курс «Глубокое обучение»

Свёрточные нейронные сети

Александр Дьяконов

26 сентября 2022 года

План

Что такое изображение

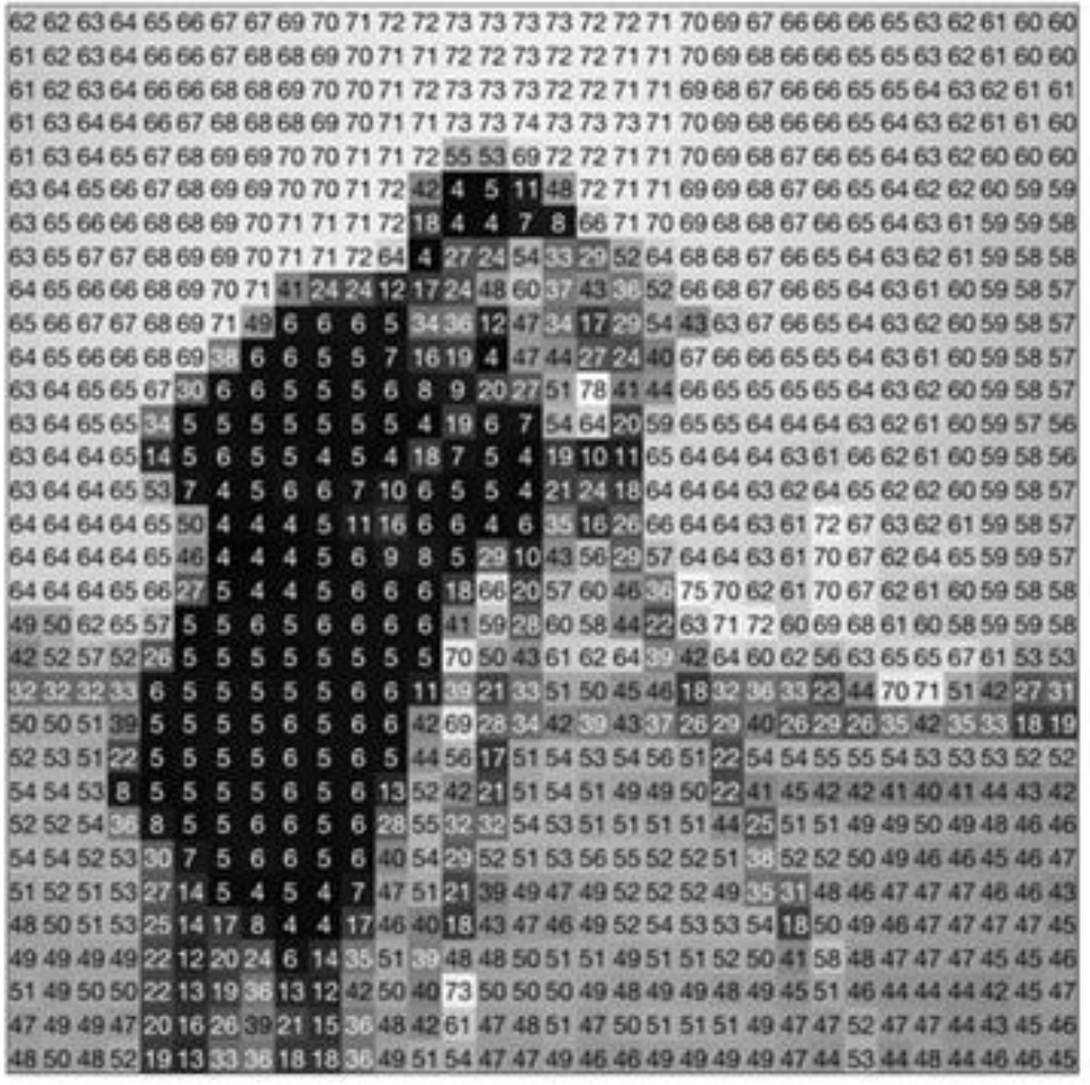
Что такое свёртка, пулинг

Свёрточная сеть – CNN

Какие бывают свёртки

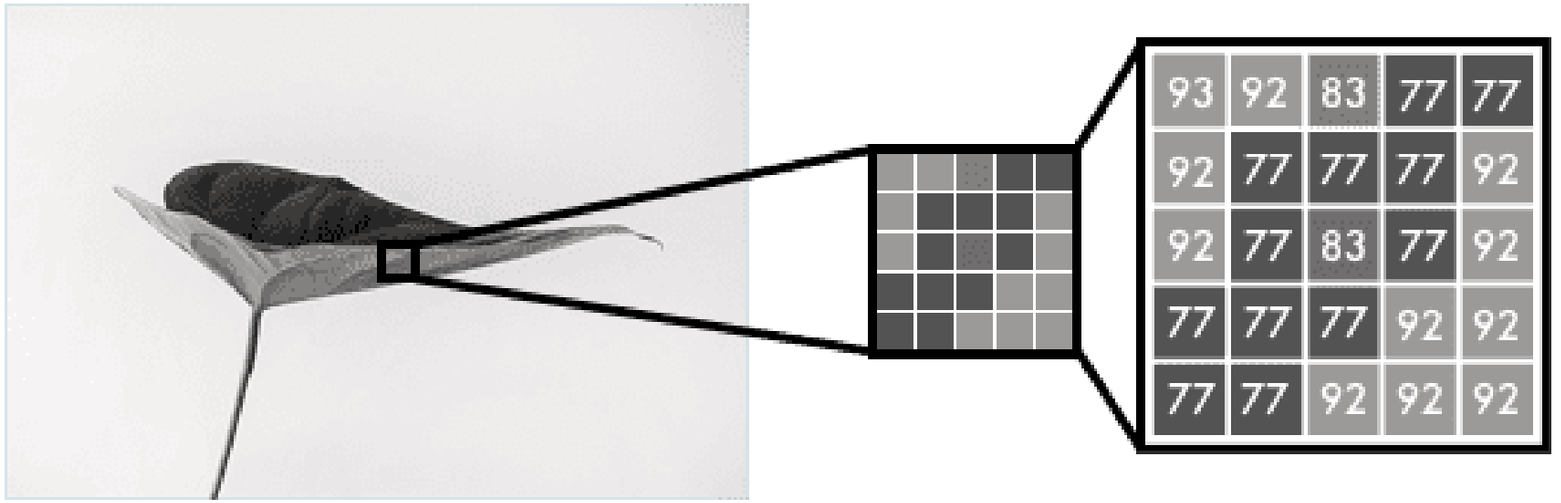
Архитектуры-чемпионы на ImageNet-е

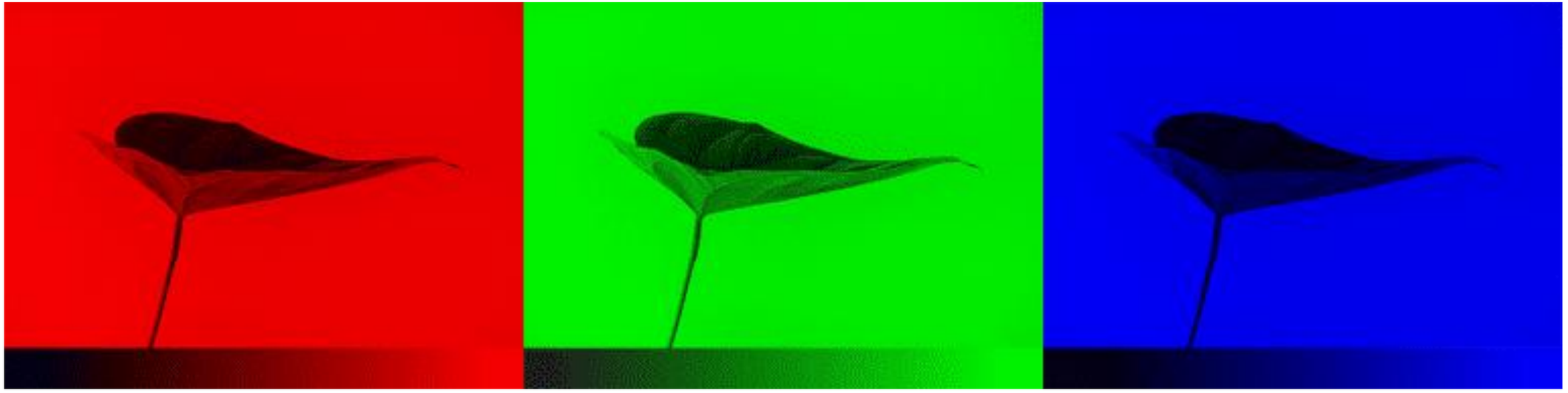
Что такое изображение – H×W-матрица



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

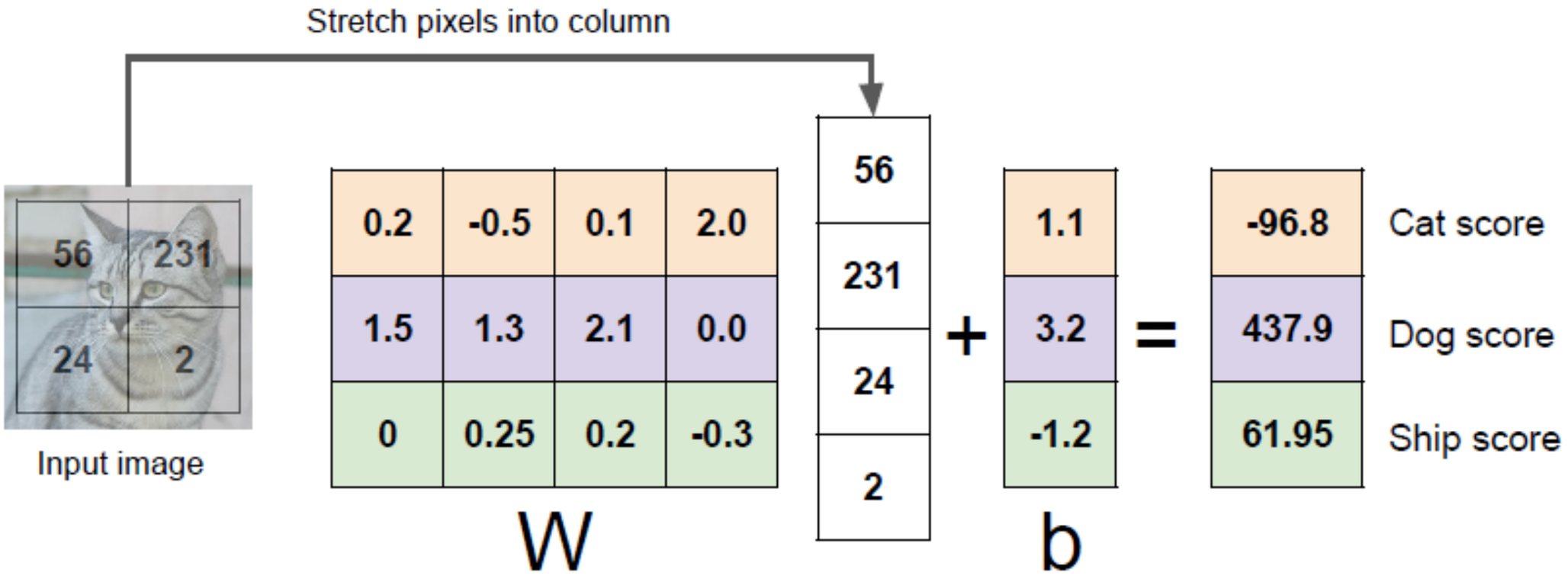
Что такое изображение – трёхмерный C×H×W -тензор





цветное – 3-х мерная целочисленная матрица (тензор)

Линейный подход к классификации на несколько классов



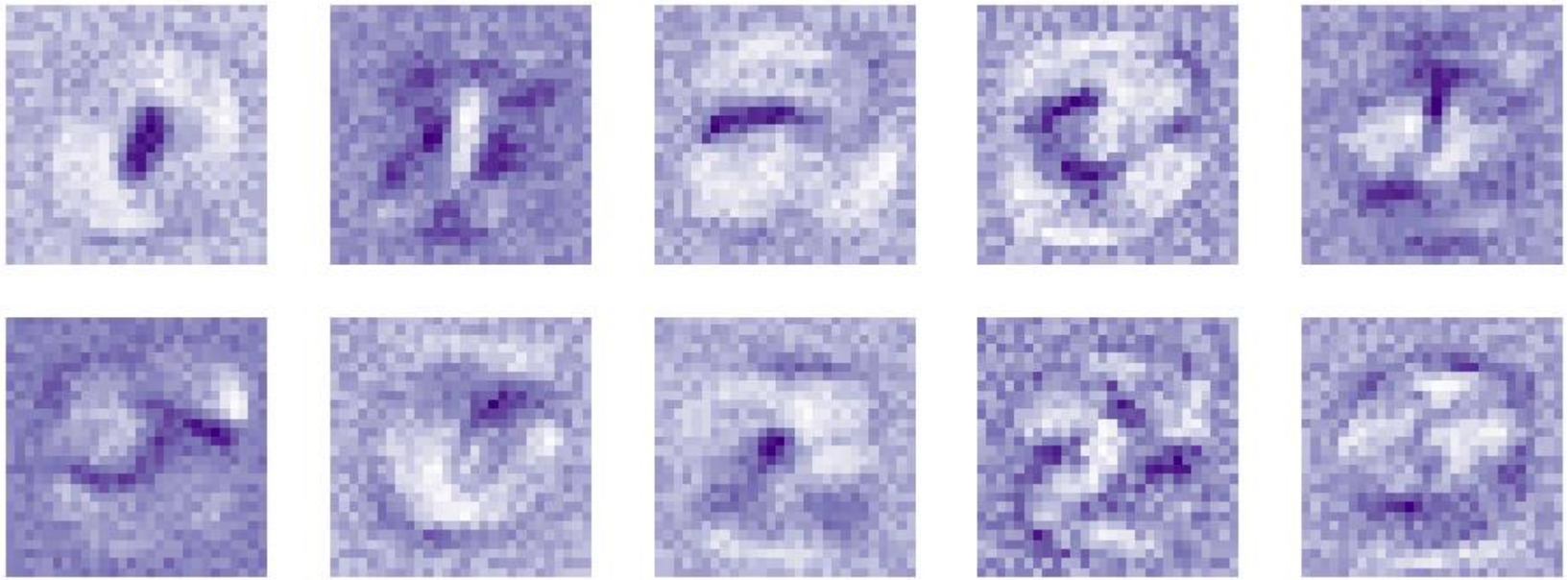
Изображение → вытянуть в вектор признаков

3 класса = 3 вектора весов

линейно получаем оценки за классы (класс по max оценке)

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

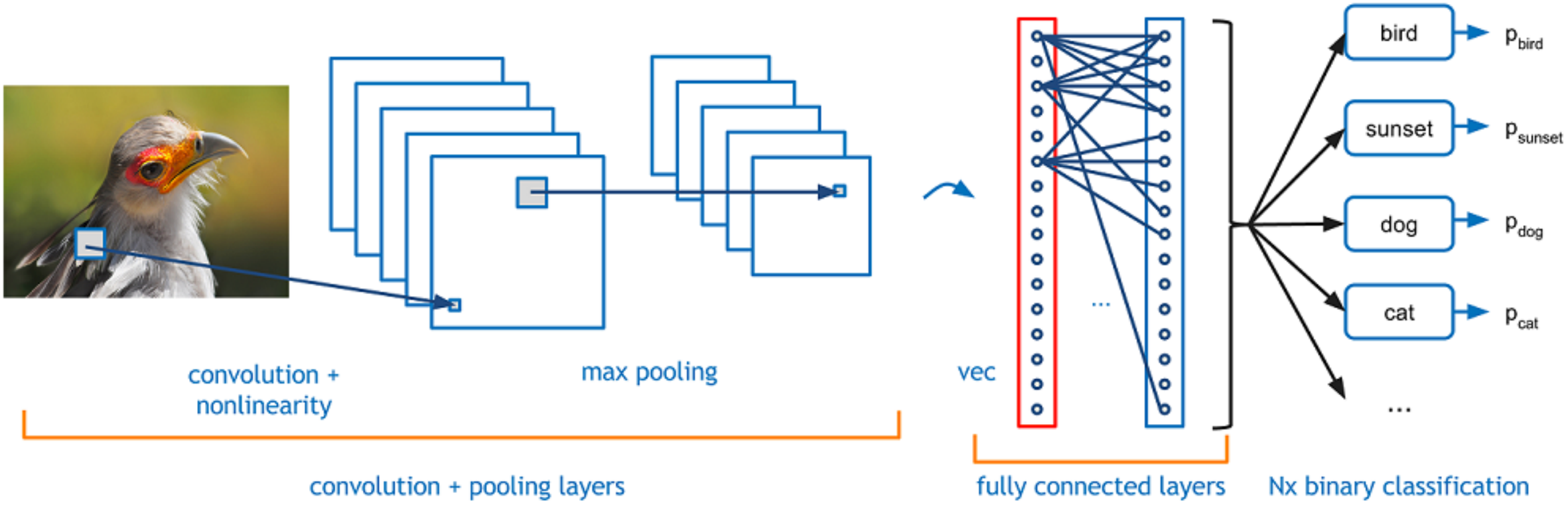
- **слишком много параметров в простой задаче!**

если изображение 256×256×3 ~ 200k,

то чтобы изображение $\rightarrow$ изображение

надо $3.9 \cdot 10^9$ параметров!

