курс «Глубокое обучение»

Свёрточные нейронные сети

Александр Дьяконов

30 сентября 2021 года

План

Что такое изображение

Как классифицировать

Свёртка (Convolution)

Отступ (Padding) Шаг (stride)

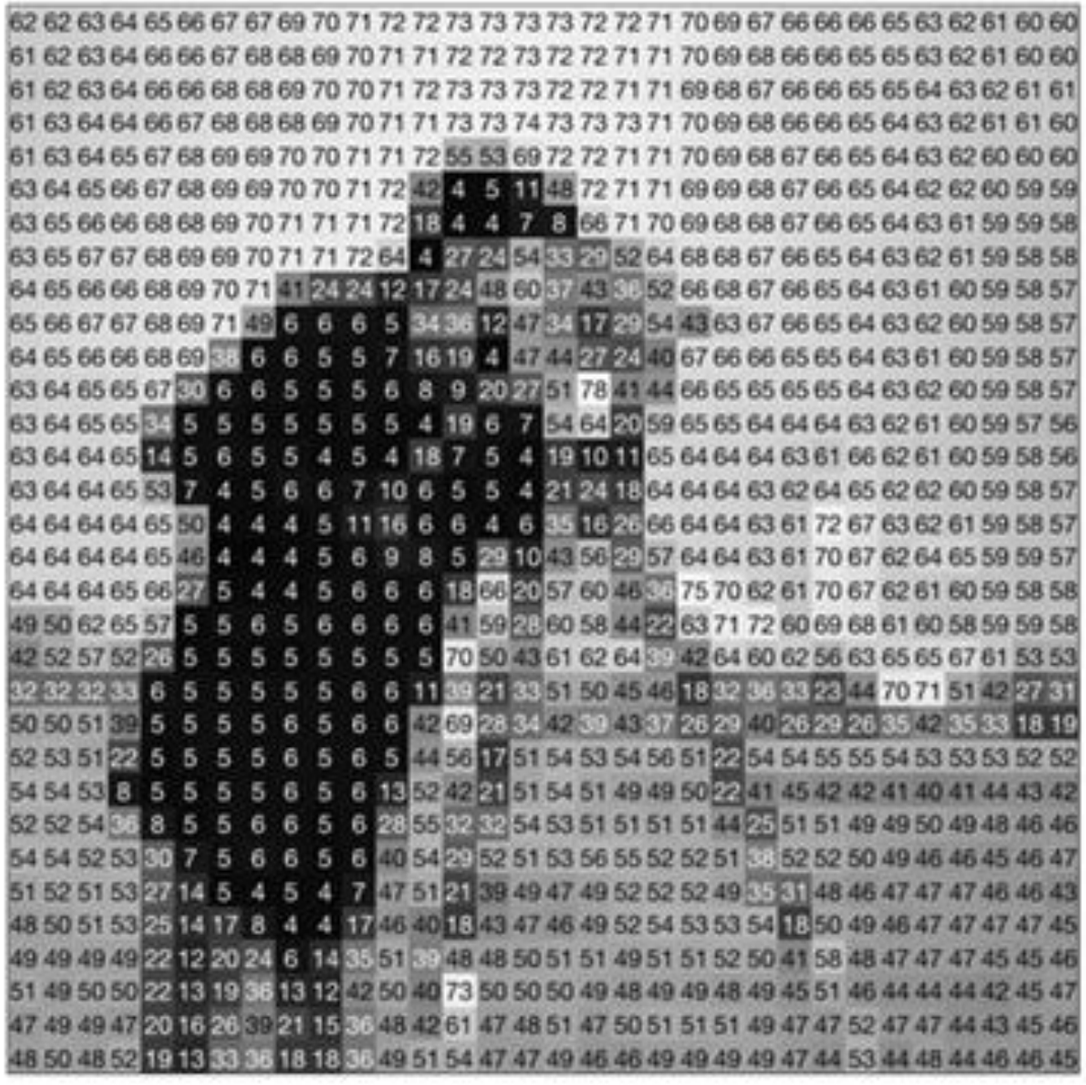
Свёрточные нейронные сети (ConvNet, CNN)

Pooling (агрегация, субдискретизация / subsampling)

Какие бывают свёртки

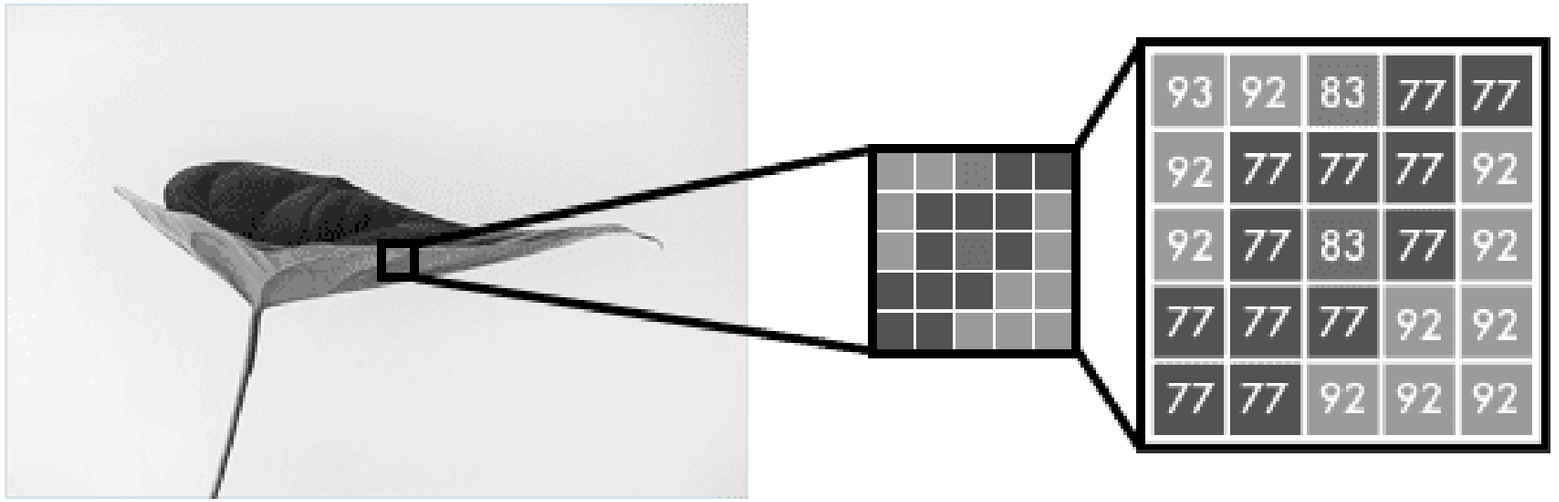
Dropout в свёрточных сетях

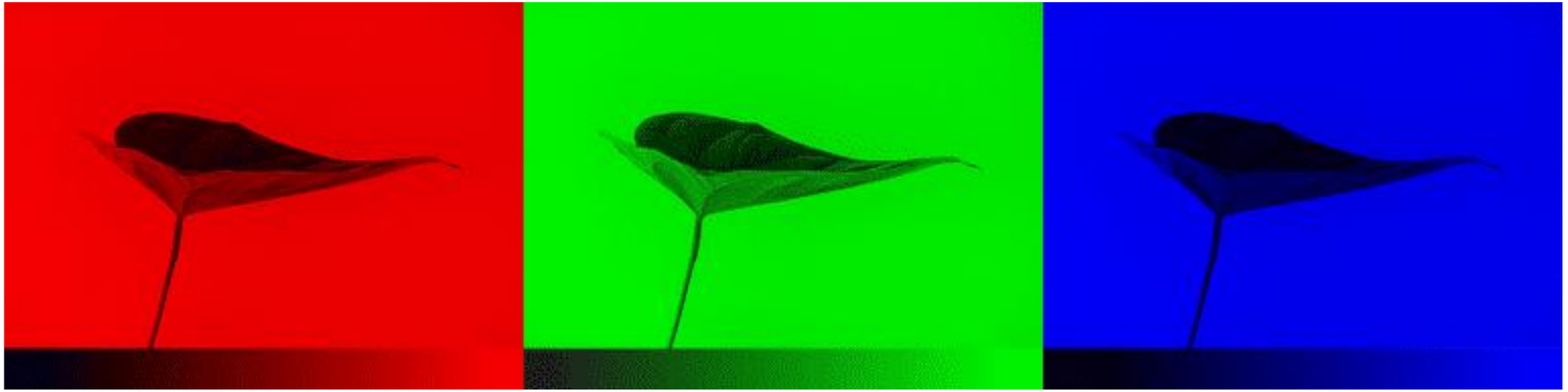
Что такое изображение – H×W-матрица



«чёрно-белое» (в градациях серого) – целочисленная матрица

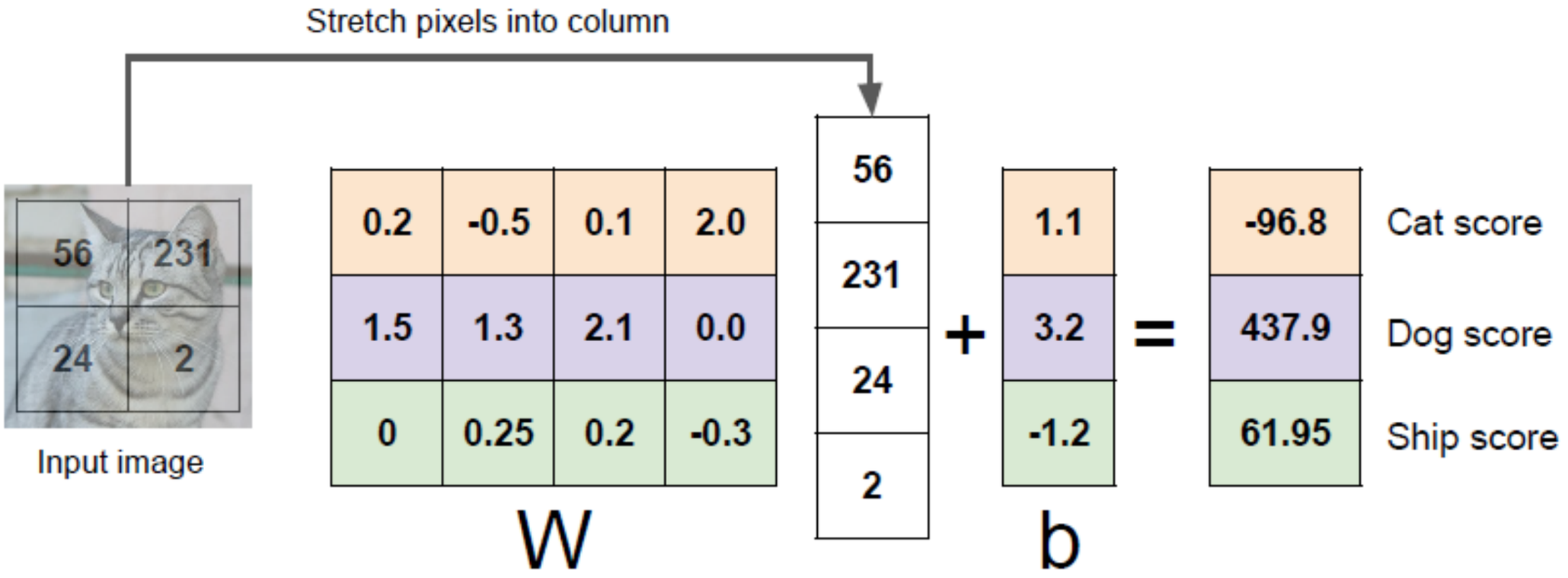
Что такое изображение – трёхмерный C×H×W -тензор





цветное – 3-х мерная целочисленная матрица (тензор)

Линейный подход к классификации на несколько классов

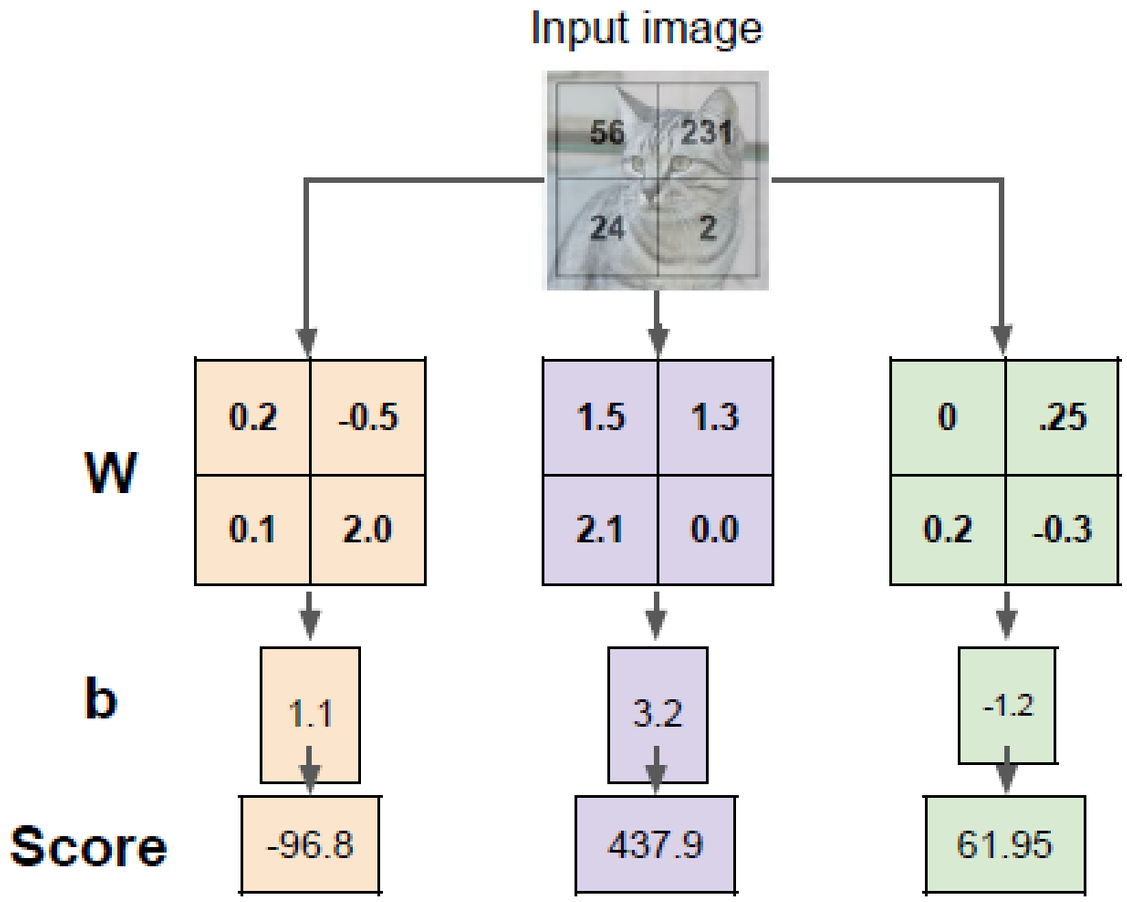


Изображение → вытянуть в вектор признаков

3 класса = 3 вектора весов

линейно получаем оценки за классы (класс по max оценке)

Линейный подход к классификации на несколько классов



Вектора весов можно потом опять «прорешейпить» в изображения:

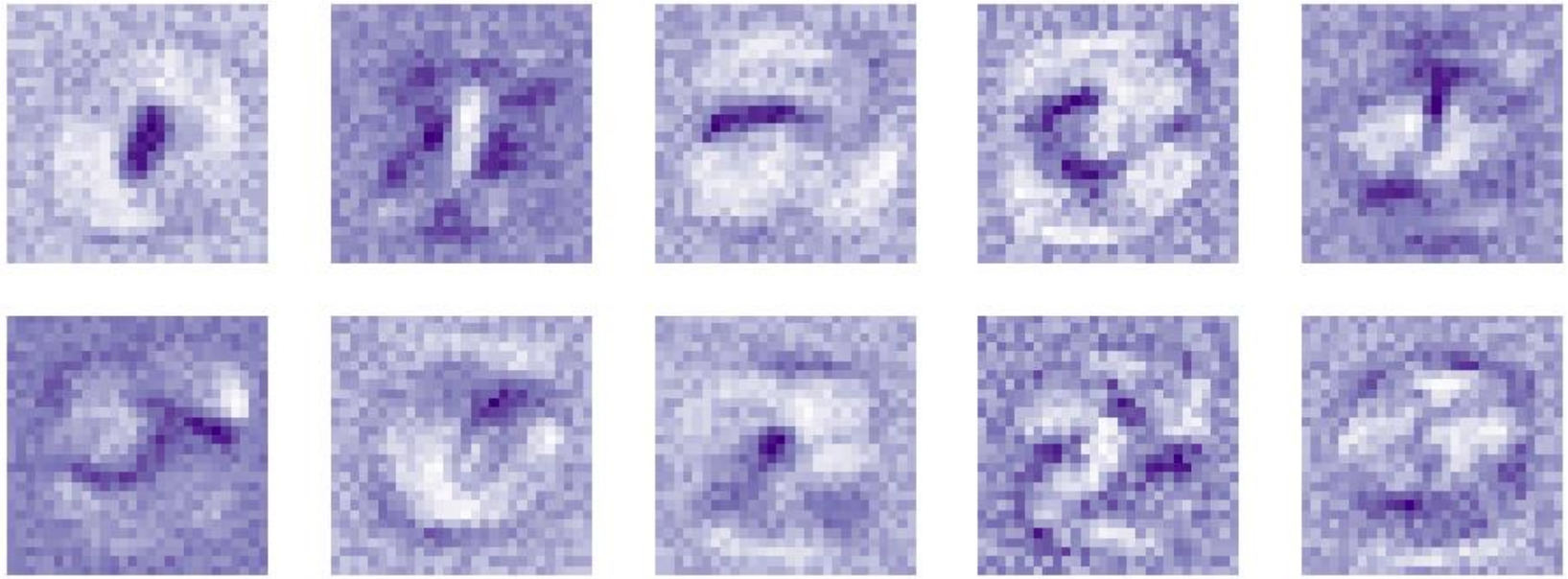
есть интерпретация – видно «как выглядят классы»

Почему такой подход к работе с изображениями не очень?

Минутка кода: наивный линейный подход

```
class Mlinear(nn.Module):
    def __init__(self, input_size, output_size):
        super(Mlinear, self).__init__()
        self.conv = nn.Conv2d(in_channels=1, out_channels=10, kernel_size=28)
        # self.fc = nn.Linear(28*28, output_size)

    def forward(self, x, verbose=False):
        x = self.conv(x)
        x = x.view(-1, 10)
        x = F.log_softmax(x, dim=1)
        return x
```



Проблемы

- **детектирование объекта в одном месте изображения**

можно решить аугментацией

но не понятно, что будет с интерпретацией и разделимостью классов

- **примитивность модели**

вряд ли подойдёт линейное правило

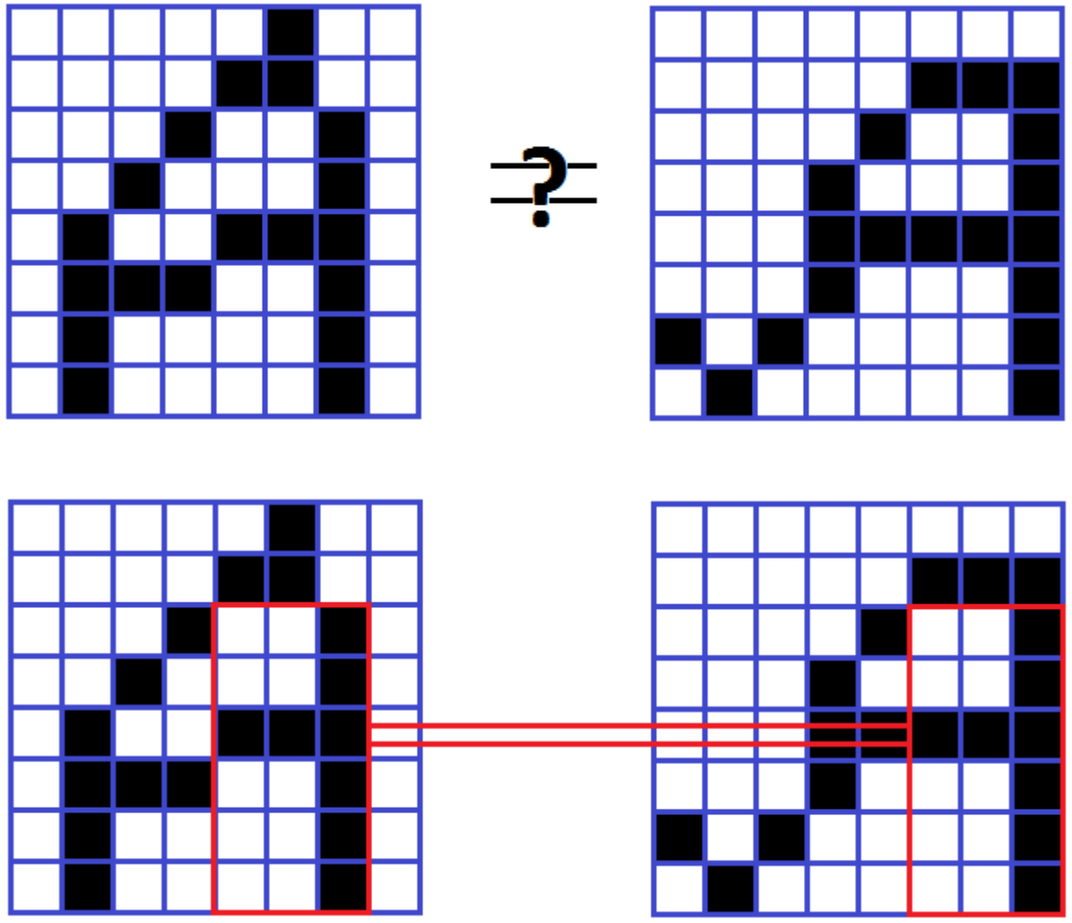
- **слишком много параметров в простой задаче!**

если изображение 256×256×3 ~ 200k,

то чтобы изображение $\rightarrow$ изображение

надо $3.9 \cdot 10^9$ параметров!

