CS 4375-03
Lecture 3-C: Convolutional Neural Networks

Feng Chen

University of Texas at Dallas

(Slides adapted from Andrew NG)

Computer Vision Problems

Image Classification

64x64

→ Cat? (0/1)

Nowadays, many real-world applications, such as virtual reality glamming and self-driving car, are driving by deep learning. For computer vision, convolutional neural networks (or CNN) is the most successful DL network for these applications.

One example is to build an image classification model to recognize cat in an image.

Computer Vision Problems

Image Classification

64x64

Cat? (0/1)

Object detection

Another example is object detection that is critical to build a self-driving car. In this image, you need to detection the location of your car and the locations of other cars in order to avoid them.

Computer Vision Problems

Image Classification

64x64

→ Cat? (0/1)

Object detection

Computer Vision Problems

Image Classification

64x64

→ Cat? (0/1)

Neural Style Transfer

Object detection

The third example is to repaint an image in different style. For example, the left figure is a human face and the right figure is a paint by Picasso. We want to repaint the left figure using the style of the right figure.

Computer Vision Problems

Image Classification

64x64

Cat? (0/1)

Object detection

Neural Style Transfer

One of the challenges in computer vision is that the input can get really big. For example, the top cat image has the dimension 64 by 64 by 3 because for each pixel we have R, G, B, three colors. If we multiply them out, the input feature has the dimension 12288. That is not too bad for this image. However, if you work on a large image, such as the cat image shown on the bottom, it has 1000 pixel by 1000 pixel and just has 1 MB file size. The dimension of this image will be 1000 by 1000 by 3 because we have R, G, B, three channels, that is equal to 3 million pixels in total.

Deep learning on large images

Image Classification

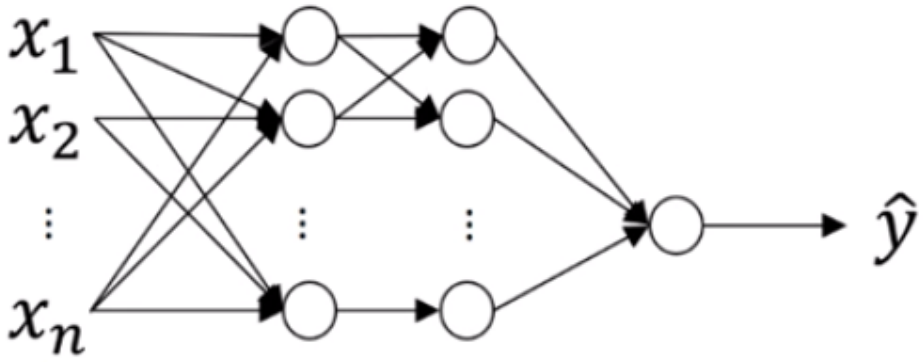
Cat? (0/1)

64x64x3 -> 12288 variables

To consider 3 million features as the input to a fully connected neural network, then the weight matrix for the first latent layer will have the dimension 1000 by 3 million that is equal to 3 billion parameters in total. This is very large. With so many parameters, it will be very difficult to find enough training data to prevent the neural network from overfitting. In addition, both the computational and memory requirement are also in feasible in regular computers.

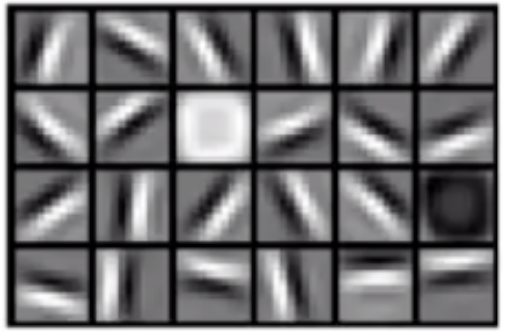
1000x1000x3 = 3 M

$W \in \mathbb{R}^{1000 \times 3\mathrm{M}}$



3 M

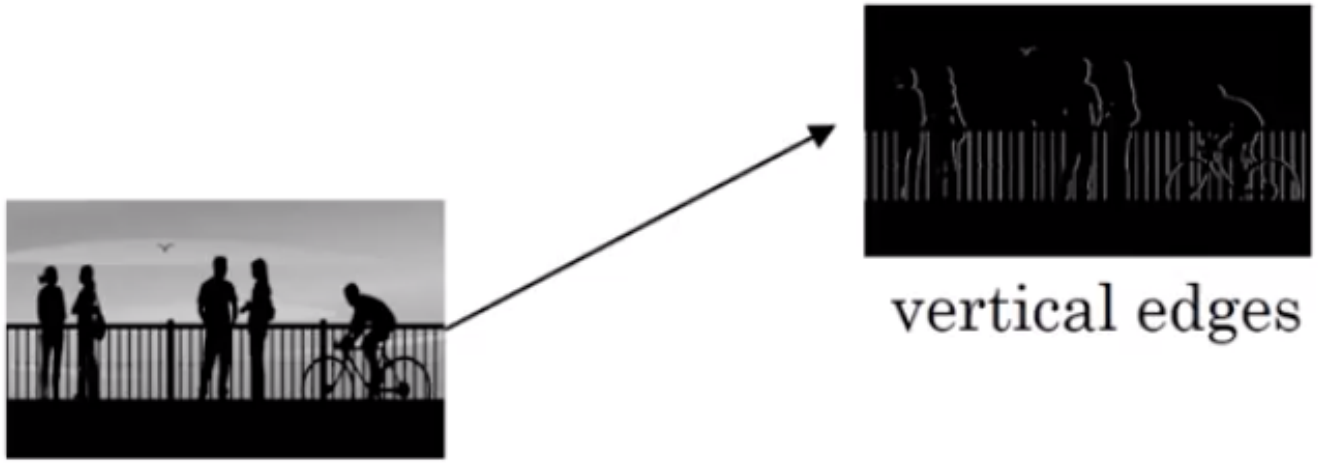
1000 neurons



The first convolutional layer detects edges

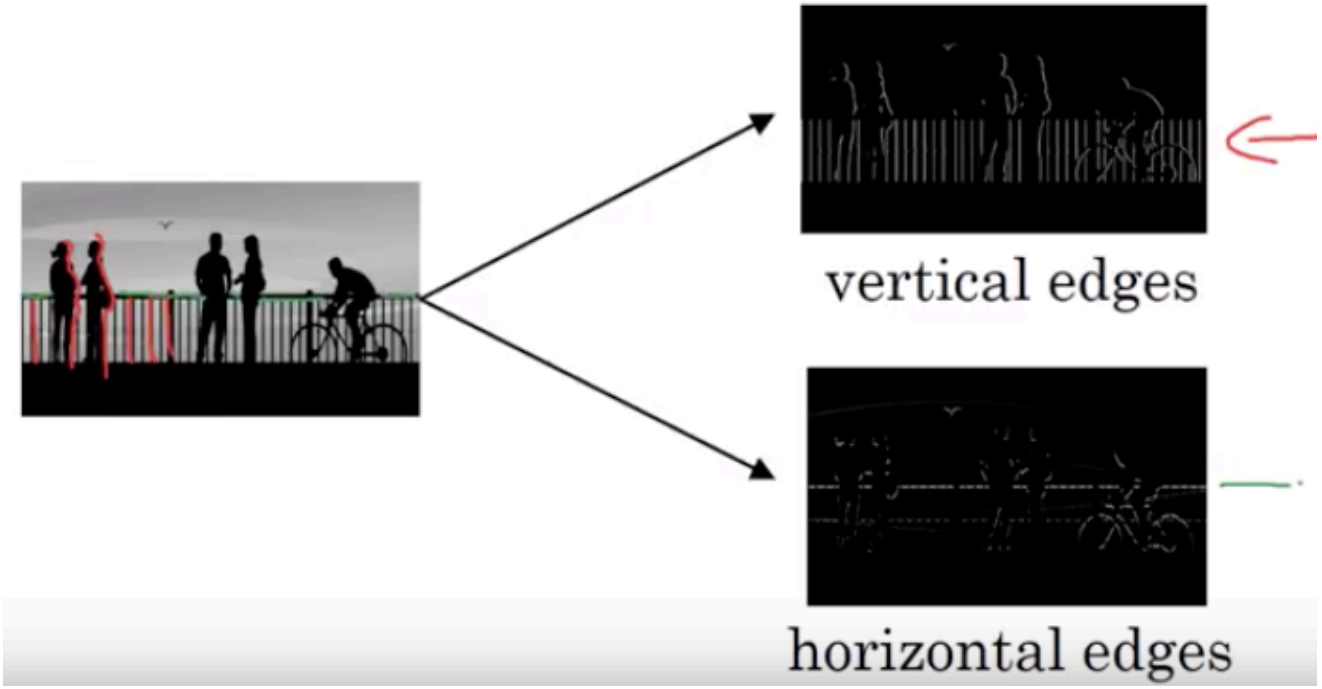
The second convolutional layer detects partial objects

The third convolutional layer may detect complete objects like human faces in this case.



For compute vision applications, we often need to use high-resolution images. In order to do that, we need to implement a convolutional operator to extract meaningful features from the image, which is one of the fundamental building blocks of convolutional neural networks. In this page, the example on the top row illustrate convolutions using the example of edge detection. Here, I show three layers of convolutional neural networks for face recognition. The first convolutional layer detects edges, the second convolutional layer detects partial objects. The third convolutional layer may detect …

For the second example, let us see how we can detect edges in the bottom image. We want computer to figure out what are objects in this image. The first thing you might do is to detect vertical edges shown on the bottom right for example. It has all those vertical lines around the railings and pedestrians.



We might also want to detect horizontal edges. In this image there is a very strong horizontal line around the railing and the bike. This example indicates the usefulness of edge detection. Now the question is: how can we detect vertical and horizontal edges from the input image like this one?

Vertical edge detection

Let us look at an example here. We have a six pixel by six pixel gray scale image. We can represent this image as six by six matrix, rather than six by six by three because there are no R, G, and G channels in this image.

Let us first consider the detection of vertical edges in this image. What you can do is to construct a three by three matrix. In terminology of convolutional neural networks, we also call this matrix a filter. I am going to construct the 3 by 3 filter that looks like this. Some papers also call this matrix a kernel instead of a filter. In the slides, I will use the filter terminology.

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

$*$

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

$=$

Vertical edge detection

In order to calculate the convoluted matrix based on this filter. We take the 3 by 3 filter and place it on top of the three by three region of the original image.

We take the element wise product: 3x1 + 1 x 1 + 2 x 1 + 0 x 0 + 5 x 0 + 7 x 0 + 1 x -1 + 8 x -1 + 2 x -1 = -5.
We can add these numbers in any order.

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

*

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

=

-5			

Vertical edge detection

We now shift the block one step to the right and place the filter on top of the region. We do the same element-wise product and then do addition. We have…

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

$*$

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

$=$

-5	-4		

Vertical edge detection

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

$*$

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

$=$

-5	-4	0	

Vertical edge detection

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

$*$

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

$=$

-5	-4	0	8

Vertical edge detection

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

$*$

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

$=$

-5	-4	0	8
-10			

Vertical edge detection

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

$*$

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

$=$

-5	-4	0	8
-10	-2		

Vertical edge detection

After we calculate all the convoluted values, what we obtain is an edge detector. You can easily see why it is called edge detector on the next page.

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

$*$

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

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

$=$

Suppose the input image is shown on the left.

Vertical edge detection

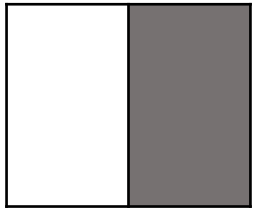
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

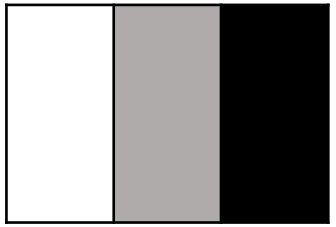
We can visualize both the input image and the filter on the bottom. -1 refers to black, 0 refers to grey, a positive value refers to white. We can clearly see there is an edge in the middle line of the input image.

$*$

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

$=$





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

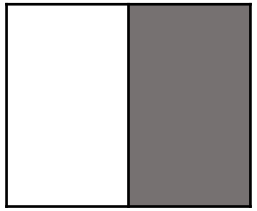
This page shows the output matrix after we conduction the convolution based on this filter.

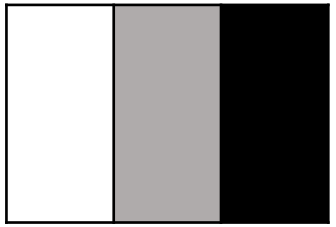
$*$

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

$=$

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





Vertical edge detection

The bottom shows the visualization of the output matrix. The white region in the middle has clearly highlighted the middle edge region in the input image.

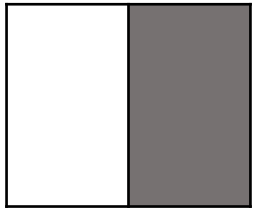
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

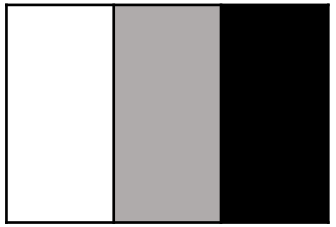
*

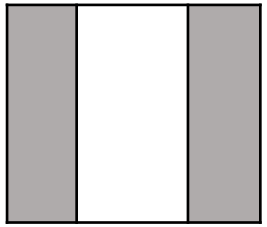
1	0	-1
1	0	-1
1	0	-1

=

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







Vertical edge detection

In this next page, I will use a different input image that also has a edge in the middle.

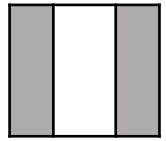
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

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

=

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



Vertical edge detection

We want to check if the filter can still detect the edge region in the middle.

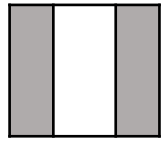
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

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

=

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



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

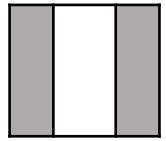
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

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

=

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



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

This page shows the output matrix of of this new input image.

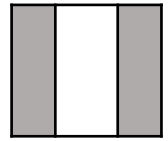
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

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

=

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



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

=

0	-30	-30	0
0	-30	-30	0
0	-30	-30	0
0	-30	-30	0

Vertical edge detection

The visualization of the output image clearly indicates that it can detect the edge region in the middle.

To summarize, a filter with edge detection is able to highlight the edge region with a different color.

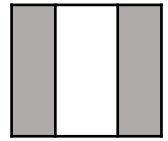
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

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

=

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



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

=

0	-30	-30	0
0	-30	-30	0
0	-30	-30	0
0	-30	-30	0

Vertical and horizontal edge detection

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

Vertical

1	**1**	**1**
0	**0**	**0**
-1	**-1**	**-1**

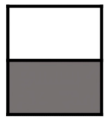
Horizontal

We have demonstrated that the left filter is able to detect vertical edges. In the following slides, I will show that the right filter is able to detect horizontal edges.

horizontal edge detection

In this page, we have two different input images that both have horizontal edge in the middle. We apply the same filter for horizontal edge detection. The output matrices demonstrate that this filter is able to detection the horizontal edge regions in both input images as well.

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



*

1	**1**	**1**
0	**0**	**0**
-1	**-1**	**-1**

Horizontal

=

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

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

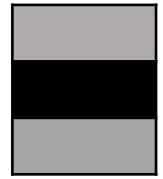
*

1	**1**	**1**
0	**0**	**0**
-1	**-1**	**-1**

Horizontal

=

0	0	0	0
-30	-30	-30	-30
-30	-30	-30	-30
0	0	0	0



horizontal edge detection

If we apply the filter to input images that do not have horizontal edges, but only vertical edges, the output matrix will be a zero matrix. That indicates that there are no edge regions in the input images.

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	**1**	**1**
0	**0**	**0**
-1	**-1**	**-1**

Horizontal

=

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

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	**1**	**1**
0	**0**	**0**
-1	**-1**	**-1**

Horizontal

=

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

Vertical and horizontal edge detection

In this page, the input image has both horizontal and vertical edge regions. If we apply a filter for horizontal edge detection, we can then detect only the horizontal edge region in the input page as shown in the next page.

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

Vertical

1	**1**	**1**
0	**0**	**0**
-1	**-1**	**-1**

Horizontal

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

*

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

Vertical and horizontal edge detection

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

Vertical

1	**1**	**1**
0	**0**	**0**
-1	**-1**	**-1**

Horizontal

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

*

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

=

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

Learning to detect edges

put a little bit more weight on the central pixel

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

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

sobel filter

In the previous slides, I have demonstrated the filter shown on the left for vertical edge detection. We can obtain a different filter by putting a little more weight on the center pixels. This filter is called sobel filter that can better detect edges that are not directly vertical as shown on the example on the bottom.

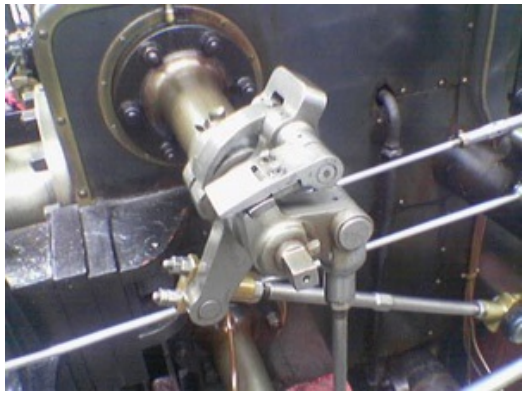
Learning to detect edges

put a little bit more weight on the central pixel

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

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

Sobel filter



* Sobel filter =

Learning to detect edges

put a little bit more weight on the central pixel

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

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

sobel filter

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

scharr filter

instead of only 1, 0, -1 we can use other excessive numbers as well.

Instead of using only 1, 0 and -1, we can use other excessive numbers as well, such as 3, 10, 3, and then -3, -10, and -3. We obtain a well-known filter called scharr filter that has swiftly different property.

Learning to detect edges

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

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

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

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

(6 x 6)

With the rise of deep learning, one of things we learned is that, when you really want to detect edges in some complicated images maybe you can just learn the filter and treat the 9 numbers of this filter matrix as parameters, so that we can take the six by six image and convolute it with your 3 by 3 filter that gives you a good edge detector. We can then learn one of the filters that we have introduced, or learn even better ones directly from the data.