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/

2-D свёртка (Convolution)

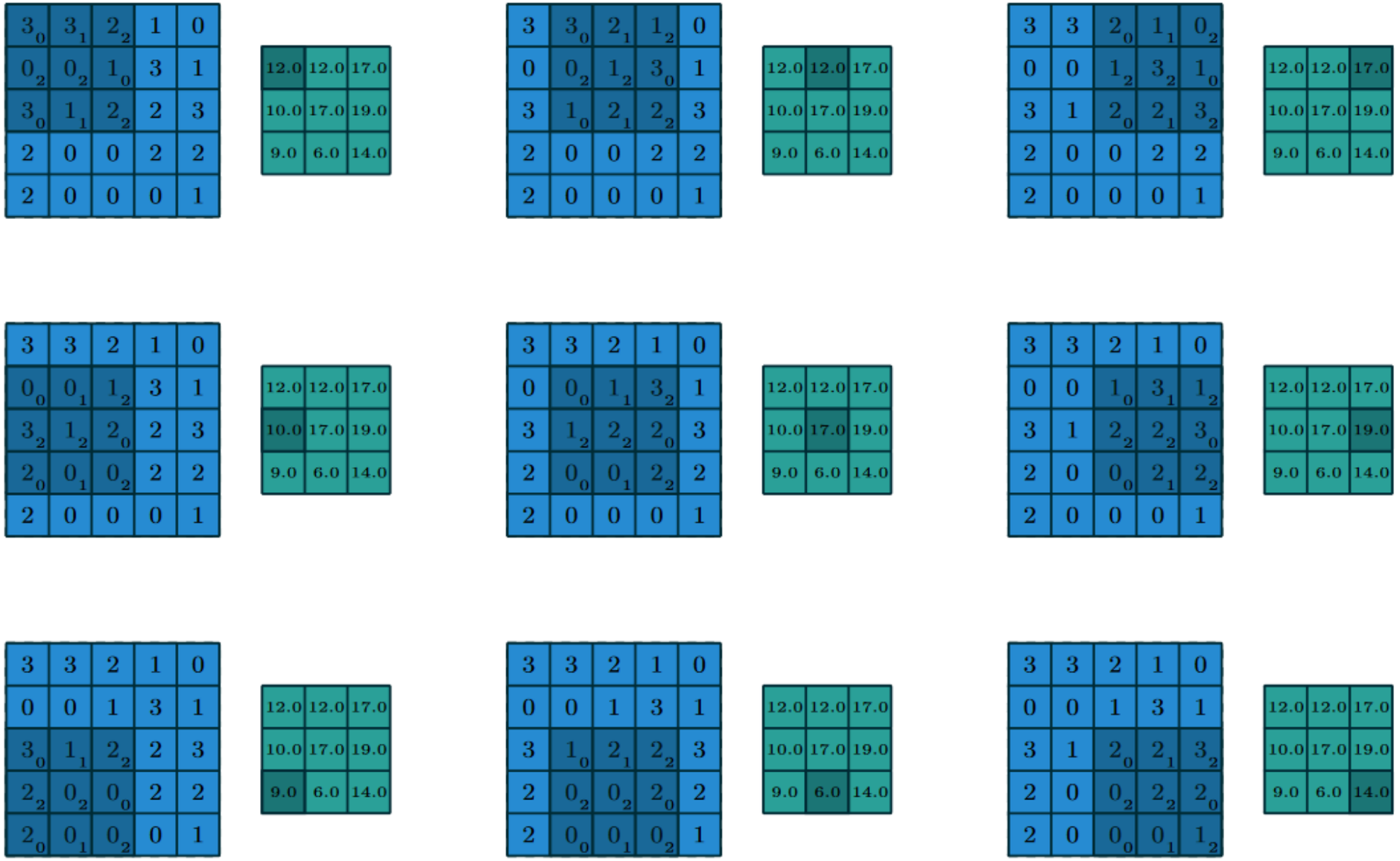
$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{r} K_{ij} I_{x+i-1, y+j-1}$$

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

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

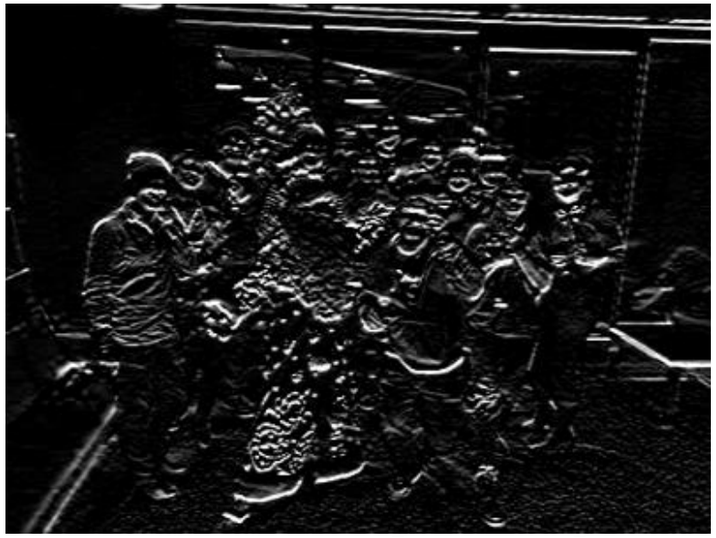
оригинал

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

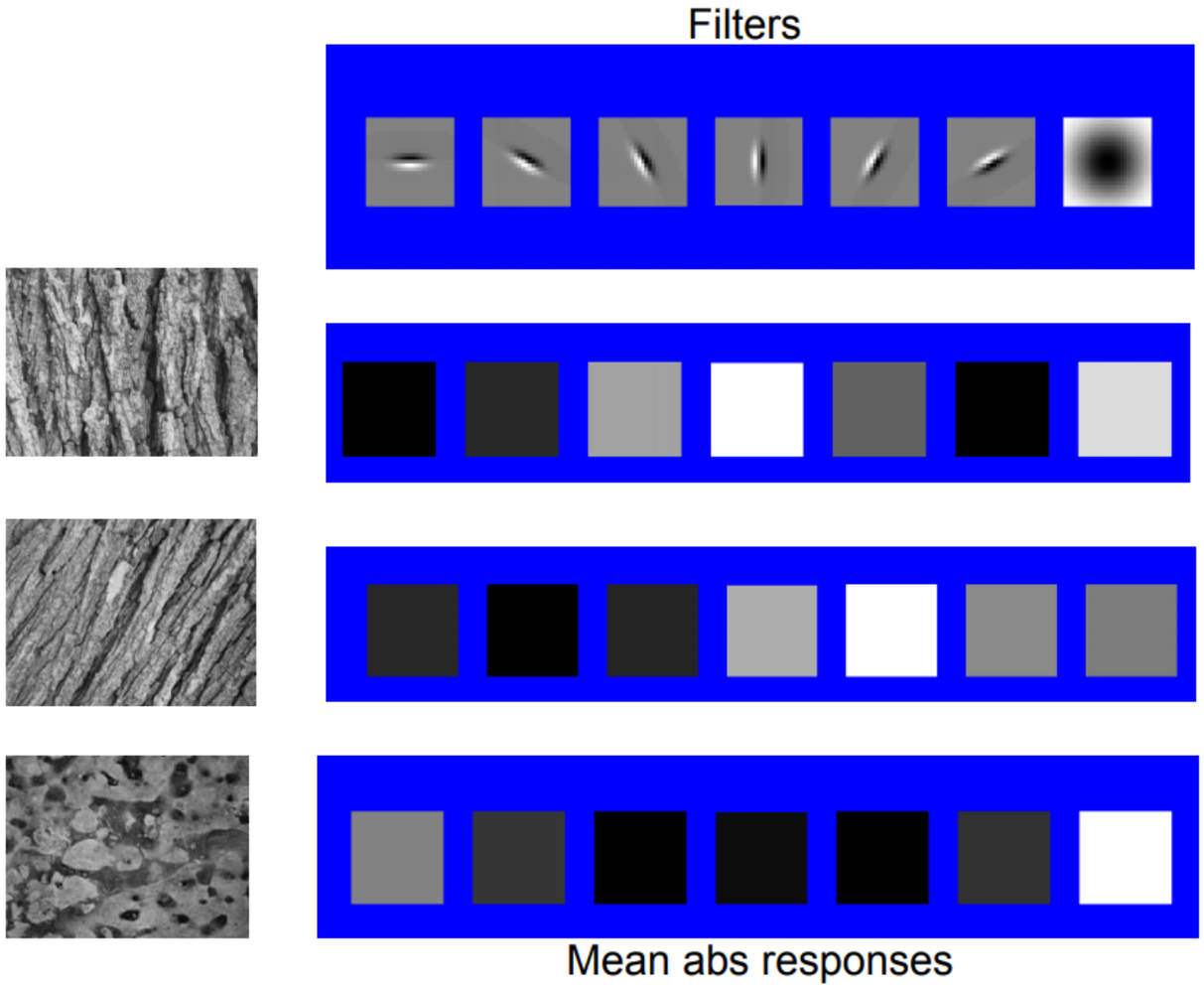
outline $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

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

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

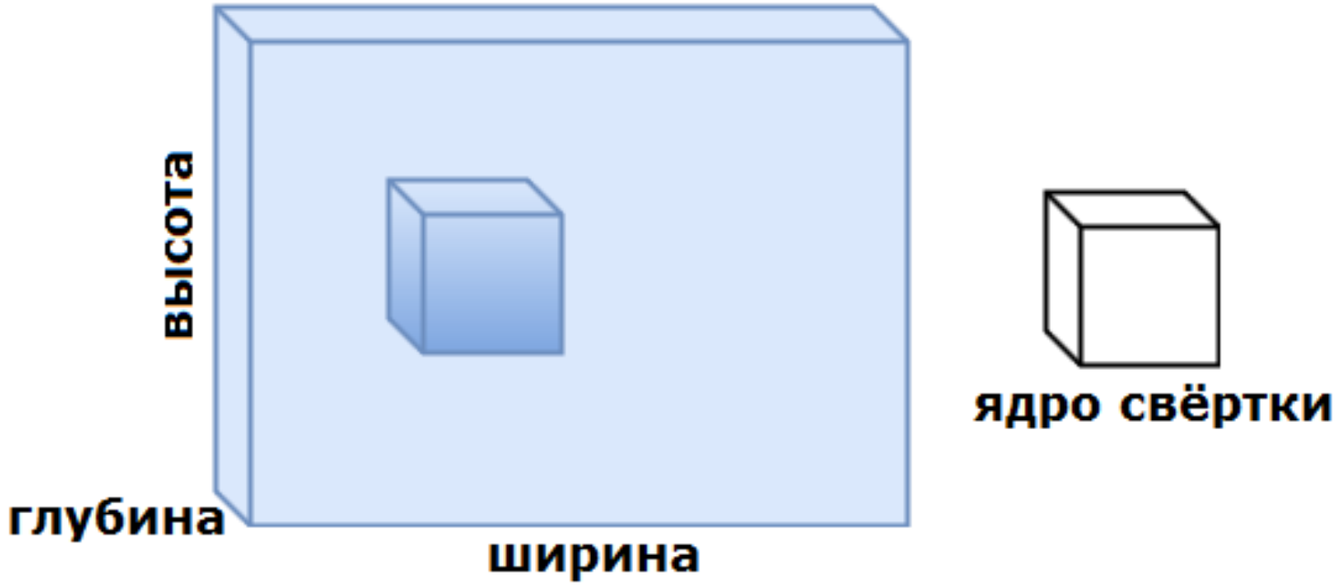
custom $\begin{bmatrix} +1 & -1 & +1 \\ -1 & 0 & -1 \\ +1 & -1 & +1 \end{bmatrix}$

Фильтры для текстур



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

Свёртка (Convolution)



Глубина (depth) / число каналов

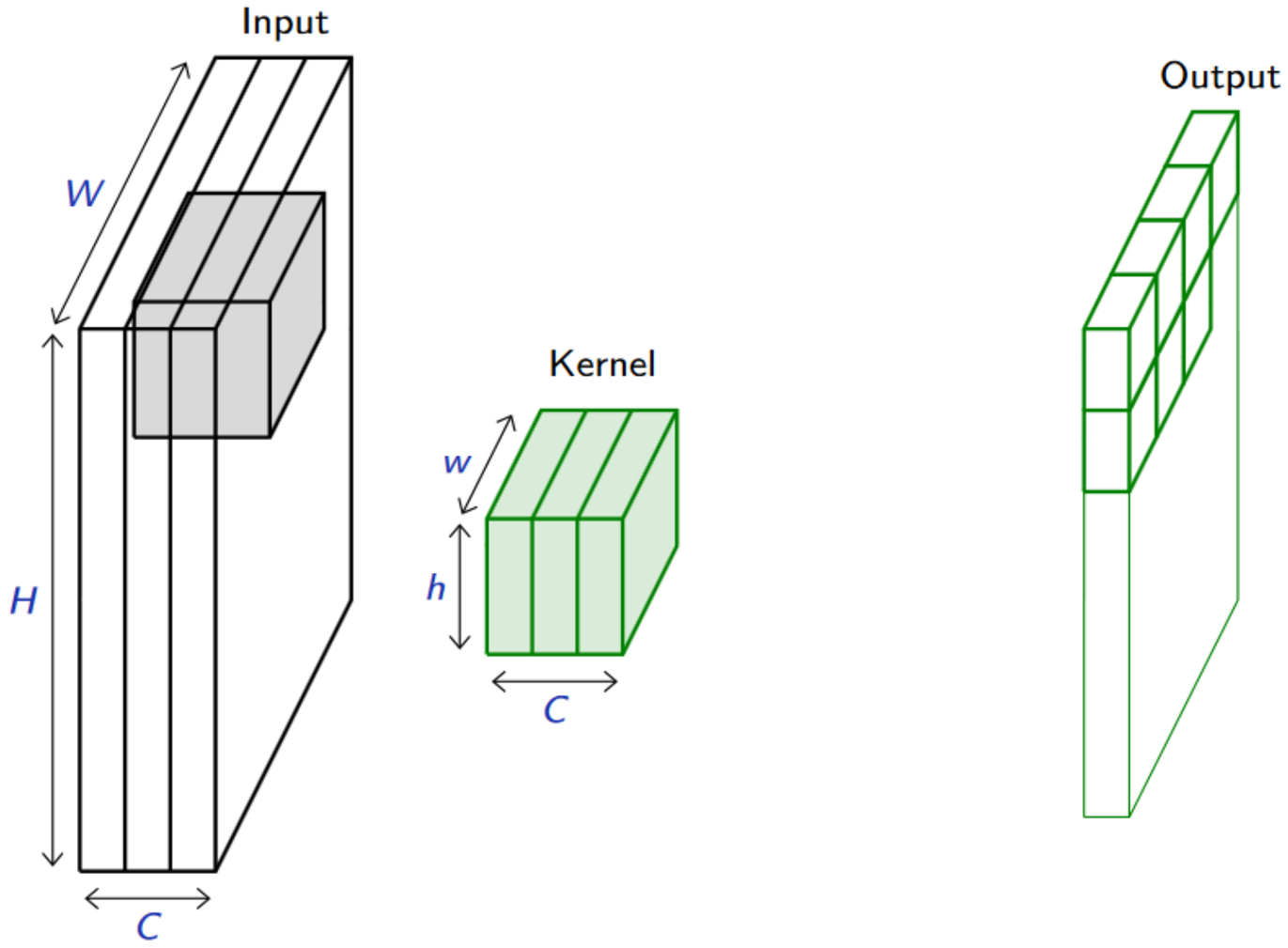
Высота (height) и ширина (width) тензора (изображения) / ядра

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

Отступ (padding) – для дополнения изображения нулями по краям

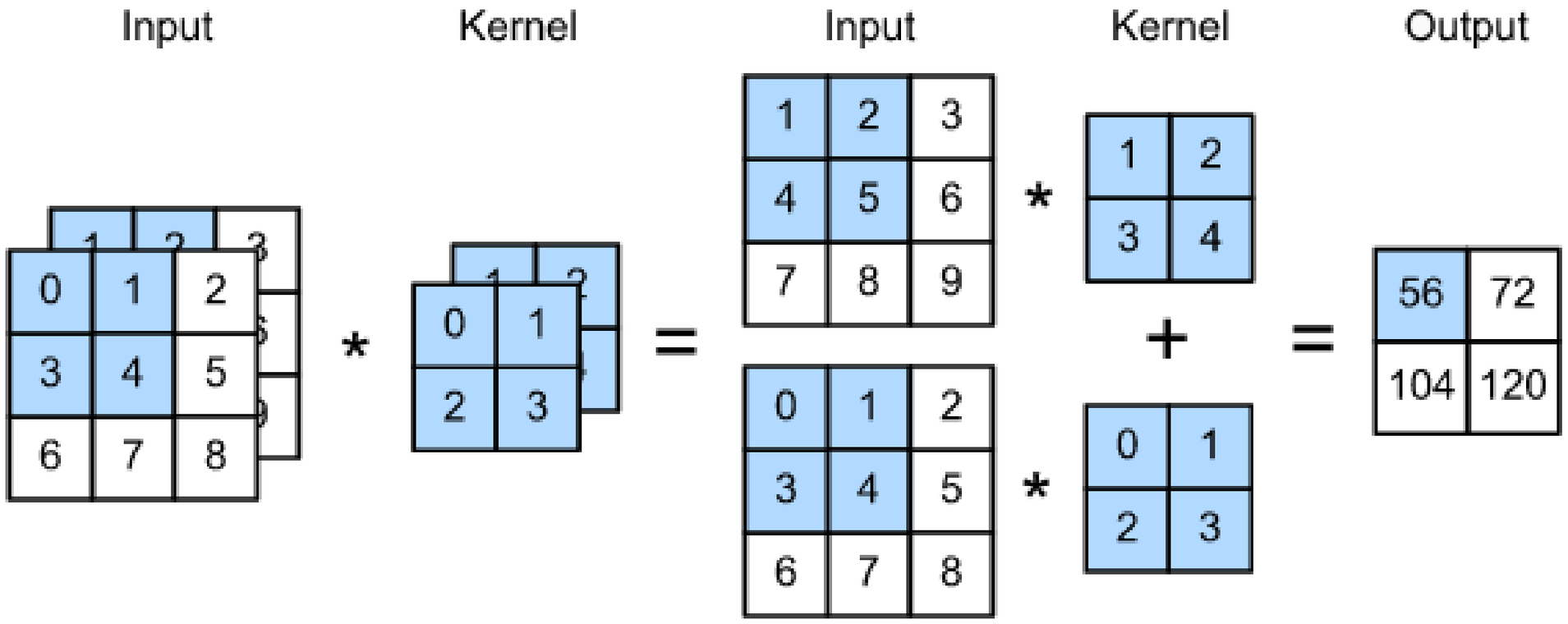
Ядро (kernel) или фильтр (filter) – размерность как у предыдущего тензора; в 3D длина и ширина меньше (глубина совпадает)

Свёртка (Convolution): глубина



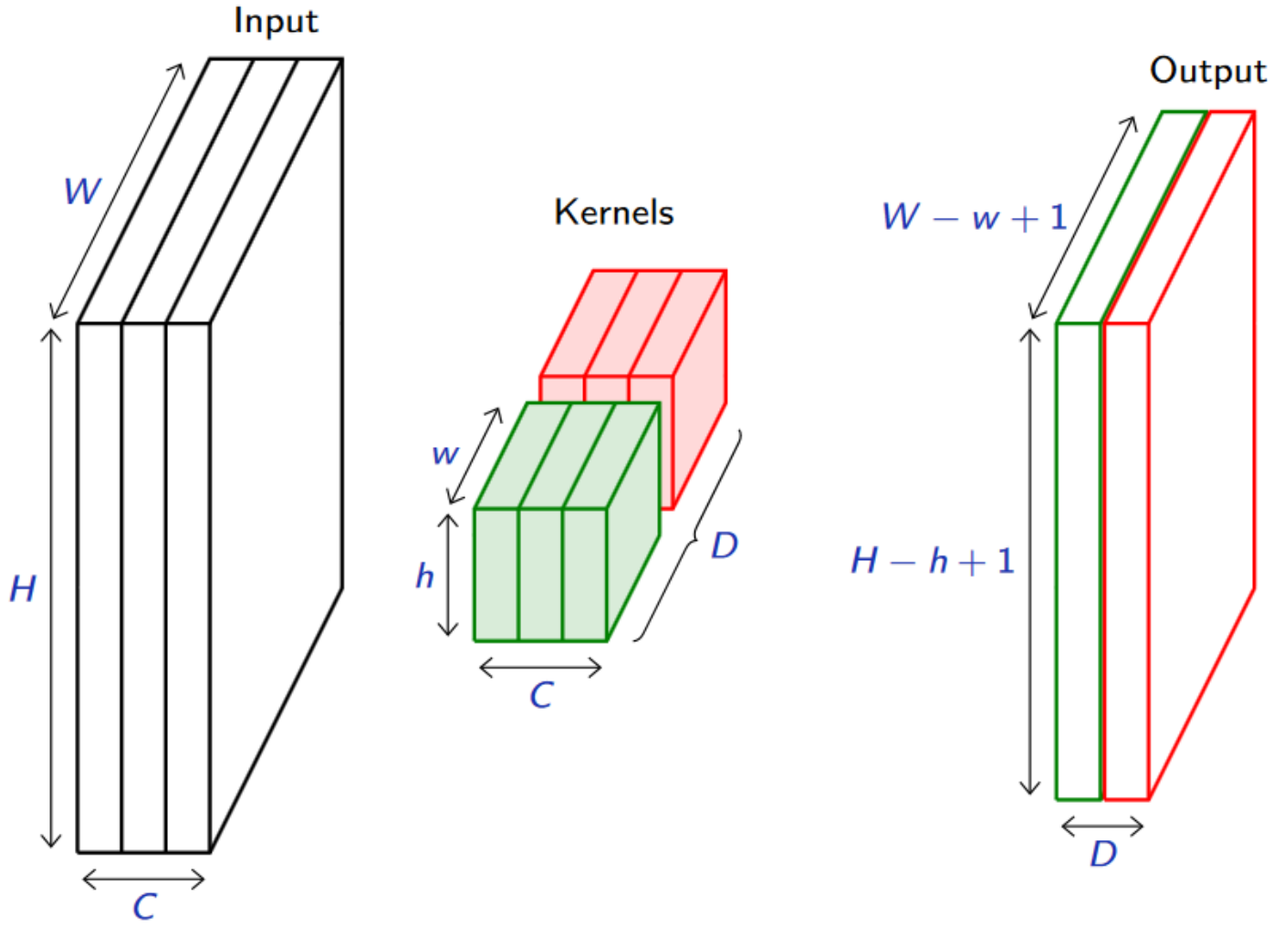
глубина тензора (число каналов) = глубина свёртки