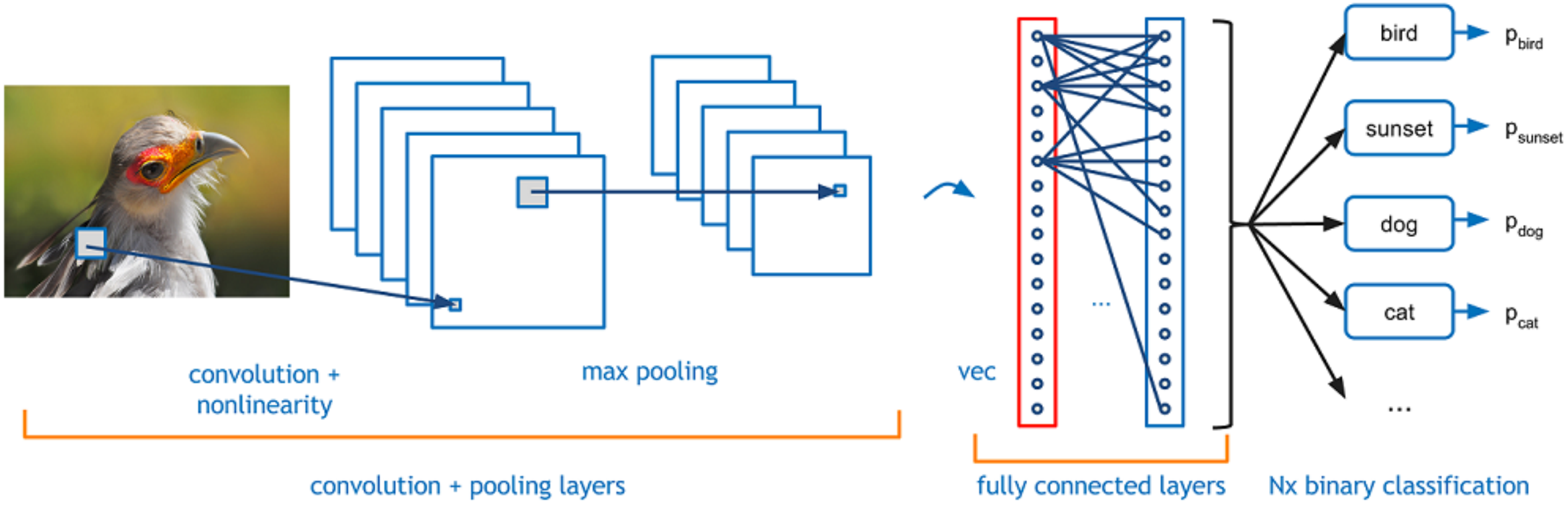
Как сравнивать изображения



хотим устойчивость к сдвигам / небольшим поворотам / сжатиям-растяжениям

хотим нахождения паттернов

Свёрточные нейронные сети (ConvNet, CNN)



– специальный вид нейронных сетей,
для обработки «равномерных сигналов»

https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

Что такое свёртка в математике

$$(f * g)(x) = \int_{\mathbb{R}^n} f(y)g(x-y)dy$$

Дискретный случай:

$$(f * g)(i) = \sum_{j \in \mathbb{Z}} f(j)g(i-j)$$

Случай, когда одна функция равна нулю за пределами окна…

свёртка коммутативна, а после этого предположения появляется асимметрия

$$(f * g)(i) = \sum_{j=-k}^{k} f(j)g(i-j) = f(-k)g(i-k) + \ldots + f(0)g(i) + \ldots + f(+k)g(i+k)$$

https://ru.wikipedia.org/wiki/Свёртка_(математический_анализ)

Что такое 1-D свёртка (Convolution)

пусть $I = (i_1, \ldots, i_n) \in \mathbb{R}^n$ **– сигнал / массив,**

$K = (k_1, \ldots, k_r) \in \mathbb{R}^r$ **– ядро свёртки, тогда свёртка:**

$$I * K = (i_1 k_1 + \ldots + i_r k_r,\ i_2 k_1 + \ldots + i_{r+1} k_r,\ \ldots,\ i_{n-r+1} k_1 + \ldots + i_n k_r) \in \mathbb{R}^{n-r+1}$$

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

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

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

Отступ (Padding)

Нулевой

-1	0	2	0	-1				
0	0	1	2	3	4	5	0	0
		-1	0	0	6	7		

Константный

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

Зеркальный

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

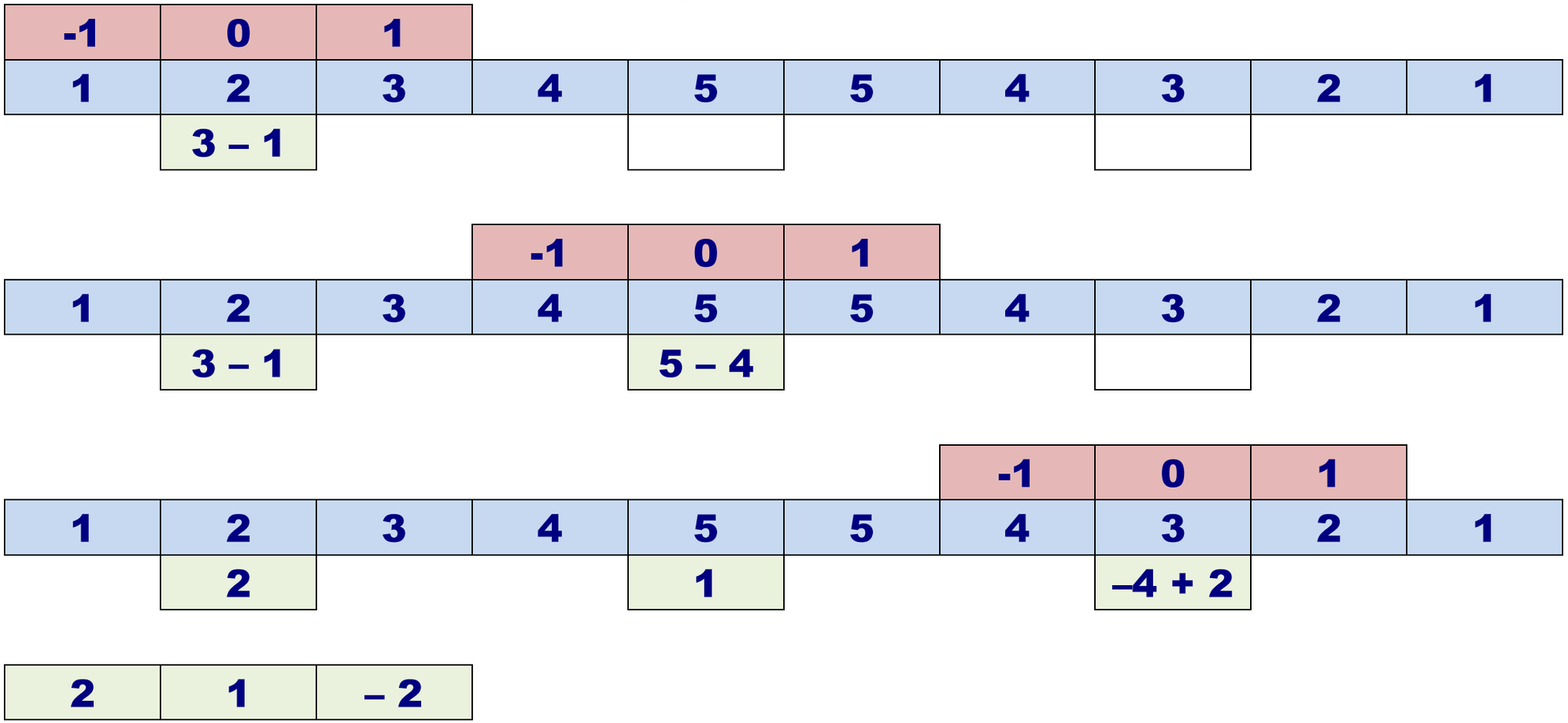
Циклический

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

позволяет получать вектор нужной длины

Шаг (stride)

свёртка с шагом 3



позволяет получать вектор небольшой длины

и делать его элементы менее коррелированными

Расширение (Dilation)

-1	0	1

свёртка с расширением 3

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

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

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

позволяет увеличить область действия свёртки («рецептивную зону»)

Минутка кода: свёртка – слой сети

```
import torch

x = torch.Tensor([1, 2, 3, 4, 5, 5, 4, 3, 2, 1]).view(1,1,10)
c = torch.nn.Conv1d(in_channels=1, out_channels=1,
                    kernel_size=3, bias=False)
c.weight = torch.nn.Parameter(torch.Tensor([-1, 0, 1]).view(1,1,3))
c(x)
tensor([[[ 2.,  2.,  2.,  1., -1., -2., -2., -2.]]],
grad_fn=<SqueezeBackward1>)

c = torch.nn.Conv1d(in_channels=1, out_channels=1,
                    kernel_size=3, bias=False, stride=3)
c(x)
tensor([[[ 2.,  1., -2.]]], grad_fn=<SqueezeBackward1>)

c = torch.nn.Conv1d(in_channels=1, out_channels=1,
                    kernel_size=3, bias=False, dilation=3)
c(x)
tensor([[[ 3.,  1., -1., -3.]]], grad_fn=<SqueezeBackward1>)
```

Для справки: в Pytorch есть слои отступов (но есть и как параметр свёртки)

Padding Layers

`nn.ReflectionPad1d`	Pads the input tensor using the reflection of the input boundary.
`nn.ReflectionPad2d`	Pads the input tensor using the reflection of the input boundary.
`nn.ReplicationPad1d`	Pads the input tensor using replication of the input boundary.
`nn.ReplicationPad2d`	Pads the input tensor using replication of the input boundary.
`nn.ReplicationPad3d`	Pads the input tensor using replication of the input boundary.
`nn.ZeroPad2d`	Pads the input tensor boundaries with zero.
`nn.ConstantPad1d`	Pads the input tensor boundaries with a constant value.
`nn.ConstantPad2d`	Pads the input tensor boundaries with a constant value.
`nn.ConstantPad3d`	Pads the input tensor boundaries with a constant value.

2-D свёртка (Convolution)

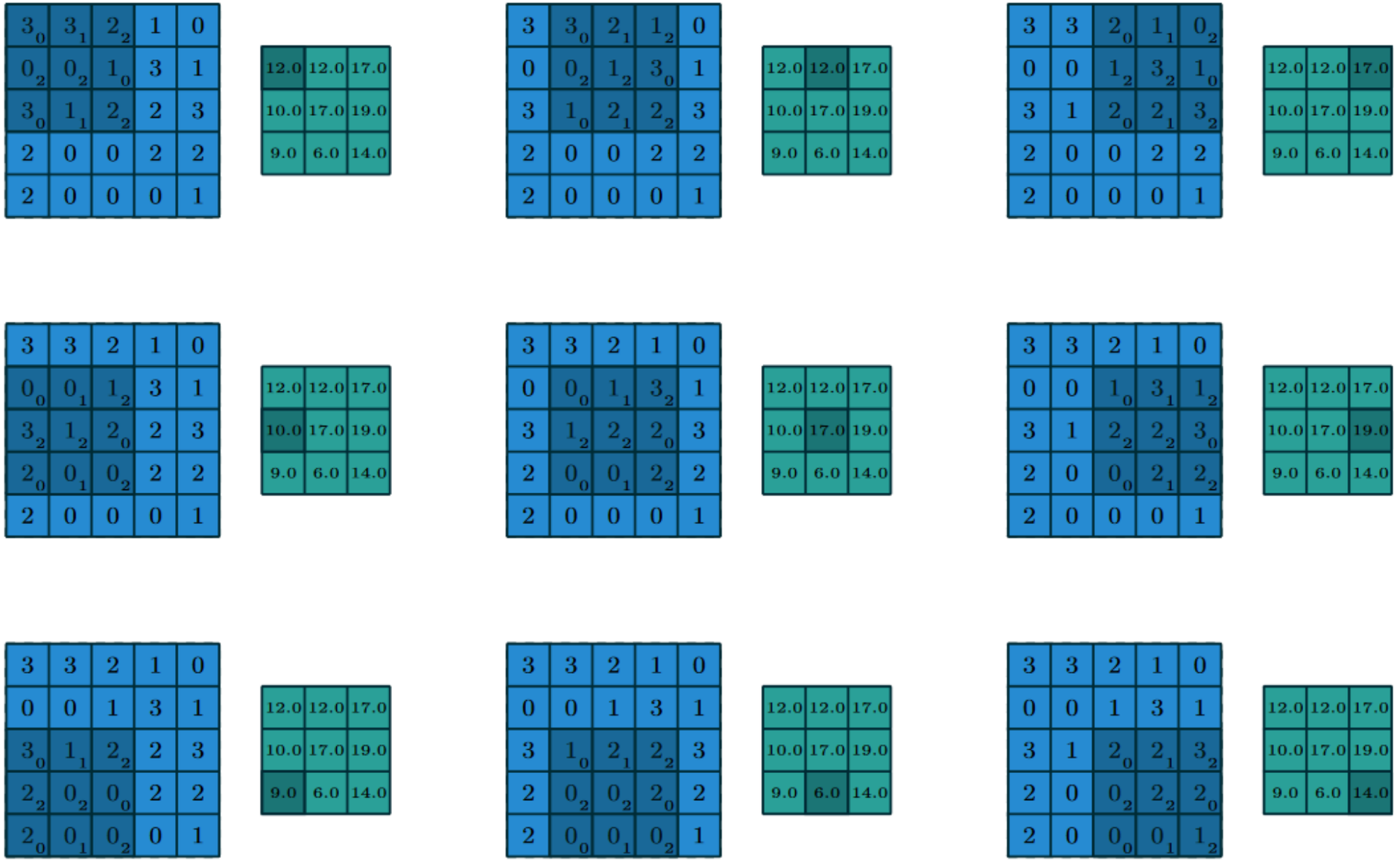
$$(I * K)_{xy} = \sum_{i=1}^{h}\sum_{j=1}^{r} K_{ij} I_{x+i-1,y+j-1}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 1 & 0 & 2 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} & \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} \\ \begin{bmatrix} 3 & 4 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} & \begin{bmatrix} 4 & 5 \\ 0 & 2 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 5 & 3 \end{bmatrix}$$

может быть немного другая индексация

хорошее объяснение: Vincent Dumoulin, Francesco Visin «A guide to convolution arithmetic for deep learning» https://arxiv.org/pdf/1603.07285.pdf

Свёртка (Convolution)



Что делает свёртка?

https://algotravelling.com/ru/%d0%bc%d0%b0%d1%88%d0%b8%d0%bd%d0%bd%d0%be%d0%b5-%d0%be%d0%b1%d1%83%d1%87%d0%b5%d0%bd%d0%b8%d0%b5/

Что делает свёртка?

исследует локальные участки изображения – ищет паттерны

Что делает свёртка?

Фильтры в CV

- устраняют шум
- находят границы
- детектируют текстуры

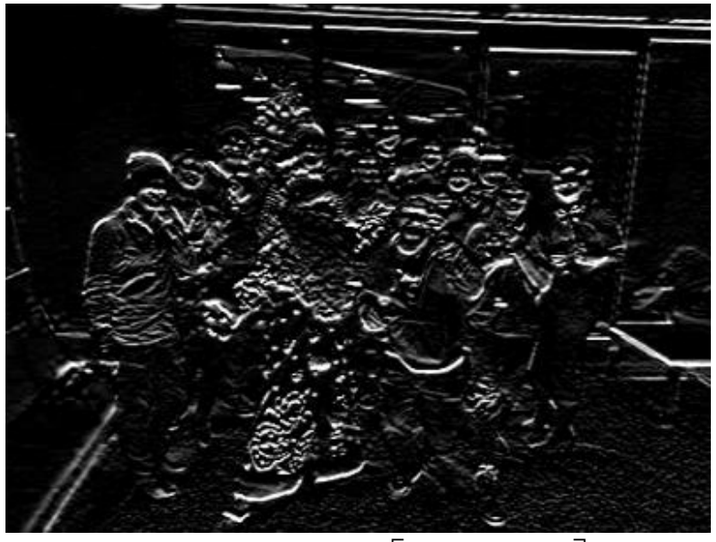
оригинал

blur

$$\begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix}$$

sharpen

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



top sobel $\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

left sobel $\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$

outline $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

bottom sobel $\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$

right sobel $\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$

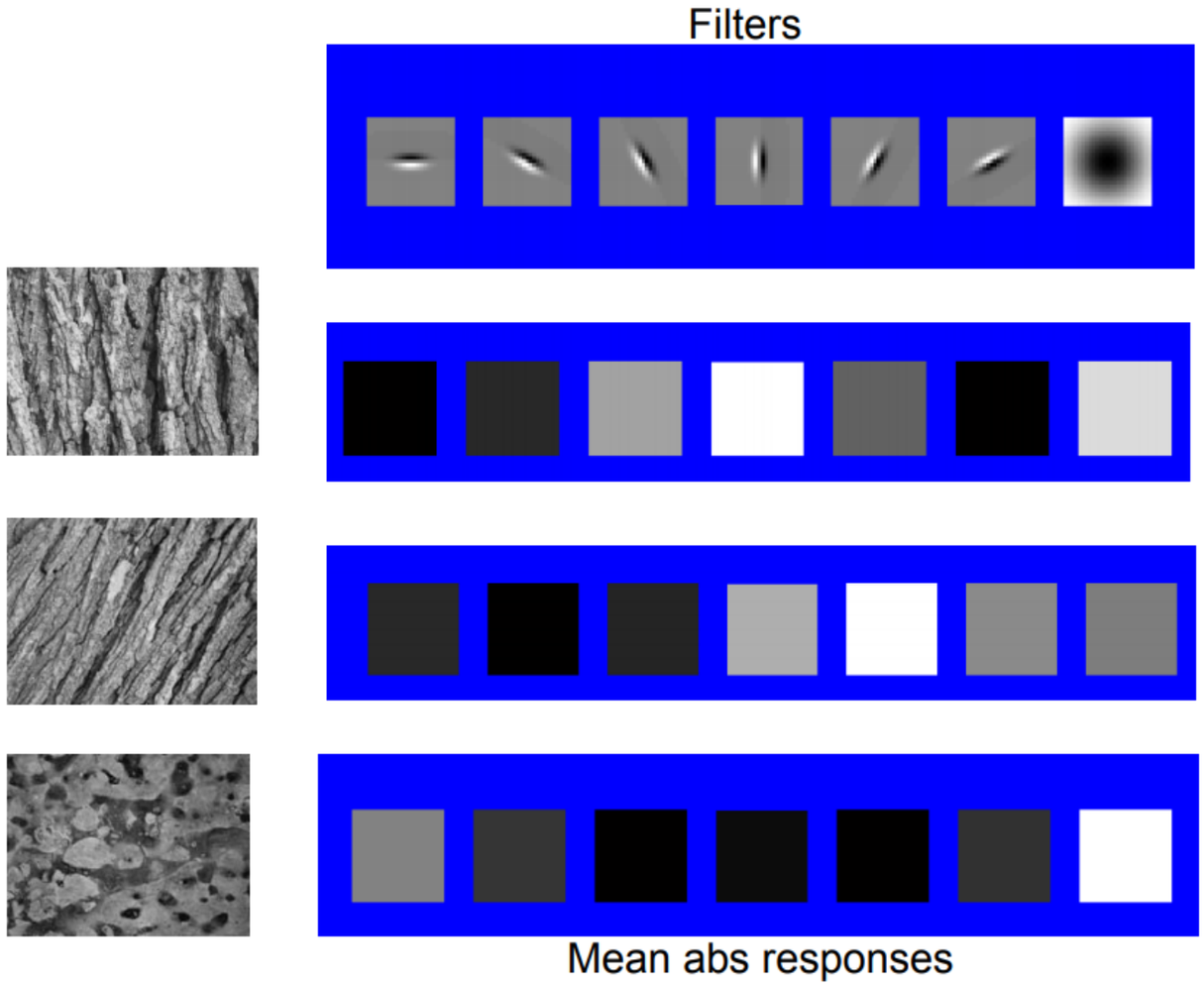
custom $\begin{bmatrix} +1 & -1 & +1 \\ -1 & 0 & -1 \\ +1 & -1 & +1 \end{bmatrix}$

Текстура в природе




https://people.cs.pitt.edu/~kovashka/cs2770_sp19/

Фильтры для текстур



Можно поиграться здесь: https://setosa.io/ev/image-kernels/

Свёртка (Convolution): мотивация

Раньше: обработка изображений – специально построенные свёртки

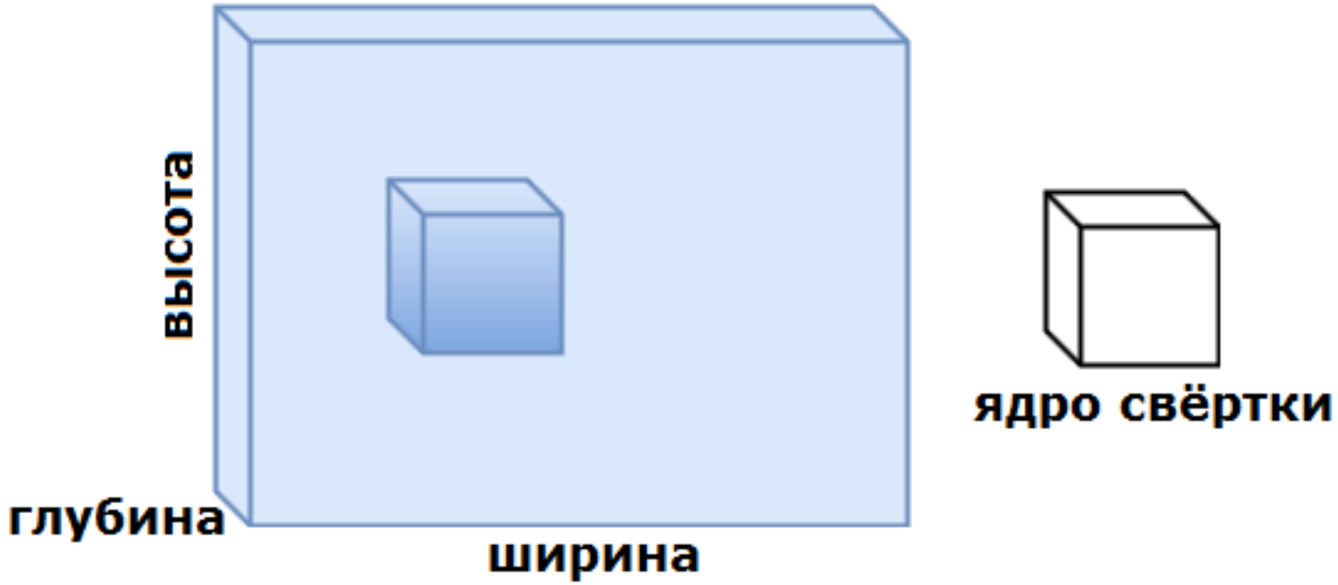
edges, corners, colors, shapes…

Сейчас: не будем специально конструировать свёртки – их параметры настроятся сами!