$*$

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

(3×3)

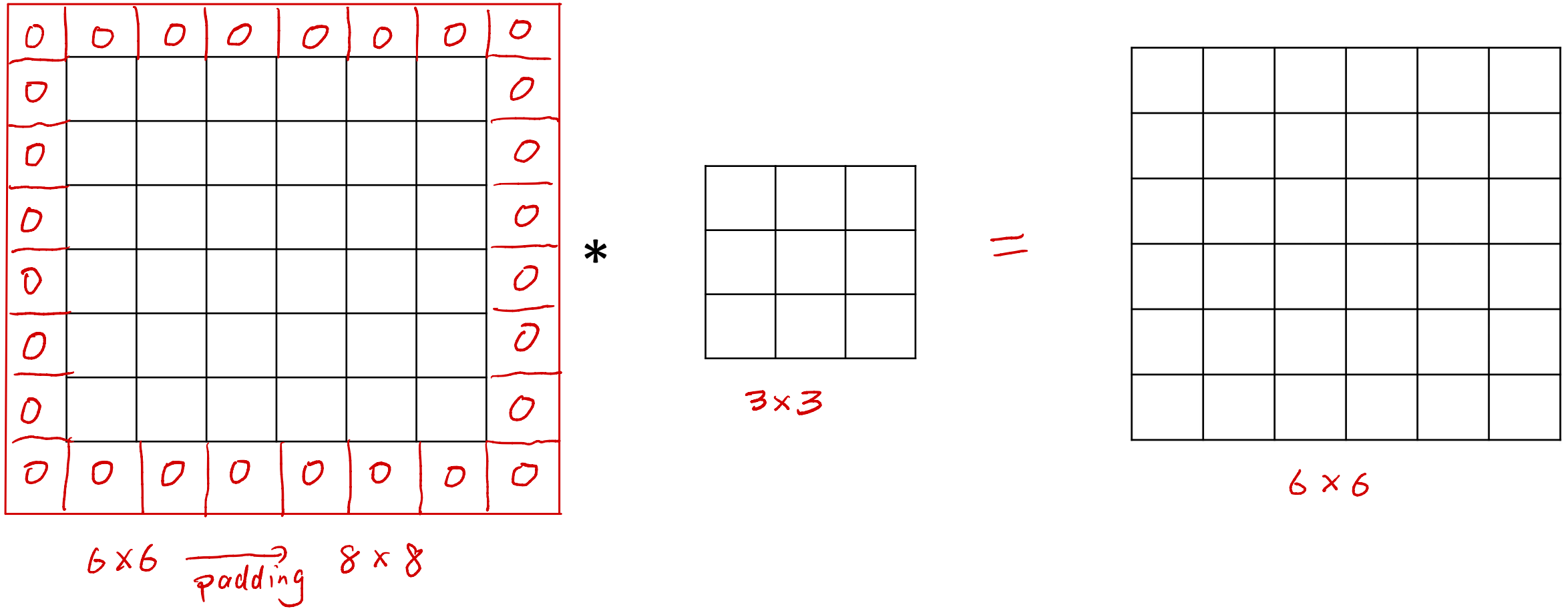
$W \in \mathbb{R}^{3\times3}$

$=$

(4 x 4)

Not matter what is the filter matrix, the output size will shrink the size by 2: from 6 to 4. We need an operation called padding that can keep the size the same.

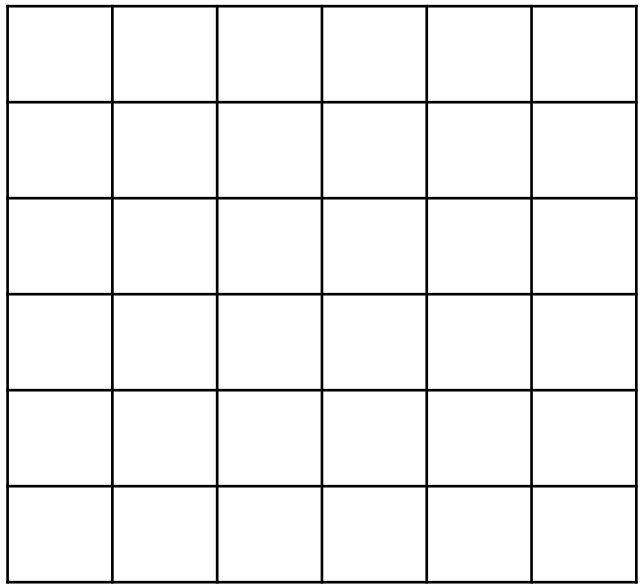
Padding



You may have noticed that the input image will shrink after one convolution. For example, given an 6 by 6 image as input, the output image after one convolution will be 4 by 4. The image will get really small after several steps of convolutions. We want to avoid that. That means we do not want image to shrink every time we do convolution. To address this we can use padding. The idea is to pad the image with an additional border of zero pixels around the edges.

In this example, after padding the zero pixels, the size of the output image is now 6 by 6, the same size of the input image.

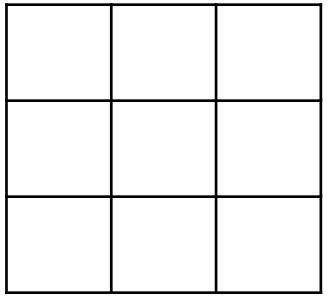
Padding



6 x 6

(n x n)

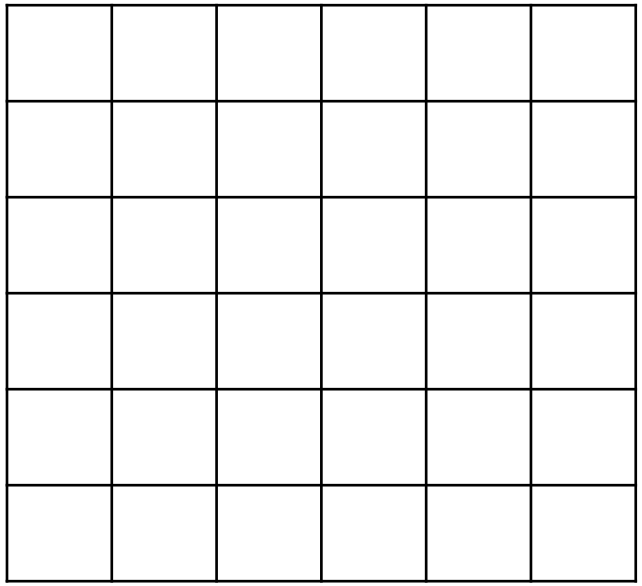
*



3 x 3

(f x f)

=



6 x 6

$(n+2p-f+1) \times (n+2p-f+1)$

$6+2\cdot1-3+1$

6

padding (p) = 1

Suppose the input image is n by n and the filter is f by f, there is a simple formula to calculate the output image size: n + 2p - f + 1. Here p refers to the number of rows of zero padding. In this example, p is equal to 1. In this example, as n = 6 and f = 3, using the formula we can calculate that the output image size 6 + 2 - 3 + 1 = 6.

Valid and Same convolutions

"convolution without padding" :

"convolution with padding (p)" :

"Same" : Pad so that output size is the same as the input size

In this previous page, given an input image of size 6 by 6 and a filter of size , by padding one row of zero pixels, the output image has the same input size. Now, given input image of any size, we need to a general rule to pad the input image such that the output size is the same as the input size.

Valid and Same convolutions

Input Image: $n \times n$, Filter: $f \times f$

"convolution without padding" : $(n - f + 1) \times (n - f + 1)$

"convolution with padding (p)" : $(n + 2 \cdot p - f + 1) \times (n + 2 \cdot p - f + 1)$

"Same" : Pad so that output size is the same as the input size

$$\left\lfloor \frac{f-1}{2} \right\rfloor$$

The question is to decide the value of f, such that n + 3p - f + 1 is equal to n.
By solving this simple equation, we can find that the pad size p is only decided by the filter size f. If f is an odd number, than the pad size p should be equal to f-2/2. If f is an even number, than it is not possible to ensure the same input and output size. In this case, we take the floor of f-1/2, ensuring that the output size is no greater than the input size.

decide the value of f such that

$n + 2p - f + 1 = n$

$\Rightarrow 2p - f + 1 = 0$

$\Rightarrow p = \left\lfloor \frac{f-1}{2} \right\rfloor$

f is often an odd number.

Strided convolution

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

*

3	4	4
1	0	2
-1	0	3

=

Strided convolution is another piece of the basic building block of convolutions as used in convolutional neural networks. Let me show you an example. Let's say you want to convolute this seven by seven image with a 3 by 3 filter shown in the middle. Instead of doing the usual way, we are going to do it with a stride of two. That means you take the element wise production as usual in the upper left. Then instead of stepping the shaded box over by one step, we are going to jump it over by two steps.

Strided convolution

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

*

3	4	4
1	0	2
-1	0	3

=

Strided convolution

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

*

3	4	4
1	0	2
-1	0	3

=

91		

Strided convolution

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

*

3	4	4
1	0	2
-1	0	3

=

91	100	

Strided convolution

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

*

3	4	4
1	0	2
-1	0	3

=

91	100	83

Strided convolution

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

*

3	4	4
1	0	2
-1	0	3

=

91	100	83
69		

Strided convolution

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

*

3	4	4
1	0	2
-1	0	3

=

91	100	83
69	91	

Strided convolution

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

*

3	4	4
1	0	2
-1	0	3

=

91	100	83
69	91	127

Strided convolution

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

*

3	4	4
1	0	2
-1	0	3

=

91	100	83
69	91	127
44		

Strided convolution

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

*

3	4	4
1	0	2
-1	0	3

=

91	100	83
69	91	127
44	72	

Strided convolution (s = 2)

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

7 x 7

*

3	4	4
1	0	2
-1	0	3

3 x 3

=

91	100	83
69	91	127
44	72	74

3 x 3

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

7×7 * 3×3 = $\underbrace{\left\lfloor \frac{7+0-3}{2} + 1 \right\rfloor}_{3} \times \underbrace{\left\lfloor \frac{7+0-3}{2} + 1 \right\rfloor}_{3}$

Summary of notation: padding only

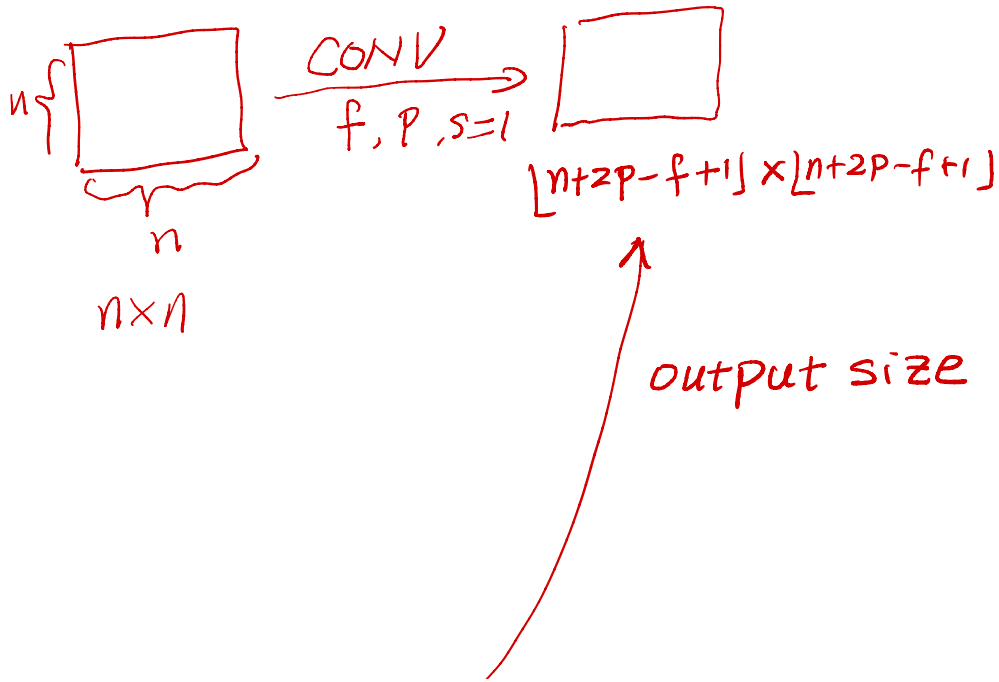
n x n image

f x f filter

padding p

stride 1 (default)

output size



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

Now I am ready to provide a summary of notations for padding only. Given a n by n input image and a f by f filter, suppose the padding size is p and the stride size is 1 (the default), then the output size is shown on the bottom.

Summary of notation: stride only

n x n image

f x f filter

padding 0

stride s

output size

$$\left\lfloor \frac{n + 2 \cdot 0 - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n + 2 \cdot 0 - f}{s} + 1 \right\rfloor$$

If padding p = 0 and the stride is s, then the output size is shown on the new formula on the bottom.

Summary of notation: padding + stride

n x n image

f x f filter

padding p

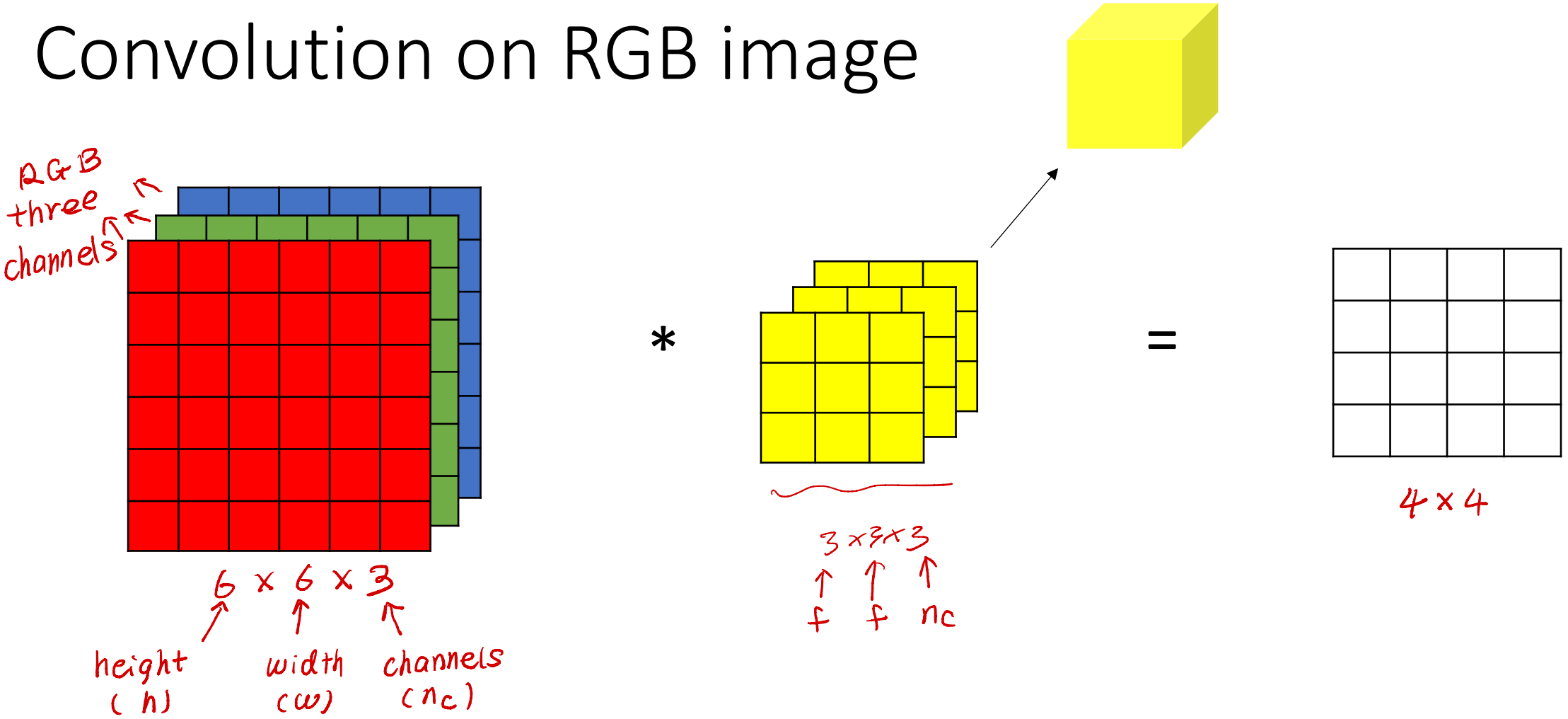
stride s

output size

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

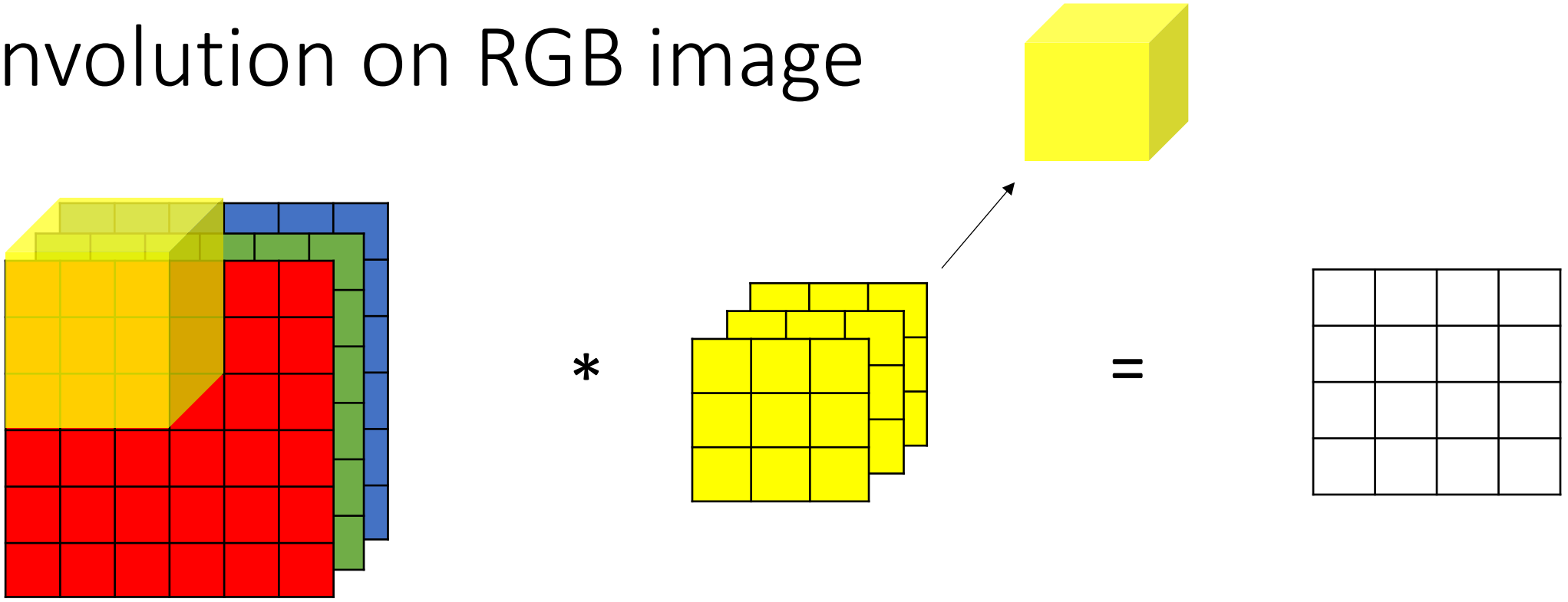
Considering both padding p and stride s, then we have the formula to calculate the outsize as:
the food of (n+2p -f)/s +1.

