Computer Vision

Aziz Temirkhanov

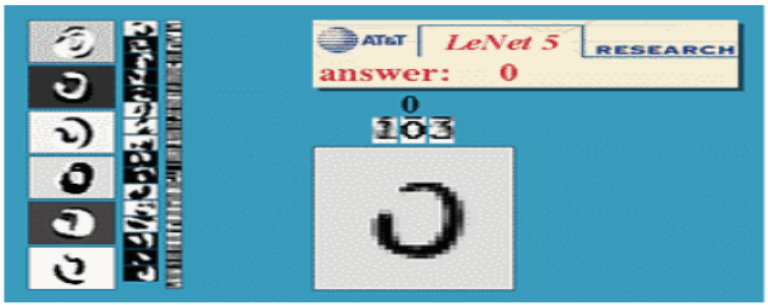
LAMBDA lab
Faculty of Computer Science
Higher School of Economics

February 10, 2025

Outline

Computer Vision

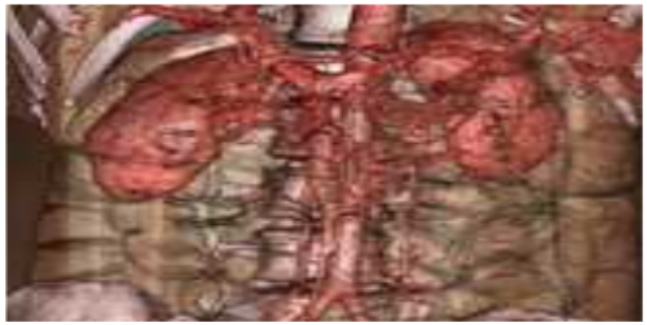
Computer Vision



(a)

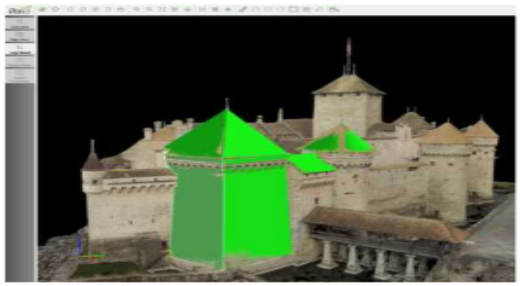
(b)

(c)



(d)

(e)



(f)

Computer Vision

- Automatic number plate recognition (ANPR)
- Retail
- Security
- Warehouse Logistics
- Medical imaging
- Self-driving cars
- ...

Computer Vision

- Image Classification
- Object Detection
- Instance Segmentation
- Semantic Segmentation
- Object counting
- ...