Важно: свёртку можно применять к изображениям любых размеров!

Нет ограничений на размеры входа…

Свёртка (Convolution)



Глубина (depth) / число каналов

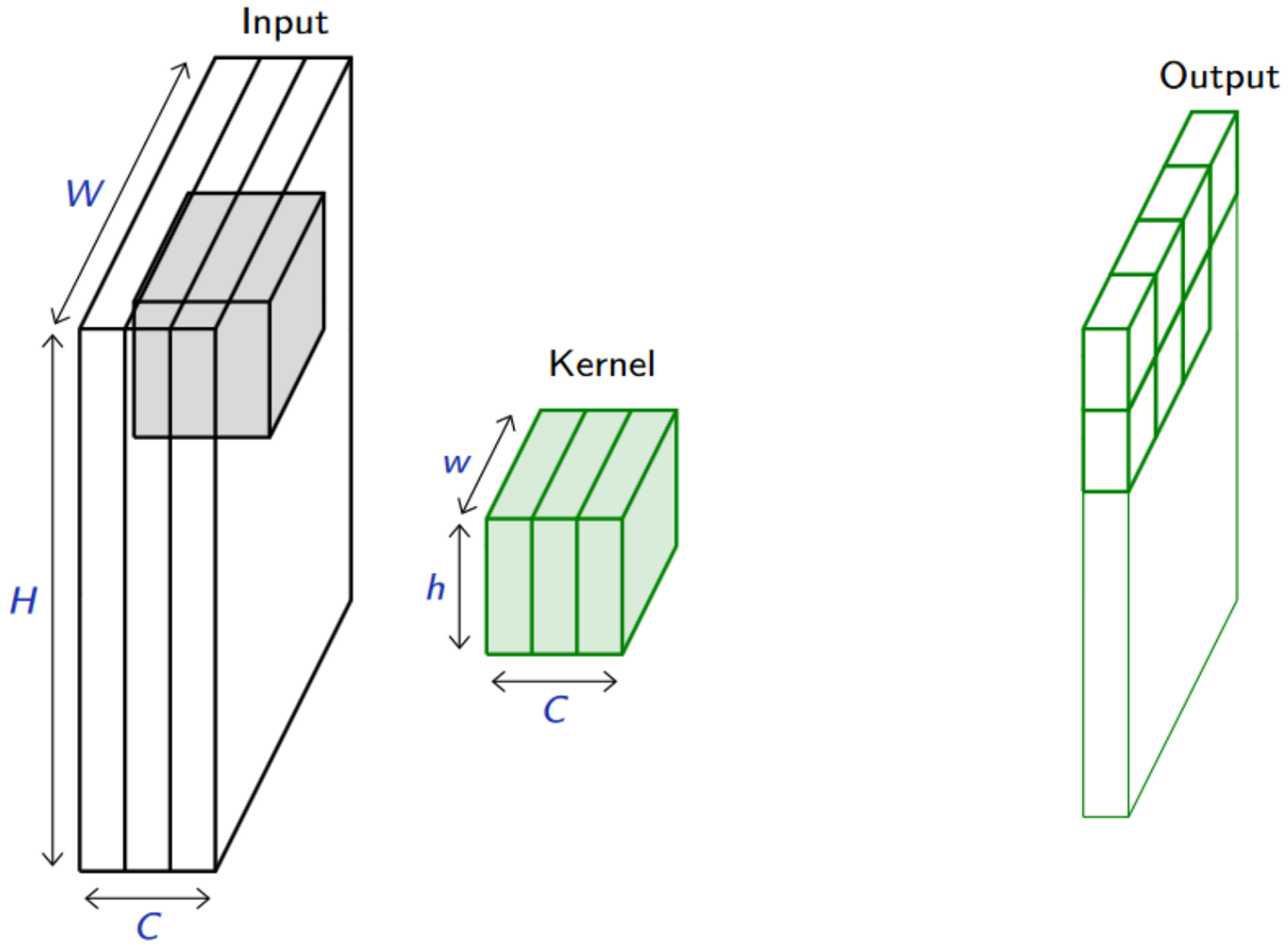
Высота (height) и ширина (width) тензора (изображения) / ядра

Шаг (stride) — на сколько смещается ядро при вычислении свёрток (чем больше, тем меньше размер итогового изображения)

Отступ (padding) – для дополнения изображения нулями по краям

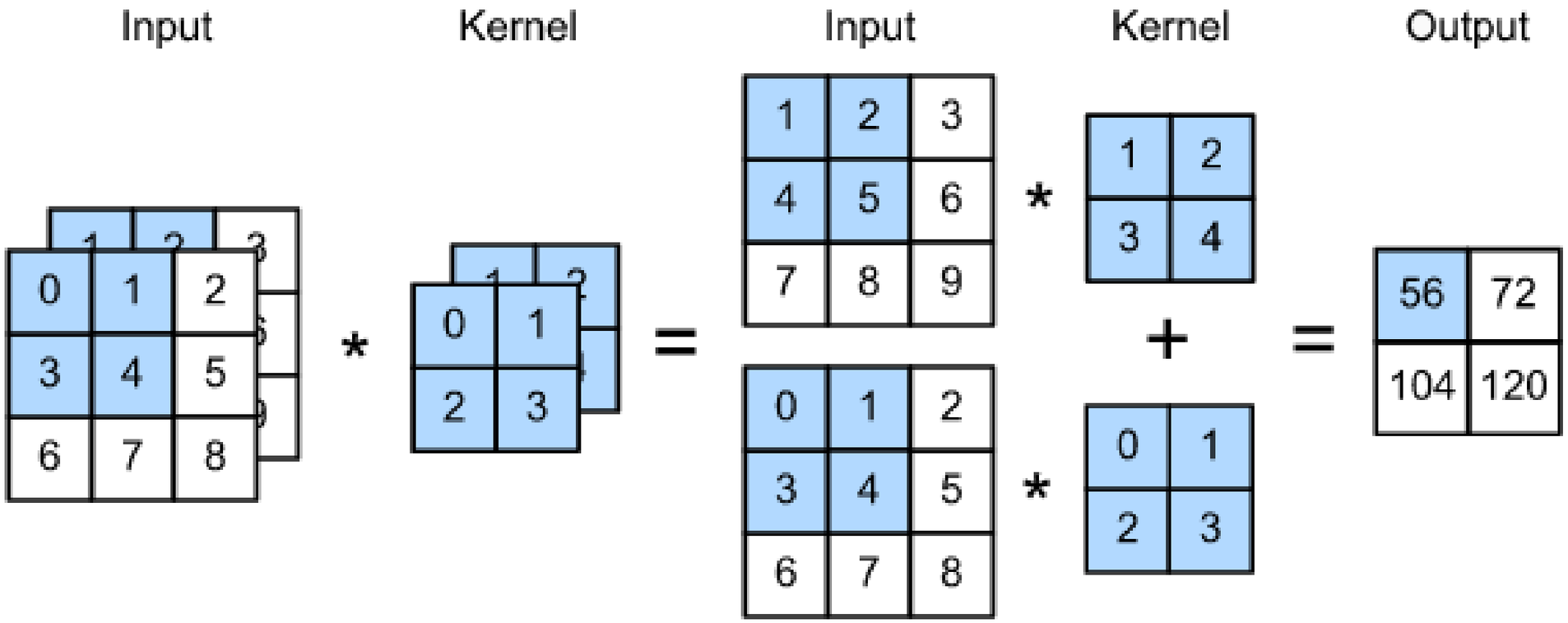
Ядро (kernel) или фильтр (filter) – размерность как у предыдущего тензора; в 3D длина и ширина меньше (глубина совпадает)

Свёртка (Convolution): глубина



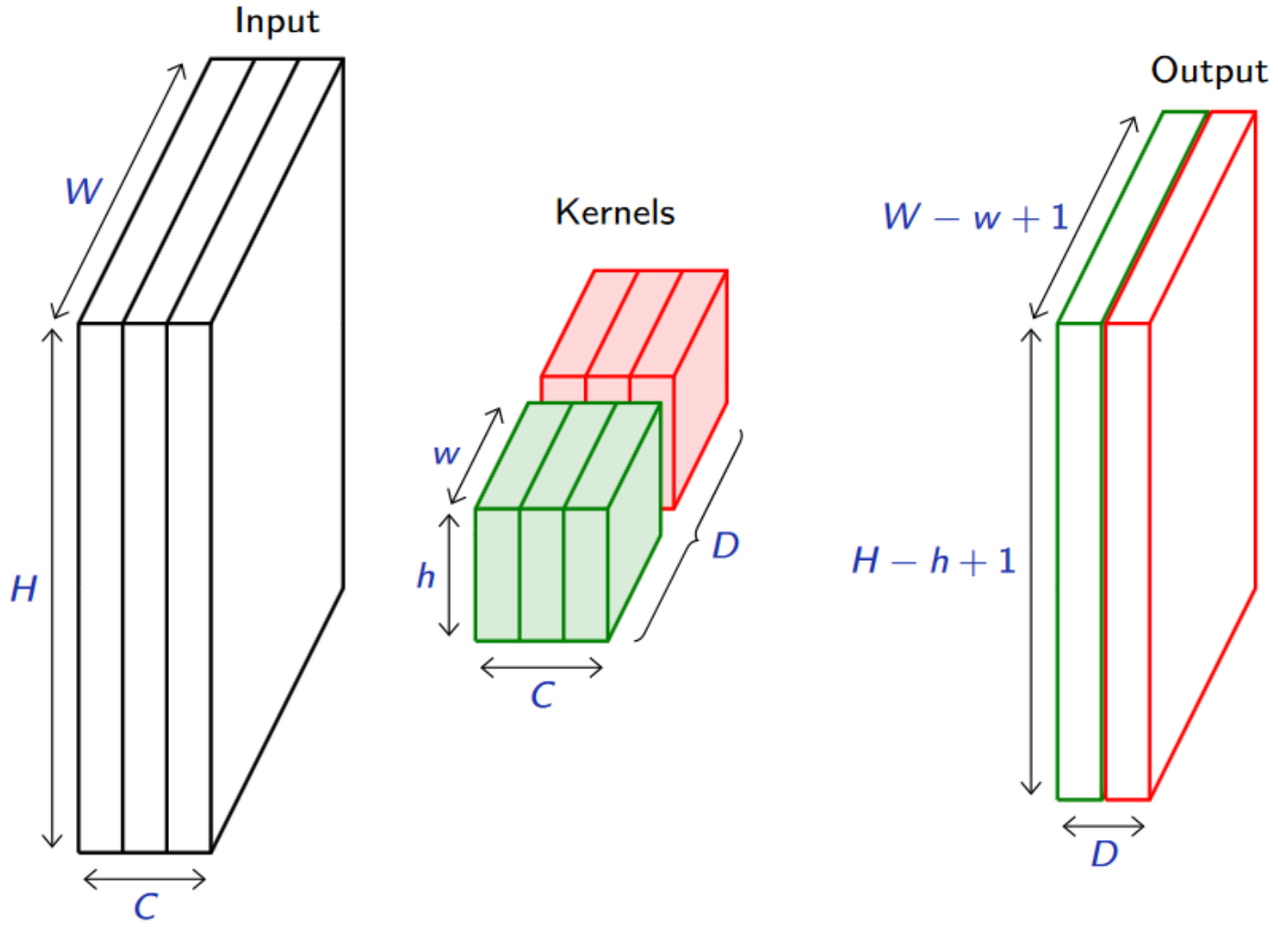
глубина тензора (число каналов) = глубина свёртки

Свёртка (Convolution): глубина



https://d2l.ai/chapter_convolutional-neural-networks/channels.html

Свёртка (Convolution): применение нескольких свёрток



каждая свёртка – 1 «лист» на выходе, k свёрток – k-канальный выход

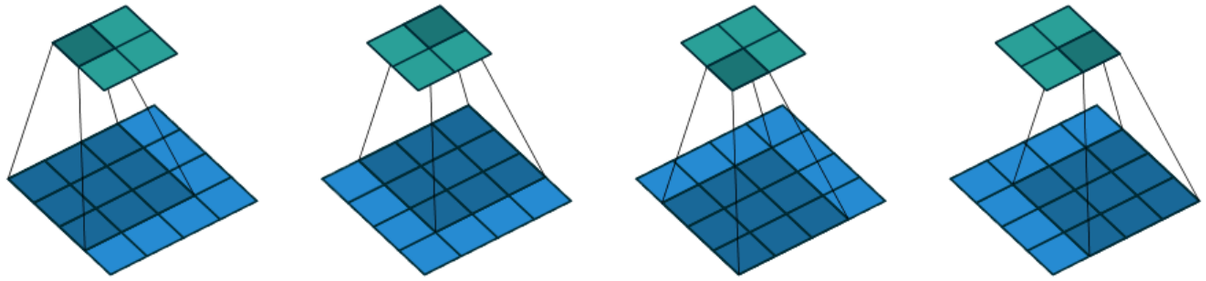
получаем на выходе тензор, глубина = число применяемых свёрток

свёрточный слой (для картинок) – 4D-массив $C_{out} \times C_{in} \times H \times W$

Свёртка (Convolution): отступы (padding) – чтобы сохранялись размеры изображения

без отступов

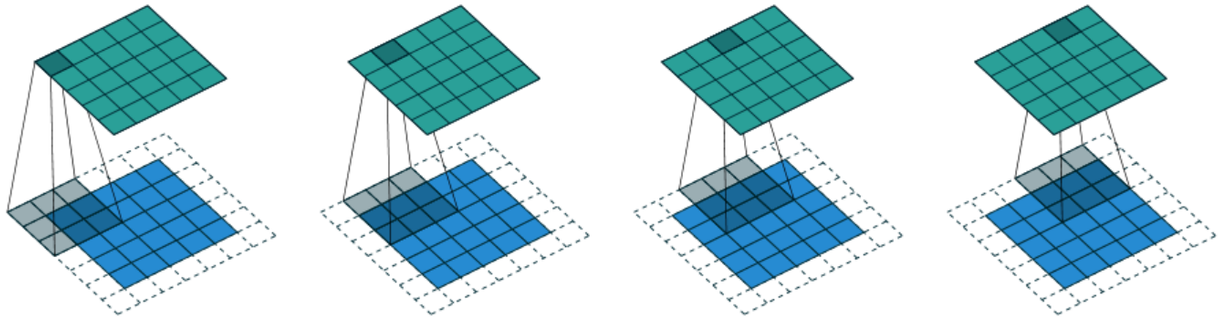
$$\begin{bmatrix}1 & 2 & 3\\ 3 & 4 & 5\\ 1 & 0 & 2\end{bmatrix} * \begin{bmatrix}0 & 1\\ 1 & -1\end{bmatrix} = \begin{bmatrix}1 & 2\\ 5 & 3\end{bmatrix}$$



с отступами

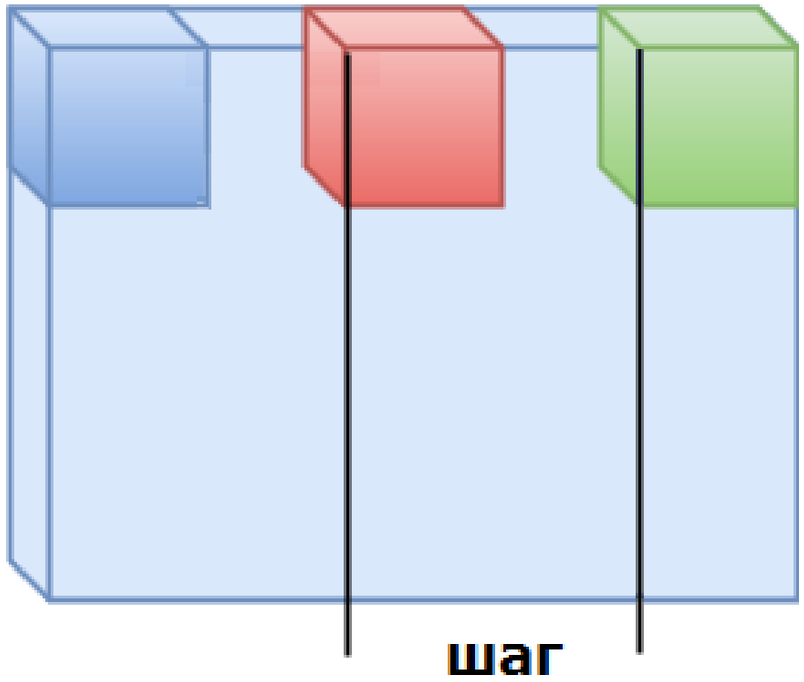
$$\begin{bmatrix}1 & 2 & 3\\ 3 & 4 & 5\\ 1 & 0 & 2\end{bmatrix} * \begin{bmatrix}0 & 1\\ 1 & -1\end{bmatrix} \equiv$$

$$\equiv \begin{bmatrix}1 & 2 & 3 & 0\\ 3 & 4 & 5 & 0\\ 1 & 0 & 2 & 0\\ 0 & 0 & 0 & 0\end{bmatrix} * \begin{bmatrix}0 & 1\\ 1 & -1\end{bmatrix} = \begin{bmatrix}1 & 2 & 5\\ 5 & 3 & 4\\ 0 & 2 & 0\end{bmatrix}$$



