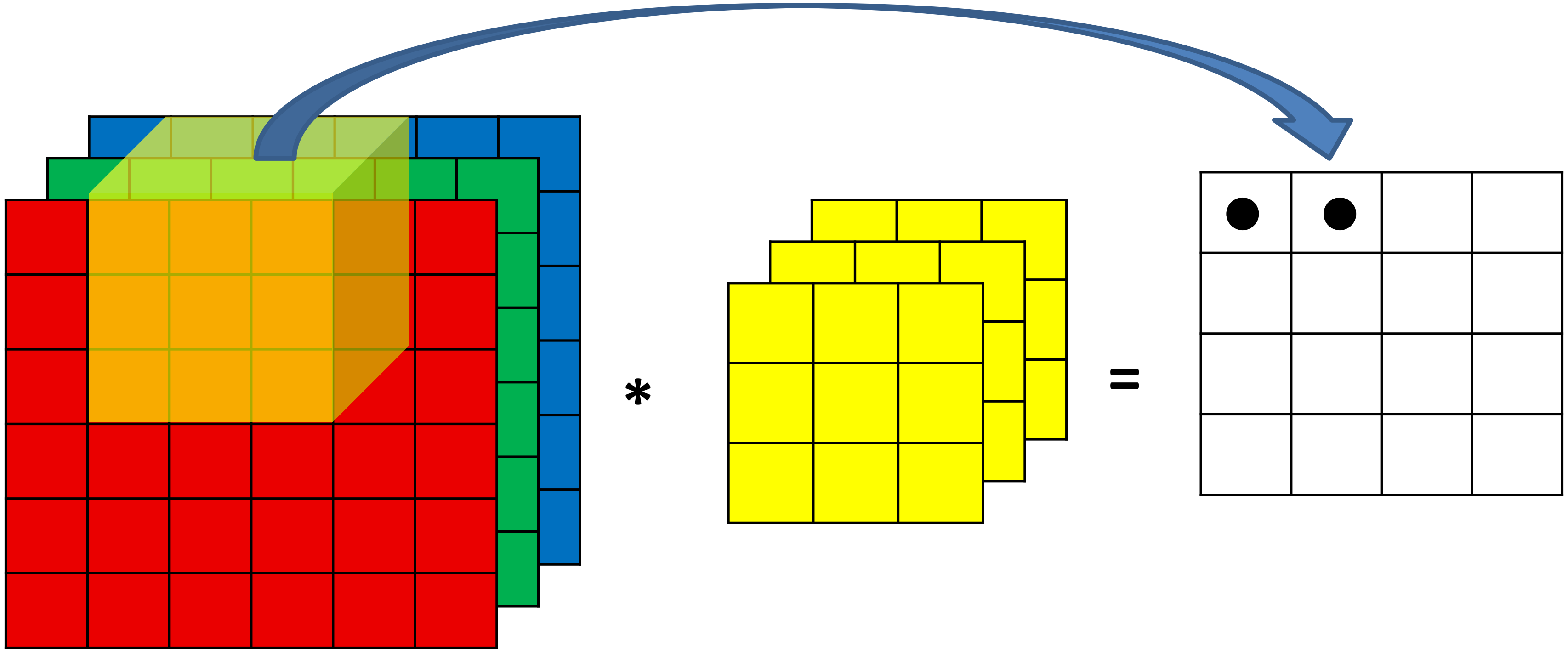
Convolutions over volumes

Convolutions on RGB images:



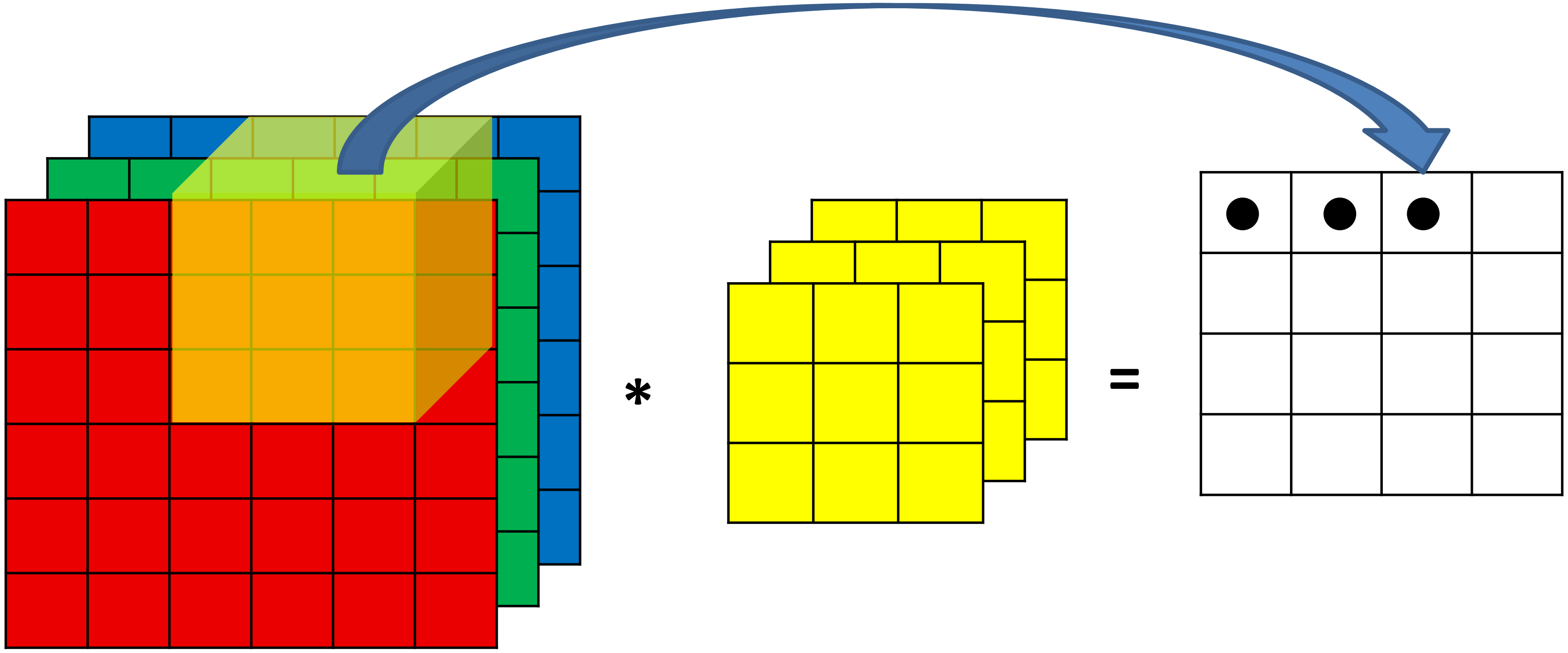
Convolutions over volumes

Convolutions on RGB images:



