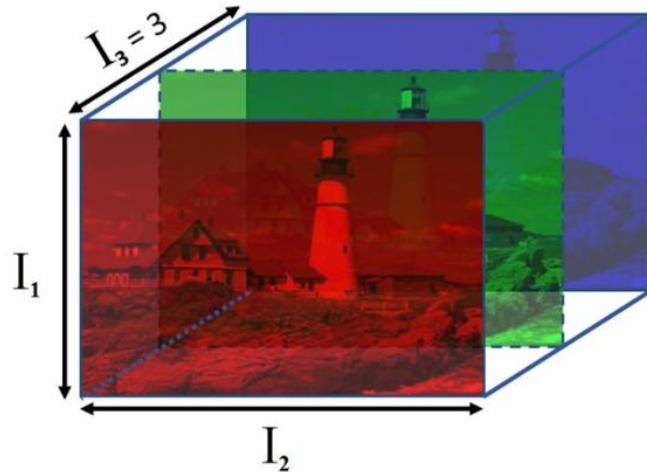


Глубокое обучение для обработки изображений

Лекция 3

Входные данные

Чаще всего изображения представляют собой тензор размерности (H , W , C). $C = 3$ для RGB, $C = 1$ для grayscale. Для RGB формата значения в каждом канале C_i лежат в диапазоне от 0 до 255, при этом $(0, 0, 0)$ соответствует черному цвету, а $(255, 255, 255)$ – белому.

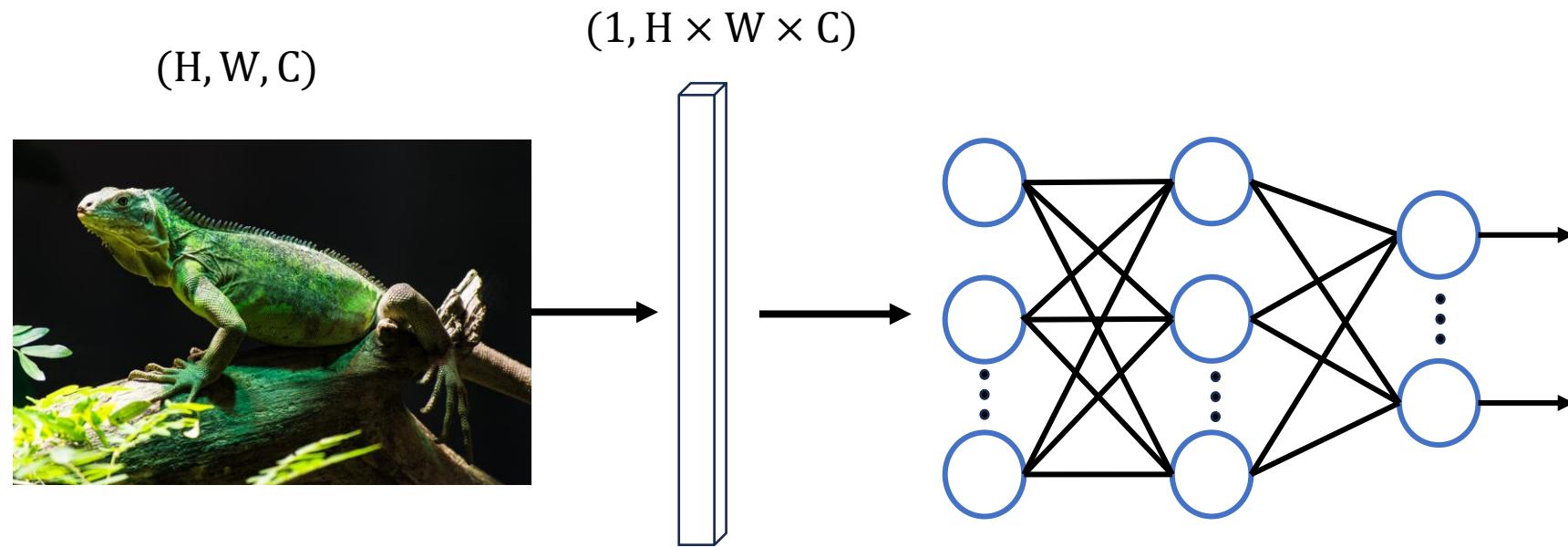


Представление
изображения в виде
тензора 3-го порядка



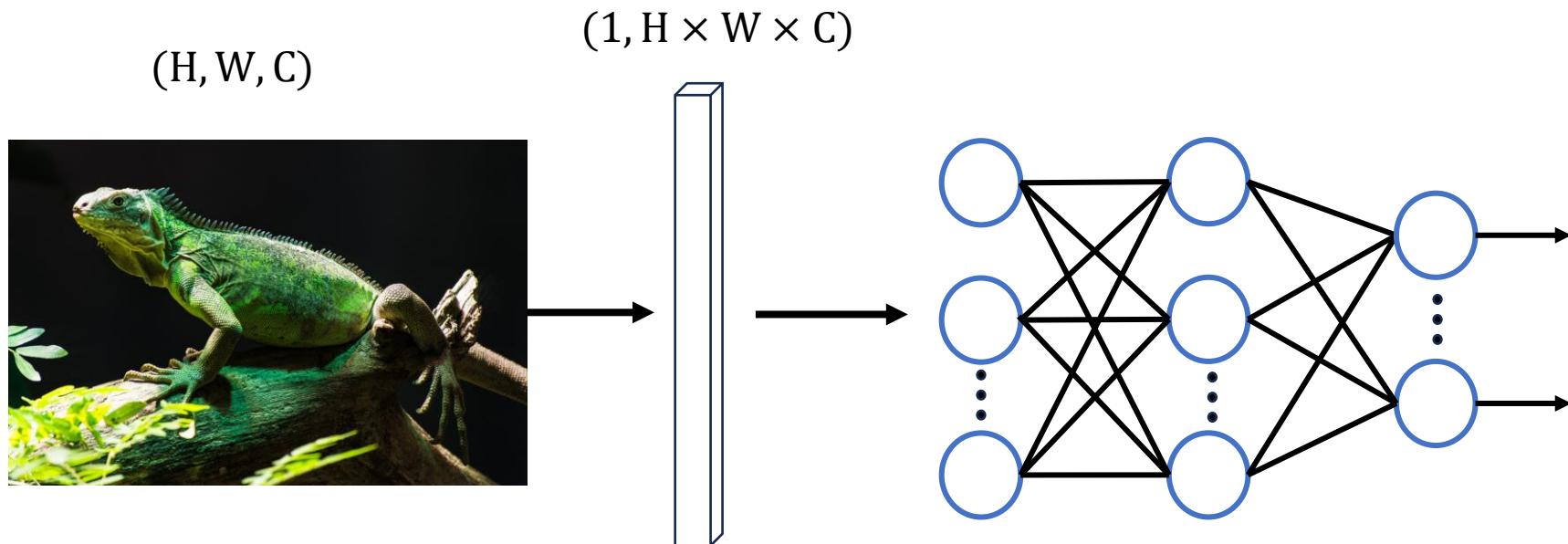
RGB (слева) и grayscale
(справа) изображения

Использование полносвязных сетей



Мы можем, например, преобразовать картинку в вектор и подать на вход нейронной сети