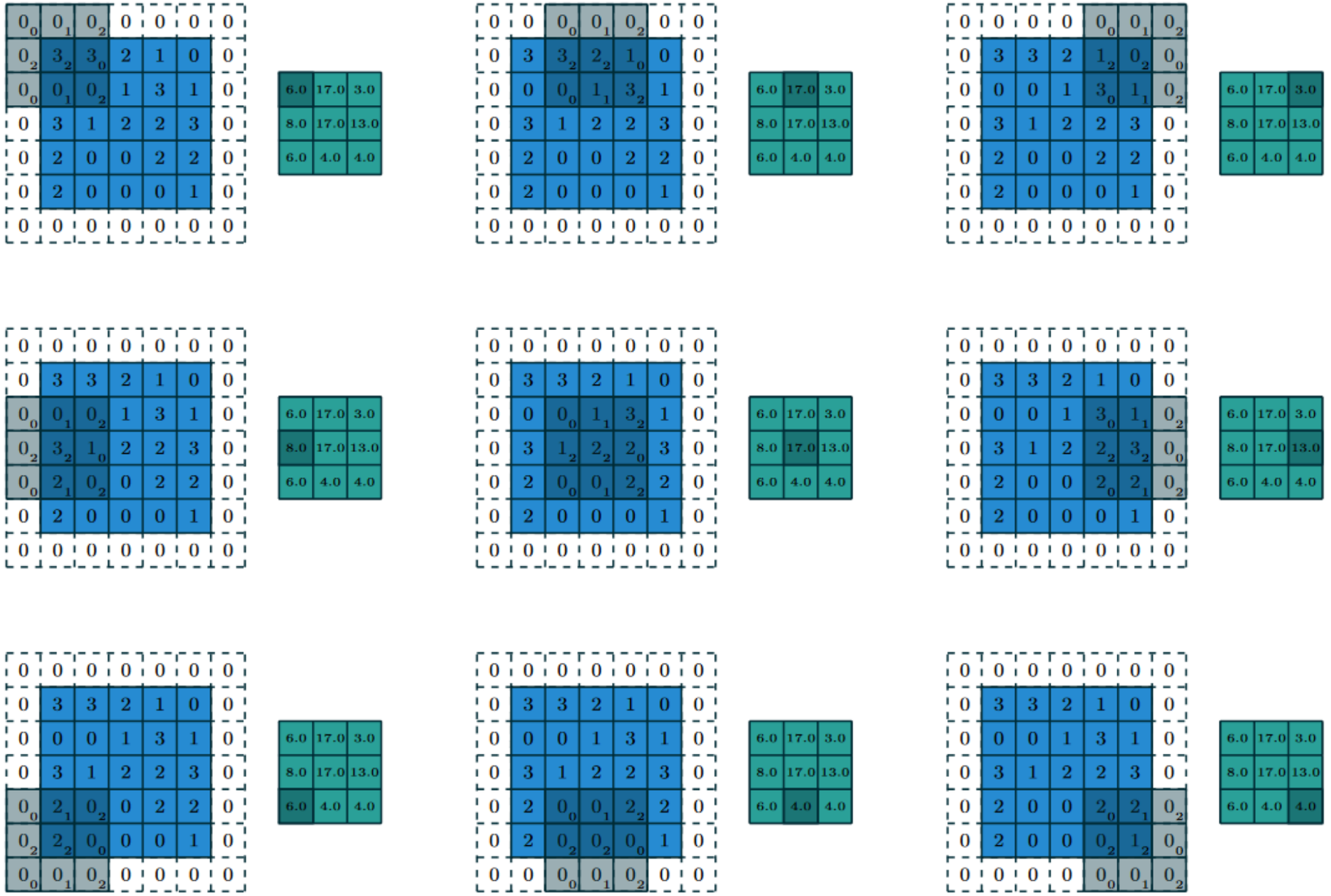
Свёртка (Convolution): шаг (stride)

смещаемся при вычислении свёртки (можно в каждой её размерности)



с шагом 2

$$\begin{bmatrix} 1 & 2 & 3 & 1 & 0 \\ 3 & 4 & 5 & 2 & 1 \\ 1 & 0 & 2 & 3 & 4 \\ 0 & 1 & 3 & 1 & 2 \\ 1 & 2 & 4 & 2 & 3 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ -5 & 2 \end{bmatrix}$$


0 0 0 0 0 0 0
0 3 3 2 1 0 0
0 0 0 1 3 1 0
0 3 1 2 2 3 0
0 2 0 0 2 2 0
0 2 0 0 0 1 0
0 0 0 0 0 0 0
6.0 17.0 3.0
8.0 17.0 13.0
6.0 4.0 4.0

Свёртка (Convolution): минутка кода

```
torch.nn.Conv2d(in_channels: int,
                out_channels: int,
                kernel_size: Union[T, Tuple[T, T]],
                stride: Union[T, Tuple[T, T]] = 1,
                padding: Union[T, Tuple[T, T]] = 0,
                dilation: Union[T, Tuple[T, T]] = 1,
                groups: int = 1,
                bias: bool = True,
                padding_mode: str = 'zeros') # 'reflect','replicate','circular'
input = torch.randn(20, 16, 50, 100)
m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2),
              dilation=(3, 1))
output = m(input)
```

in_channels, out_channels – количество каналов на входе и выходе – должны делиться на groups!

kernel_size – размеры ядра

stride – смещение (можно понижать разрешение)

padding – отступы

dilation – расстояние между точками ядра (увеличивает область зависимости)

groups – опр. связи между входом и выходом

Минутка кода: свёртка (Convolution)

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

размеры выхода (в разных реализациях – по-разному)

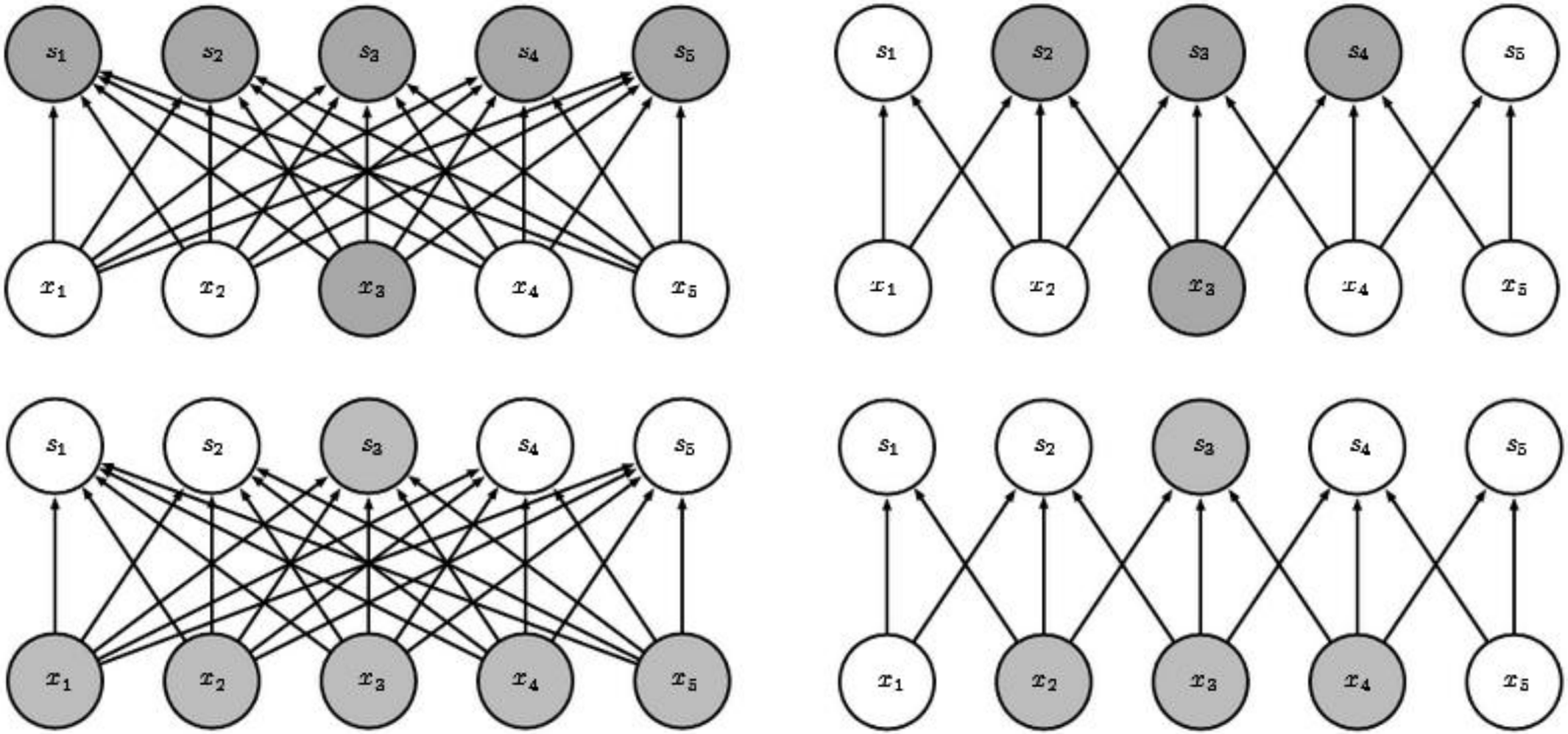
Реализация: свёртка – это линейная операция (нужно быстро умножать матрицы)

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix} * \begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix} = \begin{pmatrix} k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 & 0 \\ 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 \\ 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 \\ 0 & 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} \end{pmatrix} \cdot \begin{pmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{31} \\ x_{32} \\ x_{33} \end{pmatrix} =$$

$$= \begin{pmatrix} k_{11}x_{11} + k_{12}x_{12} + k_{21}x_{21} + k_{22}x_{22} \\ k_{11}x_{12} + k_{12}x_{13} + k_{21}x_{22} + k_{22}x_{23} \\ k_{11}x_{21} + k_{12}x_{21} + k_{21}x_{31} + k_{22}x_{32} \\ k_{11}x_{22} + k_{12}x_{23} + k_{21}x_{32} + k_{22}x_{33} \end{pmatrix}$$

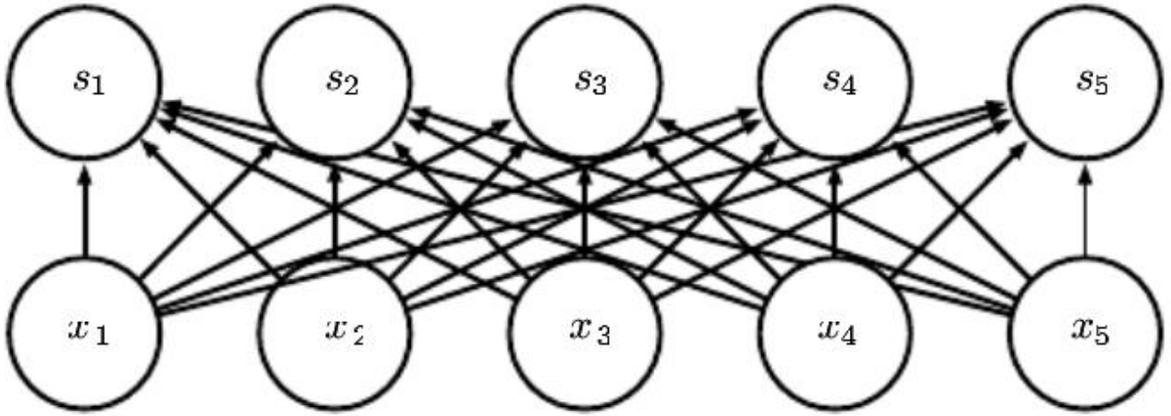
$$\sim \begin{bmatrix} k_{11}x_{11} + k_{12}x_{12} + k_{21}x_{21} + k_{22}x_{22} & k_{11}x_{12} + k_{12}x_{13} + k_{21}x_{22} + k_{22}x_{23} \\ k_{11}x_{21} + k_{12}x_{22} + k_{21}x_{31} + k_{22}x_{32} & k_{11}x_{22} + k_{12}x_{23} + k_{21}x_{32} + k_{22}x_{33} \end{bmatrix}$$

Разреженные взаимодействия (sparse interactions)



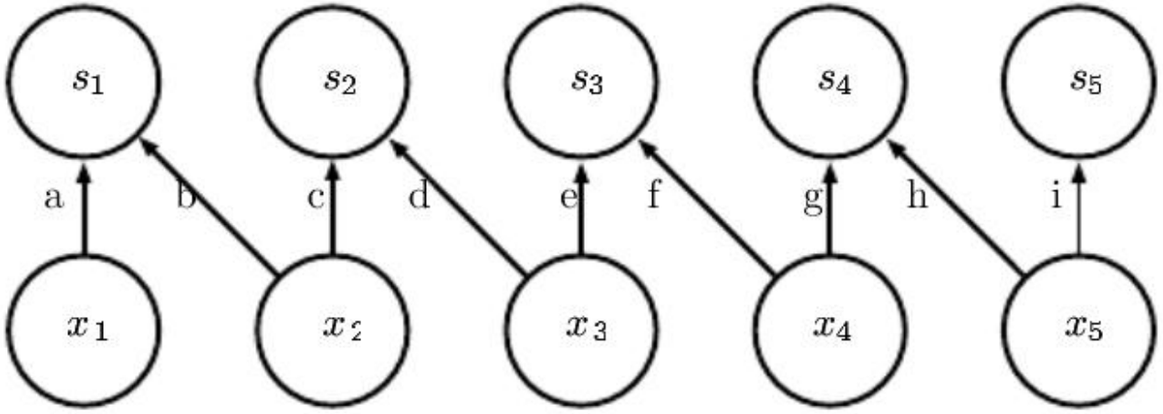
http://www.deeplearningbook.org/contents/convnets.html

Полная связность (full connections)



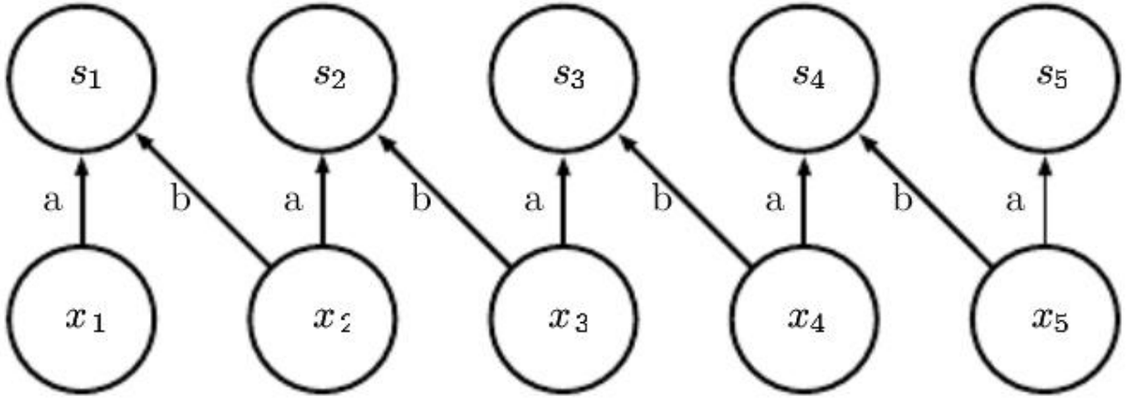
Локальная связность (local connections)

неразделяемая свёртка (unshared convolution)

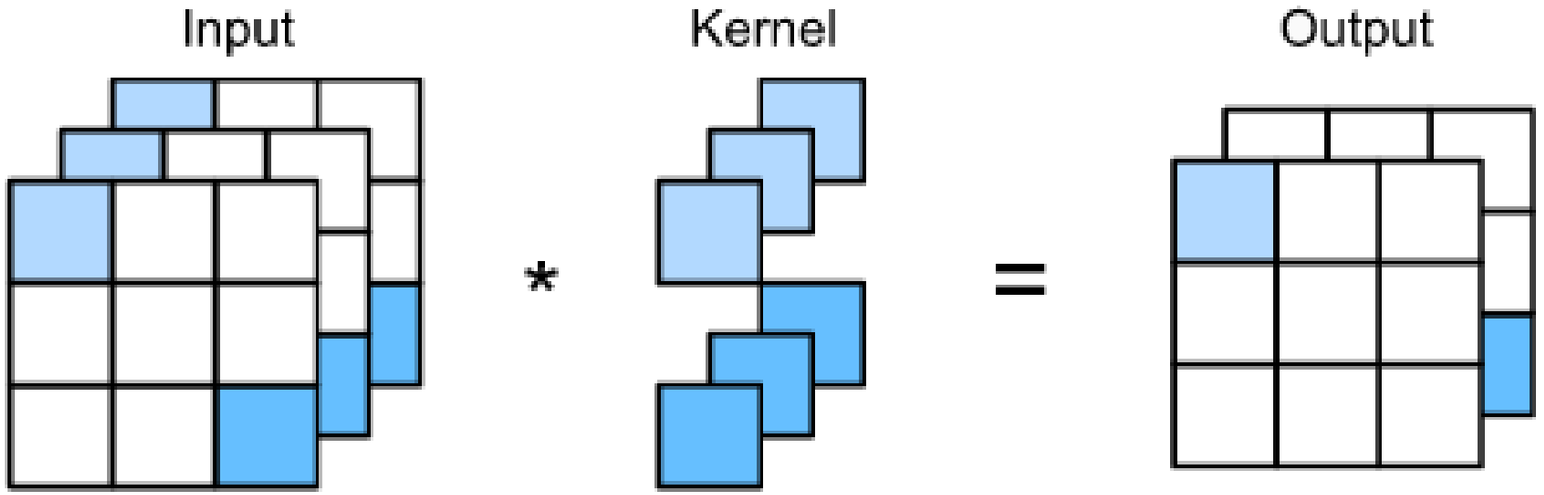


Свёртка (convolution)

разделяемая свёртка



Смысл свёрток 1×1 (Pointwise Convolution)



линейная комбинация
одинаковая для каждого пикселя
«вдоль каналов» (признаков)

потом может (должна) идти нелинейность
⇒ это маленькая НС

https://d2l.ai/chapter_convolutional-neural-networks/channels.html

Смысл свёрток 1×1 (Pointwise Convolution)

применение 32 свёрток 64×1×1:

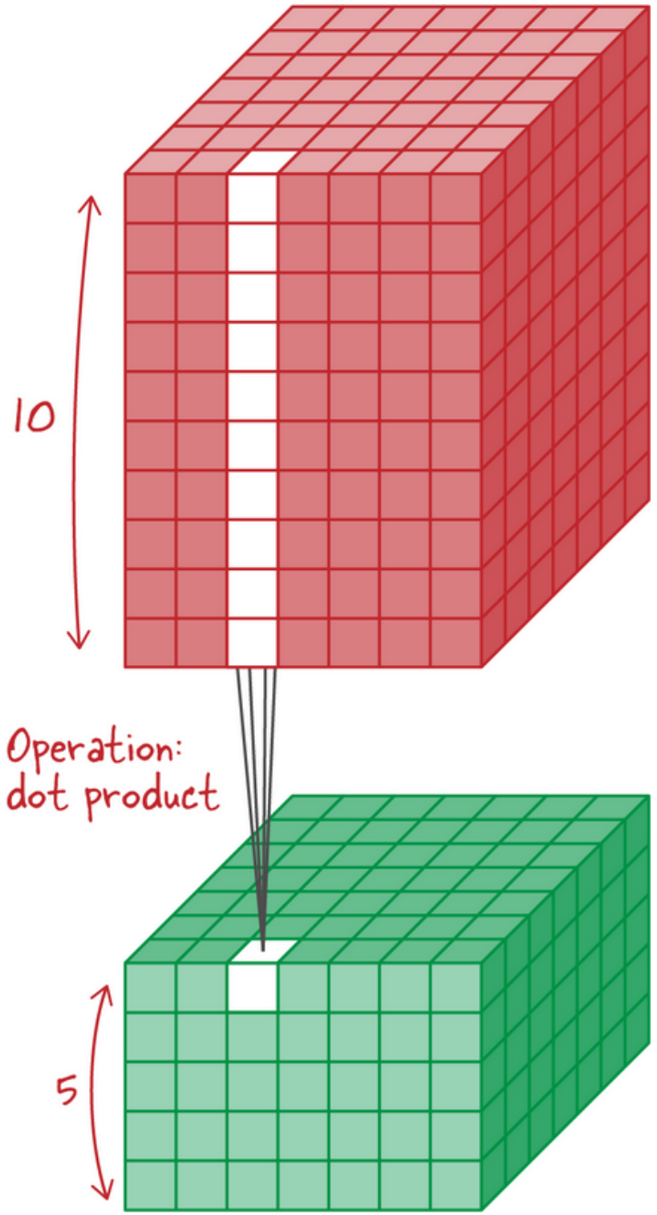
Преобразование признаков!