Свёртка (Convolution): глубина



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

Свёртка (Convolution): применение нескольких свёрток

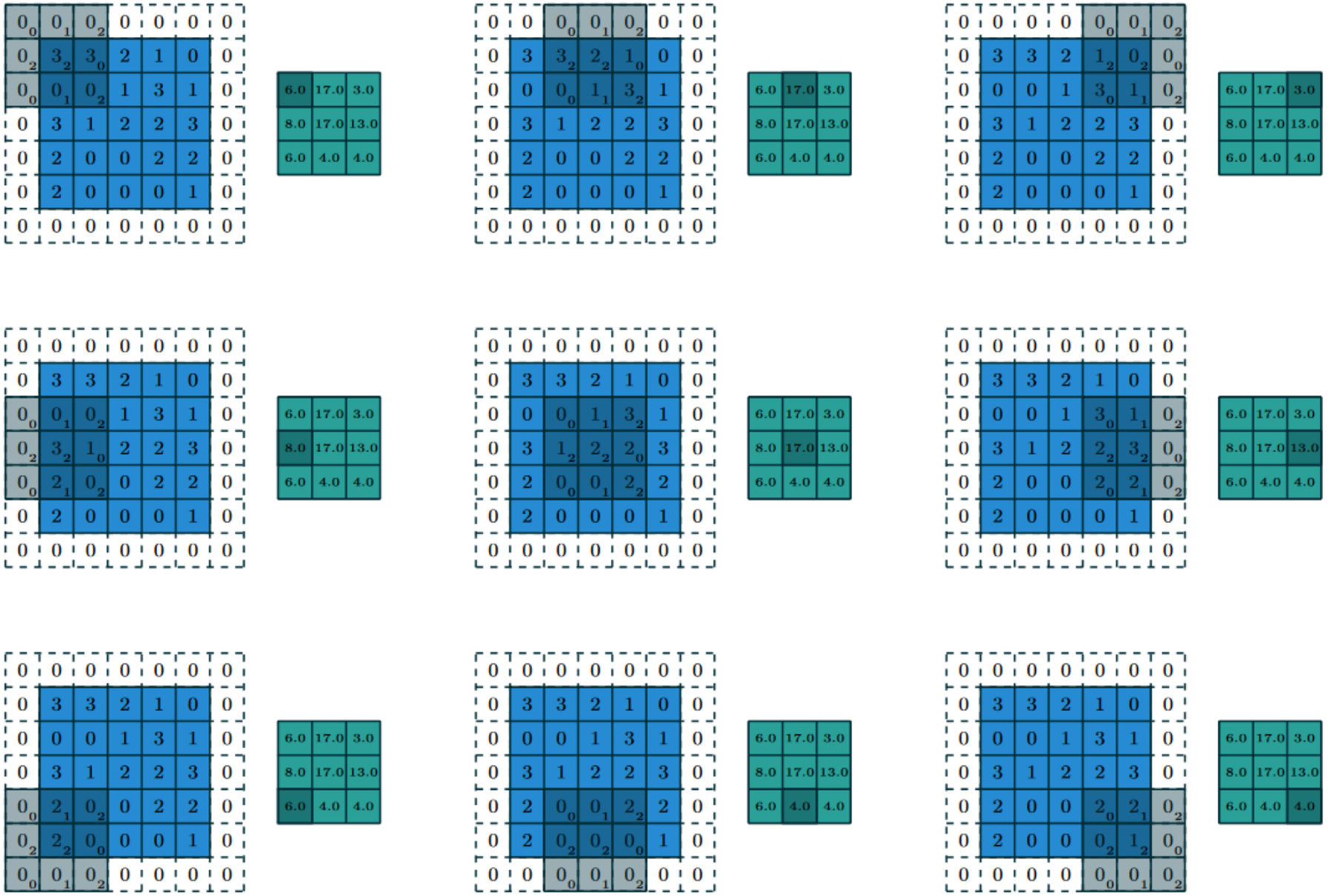


каждая свёртка – 1 «лист» на выходе, k свёрток – k-канальный выход

получаем на выходе тензор, глубина = число применяемых свёрток

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

Свёртка с отступами (padding) и шагом (stride)



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)

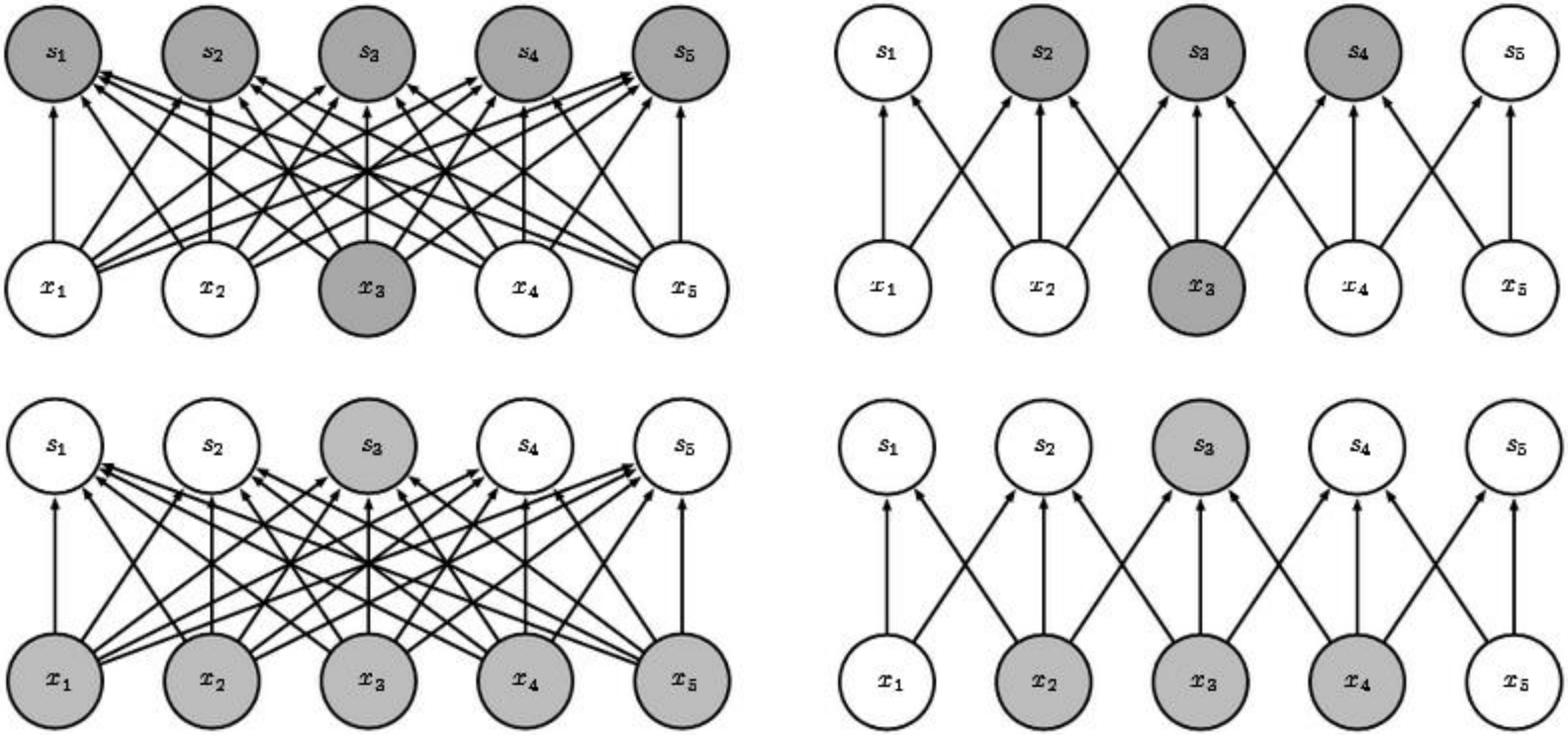
Реализация: свёртка – это линейная операция (нужно быстро умножать матрицы)

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

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

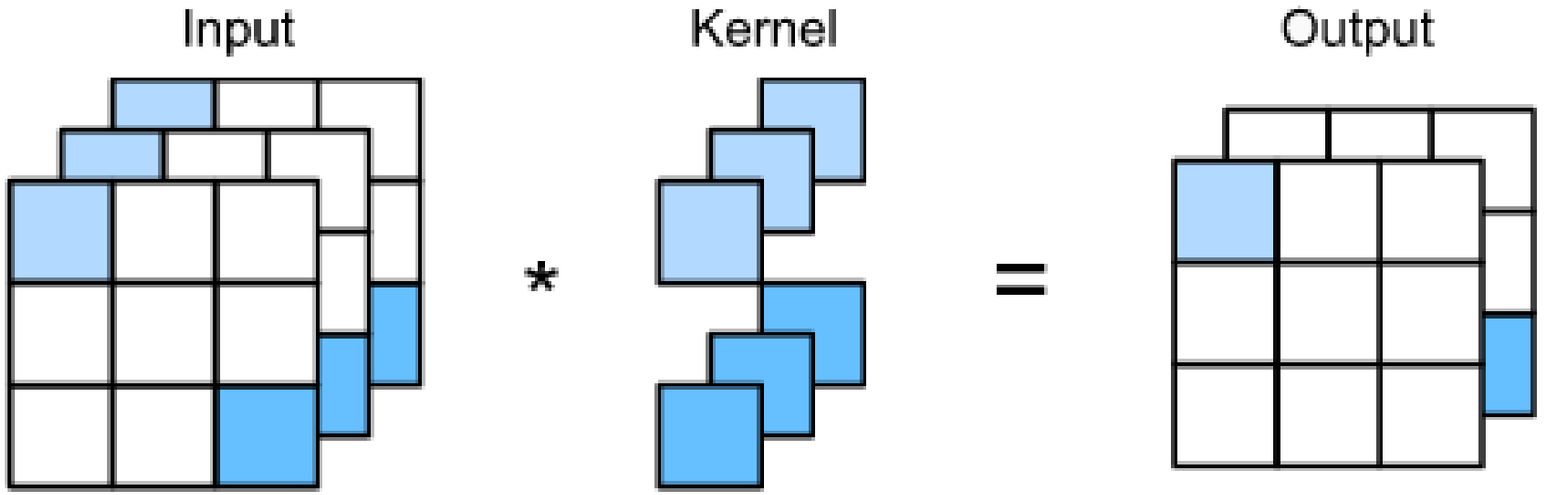
$$\sim \begin{bmatrix} k_{11}x_{11} + k_{12}x_{12} + k_{21}x_{21} + k_{22}x_{22} & k_{11}x_{12} + k_{12}x_{13} + k_{21}x_{22} + k_{22}x_{23} \\ k_{11}x_{21} + k_{12}x_{22} + k_{21}x_{31} + k_{22}x_{32} & k_{11}x_{22} + k_{12}x_{23} + k_{21}x_{32} + k_{22}x_{33} \end{bmatrix}$$

Разреженные взаимодействия (sparse interactions)



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

Смысл свёрток 1×1 (Pointwise Convolution)



линейная комбинация

одинаковая для каждого пикселя

«вдоль каналов» (признаков)

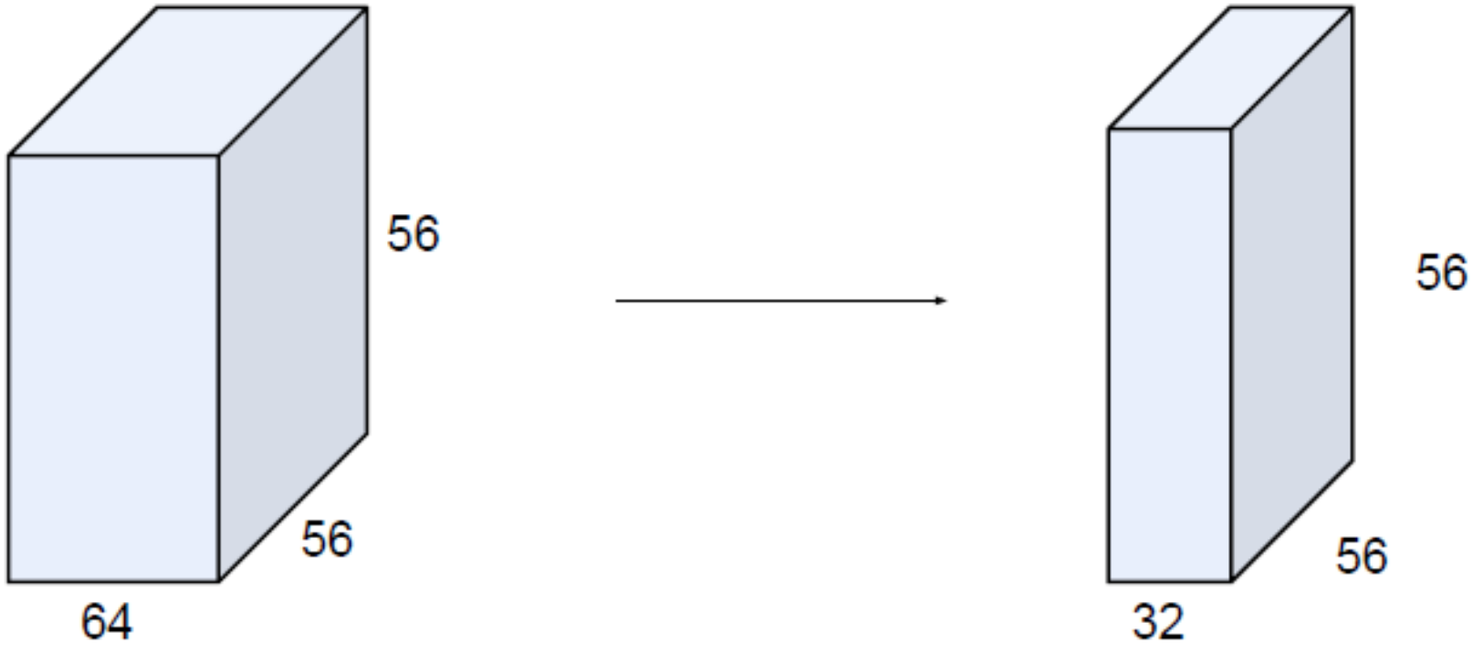
потом может (должна) идти нелинейность

$\Rightarrow$ 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!

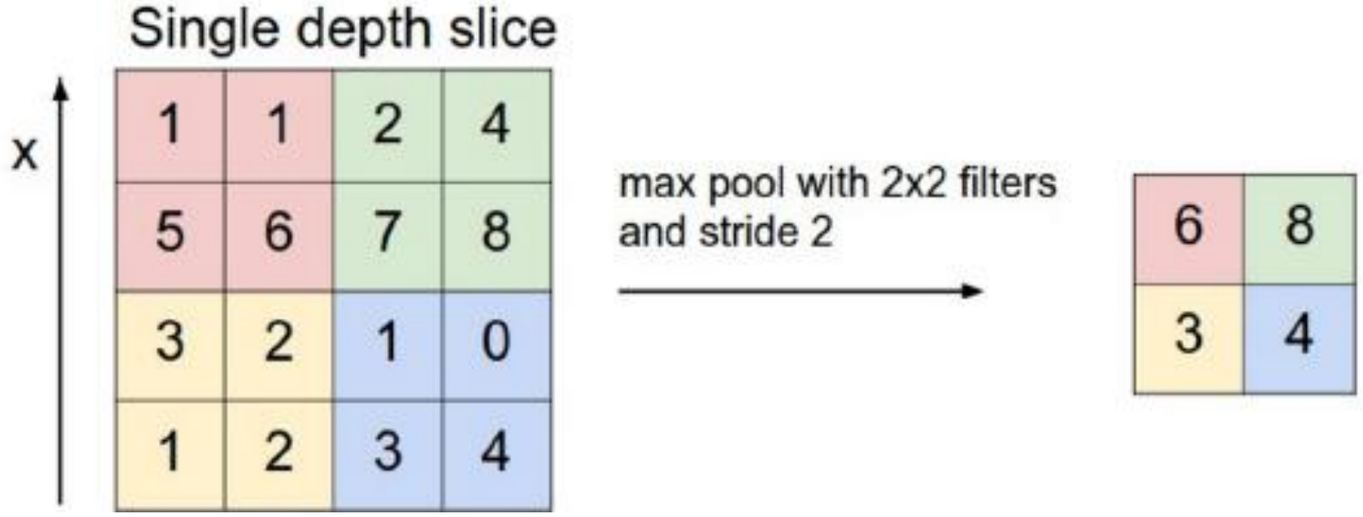
будет часто использоваться – это своеобразная мини-нейронка

+ изменение числа каналов

+ «узкое горло» в НС

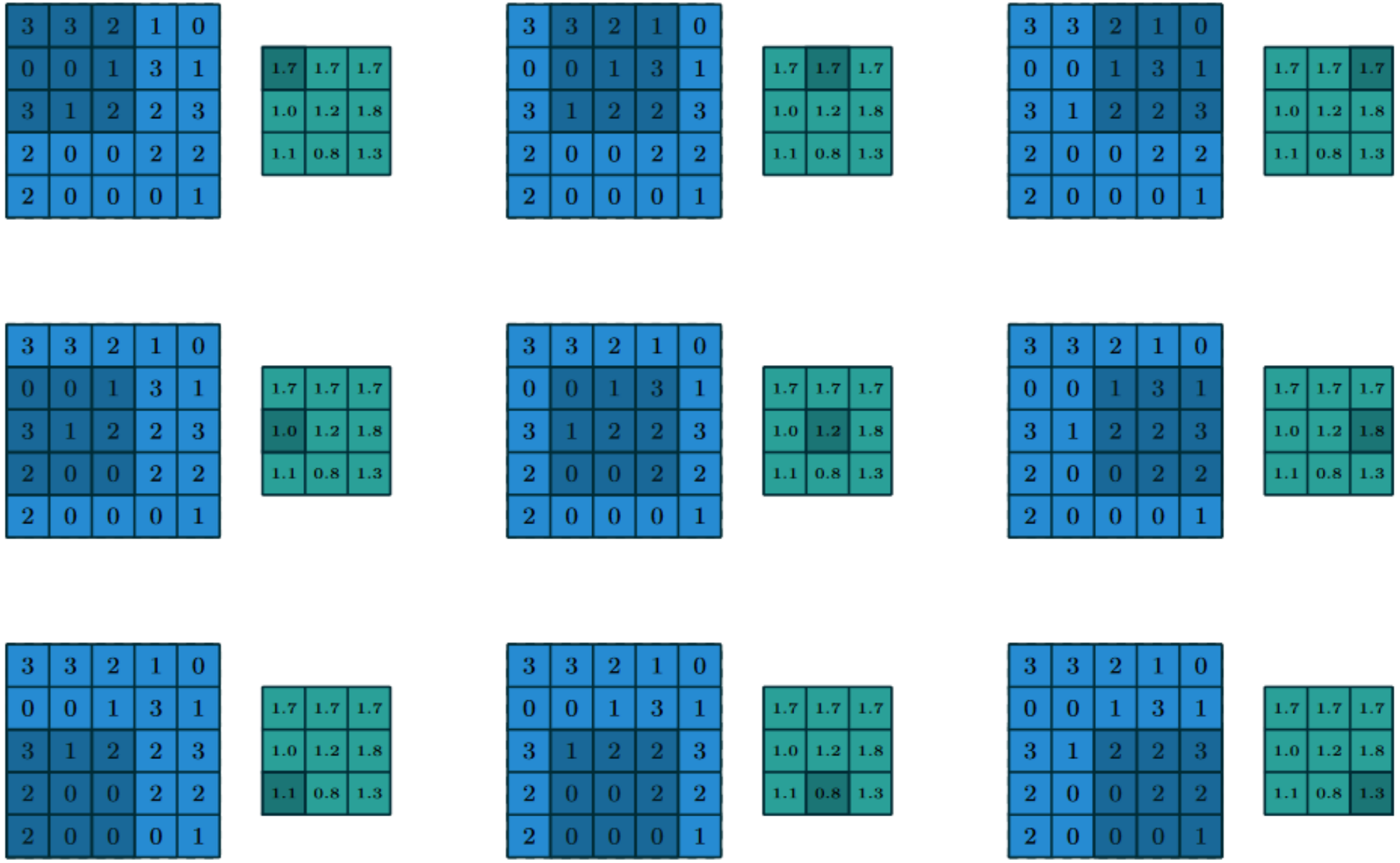
Pooling (агрегация, субдискретизация / subsampling)