Использование полносвязных сетей

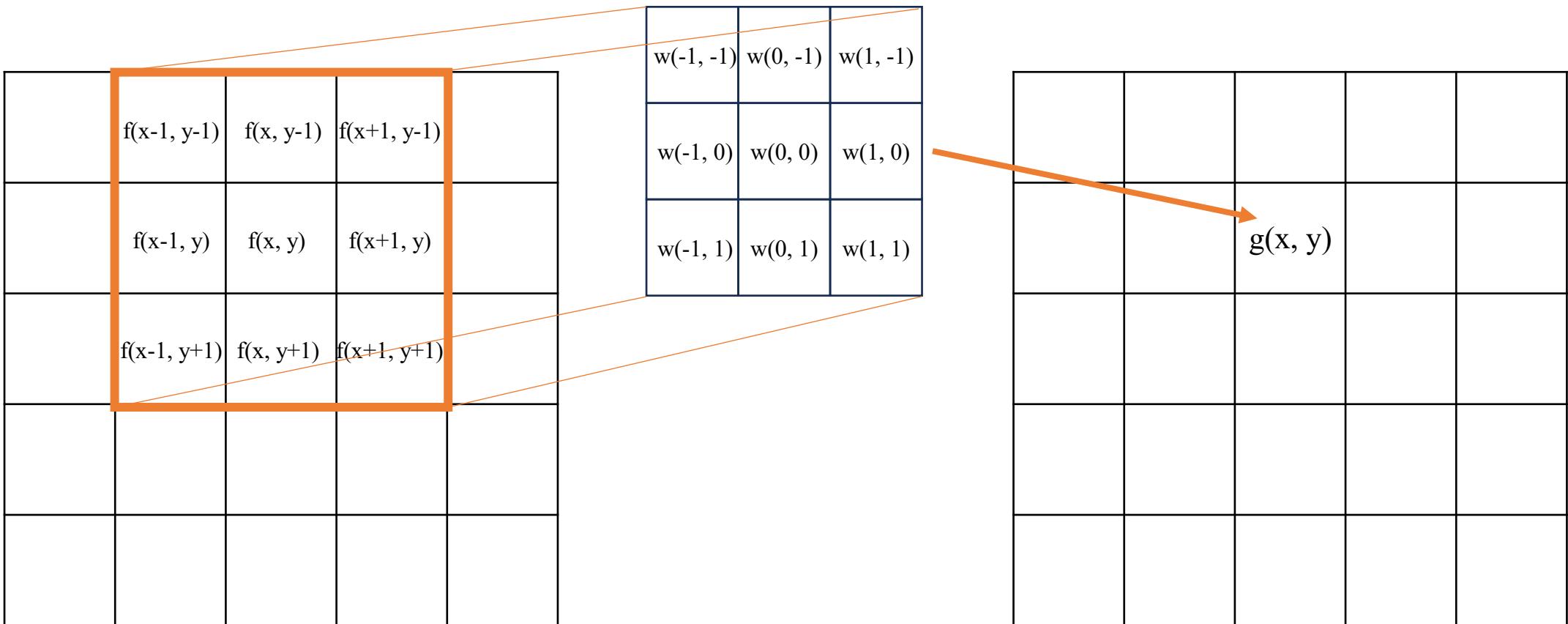


Мы можем, например, преобразовать картинку в вектор и подать на вход нейронной сети

Однако, такой подход обладает следующими недостатками:

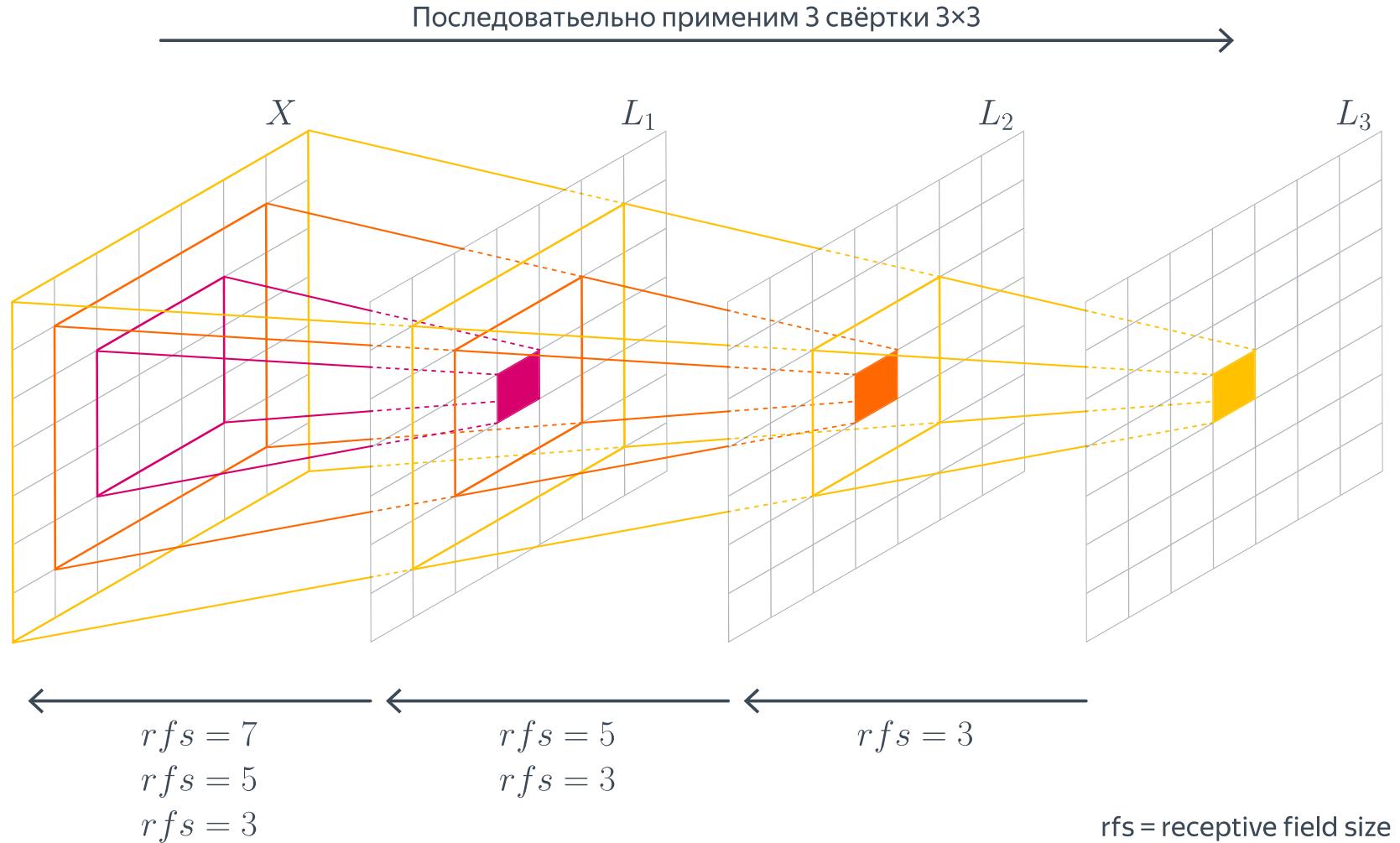
- Очень много обучаемых параметров. Размерность данных уже в первом слое будет равна $(H \times W \times C \times C_{out})$
- Использование такой архитектуры никак не учитывает структуры входного изображения

Свёрточный слой



$$g(x, y, c_{out}) = \sum_{c_{in}} \sum_{m,n} f(x + m, y + n, c_{in}) w(m, n, c_{in}, c_{out}) + b(c_{out})$$