будет часто использоваться – это своеобразная мини-нейронка

+ изменение числа каналов

+ «узкое горло» в НС

Смысл свёрток 1×1 (Pointwise Convolution)



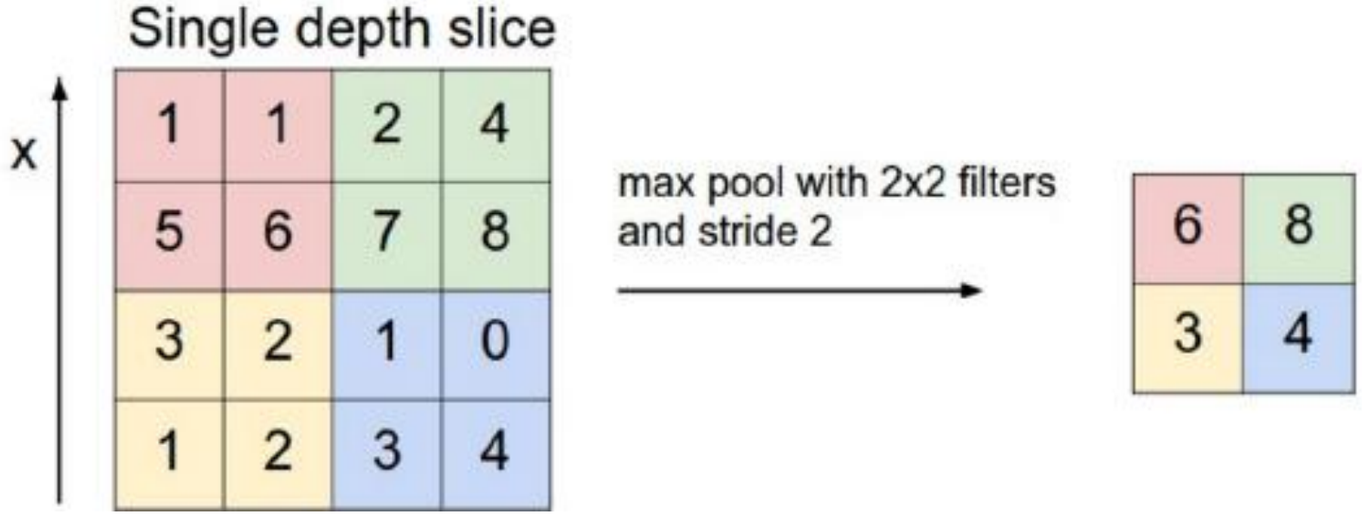
Нейрон, действующий по пикселям «вдоль каналов»

```
in_channels = 10
out_channels = 5
kernel_size = 1×1
```

[Practical Machine Learning for Computer Vision]

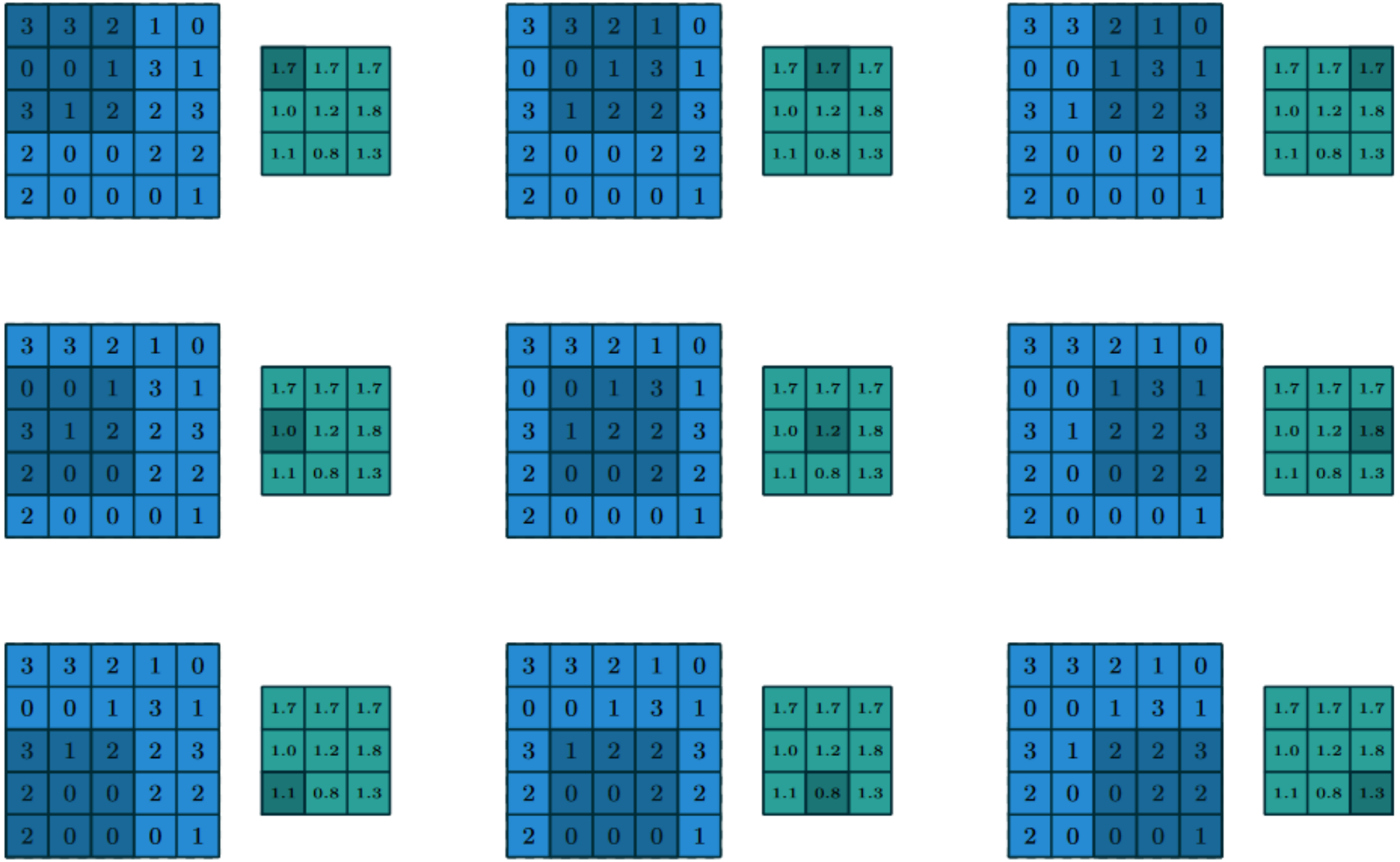
Pooling (агрегация, субдискретизация / subsampling)

для каждого признака канала надо определить,
нашли ли паттерн
используем функцию агрегации (mean, max, …)



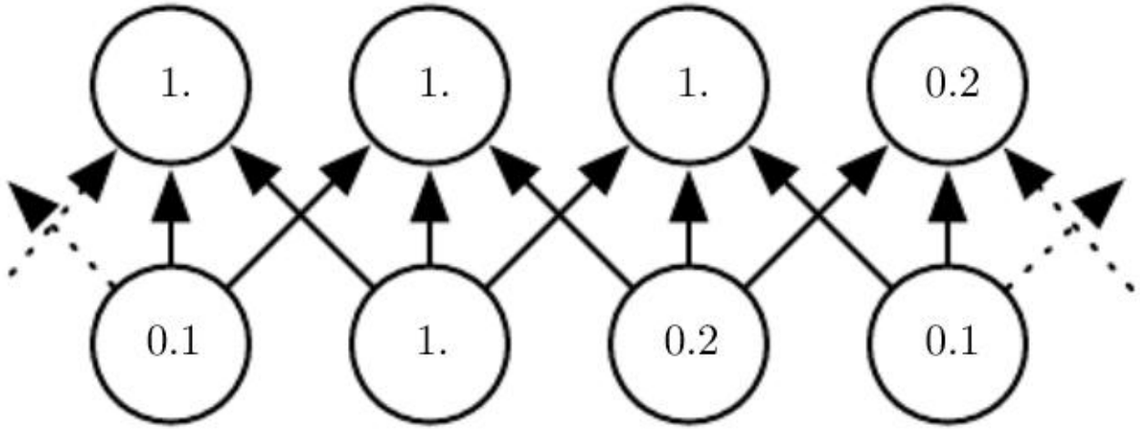
делается независимо по каналам ⇒ сохраняет число каналов (глубину тензора)

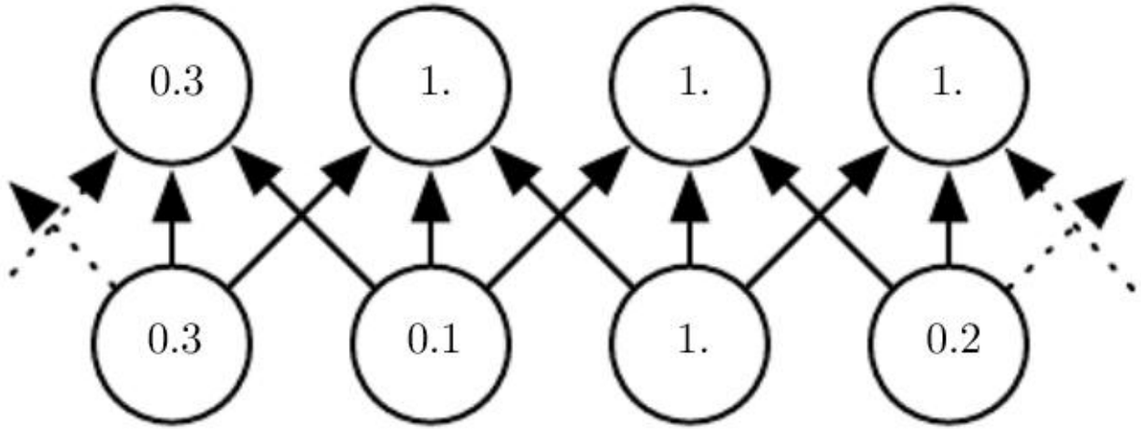
Агрегация (Pooling) усреднением



Агрегация (Pooling)

Max-pooling – инвариантность к небольшим сдвигам
«equivariant representation»





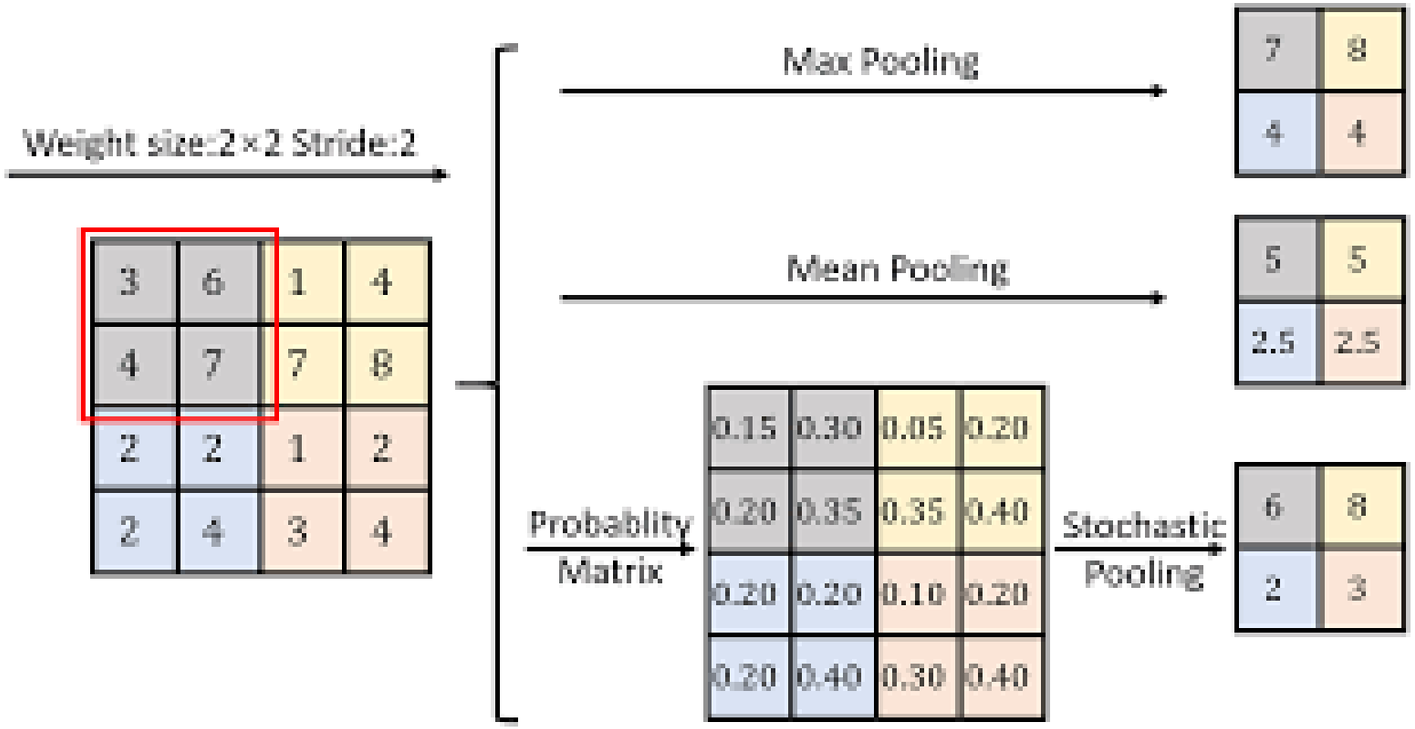
Аналог голосования…

Если надо найти кошку, то в определённой окрестности
⇒ опросить соседей, есть ли кошка

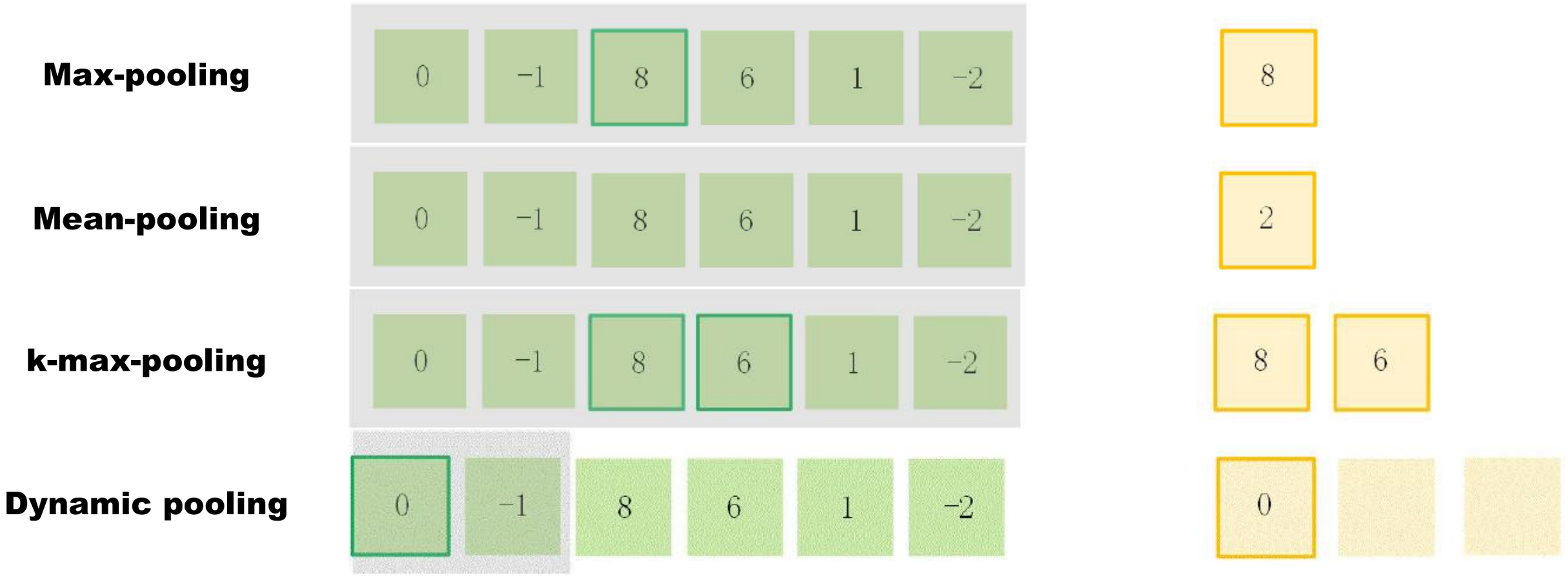
Агрегация (Pooling): виды пулинга

- усреднение
- усреднение с весами
- L2-норма
- Stochastic Pooling

выдаём значение с вероятностью ~ значение



Агрегация (Pooling): виды пулинга

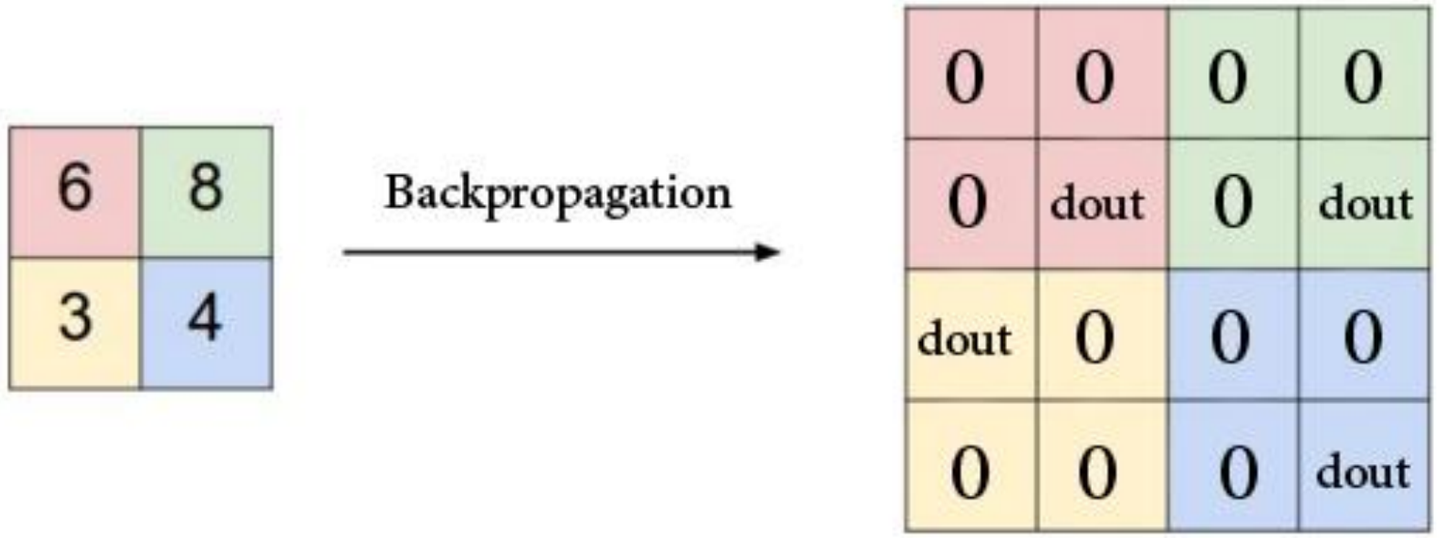


Динамический – длина может от чего-то зависеть

http://www.phontron.com/class/nn4nlp2020/schedule.html

Агрегация (Pooling): дифференцирование

При дифференцировании возвращают градиент в позициях максимумов

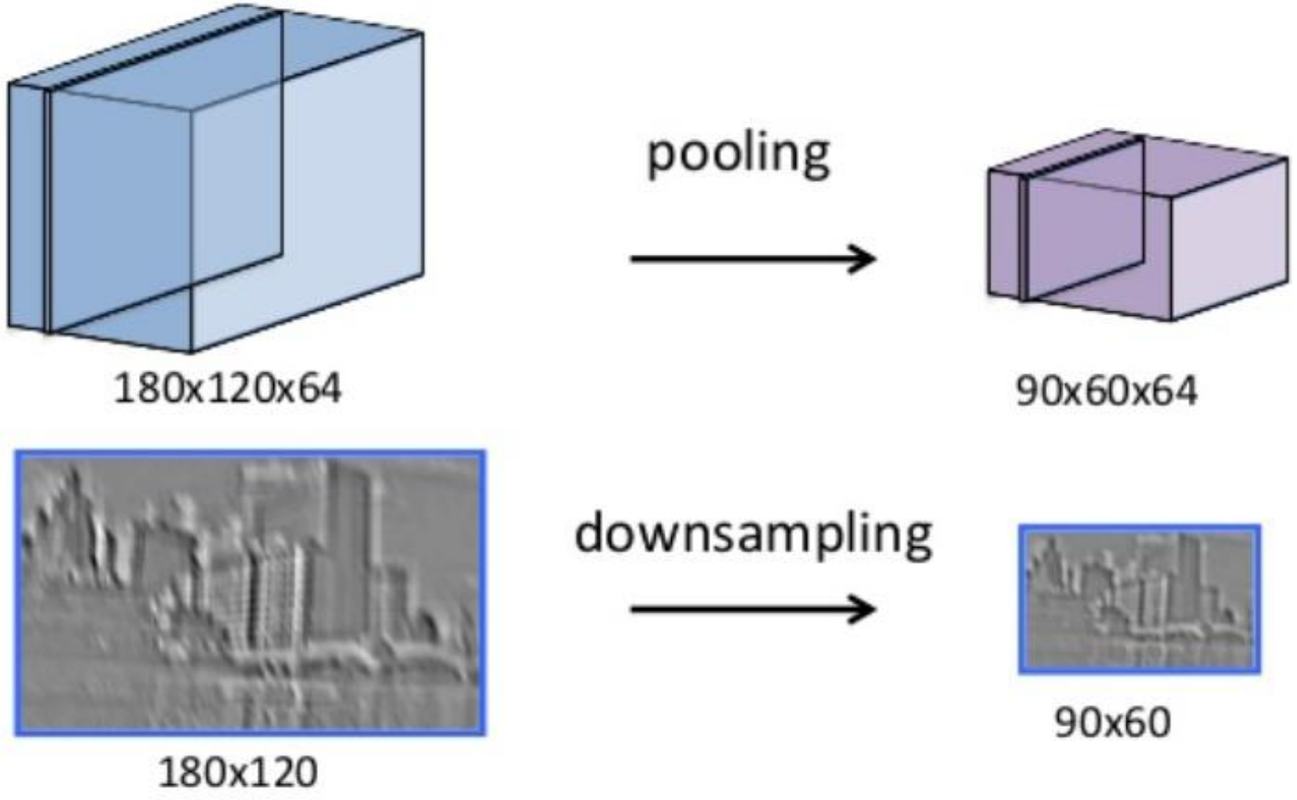


https://leonardoaraujosantos.gitbook.io/artificial-inteligence/machine_learning/deep_learning/pooling_layer

Pooling layer = downsampling layer

С помощью пулинга можно приводить изображение к нужному размеру!