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



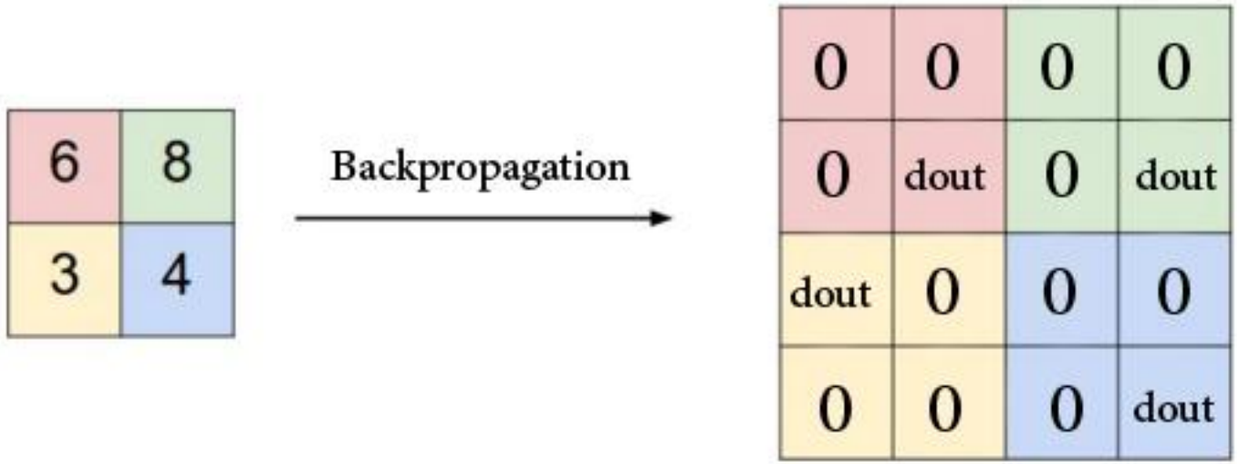
делается независимо по каналам ⇒ сохраняет число каналов (глубину тензора)

Агрегация (Pooling) усреднением



Агрегация (Pooling): дифференцирование

При дифференцировании возвращают градиент в позициях максимумов



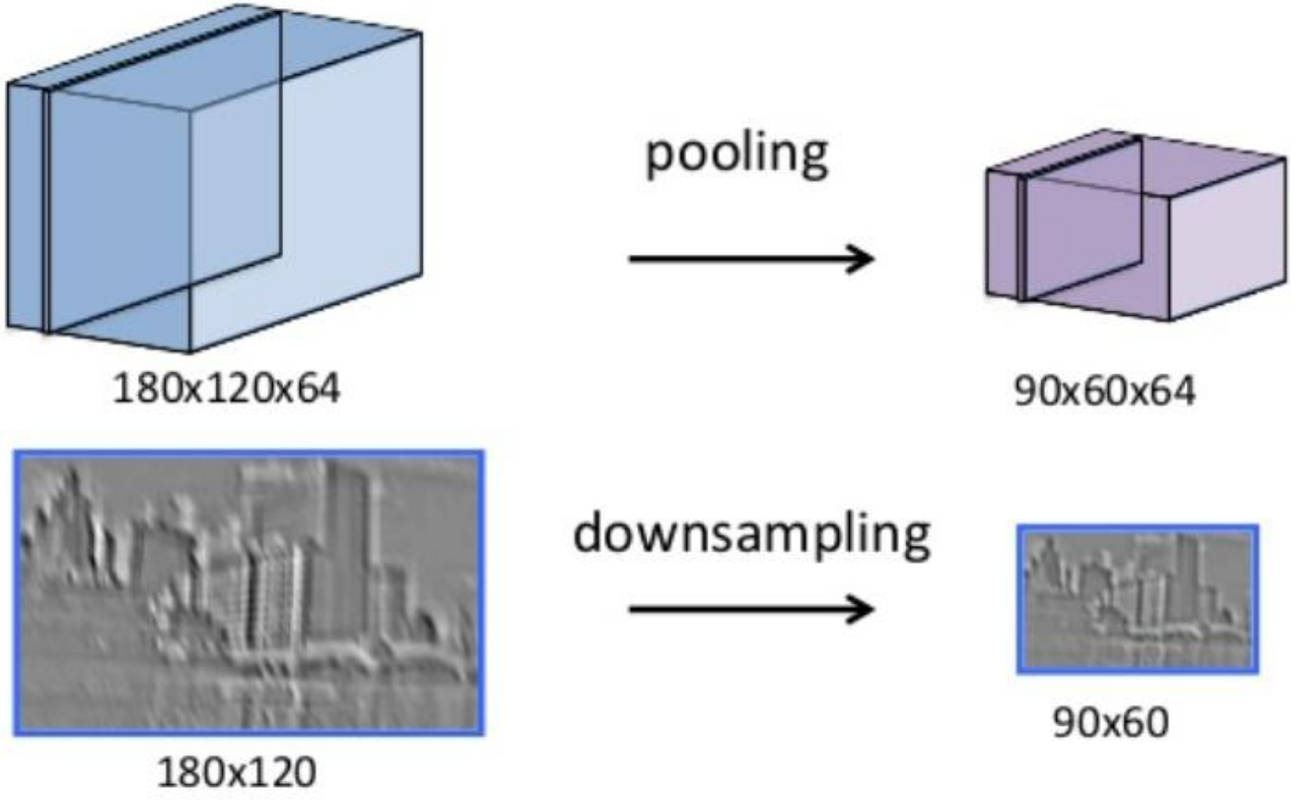
Почему… $$\frac{\partial \max[f(x,w), g(x,w)]}{\partial w} = \begin{cases} \frac{\partial f(x,w)}{\partial w}, & f(x,w) \geq g(x,w), \\ \frac{\partial g(x,w)}{\partial w}, & f(x,w) < g(x,w). \end{cases}$$

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) / локальные признаки**

нет связи нейронов «каждый с каждым»

У свёрточных НС мало весов!!!

- **разделение параметров (parameter sharing)**

одна свёртка используется «по всему изображению»

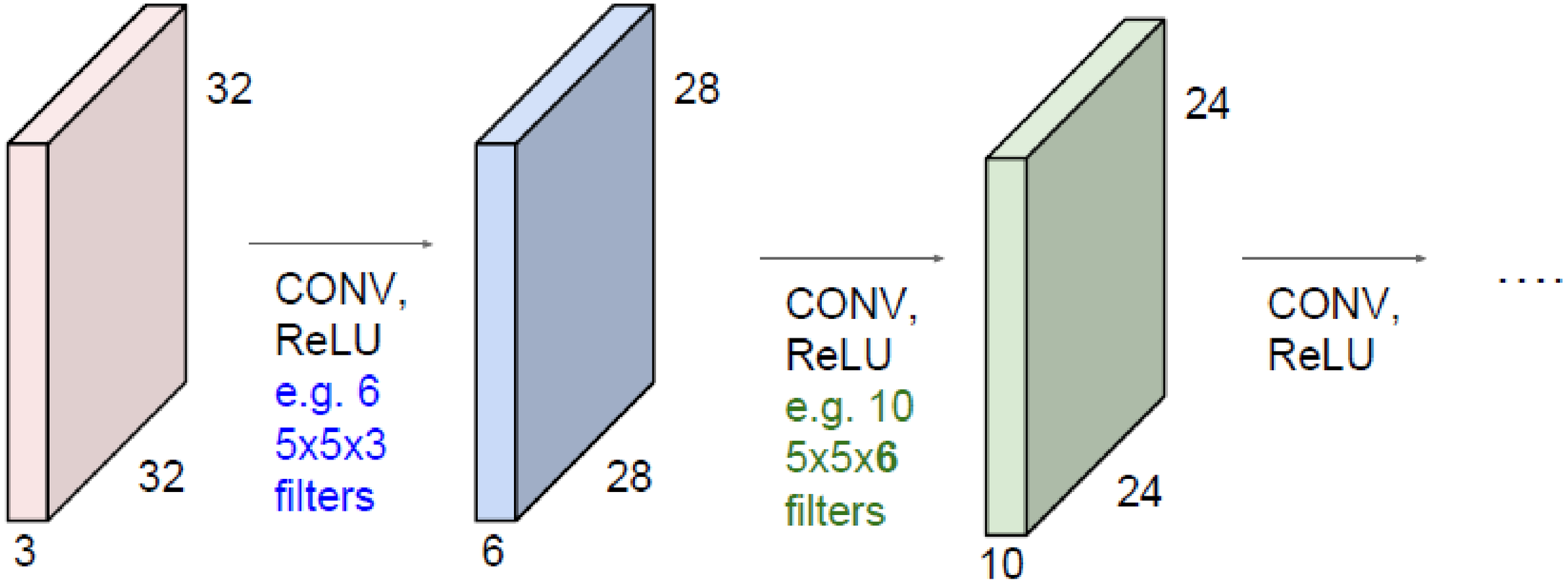
$\Rightarrow$ мало параметров

- **инвариантные преобразования (equivariant representations)**

инвариантность относительно сдвига

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

Свёрточный слой: тензор → тензор



Каждый тензор:

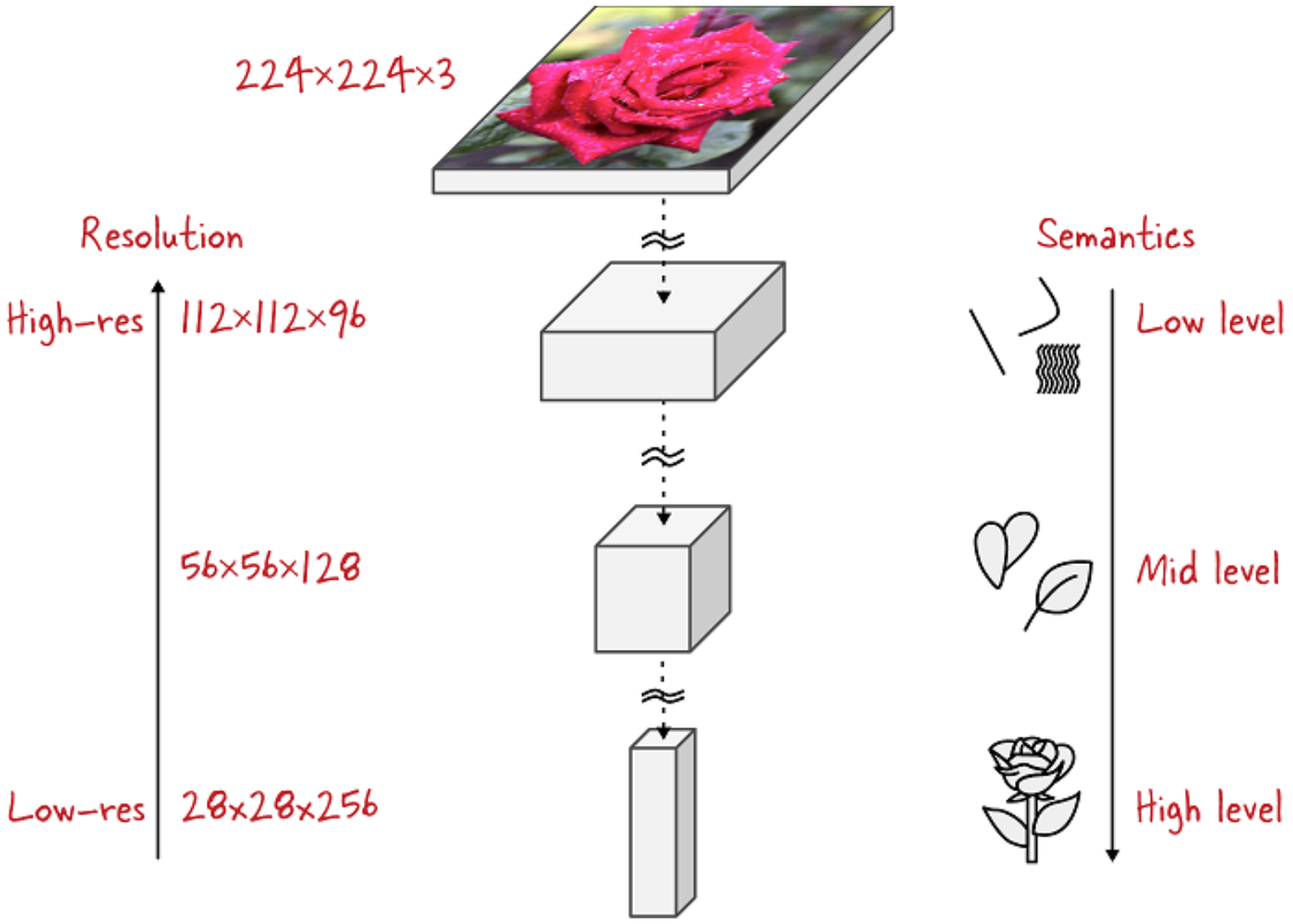
 # признаков / каналов (глубина) × высота × ширина