Receptive Field



Пример нахождения свертки

3 ₀	3 ₁	2 ₂	1	0
0 ₂	0 ₂	1 ₀	3	1
3 ₀	1 ₁	2 ₂	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3 ₀	2 ₁	1 ₂	0
0	0 ₂	1 ₂	3 ₀	1
3	1 ₀	2 ₁	2 ₂	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2 ₀	1 ₁	0 ₂
0	0	1 ₂	3 ₂	1 ₀
3	1	2 ₀	2 ₁	3 ₂
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0 ₀	0 ₁	1 ₂	3	1
3 ₂	1 ₂	2 ₀	2	3
2 ₀	0 ₁	0 ₂	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0	0 ₀	1 ₁	3 ₂	1
3	1 ₂	2 ₂	2 ₀	3
2	0 ₀	0 ₁	2 ₂	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0	0	1 ₀	3 ₁	1 ₂
3	1	2 ₂	2 ₂	3 ₀
2	0	0 ₀	2 ₁	2 ₂
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0	0	1	3	1
3 ₀	1 ₁	2 ₂	2	3
2 ₂	0 ₂	0 ₀	2	2
2 ₀	0 ₁	0 ₂	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

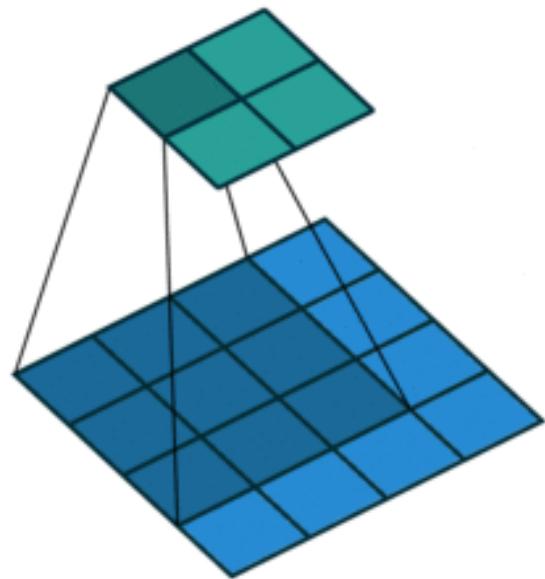
3	3	2	1	0
0	0	1 ₀	3 ₁	1
3	1 ₀	2 ₁	2 ₂	3
2	0 ₂	0 ₂	2 ₀	2
2	0 ₀	0 ₁	0 ₂	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

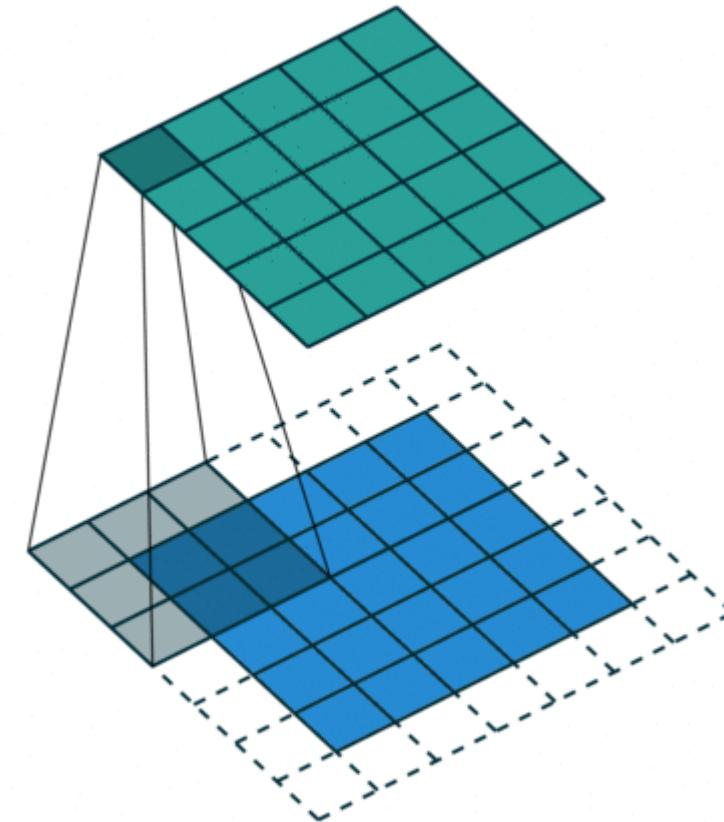
3	3	2	1	0
0	0	1	3	1
3	1	2 ₀	2 ₁	3 ₂
2	0	0 ₂	2 ₂	2 ₀
2	0	0 ₀	0 ₁	1 ₂