Convolution on RGB image

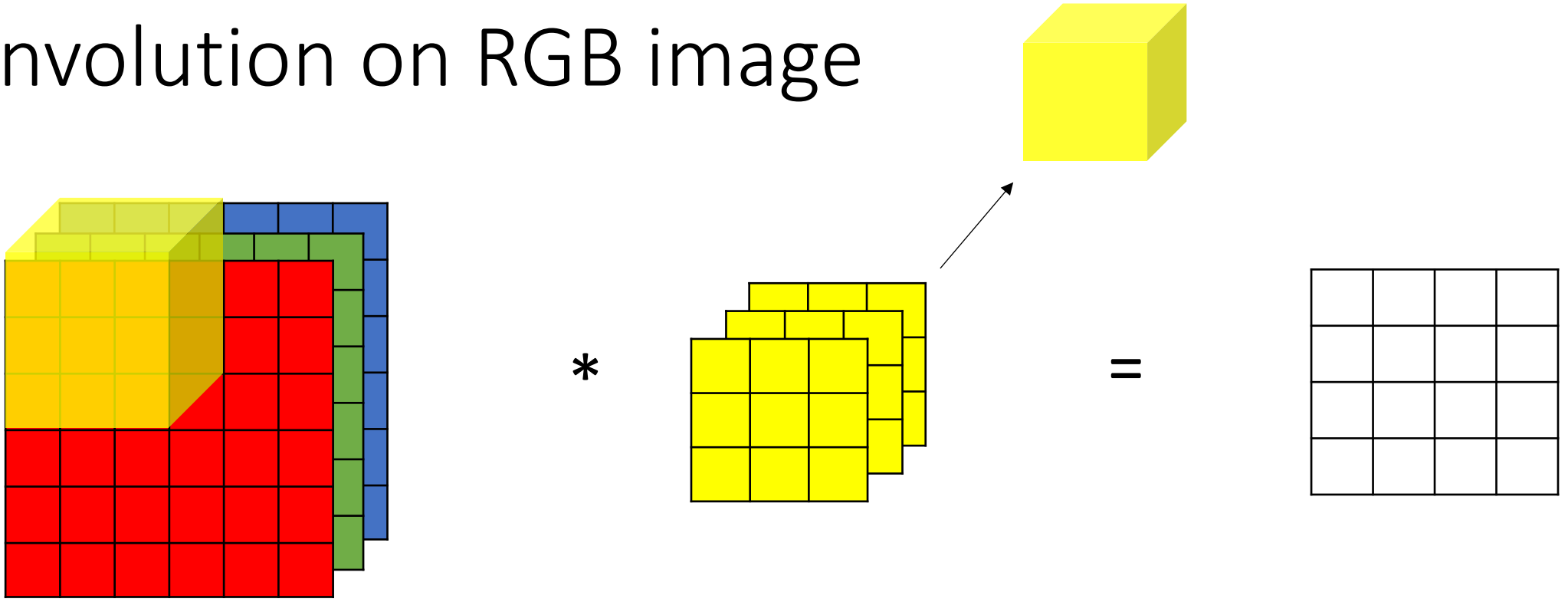


You have seen how we do convolution in 2D images. Here, I will show how to do convolution on 3D volumes or RGB image.

