Computer Vision

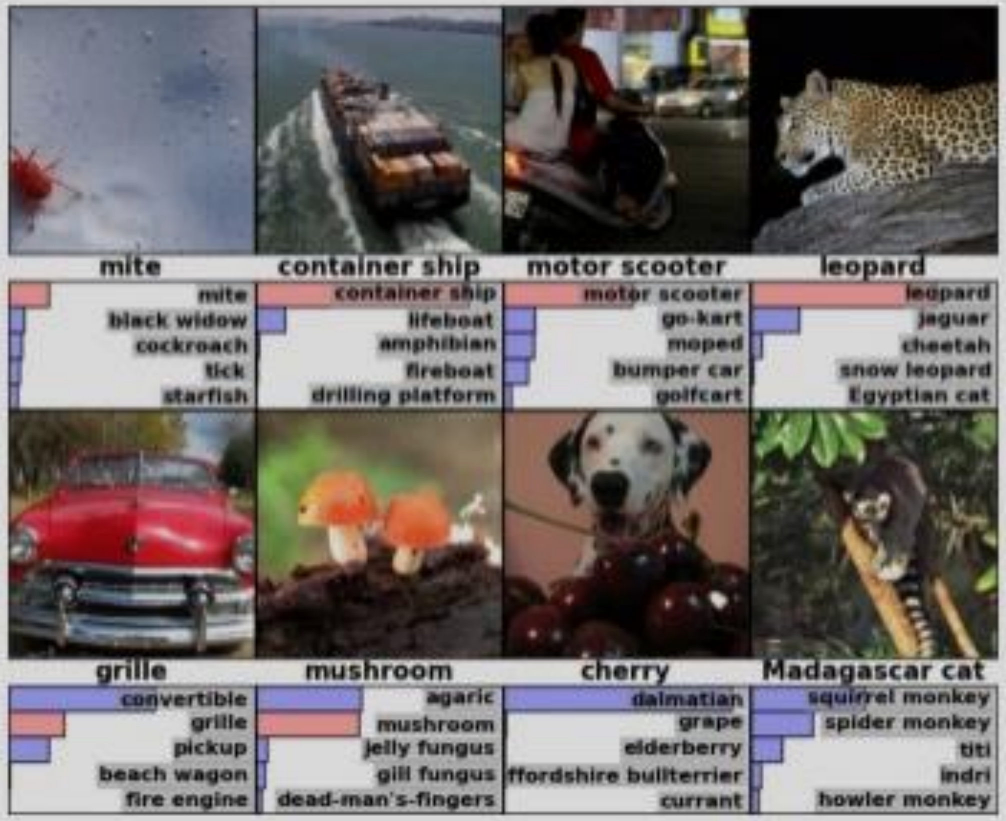
Lecture 02: Review on deep learning

ImageNet challenge

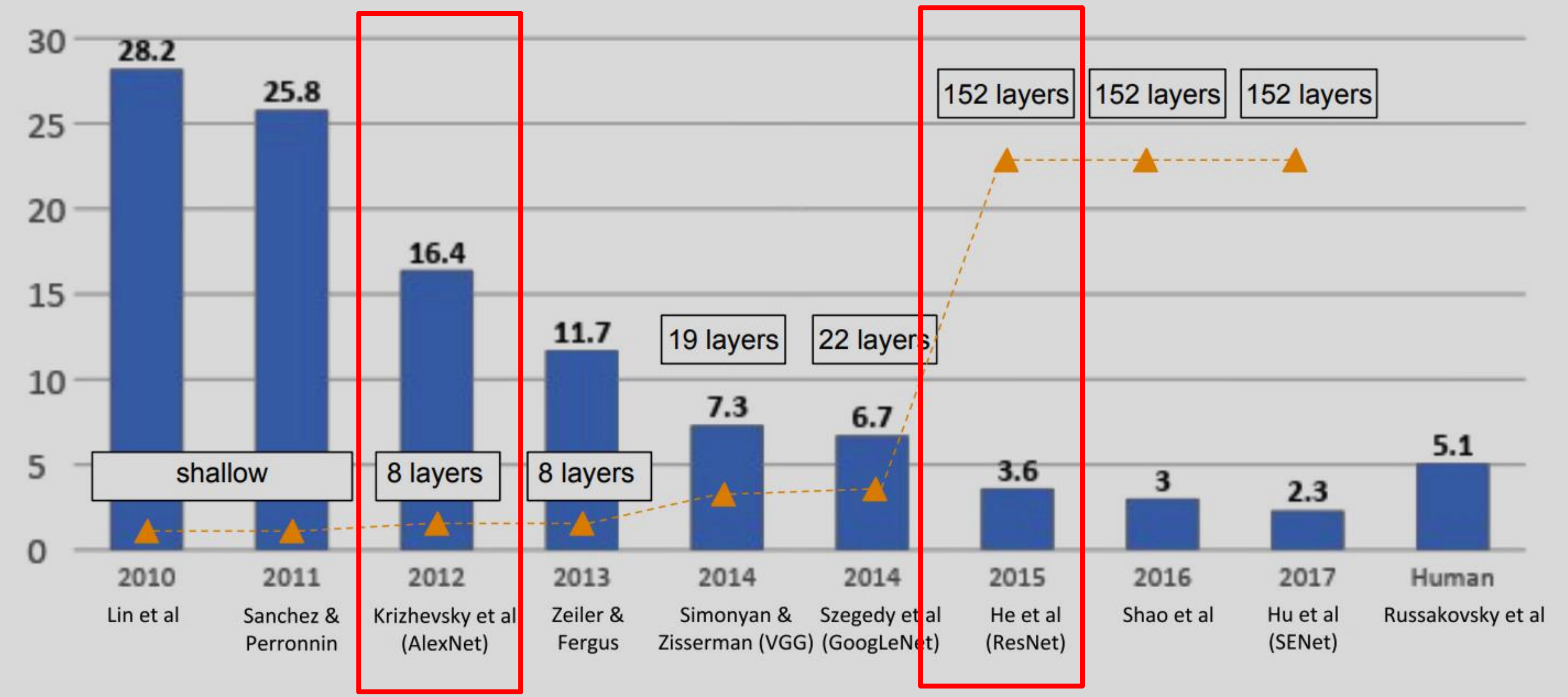
ImageNet Challenge

IMAGENET

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.

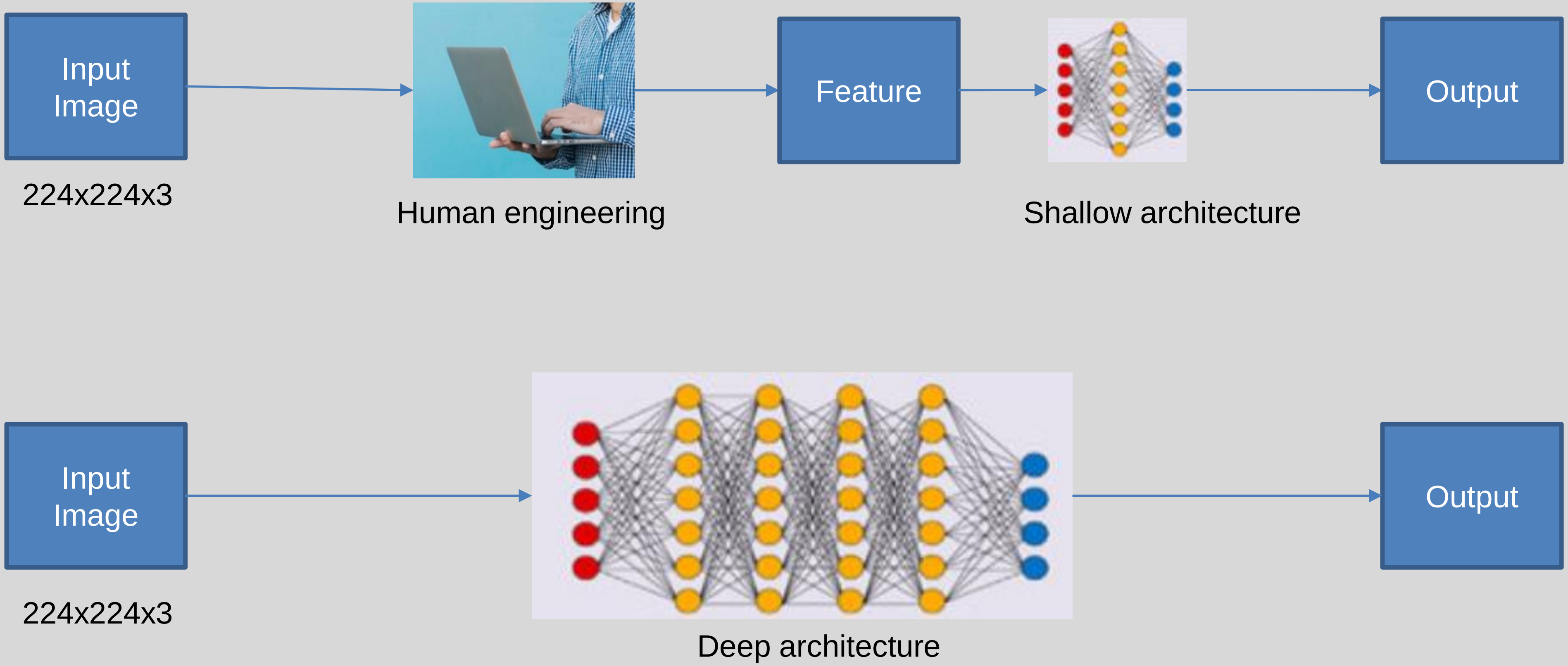


Deep learning



Year 2012: Deep learning achieved the best performance on image classification task.
Year 2015: Surpasses the human performance.

Deep vs. Machine learning



Deep vs. Machine learning

- **Extract optimal representation via End-to-End learning**
  - For the machine learning, we need to design dedicated representation by ourselves for each task.
  - Deep learning learns intermediate representation automatically for different tasks.

- **Non-linearity**
  - Deep learning delivers the capability to achieve the non-linear mappings.
  - Most computer vision applications involve data which requires non-linear mappings.

CNNs

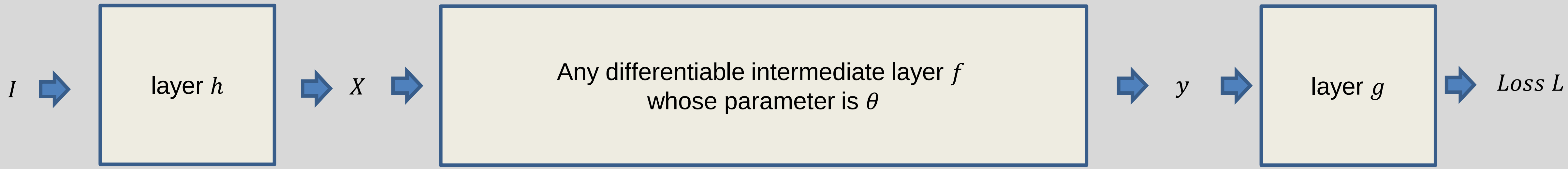
RGB image

Any Differentiable Layers

cat

Semantic labels

Differentiable layers



We need to implement three things for an intermediate layer f :

forward rule:	$y = f(\mathrm{h}(\mathrm{I});\theta)$	for	$g\big(f(\mathrm{h}(\mathrm{I});\theta)\big) = \mathrm{L}$
backward rule:	$\frac{dy}{dX}$	for	$\frac{dL}{d\mathrm{I}} = \frac{d\mathrm{X}}{d\mathrm{I}} \times \frac{dy}{d\mathrm{X}} \times \frac{dL}{dy}$
parameter update rule:	$\frac{dy}{d\theta}$	for	$\theta^{new} = \theta - \varepsilon \frac{dy}{d\theta} \times \frac{dL}{dy}$

Via chain rule, the entire architecture becomes differentiable, if each layer becomes differentiable.

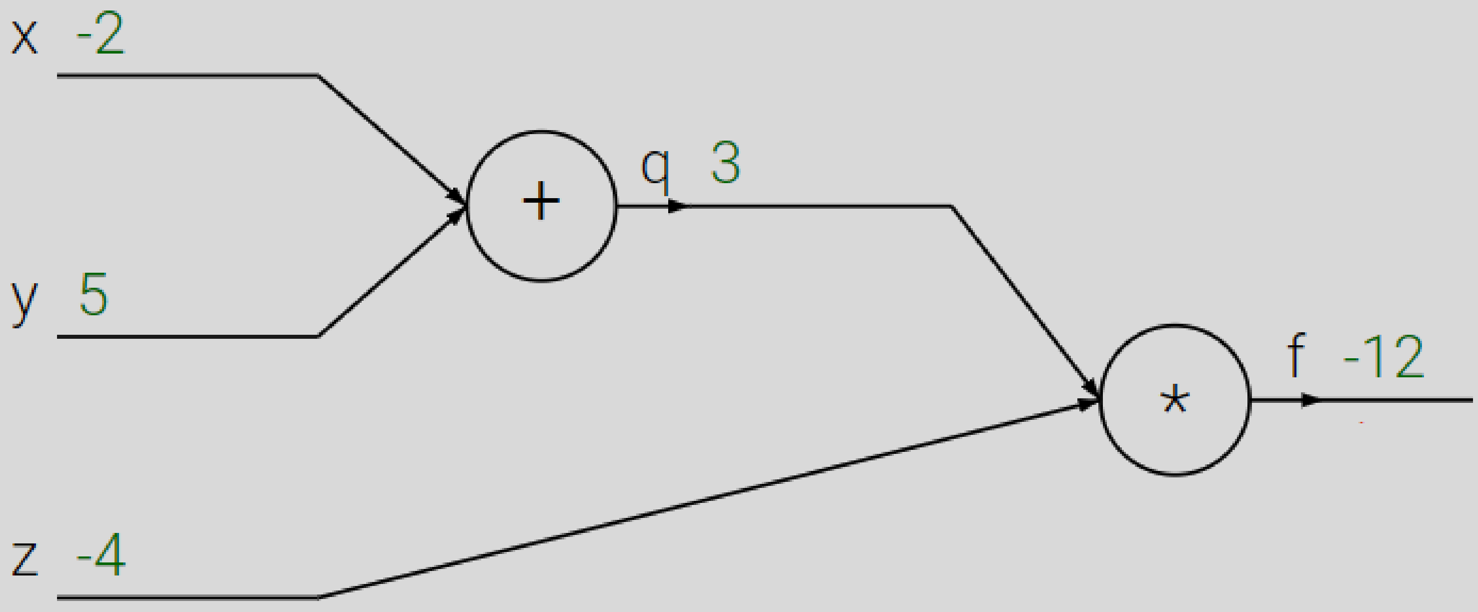
Gradient descent

$$L(\theta) = g(f(h(I); \theta))$$

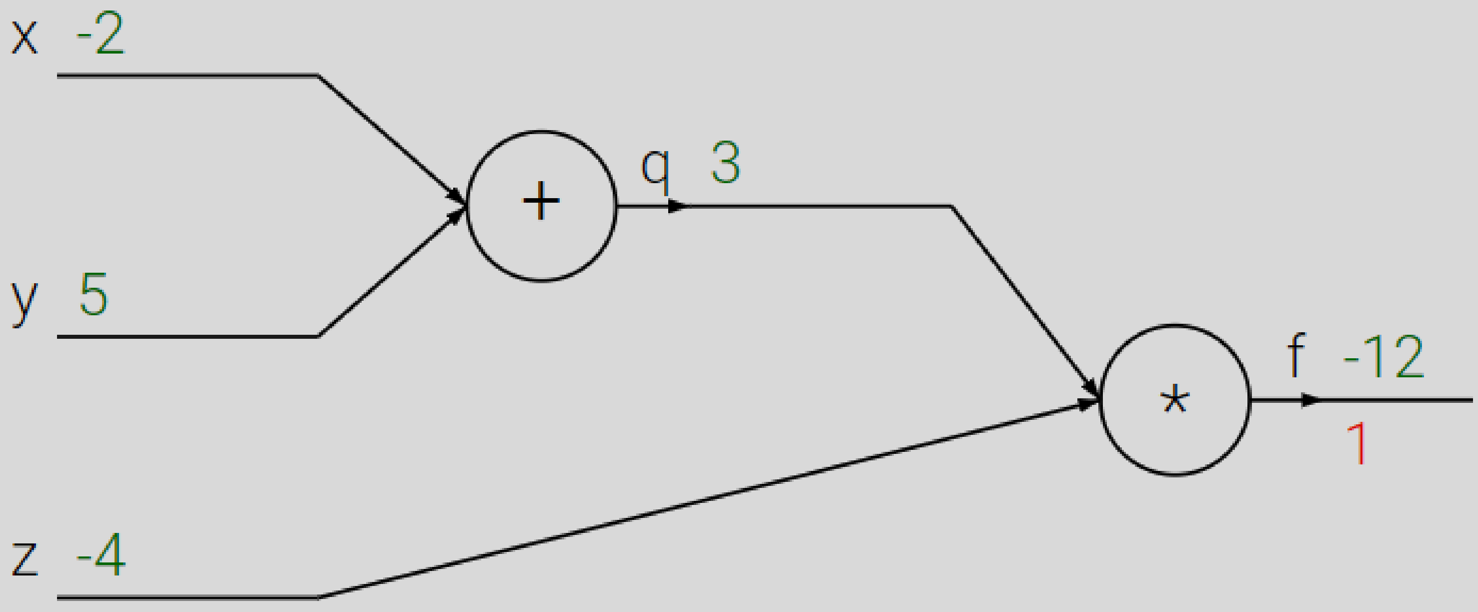
$$\theta^{\mathrm{new}} = \theta - \epsilon \frac{\partial}{\partial \theta} L(\theta)$$