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

Пример нахождения свертки

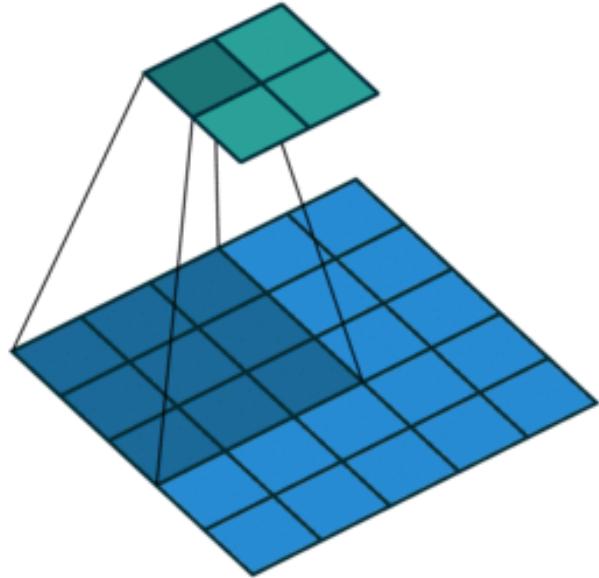


No Padding, No strides

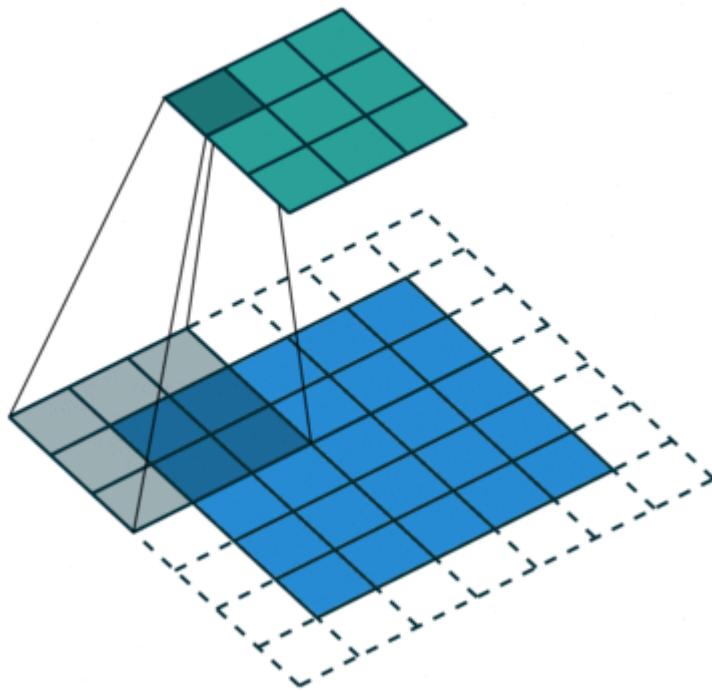


Padding, strides

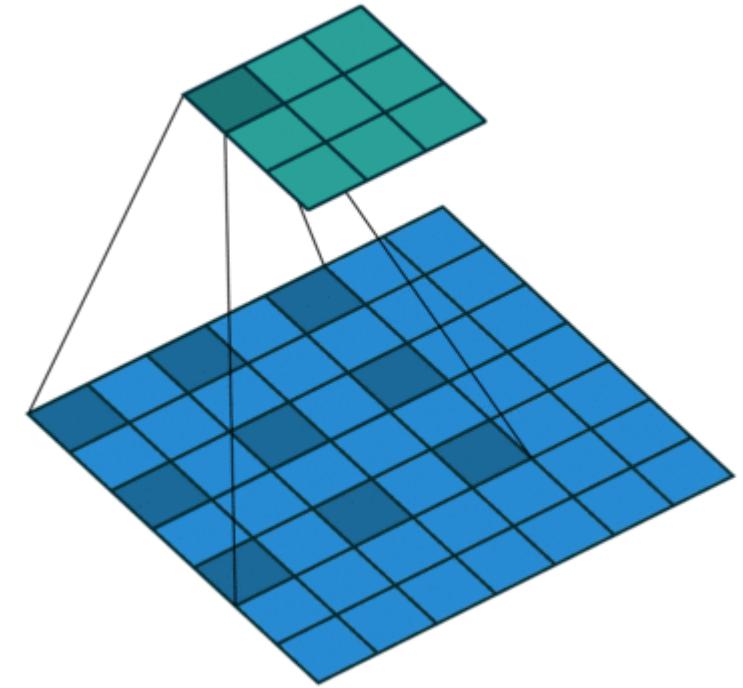
Пример нахождения свертки



No Padding, strides

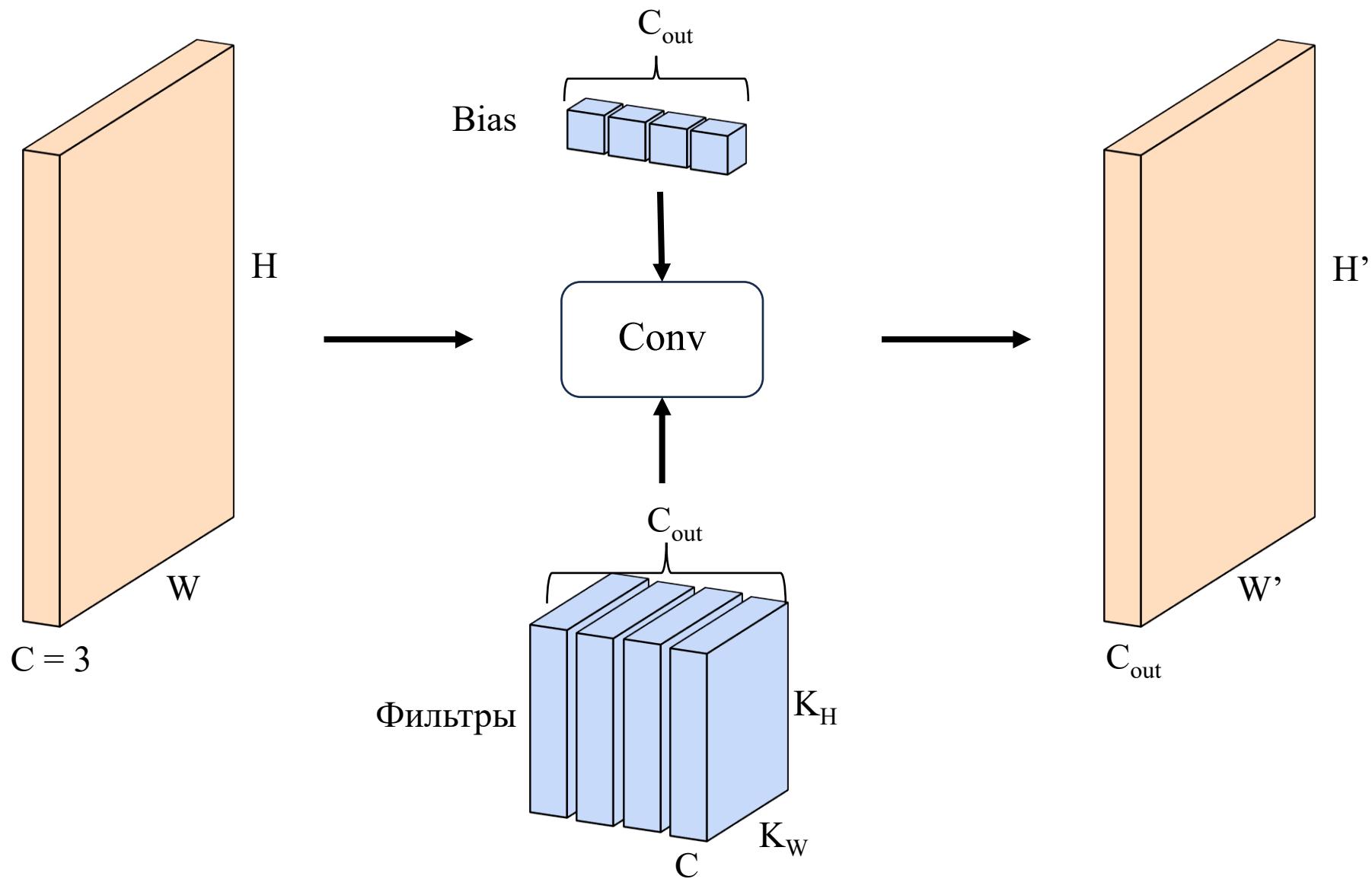


Padding, strides



No padding, no stride, dilation

Свёрточный слой



Свертка или взаимнокорреляционная функция?

CONV2D

```
CLASS torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1,  
groups=1, bias=True, padding_mode= 'zeros', device=None, dtype=None) [SOURCE]
```