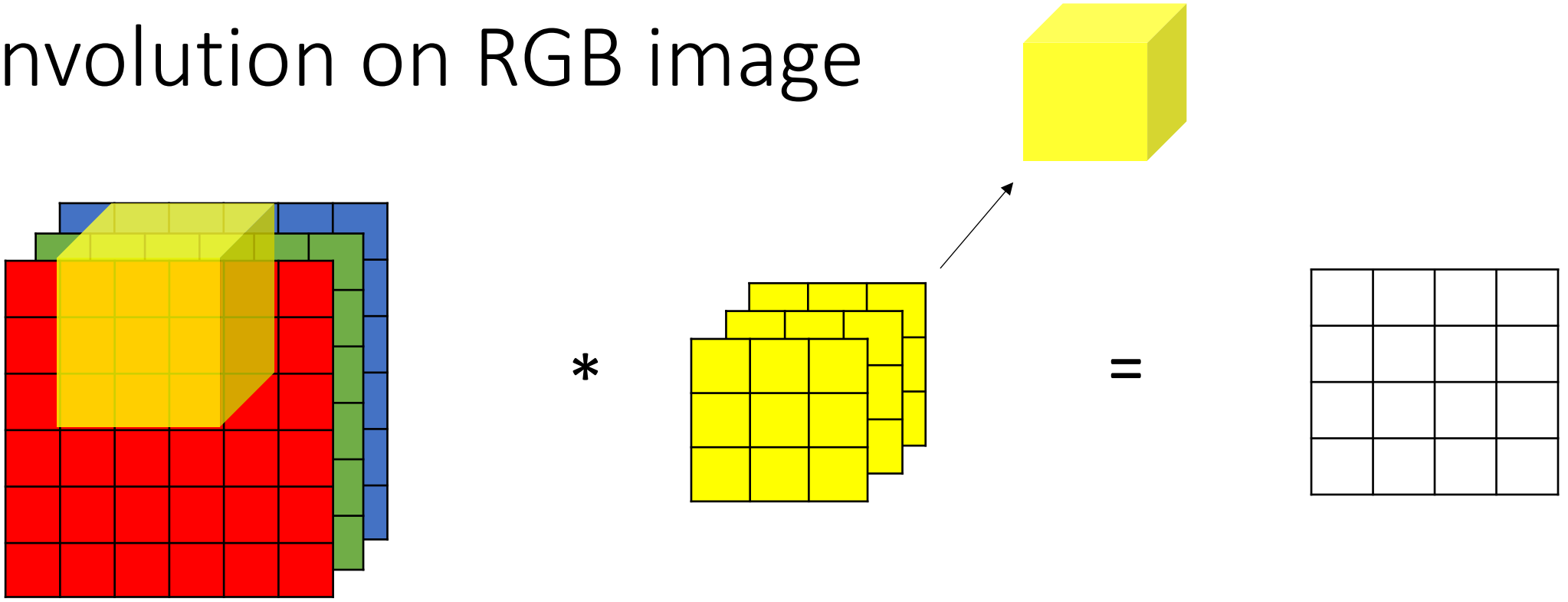
Convolution on RGB image



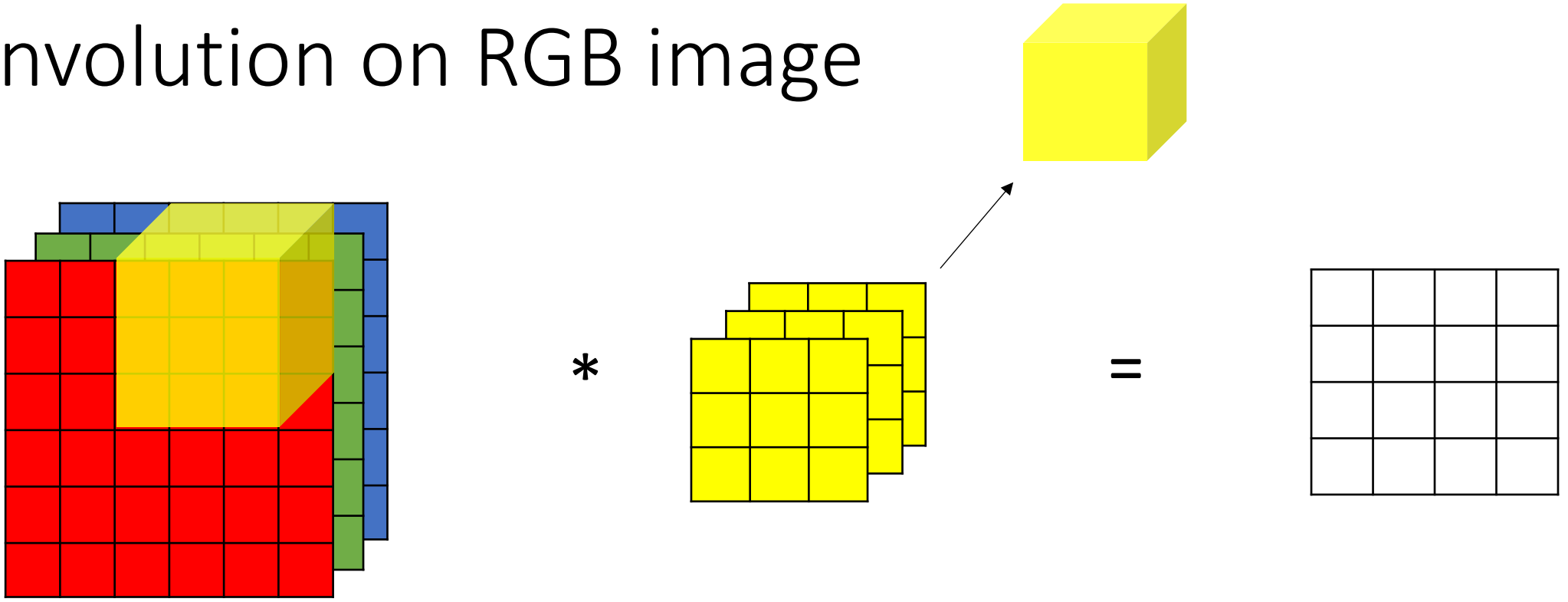
Convolution on RGB image



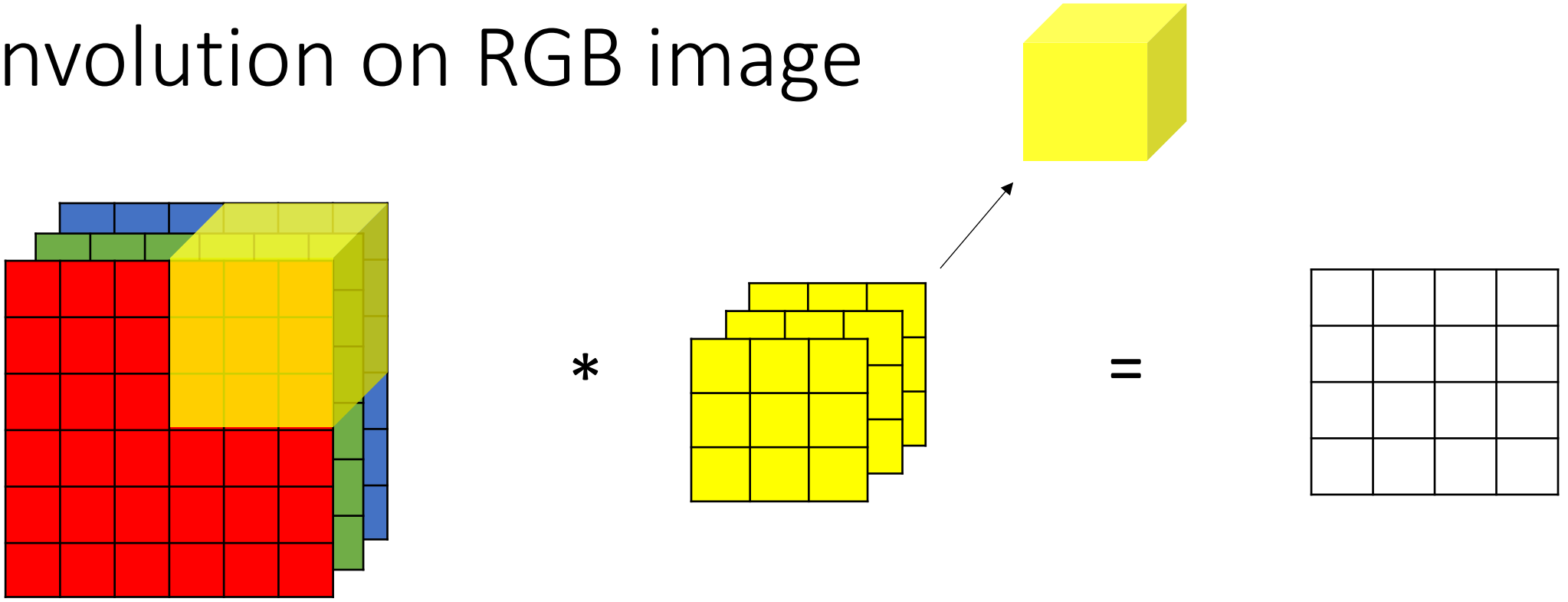
Convolution on RGB image



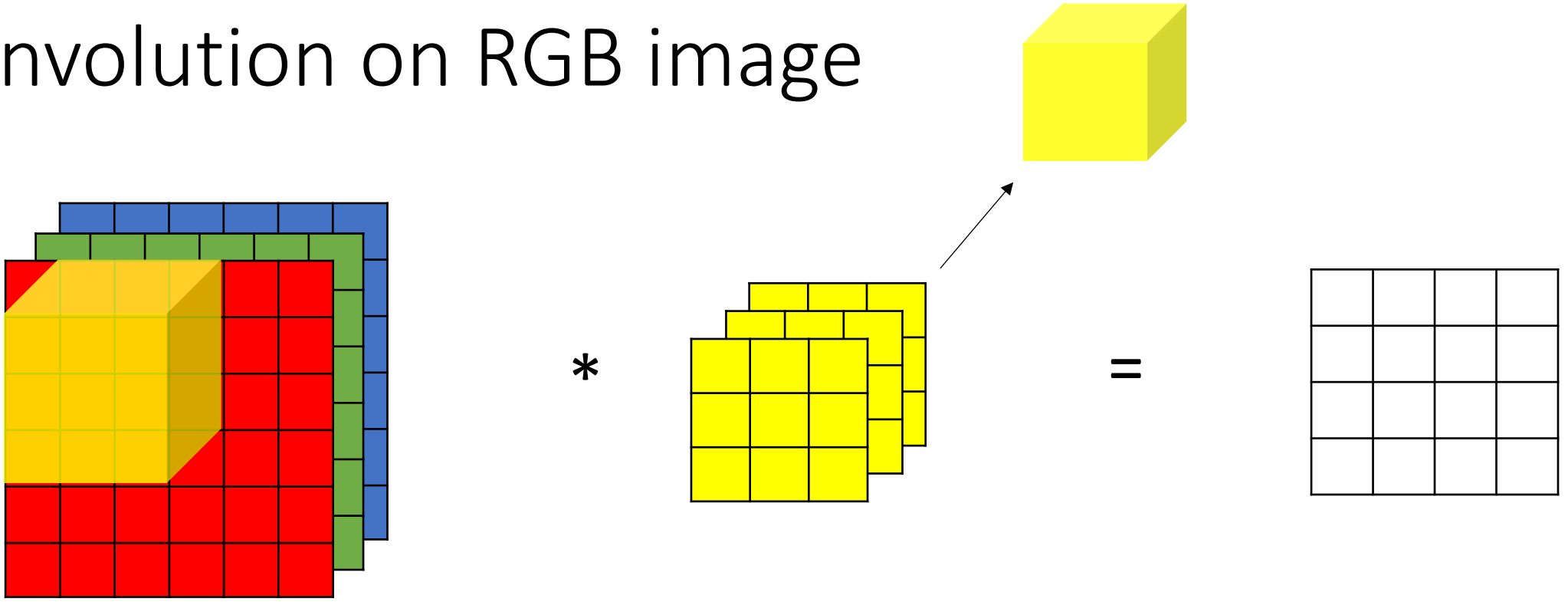
Convolution on RGB image



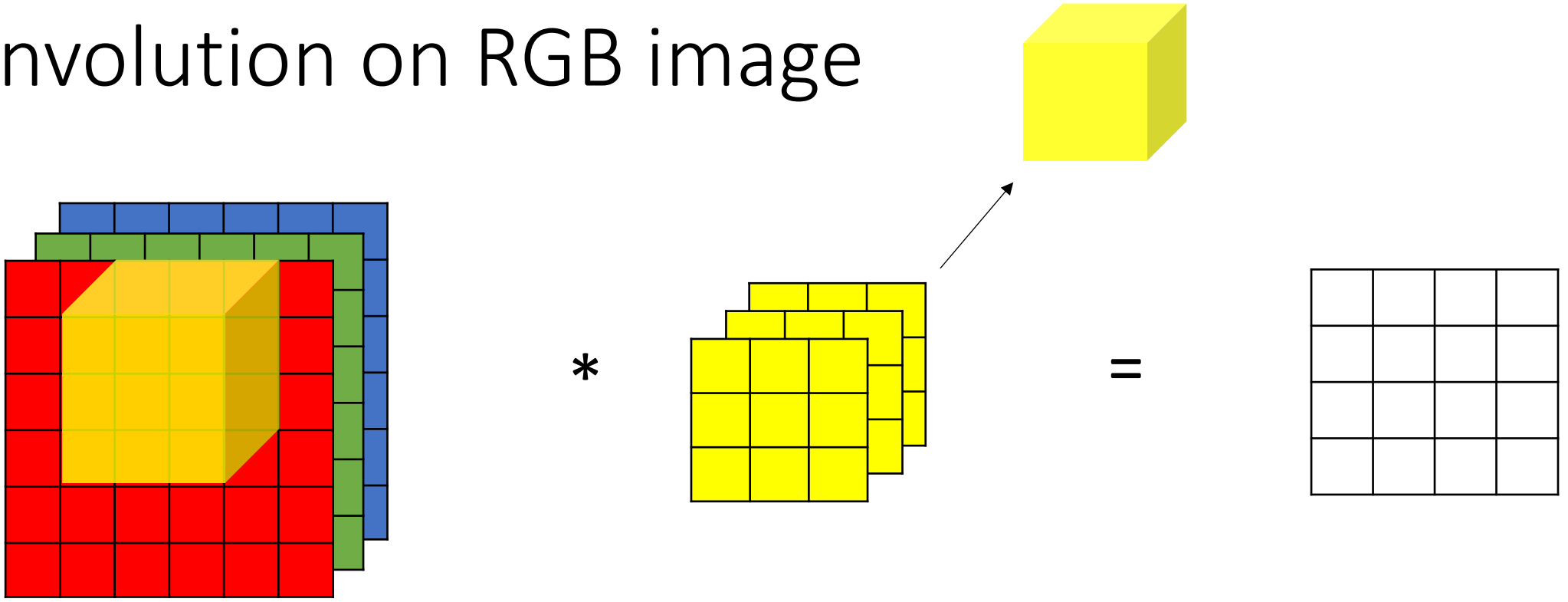
Convolution on RGB image



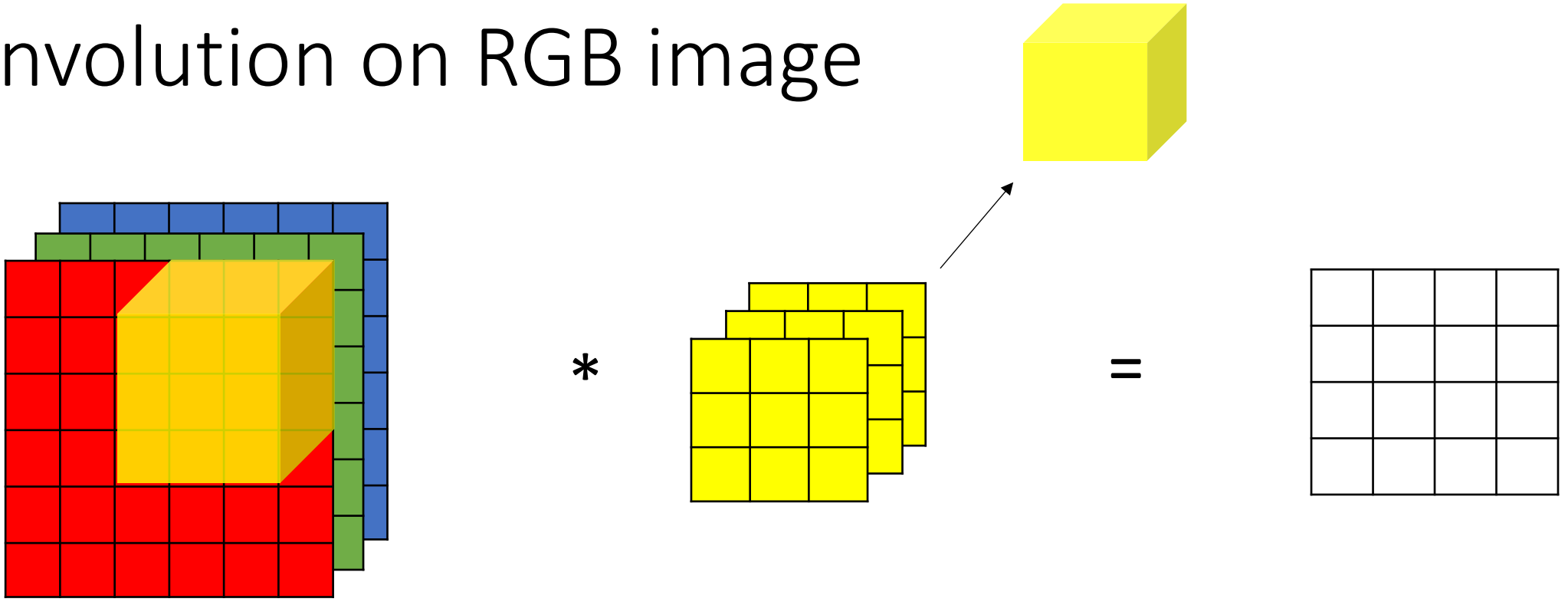
Convolution on RGB image



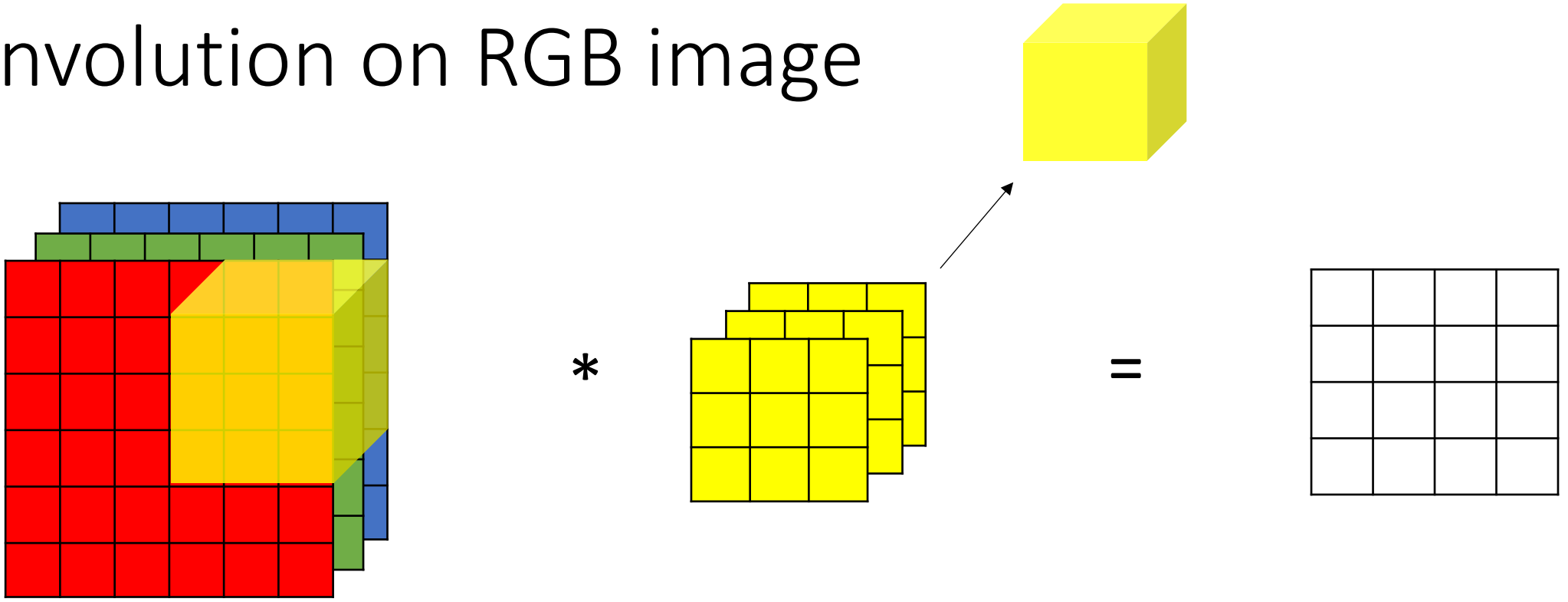
Convolution on RGB image



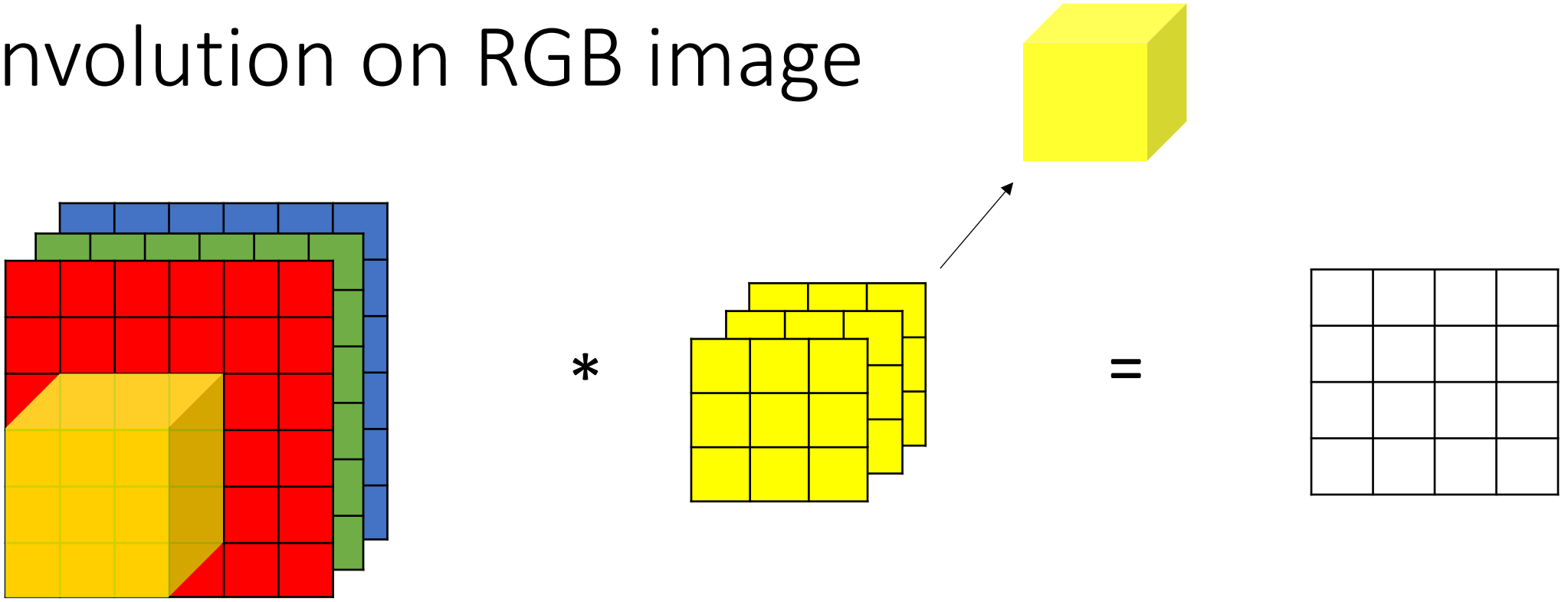
Convolution on RGB image



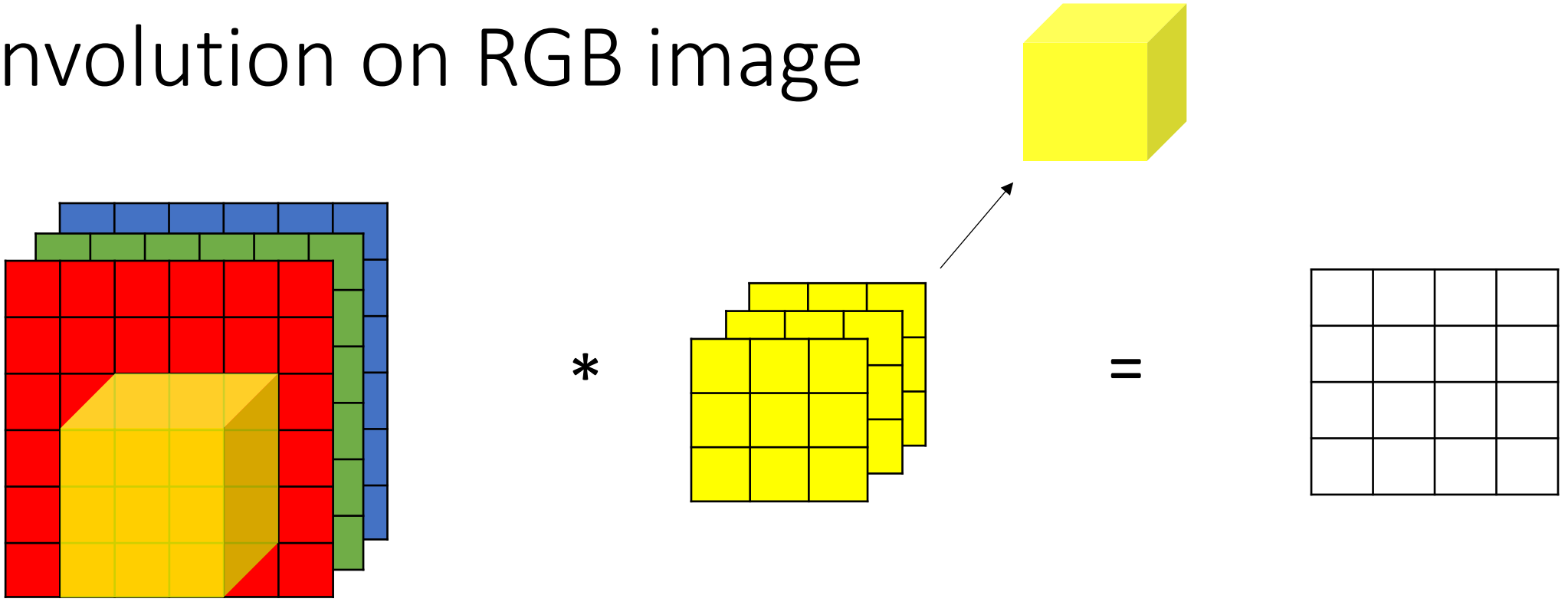
Convolution on RGB image



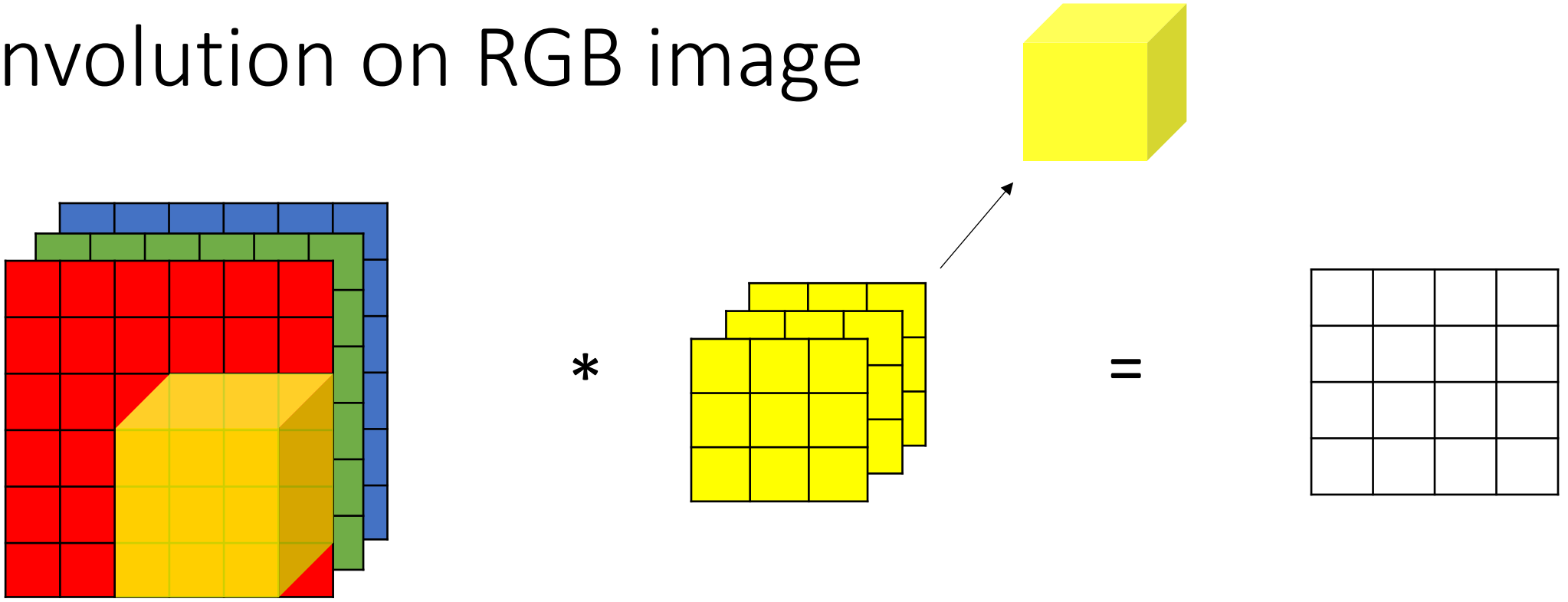
Convolution on RGB image



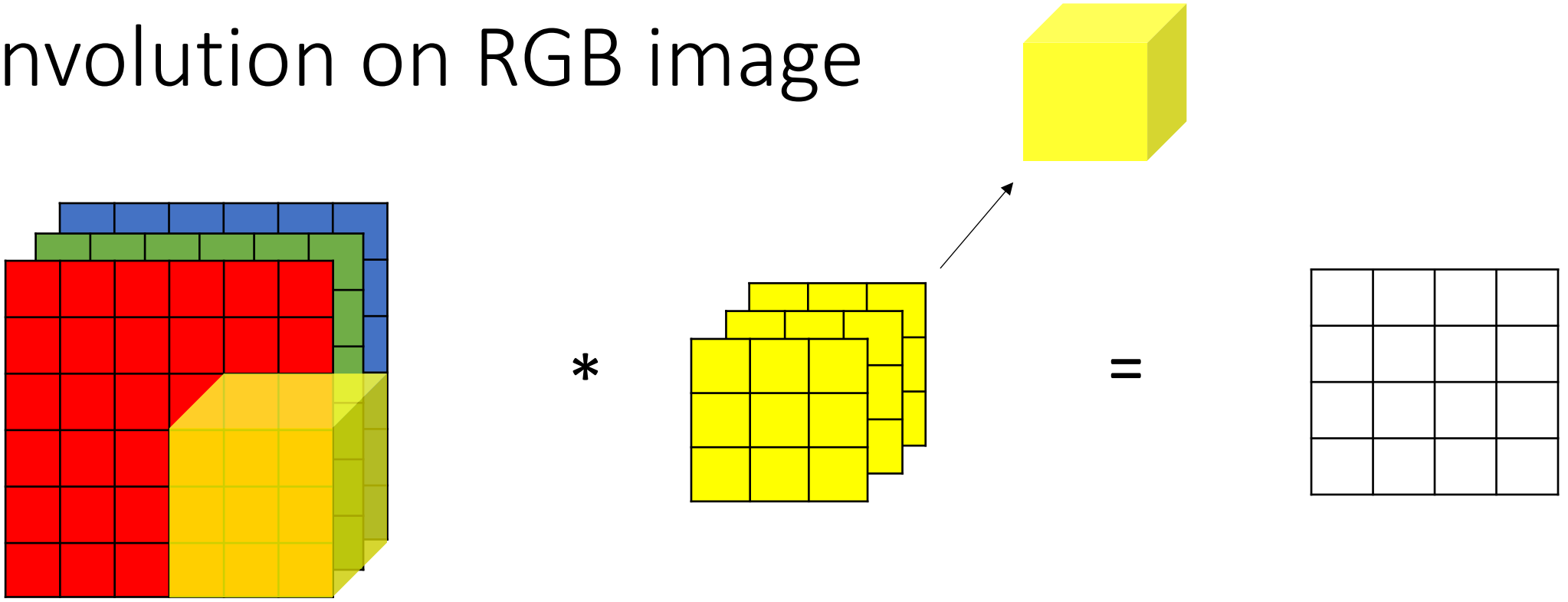
Convolution on RGB image



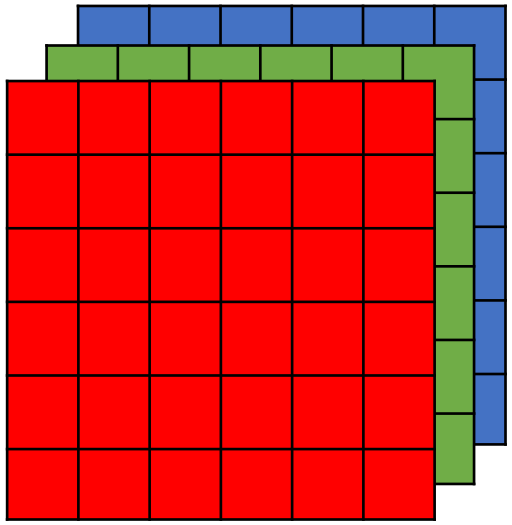
Convolution on RGB image

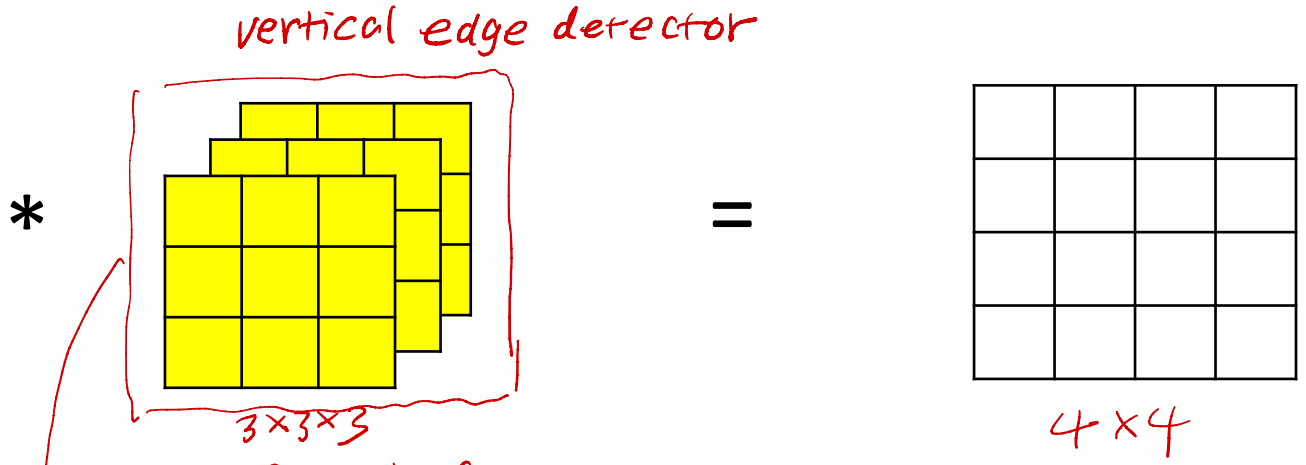


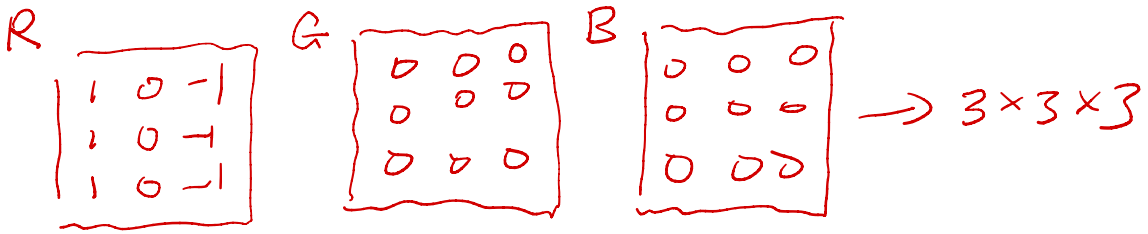
Convolution on RGB image

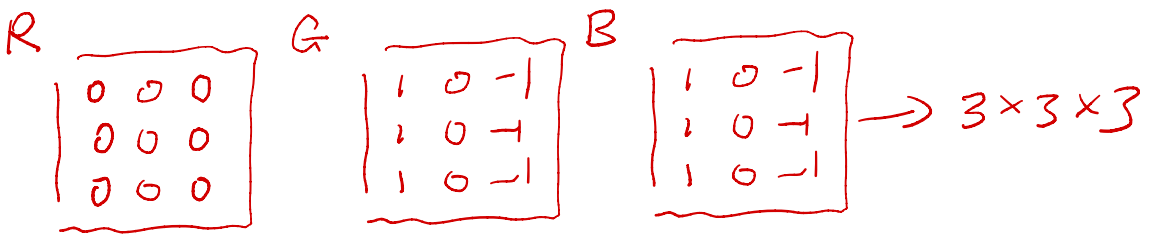


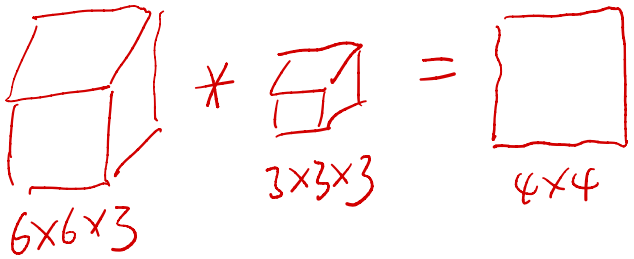