ϵ : Learning rate (small value e.g. 0.1)

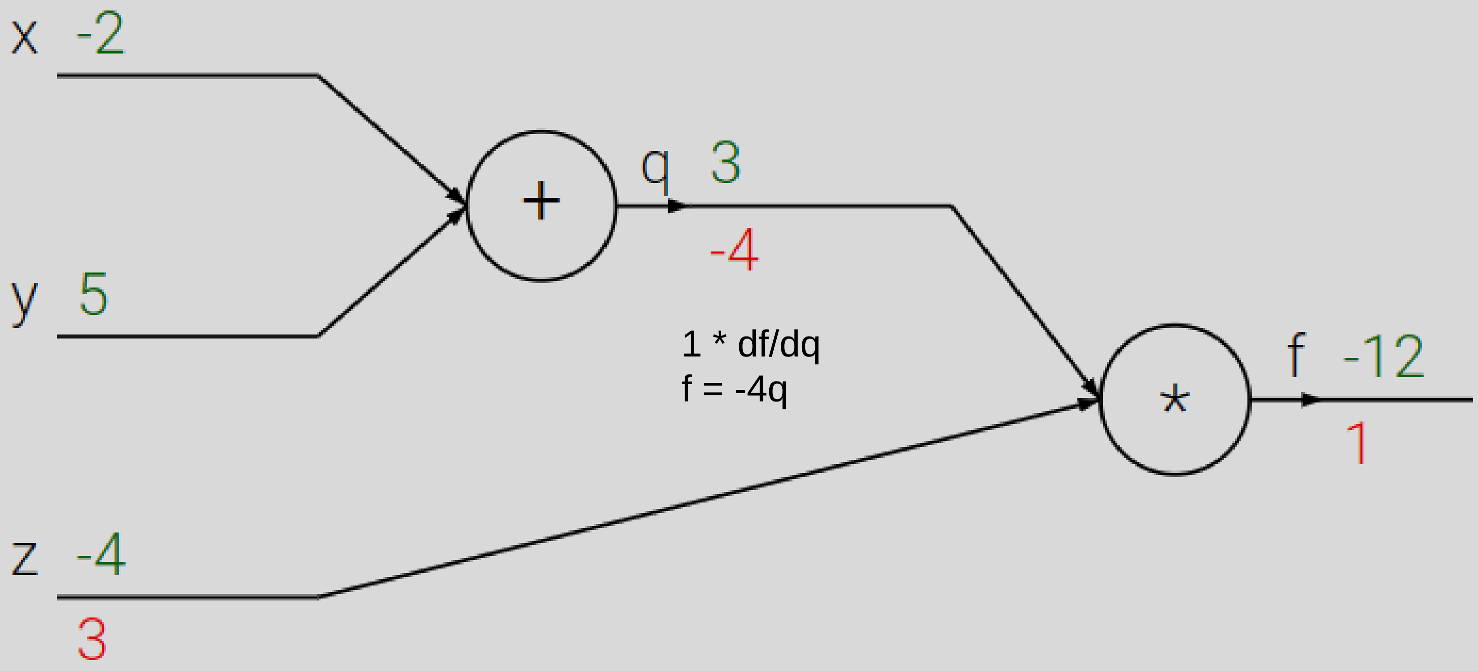
Calculating gradients in deep learning



Calculating gradients in deep learning



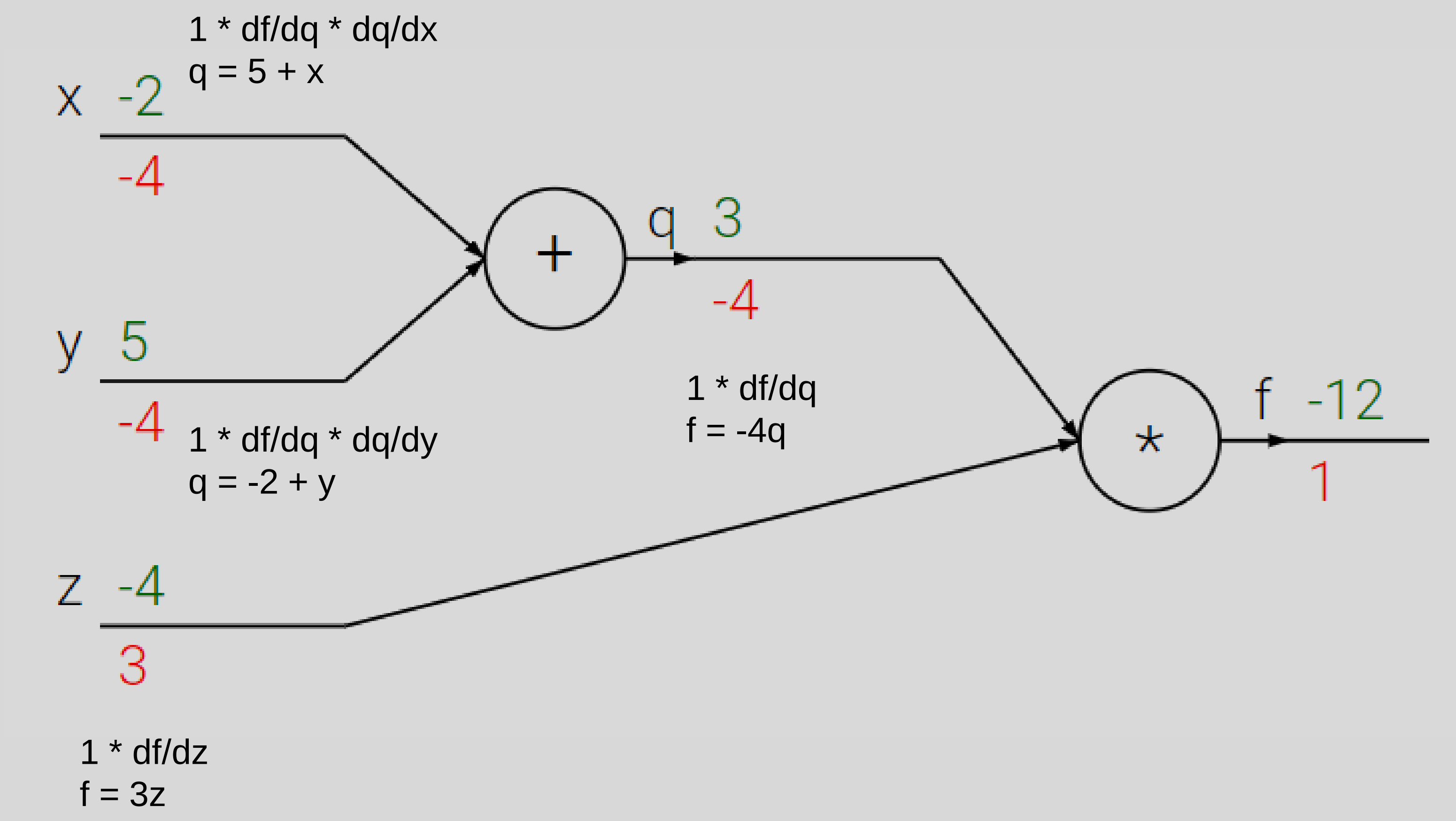
Calculating gradients in deep learning



1 * df/dq
f = -4q

1 * df/dz
f = 3z

Calculating gradients in deep learning



CNNs

RGB image

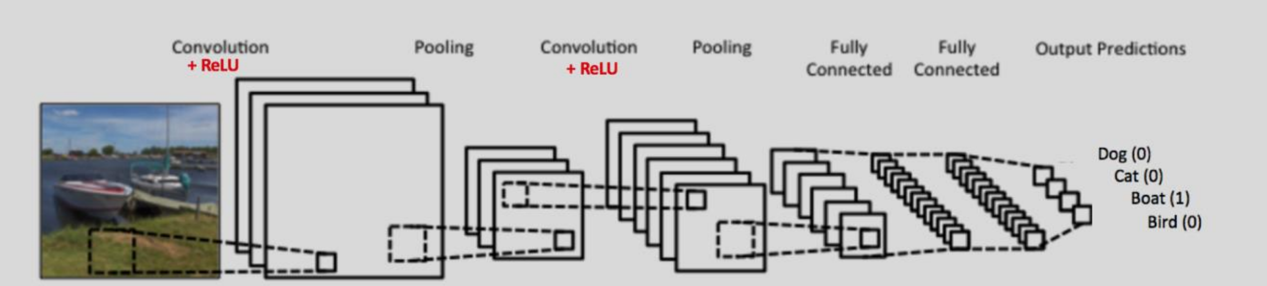
Any Differentiable Layers

cat

Semantic labels

CNNs

RGB image



cat

Semantic labels

2D Convolution

1	1	0	1	2
2	0	0	1	0
0	0	2	2	1
0	0	0	2	0
0	0	2	1	1

Image (5x5)

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

Filter kernel (3x3)

2D Convolution

1*0+1*0+0*-1
+ 2*1+0*-1+0*0
+ 0*0+0*1+2*-1

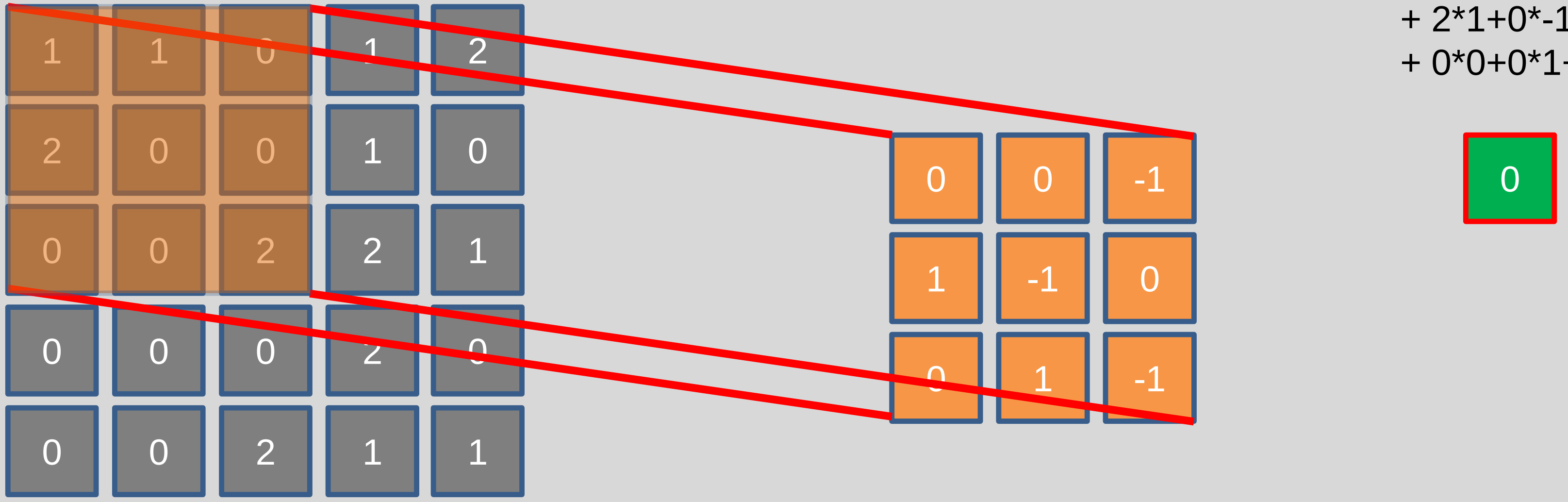
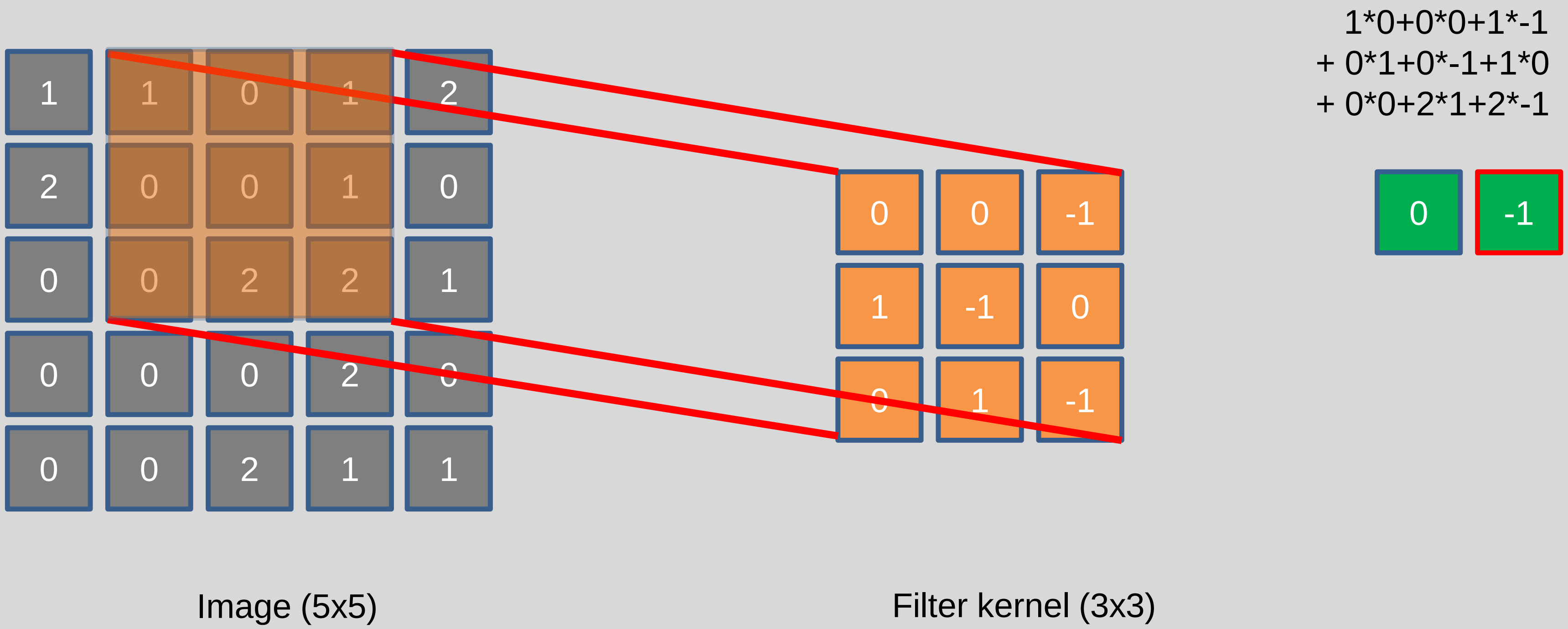


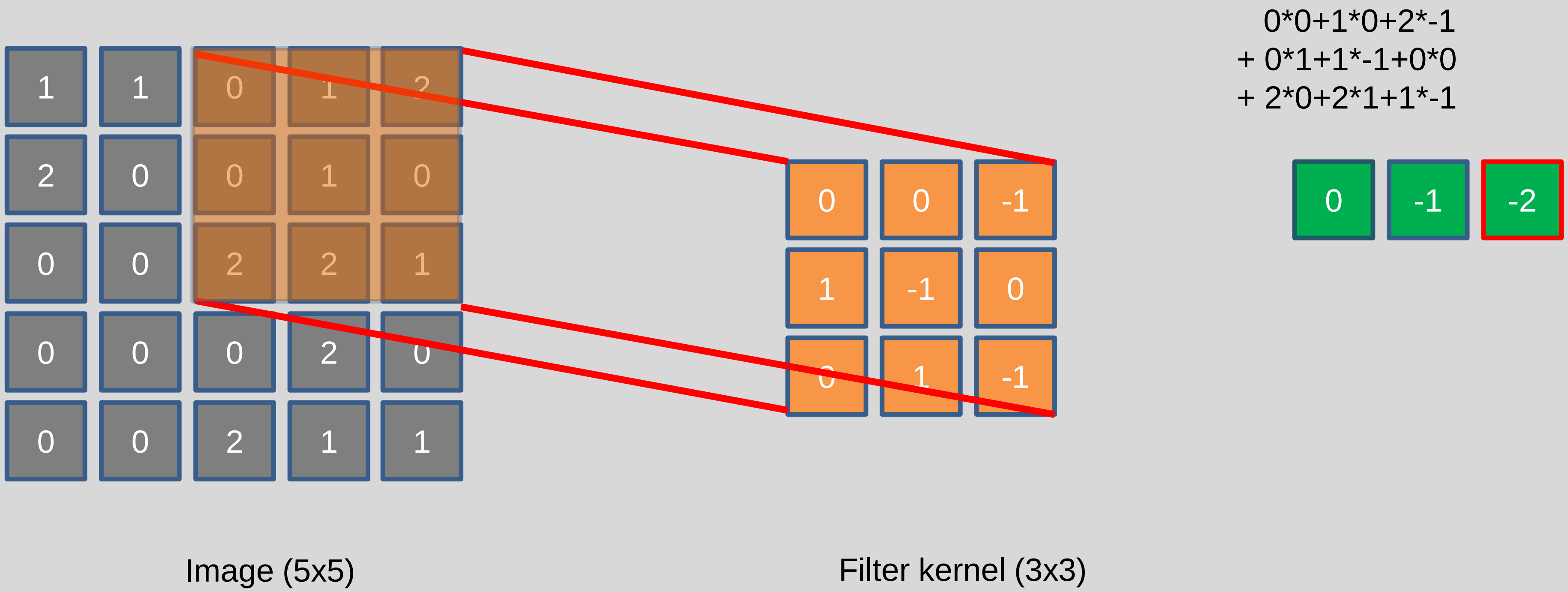
Image (5x5)