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

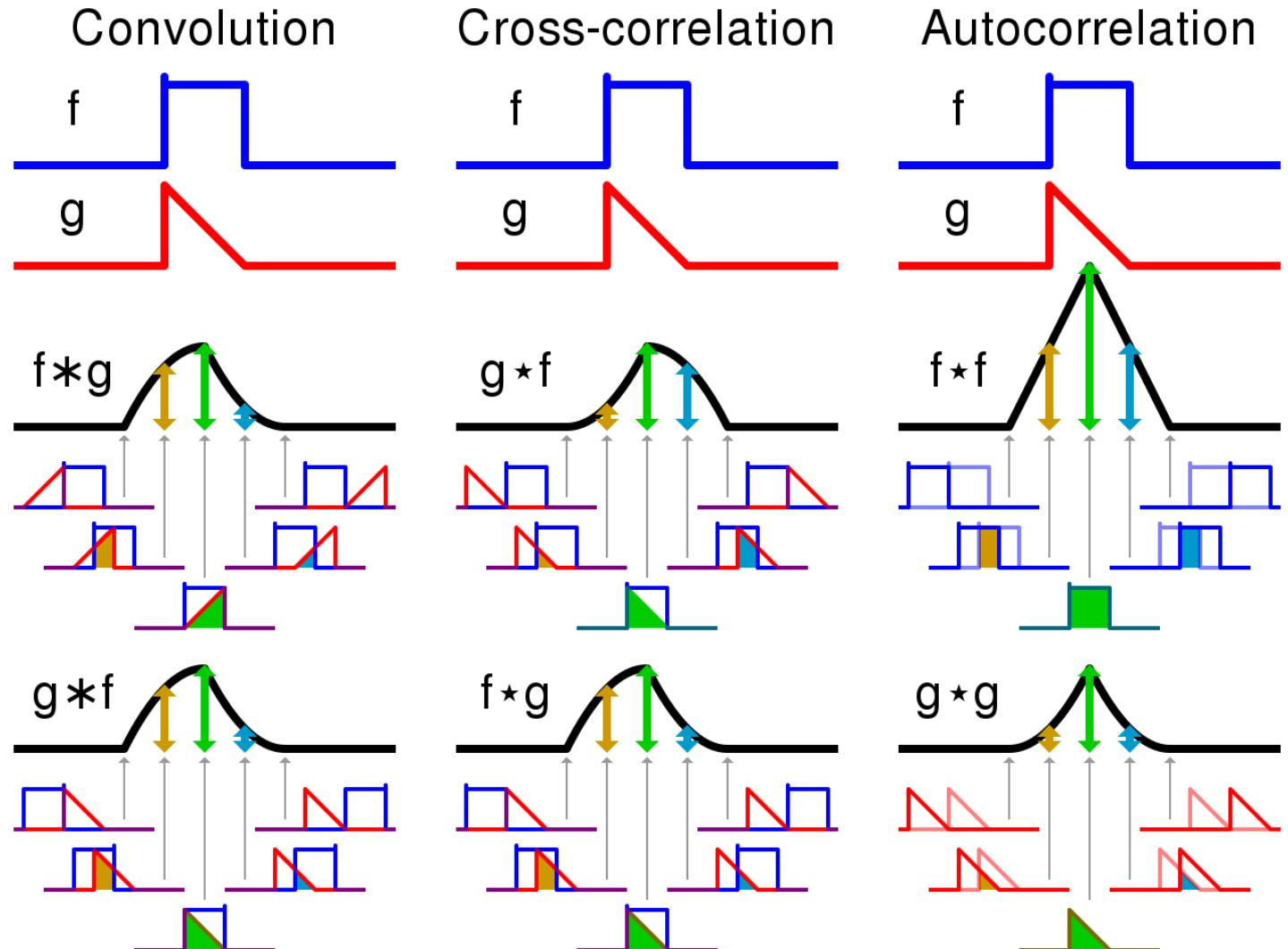
$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where \star is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

Свертка или взаимнокорреляционная функция?

$$s(t) * f(t) = \int_{-\infty}^{+\infty} s(\tau) f(t + \tau) d\tau$$

$$s(t) * f(t) = \int_{-\infty}^{+\infty} s(\tau) f(t - \tau) d\tau$$



Padding

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

Zero padding →

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

Если найти описанным образом отфильтрованное изображение, то оно будет меньшего размера, нежели исходное. Чтобы сохранить размер изображения, его дополняют пикселями по краям, например, нулями или иными значениями

Пример картинки, дополненной нулями на краях

Pooling

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

Max pool, 2x2 filter

A diagram illustrating max pooling. On the left is a 2x2 input matrix with values 2, -4, 0, 1 in the top row and 2, 2, -1, 0 in the bottom row. An arrow points to the right, where a 2x2 output matrix is shown. This output matrix has values 2 and 5 in the top row, and 2 and 4 in the bottom row. The value 2 is the maximum of the top-left 2x2 submatrix (2, -4, 0, 1), and the value 5 is the maximum of the bottom-left 2x2 submatrix (2, 2, -1, 0).

2	2
5	4

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

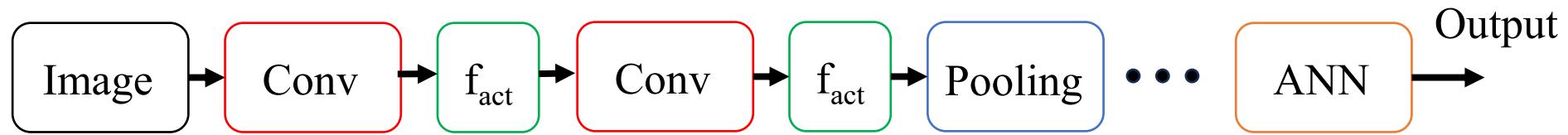
Average pool, 2x2 filter

A diagram illustrating average pooling. On the left is a 2x2 input matrix with values 2, -4, 0, 1 in the top row and 2, 2, -1, 0 in the bottom row. An arrow points to the right, where a 2x2 output matrix is shown. This output matrix has values -0.25 and 0.75 in the top row, and 1.25 and 1.75 in the bottom row. The value -0.25 is the average of the top-left 2x2 submatrix (2, -4, 0, 1), and the value 0.75 is the average of the bottom-left 2x2 submatrix (2, 2, -1, 0). The value 1.25 is the average of the top-right 2x2 submatrix (-4, 0, 1, 2), and the value 1.75 is the average of the bottom-right 2x2 submatrix (0, -1, 2, 2).

-0.25	0.75
1.25	1.75

Слой **pooling** используется для уменьшения размеров промежуточных представлений в свёрточной нейронной сети

Пример архитектуры CNN

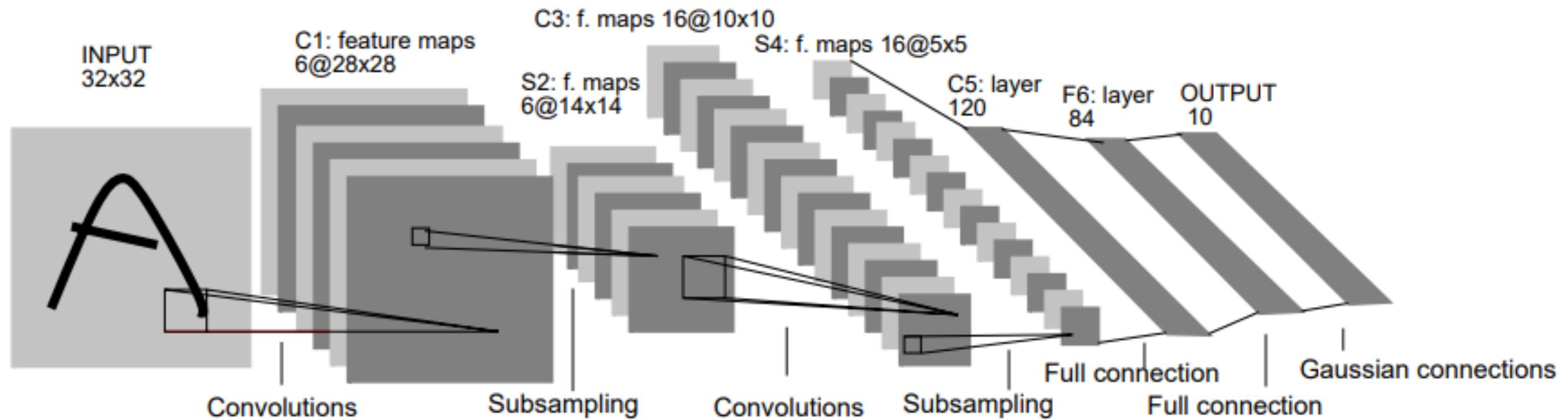


Архитектуры CNN бывают разные, но некоторые общие закономерности используются почти везде. Так, за **свёрточным слоем** идет **функция активации**; также периодически используется **pooling** слой.