Image Processing

Image Representation

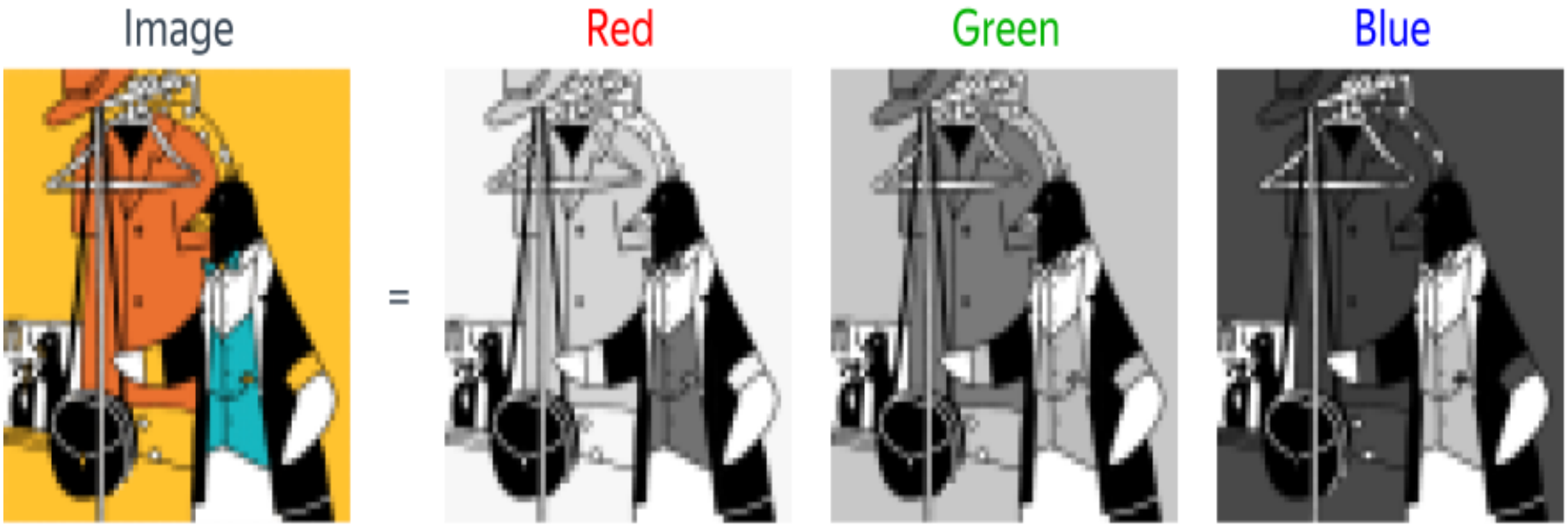


Image Representation

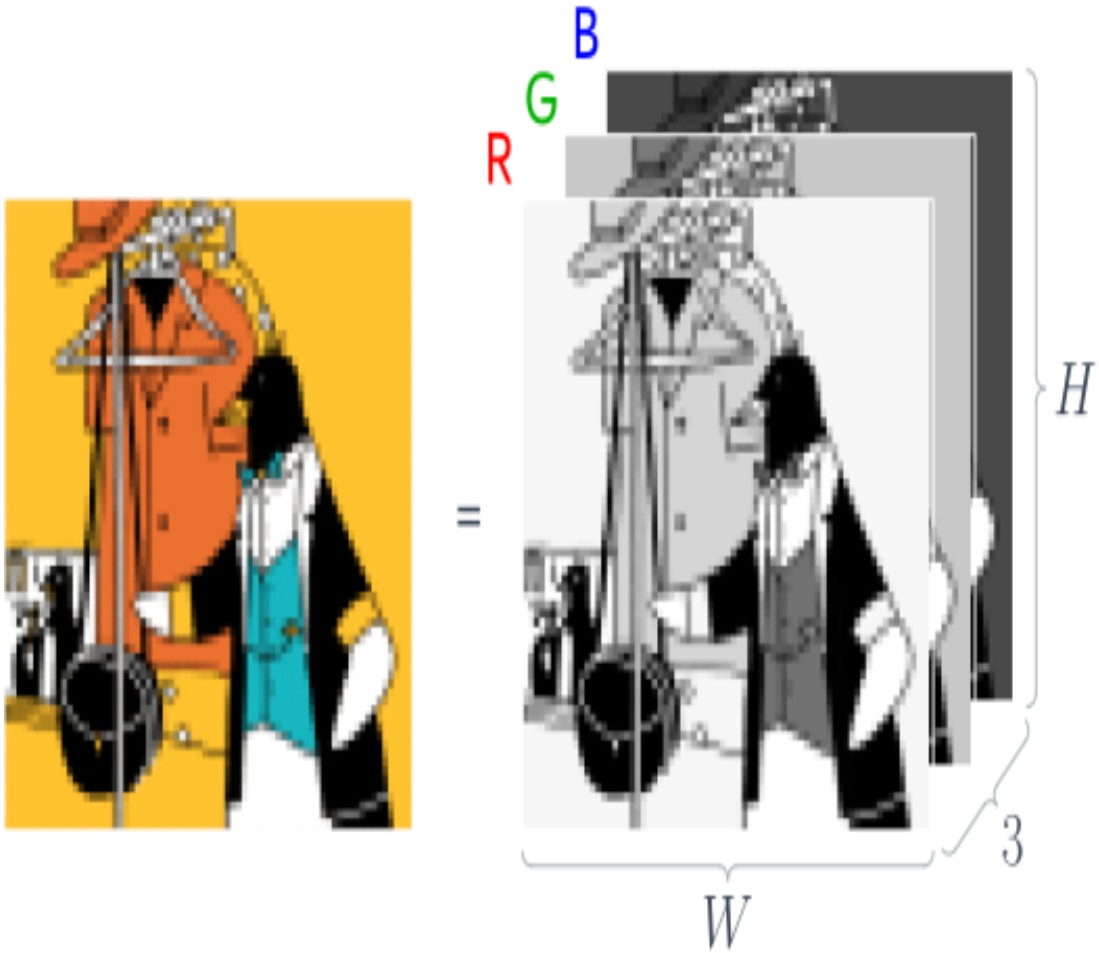


Image Representation

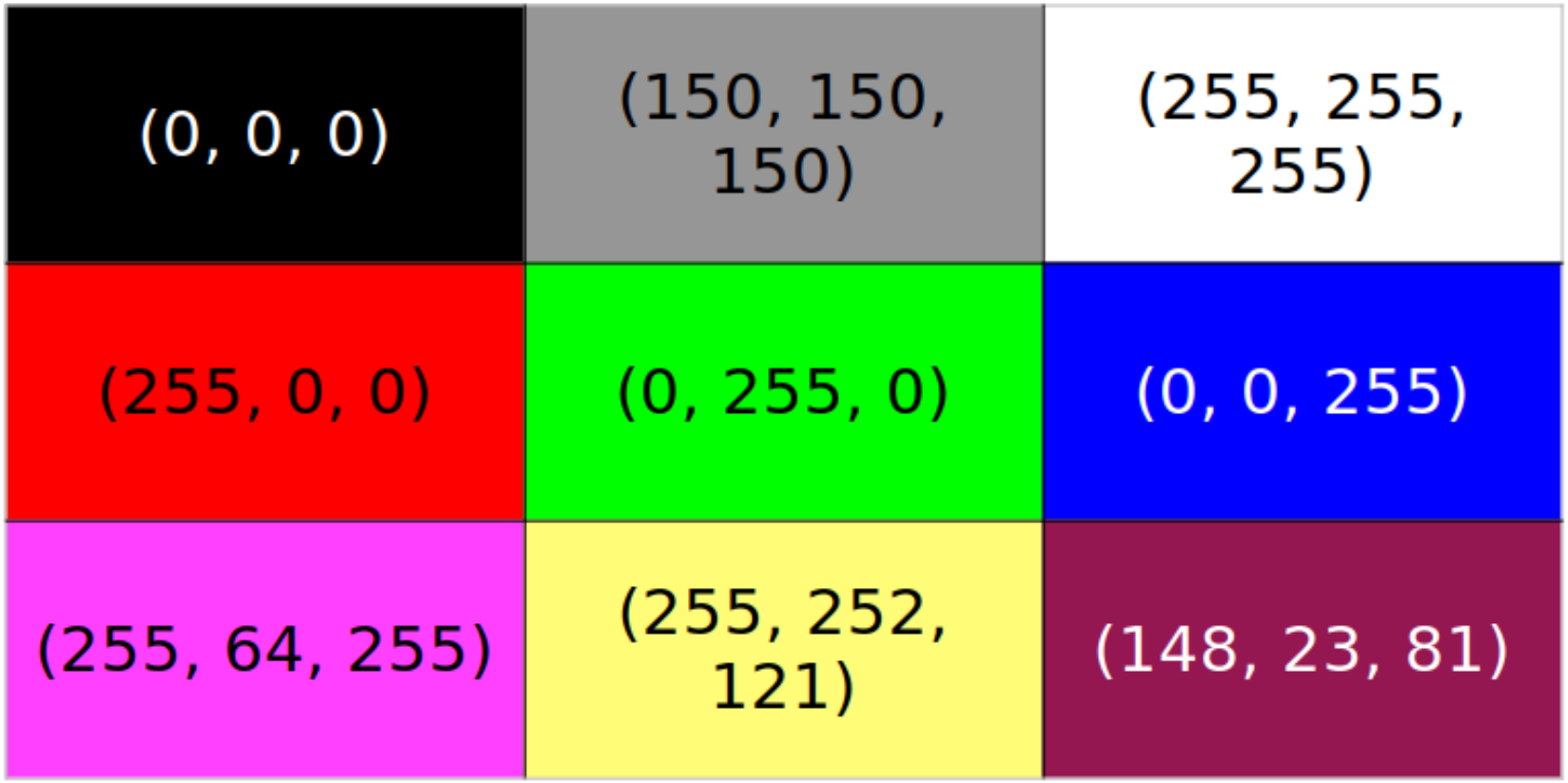
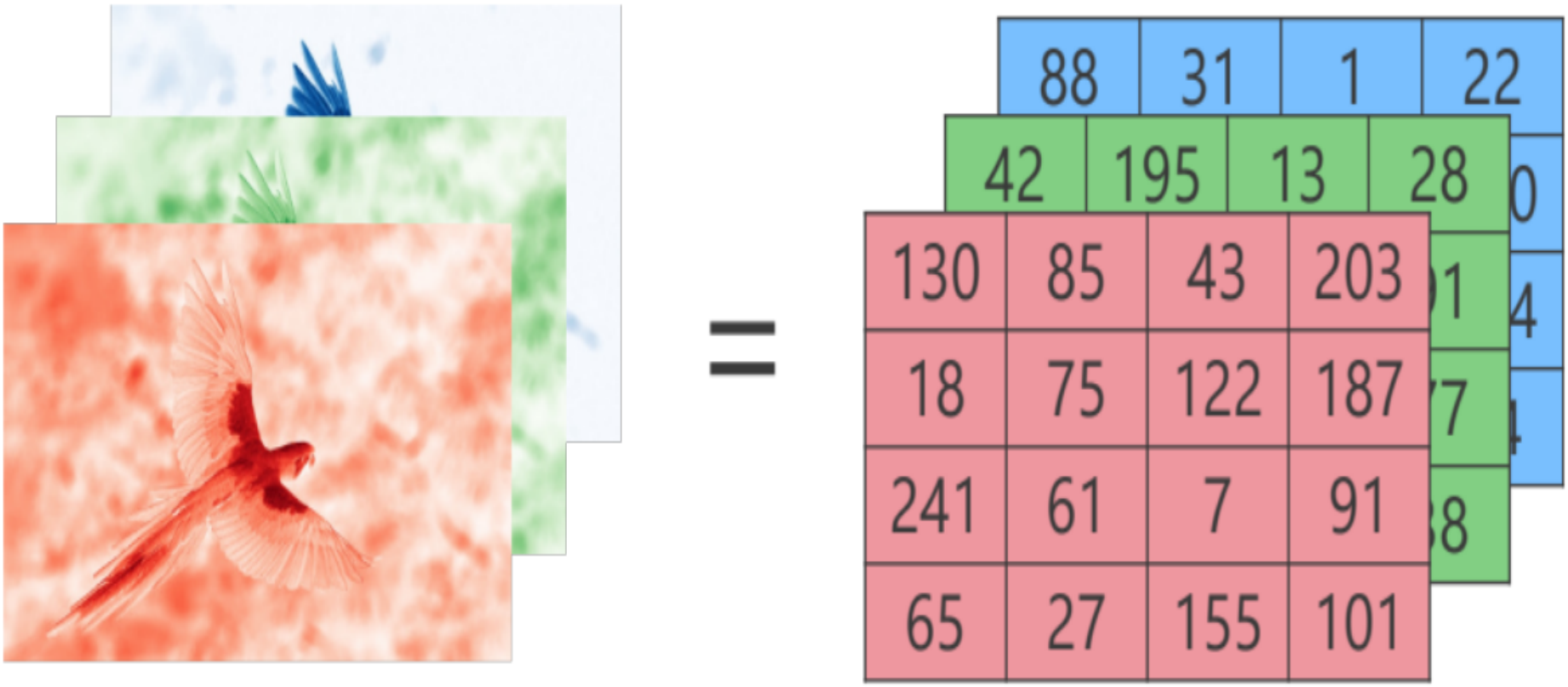