Filter kernel (3x3)

2D Convolution



2D Convolution



2D Convolution

2*0+0*0+0*-1
+ 0*1+0*-1+2*0
+ 0*0+0*1+0*-1

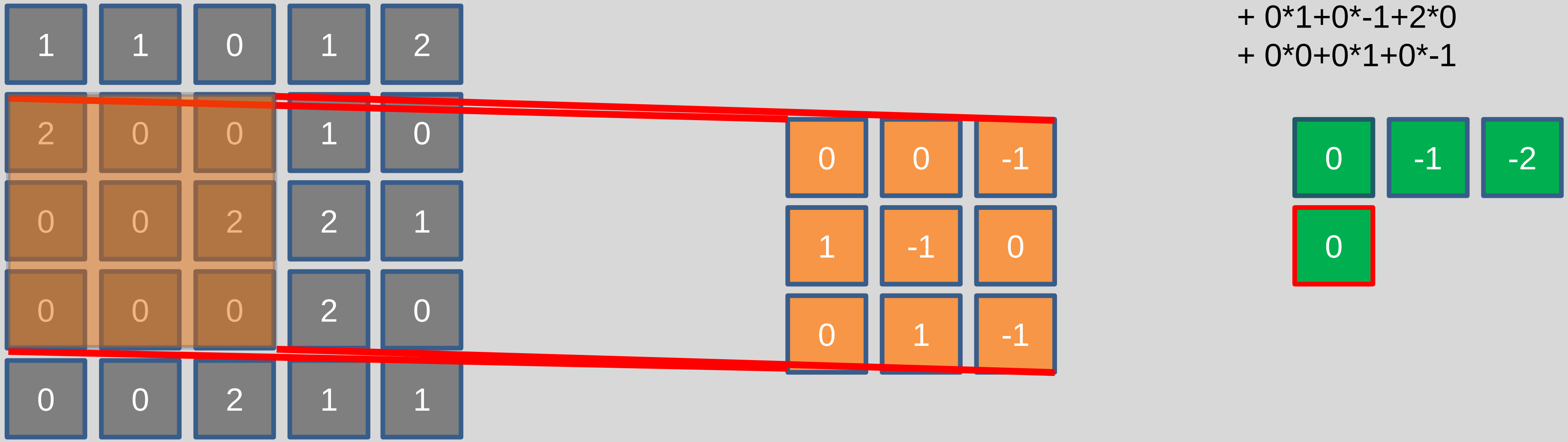


Image (5x5)

Filter kernel (3x3)

2D Convolution

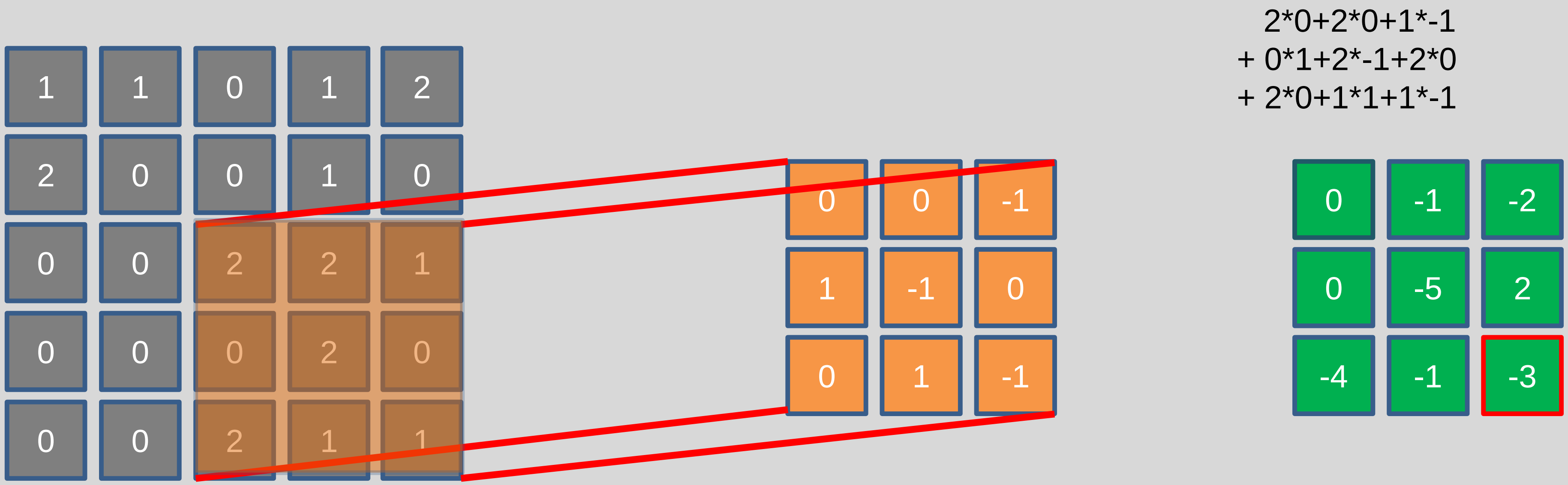


Image (5x5)

Filter kernel (3x3)

Output feature (3x3)

Ex. Image blurring

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

2D Convolution w/ zero padding

0*0+0*0+0*-1
+ 0*1+1*-1+1*0
+ 0*0+2*1+0*-1

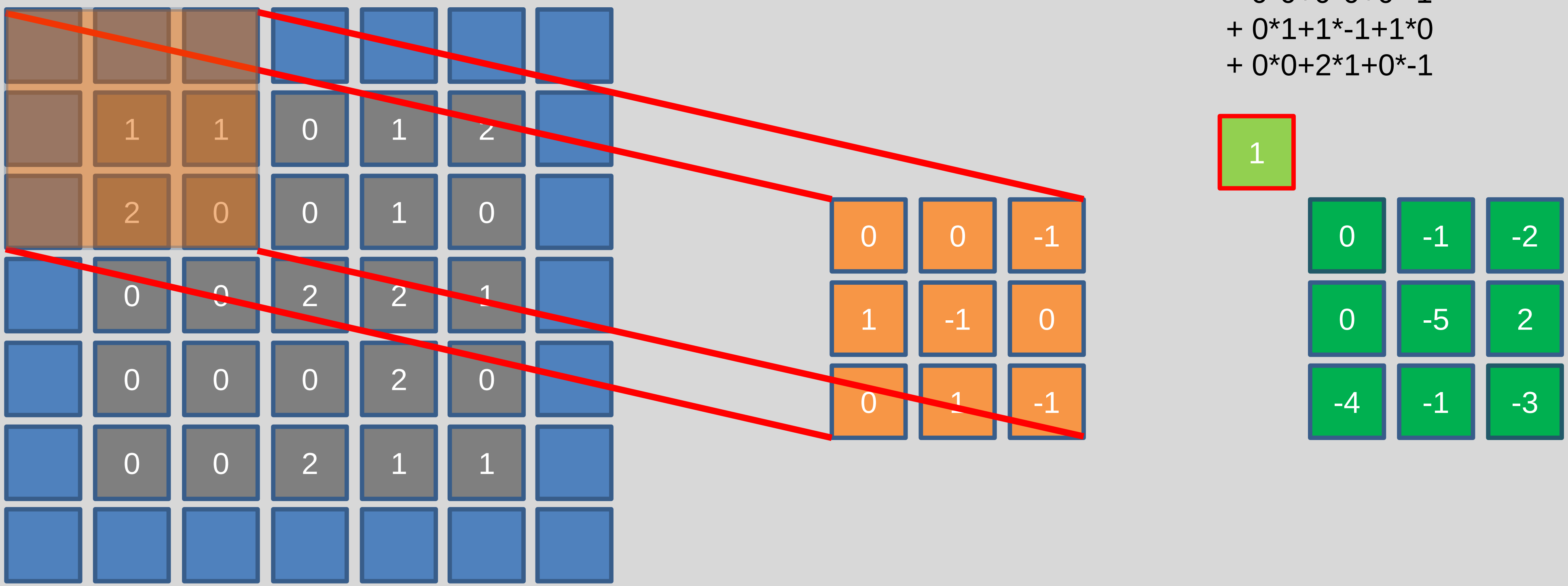


Image (5x5)

Filter kernel (3x3)

2D Convolution w/ zero padding

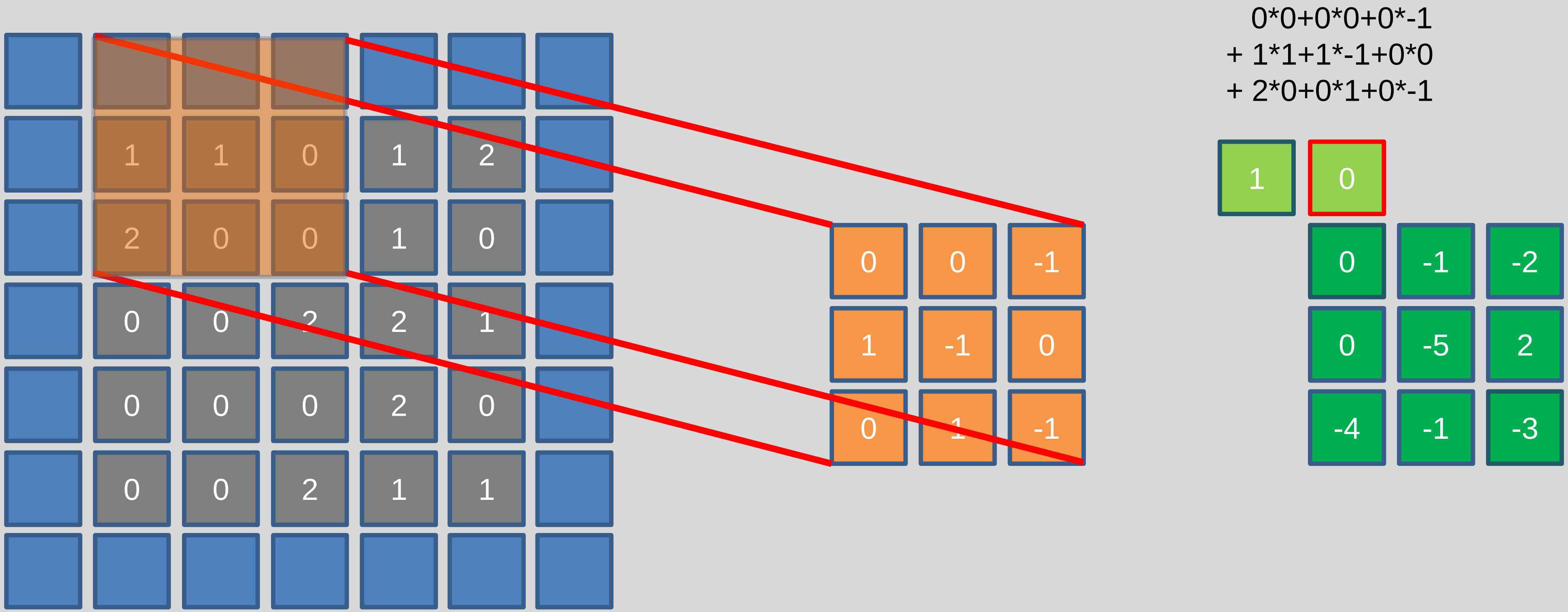


Image (5x5)

Filter kernel (3x3)

2D Convolution w/ zero padding

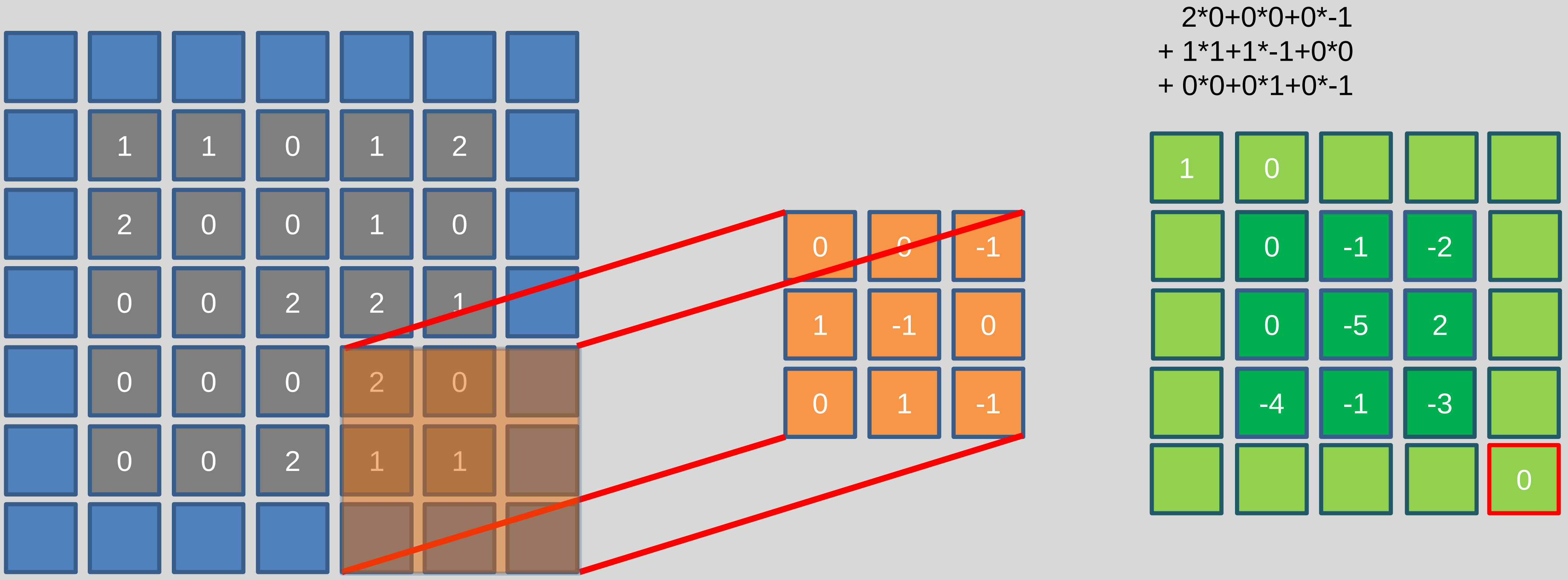


Image (5x5)

Filter kernel (3x3)

Output feature (5x5)

2D Convolution w/ zero padding

→ (K-1) / 2 padding is required to obtain the original size.

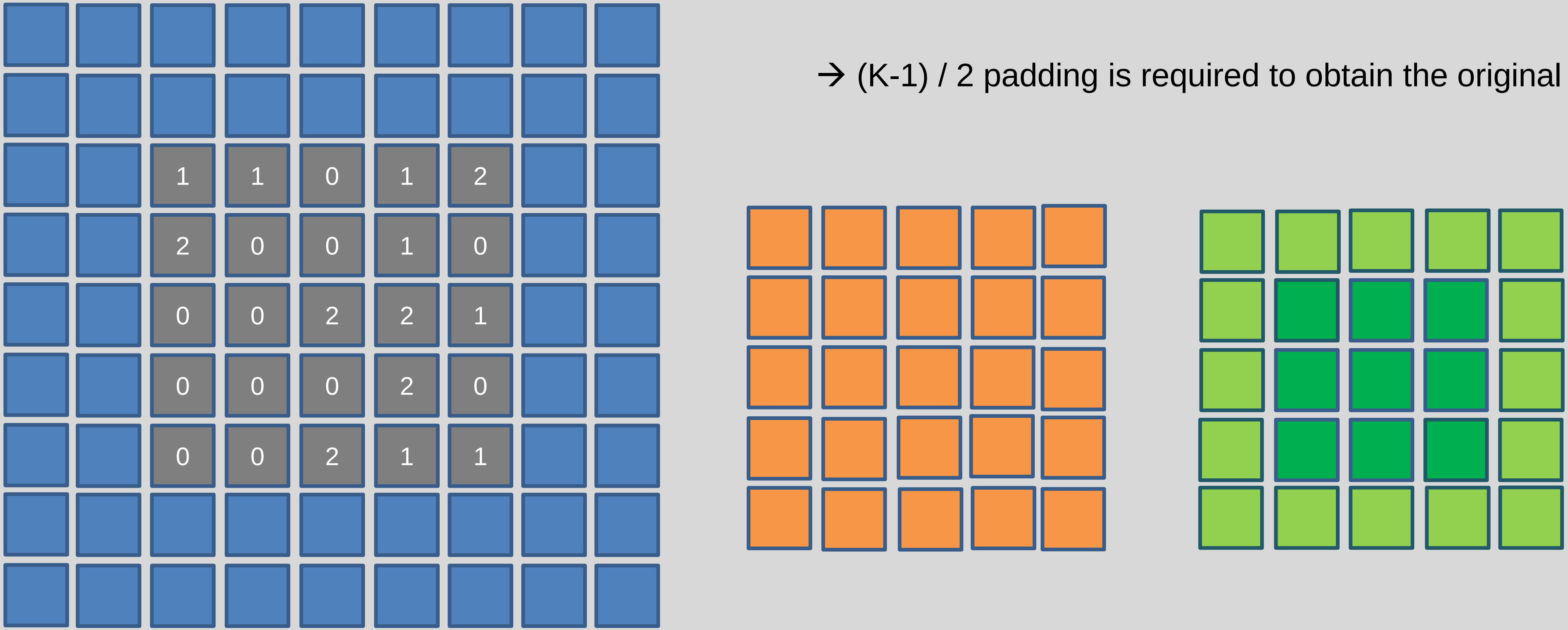


Image (5x5)

Filter kernel (5x5)

Output feature (5x5)