В конце почти всегда присутствует **полносвязная сеть (fully connected layer)**, с помощью которой мы и получаем предсказания (в некоторых задачах, например, в сегментации, FC слой отсутствует)

LeNet (1998)

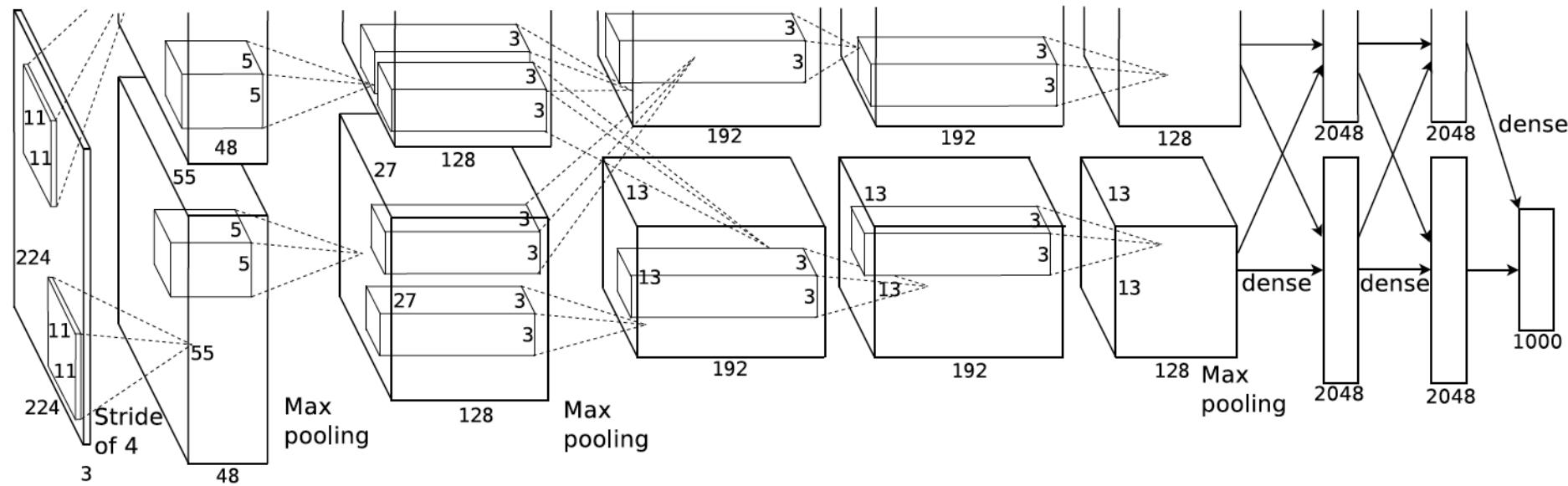
[Статья](#)



AlexNet (2012)

[Статья](#)

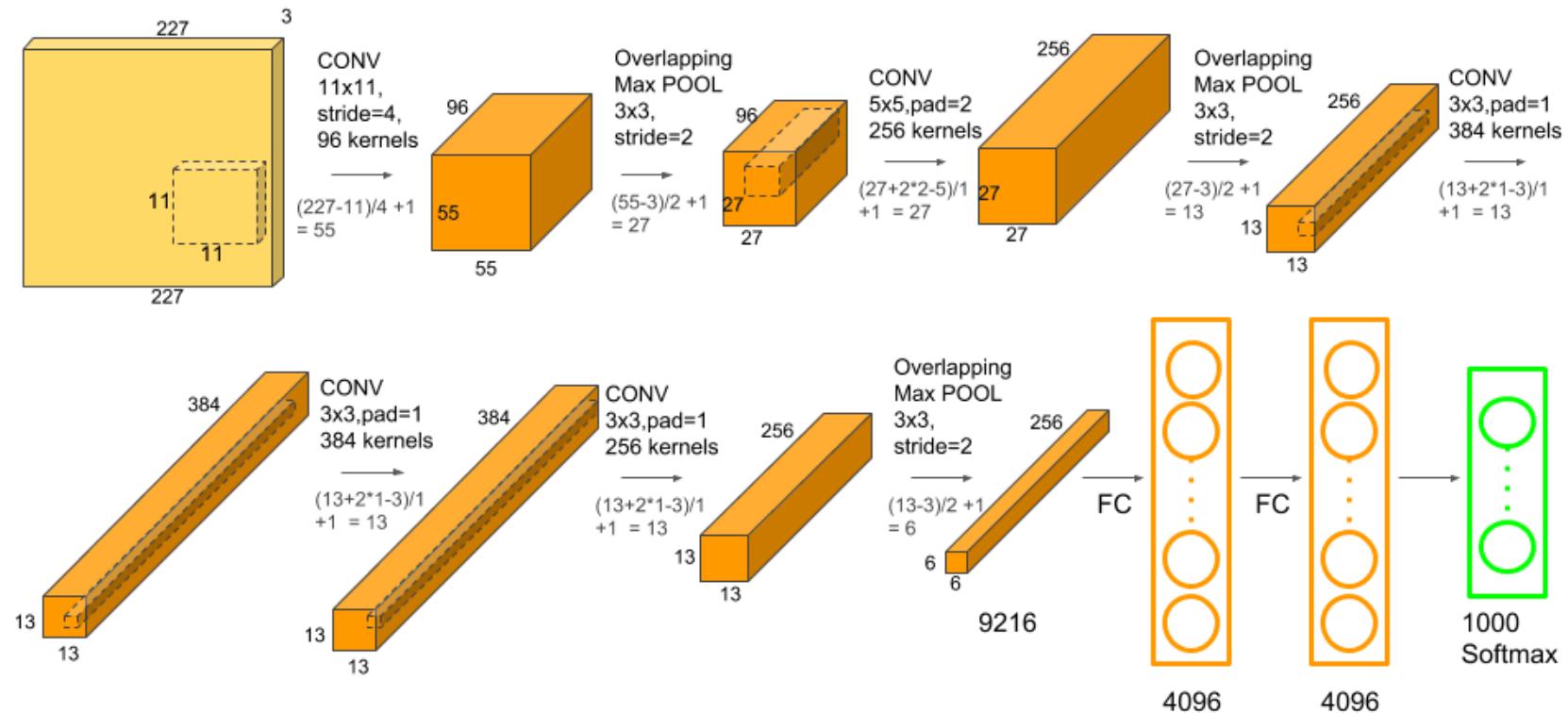
Количество параметров: ~60 миллионов



AlexNet (2012)