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])
```

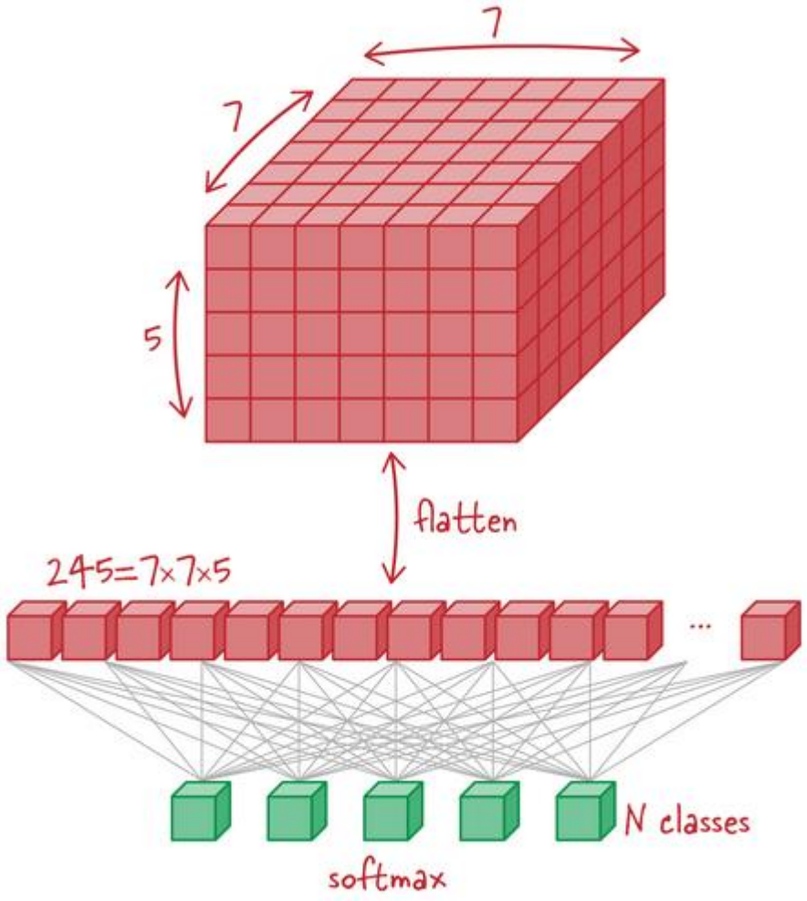
Визуализация признаков

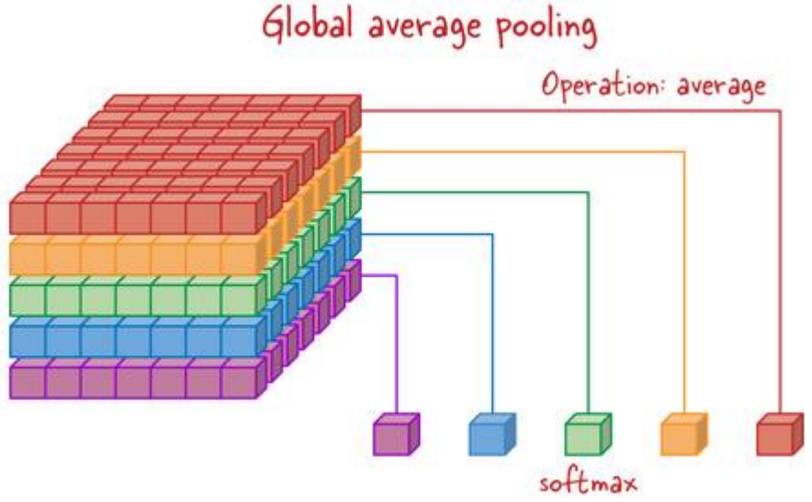


[Practical Machine Learning for Computer Vision]

Последние слои CNN – векторизация / глобальный пулинг

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

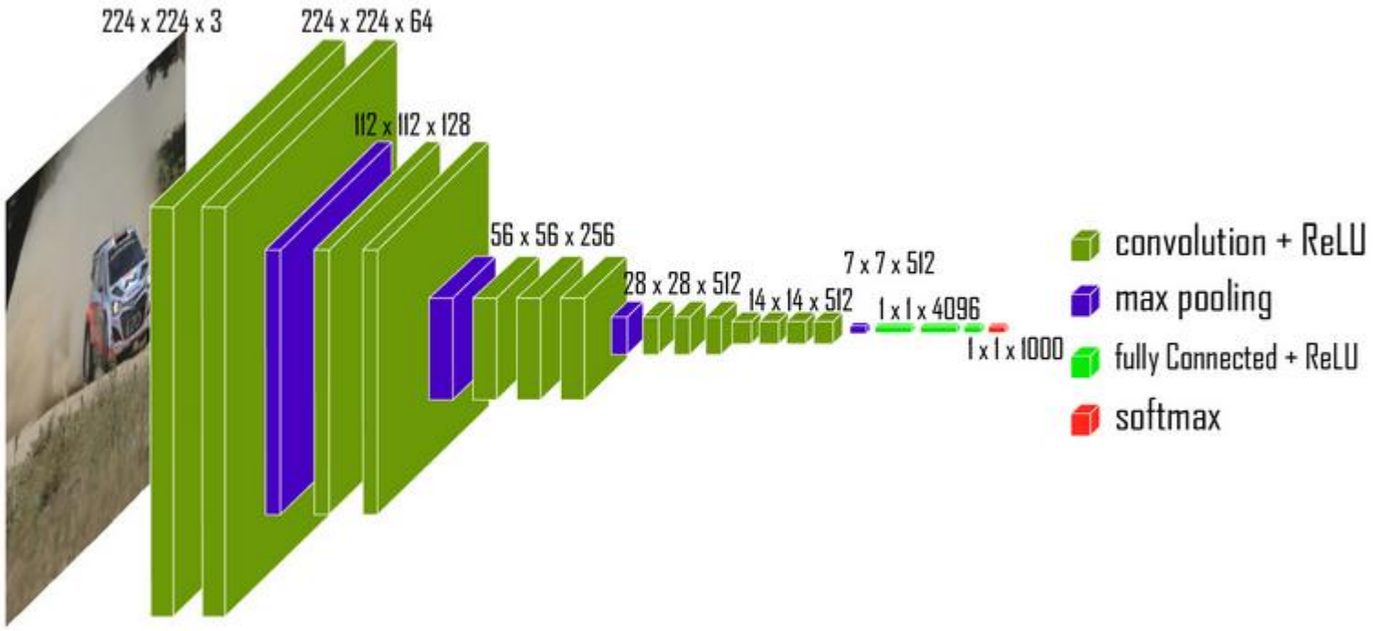


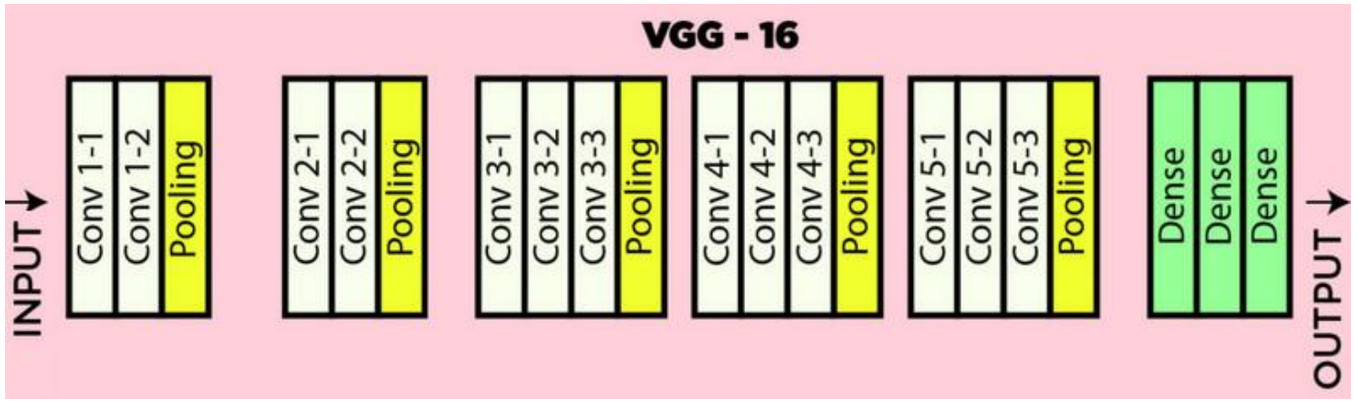


Совсем пропадает пространственная информация

[Practical Machine Learning for Computer Vision]

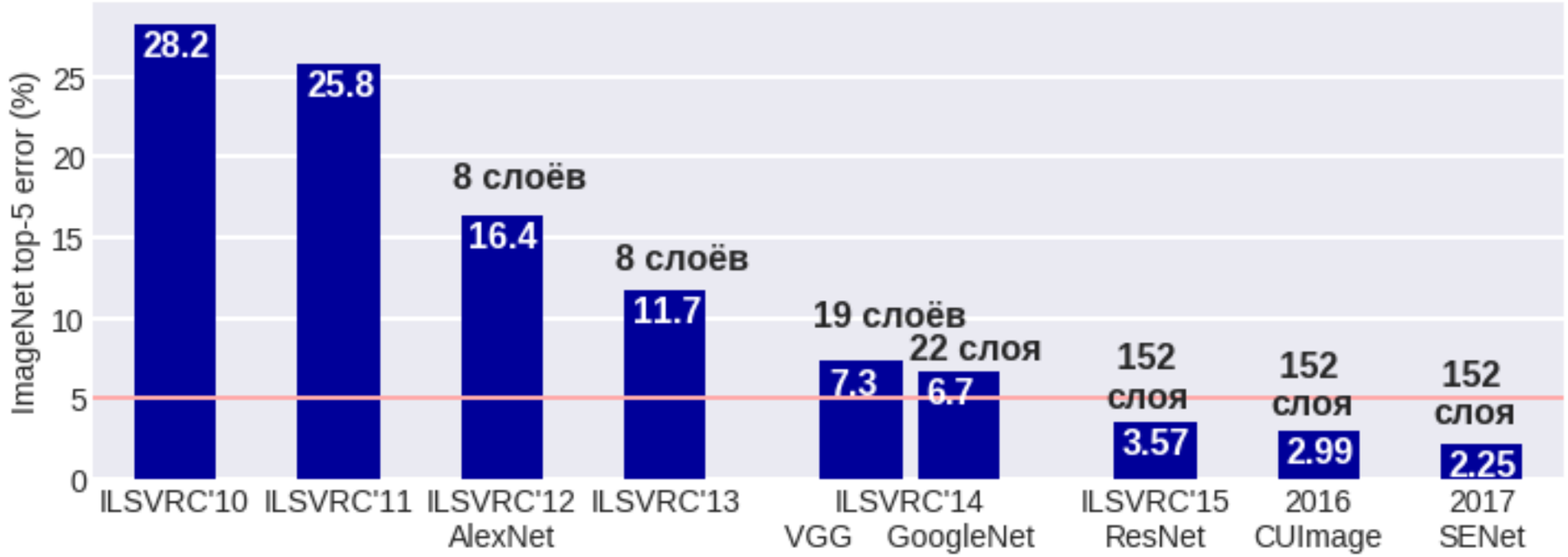
Архитектура CNN





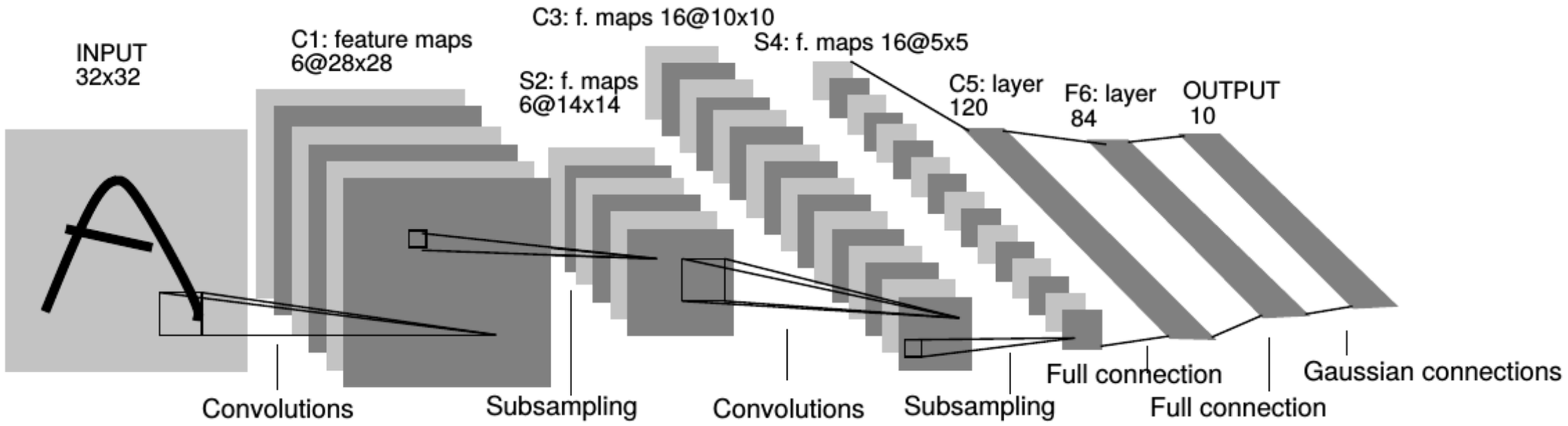
https://www.geeksforgeeks.org/vgg-16-cnn-model/

Революция в машинном обучении



ошибка человека – 5.1

LeNet-5 (1998)



5 = 2 свёрточных слоя + 3 полносвязных

свёртки 5×5

C@H×W

третий слой можно считать свёрточным, а можно полносвязным

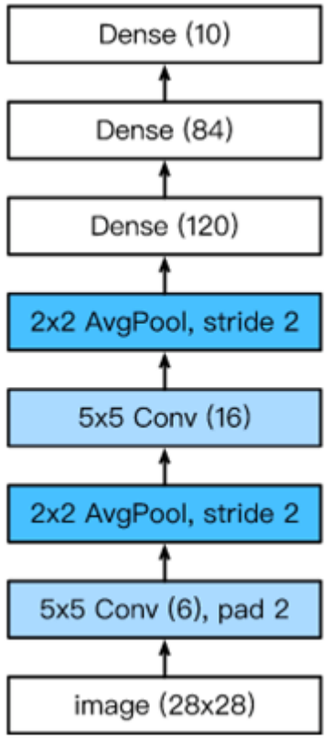
(там 5×5-свёртка)

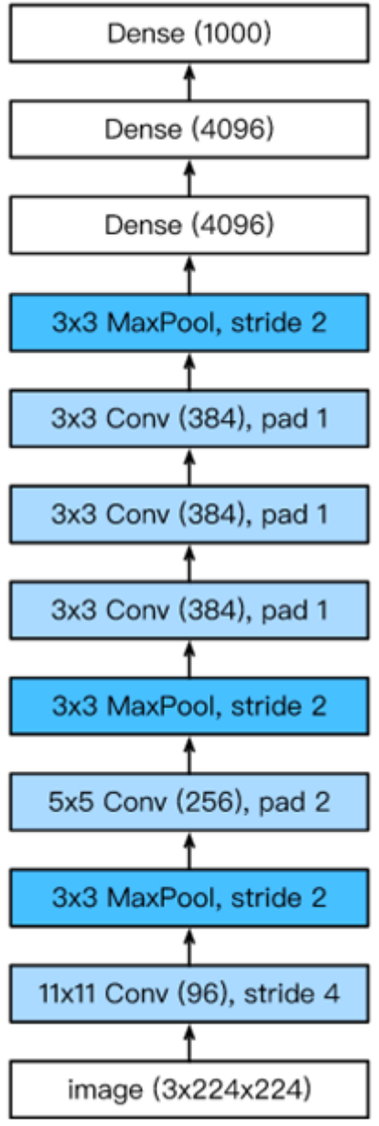
Минутка кода: модификация LeNet-5

```
class LeNet5(nn.Module):
    def __init__(self, n_classes):
        super(LeNet5, self).__init__()