Multiple filters



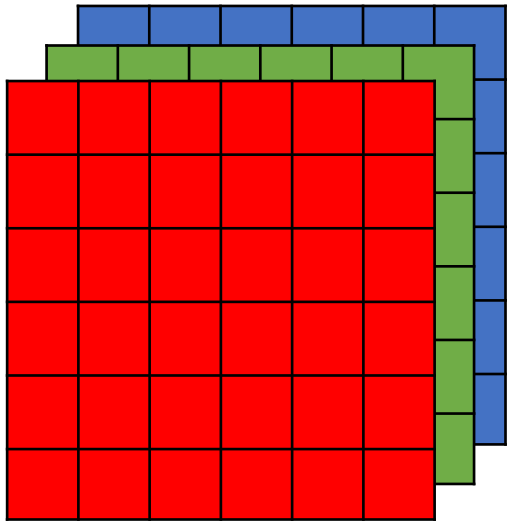








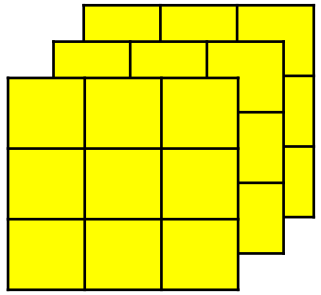
Multiple filters



6 x 6 x 3

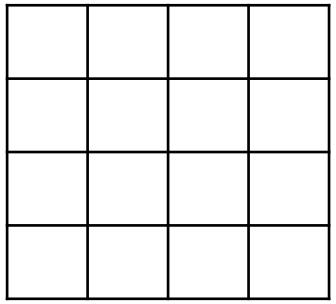
*

horizontal edge detector



3 x 3 x 3

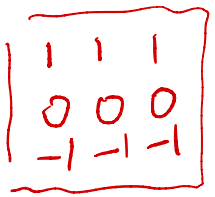
=



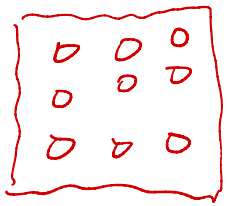
4 x 4 x 1

channel

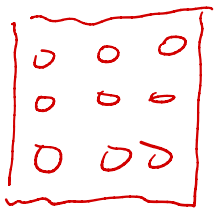
R



G

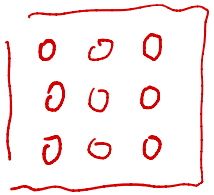


B

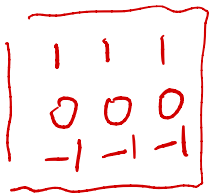


→ 3×3×3

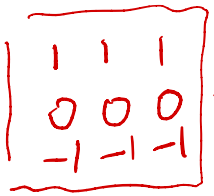
R



G

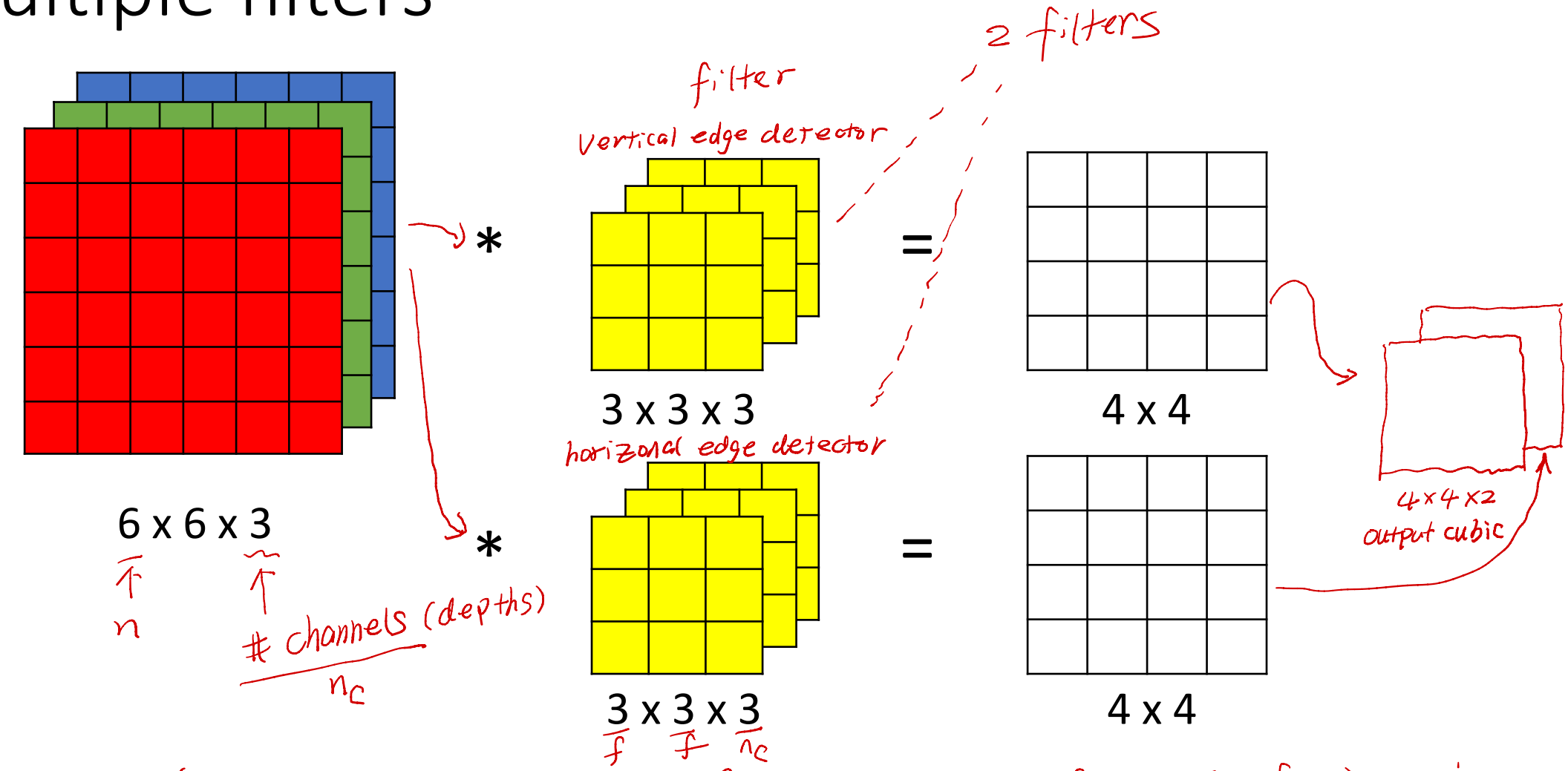


B



→ 3×3×3

Multiple filters



summary of dimensions :

$n \times n \times n_c$ * $f \times f \times n_c$ $\longrightarrow$ $(n-f+1) \times (n-f+1) \times n_c'$

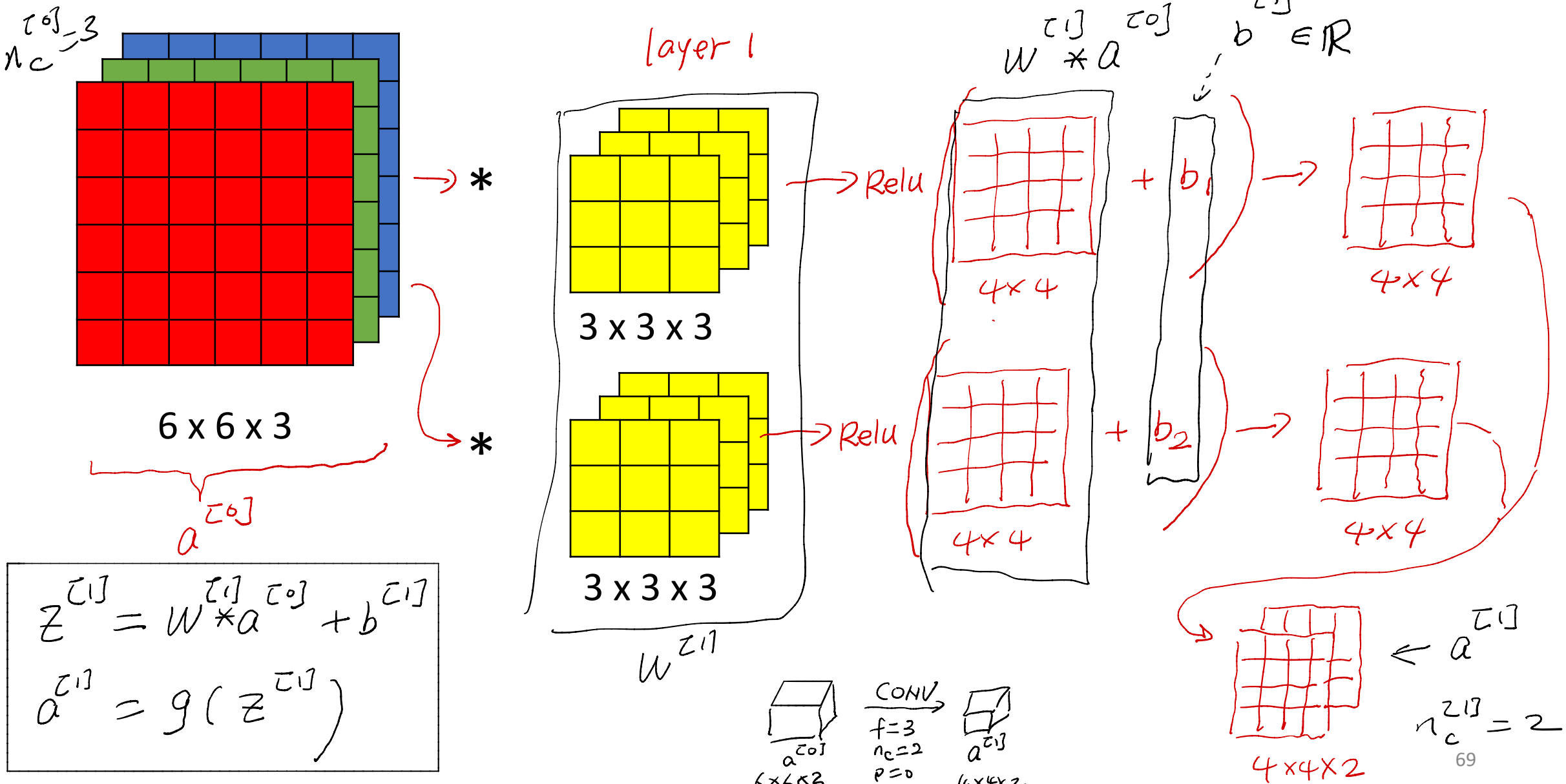
6 × 6 × 3 * 3 × 3 × 3 $\longrightarrow$ 4 × 4 × 2

n_c' = # filters

$3\times3\times3\times2+2 = 9\cdot6+2 = 54+2 = 56$ # parameters

Example of a convolution layer

Now I am ready to introduce a full picture of a convolution layer.

