Computer Vision II

Fundamentals of Artificial Intelligence

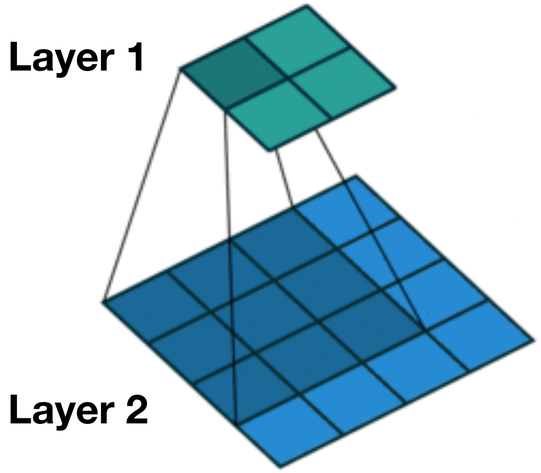
Fabien Cromieres
fabien@nlp.ist.i.kyoto-u.ac.jp
Kyoto University

Previously

- Last week, we looked at ***convolutional layers***
- And we saw that they can be mathematically represented by an operation called the "***convolution operation***"
  - Just like Fully-Connected layers can be represented by Matrix-Multiplication
- We also saw that the "convolution operation" can be applied to images with certain kernels to produce "edge detection"

Convolutional Layers

- Neurons are organized in 2-dimensional layers
- Neurons in 2 layers are only connected if they roughly belong to the same area of their respective layer
  - Eg. The neuron in the top-left corner of layer 2 is only connected to the 9 neurons in the top-left corner of layer 1



This gives spatial information to the network

Because all of the inputs of one neuron correspond to Neighboring pixels

Image: Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Convolution

Input Array

0	0	0	0	0	0
0	0	1	1	0	0
0	1	1	1	0	0
0	1	1	1	1	0
0	1	1	1	1	0
0	1	1	1	1	1

*

Kernel Array

2	0	2
0	1	0
-1	1	0

=

Output Array

1	1	1	-1
4	3	3	2
4	5	3	3
4	5	5	3

Convolution

- How we compute:

Input Array

0	0	0	0	0	0
0	0	1	1	0	0
0	1	1	1	0	0
0	1	1	1	1	0
0	1	1	1	1	0
0	1	1	1	1	1

*

Kernel Array

2	0	2
0	1	0
-1	1	0

=

Output Array

1	1	1	-1
4	3	3	2
4	5	3	3
4	5	5	3

0x2 + 0x0 + 0x2 + 0x0 + 0x1 + 1x0 + 0x-1 + 1x1 + 1x0 = 1

Convolution

- How we compute:

Input Array

0	0	0	0	0	0
0	0	1	1	0	0
0	1	1	1	0	0
0	1	1	1	1	0
0	1	1	1	1	0
0	1	1	1	1	1

*

Kernel Array

2	0	2
0	1	0
-1	1	0

=

Output Array

1	1	1	-1
4	3	3	2
4	5	3	3
4	5	5	3

0x2 + 0x0 + 1x2 + 0x0 + 1x1 + 1x0 + 0x-1 + 1x1 + 1x0 = 4

Convolution

- How we compute:

Input Array

0	0	0	0	0	0
0	0	1	1	0	0
0	1	1	1	0	0
0	1	1	1	1	0
0	1	1	1	1	0
0	1	1	1	1	1

*

Kernel Array

2	0	2
0	1	0
-1	1	0

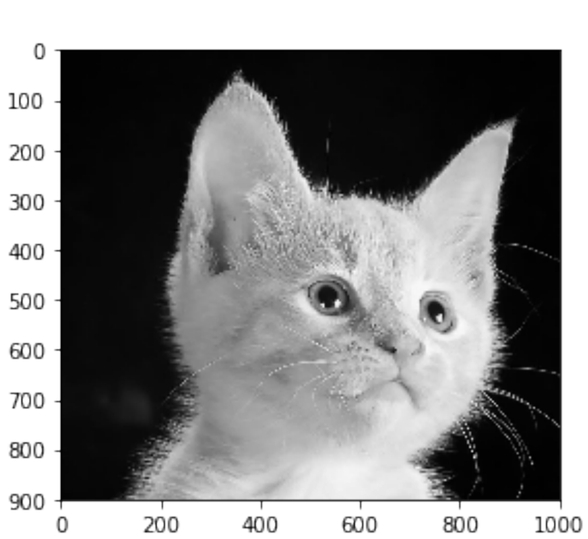
=

Output Array

1	1	1	-1
4	3	3	2
4	5	3	3
4	5	5	3

0x2 + 1x0 + 1x2 + 1x0 + 1x1 + 1x0 + 1x-1 + 1x1 + 1x0 = 3

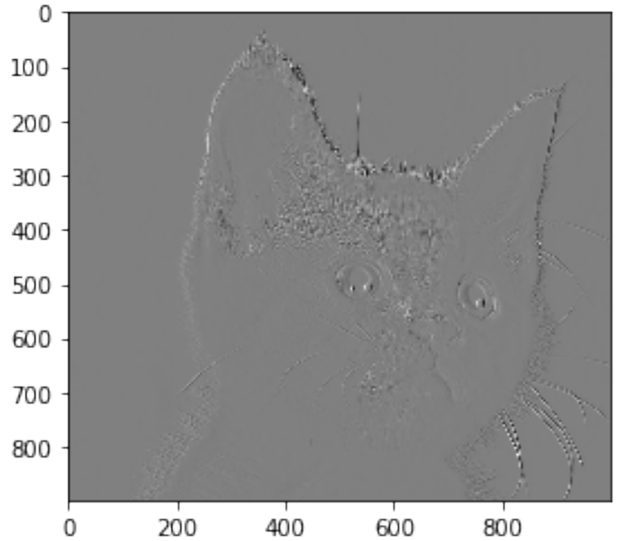
Edge Detectors

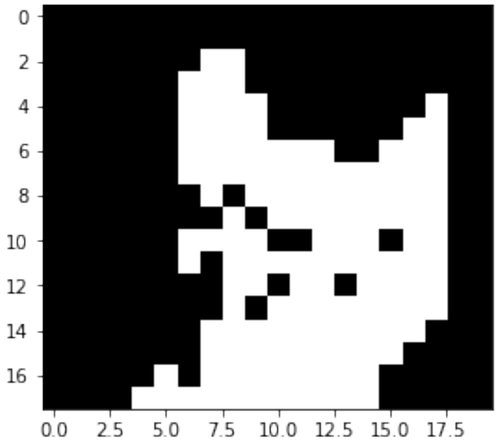


*

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

=

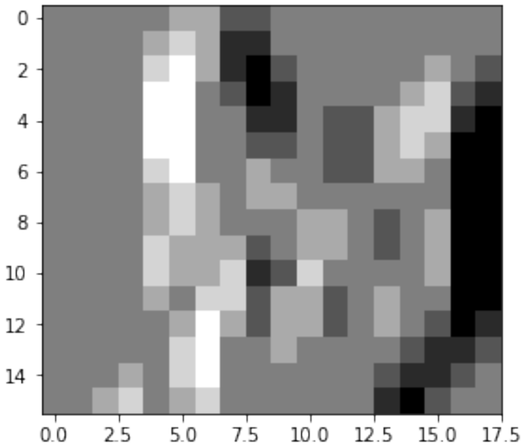




*

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

=



Today

- Today we:
  - Consider ***“volume” convolutions*** instead of the “flat” convolutions we just described
  - See one last type of layers: ***“Max-Pooling” layers***
  - Combine everything to create an ***Image Classifier***

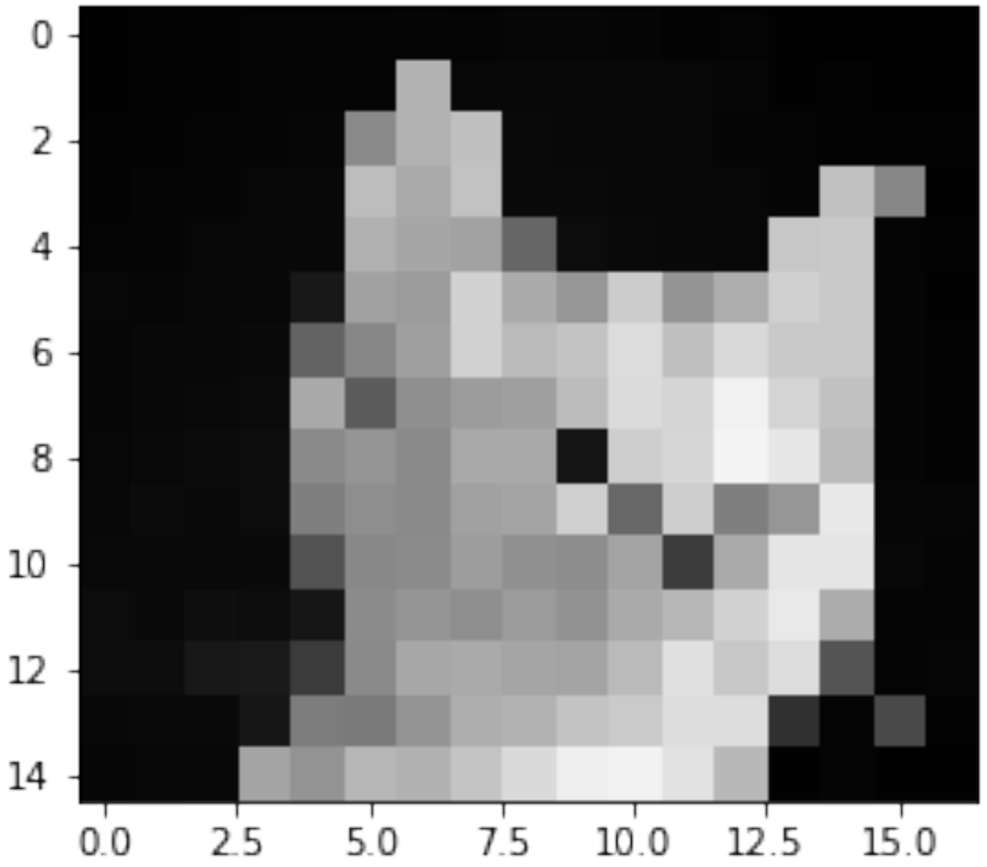
Volume Convolution

- When we discussed convolution, we considered the input was a 2D array of numbers
  - This 2D array corresponds for example, to a Black&White image

An image as a 2D array

- Greyscale image: each pixel has a grey value between 0(black) and 1 (white)

Image with 18x20 pixels (greyscale)



Array with 18x20 numbers

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.5	0.7	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.7	0.7	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.5	0.0
0.0	0.0	0.0	0.0	0.0	0.7	0.7	0.6	0.4	0.0	0.0	0.0	0.0	0.8	0.8	0.0	0.0
0.0	0.0	0.0	0.0	0.1	0.6	0.6	0.8	0.7	0.6	0.8	0.6	0.7	0.8	0.8	0.0	0.0
0.0	0.0	0.0	0.0	0.4	0.5	0.6	0.8	0.7	0.8	0.9	0.8	0.9	0.8	0.8	0.0	0.0
0.0	0.0	0.0	0.0	0.7	0.4	0.6	0.6	0.6	0.7	0.9	0.8	0.9	0.8	0.8	0.0	0.0
0.0	0.0	0.0	0.1	0.5	0.6	0.5	0.7	0.7	0.1	0.8	0.8	1.0	0.9	0.7	0.0	0.0
0.0	0.0	0.0	0.0	0.5	0.6	0.5	0.6	0.6	0.8	0.4	0.8	0.5	0.6	0.9	0.0	0.0
0.0	0.0	0.0	0.0	0.3	0.5	0.5	0.6	0.6	0.6	0.6	0.2	0.7	0.9	0.9	0.0	0.0
0.0	0.0	0.1	0.0	0.1	0.5	0.6	0.6	0.6	0.6	0.7	0.7	0.8	0.9	0.7	0.0	0.0
0.1	0.1	0.1	0.1	0.2	0.5	0.7	0.7	0.6	0.6	0.7	0.9	0.8	0.9	0.3	0.0	0.0
0.0	0.0	0.0	0.1	0.5	0.5	0.6	0.7	0.7	0.8	0.8	0.9	0.9	0.2	0.0	0.3	0.0
0.0	0.0	0.0	0.6	0.6	0.7	0.7	0.8	0.9	0.9	0.9	0.9	0.7	0.0	0.0	0.0	0.0

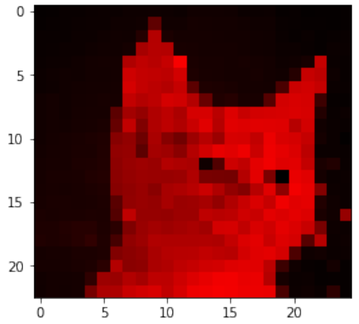
Volume Convolution

- When we discussed convolution, we considered the input was a 2D array of numbers
  - This 2D array corresponds for example, to a Black&White image
- What about color images?
  - Color images can be represented by 3D arrays