2D Convolution w/ stride

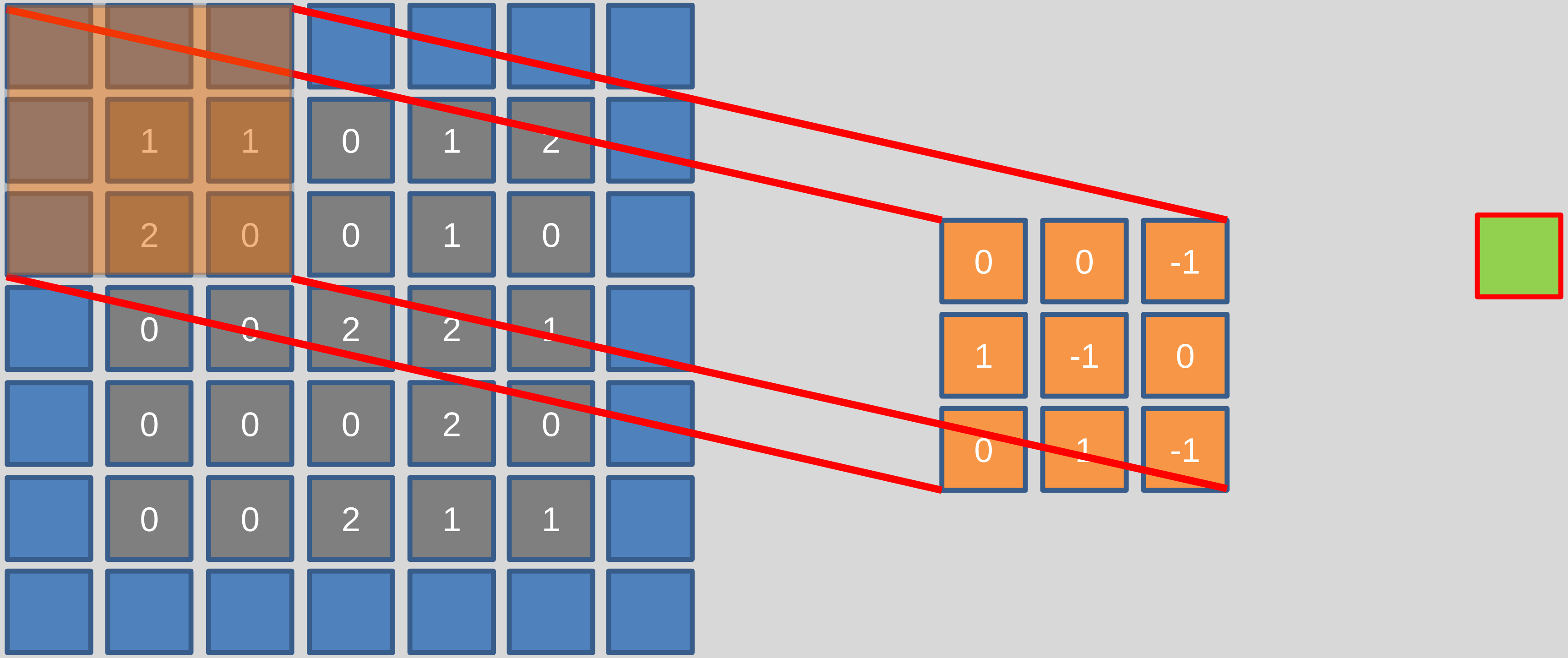


Image (5x5)

Filter kernel (3x3)

Stride=2

2D Convolution w/ stride

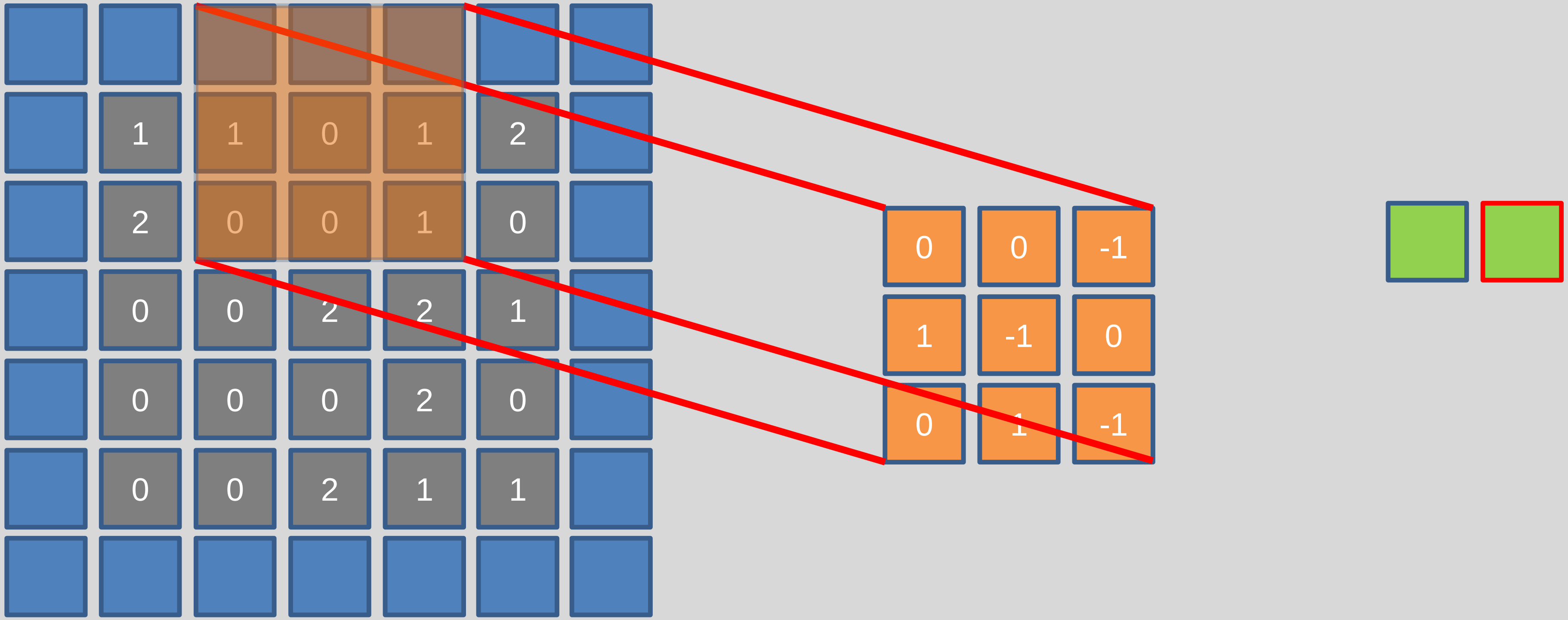


Image (5x5)

Filter kernel (3x3)

Stride=2

2D Convolution w/ stride

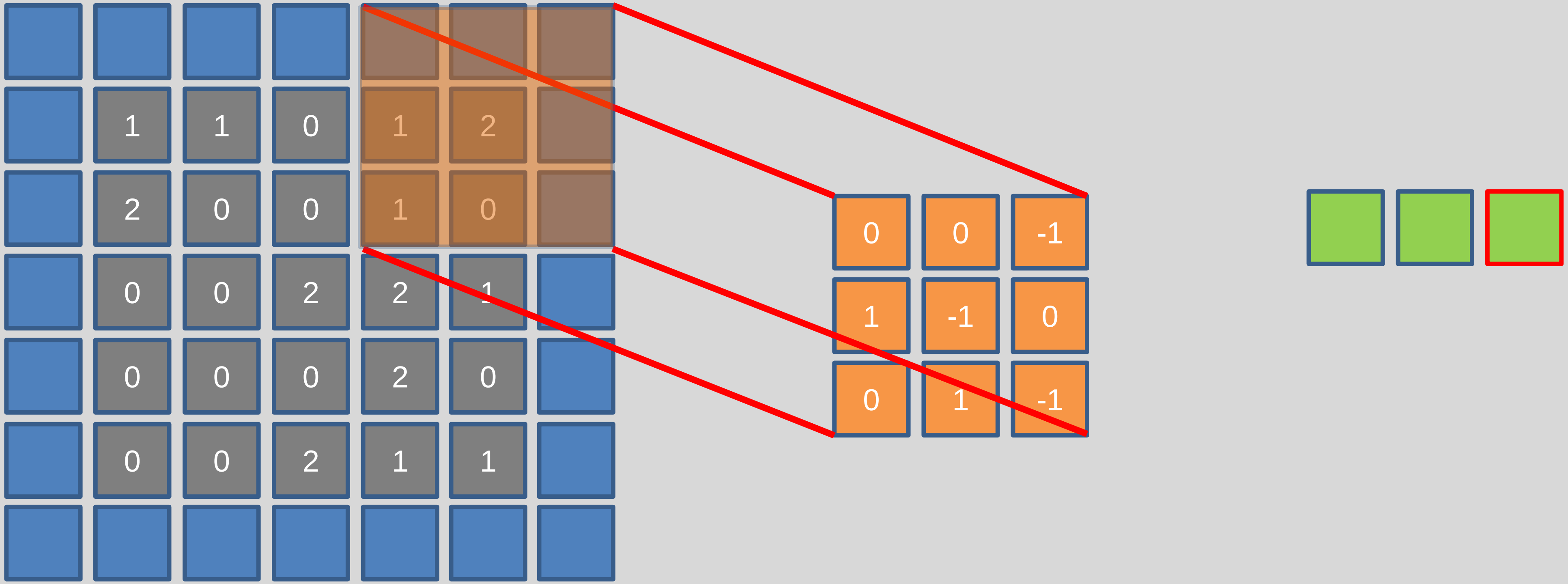


Image (5x5)

Filter kernel (3x3)

Stride=2

2D Convolution w/ stride

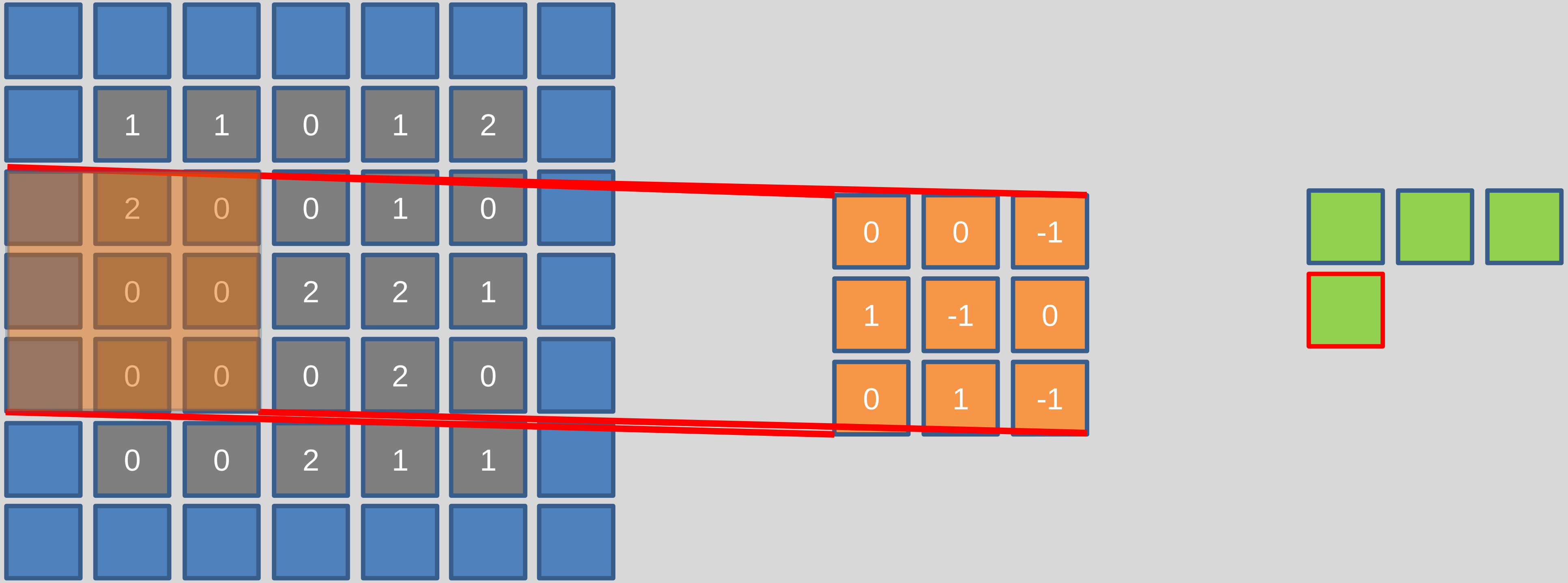


Image (5x5)

Filter kernel (3x3)

Stride=2

2D Convolution w/ stride

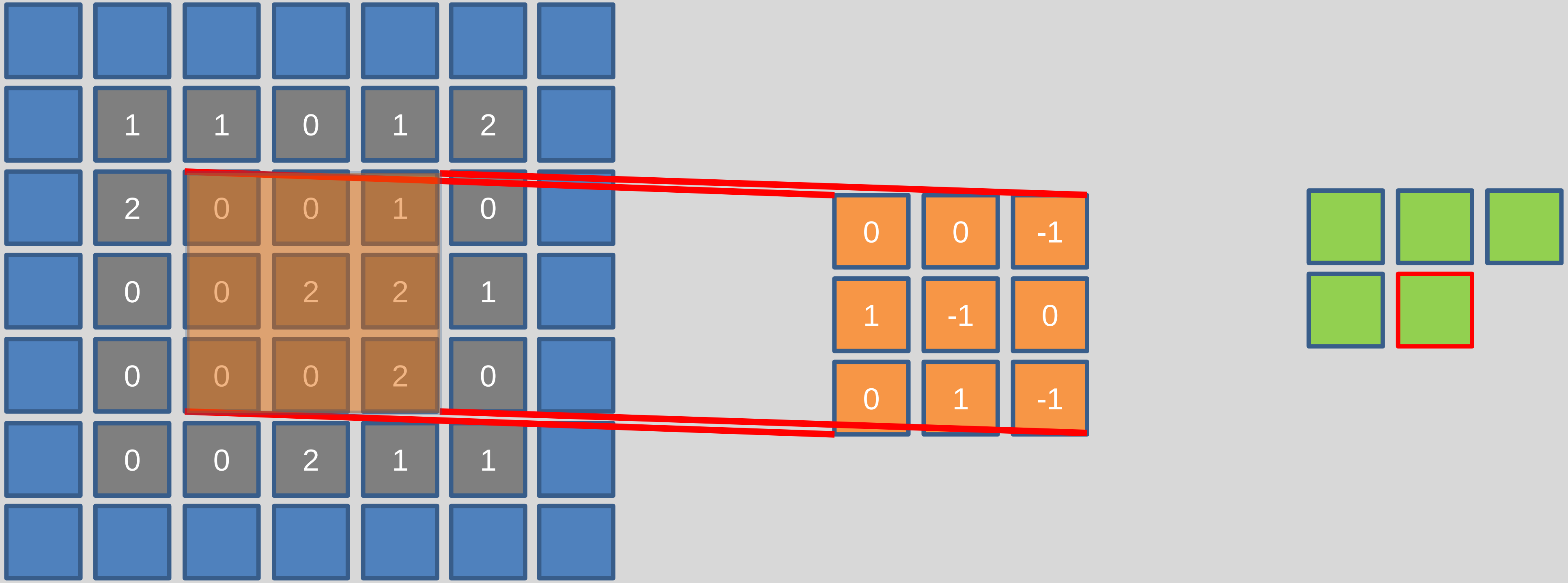


Image (5x5)

Filter kernel (3x3)