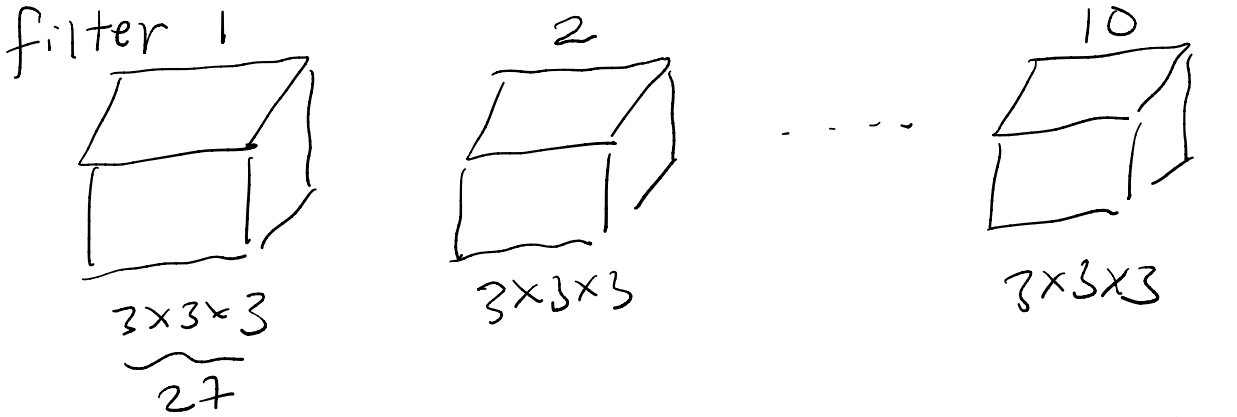


$$z^{[1]} = W^{[1]} * a^{[0]} + b^{[1]}$$

$$a^{[1]} = g(z^{[1]})$$

Number of parameters in one layer

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



$27 \times 10 + \underbrace{10}_{\text{bias}} = 270 + 10 = 280$

Summary of notation

If layer ℓ is a convolution layer:

INPUT: height, width, Number of channels

$$n_H^{[\ell-1]} \times n_W^{[\ell-1]} \times n_C^{[\ell-1]}$$

Hyper Parameters:

$f^{[\ell]}$ = filter size

$p^{[\ell]}$ = padding

$s^{[\ell]}$ = stride

$n_c^{[\ell]}$ = number of filters

OUTPUT : $n_H^{[\ell]} = \left\lfloor \frac{n_H^{[\ell-1]} + 2p^{[\ell]} - f^{[\ell]}}{s^{[\ell]}} + 1 \right\rfloor$

$n_W^{[\ell]} = \left\lfloor \frac{n_W^{[\ell-1]} + 2p^{[\ell]} - f^{[\ell]}}{s^{[\ell]}} + 1 \right\rfloor$

$n_c^{[\ell]}$ = Number of filters

channels of the output volume

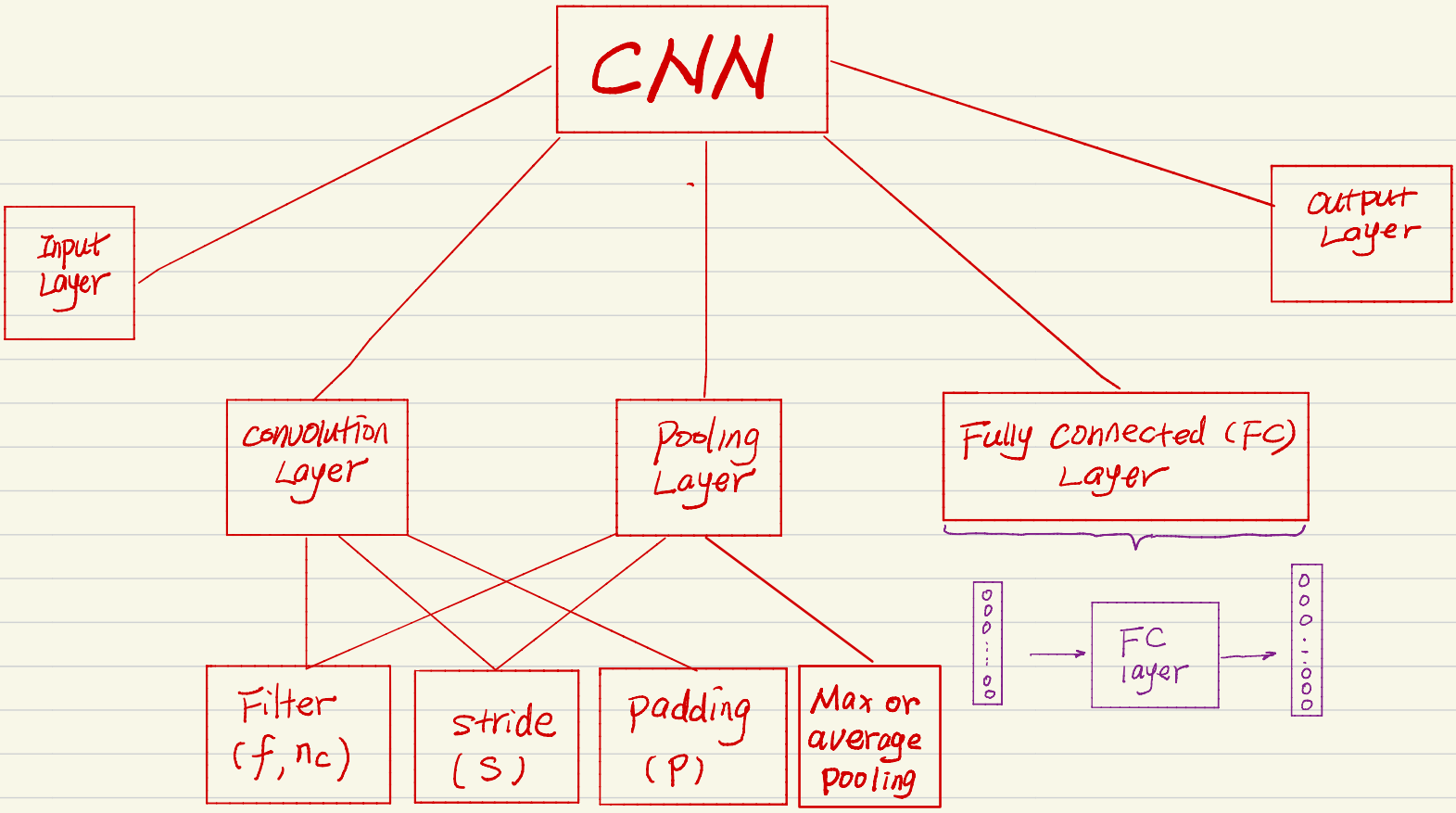
Each filter is : $f^{[\ell]} \times f^{[\ell]} \times n_c^{[\ell-1]}$

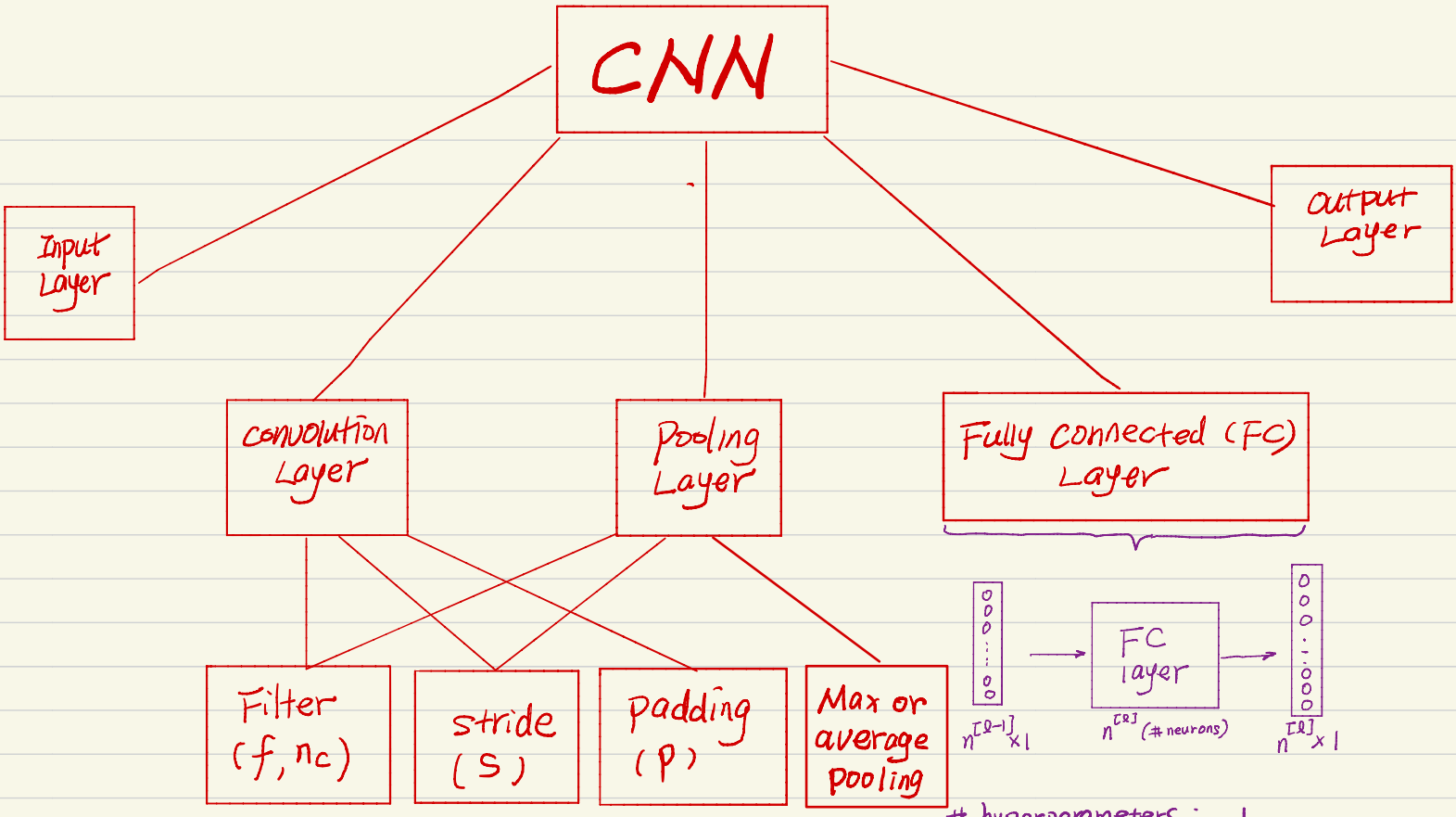
Activations (output) : $n_H^{[\ell]} \times n_W^{[\ell]} \times n_c^{[\ell]}$

weights : $f^{[\ell]} \times f^{[\ell]} \times n_c^{[\ell-1]} \times n_c^{[\ell]}$

$n_c^{[\ell]}$ ← # filters in layer ℓ.

bios : $n_c^{[\ell]}$


CNN
Input Layer
output Layer
convolution Layer
Pooling Layer
Fully connected (FC) Layer
FC layer
Filter (f, nc)
stride (S)
Padding (P)
Max or average pooling



hyperparameters : 1

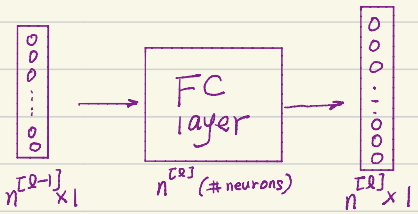
$n^{[l]}$ # neurons in layer l.

parameters : $n^{[l]} \times n^{[l-1]} + n^{[l]}$

$W^{[l]}$ $b^{[l]}$

CNN

- Input Layer
- convolution Layer
  - Filter (f, n_c)
  - stride (s)
  - padding (p)
- Pooling Layer
  - Max or average pooling
- Fully connected (FC) Layer

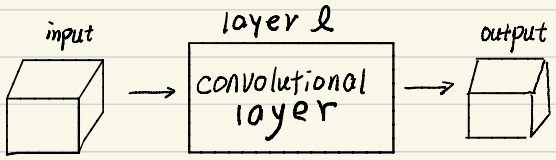


hyperparameters : 1

$n^{[l]}$ = # neurons in layer l.

parameters : $n^{[l]} \times n^{[l-1]} + n^{[l]}$

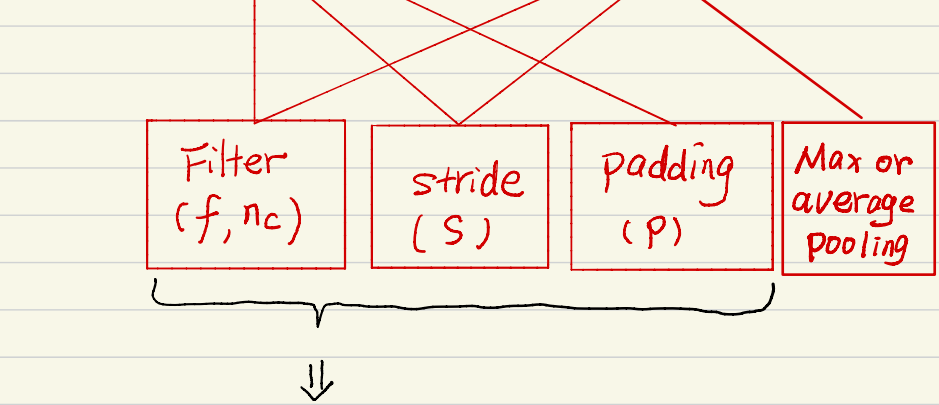
$W^{[l]}$ $b^{[l]}$



CNN

- Input Layer
- output Layer
- convolution Layer
- Pooling Layer
- Fully connected (FC) Layer

Filter (f, n_c) | stride (s) | padding (p) | Max or average pooling

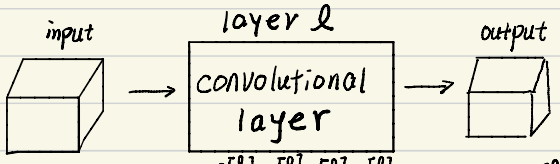


hyperparameters : 1

$n^{[l]}$: # neurons in layer l.

parameters : $n^{[l]} \times n^{[l-1]} + n^{[l]}$

$W^{[l]}$ $b^{[l]}$



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

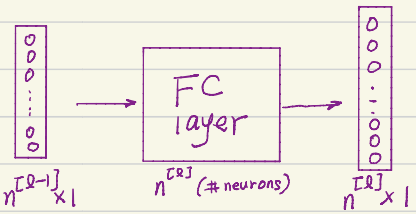
$f^{[l]}, n_c^{[l]}, s^{[l]}, p^{[l]}$

$$\left\lfloor \frac{n_H^{[l-1]} + 2 \cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times \left\lfloor \frac{n_W^{[l-1]} + 2 \cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times n_c^{[l]}$$

CNN

- Input Layer
- output Layer
- convolution Layer
- Pooling Layer
- Fully connected (FC) Layer

Filter (f, n_c) | stride (s) | padding (p) | Max or average pooling

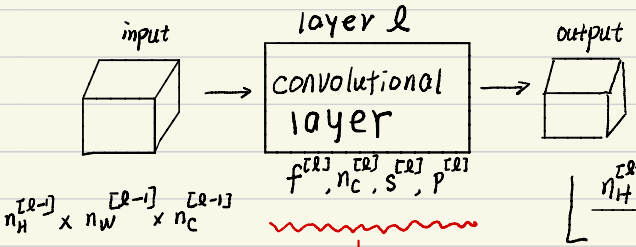


hyperparameters : 1

neurons in layer l.

parameters : $n^{[l]} \times n^{[l-1]} + n^{[l]}$

$W^{[l]}$ $b^{[l]}$



$$\left\lfloor \frac{n_H^{[l-1]} + 2 \cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times \left\lfloor \frac{n_W^{[l-1]} + 2 \cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times n_c^{[l]}$$

layers

parameters : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]} + n_c$ (weights) (bias)

hyperparameters : $1 + 1 + 1 + 1 = 4$

Convolutional layer

Pooling layer: Max pooling

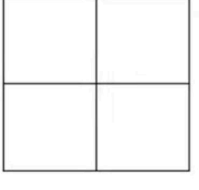
1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

4×4

Pooling layer: Max pooling

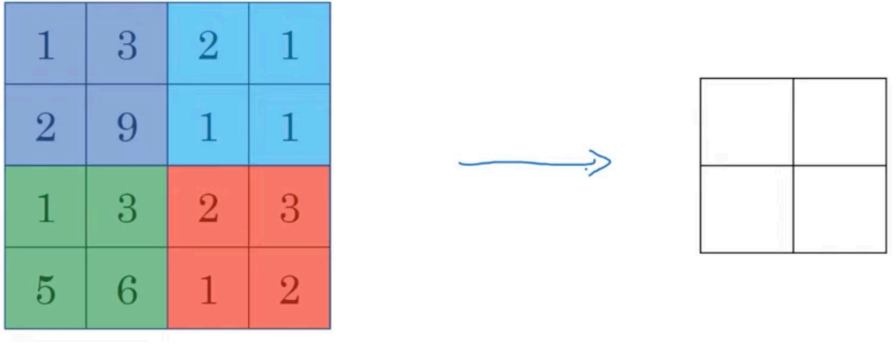
1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

4×4



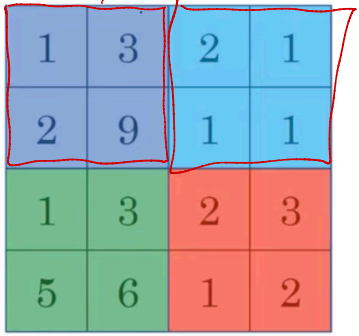
output of this specific pooling is 2×2 matrix

Pooling layer: Max pooling



Pooling layer: Max pooling

some set of features (output of edge detection)

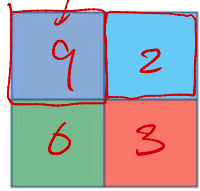


4×4

hyperparameters

f = 2

s = 2



2×2

intuition:

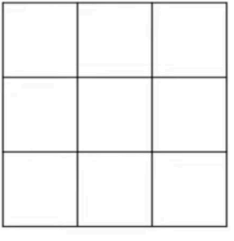
large number indicates the largest signal with the region

output each of the output will be max of the corresponding shaded region

Pooling layer: Max pooling

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

5×5



3 × 3

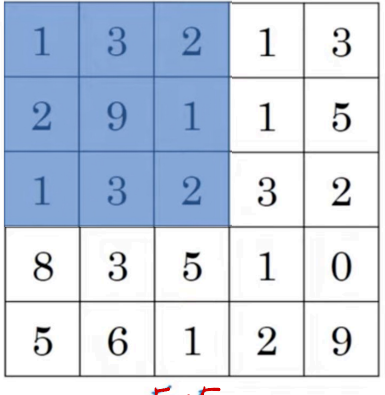
f = 3
S = 1

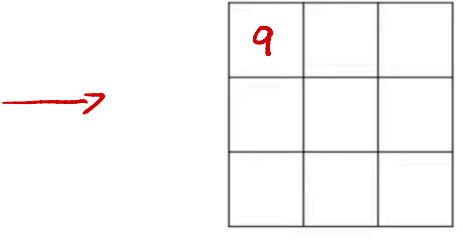
$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$