Color images

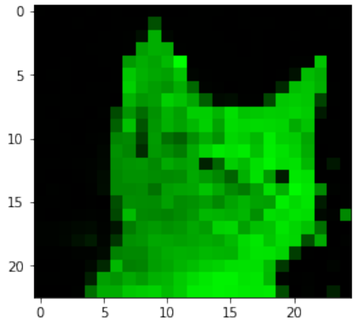
R channel (red): 18x20 array

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.5	0.7	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.7	0.7	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.5	0.0
0.0	0.0	0.0	0.0	0.0	0.7	0.7	0.6	0.4	0.0	0.0	0.0	0.0	0.0	0.8	0.8	0.0	0.0
0.0	0.0	0.0	0.0	0.1	0.6	0.6	0.8	0.7	0.6	0.8	0.6	0.7	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.4	0.5	0.6	0.8	0.7	0.8	0.9	0.8	0.9	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.7	0.4	0.6	0.6	0.6	0.7	0.9	0.8	0.9	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.1	0.5	0.6	0.5	0.7	0.7	0.1	0.8	0.8	1.0	0.9	0.7	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.5	0.6	0.5	0.6	0.6	0.8	0.4	0.8	0.5	0.6	0.9	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.3	0.5	0.5	0.6	0.6	0.6	0.6	0.2	0.7	0.9	0.9	0.0	0.0	0.0
0.0	0.0	0.1	0.0	0.1	0.5	0.6	0.6	0.6	0.6	0.7	0.7	0.8	0.9	0.7	0.0	0.0	0.0
0.1	0.1	0.1	0.1	0.2	0.5	0.7	0.7	0.6	0.6	0.7	0.9	0.8	0.9	0.3	0.0	0.0	0.0
0.0	0.0	0.0	0.1	0.5	0.5	0.6	0.7	0.7	0.8	0.8	0.9	0.9	0.2	0.0	0.3	0.0	
0.0	0.0	0.0	0.6	0.6	0.7	0.7	0.8	0.9	0.9	0.9	0.9	0.7	0.0	0.0	0.0	0.0	0.0



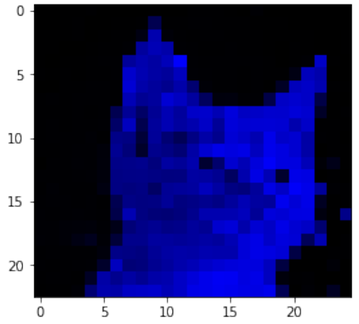
G channel (green): 18x20 array

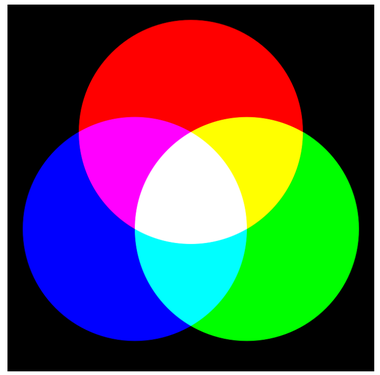
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.5	0.7	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.7	0.7	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.5	0.0
0.0	0.0	0.0	0.0	0.0	0.7	0.7	0.6	0.4	0.0	0.0	0.0	0.0	0.0	0.8	0.8	0.0	0.0
0.0	0.0	0.0	0.0	0.1	0.6	0.6	0.8	0.7	0.6	0.8	0.6	0.7	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.4	0.5	0.6	0.8	0.7	0.8	0.9	0.8	0.9	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.7	0.4	0.6	0.6	0.6	0.7	0.9	0.8	0.9	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.1	0.5	0.6	0.5	0.7	0.7	0.1	0.8	0.8	1.0	0.9	0.7	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.5	0.6	0.5	0.6	0.6	0.8	0.4	0.8	0.5	0.6	0.9	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.3	0.5	0.5	0.6	0.6	0.6	0.6	0.2	0.7	0.9	0.9	0.0	0.0	0.0
0.0	0.0	0.1	0.0	0.1	0.5	0.6	0.6	0.6	0.6	0.7	0.7	0.8	0.9	0.7	0.0	0.0	0.0
0.1	0.1	0.1	0.1	0.2	0.5	0.7	0.7	0.6	0.6	0.7	0.9	0.8	0.9	0.3	0.0	0.0	0.0
0.0	0.0	0.0	0.1	0.5	0.5	0.6	0.7	0.7	0.8	0.8	0.9	0.9	0.2	0.0	0.3	0.0	
0.0	0.0	0.0	0.6	0.6	0.7	0.7	0.8	0.9	0.9	0.9	0.9	0.7	0.0	0.0	0.0	0.0	0.0



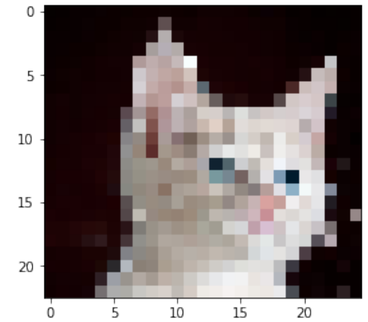
B channel (blue): 18x20 array

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.5	0.7	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.7	0.7	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.5	0.0
0.0	0.0	0.0	0.0	0.0	0.7	0.7	0.6	0.4	0.0	0.0	0.0	0.0	0.0	0.8	0.8	0.0	0.0
0.0	0.0	0.0	0.0	0.1	0.6	0.6	0.8	0.7	0.6	0.8	0.6	0.7	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.4	0.5	0.6	0.8	0.7	0.8	0.9	0.8	0.9	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.7	0.4	0.6	0.6	0.6	0.7	0.9	0.8	0.9	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.1	0.5	0.6	0.5	0.7	0.7	0.1	0.8	0.8	1.0	0.9	0.7	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.5	0.6	0.5	0.6	0.6	0.8	0.4	0.8	0.5	0.6	0.9	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.3	0.5	0.5	0.6	0.6	0.6	0.6	0.2	0.7	0.9	0.9	0.0	0.0	0.0
0.0	0.0	0.1	0.0	0.1	0.5	0.6	0.6	0.6	0.6	0.7	0.7	0.8	0.9	0.7	0.0	0.0	0.0
0.1	0.1	0.1	0.1	0.2	0.5	0.7	0.7	0.6	0.6	0.7	0.9	0.8	0.9	0.3	0.0	0.0	0.0
0.0	0.0	0.0	0.1	0.5	0.5	0.6	0.7	0.7	0.8	0.8	0.9	0.9	0.2	0.0	0.3	0.0	
0.0	0.0	0.0	0.6	0.6	0.7	0.7	0.8	0.9	0.9	0.9	0.9	0.7	0.0	0.0	0.0	0.0	0.0





Superpose all channels



In a computer, a color image is usually represented in the RGB format

We separate the color of the images in Red, Green and Blue Components

We have then 3 images that can each be represented by a 2D array

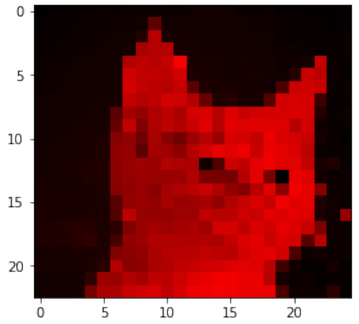
Color images

- In a computer, a color image is usually represented in the ***RGB format***
- We separate the color of the images in Red, Green and Blue Components
- We have then 3 images that can each be represented by a 2D array
  - One color image = three 2D arrays

Color images

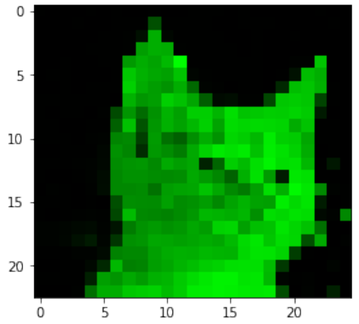
R channel (red): 18x20 array

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.5	0.7	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.7	0.7	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.5	0.0
0.0	0.0	0.0	0.0	0.0	0.7	0.7	0.6	0.4	0.0	0.0	0.0	0.0	0.0	0.8	0.8	0.0	0.0
0.0	0.0	0.0	0.0	0.1	0.6	0.6	0.8	0.7	0.6	0.8	0.6	0.7	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.4	0.5	0.6	0.8	0.7	0.8	0.9	0.8	0.9	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.7	0.4	0.6	0.6	0.6	0.7	0.9	0.8	0.9	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.1	0.5	0.6	0.5	0.7	0.7	0.1	0.8	0.8	1.0	0.9	0.7	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.5	0.6	0.5	0.6	0.6	0.8	0.4	0.8	0.5	0.6	0.9	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.3	0.5	0.5	0.6	0.6	0.6	0.6	0.2	0.7	0.9	0.9	0.0	0.0	0.0
0.0	0.0	0.1	0.0	0.1	0.5	0.6	0.6	0.6	0.6	0.7	0.7	0.8	0.9	0.7	0.0	0.0	0.0
0.1	0.1	0.1	0.1	0.2	0.5	0.7	0.7	0.6	0.6	0.7	0.9	0.8	0.9	0.3	0.0	0.0	0.0
0.0	0.0	0.0	0.1	0.5	0.5	0.6	0.7	0.7	0.8	0.8	0.9	0.9	0.2	0.0	0.3	0.0	
0.0	0.0	0.0	0.6	0.6	0.7	0.7	0.8	0.9	0.9	0.9	0.9	0.7	0.0	0.0	0.0	0.0	0.0



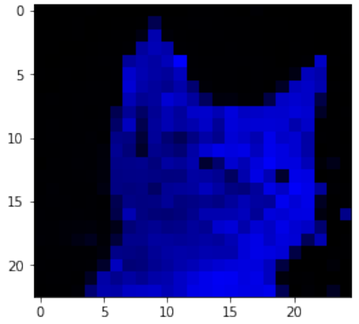
G channel (green): 18x20 array

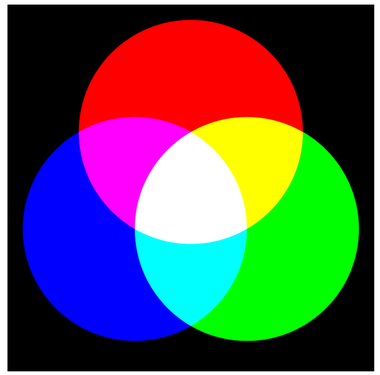
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.5	0.7	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.7	0.7	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.5	0.0
0.0	0.0	0.0	0.0	0.0	0.7	0.7	0.6	0.4	0.0	0.0	0.0	0.0	0.0	0.8	0.8	0.0	0.0
0.0	0.0	0.0	0.0	0.1	0.6	0.6	0.8	0.7	0.6	0.8	0.6	0.7	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.4	0.5	0.6	0.8	0.7	0.8	0.9	0.8	0.9	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.7	0.4	0.6	0.6	0.6	0.7	0.9	0.8	0.9	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.1	0.5	0.6	0.5	0.7	0.7	0.1	0.8	0.8	1.0	0.9	0.7	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.5	0.6	0.5	0.6	0.6	0.8	0.4	0.8	0.5	0.6	0.9	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.3	0.5	0.5	0.6	0.6	0.6	0.6	0.2	0.7	0.9	0.9	0.0	0.0	0.0
0.0	0.0	0.1	0.0	0.1	0.5	0.6	0.6	0.6	0.6	0.7	0.7	0.8	0.9	0.7	0.0	0.0	0.0
0.1	0.1	0.1	0.1	0.2	0.5	0.7	0.7	0.6	0.6	0.7	0.9	0.8	0.9	0.3	0.0	0.0	0.0
0.0	0.0	0.0	0.1	0.5	0.5	0.6	0.7	0.7	0.8	0.8	0.9	0.9	0.2	0.0	0.3	0.0	
0.0	0.0	0.0	0.6	0.6	0.7	0.7	0.8	0.9	0.9	0.9	0.9	0.7	0.0	0.0	0.0	0.0	0.0



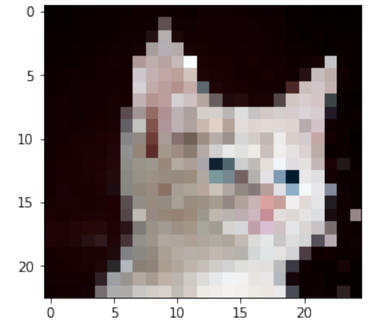
B channel (blue): 18x20 array

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.5	0.7	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.7	0.7	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.5	0.0
0.0	0.0	0.0	0.0	0.0	0.7	0.7	0.6	0.4	0.0	0.0	0.0	0.0	0.0	0.8	0.8	0.0	0.0
0.0	0.0	0.0	0.0	0.1	0.6	0.6	0.8	0.7	0.6	0.8	0.6	0.7	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.4	0.5	0.6	0.8	0.7	0.8	0.9	0.8	0.9	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.7	0.4	0.6	0.6	0.6	0.7	0.9	0.8	0.9	0.8	0.8	0.0	0.0	0.0
0.0	0.0	0.0	0.1	0.5	0.6	0.5	0.7	0.7	0.1	0.8	0.8	1.0	0.9	0.7	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.5	0.6	0.5	0.6	0.6	0.8	0.4	0.8	0.5	0.6	0.9	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.3	0.5	0.5	0.6	0.6	0.6	0.6	0.2	0.7	0.9	0.9	0.0	0.0	0.0
0.0	0.0	0.1	0.0	0.1	0.5	0.6	0.6	0.6	0.6	0.7	0.7	0.8	0.9	0.7	0.0	0.0	0.0
0.1	0.1	0.1	0.1	0.2	0.5	0.7	0.7	0.6	0.6	0.7	0.9	0.8	0.9	0.3	0.0	0.0	0.0
0.0	0.0	0.0	0.1	0.5	0.5	0.6	0.7	0.7	0.8	0.8	0.9	0.9	0.2	0.0	0.3	0.0	
0.0	0.0	0.0	0.6	0.6	0.7	0.7	0.8	0.9	0.9	0.9	0.9	0.7	0.0	0.0	0.0	0.0	0.0