        self.feature_extractor = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, stride=1),
            nn.Tanh(),
            nn.AvgPool2d(kernel_size=2),
            nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5, stride=1),
            nn.Tanh(),
            nn.AvgPool2d(kernel_size=2),
            nn.Conv2d(in_channels=16, out_channels=120, kernel_size=5, stride=1),
            nn.Tanh()
        )
        self.classifier = nn.Sequential(
            nn.Linear(in_features=120, out_features=84),
            nn.Tanh(),
            nn.Linear(in_features=84, out_features=n_classes),
        )
    def forward(self, x):
        x = self.feature_extractor(x)
        x = torch.flatten(x, 1)
        logits = self.classifier(x)
        probs = F.softmax(logits, dim=1)
        return logits, probs
```

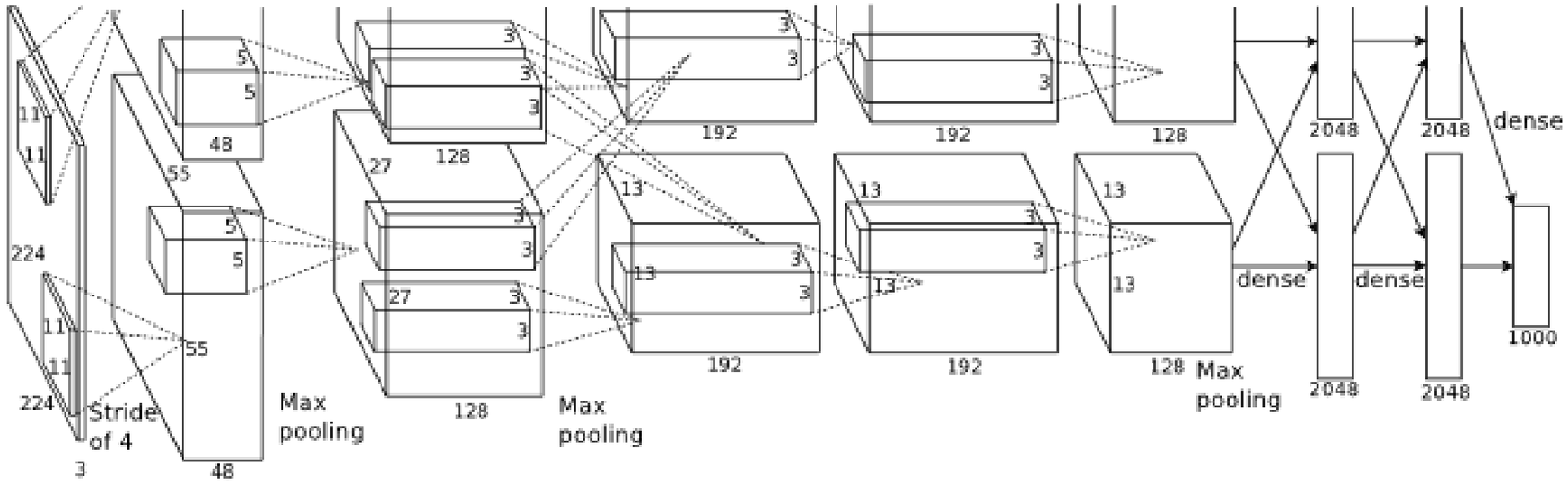
https://github.com/erykml/medium_articles/blob/master/Computer%20Vision/lenet5_pytorch.ipynb

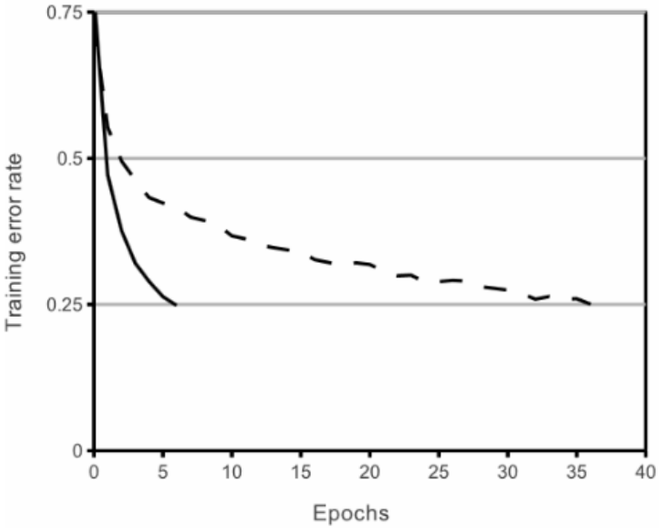
LeNet-5 → AlexNet





AlexNet (2012)



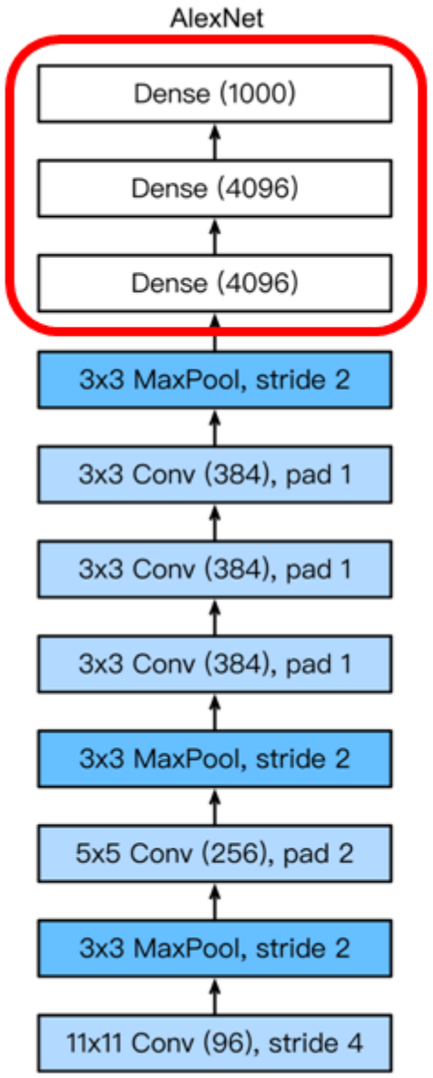


- **ReLU – после ∀ conv и dense слоя (см. рис, скорость 6×)**
  - **MaxPool (вместо AvgPool), полно-связные слои**
    - **Data augmentation**
  - **Dropout 0.5 (но и время обучения 2×)**
  - **Batch size = 128, SGD Momentum = 0.9**
    - **60M параметров / 650K нейронов**
    - **1 неделя на 2 GPU (50x над CPU)**
      - **7 скрытых слоёв**

Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons **(dashed line)**. The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton “ImageNet Classification with Deep Convolutional Neural Networks” // https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Параметры и сложность сетей

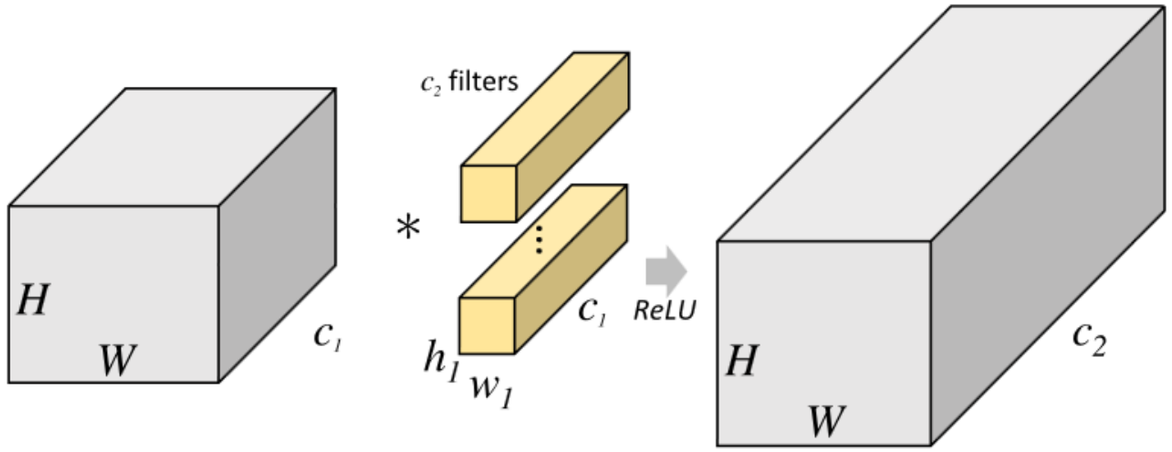


Layer Name	Tensor Size	Weights	Biases	Parameters
Input Image	227x227x3	0	0	0
Conv-1	55x55x96	34,848	96	34,944
MaxPool-1	27x27x96	0	0	0
Conv-2	27x27x256	614,400	256	614,656
MaxPool-2	13x13x256	0	0	0
Conv-3	13x13x384	884,736	384	885,120
Conv-4	13x13x384	1,327,104	384	1,327,488
Conv-5	13x13x256	884,736	256	884,992
MaxPool-3	6x6x256	0	0	0
FC-1	4096×1	37,748,736	4,096	37,752,832
FC-2	4096×1	16,777,216	4,096	16,781,312
FC-3	1000×1	4,096,000	1,000	4,097,000
Output	1000×1	0	0	0
Total				**62,378,344**

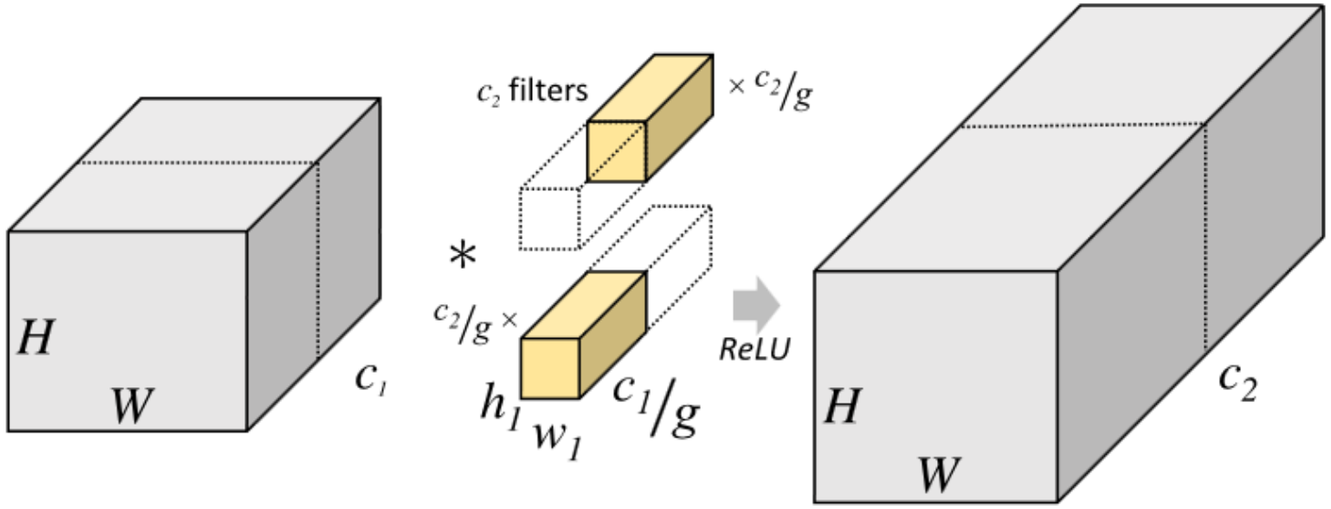
https://github.com/aws-samples/aws-machine-learning-university-accelerated-cv/tree/master/slides

Какие бывают свёртки: Group Convolutions

обычная ситуация



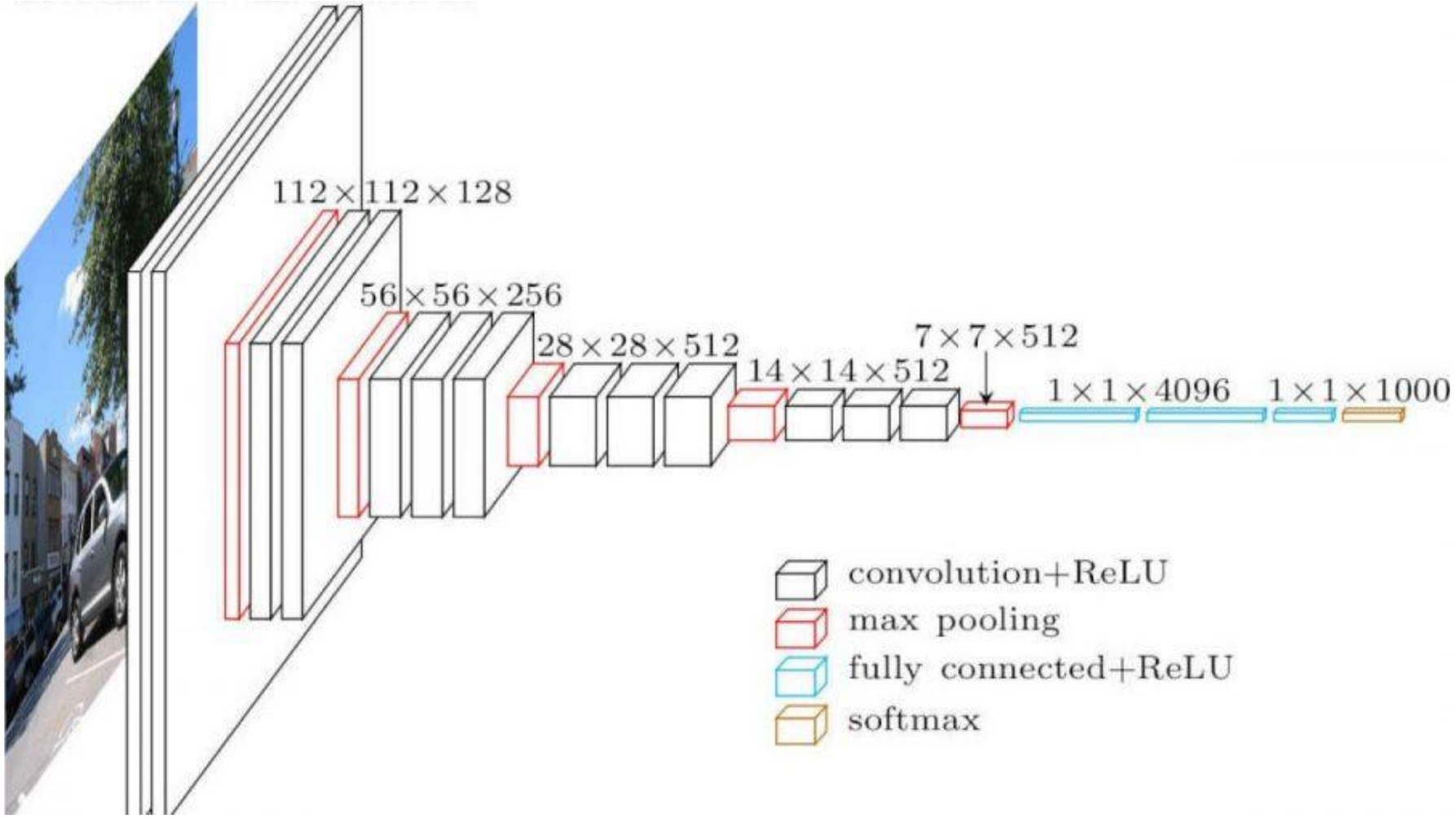
с разбиением на две группы



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/

VGG (2014, VGG group, Oxford)



K. Simonyan, A. Zisserman «Very Deep Convolutional Networks for Large-Scale Image Recognition» https://arxiv.org/pdf/1409.1556.pdf

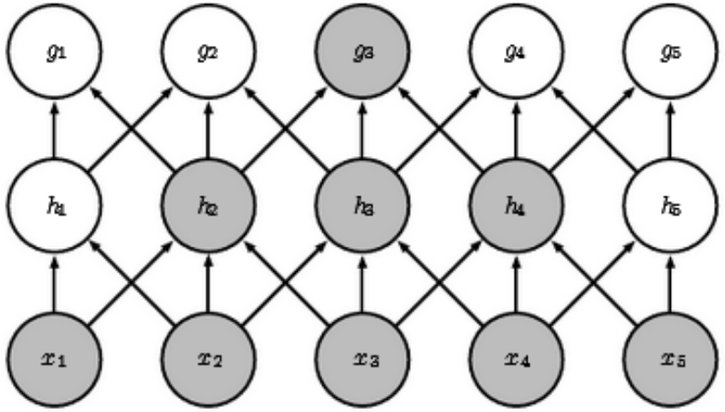
VGG (2014)

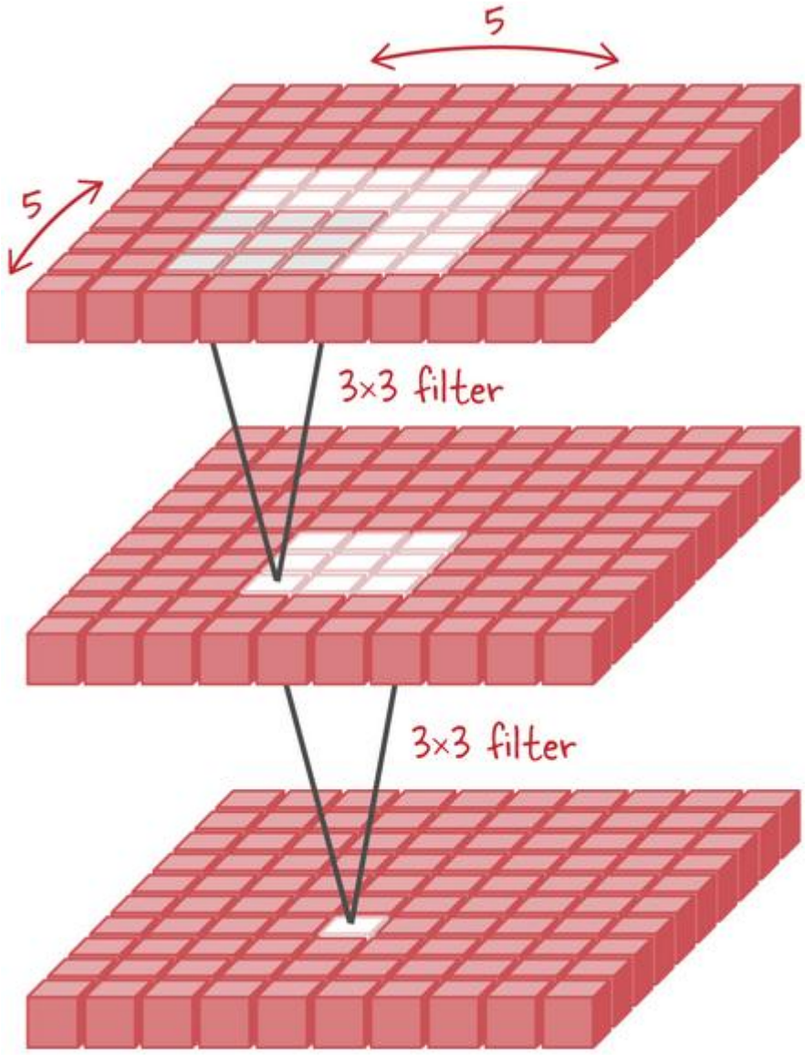
- 5 Convolutional blocks + 3 Fully Connected Layers
- вход = 256×256 (здесь 224×224)
- ReLu
- каскад 3×3 свёрток (замена 7×7)
- 138M (133-144) параметров (очень тяжеловесная)
- 3 недели 4 GPU, тоже использовали ансамбль
- большинство вычислений – в первых свёртках, большинство параметров – в конце
- batch size = 256
- momentum = 0.9
- L2-reg ~ $5 \cdot 10^{-4}$
- dropout = 0.5 (2 первых полносвязных слоя)
- LR = 10^{-2} (/ 10 после стабилизации)
- 370K итерация (74 эпох)
- более глубокие сети инициализировались по обученным простым (A-D)

можно просто «хорошую инициализацию»

C++ Caffe

Идея каскада свёрток





«Receptive field»

- **меньше параметров**
- **быстрее**
- **дополнительные нелинейности**

[Practical Machine Learning for Computer Vision]

Минутка кода: VGG

```
import torch
model = torch.hub.load('pytorch/vision:v0.6.0', 'vgg11', pretrained=True)
# or any of these variants
# model = torch.hub.load('pytorch/vision:v0.6.0', 'vgg11_bn', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.6.0', 'vgg13', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.6.0', 'vgg13_bn', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.6.0', 'vgg16', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.6.0', 'vgg16_bn', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.6.0', 'vgg19', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.6.0', 'vgg19_bn', pretrained=True)
model.eval()
```

есть готовые модели:

https://pytorch.org/docs/stable/torchvision/models.html

Минутка кода: VGG

```
class VGG(nn.Module):

    def __init__(self, features, num_classes=1000):
        super(VGG, self).__init__()
        self.features = features
        self.classifier = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, num_classes),
        )
        self._initialize_weights()
```

```
    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x
```

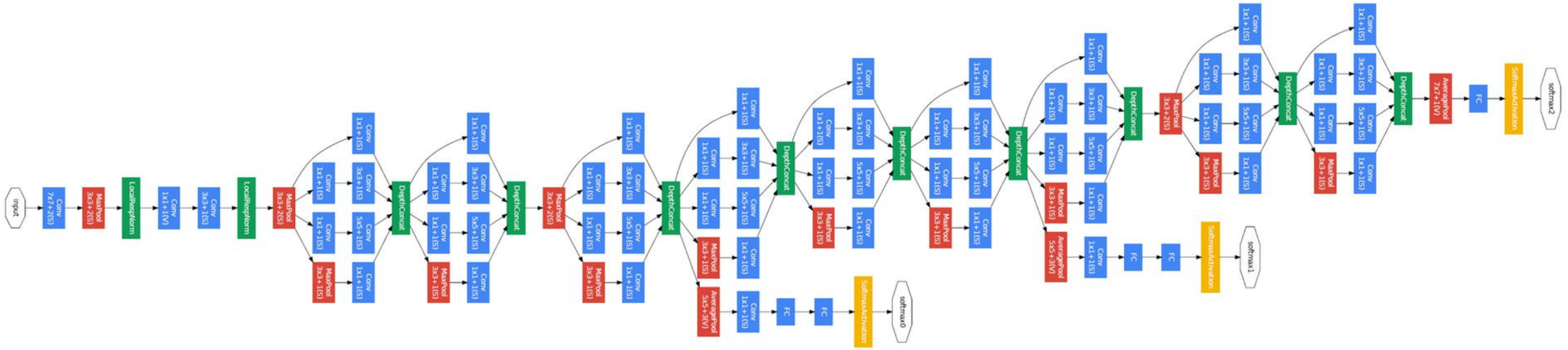
```
cfg = {
    'A': [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
    'B': [64, 64, 'M', 128, 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
    # ... }
```

```
def make_layers(cfg, batch_norm=False):
    layers = []
    in_channels = 3
    for v in cfg:
        if v == 'M':
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        else:
            conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1)
            if batch_norm:
                layers += [conv2d, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]
            else:
                layers += [conv2d, nn.ReLU(inplace=True)]
            in_channels = v
    return nn.Sequential(*layers)