(его можно/нужно делать с шагом)



Почему не уменьшают свёртками с шагом?

1. Shift invariance

2. Non-linearity

в дополнении к ReLu

3. Speed

пулинг быстрее свёртки

пулинг в каждой локальной области независим от значений в других областях!

https://www.quora.com/Why-would-we-do-max-pooling-when-we-can-downsample-by-Strided-convolution

Минутка кода: агрегация (Pooling)

```
torch.nn.MaxPool2d(kernel_size: Union[T, Tuple[T, ...]],
                   stride: Optional[Union[T, Tuple[T, ...]]] = None,
                   padding: Union[T, Tuple[T, ...]] = 0,
                   dilation: Union[T, Tuple[T, ...]] = 1,
                   return_indices: bool = False,
                   ceil_mode: bool = False) # как вычислять размеры результата

input = torch.randn(20, 16, 50, 32)
m = nn.MaxPool2d((3, 2), stride=(2, 1))
output = m(input)
```

Shape:

- Input: (N, C, H_{in}, W_{in})
- Output: (N, C, H_{out}, W_{out}), where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 * \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 * \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Устройство слоя свёрточной НС:

свёрточная часть: [свёртка $\rightarrow$ нелинейность $\rightarrow$ пулинг] $\times$ k

Мотивация:

- **разреженные взаимодействия (sparse interactions)**

нет связи нейронов «каждый с каждым»

У свёрточных НС мало весов!!!

- **разделение параметров (parameter sharing)**

одна свёртка используется «по всему изображению»

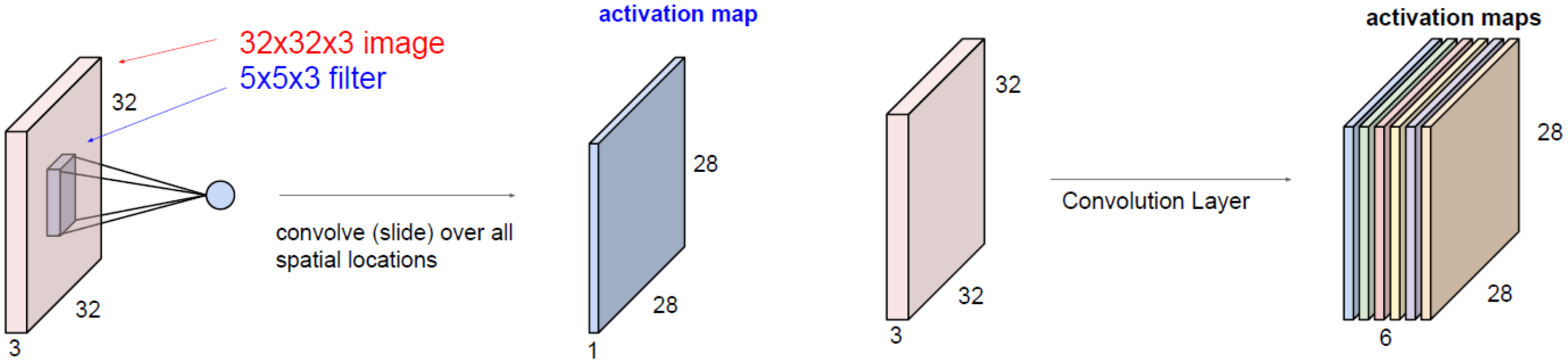
$\Rightarrow$ мало параметров

- **инвариантные преобразования (equivariant representations)**

инвариантность относительно сдвига

http://www.deeplearningbook.org/contents/convnets.html

Свёрточный слой: тензор → тензор

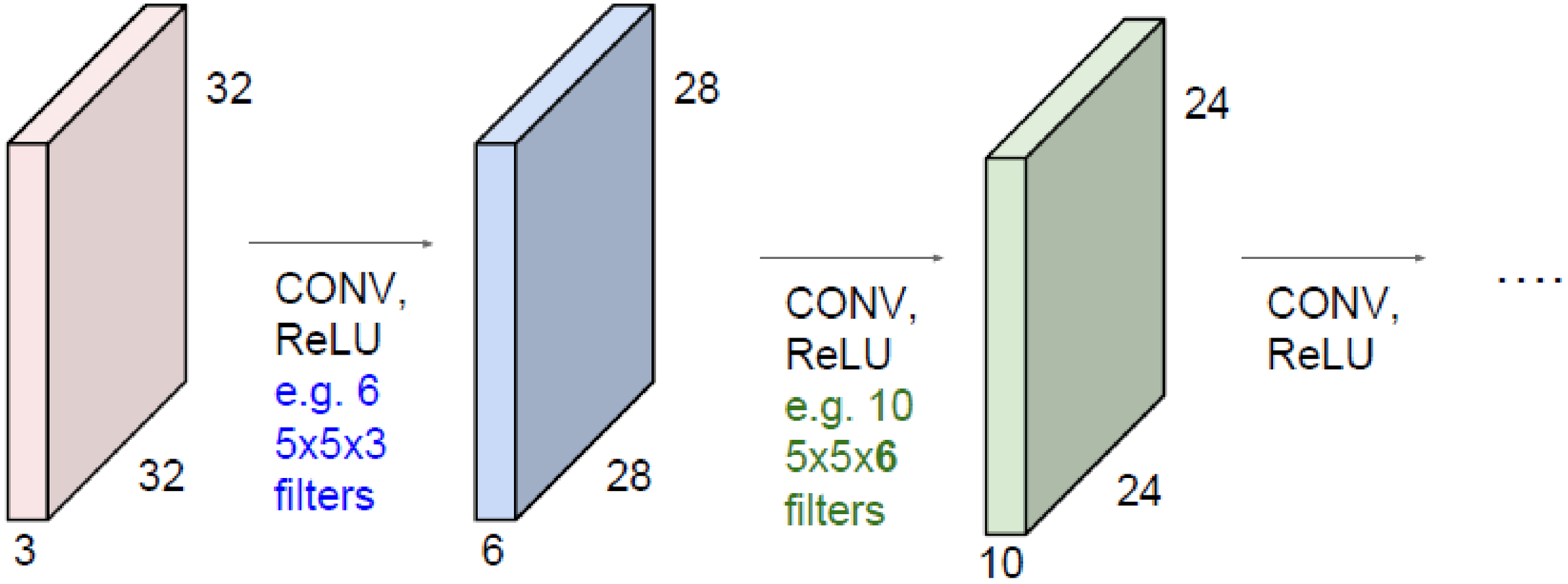


каждый слой (свёртка → нелинейность → пулинг) переводит тензор → тензор

важно, что всегда получаем тензор,
возможно, других размеров

32×32×3 → 28×28×6 (карта признаков)

Свёрточная НС: тензор → тензор



Каждый тензор:

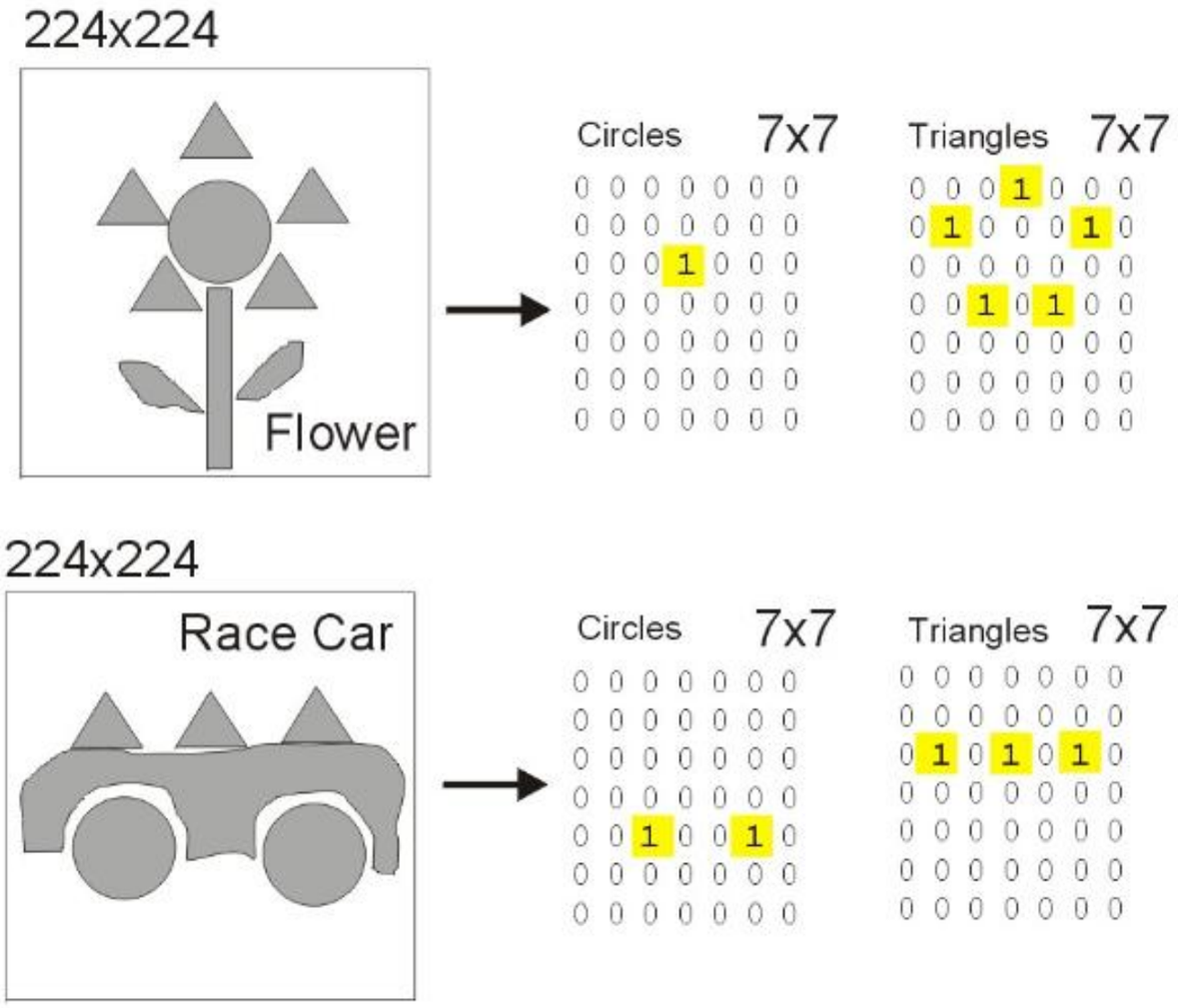
 # признаков / каналов (глубина) × ширина × высота

Свёрточная НС: тензор → тензор

```
f = nn.Conv2d(in_channels=1, out_channels=1, kernel_size=3)
x = torch.randn(1, 1, 28, 28)
print (x.shape)
x = f(x)
print (x.shape)
x = F.max_pool2d(x, kernel_size=2)
print (x.shape)
x = f(x)
print (x.shape)
x = F.max_pool2d(x, kernel_size=2)
print (x.shape)
```

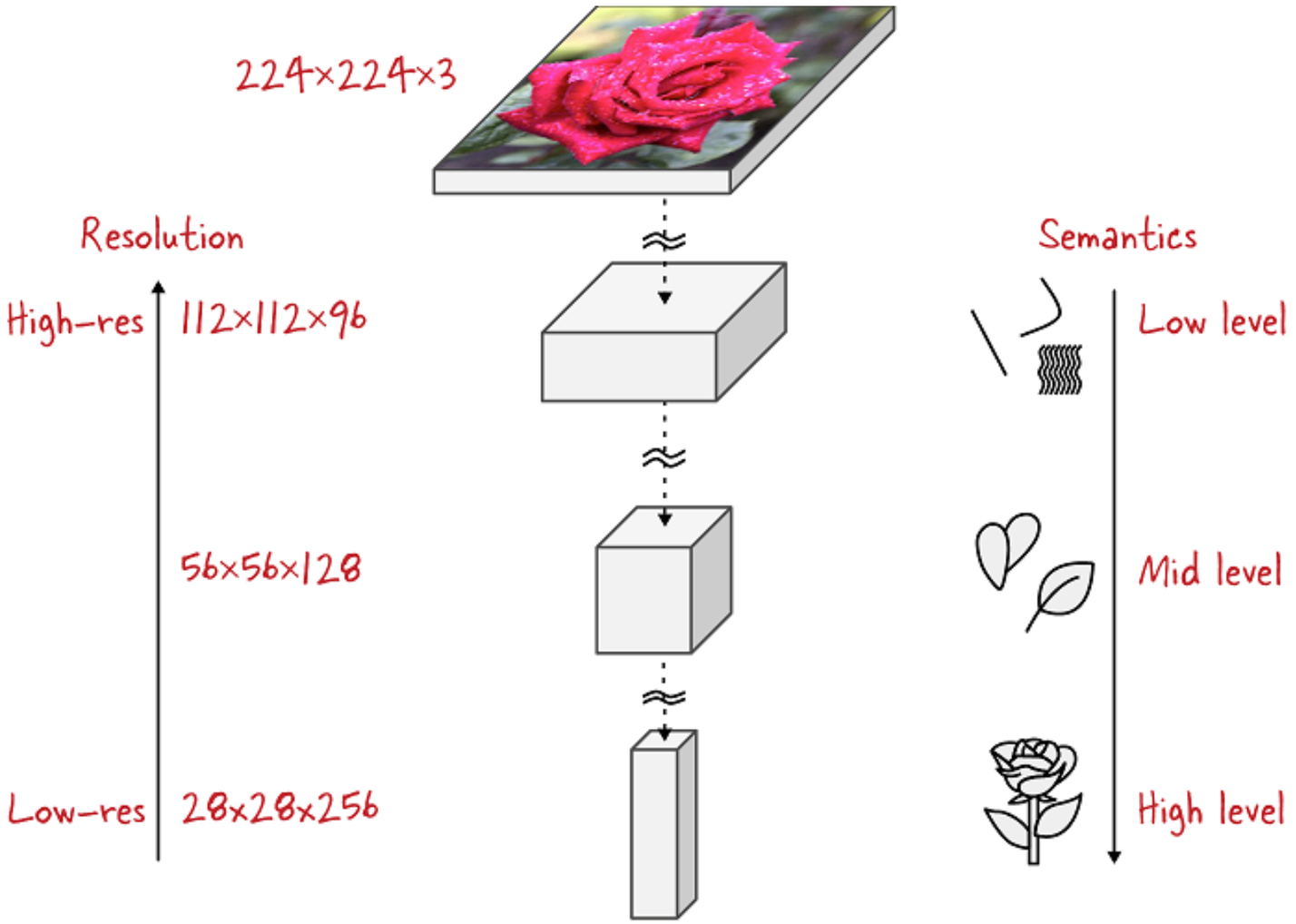
```
torch.Size([1, 1, 28, 28])
torch.Size([1, 1, 26, 26])
torch.Size([1, 1, 13, 13])
torch.Size([1, 1, 11, 11])
torch.Size([1, 1, 5, 5])
```

Визуализация признаков: за что могут отвечать свёртки последующих слоёв…



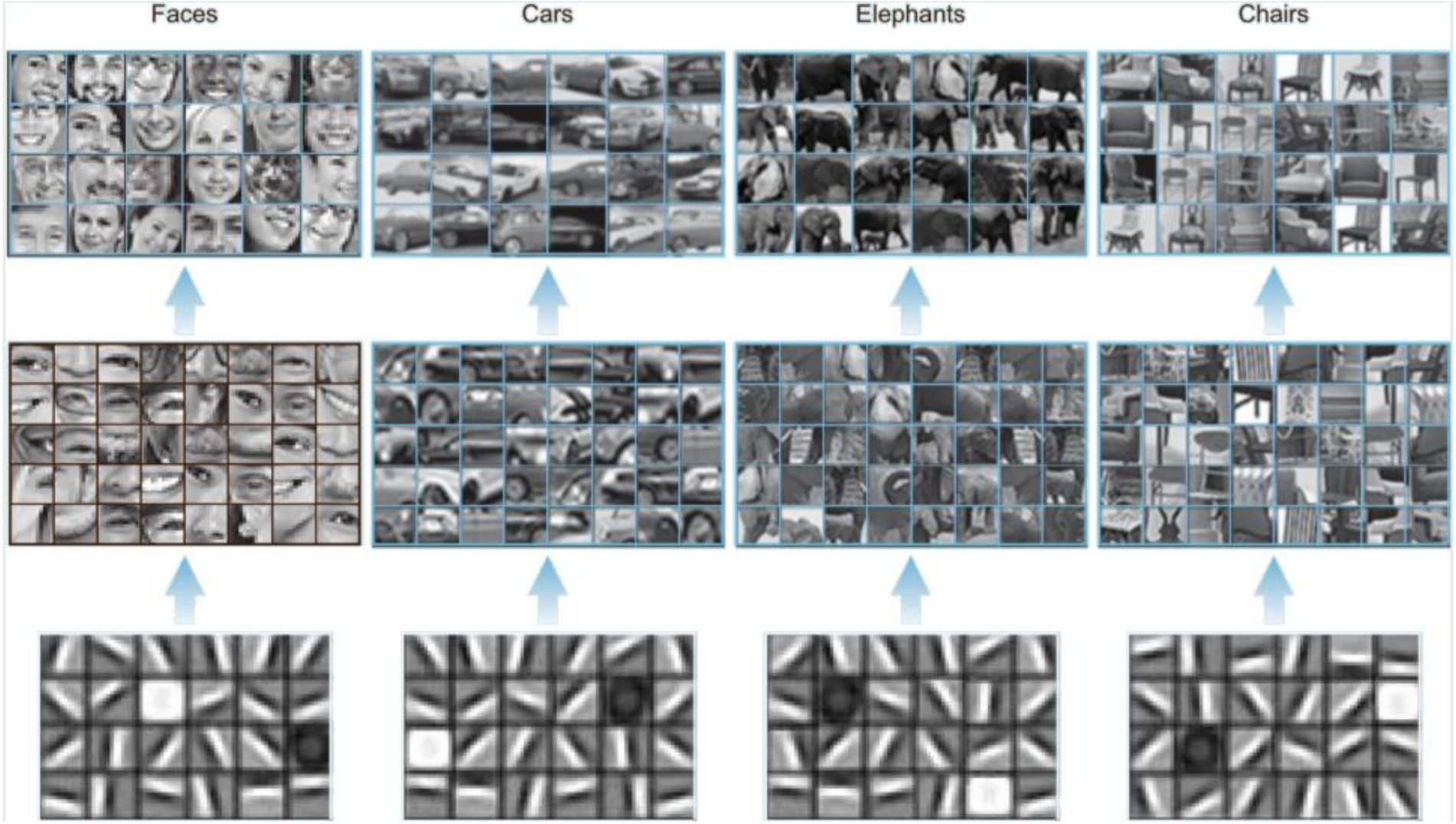
https://www.kaggle.com/c/siim-isic-melanoma-classification/discussion/160147

Визуализация признаков



[Practical Machine Learning for Computer Vision]