Superpose all channels



In a computer, a color image is usually represented in the RGB format

We separate the color of the images in Red, Green and Blue Components

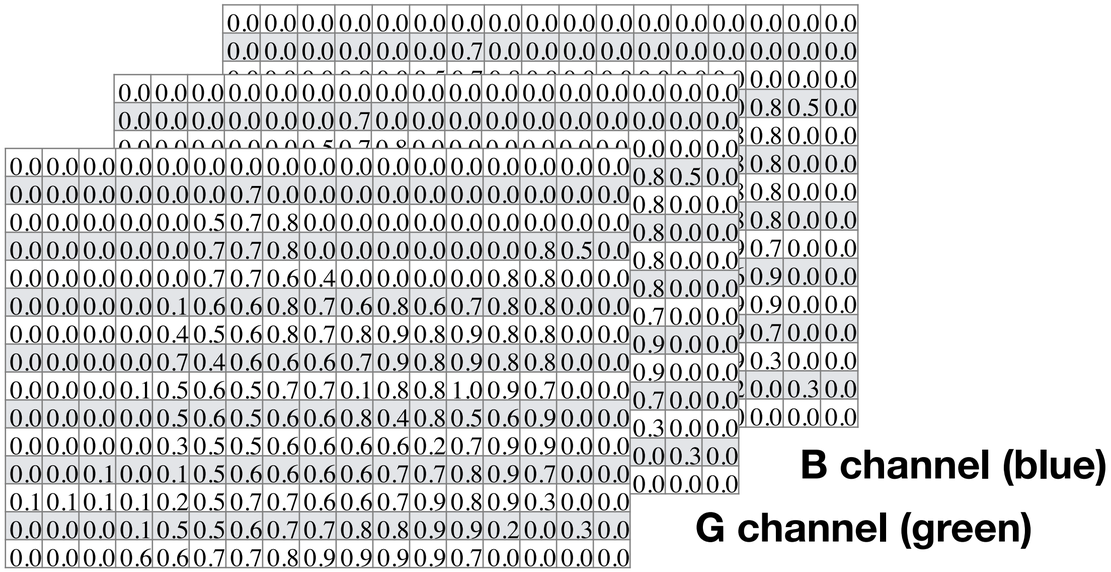
We have then 3 images that can each be represented by a 2D array

Color images

- In a computer, a color image is usually represented in the ***RGB format***
- We separate the color of the images in Red, Green and Blue Components
- We have then 3 images that can each be represented by a 2D array
  - One color image = three 2D arrays
  - One color image = one 3D array

Color images

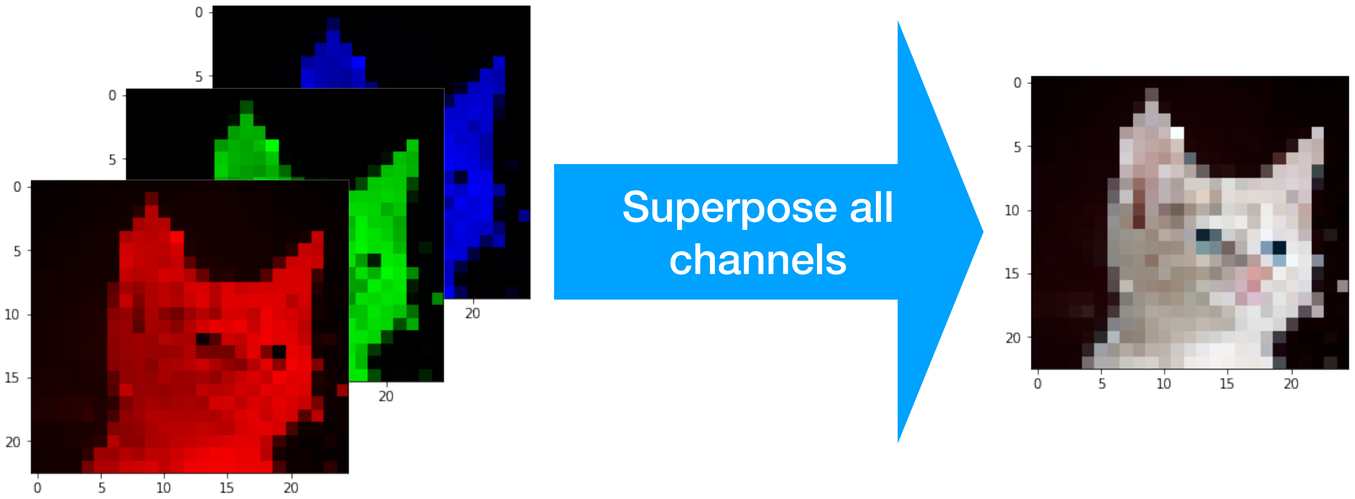
An 18x20x3 array



B channel (blue)

G channel (green)

R channel (red)



In a computer, a color image is usually represented in the <u>RGB format</u>

We separate the color of the images in Red, Green and Blue Components

We have then 3 images that can each be represented by a 2D array

Volume Convolution

- When we discussed convolution, we considered the input was a 2D array of numbers
  - A 2D array corresponds for example, to a Black&White image
- What about color images?
  - Color images can be represented by 3D arrays
  - In Image processing, it is a convention to call the 3rd dimension the ***“channel”*** dimension
- How do we apply convolutions to 3D arrays?

“Flat” Convolution

- How we compute:

Input Array

0	0	0	0	0	0
0	0	1	1	0	0
0	1	1	1	0	0
0	1	1	1	1	0
0	1	1	1	1	0
0	1	1	1	1	1

*

Kernel Array

2	0	2
0	1	0
-1	1	0

=

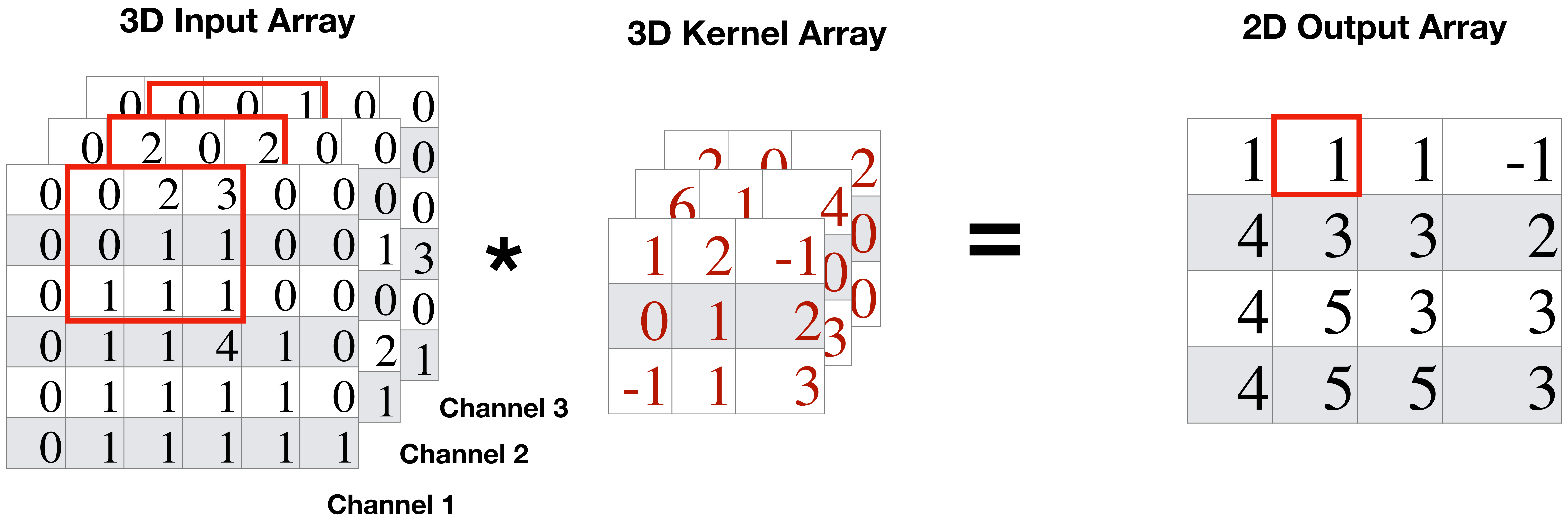
Output Array

1	1	1	-1
4	3	3	2
4	5	3	3
4	5	5	3

0x2 + 1x0 + 1x2 + 1x0 + 1x1 + 1x0 + 1x-1 + 1x1 + 1x0 = 3

Volume Convolution

- We now consider we have a Kernel Array whose 3rd dimension is the same as the input
- Computation is the same as for 2D input, but we sum across channels
- Result is a 2D array



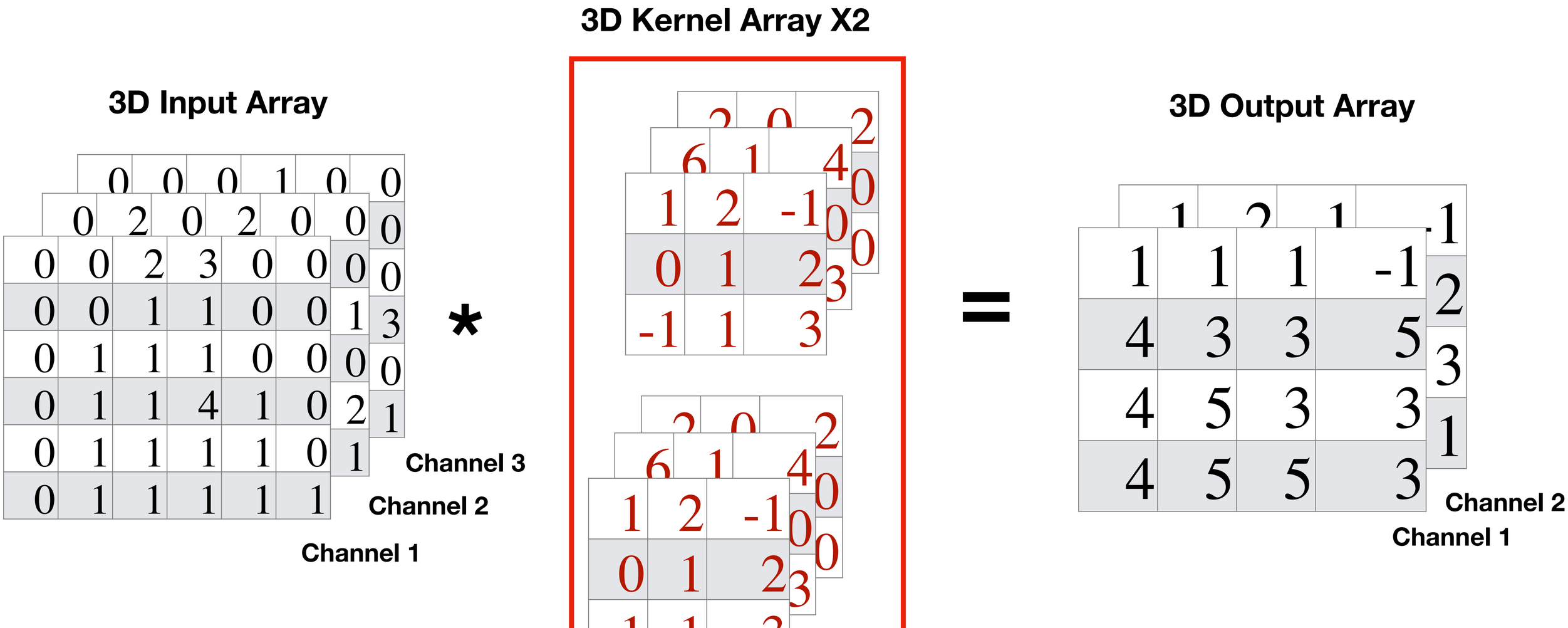
0x1 + 2x2 + 3x-1+ 0x0 + 1x1 + 1x2 + 1x-1 + 1x1 + 1x3 + 2x6 +0x1 +2x4 + … = 1

Volume Convolution

- If we apply a 3D Kernel to a 3D input, we get a 2D array
- Can we get a 3D output array?
  - Yes, by using more than one kernel

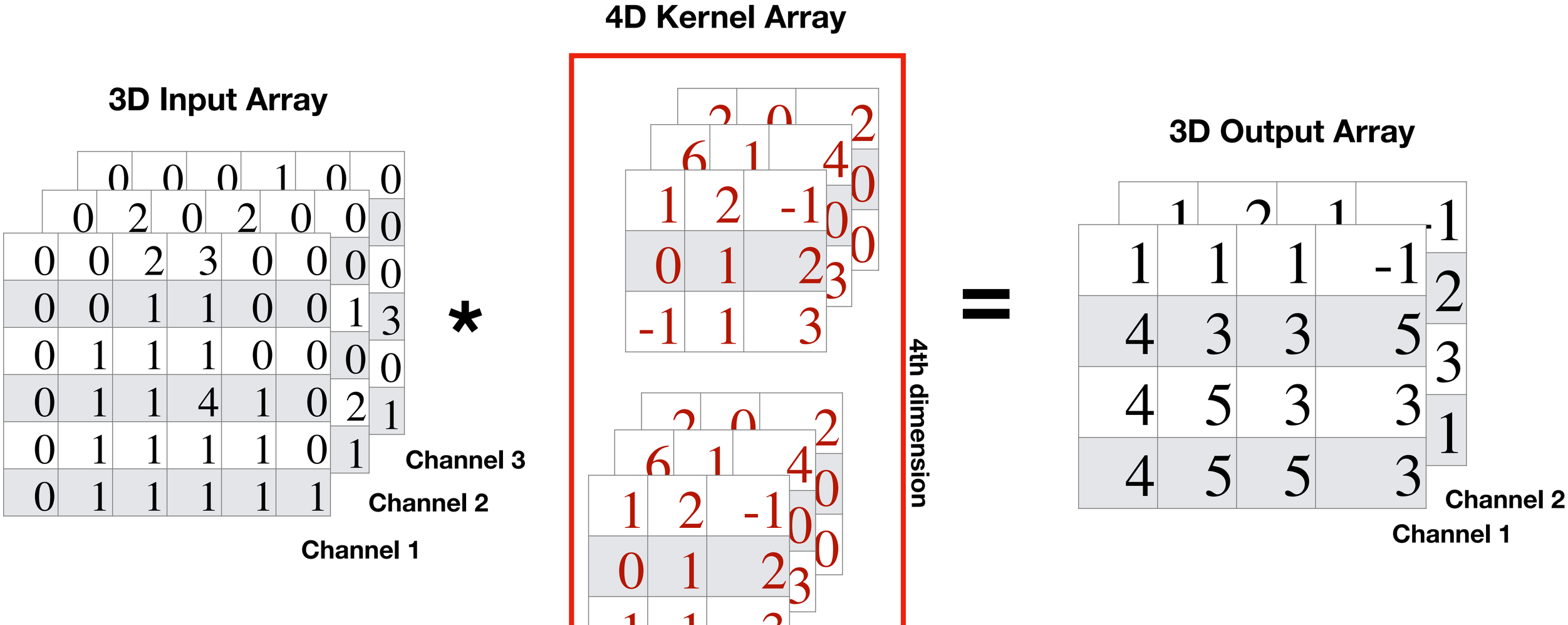
Volume Convolution

- Also, we can convolve the input with more than one kernel at a time to produce a 3D output with more than one channel