Image Representation

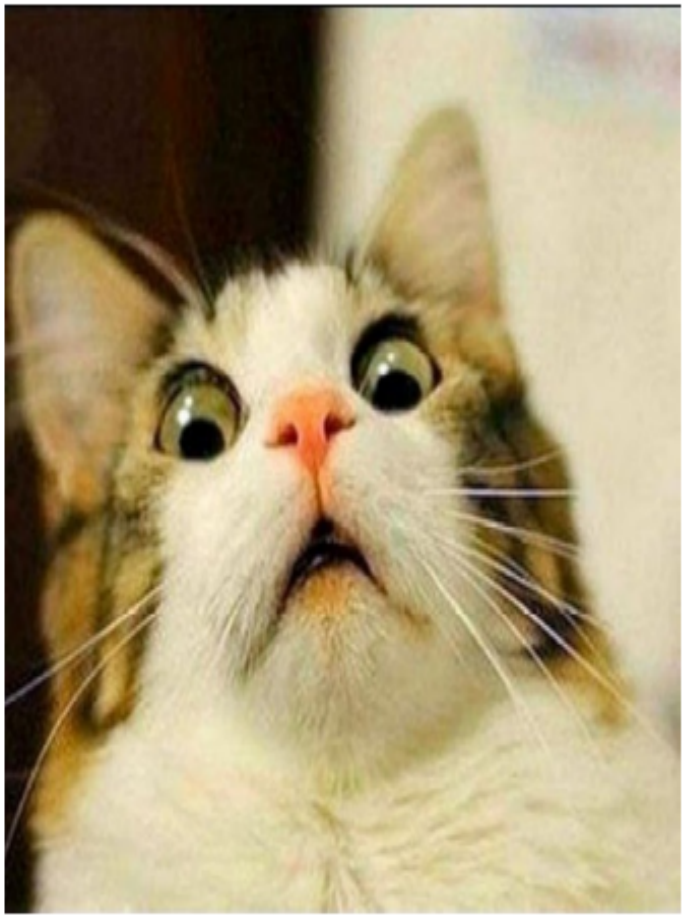
Image Representation



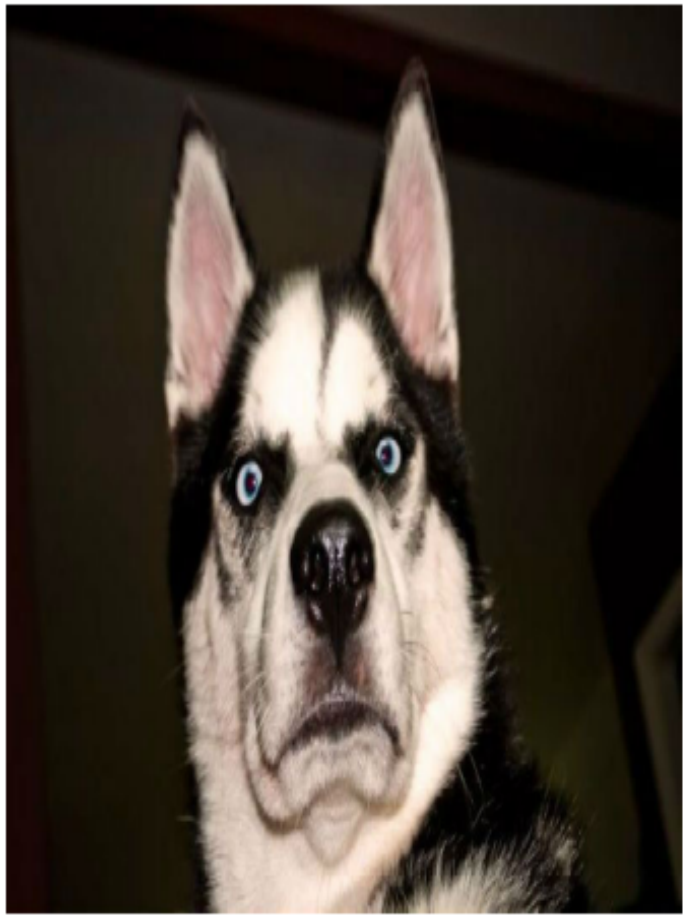
Classification

Classification

- Let's take a look at a simple binary classification example
- How to feed an input to the network?



VS



Classification

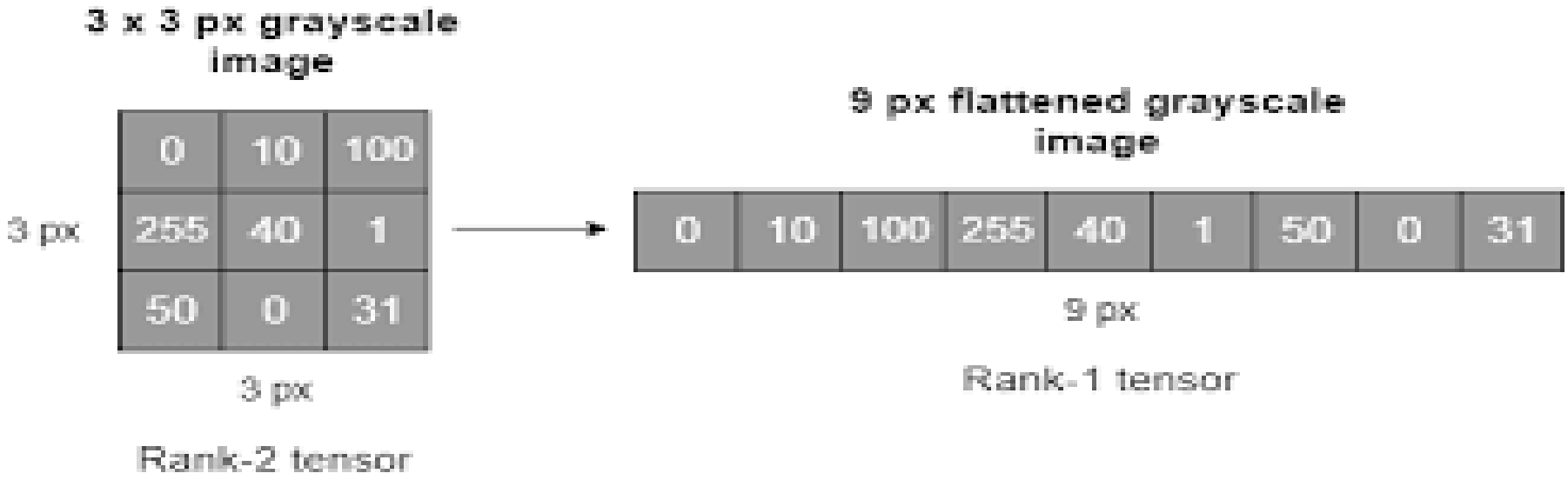
For simplicity, assume we have only one channel (grayscale image)

=

3	3	5	1	2	3
3	4	0	1	4	5
5	1	2	3	2	0
5	0	4	5	2	1
3	4	3	1	0	2
2	0	4	3	5	1

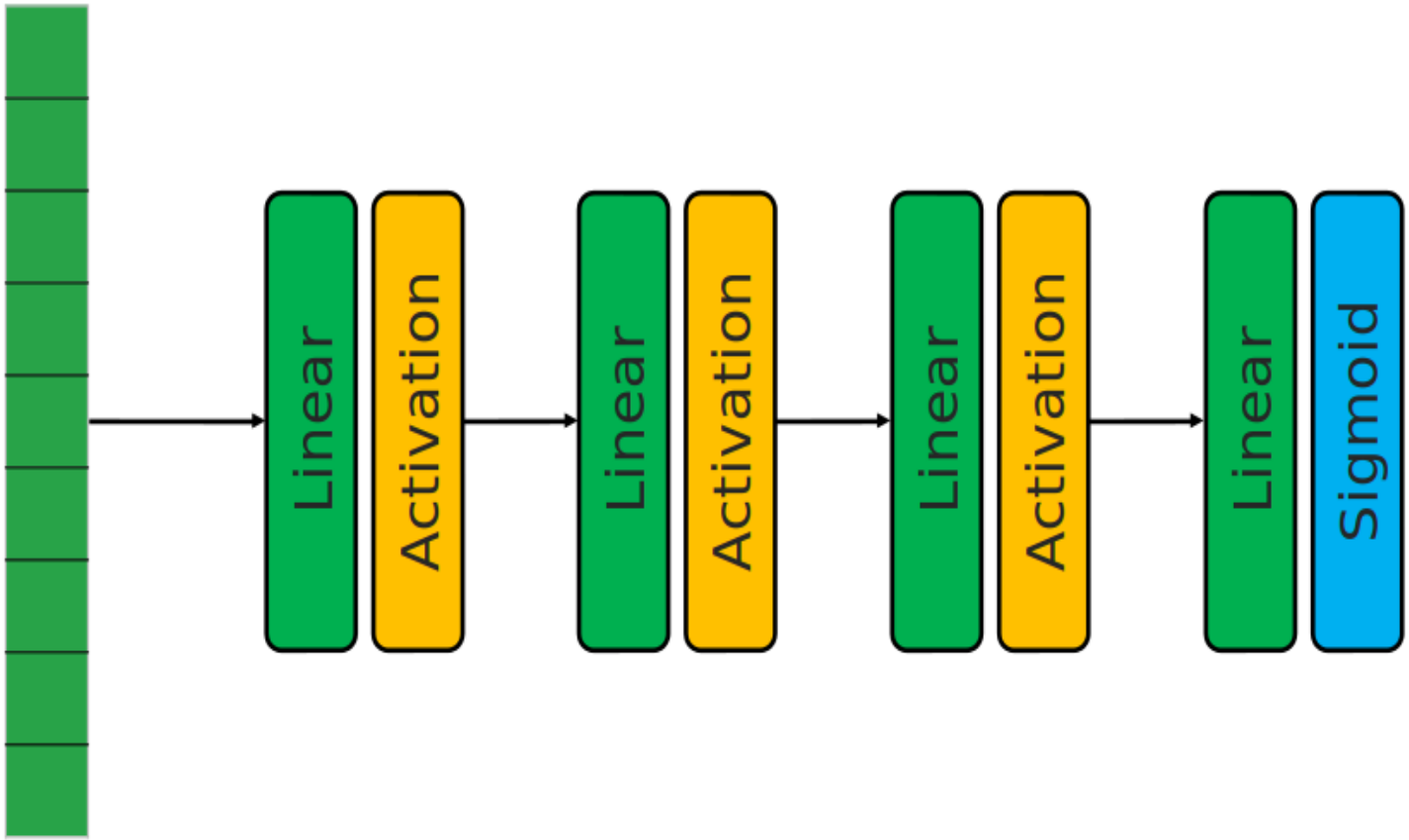
Classification

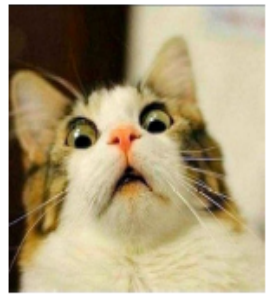
Then, to feed into our FC Neural Network, we have to **flatten** an image:

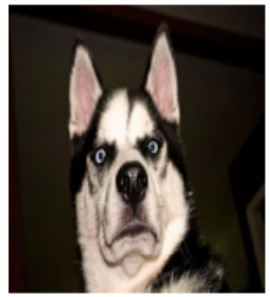


Classification

Then, we can train our NN as usual:







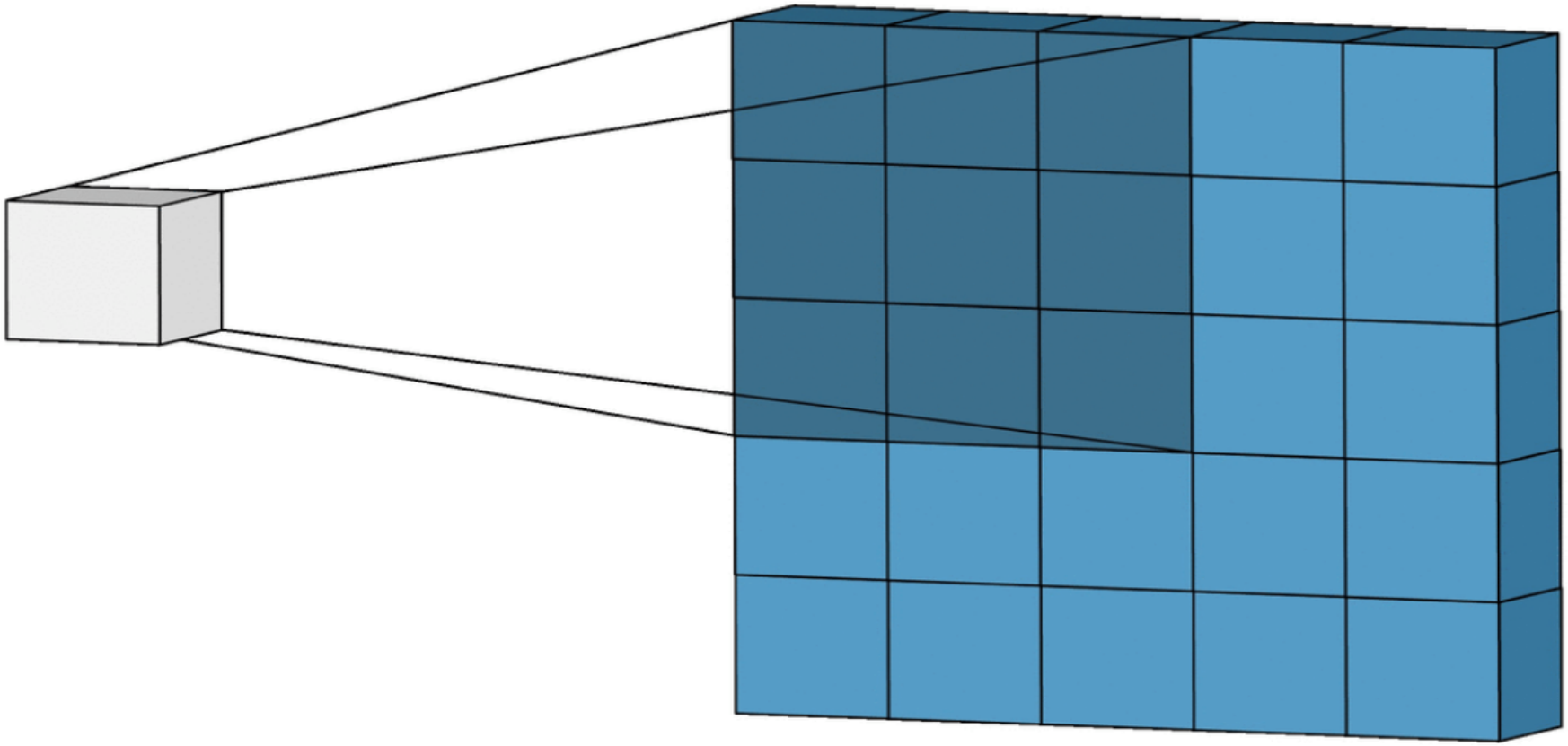
Classification

But this approach surely has some issues. Can you name a few?

CNN

Convolutions

Introduce a **convolution** operation or layer.



Convolutions

Convolution is a subclass of linear filters, that applies some **kernel** or **filter** to an image. Since the size of a kernel is lower than the size of an image, this filtering is applied in some neighborhoods:

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

$f(x,y)$

$*$

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

$h(x,y)$

$=$

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

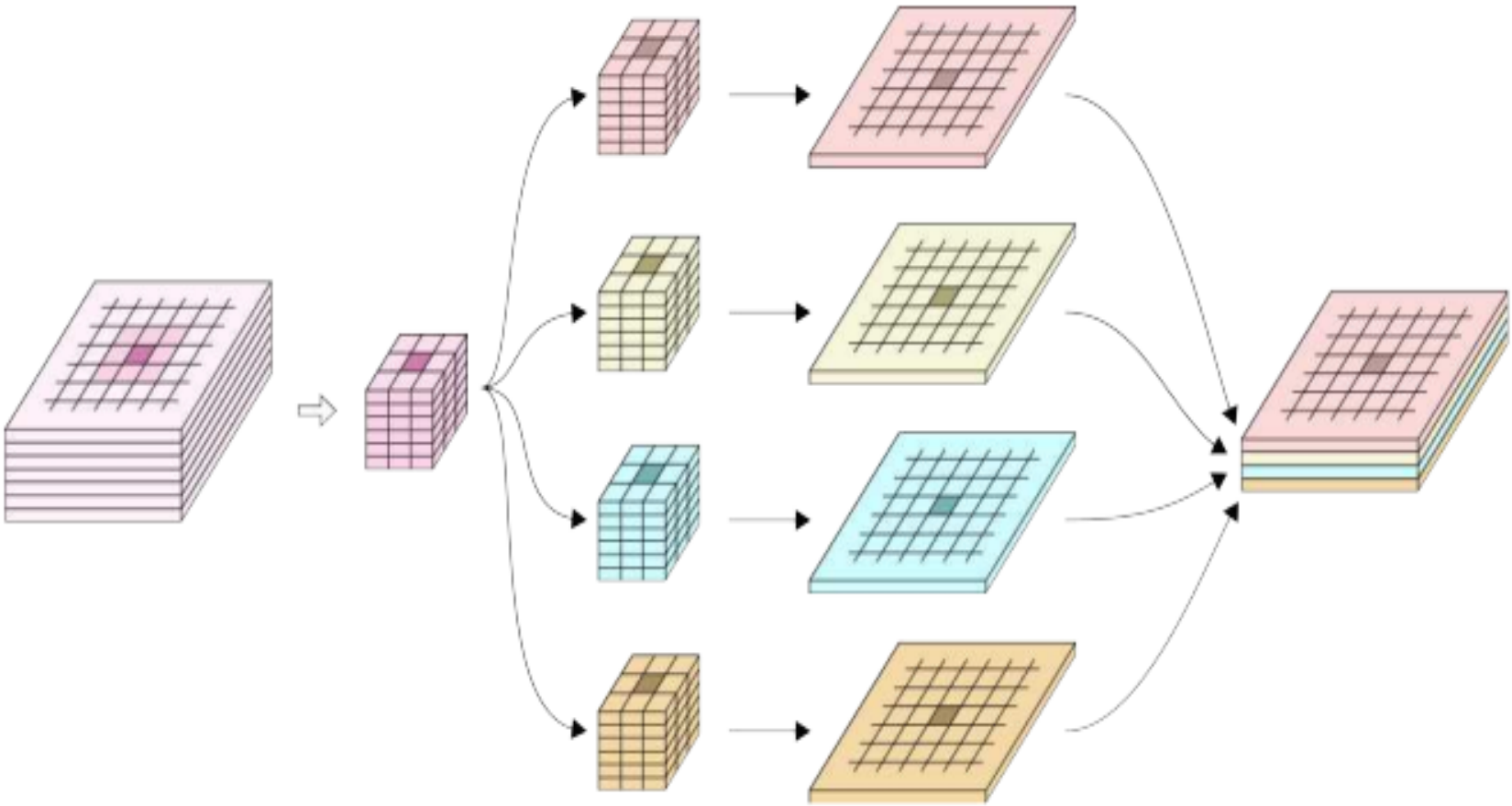
$g(x,y)$

Convolutions

In Deep Learning, this type of layer is applied consequently to a whole image, while this kernel is moving (**sliding**) across the image step-by-step. It usually reduces the image's height and width

Convolutions

2D convolution with multiple input and output channels:

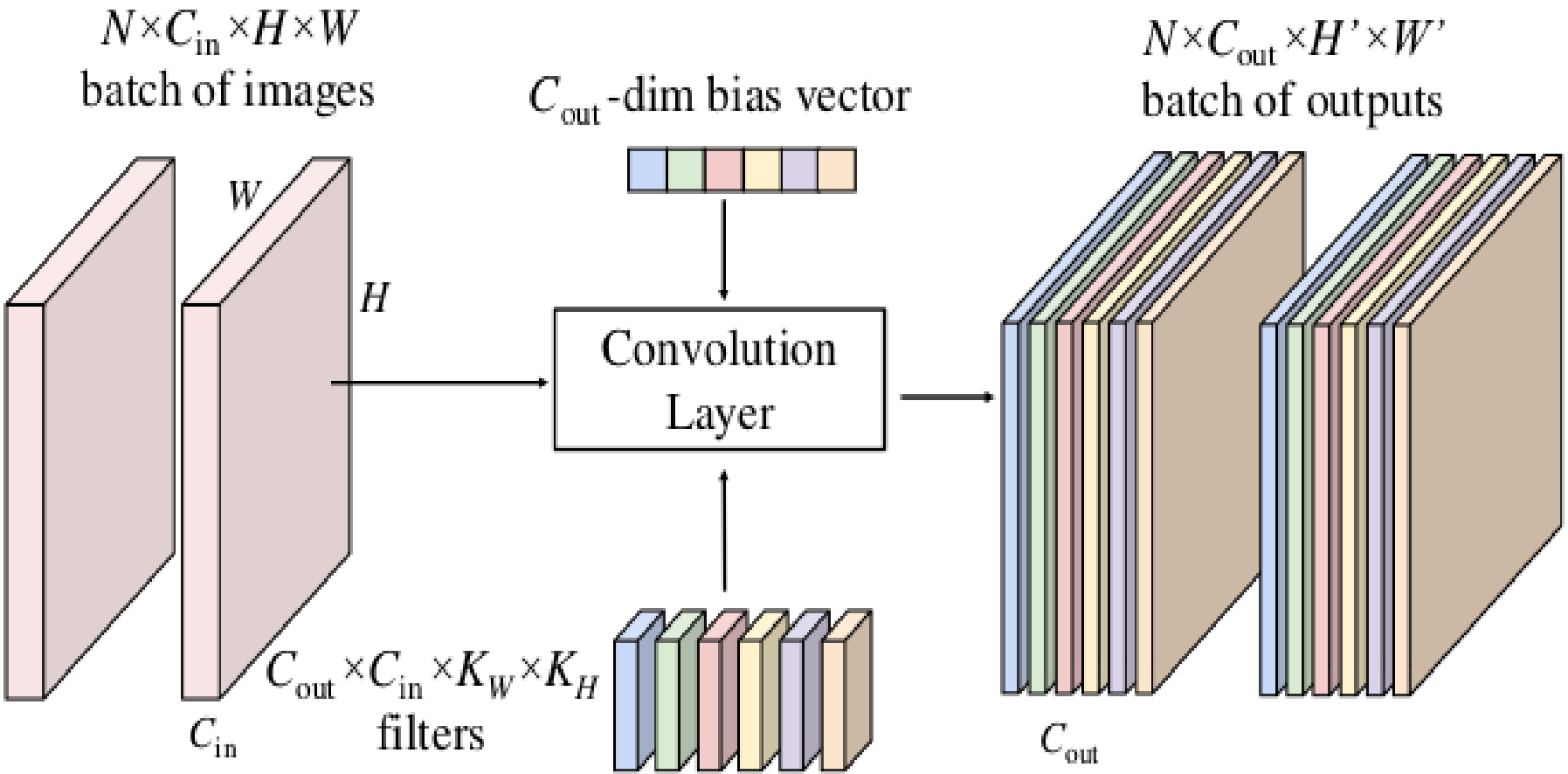


Convolutions

Each 2D convolution kernel takes as input all of the C_1 channels in the preceding layer, windowed to a small area, and produces the values (after the activation function non-linearity) in one of the C_2 channels in the next layer. We have $S^2 \times C_1$ kernel weights for each output channel, so the total number of learnable parameters in each convolutional layer is $S^2 C_1 C_2$. In this figure, we have $C_1 = 6$ input channels and $C_2 = 4$ output channels, with an $S = 3$ convolution window, for a total of $9 \times 6 \times 4$ learnable weights, shown in the middle column of the figure. Since the convolution is applied at each of the $W \times H$ pixels in a given layer, the amount of computation (multiply-adds) in each forward and backward pass over one sample in a given layer is $WHS^2 C_1 C_2$.

Convolutions

2D convolution with multiple batches, input, and output channels:



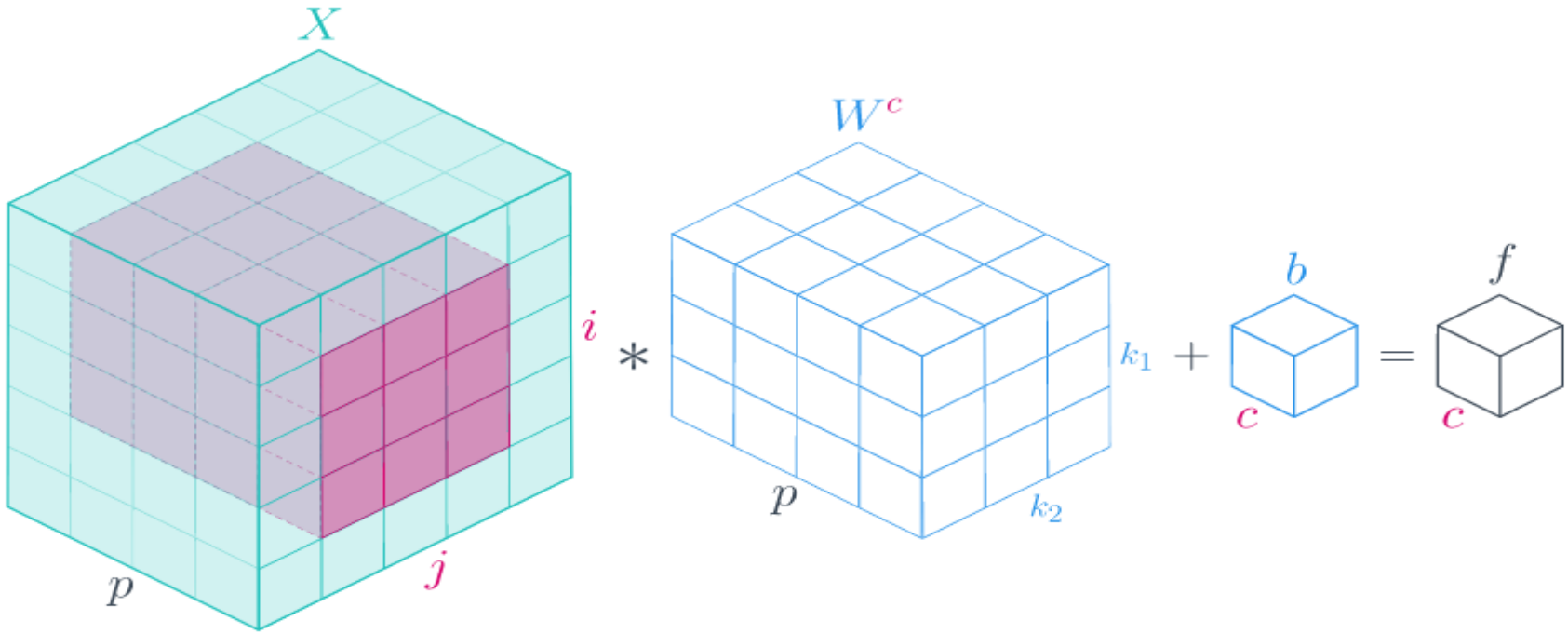
Convolutions

When doing mini-batch gradient descent, a whole batch of training images or features is passed into a convolutional layer, which takes as input all of the C_{in} channels in the preceding layer, windowed to a small area. It produces the values (after the activation function non-linearity) in one of the C_{out} channels in the next layer. As before, for each output channel, we have $K_w \times K_h \times C_{in}$ kernel weights, so the total number of learnable parameters in each convolutional layer is $K_w \times K_h \times C_{in} \times C_{out}$. In this figure, we have $C_{in} = 3$ input channels and $C_{out} = 6$ output channels.

Convolutions

Let's formally define convolution operation. Given:

$$k - \text{kernel size}; X \in \mathbb{R}^{H \times W \times C_{in}}, f \in \mathbb{R}^{H \times W \times C_{out}}$$



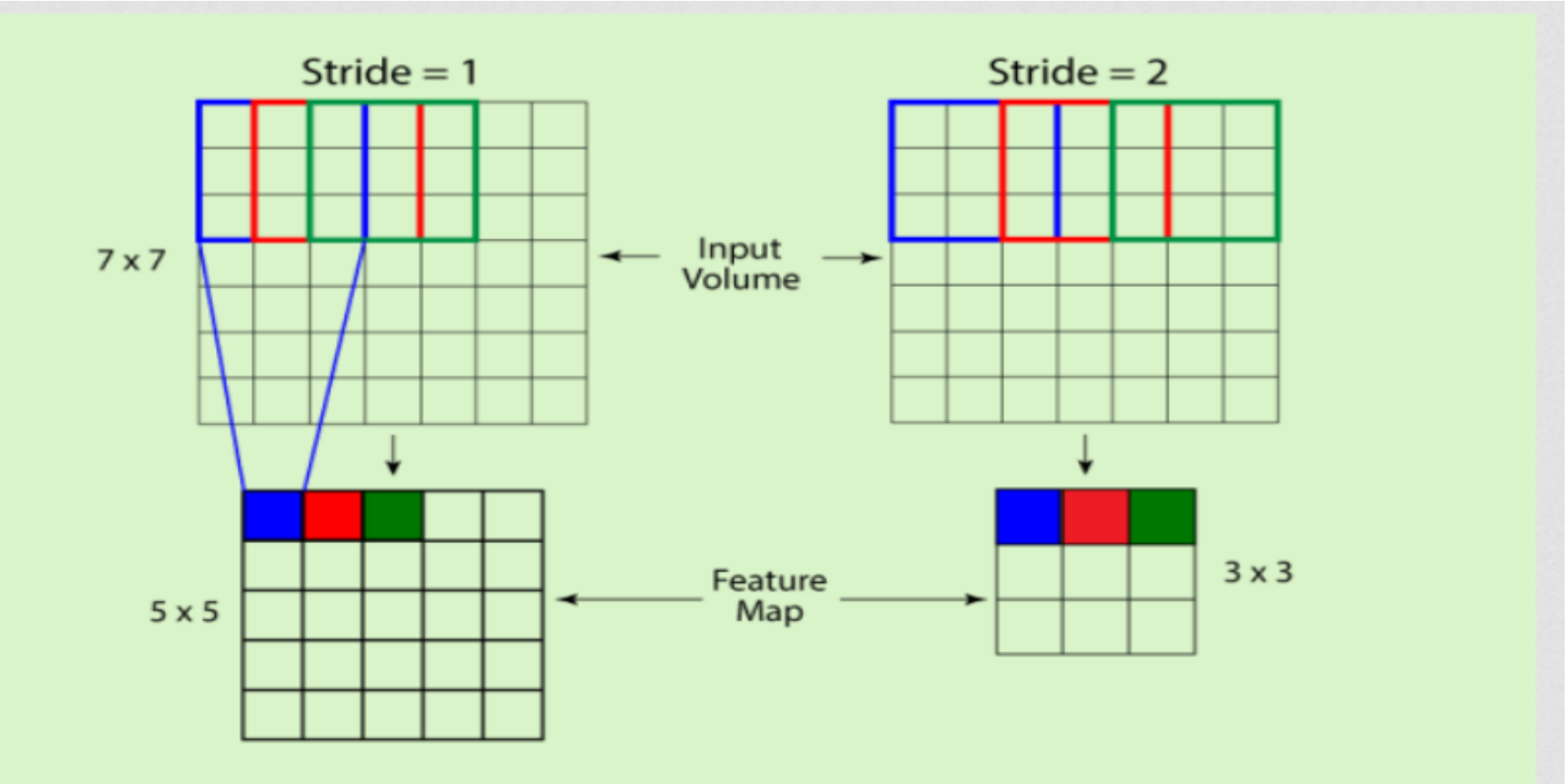
$$f_{ijc} = \sum_{p=1}^{C_{in}} \sum_{k_1=-\lfloor \frac{k}{2} \rfloor}^{\frac{k}{2}} \sum_{k_2=-\lfloor \frac{k}{2} \rfloor}^{\frac{k}{2}} W^c_{\lfloor \frac{k}{2} \rfloor + 1 + k_1, \lfloor \frac{k}{2} \rfloor + 1 + k_2, p} X_{i+k_1, j+k_2, p} + b_c$$