$= \left\lfloor \frac{5+0-3}{1} + 1 \right\rfloor = 3$

an example with different hyperparameters

Pooling layer: Max pooling





f = 3
s = 1

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

$= \lfloor \frac{5+0-3}{1} + 1 \rfloor = 3$

an example with different hyperparameters

Pooling layer: Max pooling

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

5×5

9	9	

$f = 3$
$s = 1$

3 × 3

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

$= \left\lfloor \frac{5+0-3}{1} + 1 \right\rfloor = 3$

Pooling layer: Max pooling

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

5×5

9	9	5

3 × 3

f = 3
s = 1

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

$= \lfloor \frac{5+0-3}{1} + 1 \rfloor = 3$

Pooling layer: Max pooling

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

5×5

9	9	5
9		

3 × 3

$f = 3$
$s = 1$

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

$= \left\lfloor \frac{5+0-3}{1} + 1 \right\rfloor = 3$

Pooling layer: Max pooling

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

5×5

9	9	5
9	9	

3 × 3

f = 3
s = 1

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

$= \lfloor \frac{5+0-3}{1} + 1 \rfloor = 3$

Pooling layer: Max pooling

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

5 × 5

9	9	5
9	9	5

3 × 3

$f = 3$
$s = 1$

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

$= \left\lfloor \frac{5+0-3}{1} + 1 \right\rfloor = 3$

Pooling layer: Max pooling

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

5×5

9	9	5
9	9	5
8		

3 × 3

$f = 3$
$s = 1$

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

$= \left\lfloor \frac{5+0-3}{1} + 1 \right\rfloor = 3$

Pooling layer: Max pooling

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

5×5

9	9	5
9	9	5
8	6	

$f = 3$
$s = 1$

3 × 3

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

$= \left\lfloor \frac{5+0-3}{1} + 1 \right\rfloor = 3$

Pooling layer: Max pooling

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

5×5

9	9	5
9	9	5
8	6	9

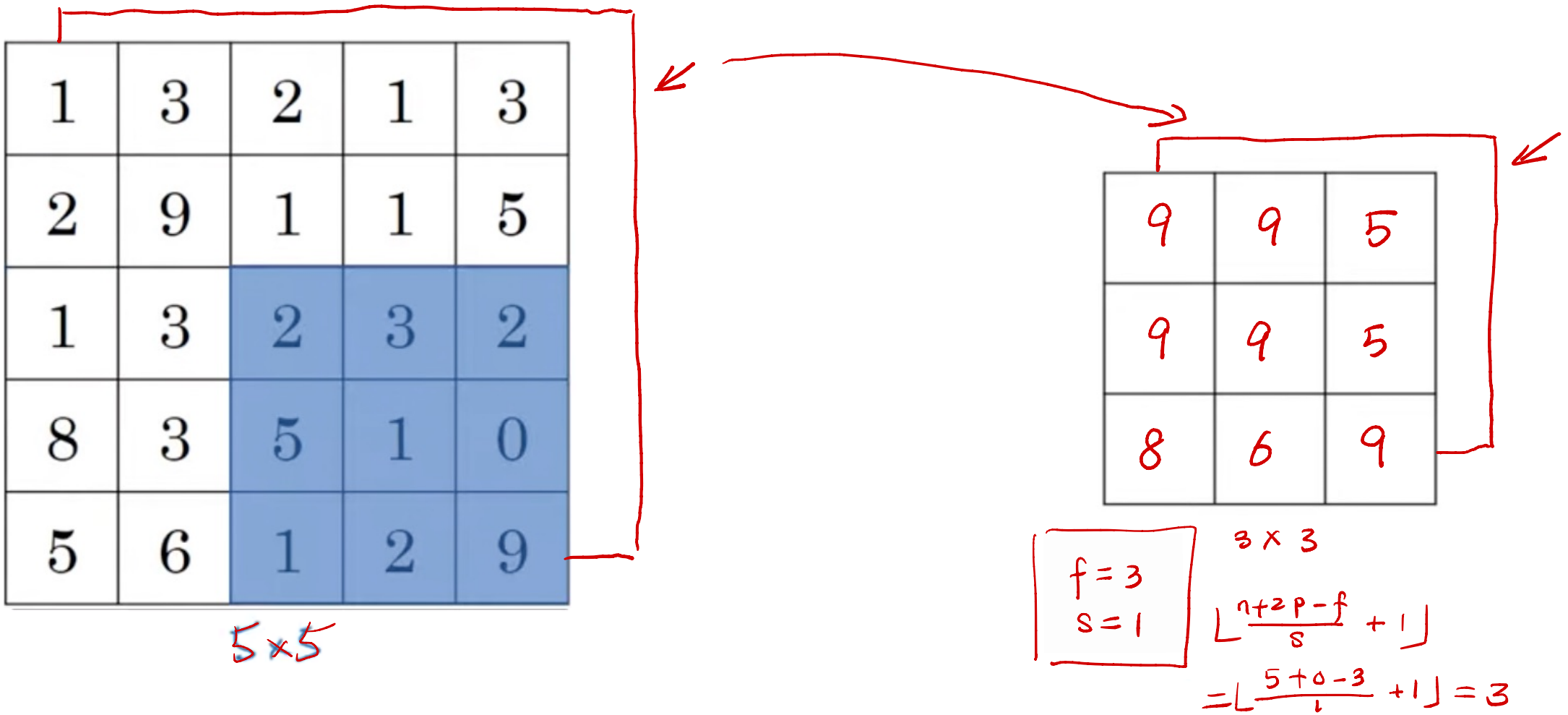
3 × 3

$f = 3$
$s = 1$

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

$= \left\lfloor \frac{5+0-3}{1} + 1 \right\rfloor = 3$

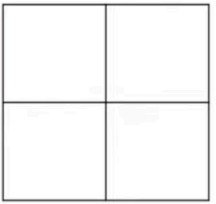
Pooling layer: Max pooling



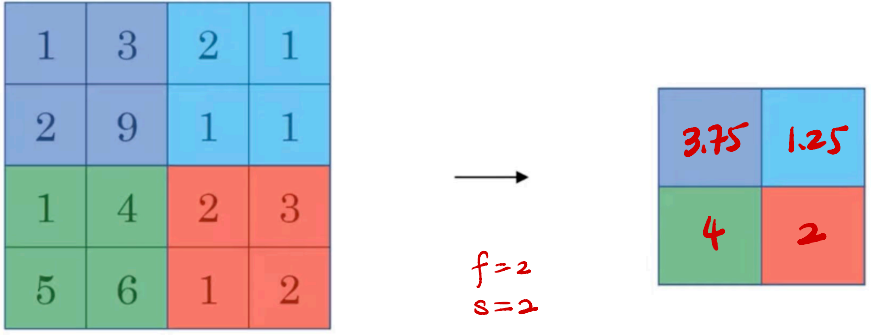
Pooling layer: Average pooling

1	3	2	1
2	9	1	1
1	4	2	3
5	6	1	2

f = 2
s = 2



Pooling layer: Average pooling



Summary of pooling

Hyperparameters:

f : filter size

s : stride

Max or average pooling

Popular settings

$f=2, s=2$

$f=3, s=2$

(typically, no padding)

NO parameters to learn.

INPUT: $n_H \times n_W \times n_C$

OUTPUT:

$$\left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times n_C$$

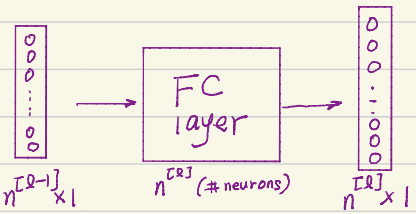
equal

Pooling is applied to each channel seperately.

CNN

- Input Layer
- output Layer
- convolution Layer
- Pooling Layer
- Fully connected (FC) Layer

Filter (f, n_c) | stride (s) | padding (p) | Max or average pooling



hyperparameters : 1

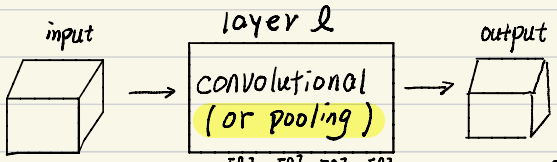
$n^{[l]}$ # neurons in layer l.

parameters : $n^{[l]} \times n^{[l-1]} + n^{[l]}$

$W^{[l]}$ $b^{[l]}$

same input/output size

$p = \left\lfloor \frac{f-1}{2} \right\rfloor$



$$\left\lfloor \frac{n_H^{[l-1]} + 2 \cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times \left\lfloor \frac{n_W^{[l-1]} + 2 \cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times n_c^{[l]}$$

layers

parameters : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]} + n_c$

($f \times f \times n_c \times n_c$: weights; n_c: bias)

hyperparameters : $1+1+1+1 = 4$

Convolutional layer

parameters : 0

hyperparameters : $1+1+1 = 3$

Pooling layer

CNN

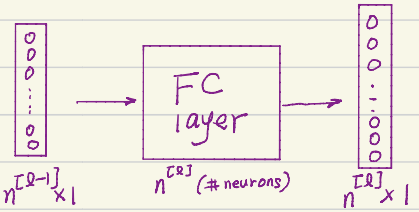
Input Layer

Output Layer

convolution Layer

Pooling Layer

Fully connected (FC) Layer



Filter (f, n_c)

stride (s)

padding (p)

Max or average pooling

hyperparameters : 1

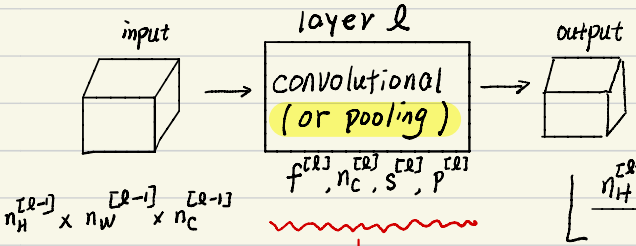
same input/output size

$p = \lfloor \frac{f-1}{2} \rfloor$

$n^{[l]}$: # neurons in layer l.

parameters : $n^{[l]} \times n^{[l-1]} + n^{[l]}$

$W^{[l]}$ $b^{[l]}$



$$\left\lfloor \frac{n_H^{[l-1]} + 2 \cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times \left\lfloor \frac{n_W^{[l-1]} + 2 \cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times n_c^{[l]}$$

layers

parameters : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]} + n_c^{[l]}$ (weights) (bias)

hyperparameters : 1 + 1 + 1 + 1 = 4

Convolutional layer

parameters : 0

hyperparameters : 1 + 1 + 1 = 3

Pooling layer

Special Note for Pooling Layer

① $p^{[l]}$ is always set to 0, because it is rare to do padding for pooling

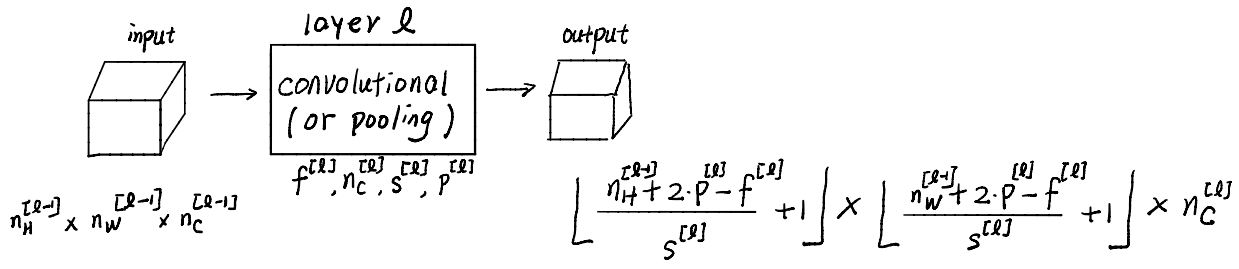
② $n_c^{[l]} = n_c^{[l-1]}$ that means # channels of the output is always the same as # channels of the input.

③ It has three hyperparameters, including filter size (f), stride size (s), and the aggregate function ("max" or "average")

Example CNN network - 1

INPUT (39x39x3)
CONV1 (f=3, $n_C = 10$, s=1, p=0)
CONV2 (f=5, $n_C = 20$, s=2, p=0)
CONV3 (f=5, $n_C = 40$, s=2, p=0)
softmax (10,1)

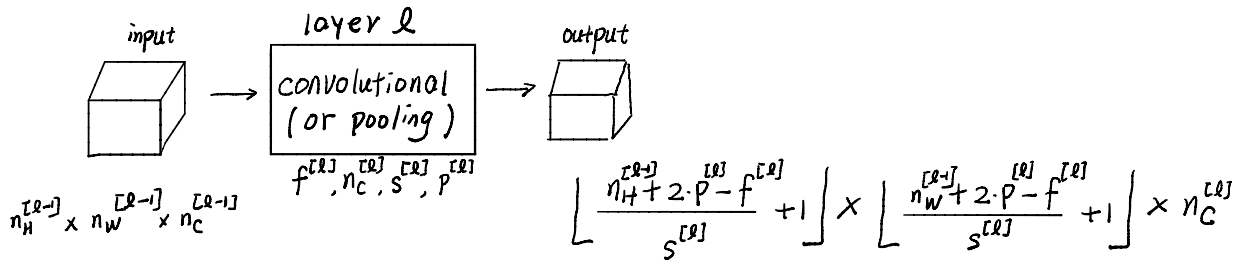
Example CNN network - 1

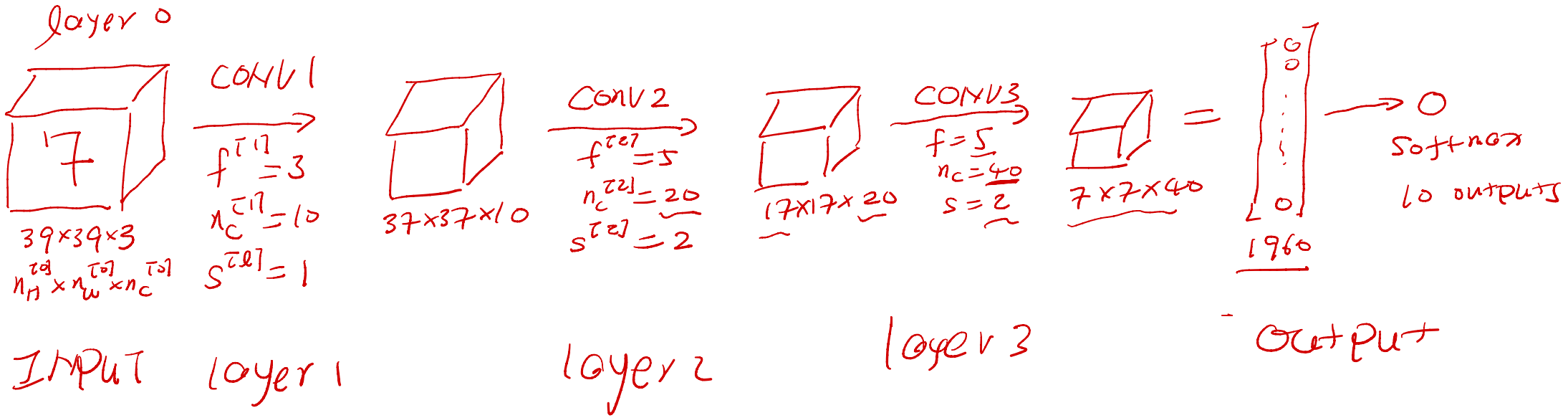


INPUT (39x39x3)
CONV1 (f=3, $n_C = 10$, s=1, p=0)
CONV2 (f=5, $n_C = 20$, s=2, p=0)
CONV3 (f=5, $n_C = 40$, s=2, p=0)
softmax (10,1)

Example CNN network - 1

INPUT (39x39x3)
CONV1 (f=3, $n_C = 10$, s=1, p=0)
CONV2 (f=5, $n_C = 20$, s=2, p=0)
CONV3 (f=5, $n_C = 40$, s=2, p=0)
softmax (10,1)



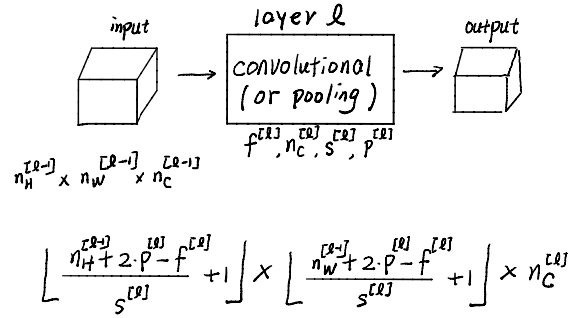


Example CNN network - 2 (LeNet)

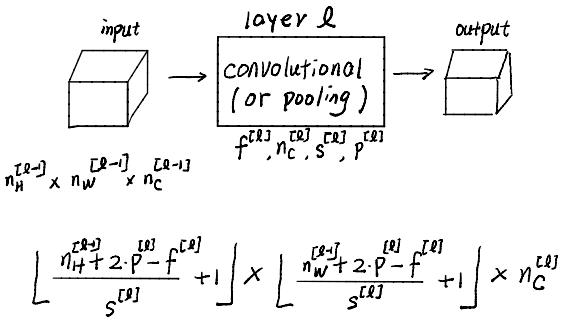
By defaut, $p = 0$.