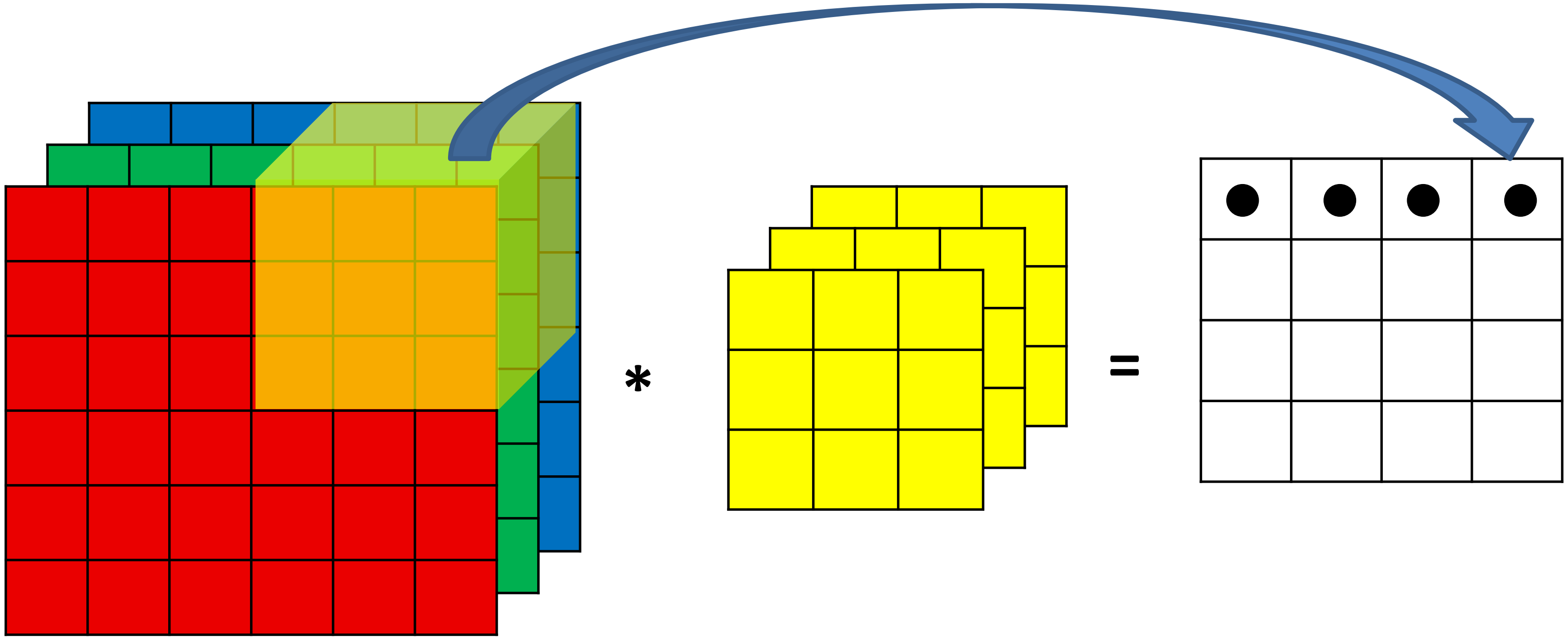
Convolutions over volumes

Convolutions on RGB images:



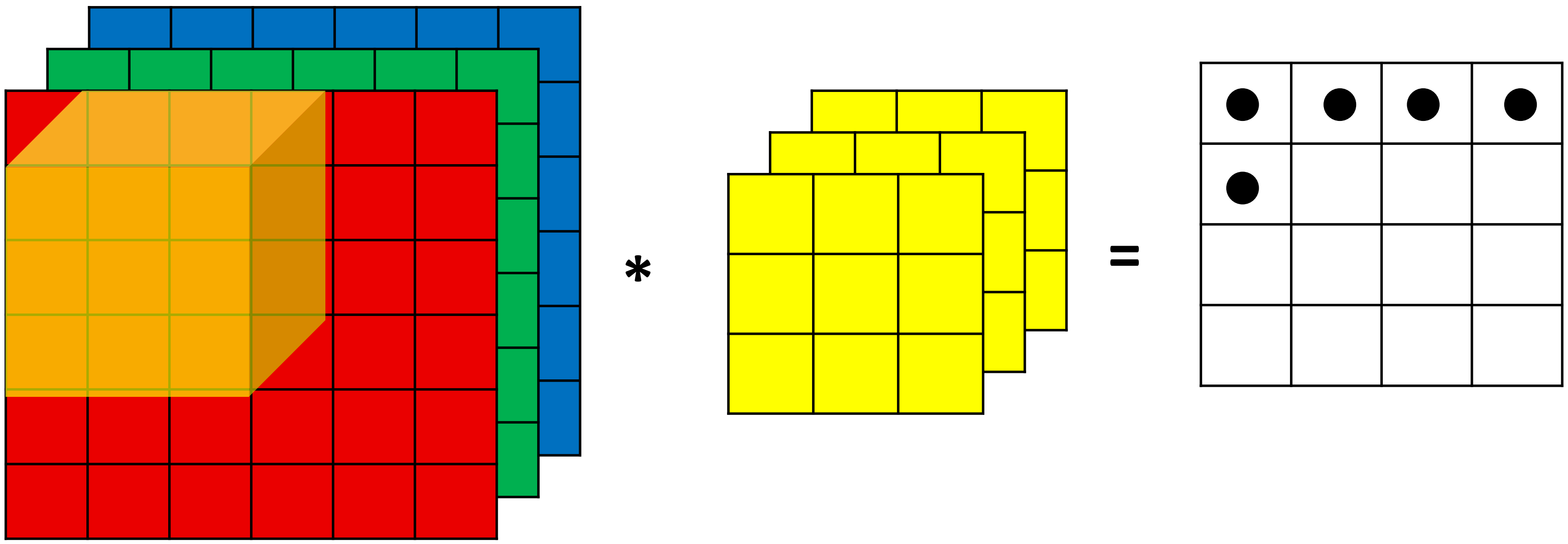
Convolutions over volumes

Convolutions on RGB images:



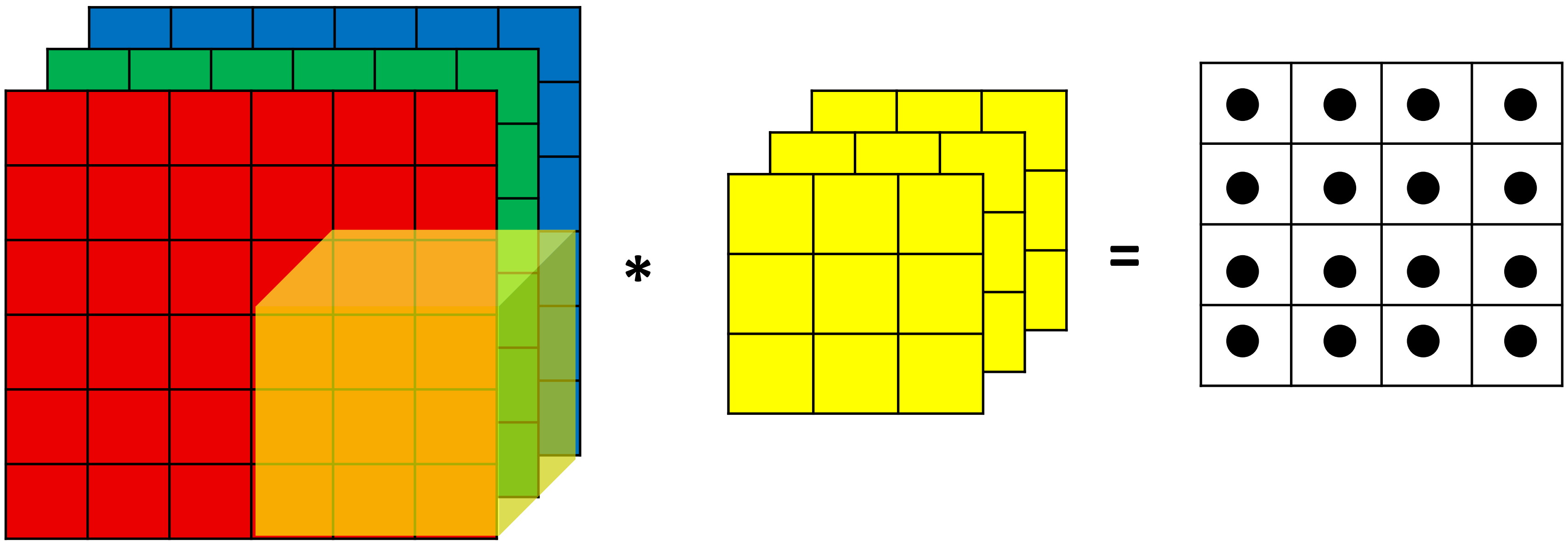
Convolutions over Volumes

Convolutions on RGB images:

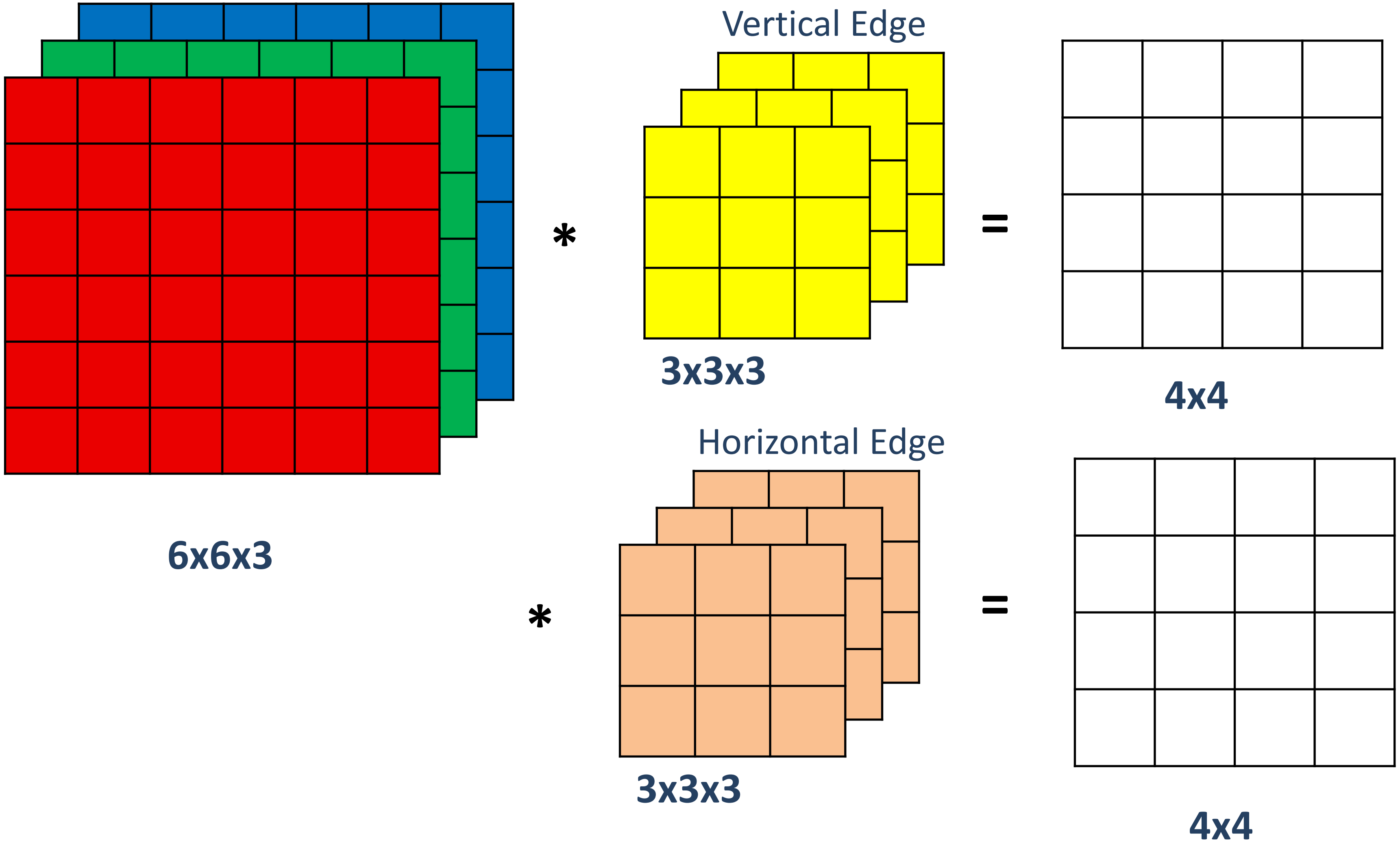


Convolutions over Volumes

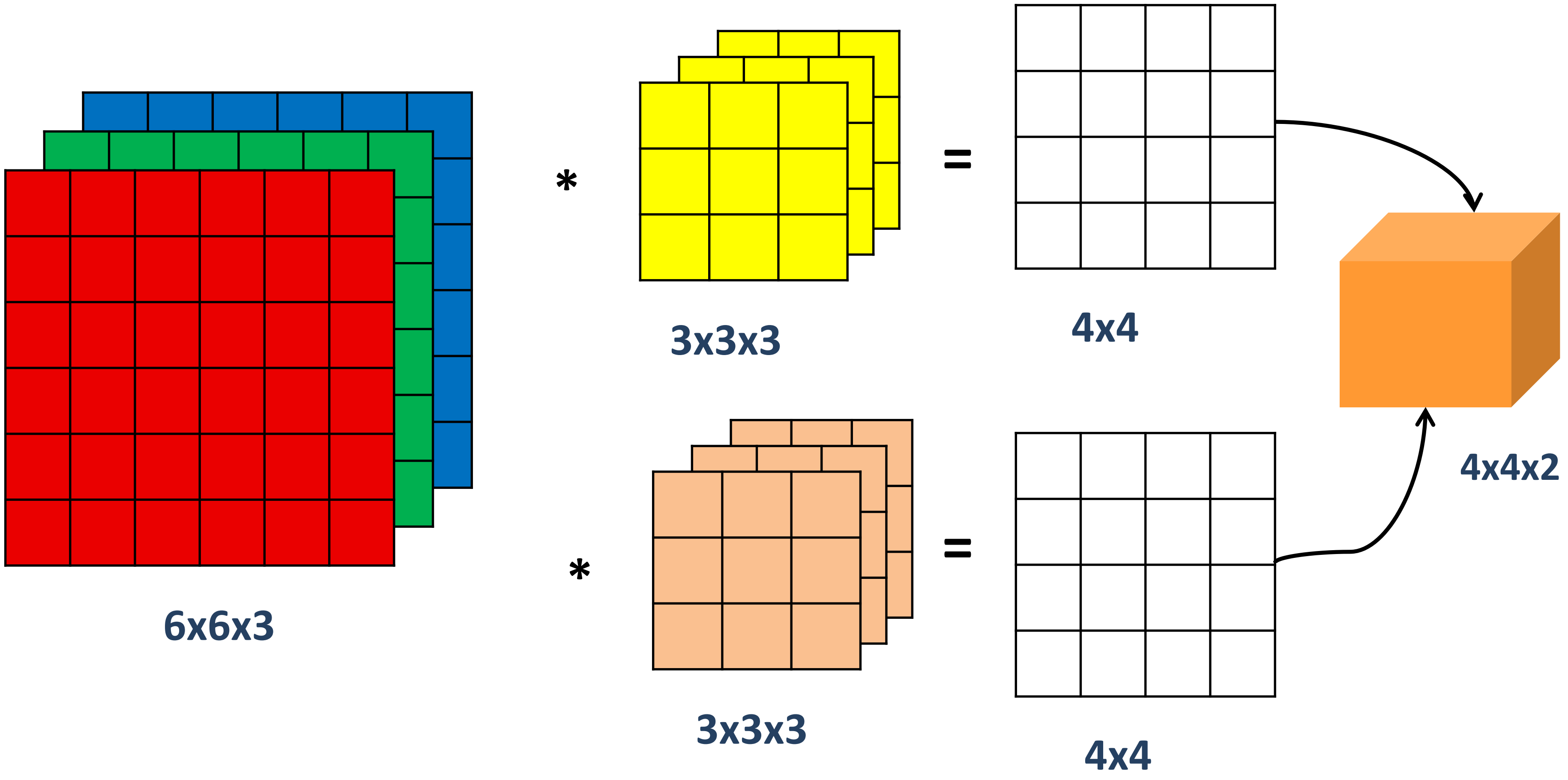
Convolutions on RGB images:



Multiple filters



Multiple filters



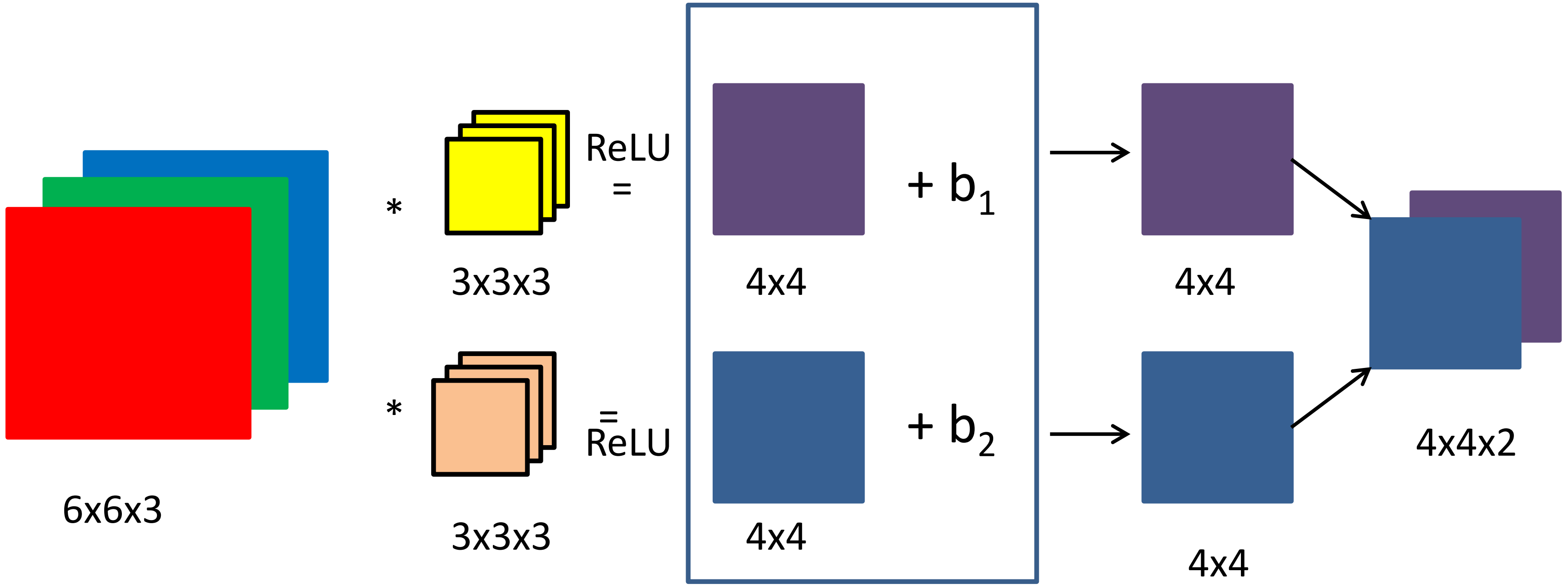
Multiple filters

$$\underbrace{n}_{\text{Height}} \times \underbrace{n}_{\text{Width}} \times \underbrace{n_c}_{\text{\#channels}} * \underbrace{f}_{\text{Height}} \times \underbrace{f}_{\text{Width}} \times \underbrace{n_c}_{\text{\#channels}} = n\text{-}f\text{+}1 \times n\text{-}f\text{+}1 \times \underbrace{N_c'}_{\text{\#filters}}$$

6 x 6 x 3 * 3 x 3 x 3 = 6-3+1 x 6-3+1 x 2

4 x 4 x 2

One layer of a convolutional network



Where b_1 and b_2 is $\mathbb{R}$

Number of parameters in one layer

If you have 10 filters that are 3 x 3 x 3 in one layer of a neural network, how many parameters does that layer have?

- Input image: 6x6x3 # a_0
- 10 Filters: 3x3x3 # W_1
- Result image: 4x4x10 # $W_1 a_0$
- Add b (bias) with 10x1 will get us : 4x4x10 image # $W_1 a_0 + b$
- Apply $ReLU$ will get us: 4x4x10 image # a_1 = RELU($W_1 a_0 + b$)
- In the last result $p=0$, $s=1$
- Hint number of parameters here are: (3x3x3x10) + 10 = 280

Summary of notation

If layer l is a conv layer:

Hyperparameters

$f^{[l]}$ = filter size

$p^{[l]}$ = padding # Default is zero

$s^{[l]}$ = stride

$n_c^{[l]}$ = number of filters

Input: $n^{[l-1]}$ x $n^{[l-1]}$ x $n_c^{[l-1]}$ Or

$n_H^{[l-1]}$ x $n_W^{[l-1]}$ x $n_c^{[l-1]}$

Output: $n^{[l]}$ x $n^{[l]}$ x $n_c^{[l]}$ Or

$n_H^{[l]}$ x $n_W^{[l]}$ x $n_c^{[l]}$

Where $n^{[l]} = (n^{[l-1]} + 2\,p^{[l]} - f^{[l]} / s^{[l]}) + 1$

Each filter is: $f^{[l]}$ x $f^{[l]}$ x $n_c^{[l-1]}$

Activations: $a^{[l]}$ is $n_H^{[l]}$ x $n_W^{[l]}$ x $n_c^{[l]}$

$A^{[l]}$ is m x $n_H^{[l]}$ x $n_W^{[l]}$ x $n_c^{[l]}$

In batch or minbatch training

Weights: $f^{[l]} * f^{[l]} * n_c^{[l-1]} * n_c^{[l-1]}$

bias: $(1, 1, 1, n_c^{[l-1]})$

Typical convolutional network has three stages

- ***Convolution*** — several convolution to produce linear activation
- ***Pooling*** — Output is updated with a summary of statistics of nearby inputs
- ***Fully connected***