Hyperparams

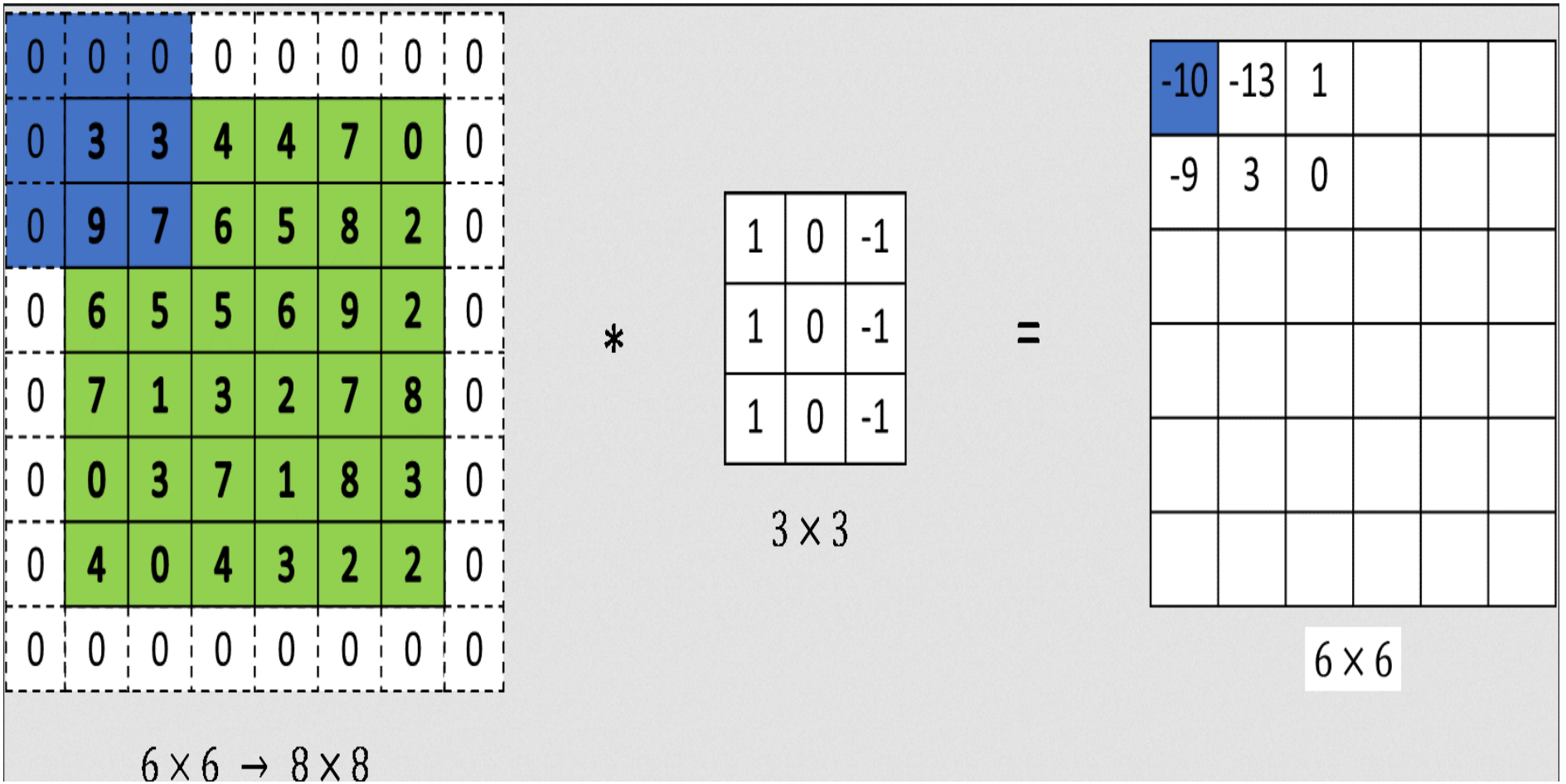
The convolution layer is a special type of linear transform, so activation layers are still required.
A convolution layer contains a lot of trainable parameters, defined by the layer's hyperparameters. Those hp on the other hand define a set of filters that layer learns throughout the training process.
Convolution hyperparameters:

- Kernel size (k)
- Stride (s)
- Padding (p)
- Dilation
- Convolution H and W

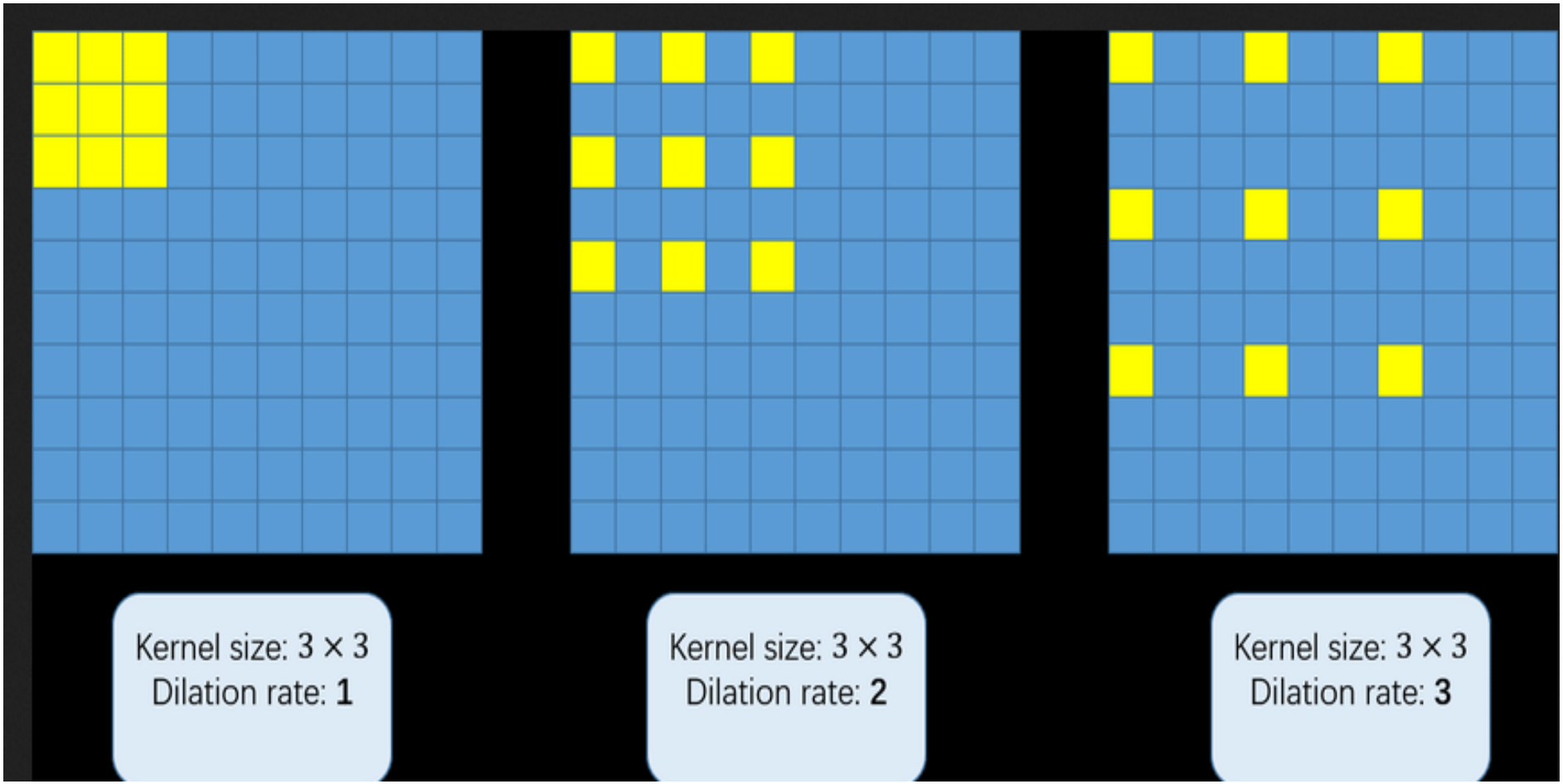
Stride



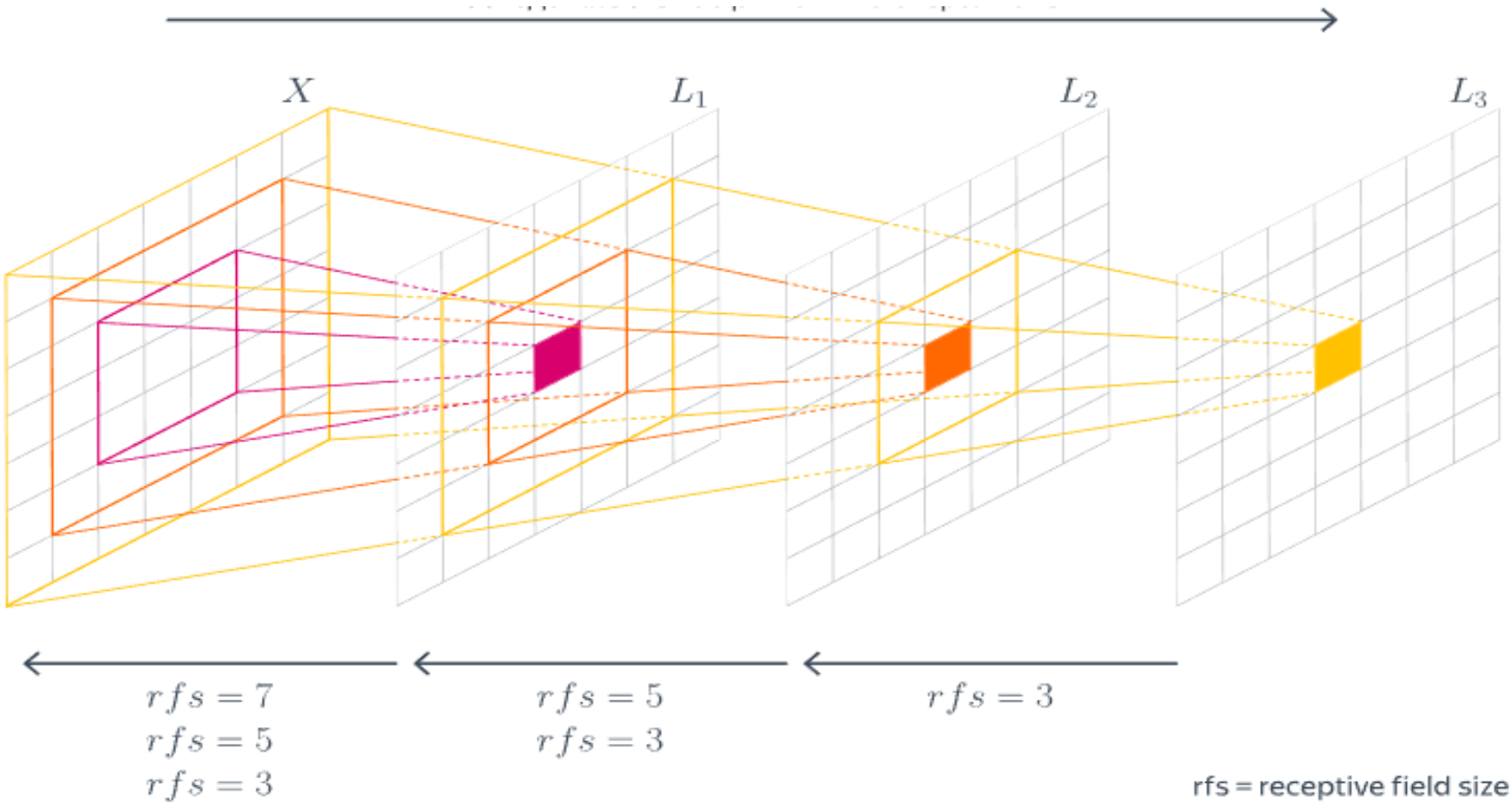
Padding


0 0 0 0 0 0 0 0
0 3 3 4 4 7 0 0
0 9 7 6 5 8 2 0
0 6 5 5 6 9 2 0
0 7 1 3 2 7 8 0
0 0 3 7 1 8 3 0
0 4 0 4 3 2 2 0
0 0 0 0 0 0 0 0
6 × 6 → 8 × 8
*
1 0 -1
1 0 -1
1 0 -1
3 × 3
=
-10 -13 1
-9 3 0
6 × 6

Dilation

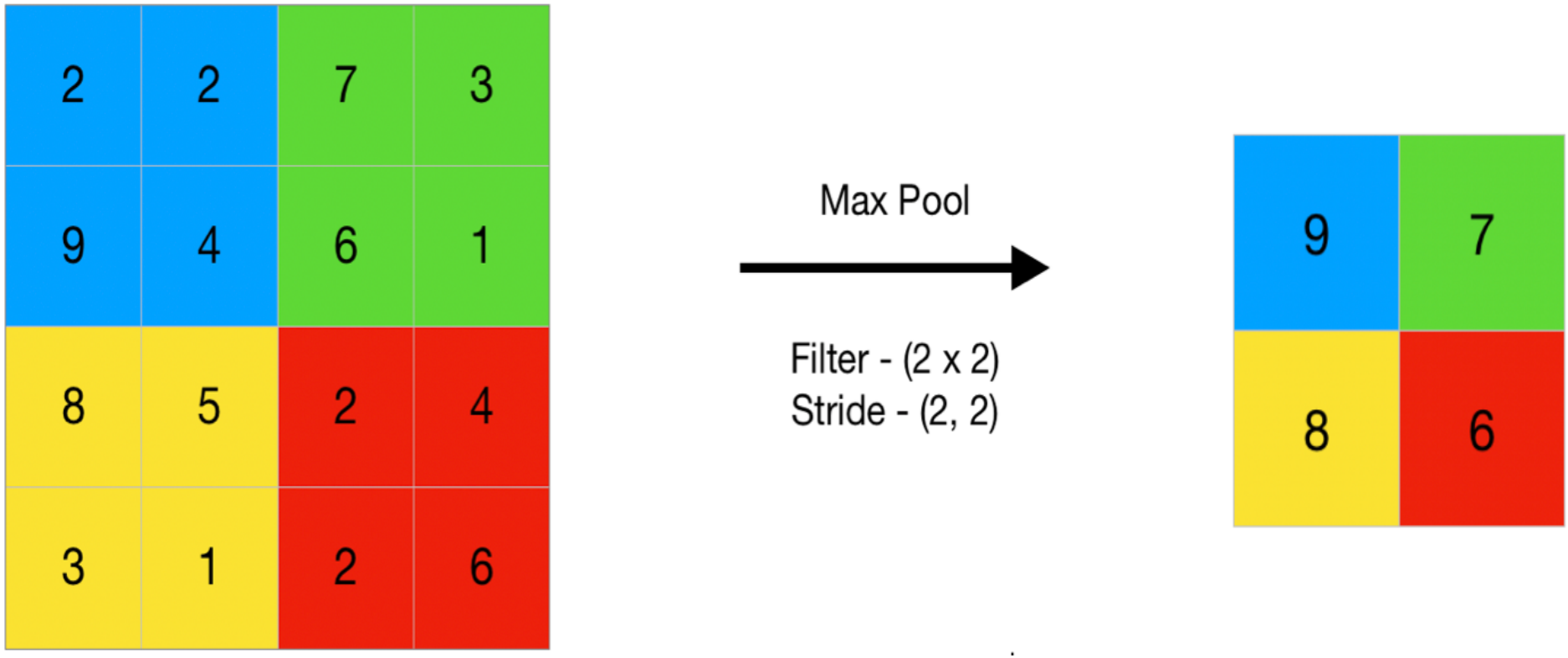

Kernel size: 3 × 3
Dilation rate: 1
Kernel size: 3 × 3
Dilation rate: 2
Kernel size: 3 × 3
Dilation rate: 3