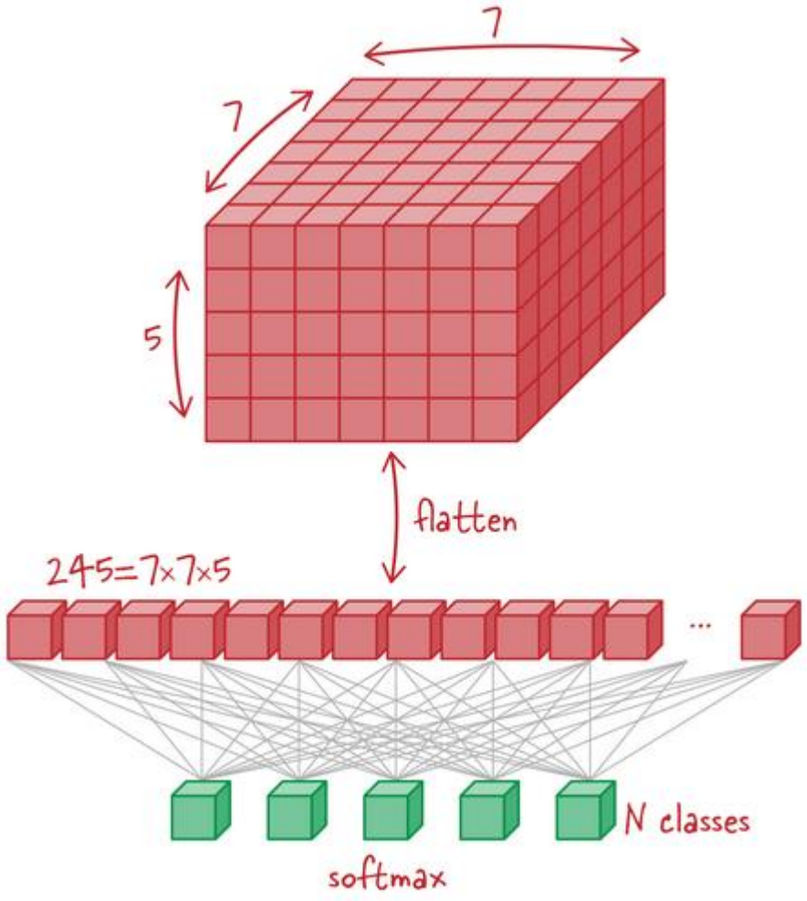
Визуализация признаков

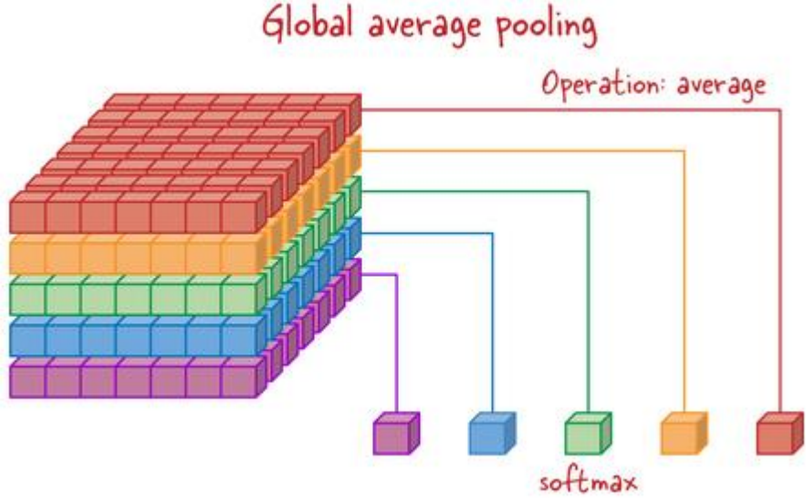


[Mohamed Elgendy]

Последние слои CNN – векторизация / глобальный пулинг

как перейти от H×W-пространства к пространству «однородных признаков»



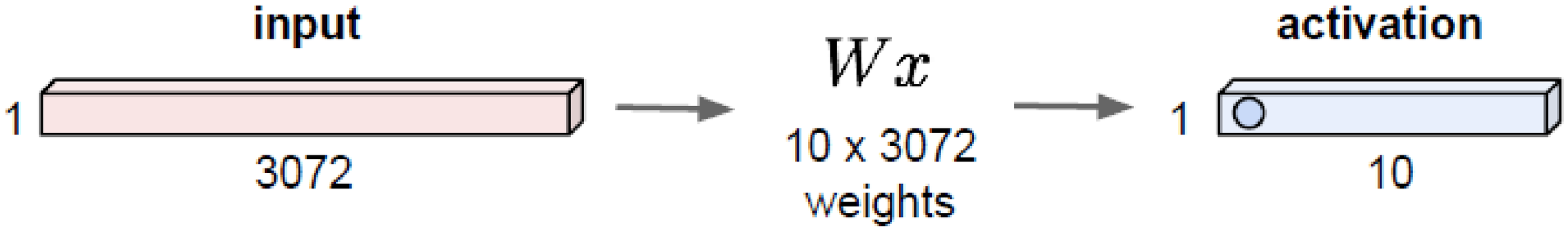


Совсем пропадает пространственная информация

[Practical Machine Learning for Computer Vision]

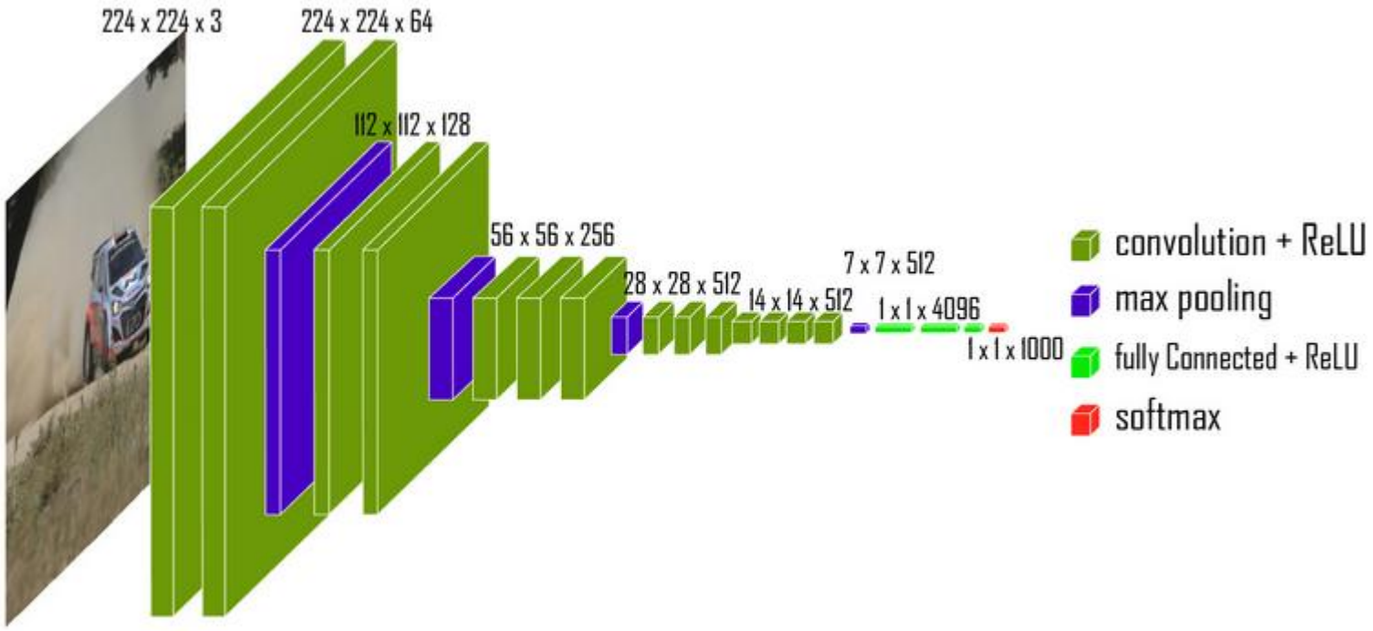
Последние слои CNN – «полносвзяная часть»

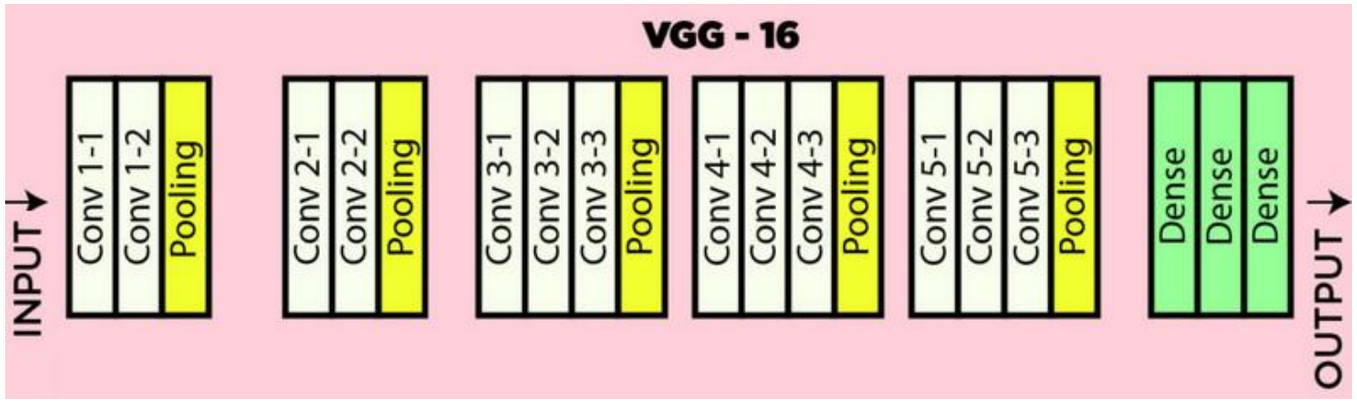
тензор 3×32×32 → 3072D-вектор → линейный слой → активация:



в конце свёрточной сети «обычные» полносвязные слои
для решения задачи классификации / регрессии
есть специальные сети без этих слоёв (дальше)

Архитектура CNN





https://www.geeksforgeeks.org/vgg-16-cnn-model/

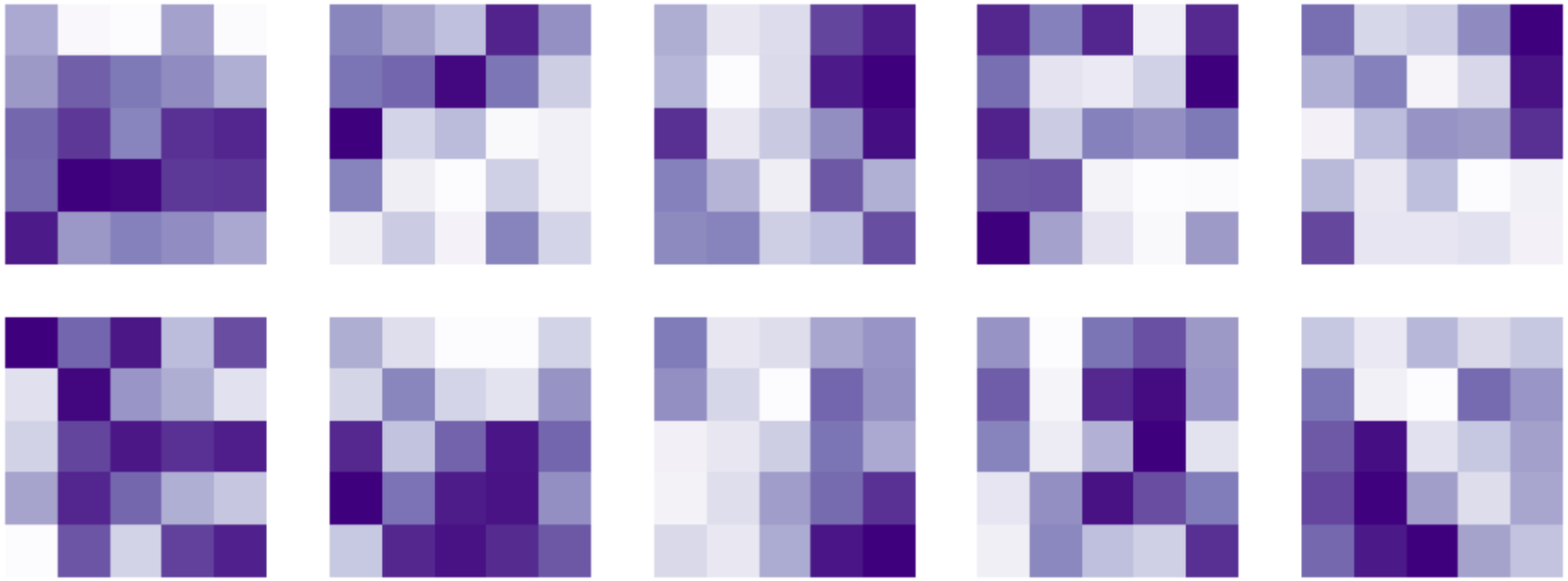
Минутка кода

```
class CNN(nn.Module):
    def __init__(self, input_size, n_feature, output_size):
        super(CNN, self).__init__()
        self.n_feature = n_feature
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=n_feature, kernel_size=5)
        self.conv2 = nn.Conv2d(n_feature, n_feature, kernel_size=5)
        self.fc1 = nn.Linear(n_feature*4*4, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x, verbose=False):
        x = self.conv1(x)
        x = F.relu(x)
        x = F.max_pool2d(x, kernel_size=2)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, kernel_size=2)
        x = x.view(-1, self.n_feature*4*4)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.log_softmax(x, dim=1)
        return x
```

Минутка кода

Выученные фильтры:

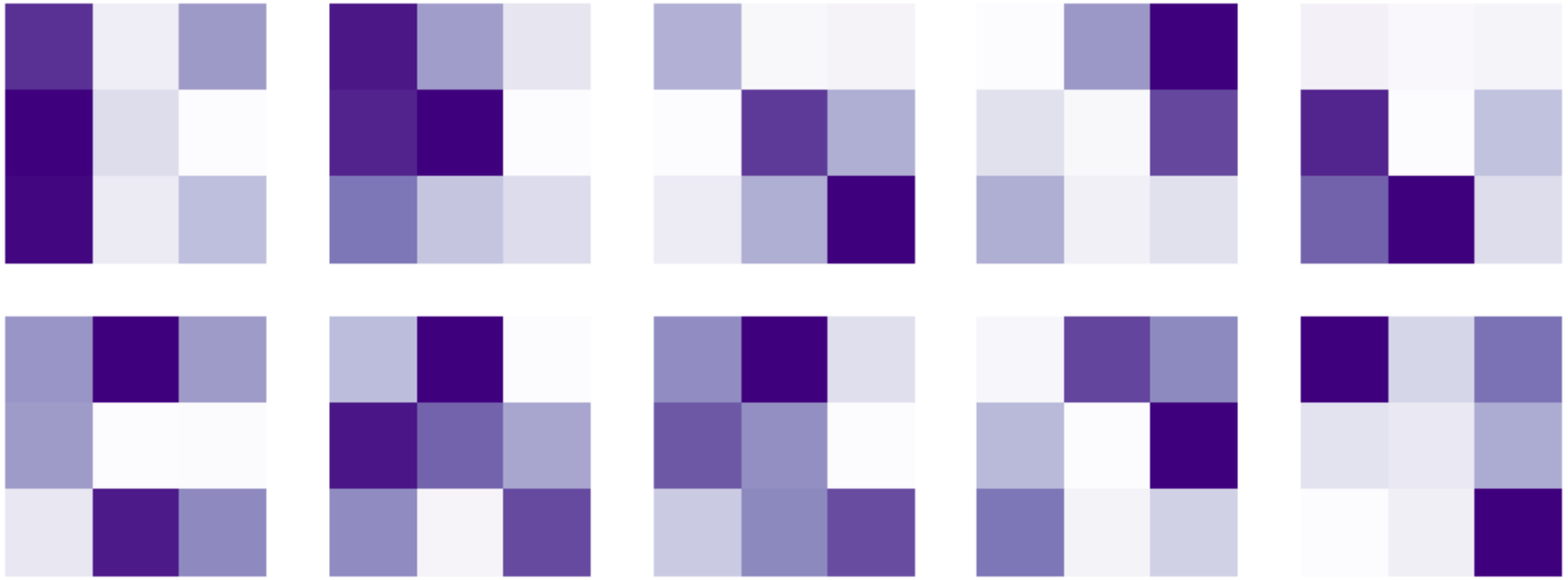


линейная модель: Average loss: 0.2849, Accuracy: 9203/10000 (92%)

свёрточная сеть: Average loss: 0.0784, Accuracy: 9753/10000 (98%)

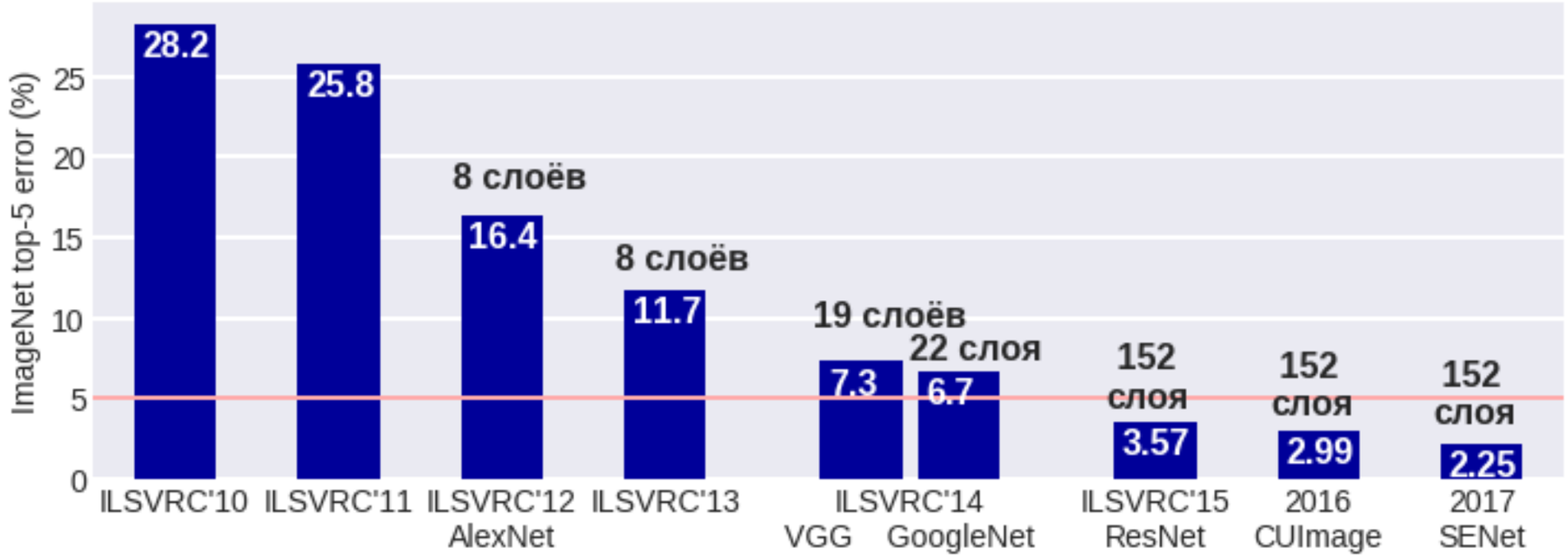
Минутка кода

Если сделать 3×3-свёртки:



но тут маленький датасет, нет аугментаций…

Революция в машинном обучении



ошибка человека – 5.1

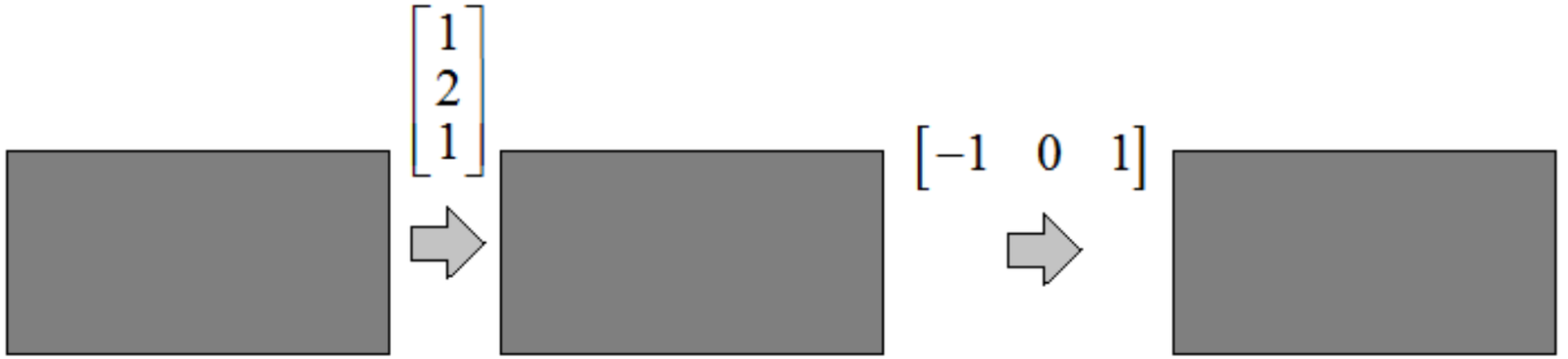
Какие бывают свёртки: Spatial Separable Convolutions

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

идея – факторизовать свёртку,

тогда параметров для обучения свёртки не k^2, а $2k$

не все свёртки так представимы!