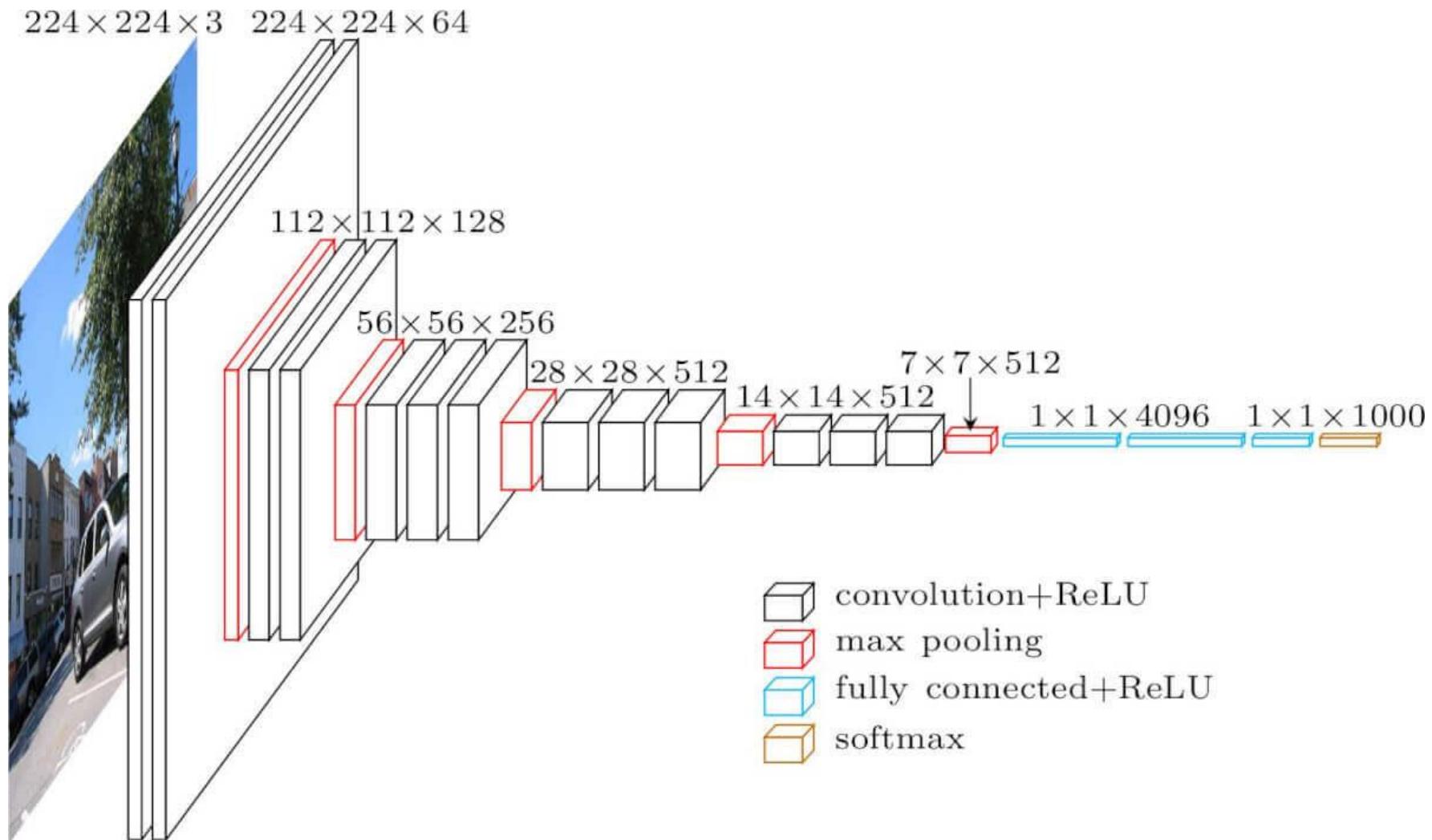
[Статья](#)

Количество параметров: ~60 миллионов



VGG16 (2014)

[Статья](#)

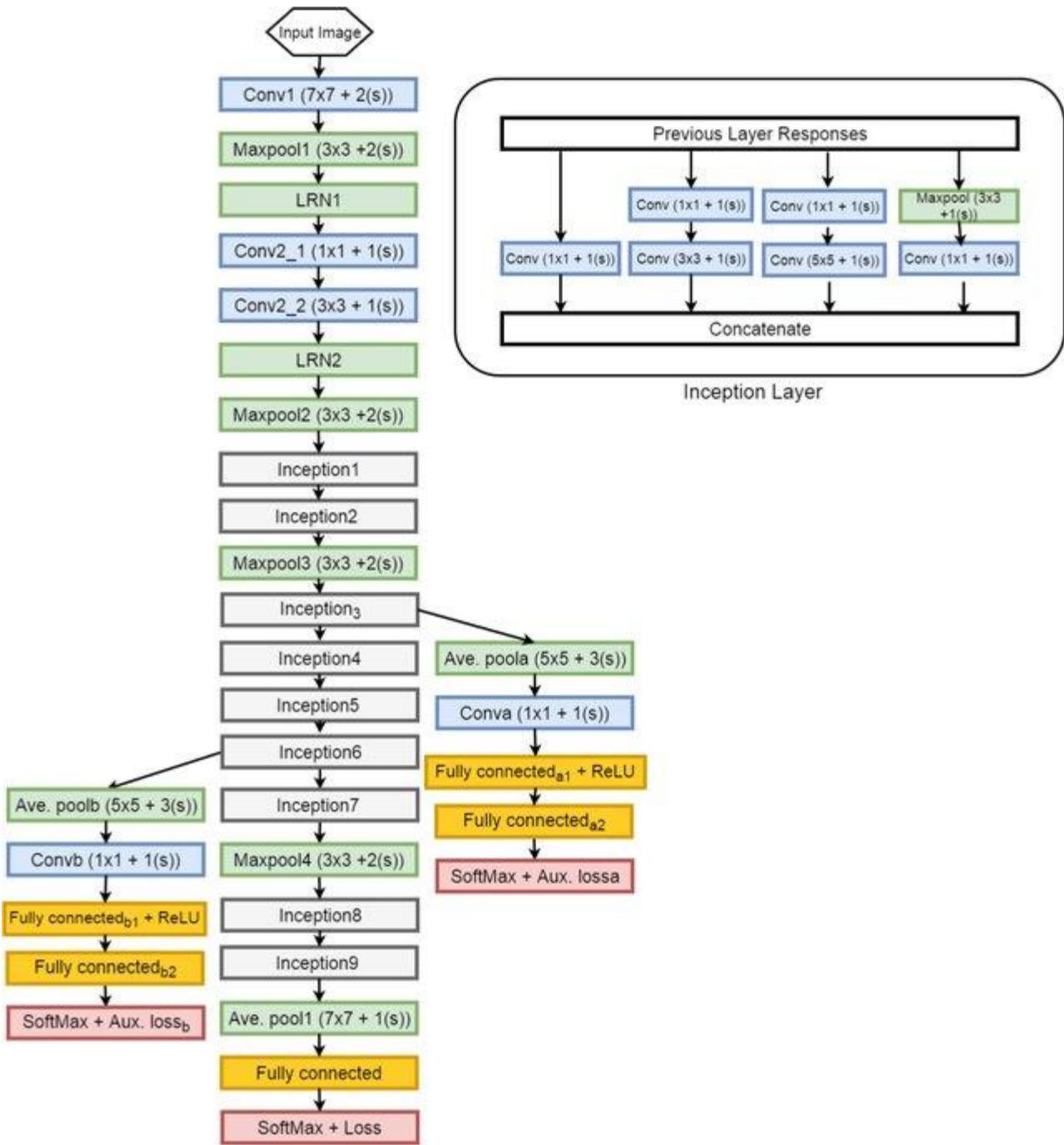


Количество параметров: ~140 миллионов

GoogleNet (Inception) (2014)

[Статья](#)

Количество параметров: ~5 миллионов



GoogleNet (Inception) (2014)

[Статья](#)