Receptive Field

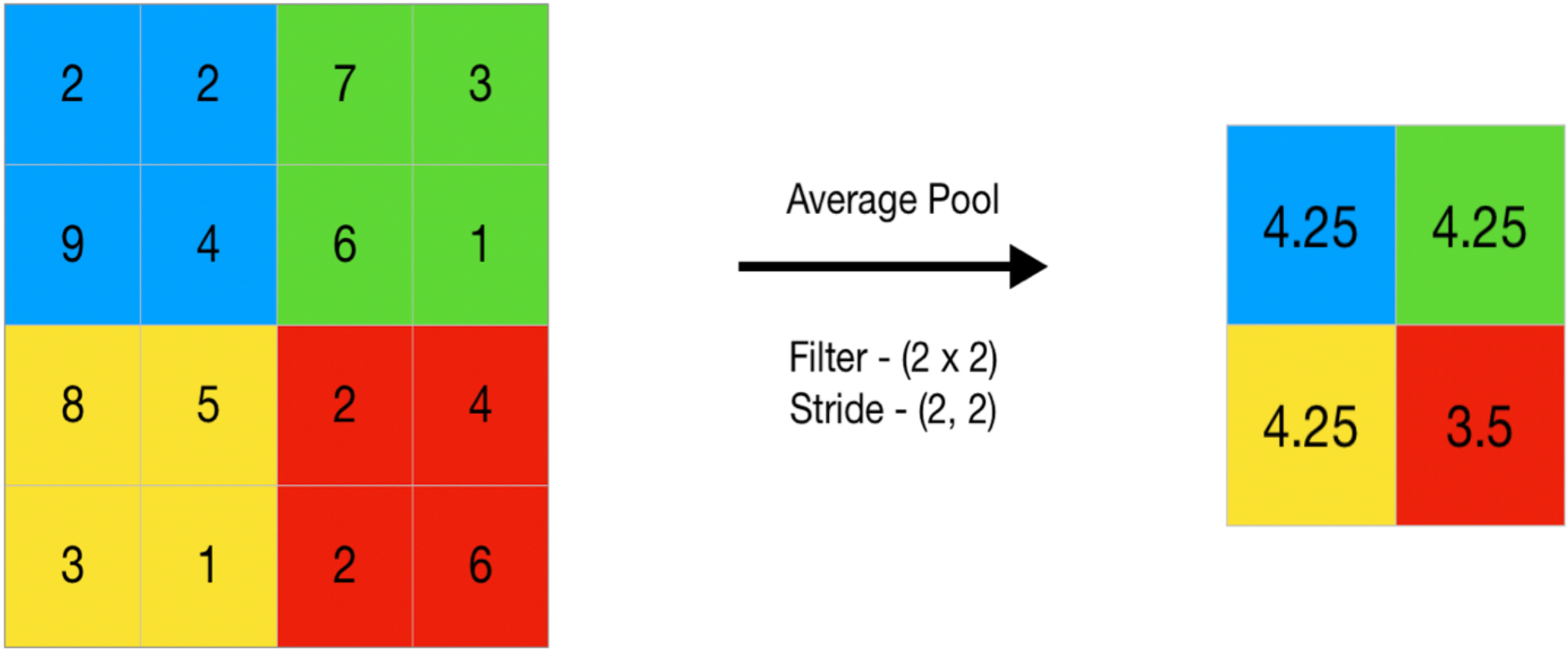


Pooling

Max Pooling



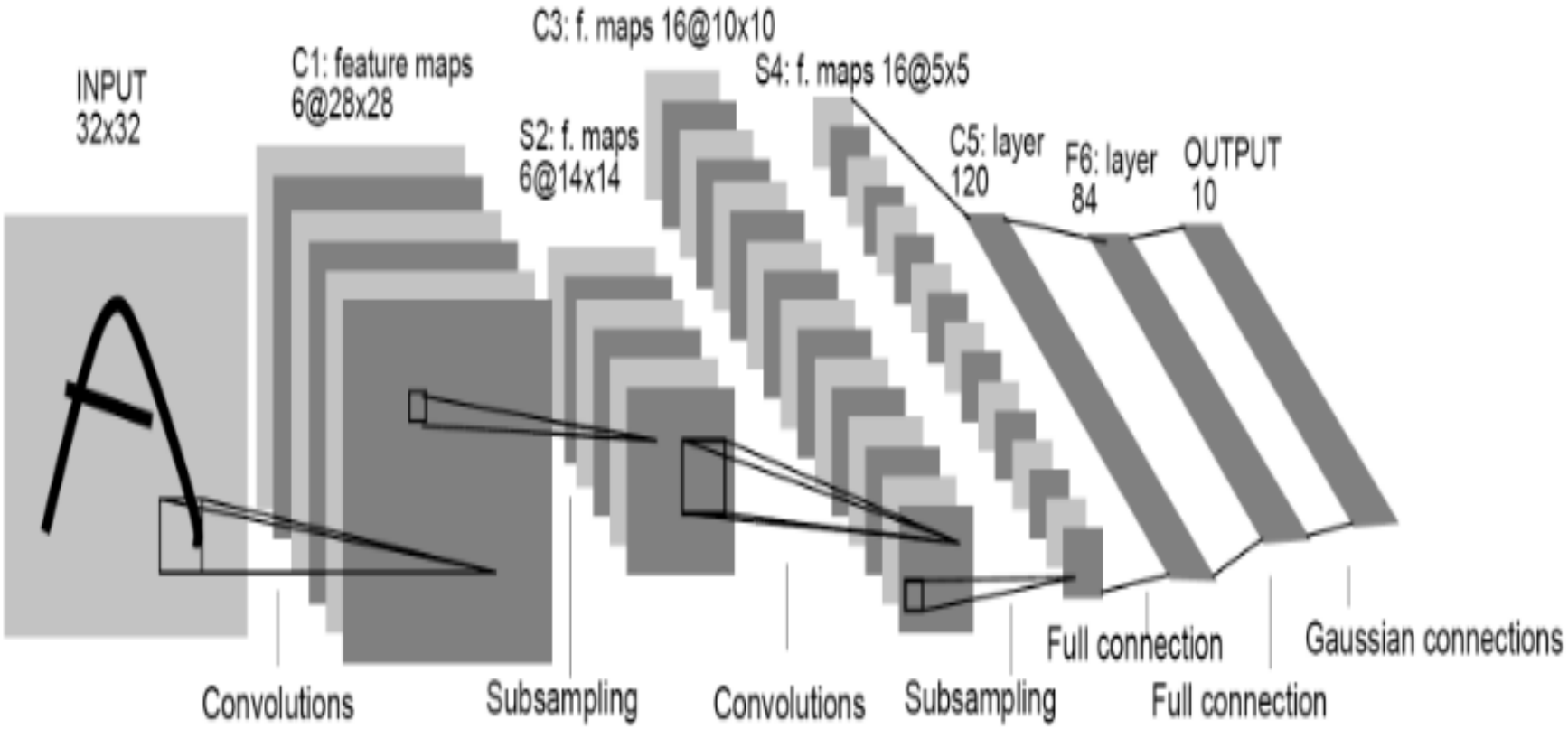
Average Pooling



Architecture topology

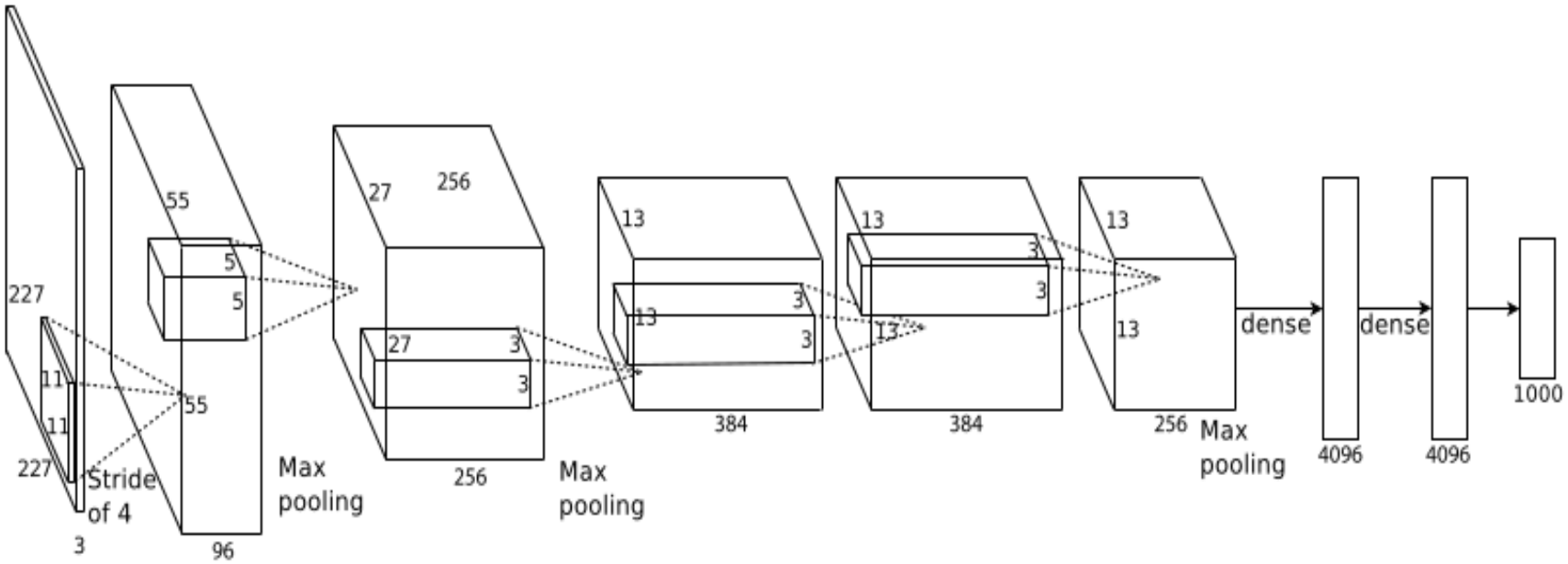
LeNet

First CNN:



AlexNet

Second CNN:



Has ReLU activations after every convolution layer, max pooling operators and two dense layers.


Thank
you