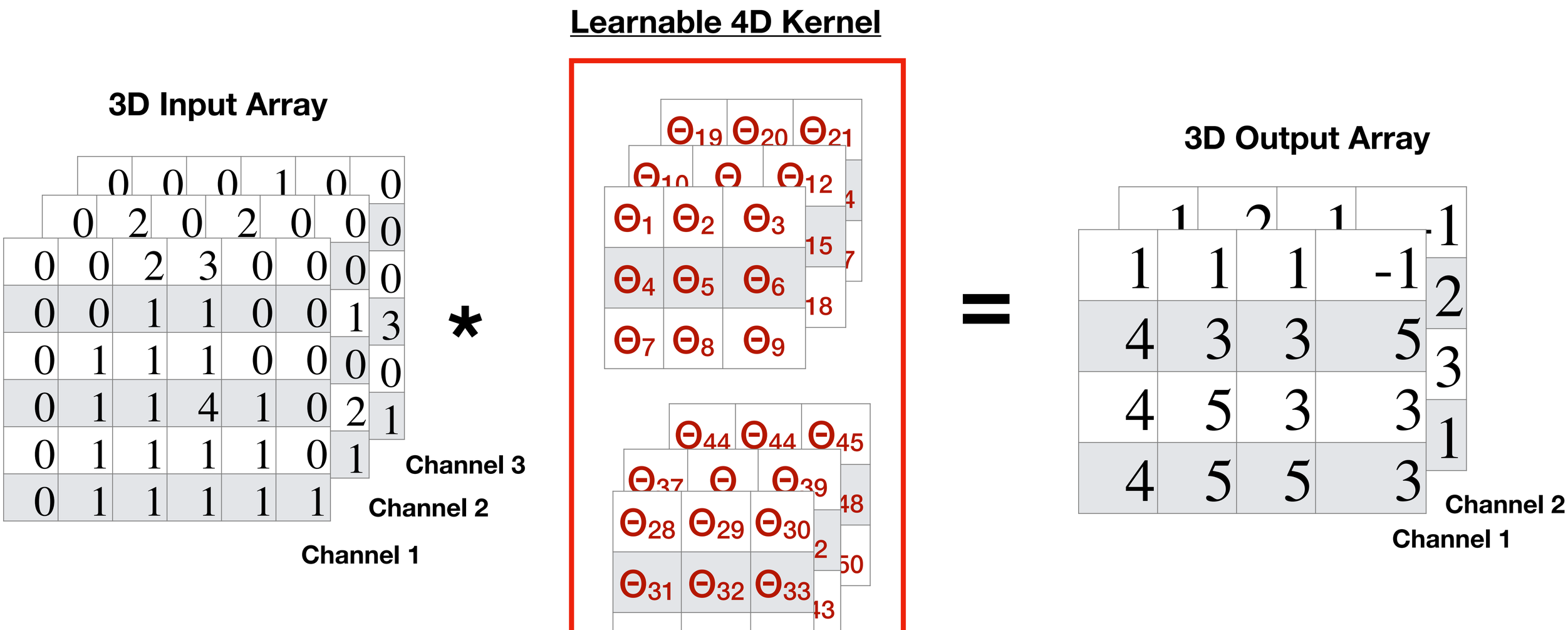
Volume Convolution

- Also, we can convolve the input with more than one kernel at a time to produce a 3D output with more than one channel



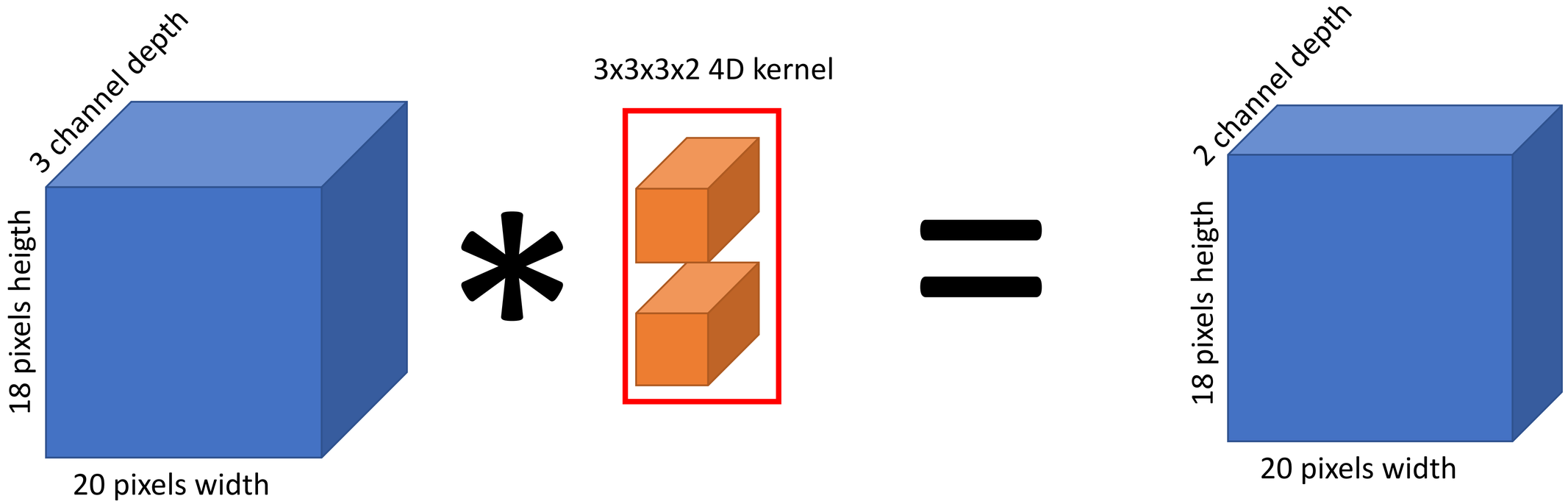
Volume Convolution

- The values of the 4D kernel is what we are going to learn when we train our Convolutional Network



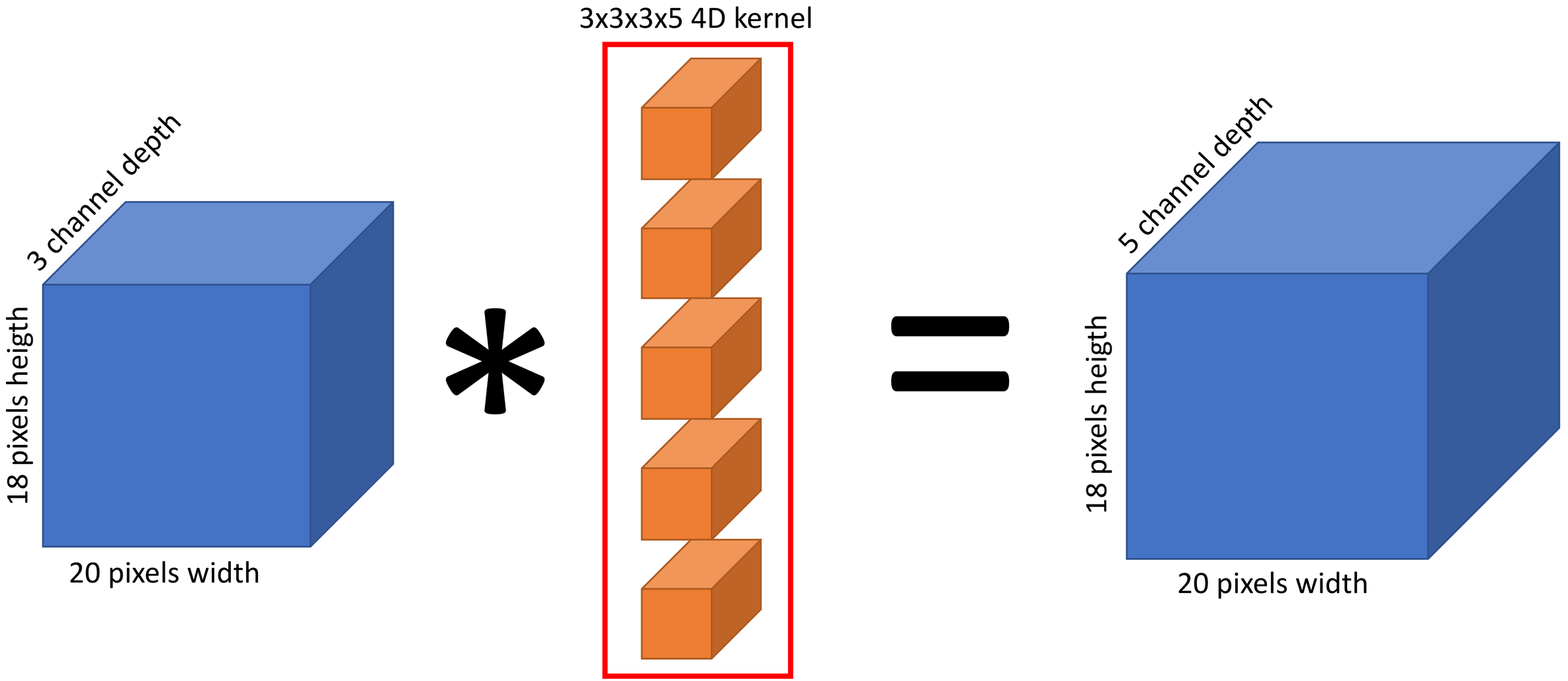
Volume Convolution

- For visualization, it can be interesting to forget the numbers, and just look at 3D arrays as if they were 3D shapes



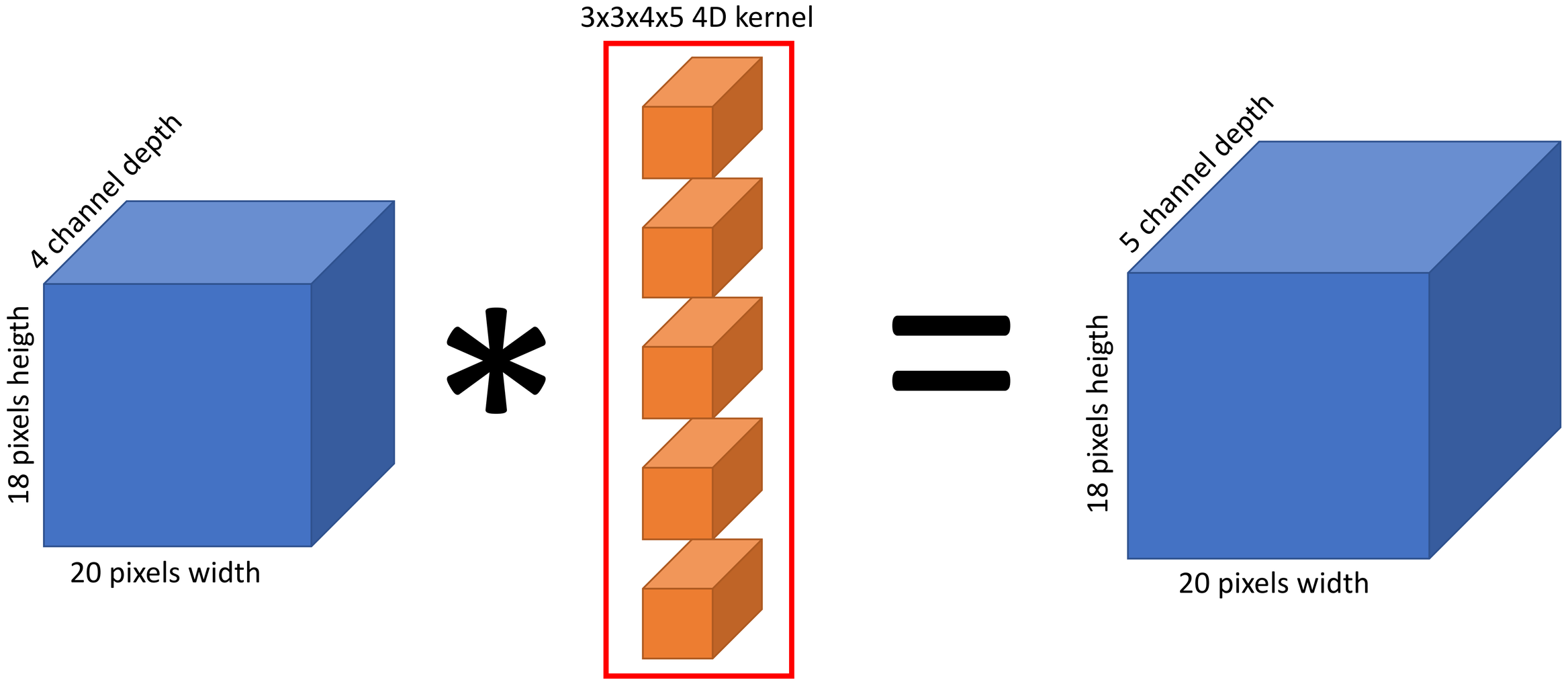
Volume Convolution

- We can generalize to any number of input and output channels



Volume Convolution

- We can generalize to any number of input and output channels



Volume Convolution

- In short the most important thing to remember:
  - Convolutions can take a 3D array as input
  - Can produce a 3D array as output
  - The number of *channels* (ie. third dimension) of input and output array can be different
- Now, we are going to see one last operation: ***“Max Pooling***"