Stride=2

2D Convolution w/ stride

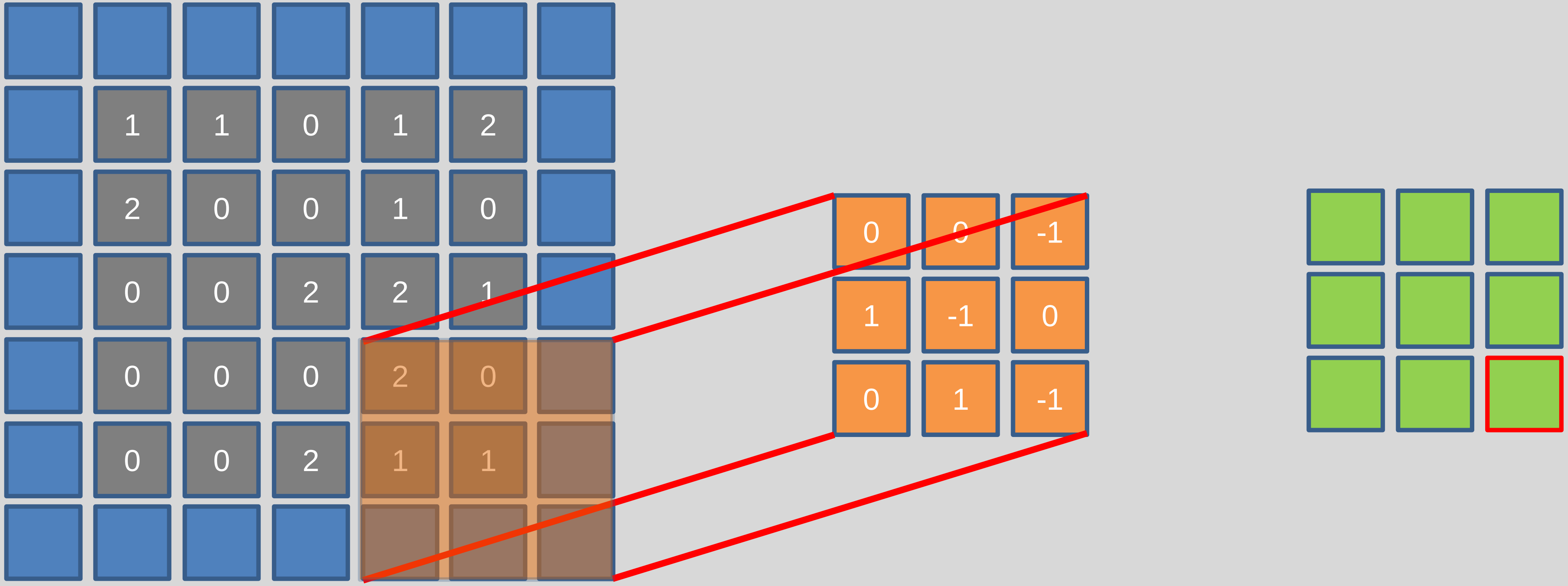
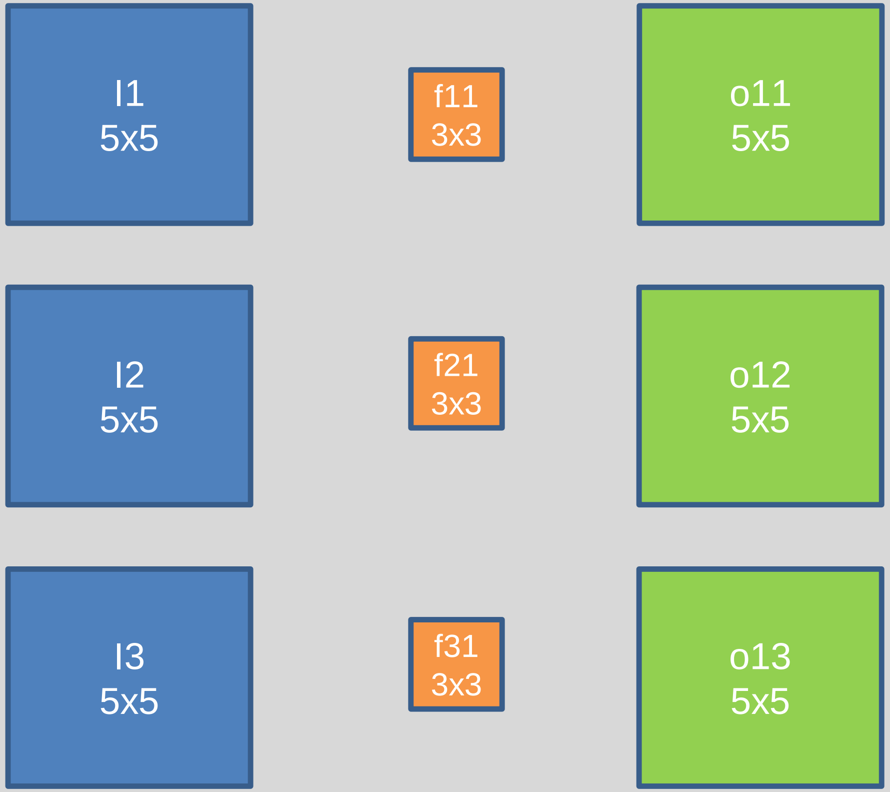


Image (5x5)

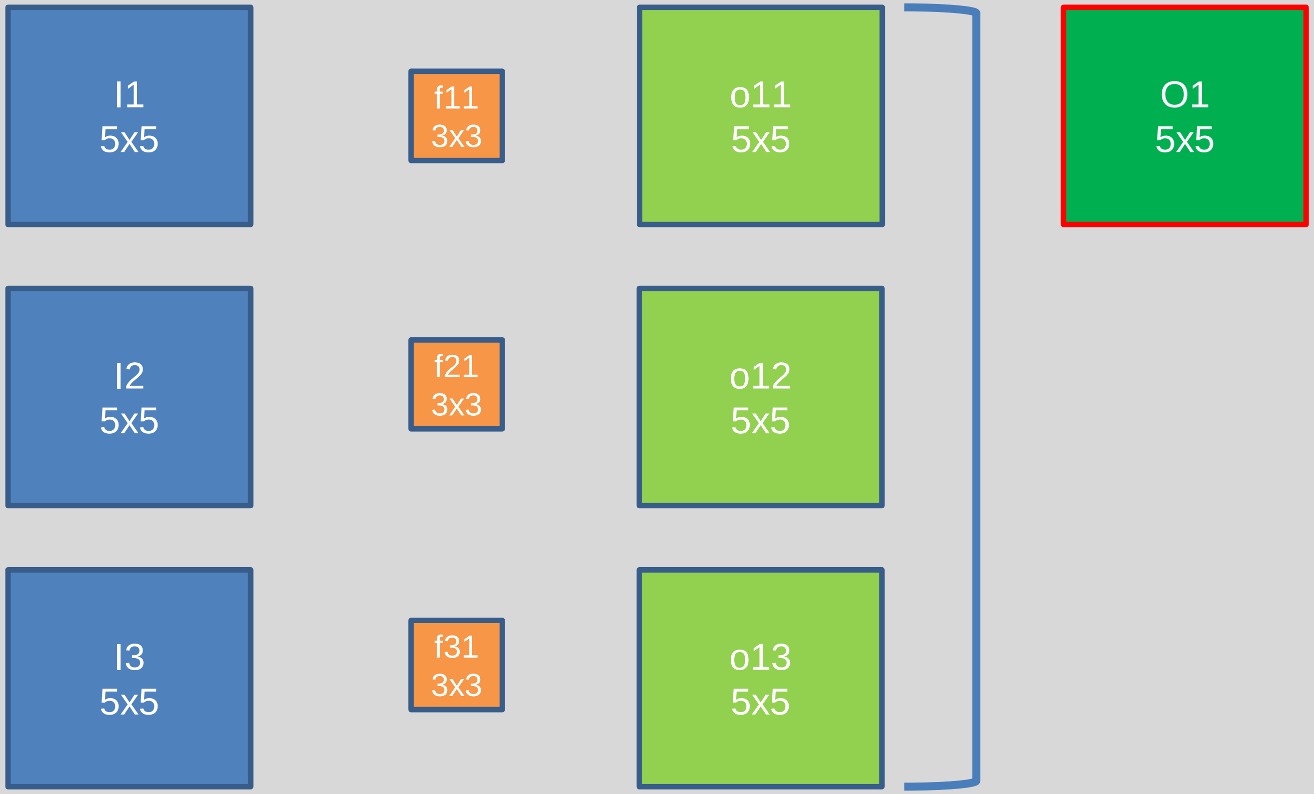
Filter kernel (3x3)