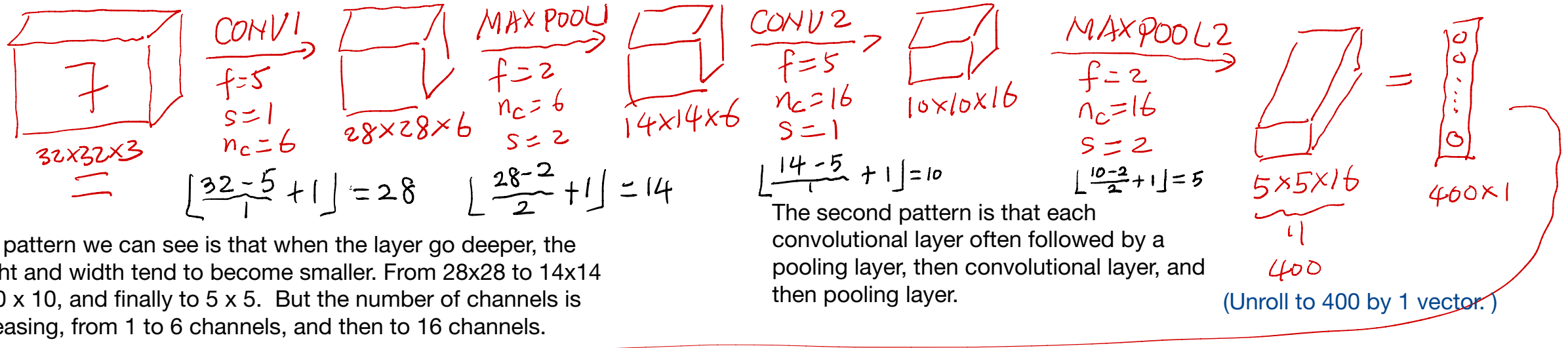
INPUT (32 x 32 x 3)
CONV1 (f = 5, n_c = 6, s = 1)
MAX-POOL1 (f = 2, n_c =6, s = 2)
CONV2 (f=5, n_c =16, s = 1)
MAX-POOL (f = 2, n_c =16, s = 2)
FC3 (120, 1)
FC4 (84, 1)
Softmax(10)

Example CNN network - 2 (LeNet)



Example CNN network - 2 (LeNet)

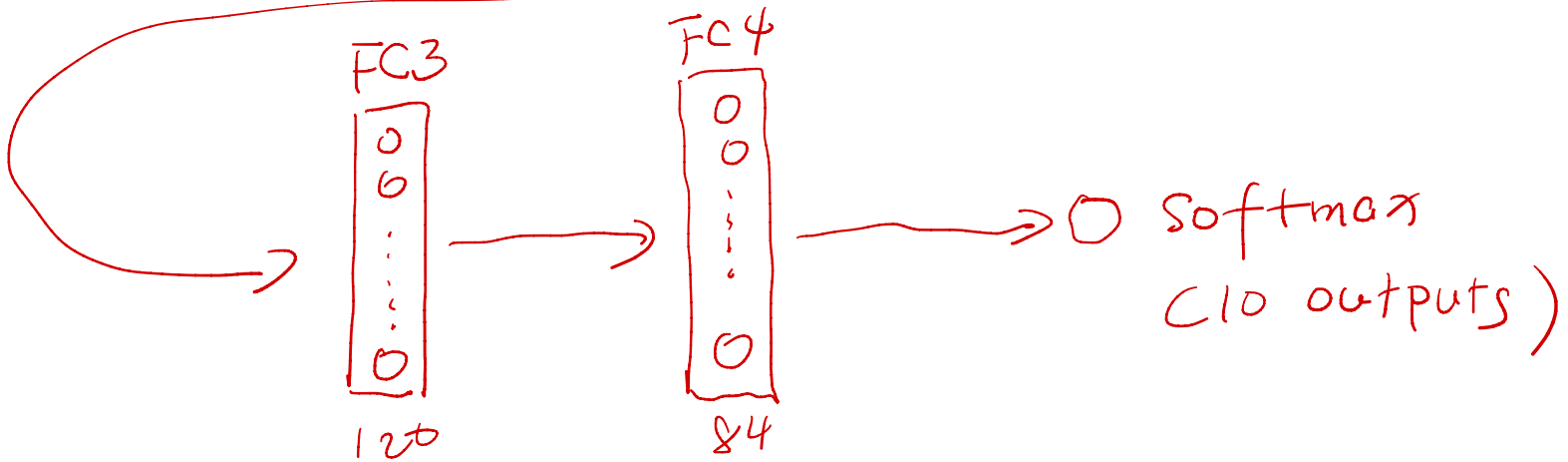




One pattern we can see is that when the layer go deeper, the height and width tend to become smaller. From 28x28 to 14x14 to 10 x 10, and finally to 5 x 5. But the number of channels is increasing, from 1 to 6 channels, and then to 16 channels.

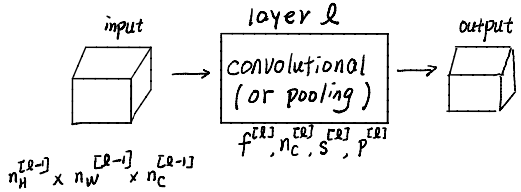
The second pattern is that each convolutional layer often followed by a pooling layer, then convolutional layer, and then pooling layer.

(Unroll to 400 by 1 vector.)



The last pattern is that the last several layers are always fully connected layers with number of neurons decreasing from 120, to 84, and finally 10 outputs for ten class classification in this example network.

Example CNN network – 3 (AlexNet)



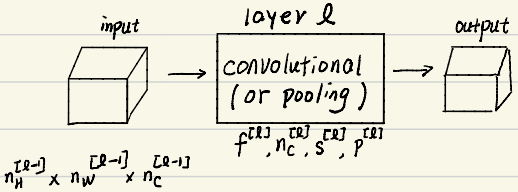
$$\left\lfloor \frac{n_H^{[l-1]} + 2 \cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times \left\lfloor \frac{n_W^{[l-1]} + 2 \cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times n_c^{[l]}$$

$$p = \left\lfloor \frac{f-1}{2} \right\rfloor$$

INPUT (227x227x3)
CONV1 (f=11, $n_c = 96$, s=4)
MAX-POOL1 (f=3, s=2)
CONV2 (f=5, $n_c = 256$, s=1), same
MAX-POOL2 (f=3, s=2)
CONV3 (f=3, $n_c = 384$, s=1), same
CONV4 (f=3, $n_c = 384$, s=1), same
CONV5 (f=3, $n_c = 256$, s=1), same
MAX-POOL3 (f=3, s=2)
FC1 (4096)
FC2 (4096)
Softmax (1000)

"same" means that the input and output have the same width and height.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "*Imagenet classification with deep convolutional neural networks.*" *Advances in neural information processing systems*. 2012. (70,500 citations)

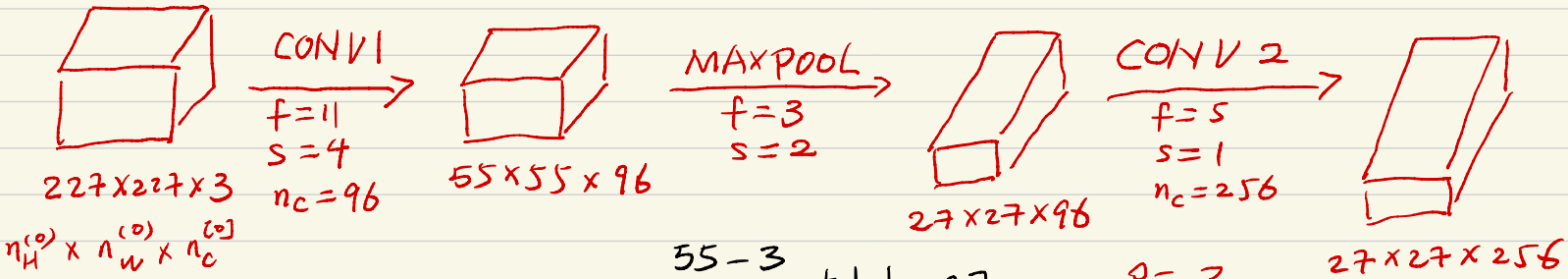


$$\left\lfloor \frac{n_H^{[l-1]} + 2 \cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times \left\lfloor \frac{n_W^{[l-1]} + 2 \cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times n_C^{[l]}$$

$$p = \left\lfloor \frac{f-1}{2} \right\rfloor$$

INPUT (227x227x3)
CONV1 (f=11, $n_c = 96$, s=4)
MAX-POOL1 (f=3, s=2)
CONV2 (f=5, $n_c = 256$, s=1), same
MAX-POOL2 (f=3, s=2)
CONV3 (f=3, $n_c = 384$, s=1), same
CONV4 (f=3, $n_c = 384$, s=1), same
CONV5 (f=3, $n_c = 256$, s=1), same
MAX-POOL3 (f=3, s=2)
FC1 (4096)
FC2 (4096)
Softmax (1000)

$p = \left\lfloor \frac{f-1}{2} \right\rfloor$



$$\left\lfloor \frac{227 + 2 \cdot 0 - 11}{4} + 1 \right\rfloor = 55$$

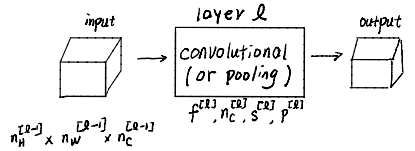
$$\left\lfloor \frac{55 - 3}{2} + 1 \right\rfloor = 27$$

$p = 2$

$$\left\lfloor \frac{27 + 2 \cdot 2 - 5}{1} + 1 \right\rfloor = 27$$

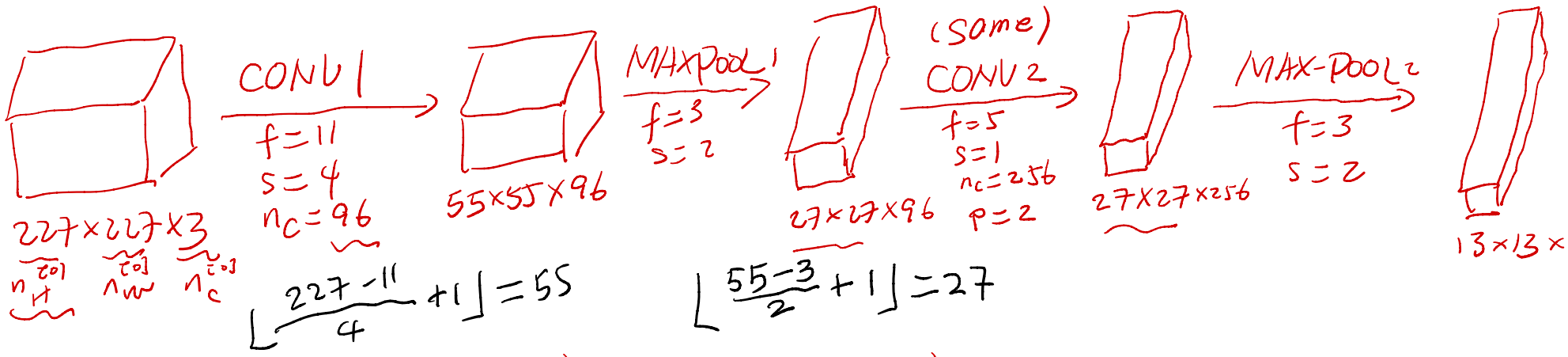
Alexnet

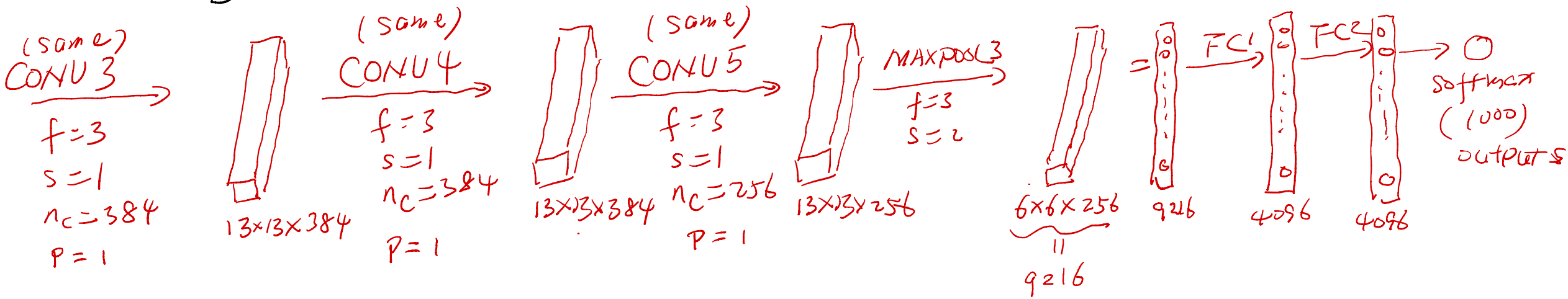
Example CNN network - 3



$$\left\lfloor \frac{n_H^{[l-1]} + 2\cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times \left\lfloor \frac{n_W^{[l-1]} + 2\cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times n_C^{[l]}$$

INPUT (227x227x3)
CONV1 (f=11, n_c = 96, s=4)
MAX-POOL1 (f=3, s=2)
CONV2 (f=5, n_c = 256, s=1), same
MAX-POOL2 (f=3, s=2)
CONV3 (f=3, n_c = 384, s=1), same
CONV4 (f=3, n_c = 384, s=1), same
CONV5 (f=3, n_c = 256, s=1), same
MAX-POOL3 (f=3, s=2)
FC1 (4096)
FC2 (4096)
Softmax (1000)





$$\left\lfloor \frac{f-1}{2} \right\rfloor$$

$$= \left\lfloor \frac{3-1}{2} \right\rfloor = 1$$

Example CNN network – 4 (VGG-16)

CONV = 3x3 filter, s = 1, same

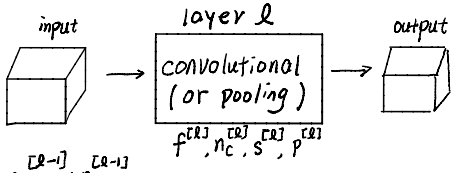
MAX-POOL = 2x2 filter, s = 2

Simonyan, Karen, and Andrew Zisserman. "*Very deep convolutional networks for large-scale image recognition*." *arXiv preprint arXiv:1409.1556* (2014). (44,780 citations).

INPUT (224x224x3)
CONV ($n_C = 64$) X 2
MAX-POOL
CONV ($n_C = 128$) X 2
MAX-POOL
CONV ($n_C = 256$) X 3
MAX-POOL
CONV ($n_C = 256$) X 3
MAX-POOL
CONV ($n_C = 256$) X 3
MAX-POOL
FC (4096)
FC (4096)
Softmax (1000,1)

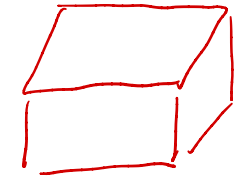
CONV = 3×3 filter, S = 1, MAXPOOL = 2×2, S = 2

Example CNN network - 4



$$\left\lfloor \frac{n_H^{[l-1]} + 2 \cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times \left\lfloor \frac{n_W^{[l-1]} + 2 \cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times n_C^{[l]}$$

INPUT (224x224x3)
CONV ($n_C = 64$) X 2
MAX-POOL
CONV ($n_C = 128$) X 2
MAX-POOL
CONV ($n_C = 256$) X 3
MAX-POOL
CONV ($n_C = 256$) X 3
MAX-POOL
CONV ($n_C = 256$) X 3
MAX-POOL
FC (4096)
FC (4096)
Softmax (1000,1)



224×224×3

CONV → 224×224×64 (CONV → 224×224×64) MAXPOOL → 112×112×64

$n_C = 64$, $p = \left\lfloor \frac{3-1}{2} \right\rfloor = 1$

$n_C = 64$, $p = 1$

$f = 2$, $s = 2$

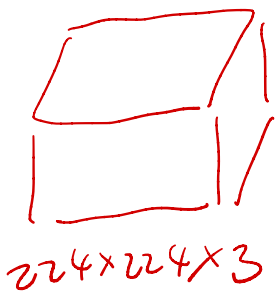
$$\left\lfloor \frac{224 - 2}{2} + 1 \right\rfloor = 112$$

MAXPOOL →

VGG-16 CONV = 3×3 filter, S = 1, same MAXPOOL = 2×2, S = 2

Example CNN network - 4

INPUT (224x224x3)
CONV (n_C = 64) X 2
MAX-POOL
CONV (n_C = 128) X 2
MAX-POOL
CONV (n_C = 256) X 3
MAX-POOL
CONV (n_C = 256) X 3
MAX-POOL
CONV (n_C = 256) X 3
MAX-POOL
FC (4096)
FC (4096)
Softmax (1000,1)



224×224×3

CONV, n_C = 64, ×2 → 224×224×64

MAXPOOL → 112×112×64

CONV, n_C = 128, ×2 → 112×112×128

MAXPOOL → 56×56×128

CONV, n_C = 256, ×3 → 56×56×256

MAXPOOL → 28×28×256

CONV, n_C = 256, ×3 → 28×28×256

MAXPOOL → 14×14×256

CONV, n_C = 256, ×3 → 14×14×256

MAXPOOL → 7×7×256 = 12544

FC → 4096

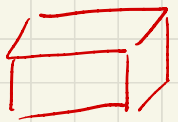
FC → 4096

Softmax (1000 outputs)

$\left\lfloor \frac{14-2}{2} + 1 \right\rfloor = 7$

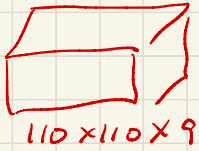
Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	~~208~~ 608
POOL1	(14,14,8)	1,568	0
CONV2 (f=5, s=1)	(10,10,16)	1,600	~~416~~ 3216
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	~~48,001~~ 48120
FC4	(84,1)	84	~~10,081~~ 10164
Softmax	(10,1)	10	~~841~~ 850



$224 \times 224 \times 3$

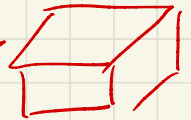
CONV1
$f=4$
$s=2$
$n_c=96$



$110 \times 110 \times 96$

$\left\lfloor \frac{224+0-4}{2} + 1 \right\rfloor = 111$

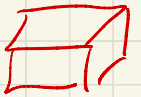
MAX-POOL1
$f=3$
$s=3$



$37 \times 37 \times 96$

$\left\lfloor \frac{111+0-3}{3} + 1 \right\rfloor = \left\lfloor \frac{108}{3} + 1 \right\rfloor = 36+1 = 37$

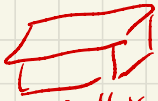
CONV2
$f=3$
$s=2$
$n_c=384$



$18 \times 18 \times 384$

$\left\lfloor \frac{37+0-3}{2} + 1 \right\rfloor = \lfloor 17+1 \rfloor = 18$

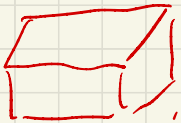
MAX-POOL
$f=3, s=1$



$16 \times 16 \times 384$

$\left\lfloor \frac{18+0-3}{1} + 1 \right\rfloor = 16$

CONV3
$f=6$
$s=2$
$n_c=256$
$p=2$



$8 \times 8 \times 256$

$\left\lfloor \frac{16-6+2\times 2}{2} + 1 \right\rfloor = 8$

=

$8 \times 8 \times 256$

1124

256

softmax

(1000 outputs)