Max Pooling

- Another operation commonly used in CNN: Max Pooling
- Simply takes the max of an area of the input
- Used to reduce the size of the input

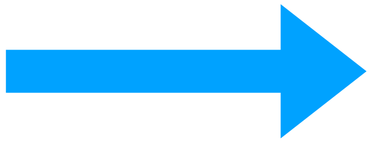
1	1	1	-1
4	3	3	2
4	5	3	3
4	5	5	3

4	3
5	5

Max Pooling

- Another operation commonly used in CNN: Max Pooling
- Simply takes the max of an area of the input
- Used to reduce the size of the input

1	1	1	-1
4	3	3	2
4	5	3	3
4	5	5	3



4	3
5	5

Max Pooling

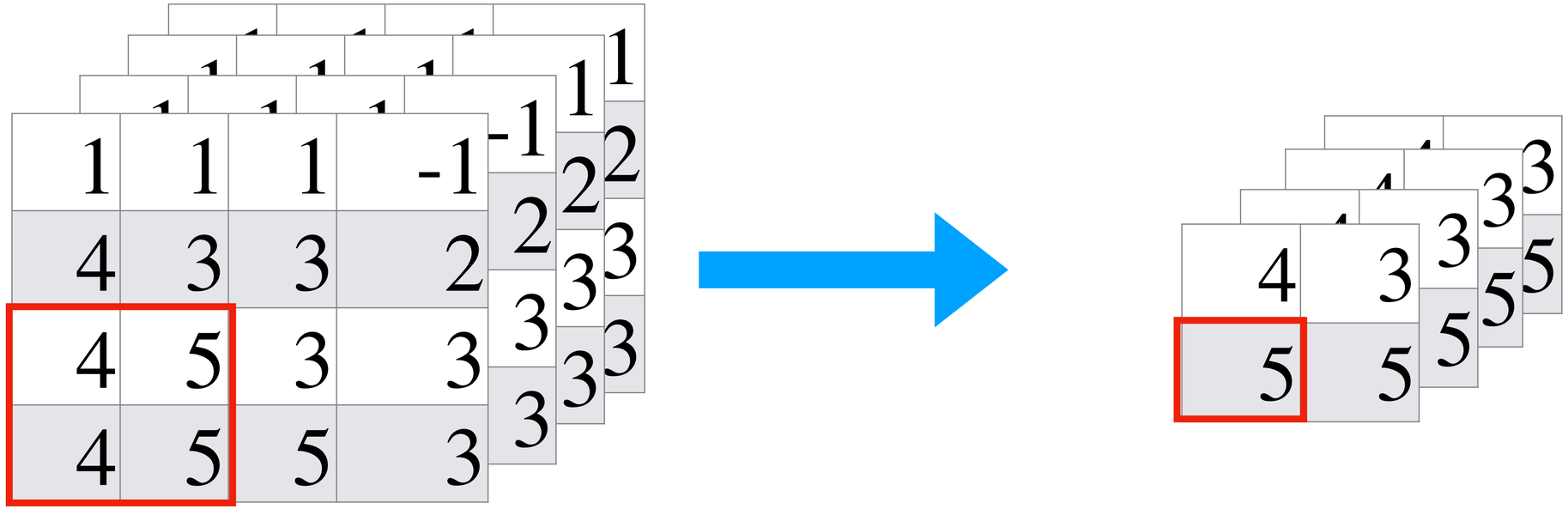
- Another operation commonly used in CNN: Max Pooling
- Simply takes the max of an area of the input
- Used to reduce the size of the input

1	1	1	-1
4	3	3	2
4	5	3	3
4	5	5	3

4	3
5	5

Max Pooling

- Another operation commonly used in CNN: Max Pooling
- Simply takes the max of an area of the input
- Used to reduce the size of the input



“Volume” Max Pooling

- Max-Pooling can be applied to a 3D array as well
- It is applied to each channel separately
  - The input is a 3D array
  - The output is a 3D array with the same number of channels as the input
  - But with other dimensions divided by 2

Max Pooling

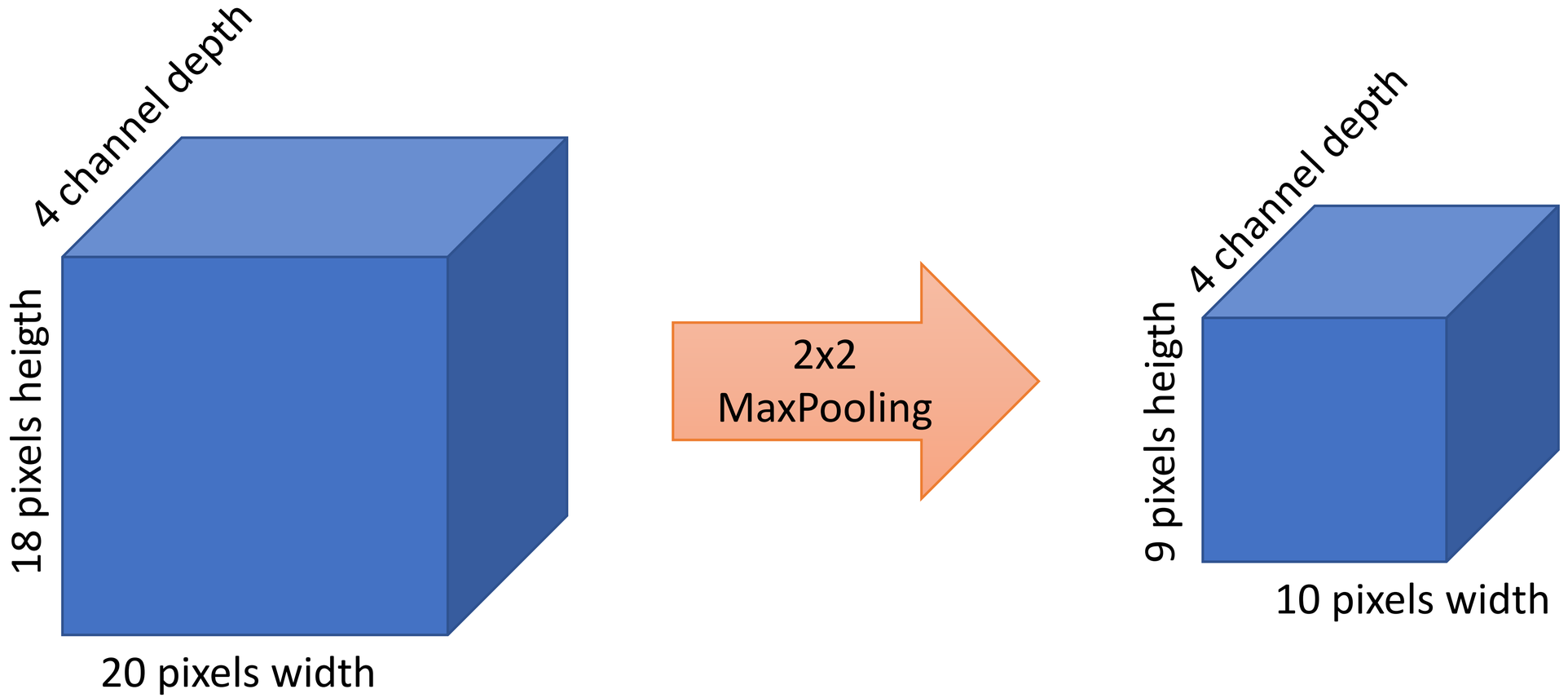

4 channel depth
18 pixels heigth
20 pixels width
2x2
MaxPooling
4 channel depth
9 pixels heigth
10 pixels width

Image Classifier

- Finally, we can look at how we build a full image classifier
- A typical modern image classifier is a Multi-Layered Neural Network
  - Input image is sent to a convolutional Layer
  - The result is sent to a Max-Pooling layer
  - The result is sent to a Convolutional Layer
  - And so on….

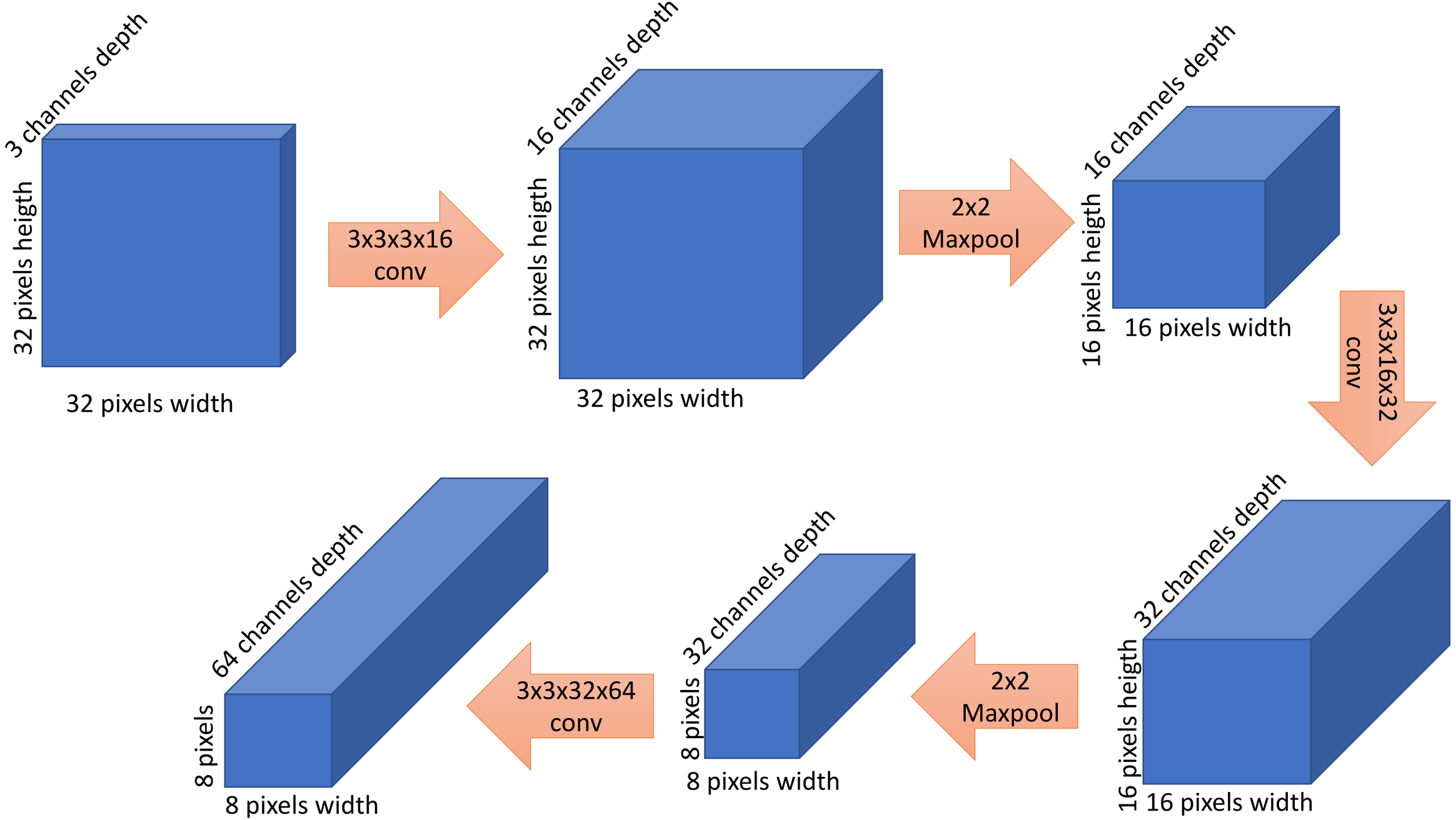

3 channels depth
32 pixels heigth
32 pixels width
3x3x3x16
conv
16 channels depth
32 pixels heigth
32 pixels width
2x2
Maxpool
16 channels depth
16 pixels heigth
16 pixels width
3x3x16x32
conv
32 channels depth
16 pixels heigth
16 pixels width
2x2
Maxpool
32 channels depth
8 pixels
8 pixels width
3x3x32x64
conv
64 channels depth
8 pixels
8 pixels width

Image Classifier

- Finally, we can look at how we build a full image classifier
- A typical modern image classifier is a Multi-Layered Neural Network
- Each Convolutional Layer + Max-Pooling Layer produce a 3D array that is “narrower” and “deeper” than the input