Stride=2

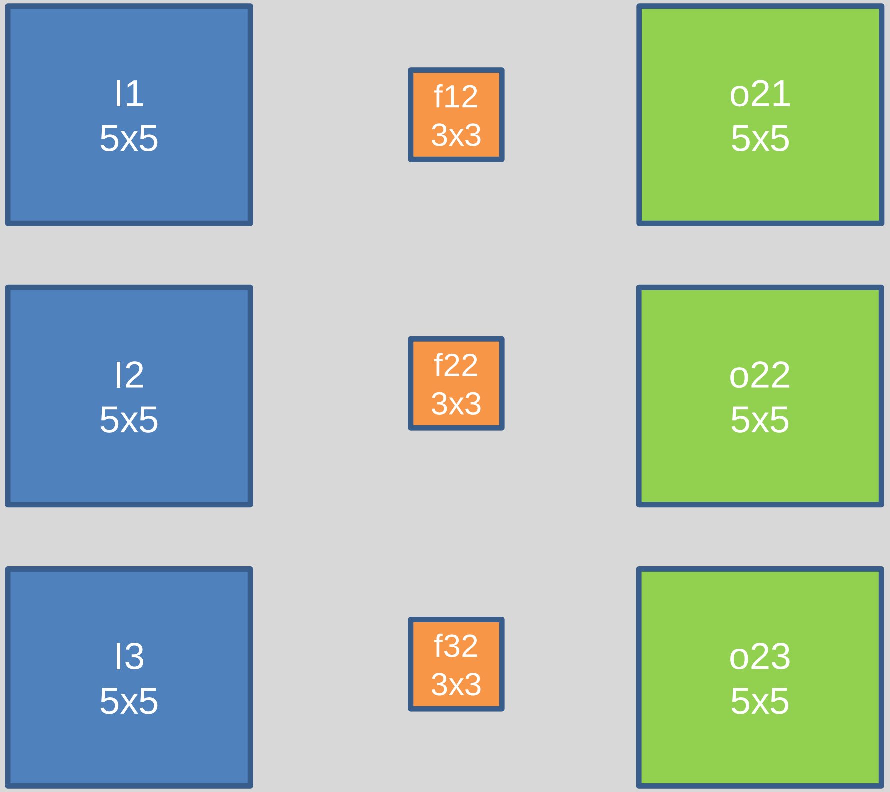
Convolutional layer



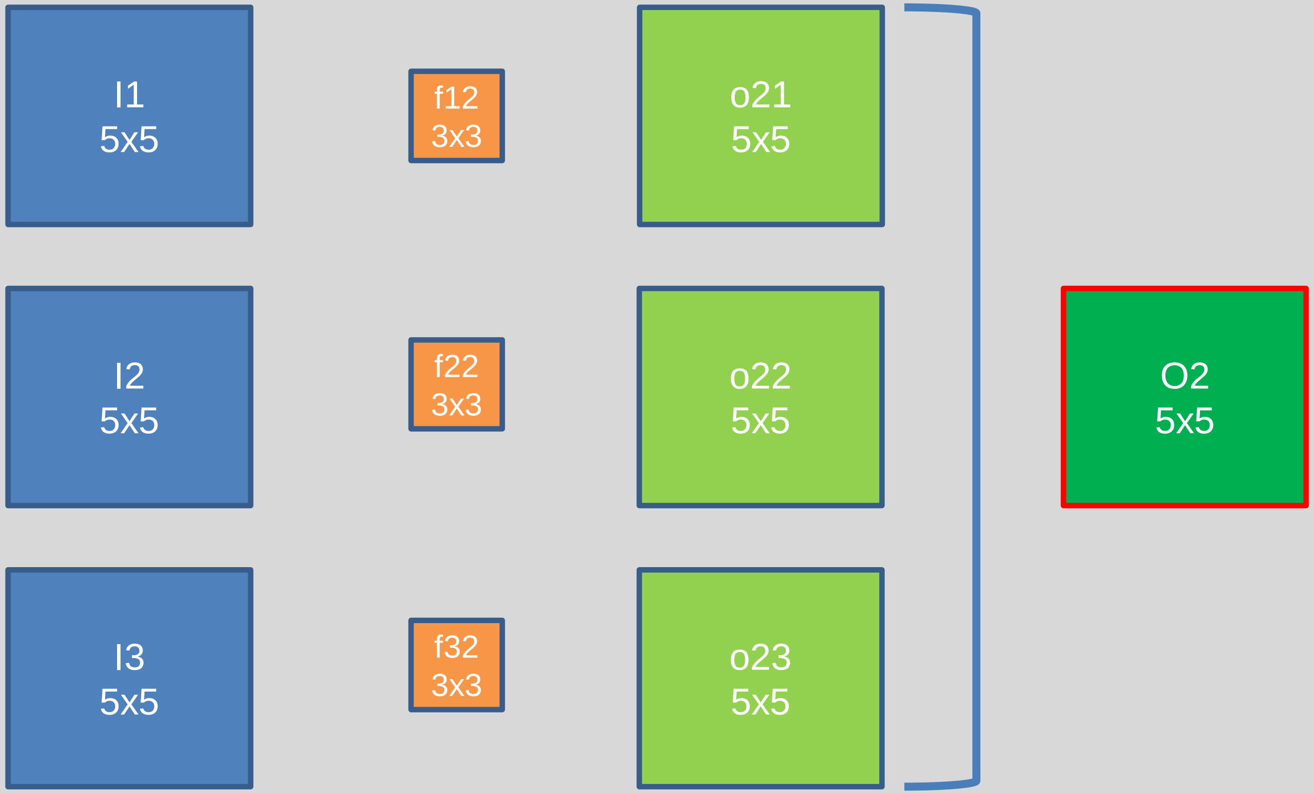
Convolutional layer



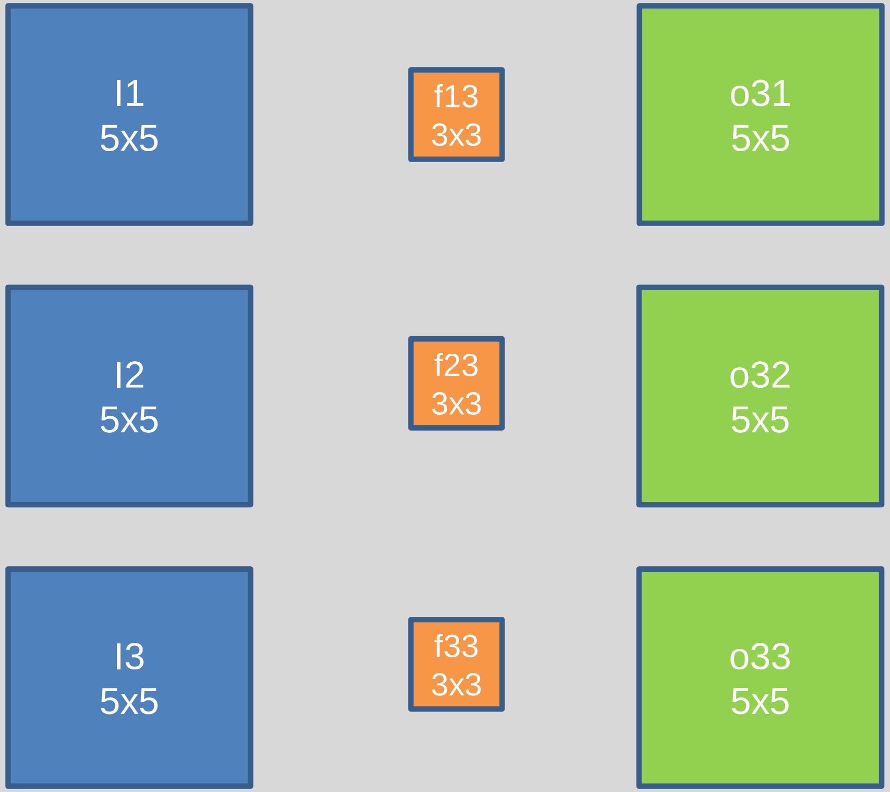
Convolutional layer



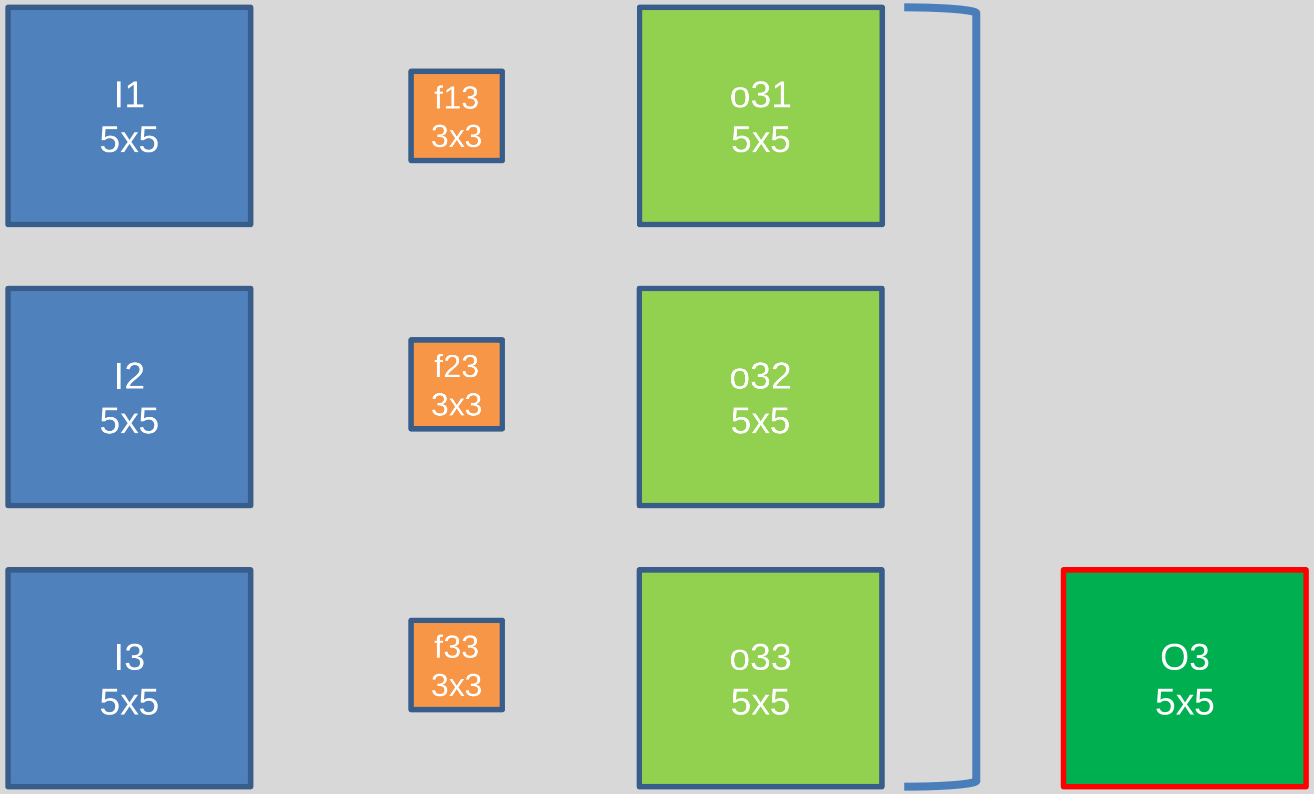
Convolutional layer



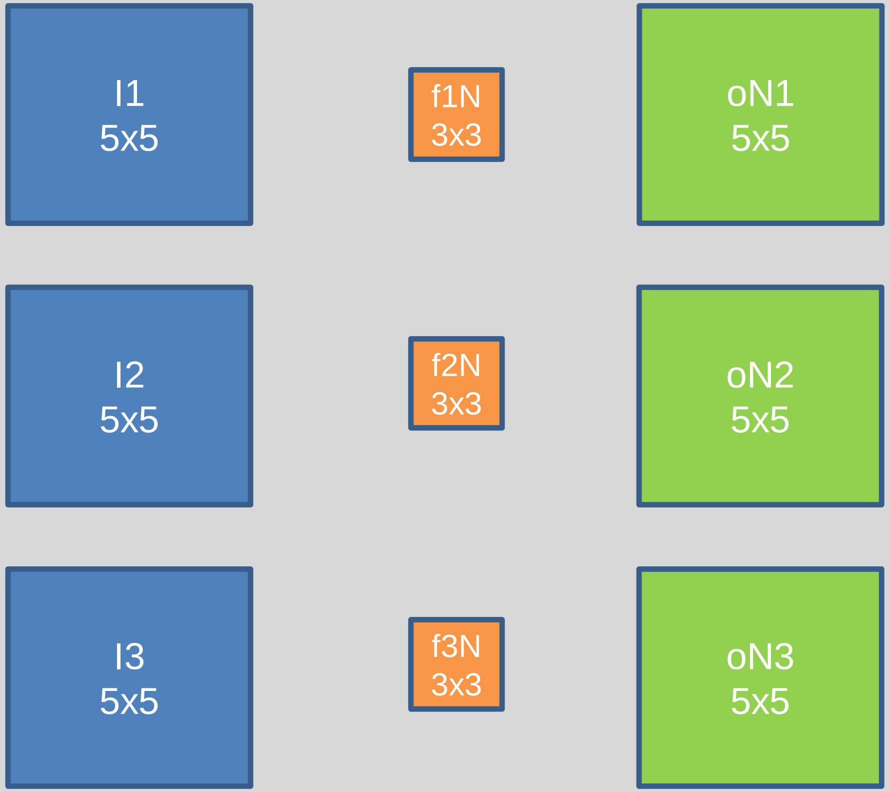
Convolutional layer



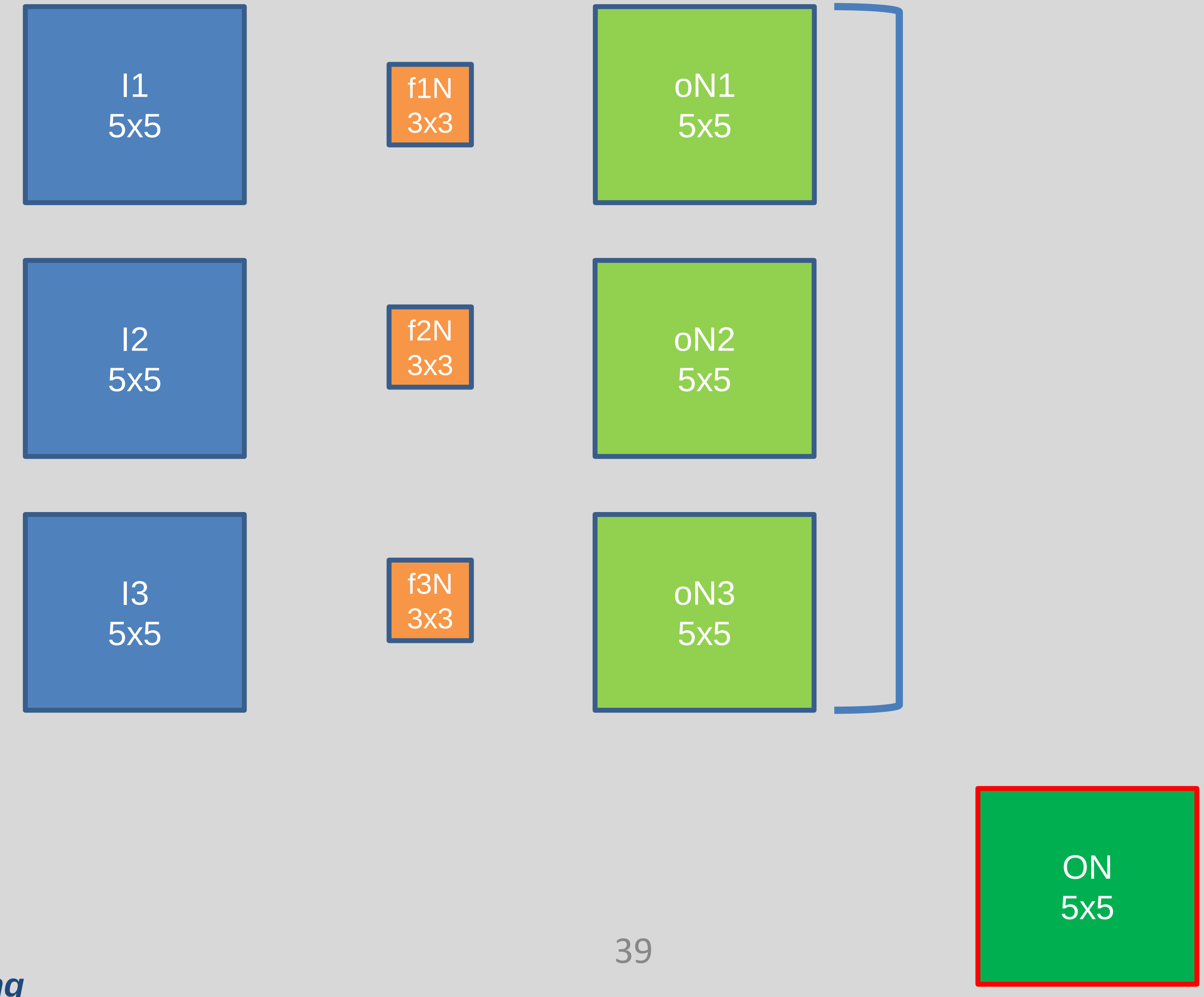
Convolutional layer



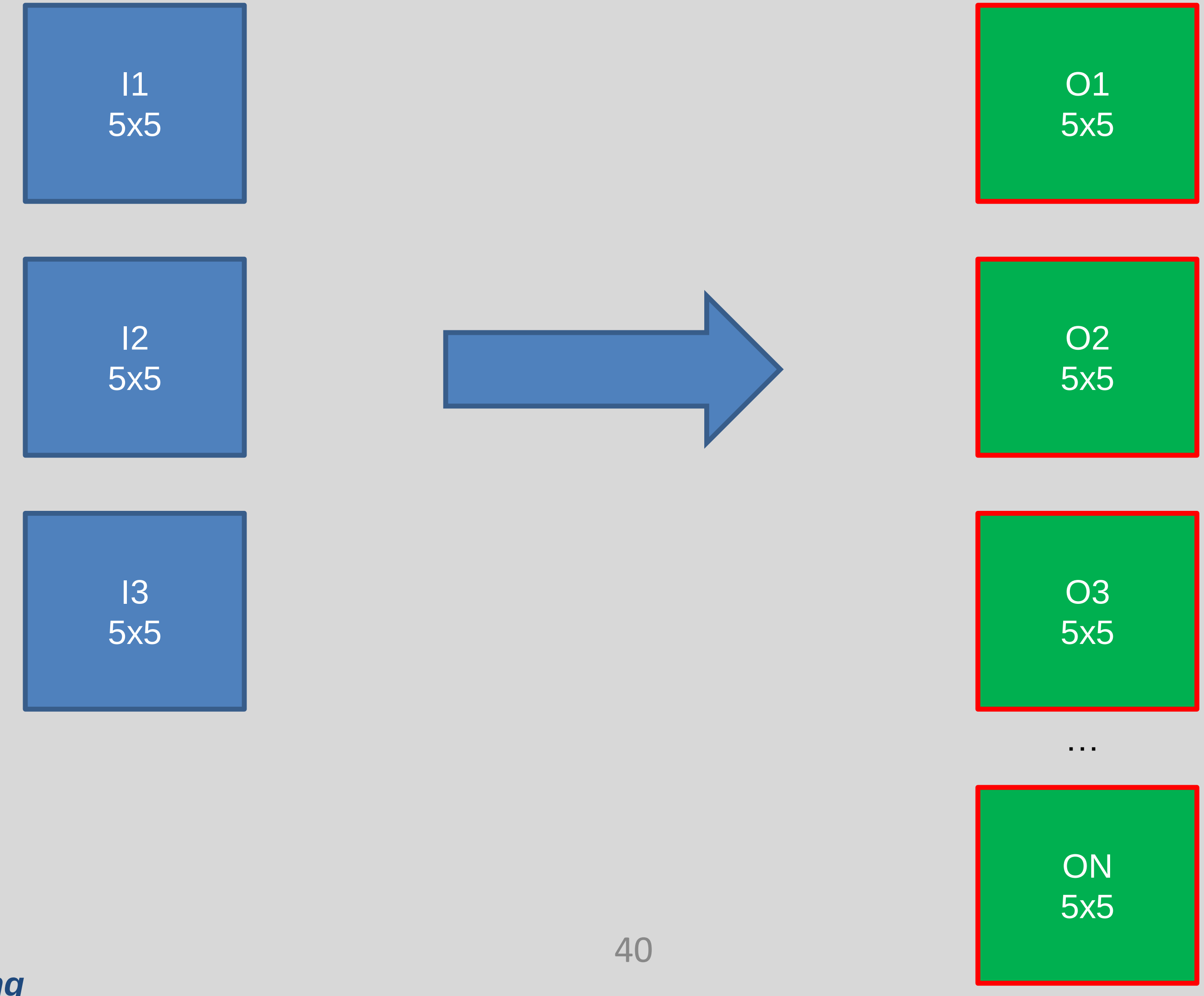
Convolutional layer



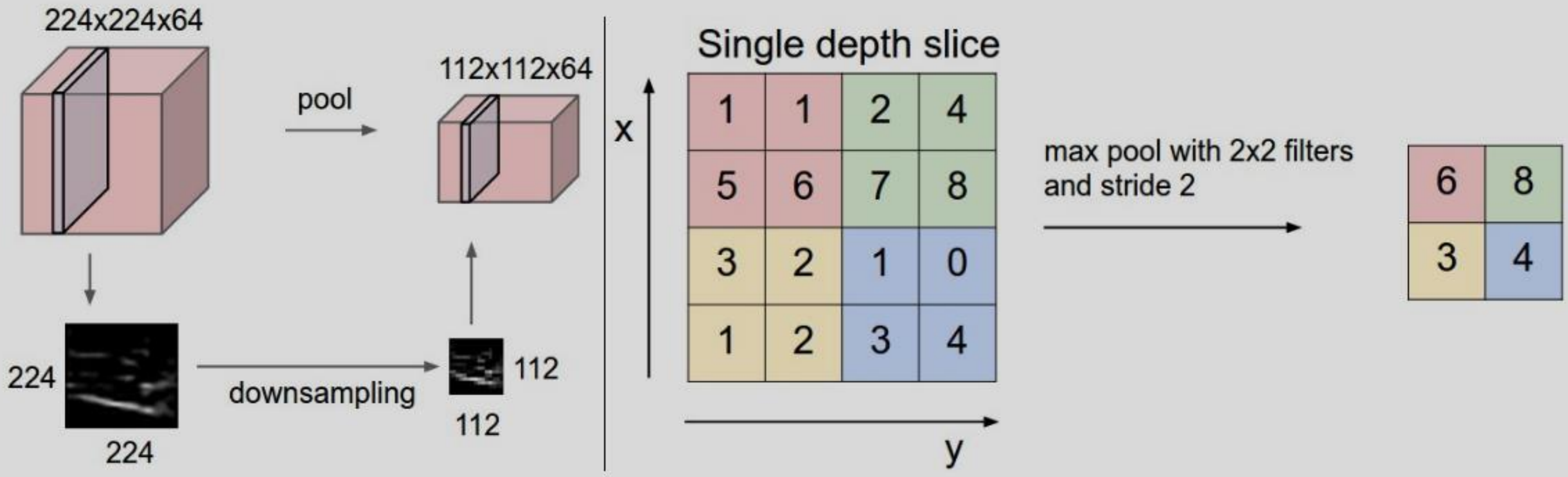
Convolutional layer



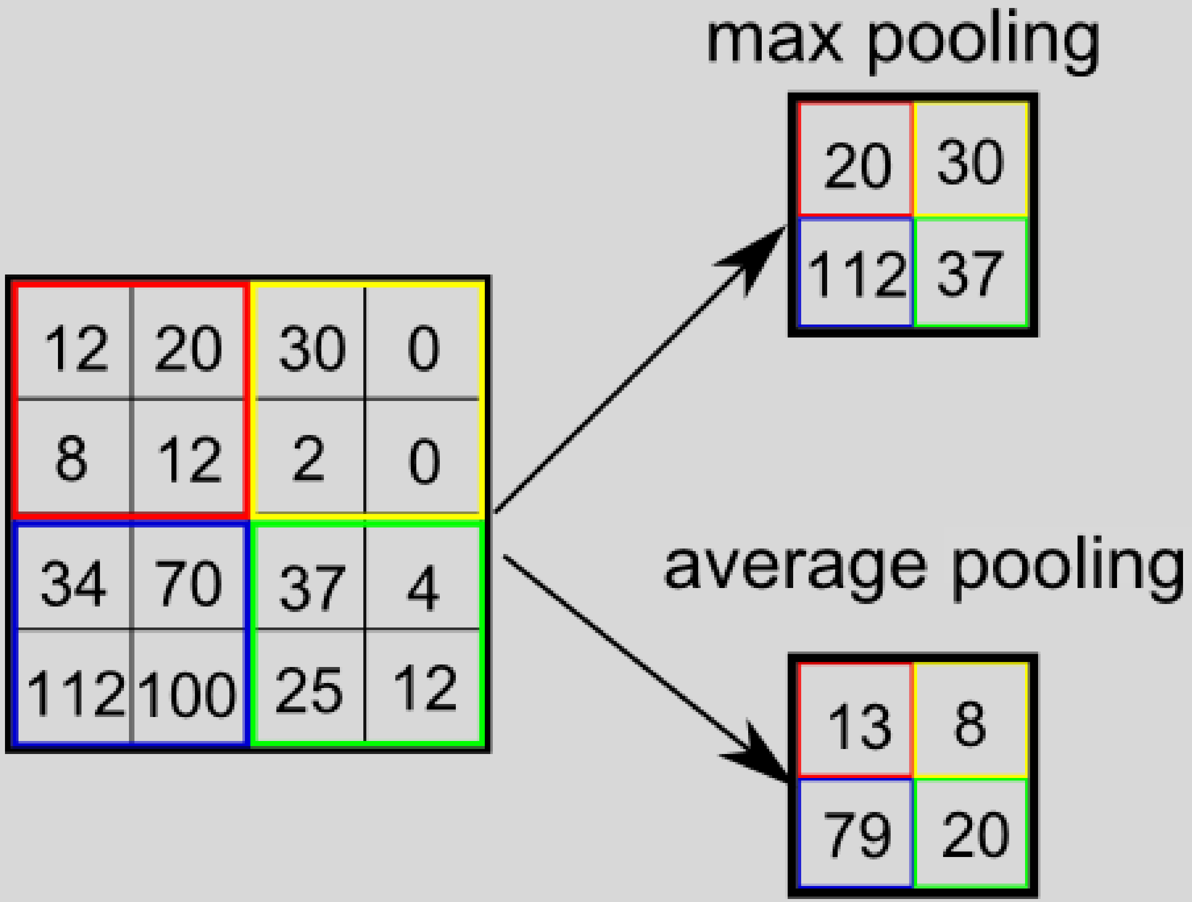
Convolutional layer



Pooling


224x224x64
pool
112x112x64
224
224
downsampling
112
112
x
y
Single depth slice
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters and stride 2
6 8
3 4

Pooling



torch.nn.MaxPool2d

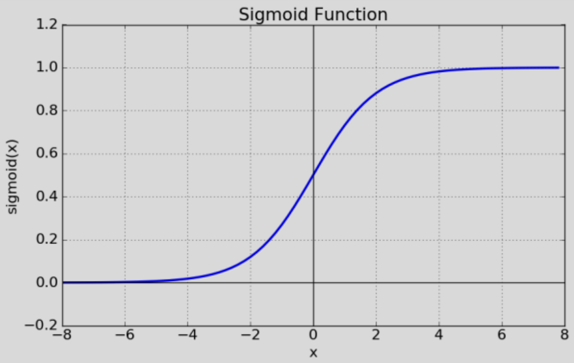
torch.nn.AvgPool2d

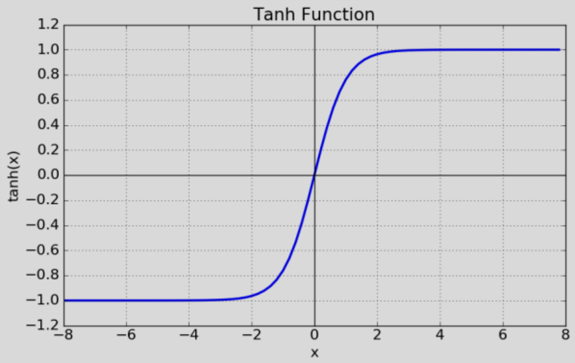
Activation layer

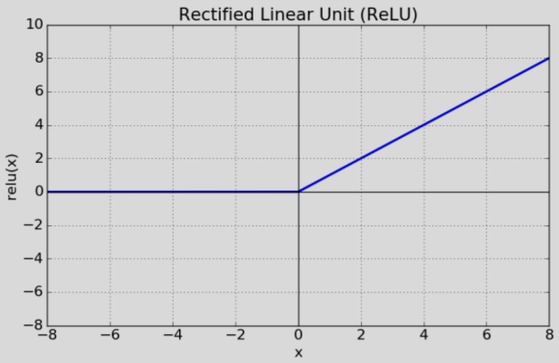
$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$f(x) = max(0, x)$$