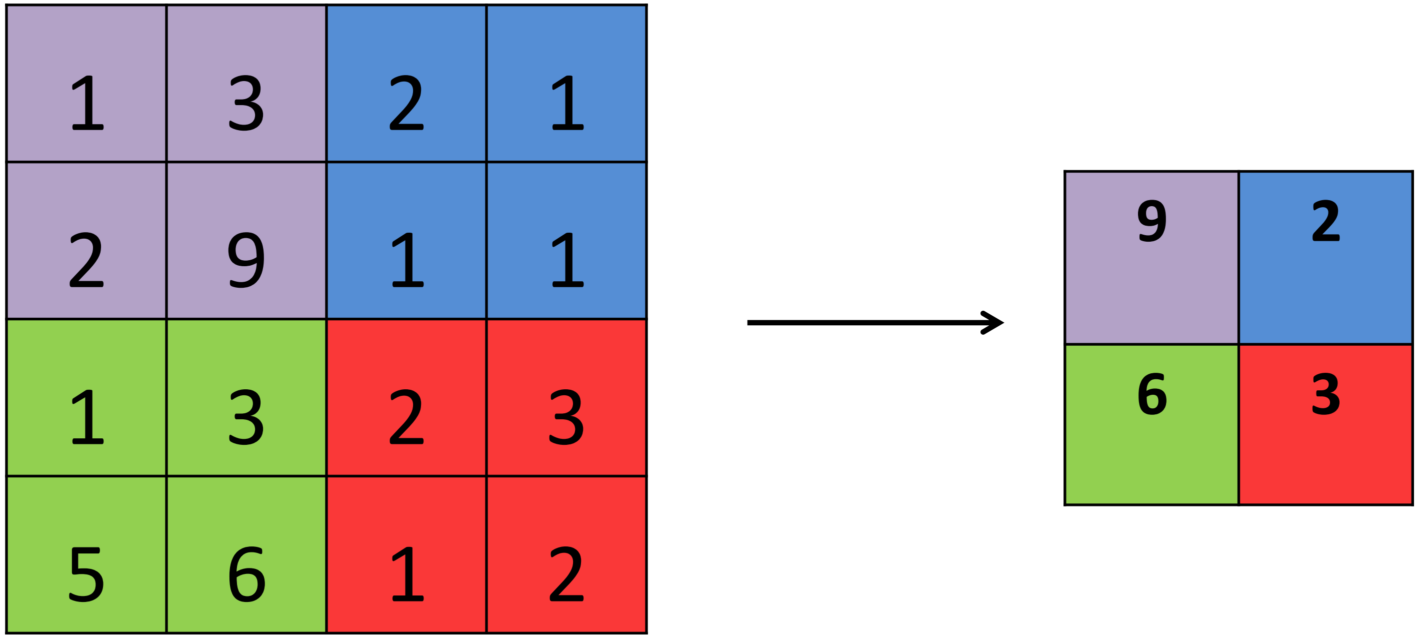
Pooling layers

CNNs often uses pooling layers to reduce the size of the inputs, speed up computation, and to make some of the features it detects more robust.

Pooling layers

Max Pooling:

f = 2, s = 2, and p = 0 hyperparameters



Pooling layers

Average Pooling:

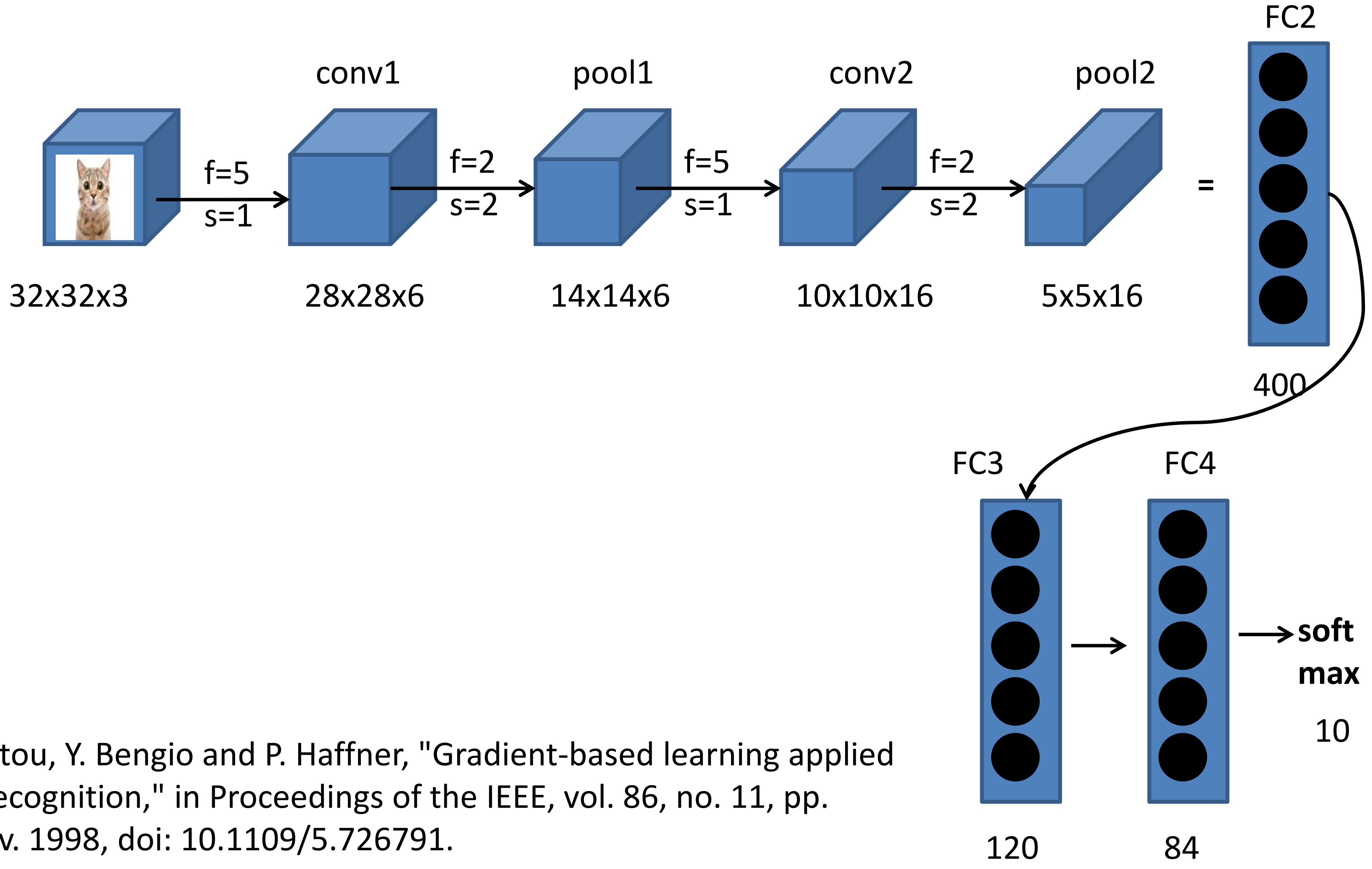
f = 2, s = 2, and p = 0 hyperparameters

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

$\longrightarrow$

3.75	**1.25**
4	**2**

Convolutional neural network example



Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

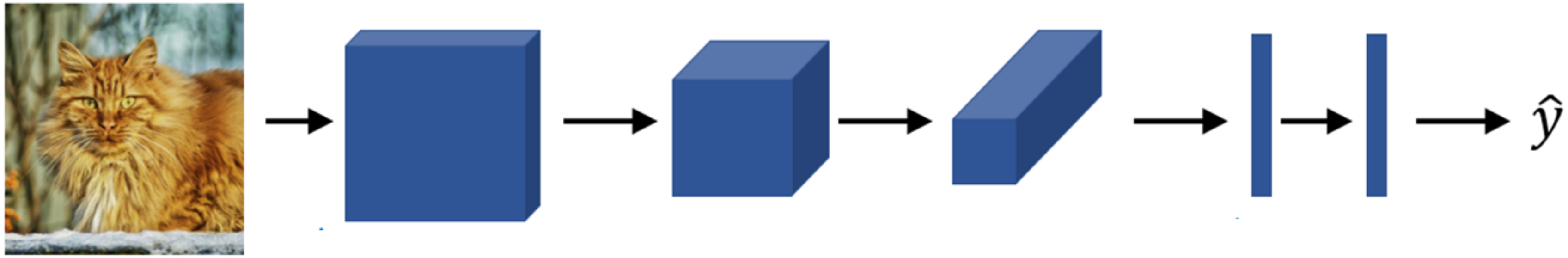
Why convolutions?

- **Parameter sharing**: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

- **Sparsity of connections**: In each layer, each output value depends only on a small number of inputs.

Why convolutions?

Training set $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$.



Cost $J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

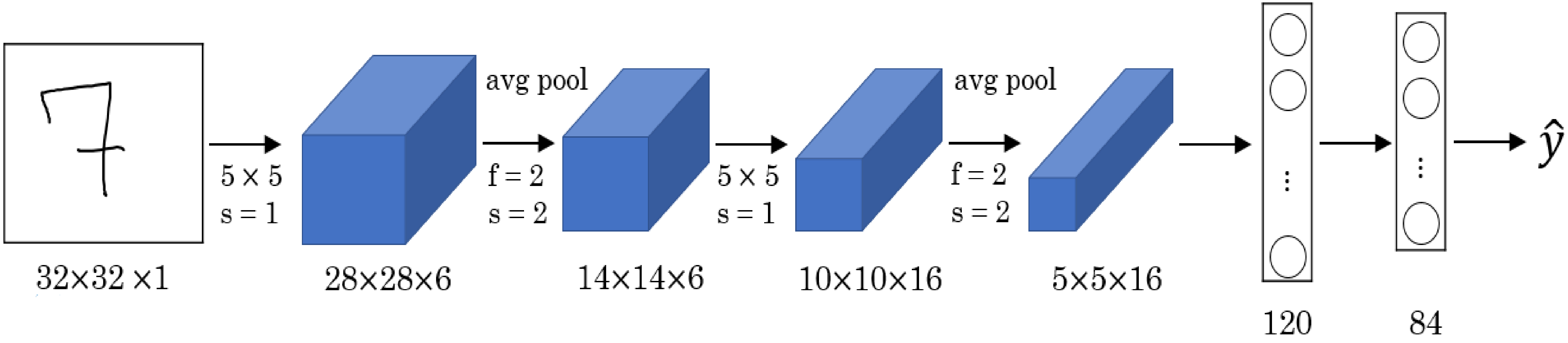
Use gradient descent to optimize parameters to reduce J

Deep convolutional models

Some classical CNN networks:

- **LeNet-5**
- **AlexNet**
- **VGG**
- **ResNet**

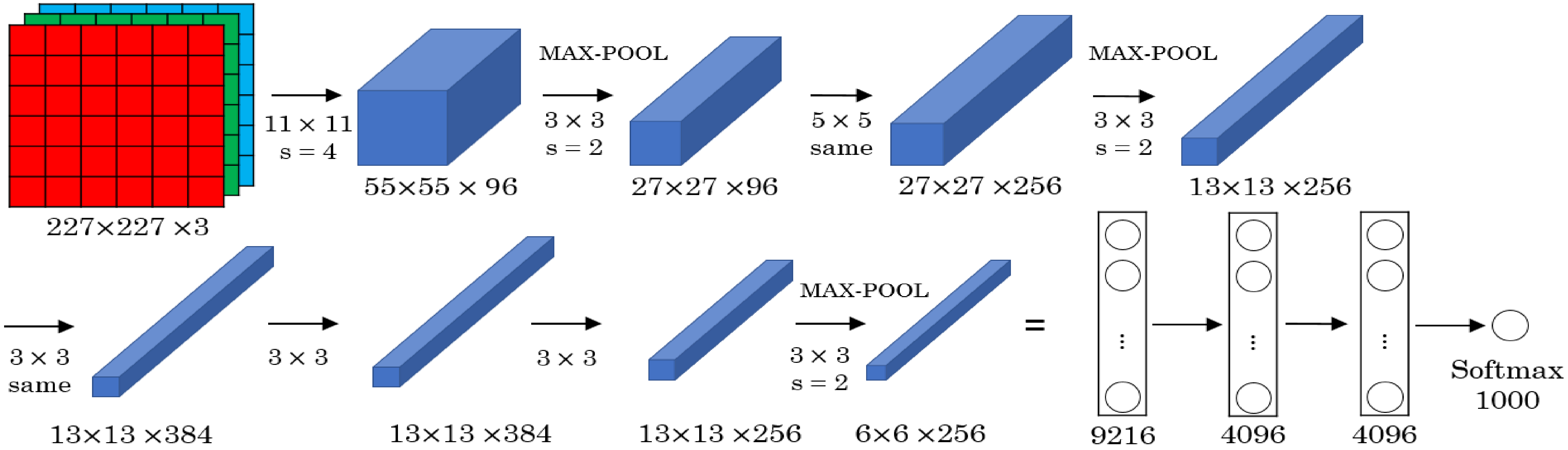
LeNet-5



- It has 60k parameters.
- The dimensions of the image decreases as the number of channels increases.
- Conv ==> Pool ==> Conv ==> Pool ==> FC ==> FC ==> softmax
- The activation function used in the paper was Sigmoid and Tanh. Modern implementation uses RELU in most of the cases