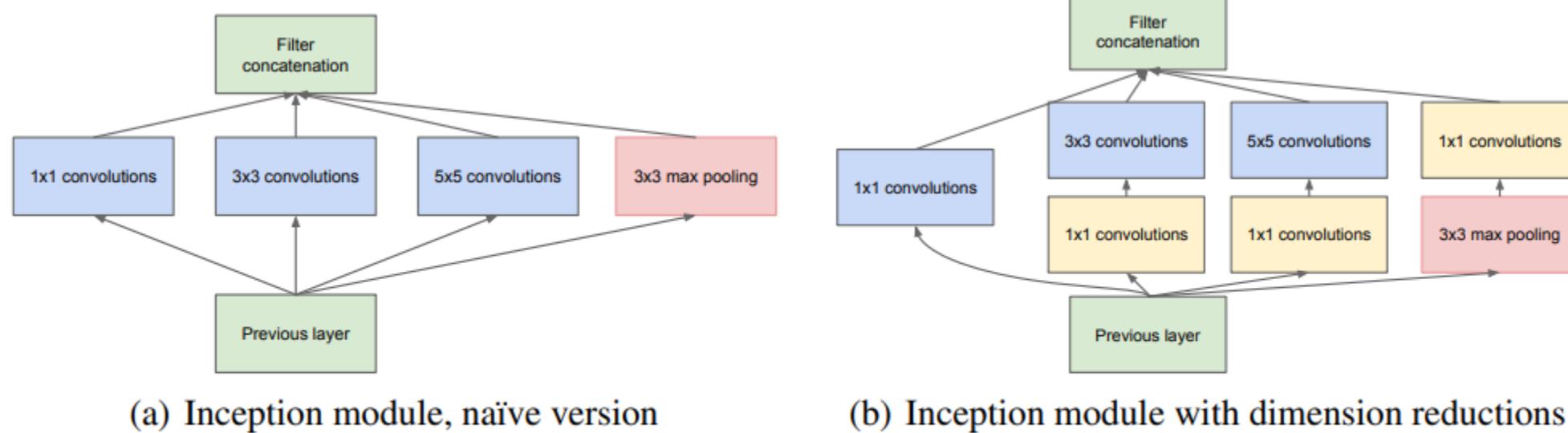


Figure 2: Inception module

ResNet (2015)

[Статья](#)

Количество параметров: ~25 миллионов
для ResNet-50

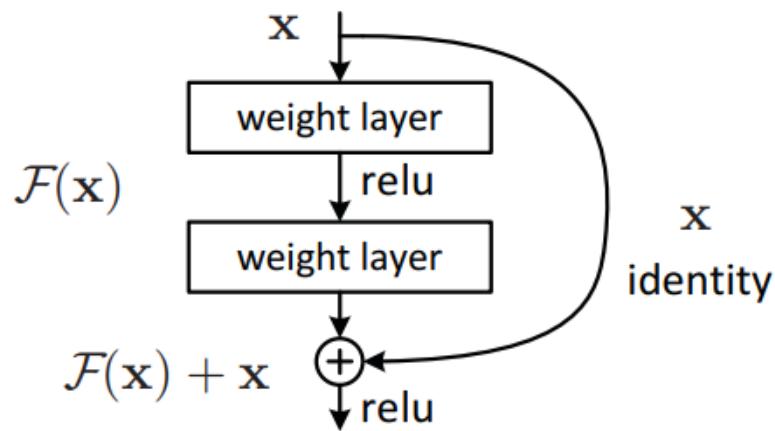
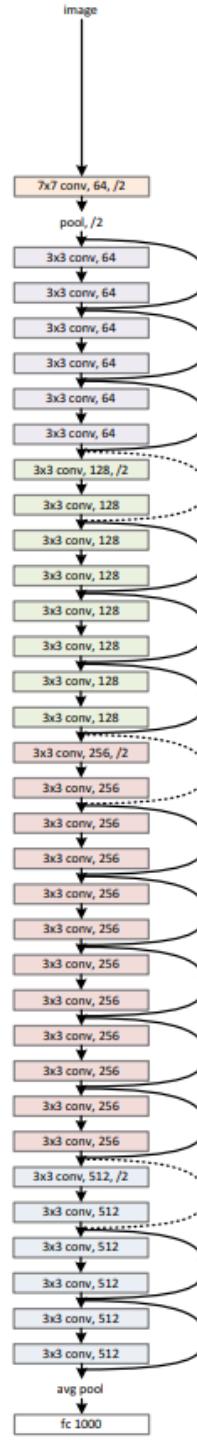


Figure 2. Residual learning: a building block.

Использование skip connections помогло преодолеть проблему **исчезающего градиента** (Vanishing gradient), что в свою очередь позволило создавать **еще более глубокие архитектуры**



EfficientNet (2019)

[Статья](#)

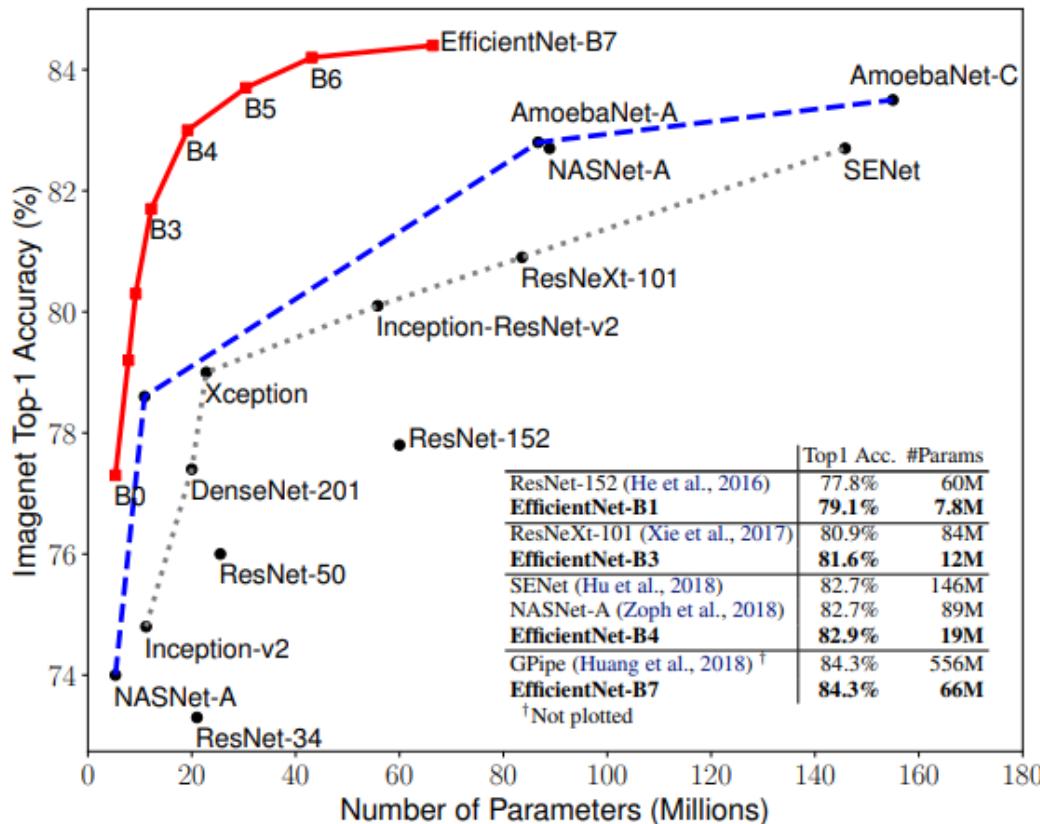


Figure 1. Model Size vs. ImageNet Accuracy. All numbers are for single-crop, single-model. Our EfficientNets significantly outperform other ConvNets. In particular, EfficientNet-B7 achieves new state-of-the-art 84.3% top-1 accuracy but being 8.4x smaller and 6.1x faster than GPipe. EfficientNet-B1 is 7.6x smaller and 5.7x faster than ResNet-152. Details are in Table 2 and 4.