def vgg11(pretrained=False, **kwargs):
    """VGG 11-layer model (configuration "A")
    Args:          pretrained (bool): If True, returns a model pre-trained on ImageNet
    """
    model = VGG(make_layers(cfg['A']), **kwargs)
    if pretrained:
        model.load_state_dict(model_zoo.load_url(model_urls['vgg11']))
    return model
```

Минутка кода: VGG

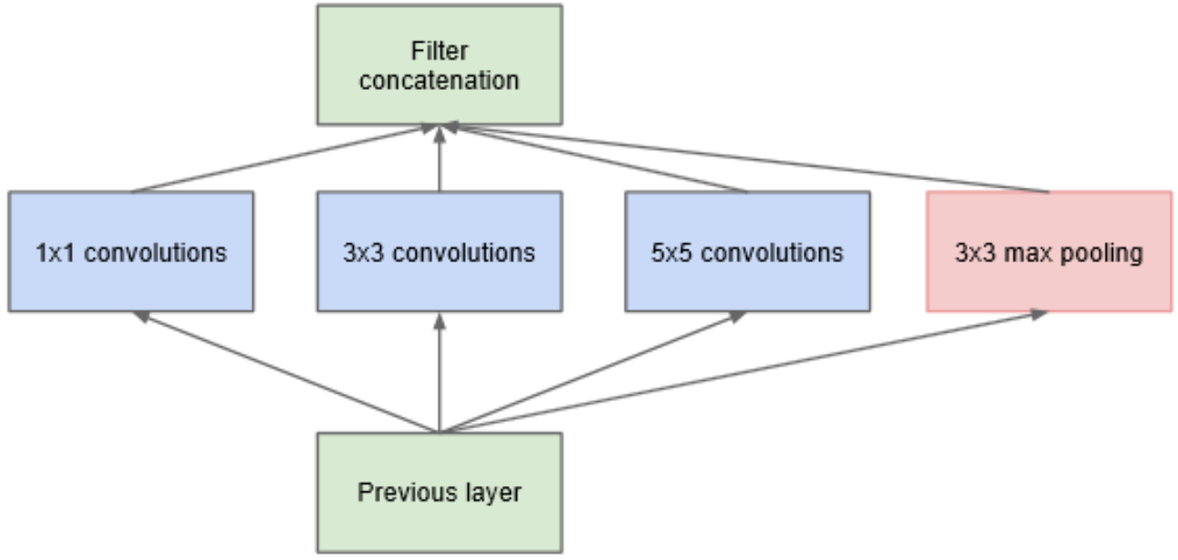
```
def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                m.weight.data.normal_(0, math.sqrt(2. / n))
                if m.bias is not None:
                    m.bias.data.zero_()
            elif isinstance(m, nn.BatchNorm2d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()
            elif isinstance(m, nn.Linear):
                m.weight.data.normal_(0, 0.01)
                m.bias.data.zero_()
```

GoogLeNet / Inception (2014)

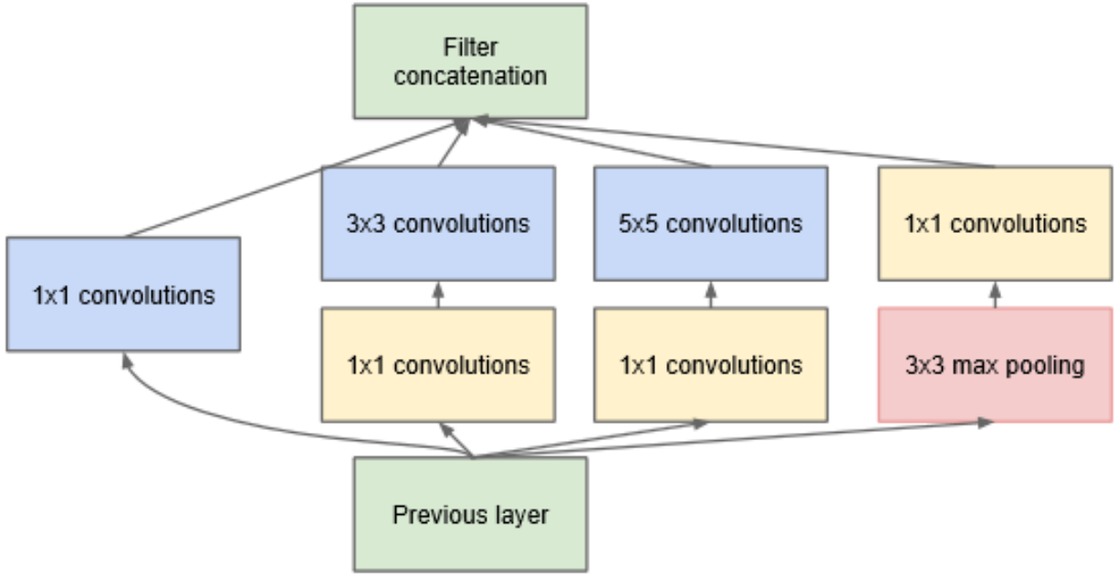


- **«конструктор» НС – «Modular Architecture»**
- **22 слоя – нет полносвязных**
- **Модуль «Inception», 1×1-свёртки,**
- **5M параметров (меньше!)**
- **дополнительные выходы классификации** (с весом 0.3 к общей ошибке)
- **тоже ансамбль (из 7)**
- **Global Average Pooling** (немного улучшает качество)

Модуль «Inception»



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

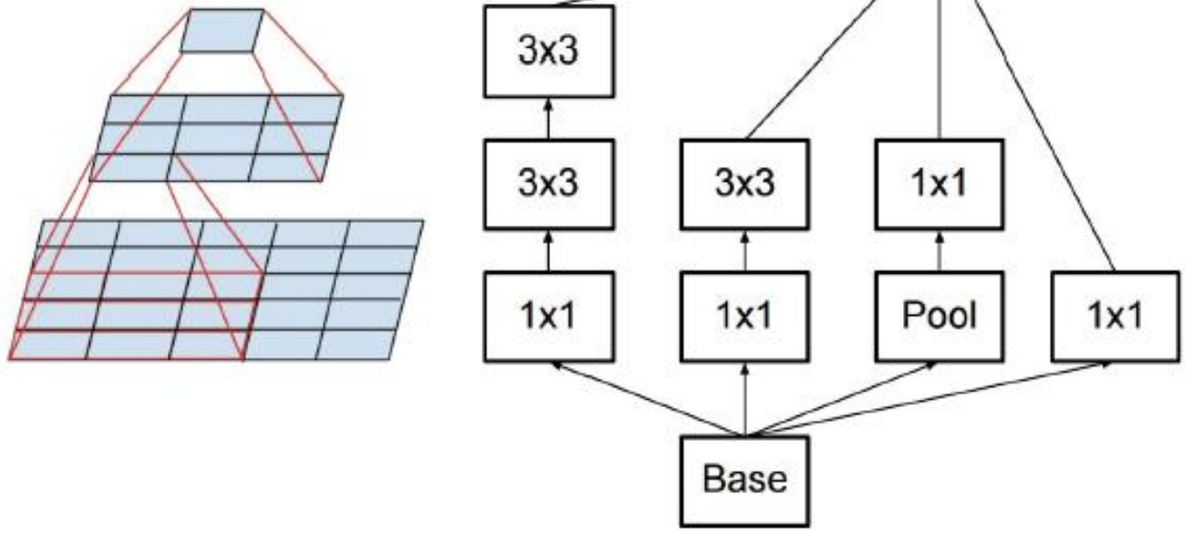
Изначальная идея – разные свёртки + пулинг

1×1-свёртки существенно уменьшают число параметров!

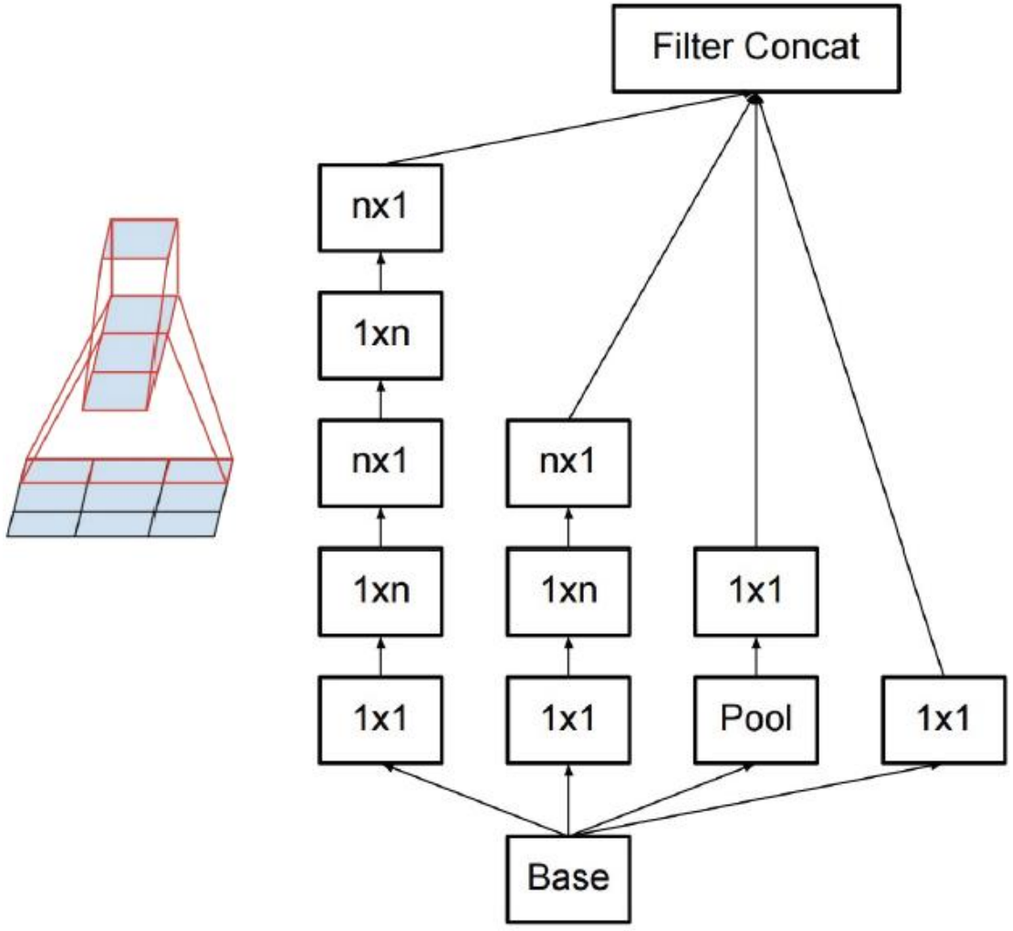
Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich «Going Deeper with Convolutions» // https://arxiv.org/abs/1409.4842

Inception v2, v3

Другое строение модулей (+batchnorm)



5×5 свёртку превратить в две 3×3



дальнейшая факторизация

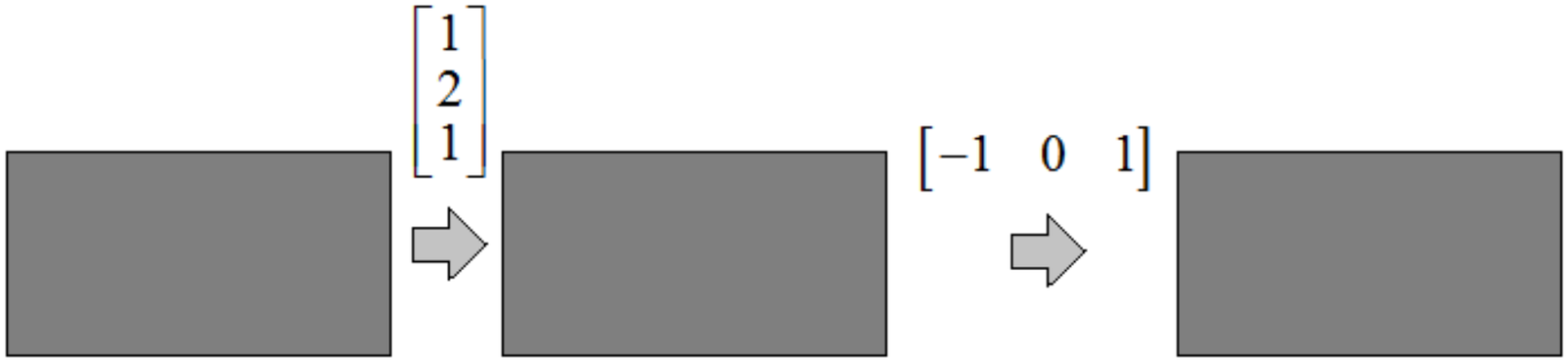
Какие бывают свёртки: Spatial Separable Convolutions

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

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: модуль Inception

```
class Inception_base(nn.Module):
    def __init__(self, depth_dim, input_size, config):
        super(Inception_base, self).__init__()
        self.depth_dim = depth_dim
        # 1x1
        self.conv1 = nn.Conv2d(input_size, out_channels=config[0][0], kernel_size=1, stride=1, padding=0)
        # 3x3_bottleneck + 3x3
        self.conv3_1 = nn.Conv2d(input_size, out_channels=config[1][0], kernel_size=1, stride=1, padding=0)
        self.conv3_3 = nn.Conv2d(config[1][0], config[1][1], kernel_size=3, stride=1, padding=1)
        # 5x5_bottleneck + 5x5
        self.conv5_1 = nn.Conv2d(input_size, out_channels=config[2][0], kernel_size=1, stride=1, padding=0)
        self.conv5_5 = nn.Conv2d(config[2][0], config[2][1], kernel_size=5, stride=1, padding=2)
        # maxpool + 1x1
        self.max_pool_1 = nn.MaxPool2d(kernel_size=config[3][0], stride=1, padding=1)
        self.conv_max_1 = nn.Conv2d(input_size, out_channels=config[3][1], kernel_size=1, stride=1,
                                    padding=0)
        self.apply(layer_init)

    def forward(self, input):
        output1 = F.relu(self.conv1(input))
        output2 = F.relu(self.conv3_1(input))
        output2 = F.relu(self.conv3_3(output2))
        output3 = F.relu(self.conv5_1(input))
        output3 = F.relu(self.conv5_5(output3))
        output4 = F.relu(self.conv_max_1(self.max_pool_1(input)))
        return torch.cat([output1, output2, output3, output4], dim=self.depth_dim)
```

https://github.com/antspy/inception_v1.pytorch

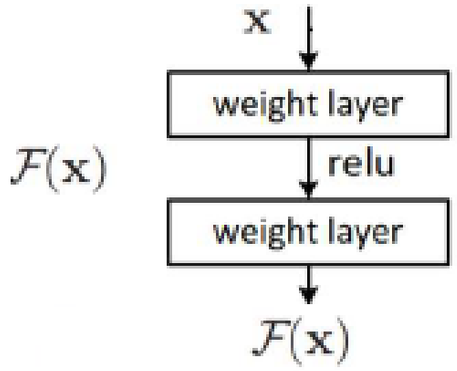
```
class Inception_v1(nn.Module):
    def __init__(self, num_classes=1000):
        super(Inception_v1, self).__init__()
        #conv2d0
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3)
        self.max_pool1 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.lrn1 = LRN(local_size=11, alpha=0.00109999999404, beta=0.5, k=2)
        # Local Response Normalization layer - пропустили...
        #conv2d1
        self.conv2 = nn.Conv2d(64, 64, kernel_size=1, stride=1, padding=0)
        #conv2d2
        self.conv3  = nn.Conv2d(64, 192, kernel_size=3, stride=1, padding=1)
        self.lrn3 = LRN(local_size=11, alpha=0.00109999999404, beta=0.5, k=2)
        self.max_pool3 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.inception_3a = Inception_base(1, 192, [[64], [96,128], [16, 32], [3, 32]]) #3a
        self.inception_3b = Inception_base(1, 256, [[128], [128,192], [32, 96], [3, 64]]) #3b
        self.max_pool_inc3= nn.MaxPool2d(kernel_size=3, stride=2, padding=0)
        self.inception_4a = Inception_base(1, 480, [[192], [ 96,204], [16, 48], [3, 64]]) #4a
        self.inception_4b = Inception_base(1, 508, [[160], [112,224], [24, 64], [3, 64]]) #4b
        self.inception_4c = Inception_base(1, 512, [[128], [128,256], [24, 64], [3, 64]]) #4c
        self.inception_4d = Inception_base(1, 512, [[112], [144,288], [32, 64], [3, 64]]) #4d
        self.inception_4e = Inception_base(1, 528, [[256], [160,320], [32,128], [3,128]]) #4e
        self.max_pool_inc4 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.inception_5a = Inception_base(1, 832, [[256], [160,320], [48,128], [3,128]]) #5a
        self.inception_5b = Inception_base(1, 832, [[384], [192,384], [48,128], [3,128]]) #5b
        self.avg_pool5 = nn.AvgPool2d(kernel_size=7, stride=1, padding=0)
        self.dropout_layer = nn.Dropout(0.4)
        self.fc = nn.Linear(1024, num_classes)
        self.apply(layer_init)
```

```
    def forward(self, input):
        output = self.max_pool1(F.relu(self.conv1(input)))
        output = self.lrn1(output)

        output = F.relu(self.conv2(output))
        output = F.relu(self.conv3(output))
        output = self.max_pool3(self.lrn3(output))

        output = self.inception_3a(output)
        output = self.inception_3b(output)
        output = self.max_pool_inc3(output)

        output = self.inception_4a(output)
        output = self.inception_4b(output)
        output = self.inception_4c(output)
        output = self.inception_4d(output)
        output = self.inception_4e(output)
        output = self.max_pool_inc4(output)

        output = self.inception_5a(output)
        output = self.inception_5b(output)
        output = self.avg_pool5(output)