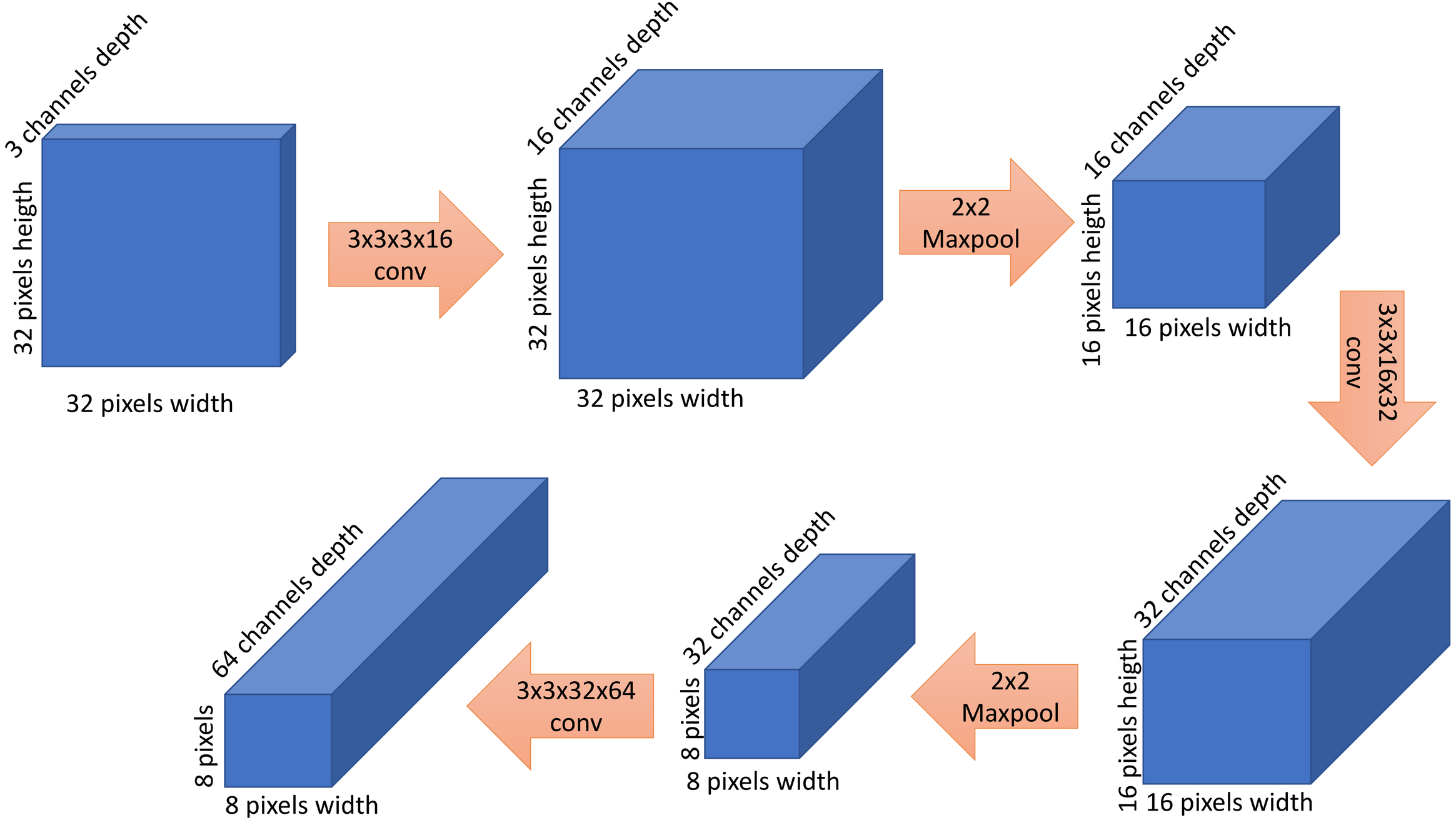
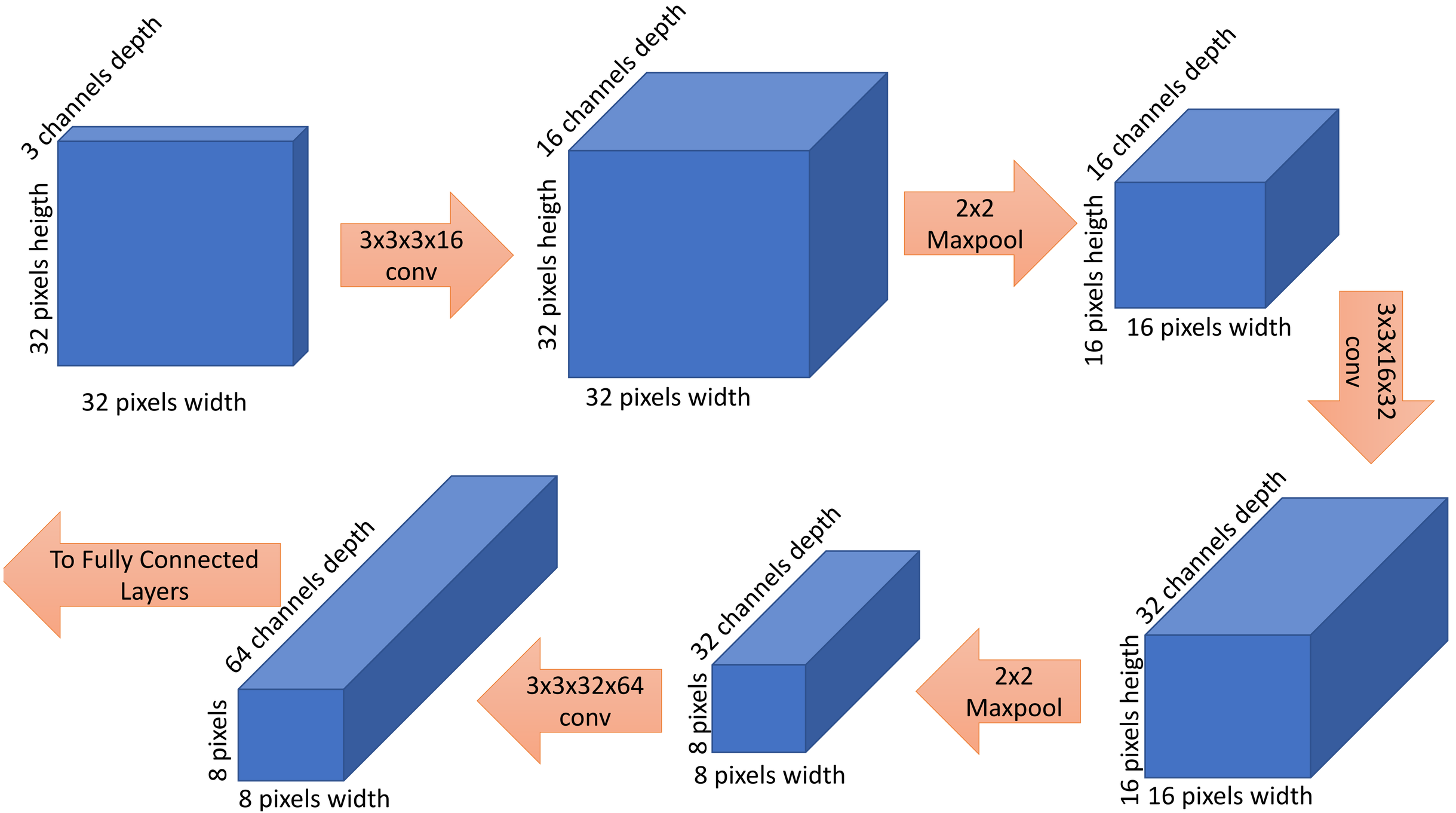
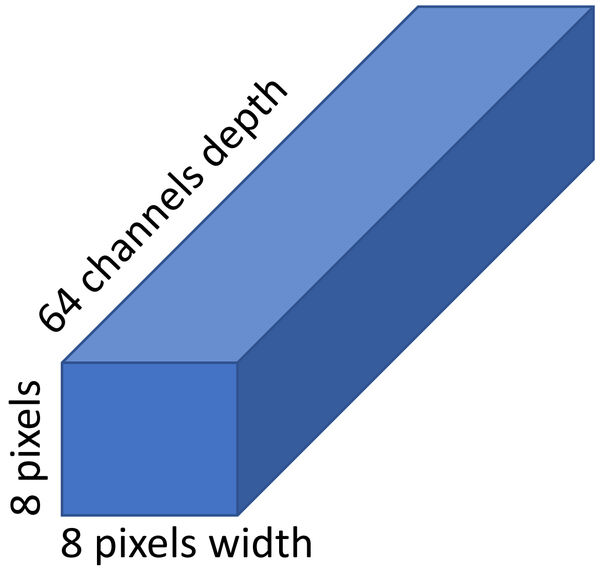

3 channels depth
32 pixels heigth
32 pixels width
3x3x3x16
conv
16 channels depth
32 pixels heigth
32 pixels width
2x2
Maxpool
16 channels depth
16 pixels heigth
16 pixels width
3x3x16x32
conv
32 channels depth
16 pixels heigth
16 pixels width
2x2
Maxpool
32 channels depth
8 pixels
8 pixels width
3x3x32x64
conv
64 channels depth
8 pixels
8 pixels width

Image Classifier

- Finally, we can look at how we build a full image classifier
- A typical modern image classifier is a Multi-Layered Neural Network
- Each Convolutional Layer + Max-Pooling Layer produce a 3D array that is "narrower" and "deeper" than the input
- At the end, we send the "deep and narrow" 3D array to a Fully-Connected Classifier


3 channels depth
32 pixels heigth
32 pixels width
3x3x3x16
conv
16 channels depth
32 pixels heigth
32 pixels width
2x2
Maxpool
16 channels depth
16 pixels heigth
16 pixels width
3x3x16x32
conv
32 channels depth
16 pixels heigth
16 pixels width
2x2
Maxpool
32 channels depth
8 pixels
8 pixels width
3x3x32x64
conv
64 channels depth
8 pixels
8 pixels width
To Fully Connected
Layers

Final Fully Connected Classifier



Fully Connected layer with 1000 neurons with 8x8x64 = 4094 inputs each

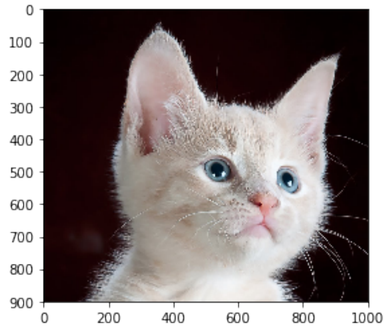
Fully Connected layer with 200 neurons with 1000 inputs each

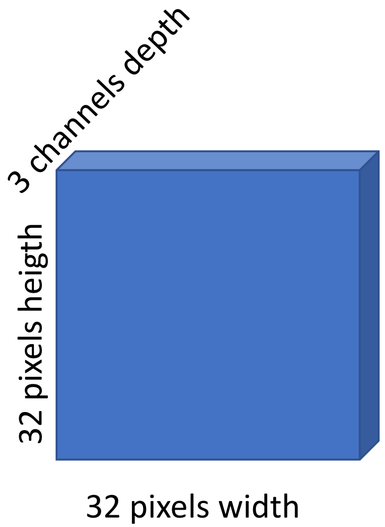
Final logistic classifier 200 inputs 1 output

Cat?

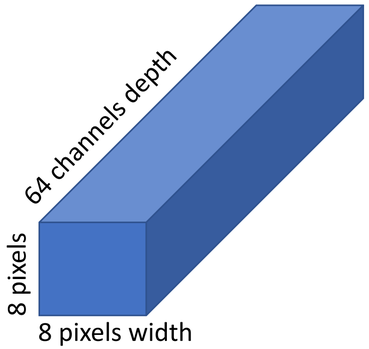
Image Classifier

Input Image as a 3D array





Alternate Convolutional Layers and Max Pooling Layers



Classifier with Fully Connected Layers

Cat?

Image Classifier

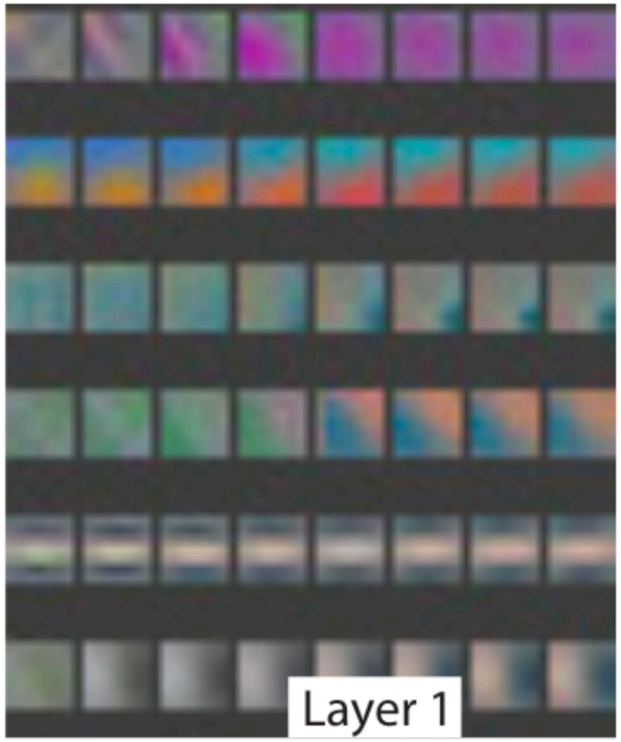
- Finally, we can look at how we build a full image classifier
- A typical modern image classifier is a Multi-Layered Neural Network
- Each Convolutional Layer + Max-Pooling Layer produce a 3D array that is "narrower" and "deeper" than the input
- At the end, we send the "deep and narrow" 3D array to a Fully-Connected Classifier