[LeCun et al., 1998. Gradient-based learning applied to document recognition]

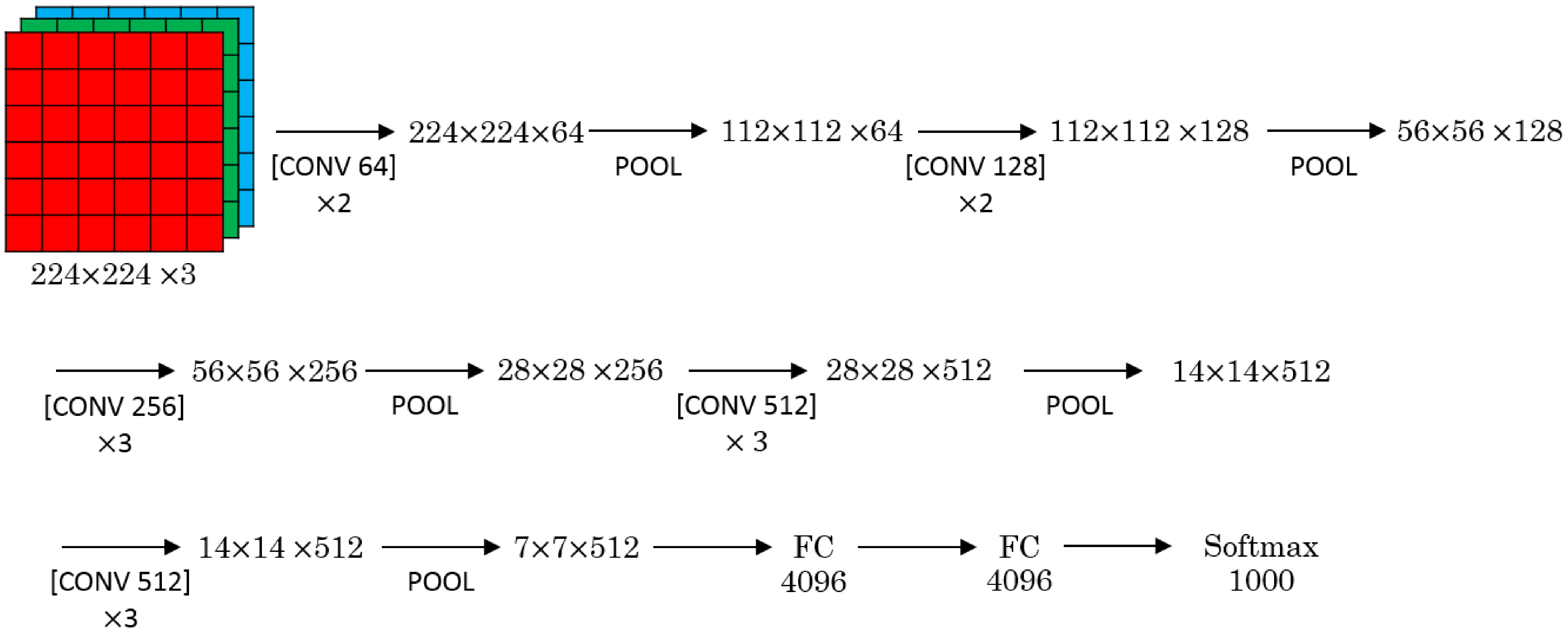
AlexNet



- Has 60 Million parameter compared to 60k parameter of LeNet-5.
- It used the RELU activation function.

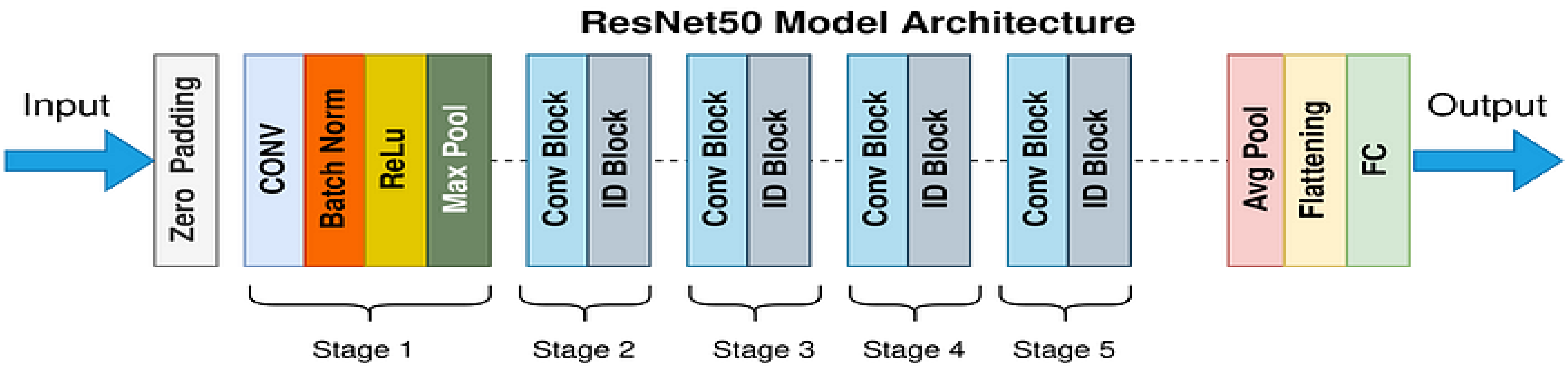
[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks]

VGG-16



[Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition]

ResNet



[He et al., 2015. Deep residual networks for image recognition]

Thank You