проводим сначала $k \times 1$-свёрстку, а потом $1 \times k$

Минутка кода: Spatial Separable Convolutions

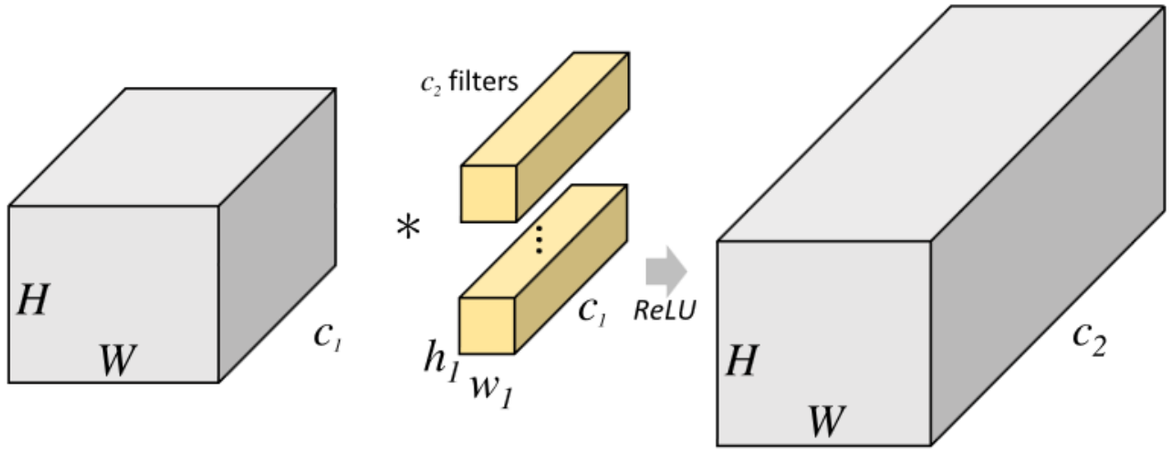
```
conv1k = torch.nn.Conv2d (1, 1, (1, k), bias = False)
convk1 = torch.nn.Conv2d (1, 1, (k, 1), bias = False)
y = convk1 (conv1k (x))
```

Здесь 1 канал → 1 канал,

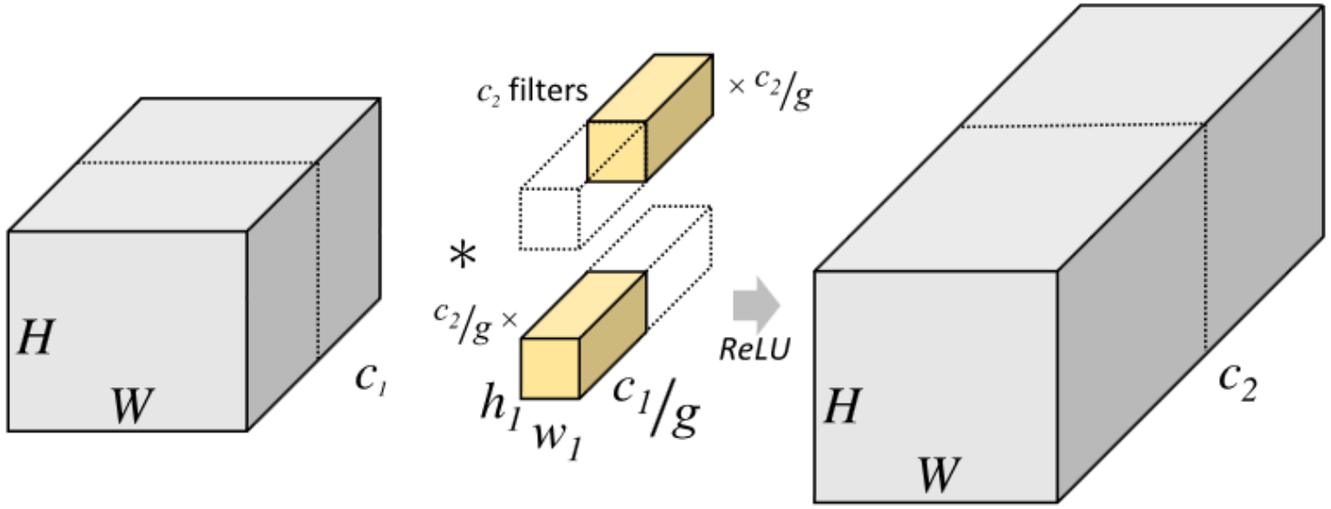
как в общем случае?

Какие бывают свёртки: Group Convolutions

обычная ситуация



с разбиением на две группы

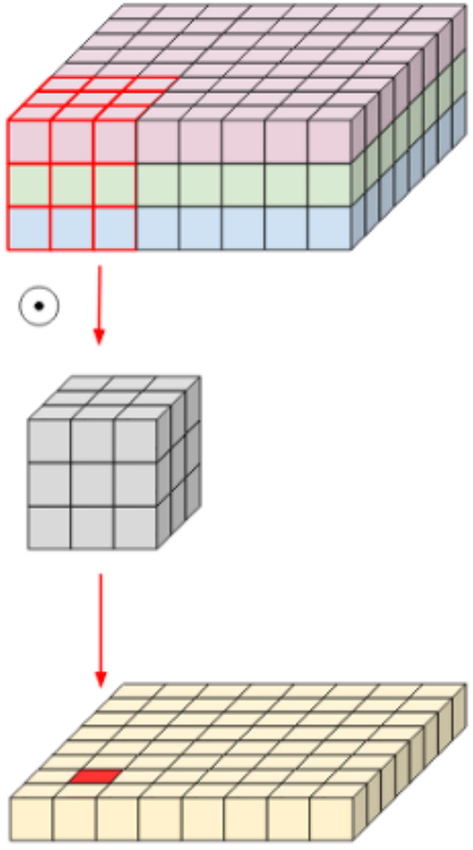


идея из AlexNet, где были ограничения по памяти
могут быть лучшие (разреженные) признаковые представления
но выходные каналы зависят от узкой группы входных

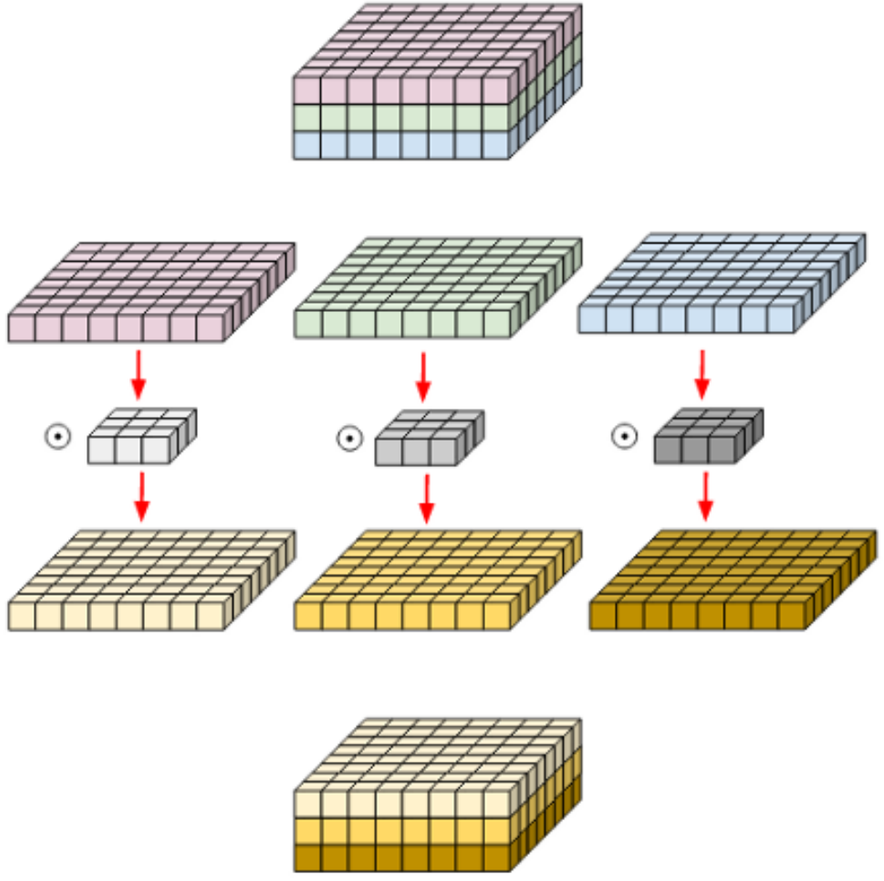
https://blog.yani.io/filter-group-tutorial/

Какие бывают свёртки

convolution



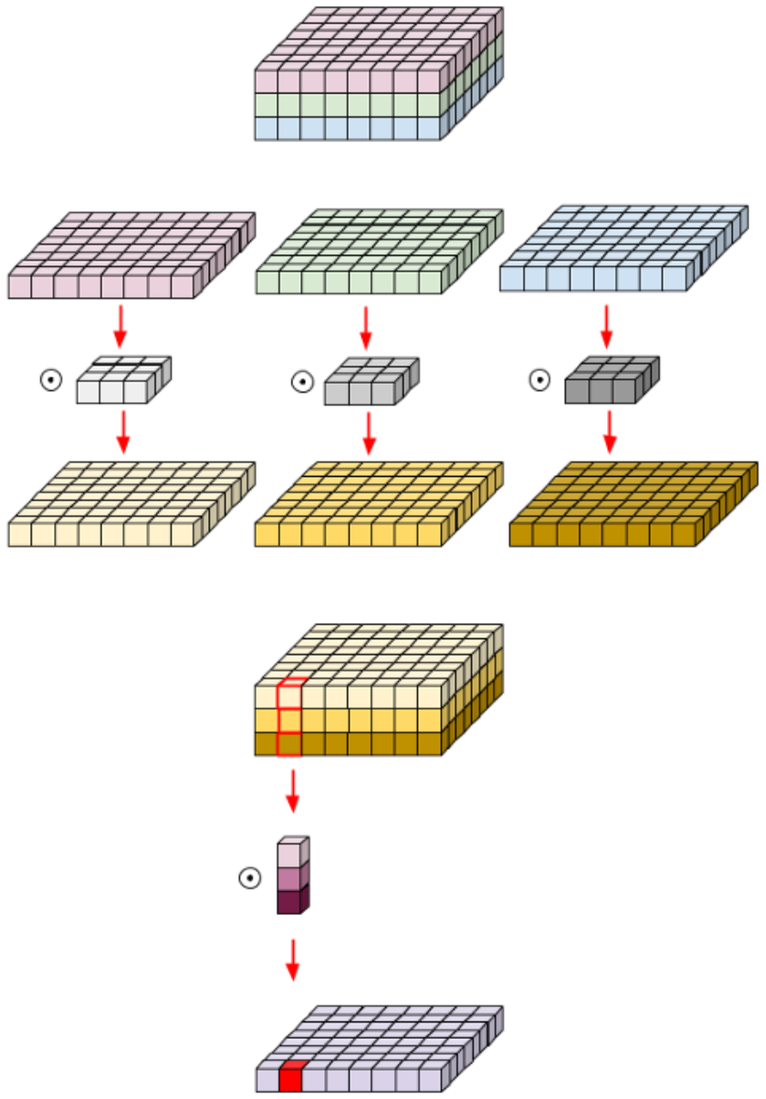
depth-wise convolution



каждый канал «сворачивается» отдельно

https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning/

Какие бывают свёртки: Depth-wise separable convolution



теперь результат зависит от всех каналов

```
S=128, F=3, inC=3, outC=16

Regular convolution:
Parameters:
   3*3*3*16 = 432
Computation cost:
   3*3*3*128*128*16 = ~7e6

Depthwise separable convolution:
Parameters:
   3*3*3+3*16 = 75
Computation cost:
   3*3*3*128*128+128*128*3*16
   = ~1.2e6
```

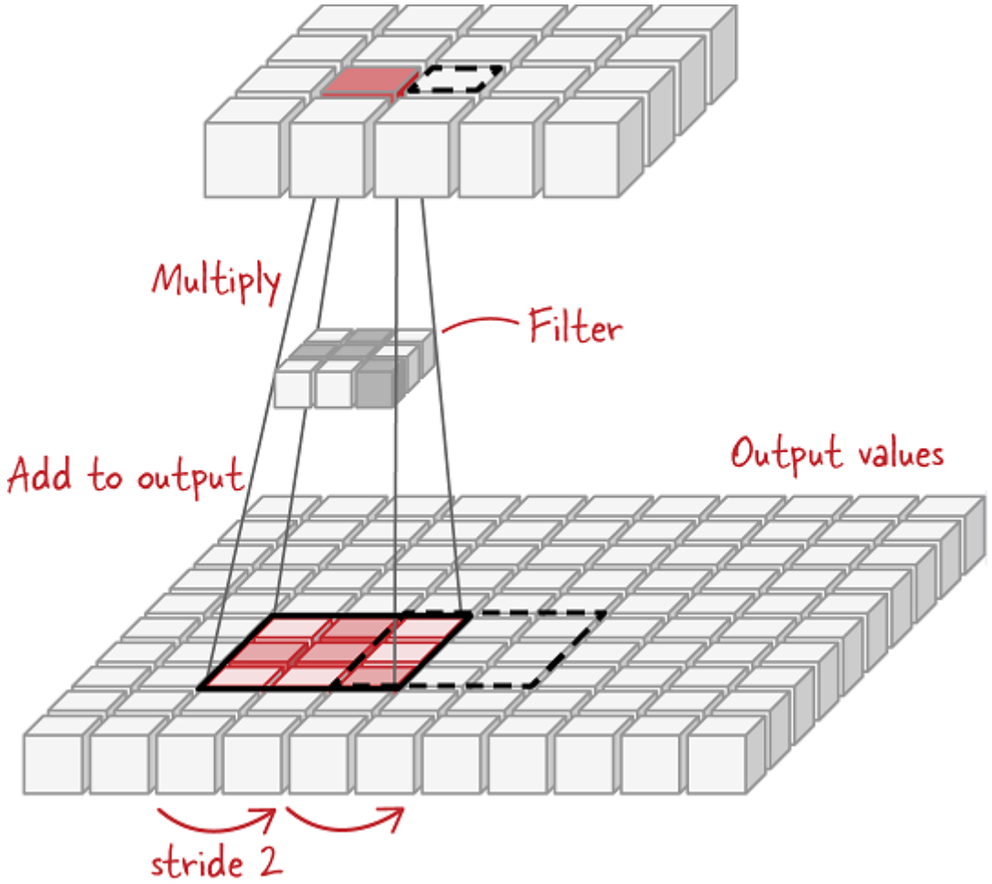
Минутка кода: Depth-wise separable convolution

```
class depthwise_separable_conv(nn.Module):
    def __init__(self, nin, nout):
        super(depthwise_separable_conv, self).__init__()
        self.depthwise = nn.Conv2d(nin, nin,
                                   kernel_size=3,
                                   padding=1,
                                   groups=nin)
        self.pointwise = nn.Conv2d(nin, nout, kernel_size=1)

    def forward(self, x):
        out = self.depthwise(x)
        out = self.pointwise(out)
        return out
```

Мотивация – во многих задачах на разных каналах приходится «примерно одинаково действовать»

Transposed convolution (~~deconvolution~~ / upconvolution / conv-transpose)



термин «deconvolution» считается плохим

~ learnable upsampling operation

[Practical Machine Learning for Computer Vision]

Transposed convolution

Напомним…

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix} * \begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix} = \underbrace{\begin{pmatrix} k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 & 0 \\ 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 \\ 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 \\ 0 & 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} \end{pmatrix}}_{H} \cdot \begin{pmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{31} \\ x_{32} \\ x_{33} \end{pmatrix}$$

Можно для «обратной» операции (увеличивающий тензор) использовать транспонированную матрицу…

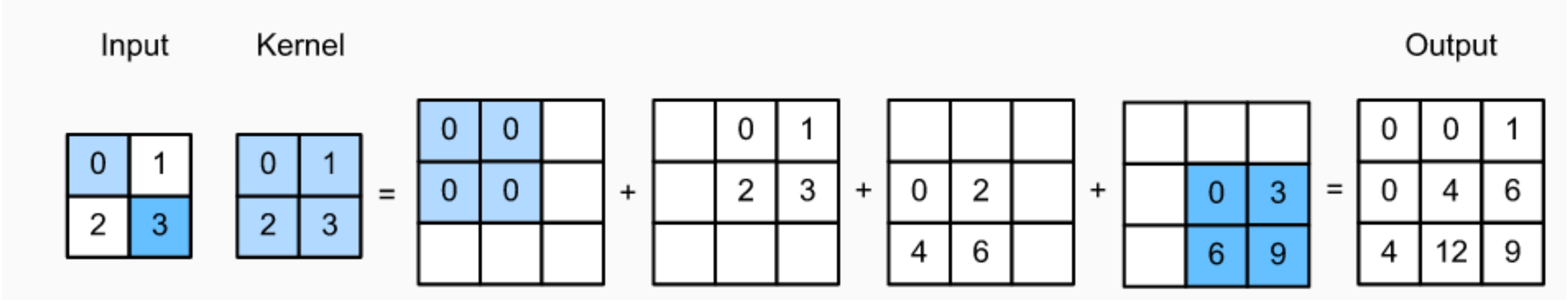
Transposed convolution

$$\begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{pmatrix} *^{\mathrm{T}} \begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix} = H^{\mathrm{T}} \cdot \begin{pmatrix} z_{11} \\ z_{21} \\ z_{12} \\ z_{22} \end{pmatrix} =$$

$$= \begin{pmatrix} k_{11}z_{11}, & k_{12}z_{11} + k_{11}z_{12}, & k_{12}z_{12}, \\ k_{21}z_{11} + k_{11}z_{21}, & k_{22}z_{11} + k_{21}z_{12} + k_{12}z_{21} + k_{11}z_{22} & k_{22}z_{12} + k_{12}z_{22}, \\ k_{21}z_{21}, & k_{22}z_{21} + k_{21}z_{22}, & k_{22}z_{22} \end{pmatrix}$$

«Обратная» свёртка увеличивает пространственное разрешение…

(можно увеличить изображение с помощью НС), эквивалентная запись:



Минутка кода: Transposed convolution

```
H = torch.arange(1, 17).float().view(1, 1, 4, 4)
print (H)
```

```
tensor([[[[ 1.,  2.,  3.,  4.],
          [ 5.,  6.,  7.,  8.],
          [ 9., 10., 11., 12.],
          [13., 14., 15., 16.]]]])
```

```
f = nn.ConvTranspose2d(in_channels=1, out_channels=1, kernel_size=2, bias=False)
f.weight.data.fill_(1.)
H2 = f(H)
print (H2, H2.shape)
```

```
tensor([[[[ 1.,  3.,  5.,  7.,  4.],
          [ 6., 14., 18., 22., 12.],
          [14., 30., 34., 38., 20.],
          [22., 46., 50., 54., 28.],
          [13., 27., 29., 31., 16.]]]], grad_fn=<SlowConvTranspose2DBackward>)
torch.Size([1, 1, 5, 5])
```

Минутка кода: Transposed convolution

```
H = torch.randn(1, 10, 20, 30)
cnv = nn.Conv2d(in_channels=10, out_channels=20, kernel_size=2)
ct = nn.ConvTranspose2d(in_channels=20, out_channels=10, kernel_size=2)
print (H.shape)
print (cnv(H).shape)
print (ct(cnv(H)).shape)
```

```
torch.Size([1, 10, 20, 30])
torch.Size([1, 20, 19, 29])
torch.Size([1, 10, 20, 30])
```

Dropout в свёрточных сетях

соседние пиксели коррелированы ⇒ по-другому надо dropout

Spatial Dropout / Dropout2D – выбрасываем каналы, а не нейроны

[Tompson et al. (2015)] но нельзя реализовать как отдельный слой;)

cutout – маска-прямоугольник

но оказалось, что лучше применять на исходном изображении как аугментацию

Stochastic depth – удаление слоёв (дальше)

Pooling Dropout

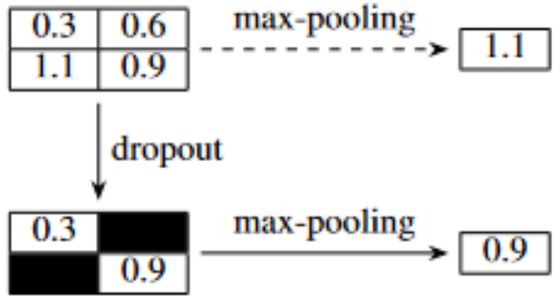


Figure 4: Max-pooling dropout in convolutional neural networks [12].

Минутка кода: Dropout

```
from torch import nn
H = torch.arange(1, 17).reshape(1, 4, 2, 2).float()
drop = nn.Dropout2d(p=0.5) # Dropout 2D - зануление каналов

print(drop(H))
```

```
tensor([[[[ 0.,  0.],
          [ 0.,  0.]],

         [[10., 12.],
          [14., 16.]],

         [[18., 20.],
          [22., 24.]],

         [[ 0.,  0.],
          [ 0.,  0.]]]])
```

Почему неожиданно большие значения?

Dropout Layers

`nn.Dropout`	During training, randomly zeroes some of the elements of the input tensor with probability `p` using samples from a Bernoulli distribution.
`nn.Dropout2d`	Randomly zero out entire channels (a channel is a 2D feature map, e.g., the j-th channel of the i-th sample in the batched input is a 2D tensor $\text{input}[i, j]$).
`nn.Dropout3d`	Randomly zero out entire channels (a channel is a 3D feature map, e.g., the j-th channel of the i-th sample in the batched input is a 3D tensor $\text{input}[i, j]$).
`nn.AlphaDropout`	Applies Alpha Dropout over the input.

Итог

В изображениях свёртки – естественная операция

- классическая линейная операция
- поиск паттернов
- реализация фильтра
- разделение параметров
- реализация разреженных взаимодействий (sparse interactions)

Естественное устройство CNN: n×[conv + activ + pool] + k×FC

какой порядок лучше в нелинейность + пулинг?

В отличие от классического CV не придумываем фильтры

Они обучаются сами!

Свёртка – первый пример разделения весов.

Есть способы экономии параметров – и ими пользуются!

Свёртки продолжают совершенствоваться

(более разумные представления, экономия параметров)

Литература

Vincent Dumoulin, Francesco Visin «A guide to convolution arithmetic for deep learning» // https://arxiv.org/pdf/1603.07285v1.pdf