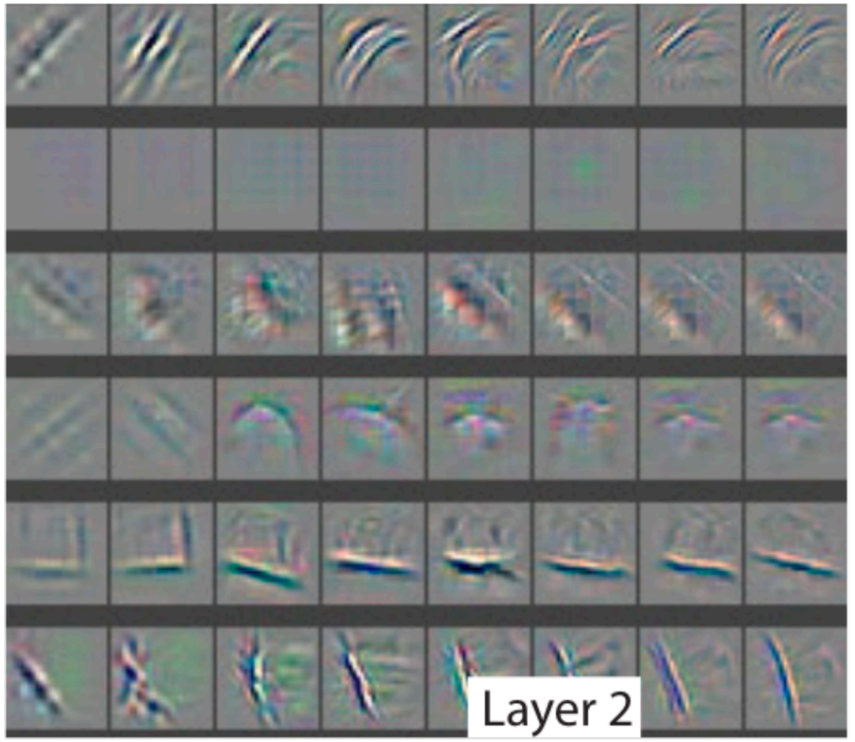
Visualization of a CNN

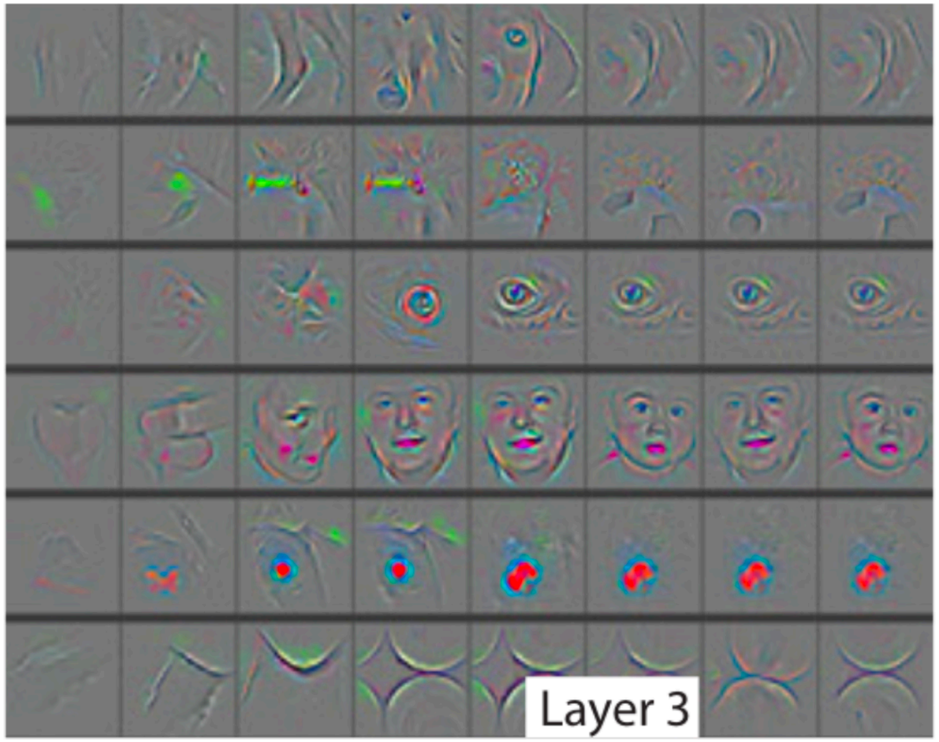
- Let us look at an excellent online visualization tool:
  - Convolutional Network
    - 3D representation: http://scs.ryerson.ca/~aharley/vis/conv/
    - 2D representation: http://scs.ryerson.ca/~aharley/vis/conv/flat.html
  - Fully Connected Network:
    - http://scs.ryerson.ca/~aharley/vis/fc/

A. W. Harley, "An Interactive Node-Link Visualization of Convolutional Neural Networks," in ISVC, pages 867-877, 2015

What do the convolution kernels learn to recognize?

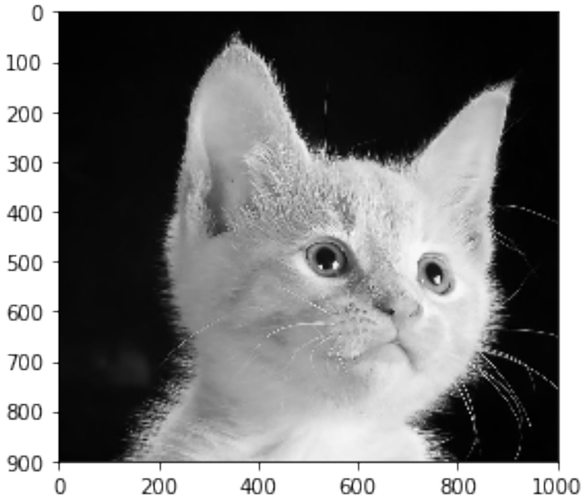






From Zeiler&Fergus “Visualizing and Understanding Convolutional Networks”

Learnable Kernels and Edge Detectors



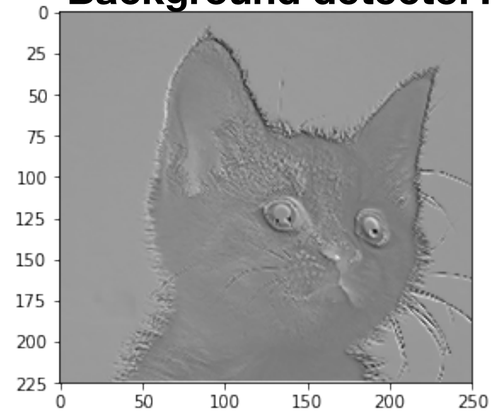
*

In practice, we will be learning these parameters from examples:

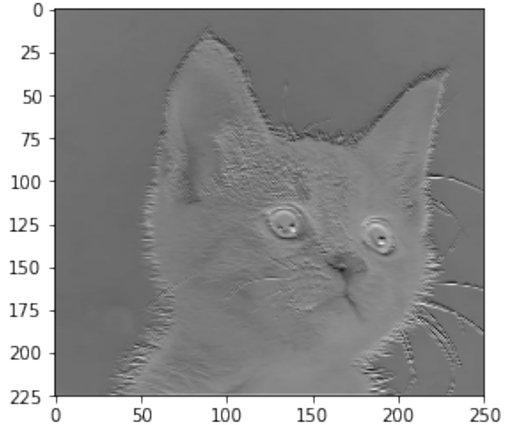
Θ_1	Θ_2	Θ_3
Θ_4	Θ_5	Θ_6
Θ_7	Θ_8	Θ_9

=

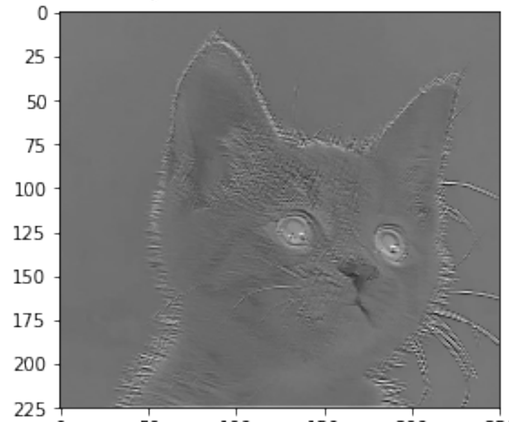
Background detector?



Fur detector?

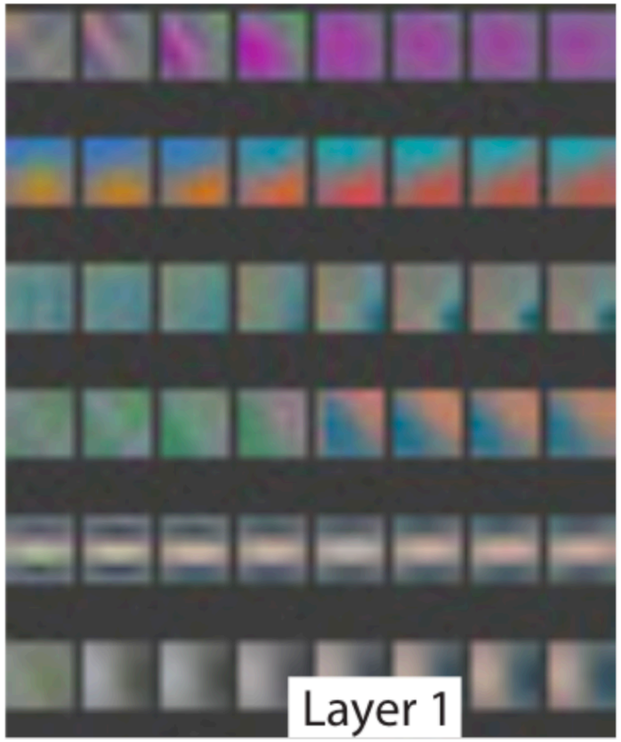


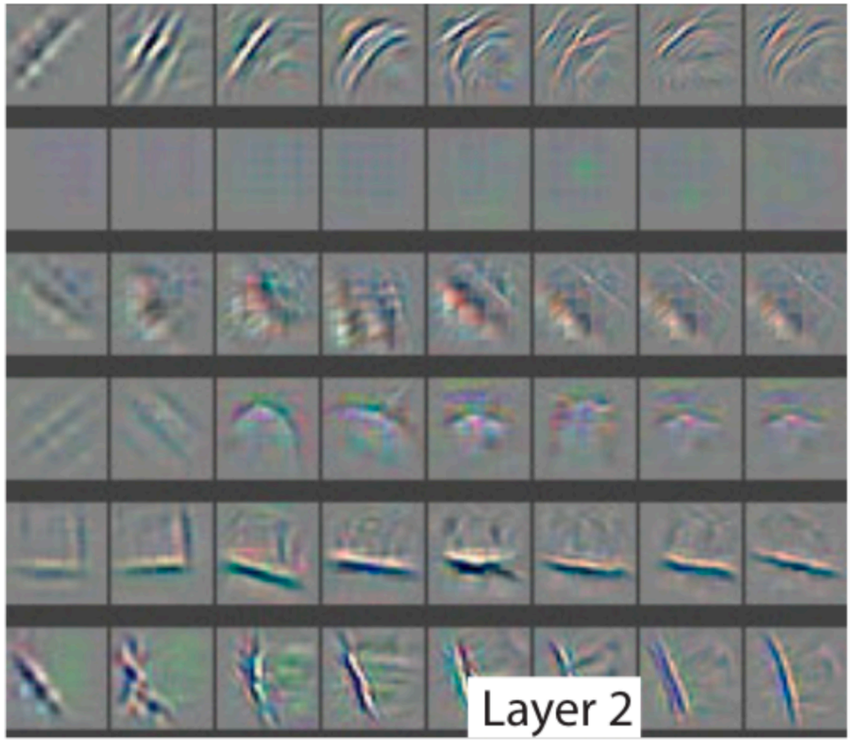
Eyes detector?

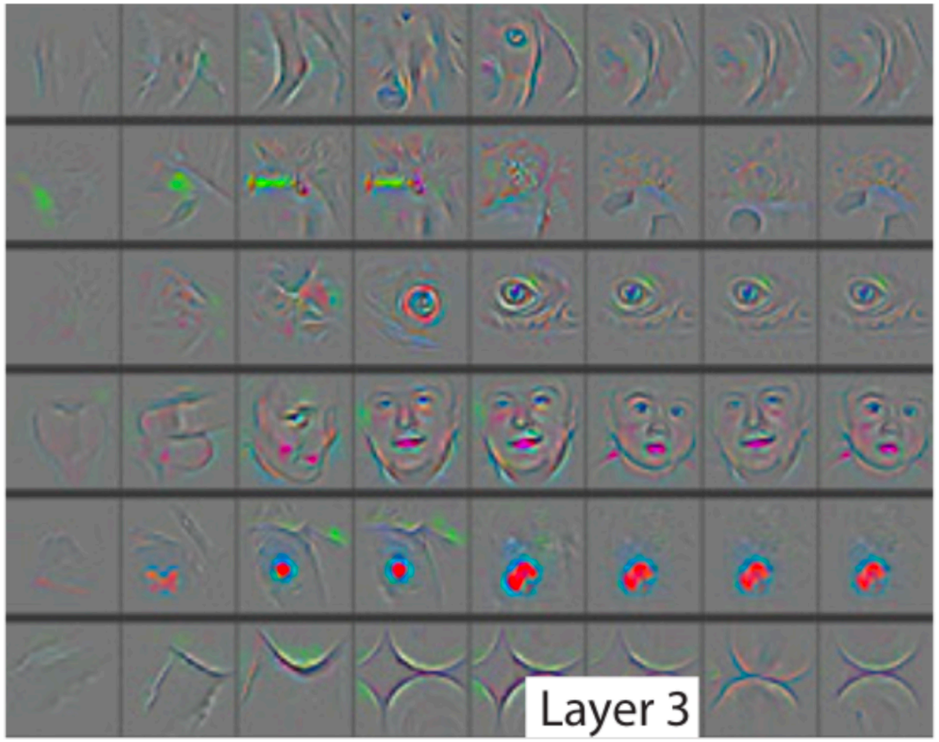


- We know some kernels can emphasize some edges in an image (edge detectors)
- Maybe by training the parameters, we discover kernels that can emphasize interesting aspects of the image?

What do the convolution kernels learn to recognize?







From Zeiler&Fergus “Visualizing and Understanding Convolutional Networks”

About parameters size

- We said that if we have a Fully-Connected Layer with ***n*** neurons and ***m*** input, it contains ***n x (m+1)*** parameters
- How many parameters in a Max-Pooling Layer ?
- How many parameters in a Convolutional Layer?