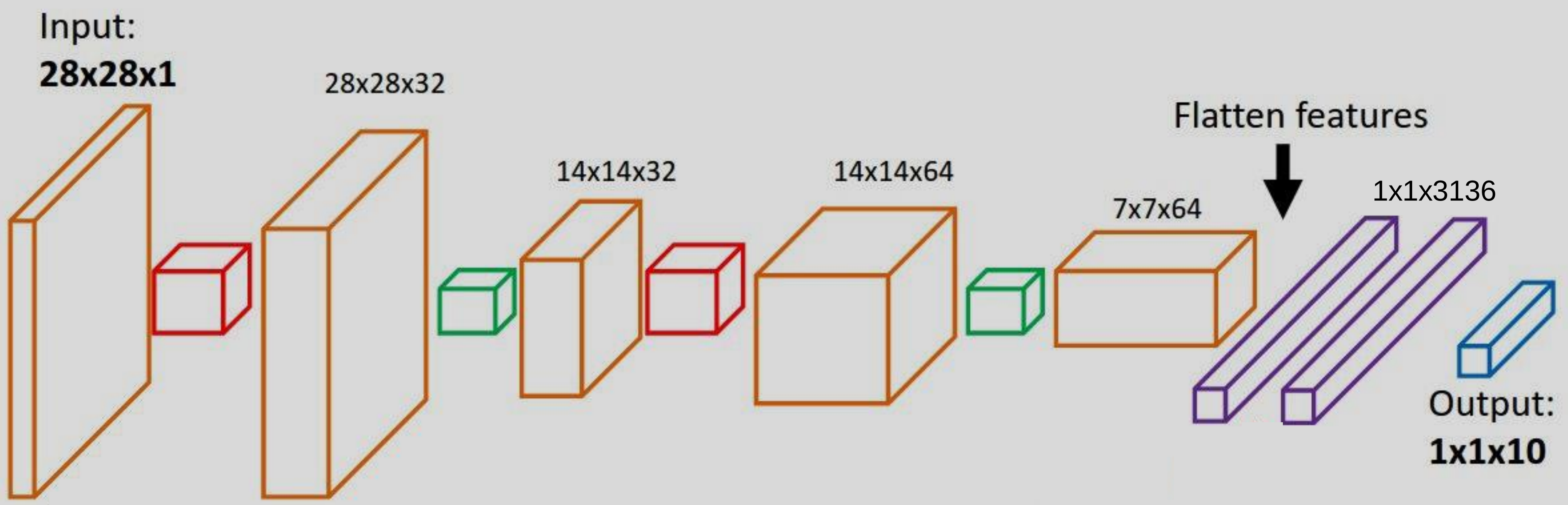


torch.nn.Sigmoid()

torch.nn.Tanh()

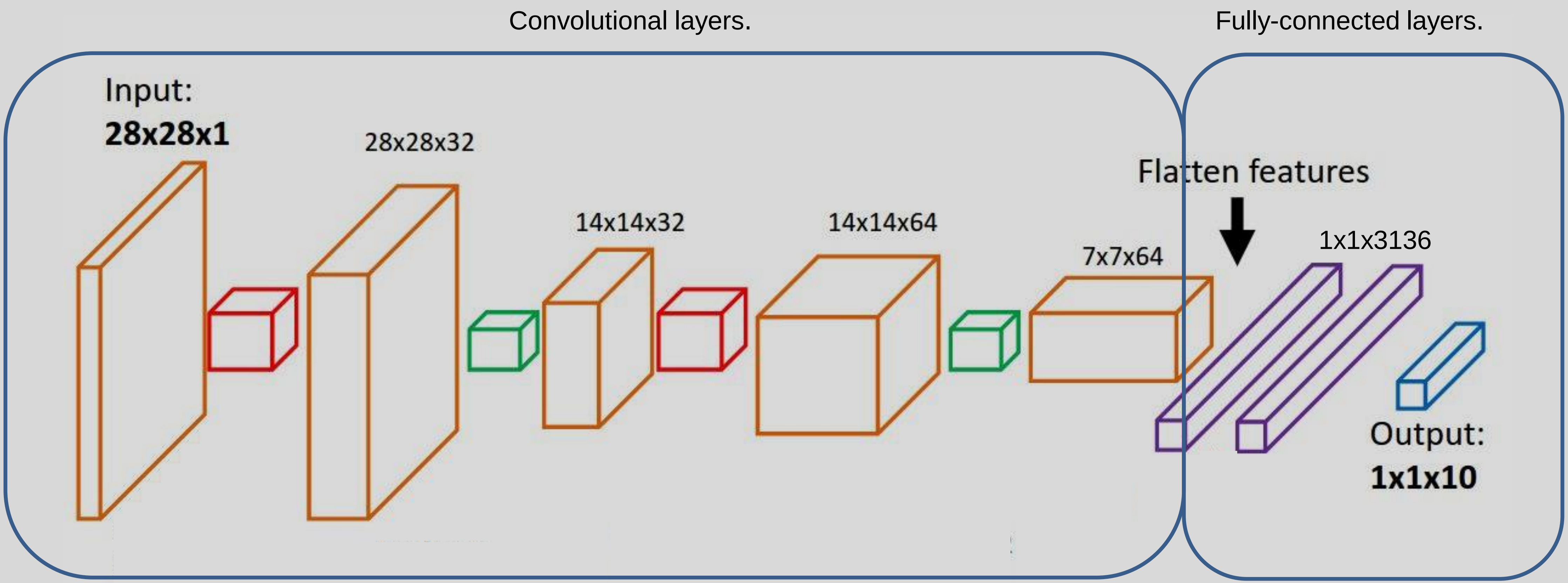
torch.nn.ReLU()

Overall CNN architecture



Combination of differentiable layers → Differentiable architecture!

Overall CNN architecture



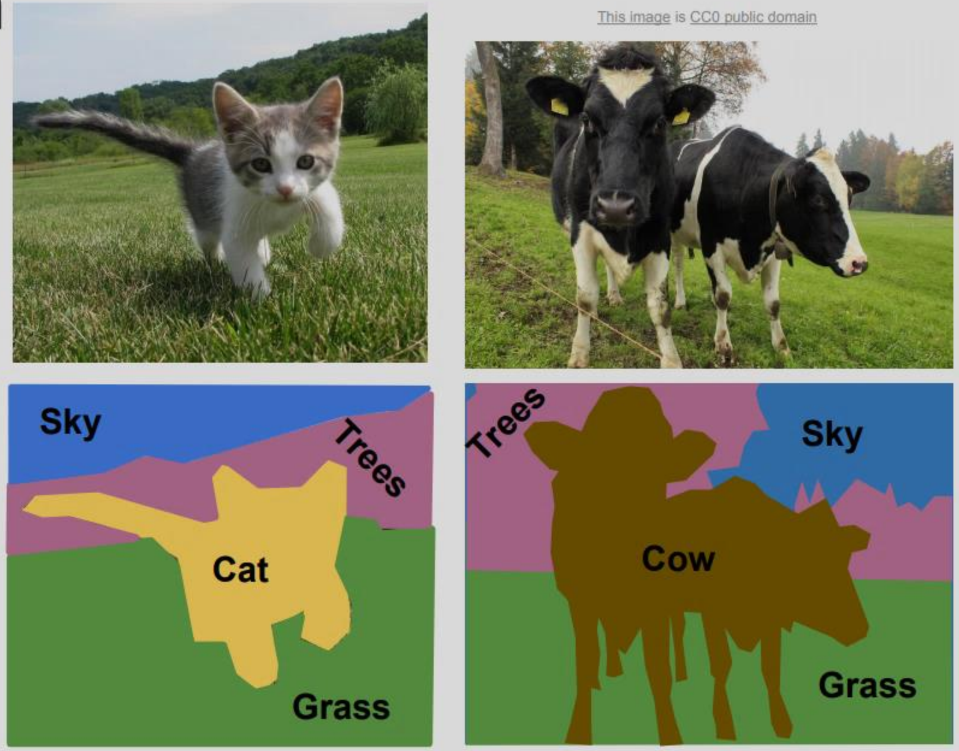
Combination of differentiable layers → Differentiable architecture!

Semantic Segmentation

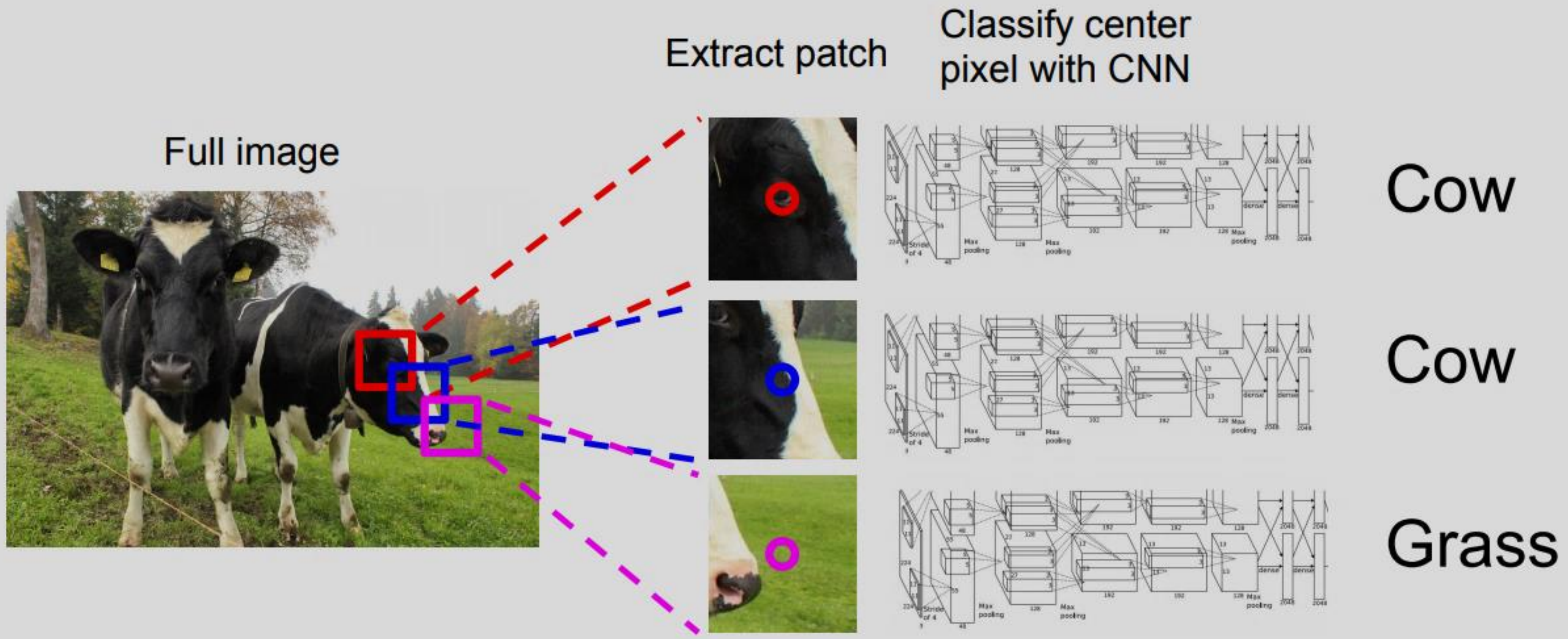
Semantic Segmentation

Label each pixel in the image with a category label

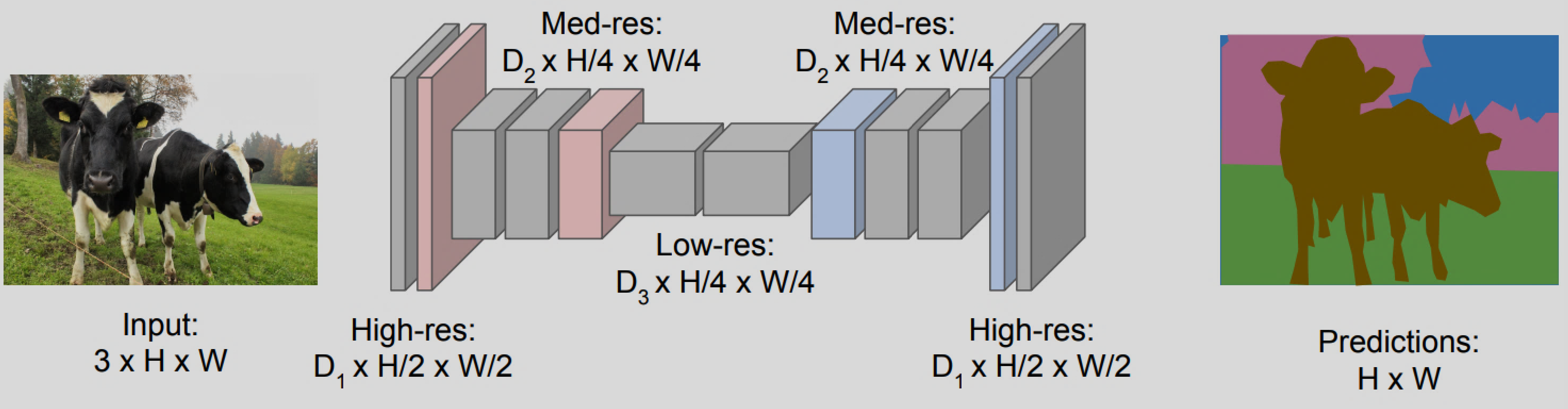
Don’t differentiate instances, only care about pixels



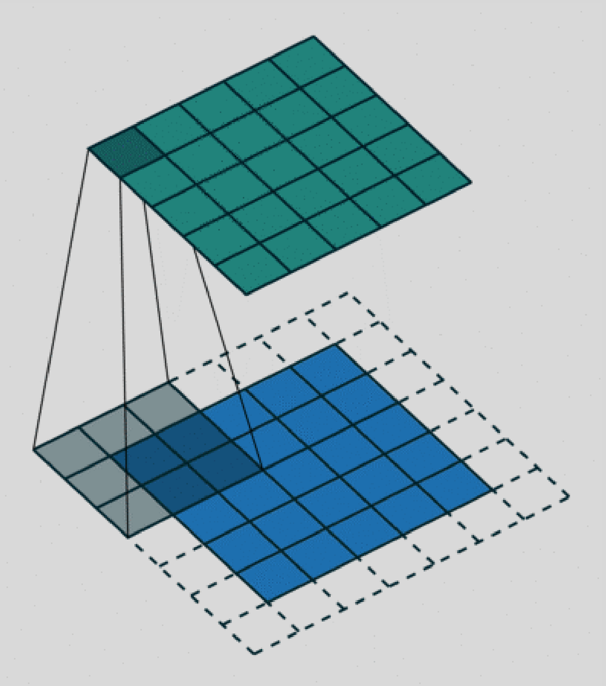
Semantic Segmentation



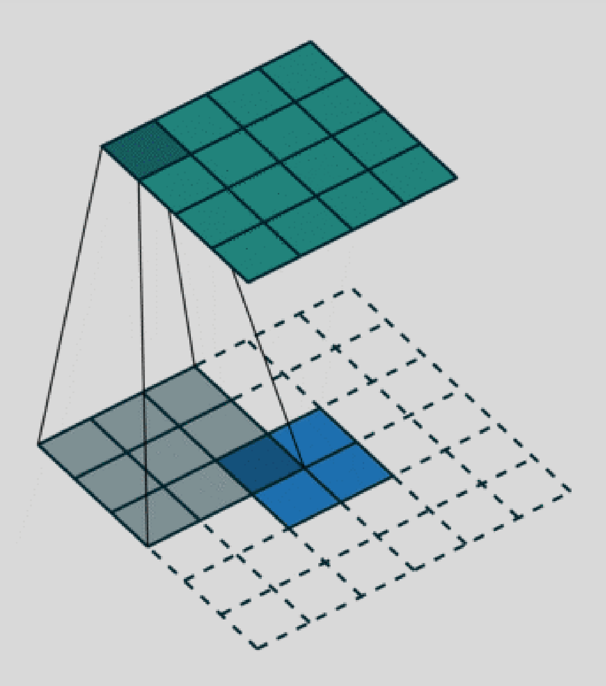
Semantic Segmentation



Transposed Convolution



Convolution operation

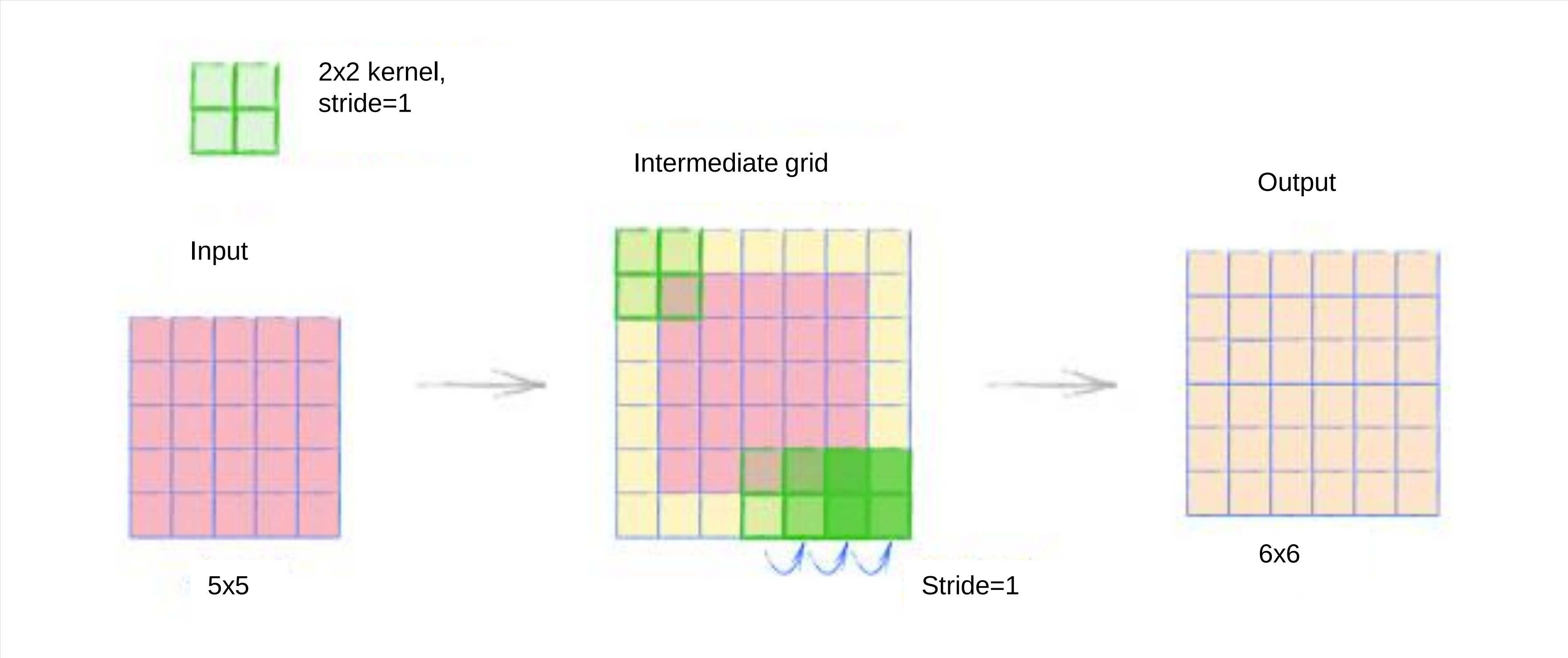


Transposed convolution operation

Transposed Convolution

Transposed Convolution with 0 padding, stride 1, 2x2 kernel:

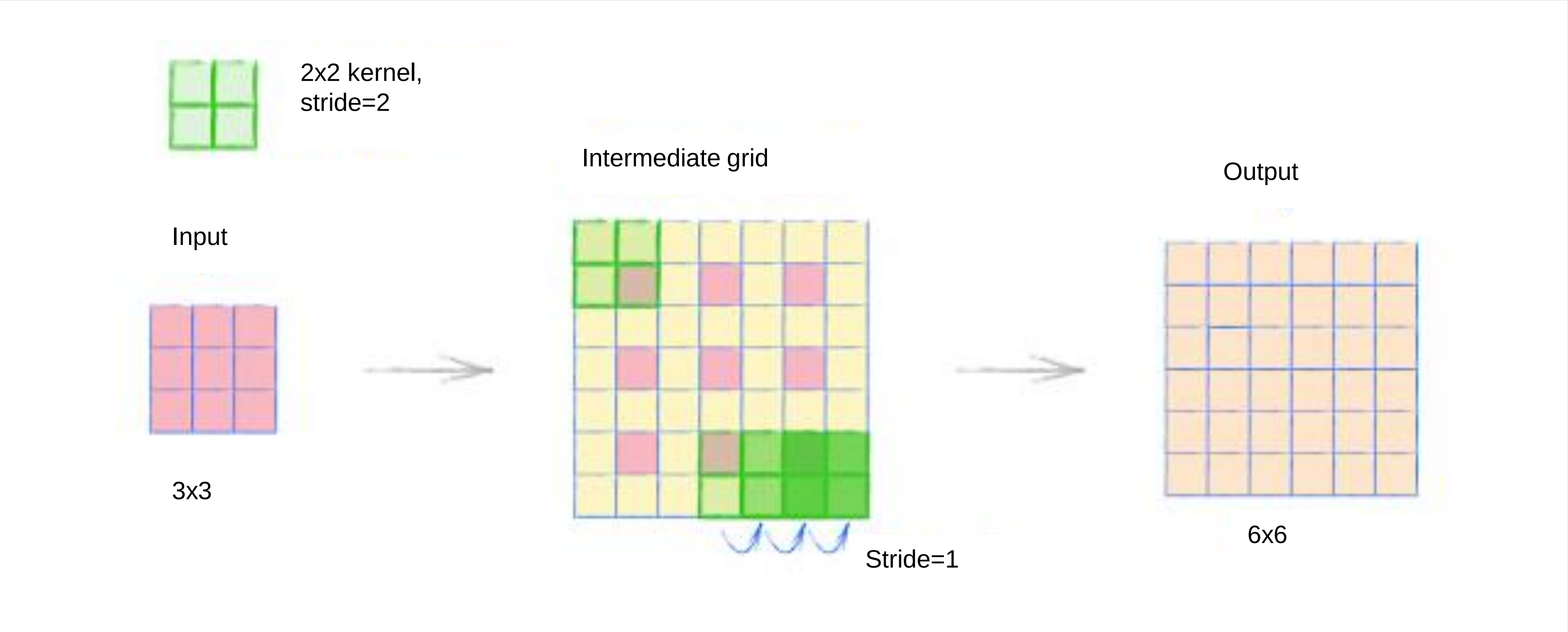
Output_size = (input_size-1)*stride – 2*padding + kernel_size + output_padding



Transposed Convolution

Transposed Convolution with 0 padding, stride 2, 2x2 kernel:

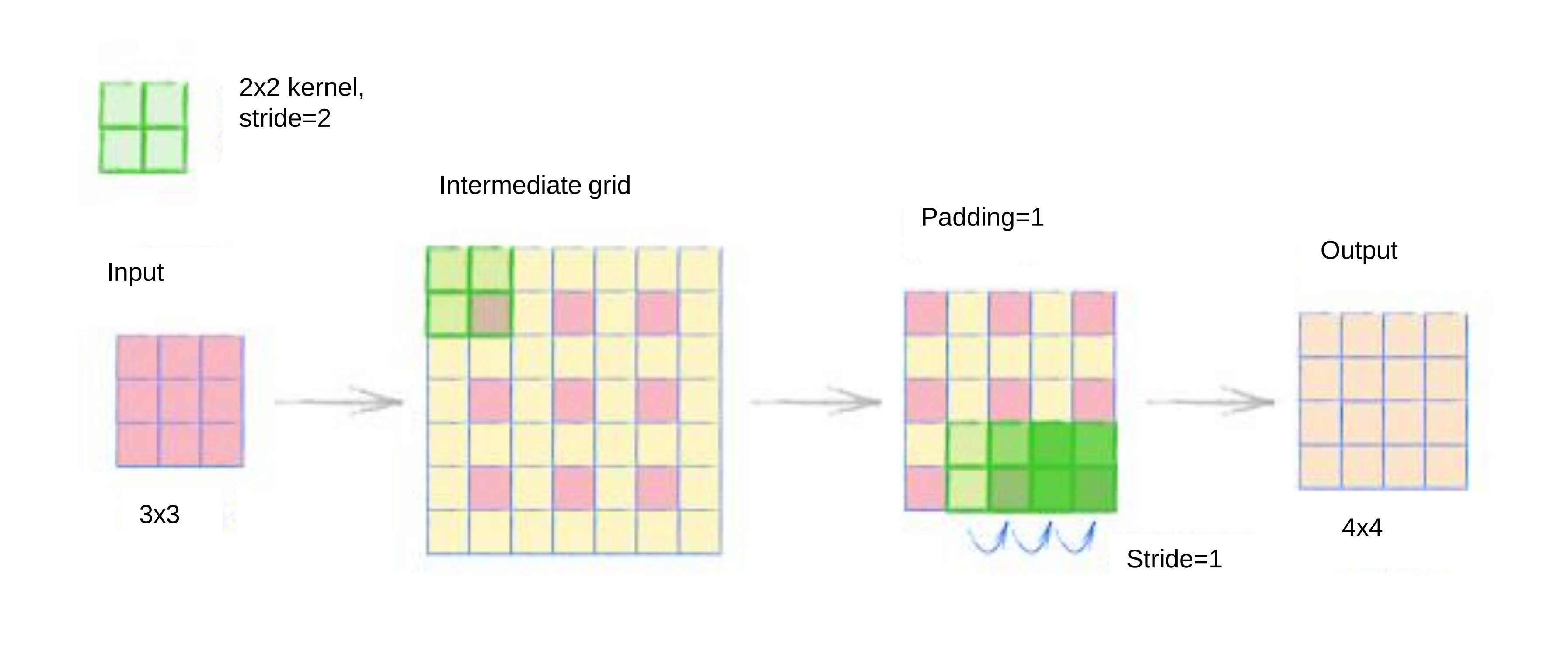
Output_size = (input_size-1)*stride – 2*padding + kernel_size + output_padding



Transposed Convolution

Transposed Convolution with 1 padding, stride 2, 2x2 kernel:

Output_size = (input_size-1)*stride – 2*padding + kernel_size + output_padding



Conv. operation

4	5	8	7
1	8	8	8
3	6	6	4
6	5	7	8

4x4 Input

1	4	1
1	4	3
3	3	1

3x3 kernel

122	148
126	134

2x2 Output

Conv. operation

4	5	8	7
1	8	8	8
3	6	6	4
6	5	7	8

4x4 Input

4
5
8
7
1
8
8
8
3
6
6
4
6
5
7
8

16x1 Input

1	4	1	0	1	4	3	0	3	3	1	0	0	0	0	0
0	1	4	1	0	1	4	3	0	3	3	1	0	0	0	0
0	0	0	0	1	4	1	0	1	4	3	0	3	3	1	0
0	0	0	0	0	1	4	1	0	1	4	3	0	3	3	1

4x16 Conv. Kernel matrix

Conv. operation

1	4	1	0	1	4	3	0	3	3	1	0	0	0	0	0
0	1	4	1	0	1	4	3	0	3	3	1	0	0	0	0
0	0	0	0	1	4	1	0	1	4	3	0	3	3	1	0
0	0	0	0	0	1	4	1	0	1	4	3	0	3	3	1

4x16 Conv. Kernel matrix

4
5
8
7
1
8
8
8
3
6
6
4
6
5
7
8

16x1 Input

122
148
126
134

4x1 Output

Transposed Conv. operation

1	0	0	0
4	1	0	0
1	4	0	0
0	1	0	0
1	0	1	0
4	1	4	1
3	4	1	4
0	3	0	1
3	0	1	0
3	3	4	1
1	3	3	4
0	1	0	3
0	0	3	0
0	0	3	3
0	0	1	3
0	0	0	1

16x4
Conv. Kernel Matrix

2
1
4
4

4x1 Input

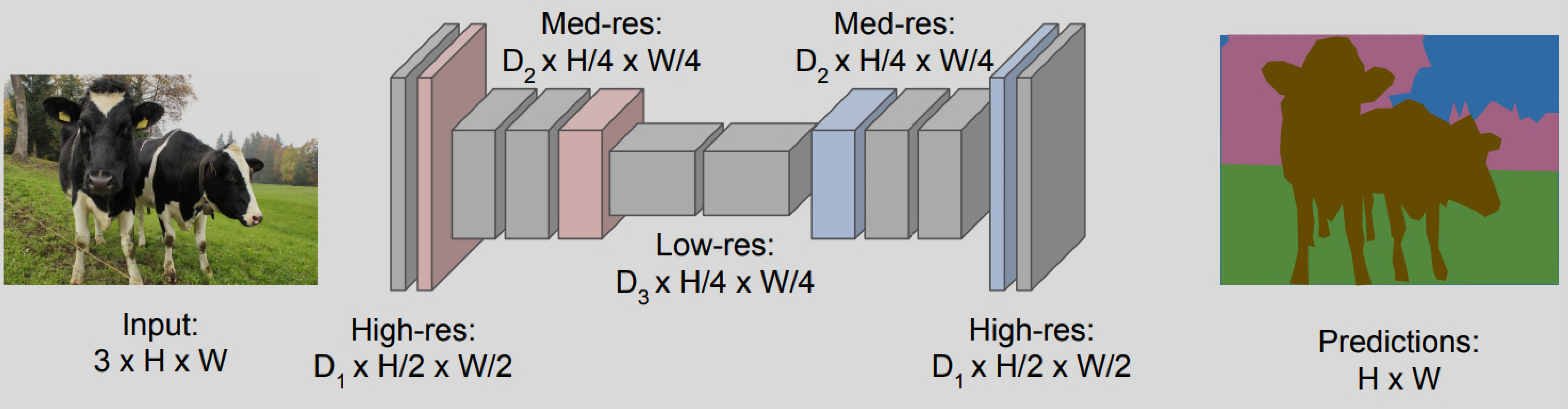
2
9
6
1
6
29
30
7
10
29
33
13
12
24
16
4

16x1 Output

2	9	6	1
6	29	30	7
10	29	33	13
12	24	16	4

4x4 Output

Achievable by differentiable layers



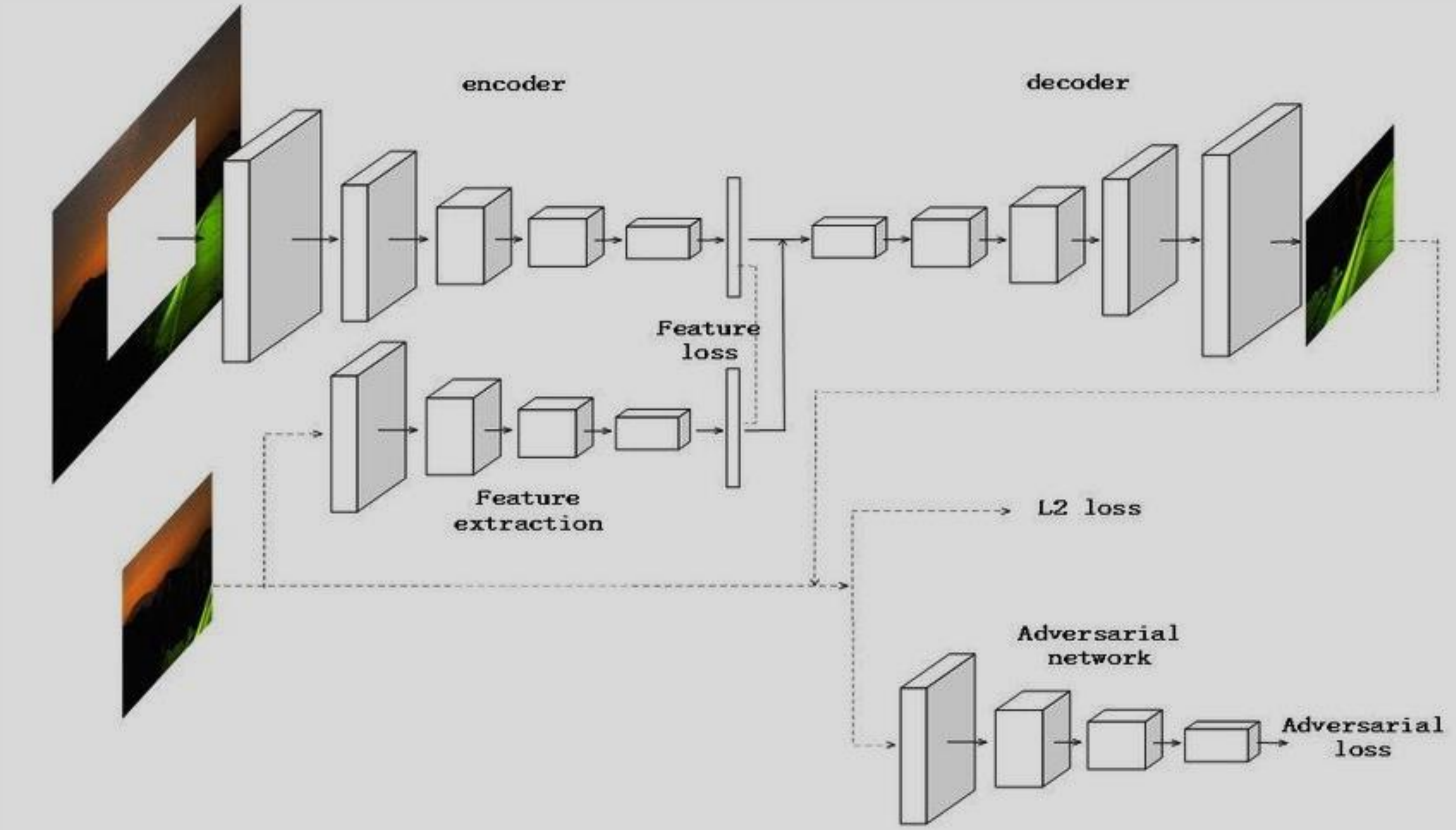
16x4
Conv. Kernel Matrix

Achievable by differentiable layers

(a) Input context (b) Human artist (c) Context Encoder ($L2$ loss) (d) Context Encoder ($L2$ + Adversarial loss)

16x4
Conv. Kernel Matrix

Achievable by differentiable layers



16x4
Conv. Kernel Matrix

Next class…

– Review on the PyTorch.

Thank you!

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

2 0 0 7