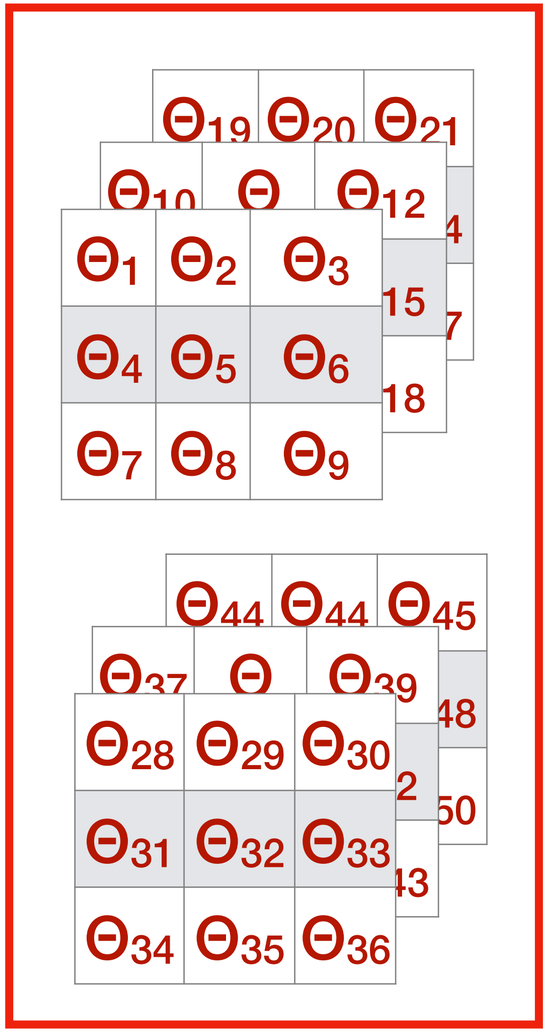
About parameters size

- We said that if we have a Fully-Connected Layer with ***n*** neurons and ***m*** input, it contains ***n x (m+1)*** parameters
- How many parameters in a Max-Pooling Layer ?
  - 0 parameters
- How many parameters in a Convolutional Layer?
  - It depends on the 4D kernel size

About parameters size

- Kernel of size ***k x k*** with ***Ci*** input channels and ***Co*** output channels
- It means we have ***Co*** neurons connected to ***k x k x Ci*** inputs for every location in the image
  - ***(k x k x Ci + 1) x Co*** parameters
- For the kernel on the right: *k=3 Ci=3 Co=2*
  - (3 x 3 x 3 +1) x 2 = 56 parameters

Learnable 4D Kernel



Example: VGG network

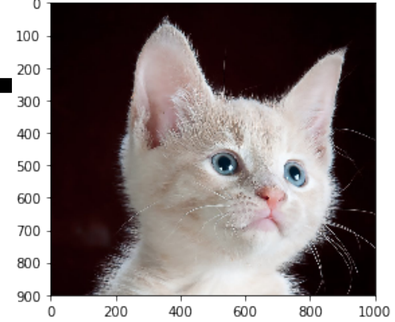


image [224x224x3]

conv-64 [224x224x64]
conv-64 [224x224x64]
maxpool [112x112x64]

conv-128 [112x112x128]
conv-128 [112x112x128]
maxpool [56x56x128]

conv-256 [56x56x256]
conv-256 [56x56x256]
maxpool [28x28x256]

conv-512 [28x28x512]
conv-512 [28x28x512]
maxpool [14x14x512]

conv-512 [14x14x512]
conv-512 [14x14x512]
maxpool [7x7x512]

FC-4096
FC-4096
FC-1000
softmax —> CAT

- "Very Deep Convolutional Networks for Large-Scale Image Recognition" Simonyan and Zisserman, 2015
- Best Image Classifier in 2015
- Today the best models have much more layers (> 100)
- Could you compute the number of parameters in each layer?
- All kernels are of size 3 (k=3)
- Conv-64 mean convolutional layer with 64 output channels
- FC-4096 means Fully Connected Layer with 4096 neurons
- Size of input can be guessed from the size of output of the previous layer

Image from https://cs.stanford.edu/people/karpathy/sfmltalk.pdf

Example: VGG network

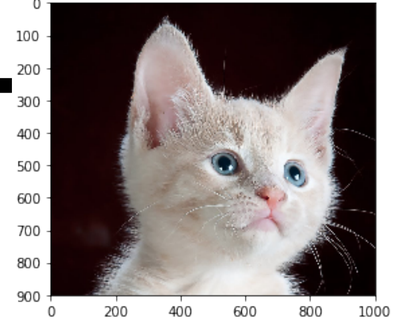


image [224x224x3]

conv-64 [224x224x64]
conv-64 [224x224x64]
maxpool [112x112x64]

conv-128 [112x112x128]
conv-128 [112x112x128]
maxpool [56x56x128]

conv-256 [56x56x256]
conv-256 [56x56x256]
maxpool [28x28x256]

conv-512 [28x28x512]
conv-512 [28x28x512]
maxpool [14x14x512]

conv-512 [14x14x512]
conv-512 [14x14x512]
maxpool [7x7x512]

FC-4096
FC-4096
FC-1000
softmax —> CAT

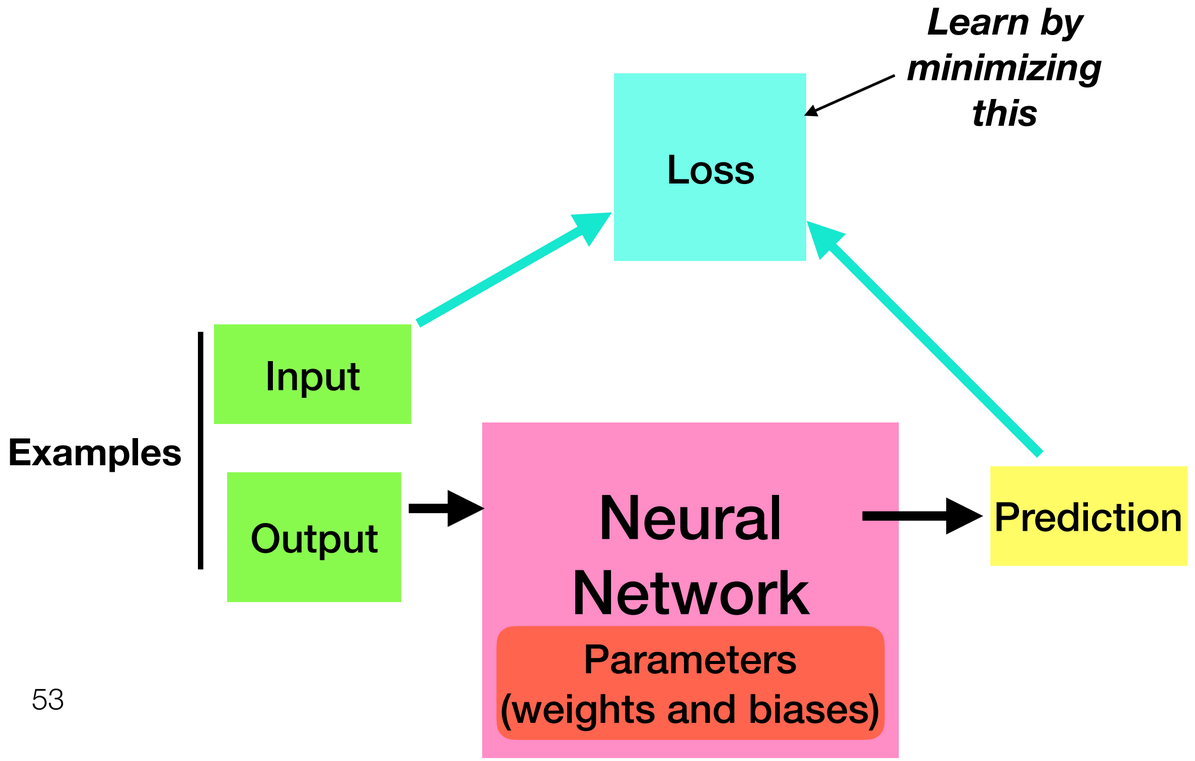
- Number of parameters for each layer:
- Conv-64 (1st): (3x3x3 + 1) x 64 = 1792
- Conv-64 (2nd): (3x3x64 + 1) x 64 = 36 928
- Conv-128 (1st): (3x3x64 + 1) x 128 = 73 856
- Conv-128 (2nd): (3x3x128 + 1) x 128 = 147 584
- Conv-256 (1st): (3x3x128 + 1) x 256 = 295 168
- Conv-256 (2nd): (3x3x256 + 1) x 256 = 590 080
- Conv-512: (3x3x256 + 1)x 512 = 1 180 160
- 3x Conv-512 (2nd step) : (3x3x512 +1)x512 = 2 359 808
- FC-4096 (1st): (7x7x512 + 1) x 4096 = 102 764 544
- FC-4096 (2nd) : (4096 + 1) x4096 = 16 781 312
- FC-1000 : (4096 + 1)x 1000 = 4 097 000
- Total: 1792 + 36 928 + 73 856 + 147 584 + 295 168 + 590 080 + 1 180 160 + 2 359 808 x3 + 102 764 544 + 16 781 312 + 4 097 000 = 133 047 848 parameters

Image from https://cs.stanford.edu/people/karpathy/sfmltalk.pdf

Supervised Learning

How to find the parameters?

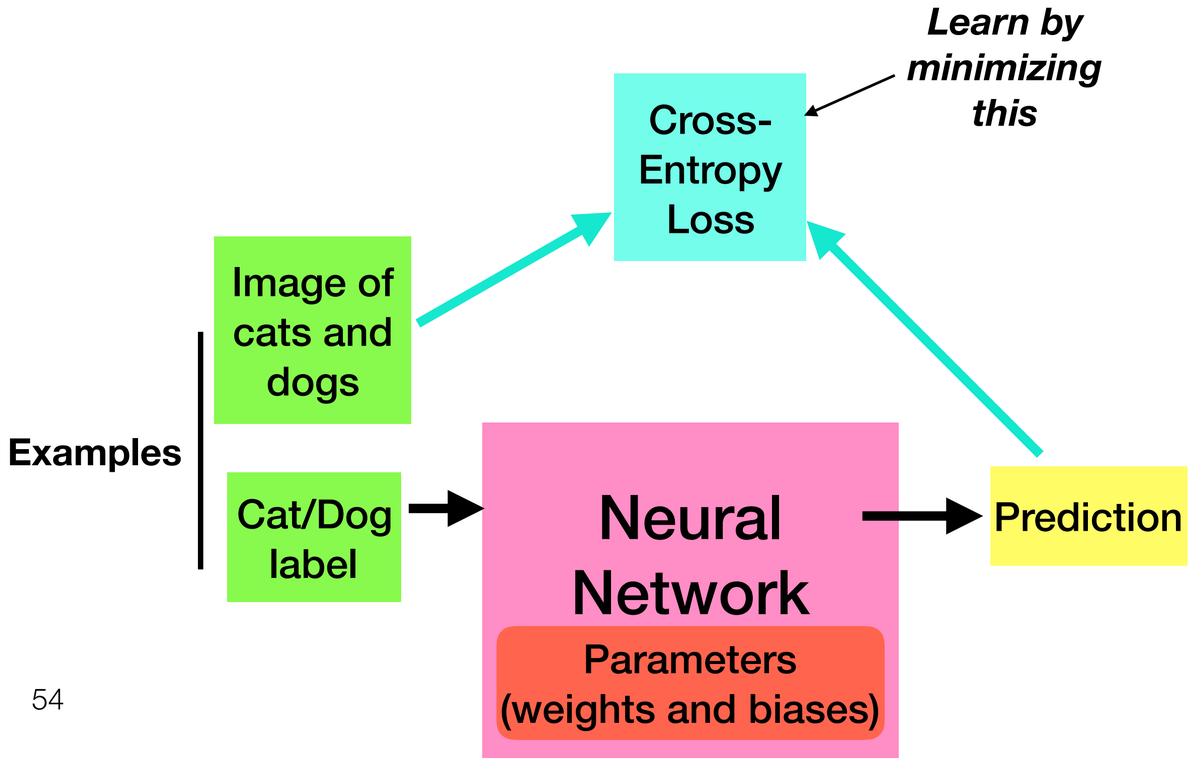
- In supervised learning, we usually have:
  - A **MODEL**: a “parameterized” function that takes input and produce output
  - A ***Loss***: A function that compute how different the model output is from the correct output
  - ***Examples*** of input and correct output (cigarets smoked, age of death)



Supervised Learning

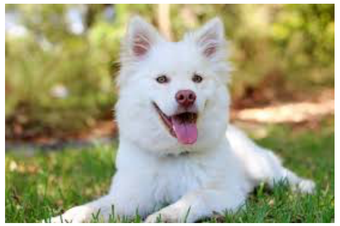
How to find the parameters?

- In supervised learning, we usually have:
  - A **MODEL**: a “parameterized” function that takes input and produce output
  - A ***Loss***: A function that compute how different the model output is from the correct output
  - ***Examples*** of input and correct output (cigarets smoked, age of death)



Actual Training

- Train your network with labeled images and you are good:



Not Cat

Cat

Cat

Not Cat

How to get 5 images for the price of one?

Data augmentation

How to get 5 images for the price of one?

A cat

Data augmentation

How to get 5 images for the price of one?

A cat

Mirroring

Still a cat

Data augmentation

How to get 5 images for the price of one?

A cat

Mirroring

Still a cat

Color change

Still a cat

Data augmentation

How to get 5 images for the price of one?

A cat

Mirroring

Still a cat

Color change

Still a cat

Cropping

Still a cat

Rotation

Still a cat

And any combination of these

Next Time

- That will be all for image recognition
- Next topic we will discuss is Text Processing