        output = output.view(-1, 1024)
        if self.fc is not None:
            output = self.dropout_layer(output)
            output = self.fc(output)
    return output
```

ResNet = Residual Network (2015)

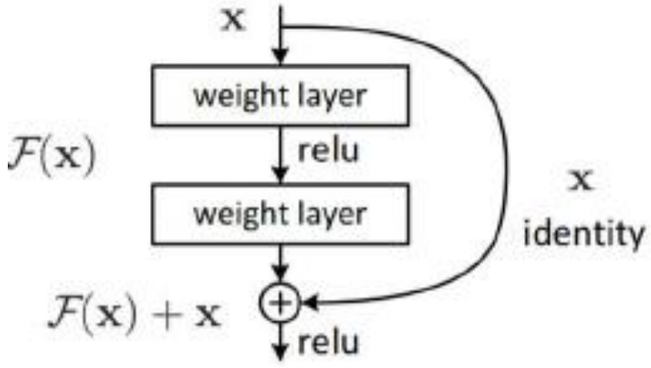
$$y = f(x)$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} f'(x)$$



$$y = f(x) + x$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} f'(x) + \frac{\partial L}{\partial y}$$



skip (shortcut) connections

упрощение реализации тождественной функции,
по крайней мере, через два слоя

Просто добавление слоёв не помогает!

Добавлять надо по-умному…

He et al. «Deep Residual Learning for Image Recognition» https://arxiv.org/pdf/1512.03385.pdf

ResNet = Residual Network (2015)

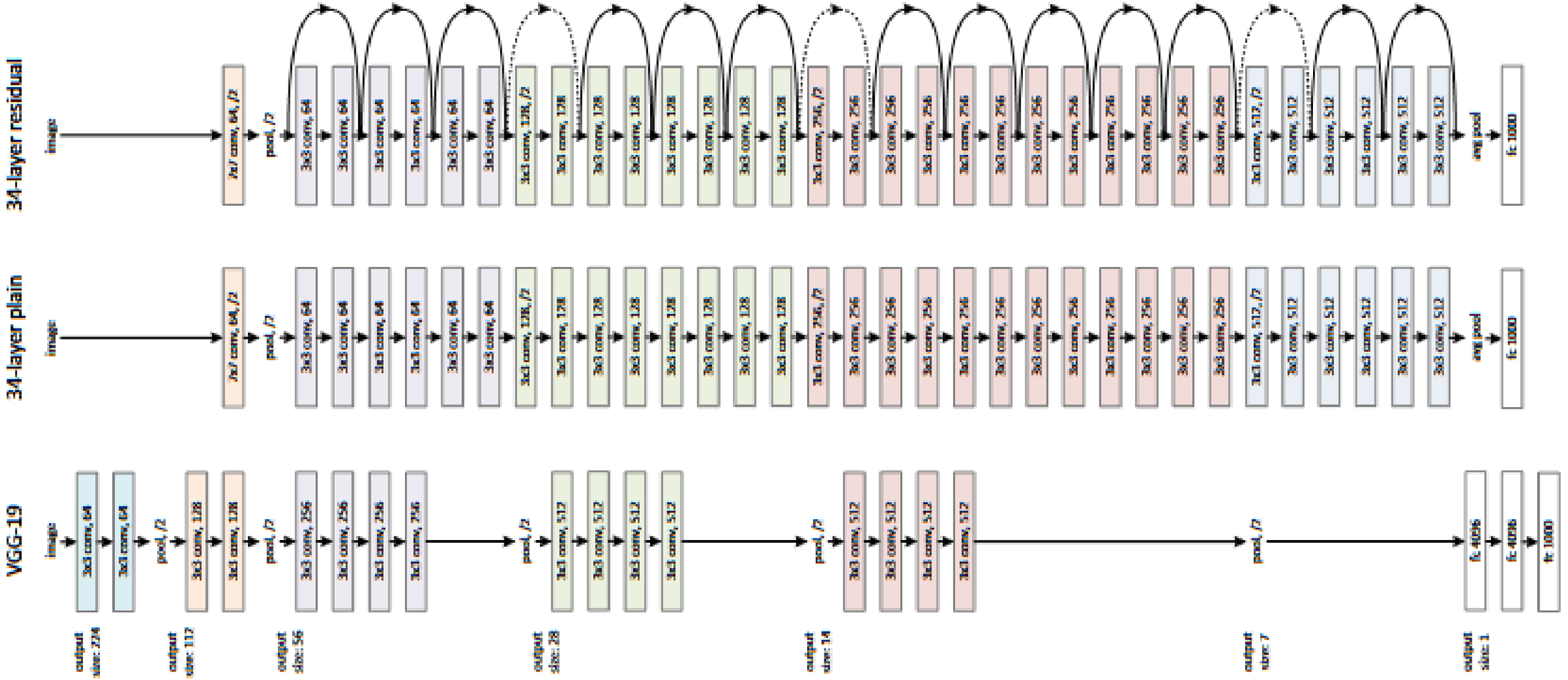


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

ResNet = Residual Network (2015)

- 152 слоя
- связи проходят через слои
- Batch Normalization после свёрток перед активациями (впервые)
- Умные инициализации весов
- SGD + Momentum (0.9)
- Mini-batch size = 256
- Нет Dropout!
- Average Global Pooling вместо FC-слоёв

ResNet = Residual Network (2015)

Deeper residual module (bottleneck)

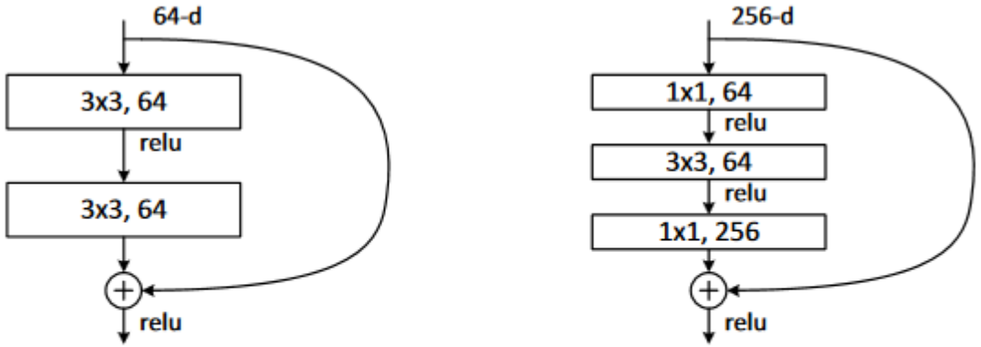
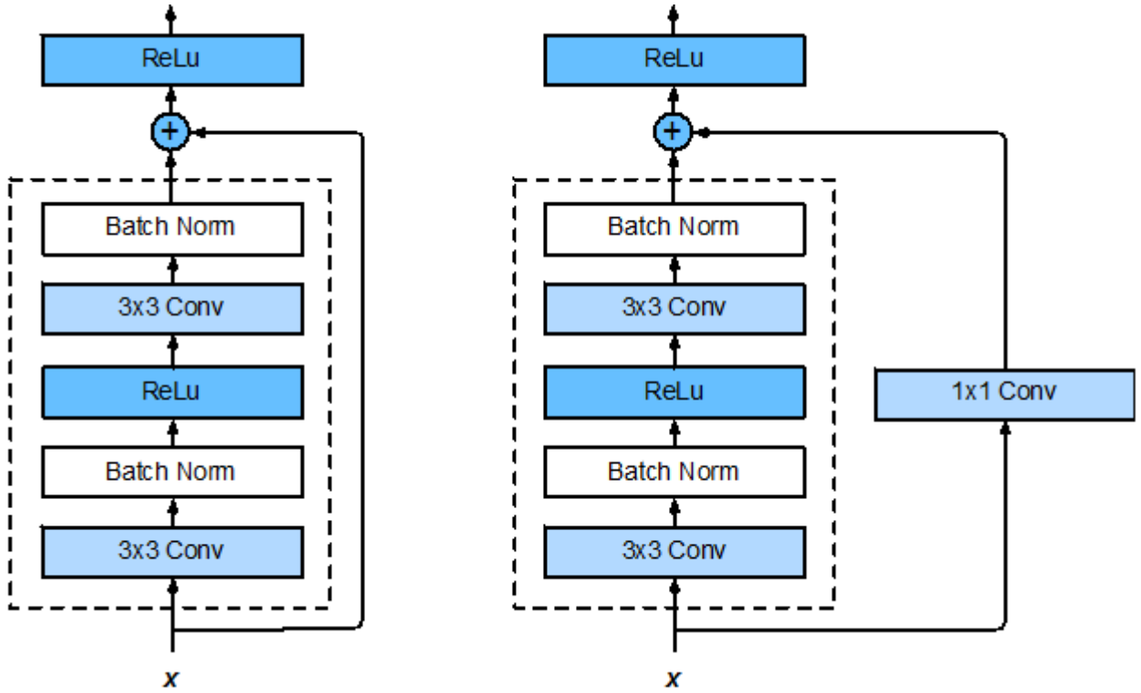


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

layer name	output size	152-layer
conv1	112×112	7×7, 64, stride 2
conv2_x	56×56	3×3 max pool, stride 2 $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax
FLOPs		11.3×10^9

Стандартное прокидывание и прокидывание с понижением размеров



не / использование свёртки при прокидывании, в коде – свёртка + BN:

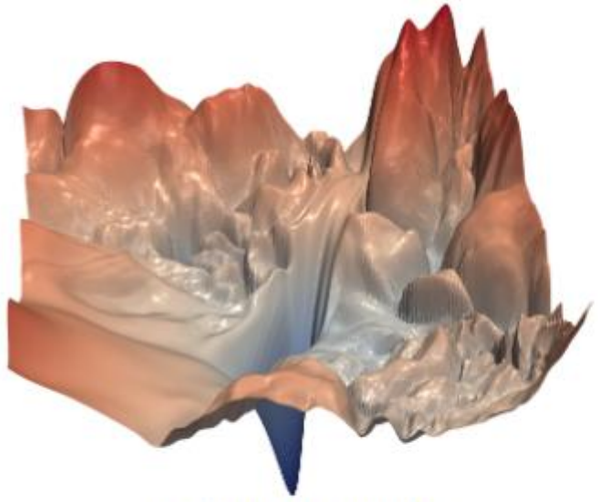
```
if stride != 1 or self.inplanes != planes * block.expansion:
    downsample = nn.Sequential(nn.Conv2d(self.inplanes, planes * block.expansion,
                                         kernel_size=1, stride=stride, bias=False),
                               nn.BatchNorm2d(planes * block.expansion))
```

Минутка кода: ResNet (остальное смотреть по ссылке!)

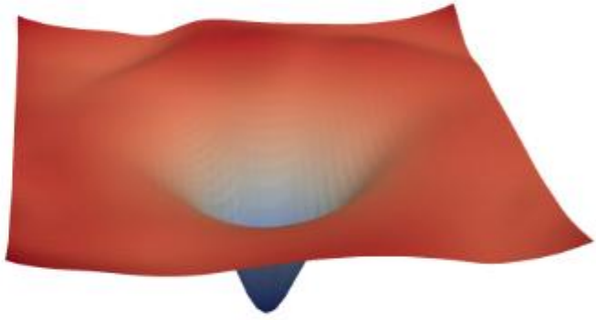
```
class BasicBlock(nn.Module): # building block ResNet 34 - не bottleneck
    expansion = 1
    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(BasicBlock, self).__init__()
        self.conv1 = conv3x3(inplanes, planes, stride)
        self.bn1 = nn.BatchNorm2d(planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(planes, planes)
        self.bn2 = nn.BatchNorm2d(planes)
        self.downsample = downsample
        self.stride = stride
    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        if self.downsample is not None:
            residual = self.downsample(x)
        out += residual
        out = self.relu(out)
        return out
```

https://chsasank.github.io/vision/_modules/torchvision/models/resnet.html

Эффект прокидывания связей

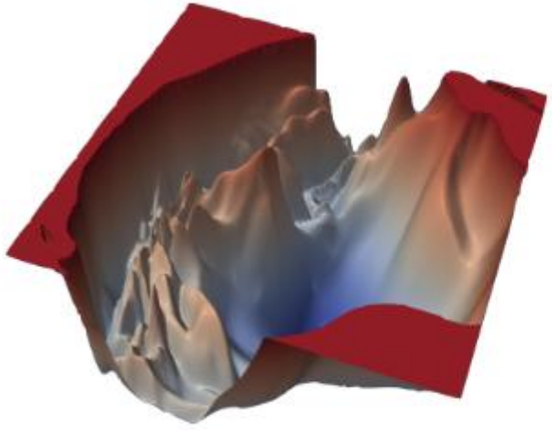


(a) without skip connections

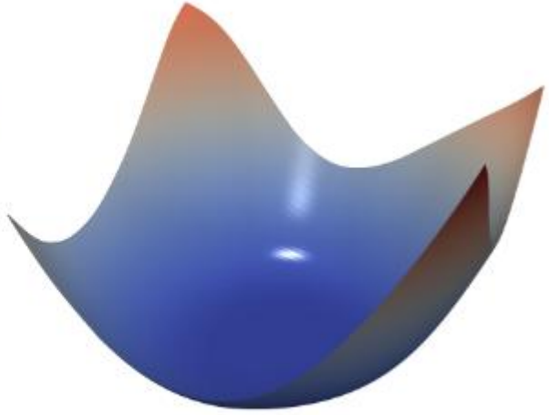


(b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.



(a) ResNet-110, no skip connections



(b) DenseNet, 121 layers

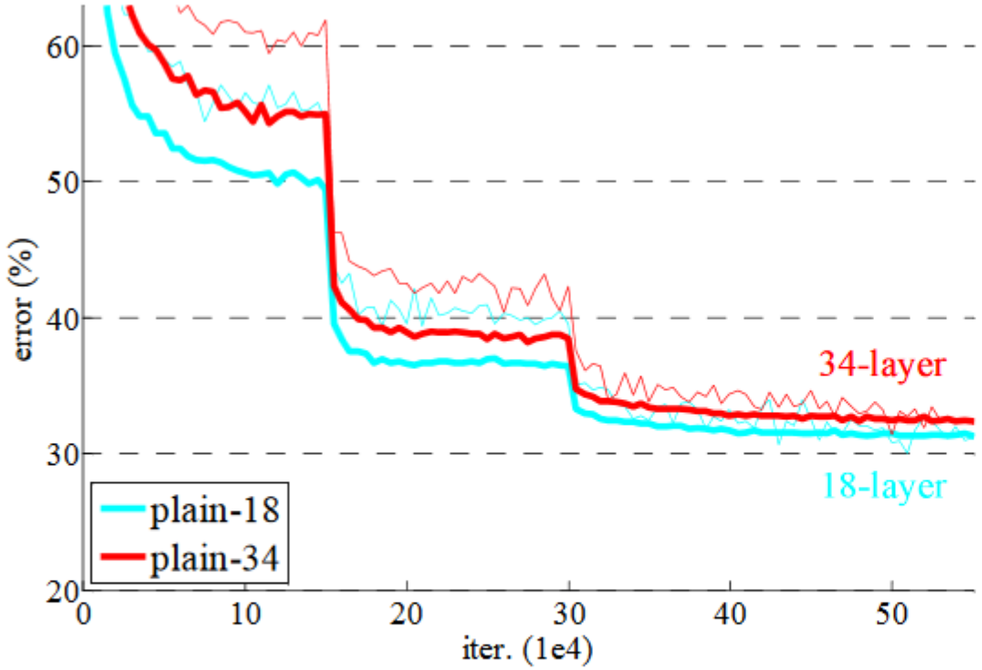
Figure 4: The loss surfaces of ResNet-110-noshort and DenseNet for CIFAR-10.

глубина и ширина «улучшают» поверхность функции ошибки

правильная оптимизация позволяет «правильно» идти по поверхности

«Visualizing the Loss Landscape of Neural Nets» https://arxiv.org/abs/1712.09913

ResNet: почему работает



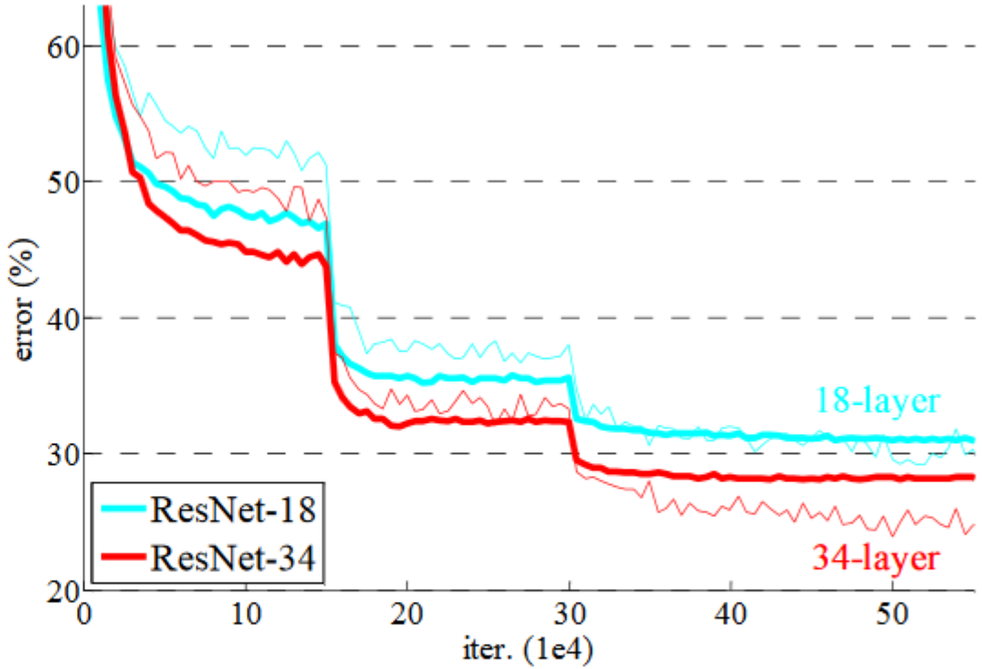


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

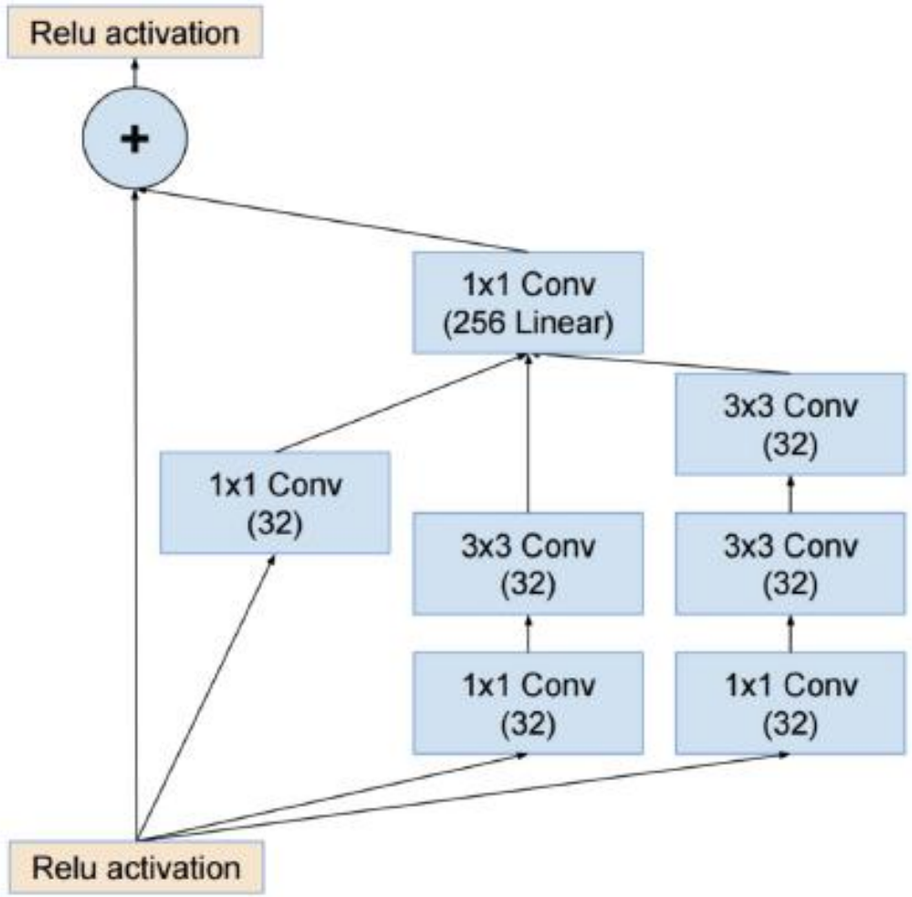
Сделали прокидывание связей – и ситуация изменилась
(глубокие сети лучше)

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

Inception-v4 (Inception-Resnet)

совмещение двух (даже больше) идей:

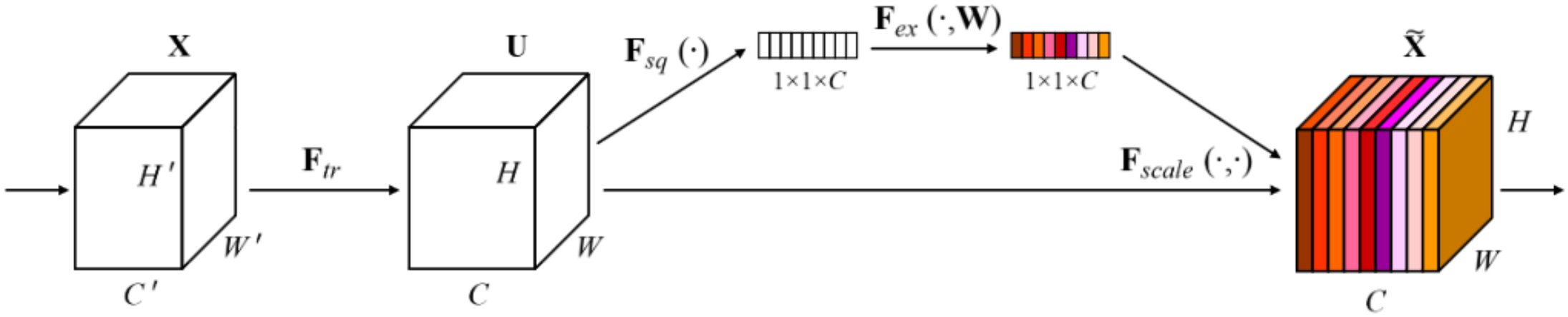


Szegedy, Christian, et al. "Inception-v4, inception-resnet and the impact of residual connections on learning." arXiv preprint arXiv:1602.07261 (2016).

SENet (Squeeze-and-Excitation Network, 2017)

обычно в CNN трансформация $F_{\text{tr}} : X_{H' \times W' \times C'} \to U_{H \times W \times C}$ **(например, свёртка)**

теперь добавим «Squeeze-and-Excitation» (SE) block $F_{\text{SE}} : U_{H \times W \times C} \to \tilde{X}_{H \times W \times C}$



не меняет размеры тензора,
но проводит адаптивную перекалибровку каналов (= признаков)

J. Hu и др. «Squeeze-and-Excitation Networks», 2018 https://arxiv.org/pdf/1709.01507.pdf

SENet (Squeeze-and-Excitation Network, 2017)

сжатие (squeeze) – агрегация по каналам (Global pooling):

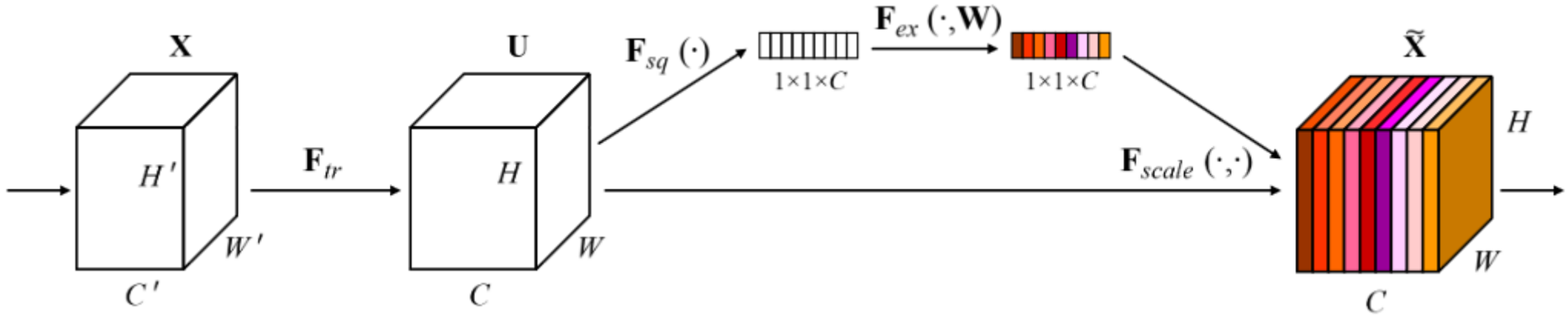
$$F_{\text{sq}} : \| u_{h,w,c} \|_{H\times W\times C} \to \left\| \frac{1}{HW} \sum_{w=1}^{W} \sum_{h=1}^{H} u_{h,w,c} \right\|_{C}$$

возбуждение (excitation) – подготовка коэффициентов (FC + ReLu + FC + Sigmoid):

$$F_{\text{ex}} = \sigma(W_{C\times k}\, \text{ReLu}(V_{k\times C} z_C))$$

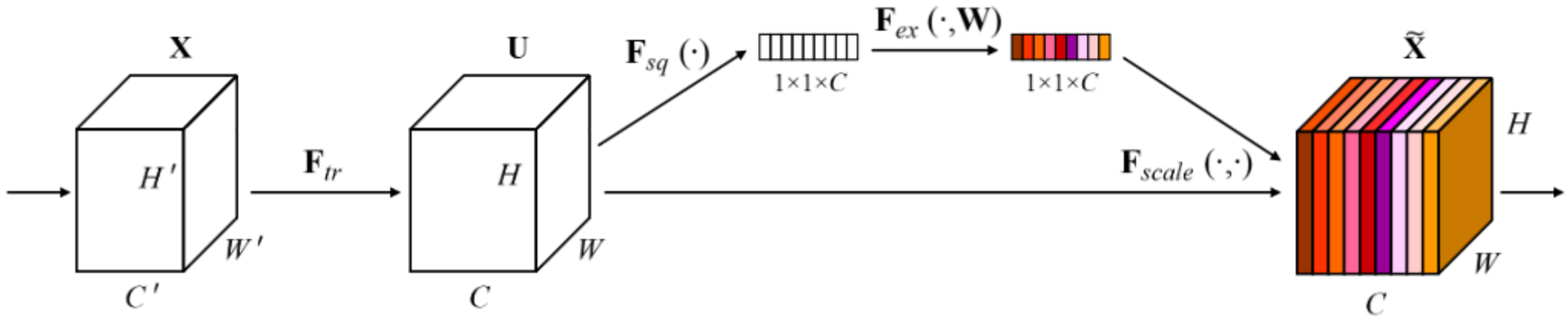
калибровка (Scale) – умножение коэффициентов на каналы

$$F_{\text{scale}} : \| u_{h,w,c} \|_{H\times W\times C} \to \| u_{h,w,c} F_{\text{ex}}(z)_c \|_{C}$$



Минутка кода: SENet

```
class SELayer(nn.Module):
    def __init__(self, channel, reduction=16):
        super(SELayer, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Sequential(nn.Linear(channel, channel // reduction, bias=False),
                                nn.ReLU(inplace=True),
                                nn.Linear(channel // reduction, channel, bias=False),
                                nn.Sigmoid())

    def forward(self, x):
        b, c, _, _ = x.size()
        y = self.avg_pool(x).view(b, c)
        y = self.fc(y).view(b, c, 1, 1)
        return x * y.expand_as(x)
```



Сравнение архитектур https://arxiv.org/pdf/1810.00736.pdf

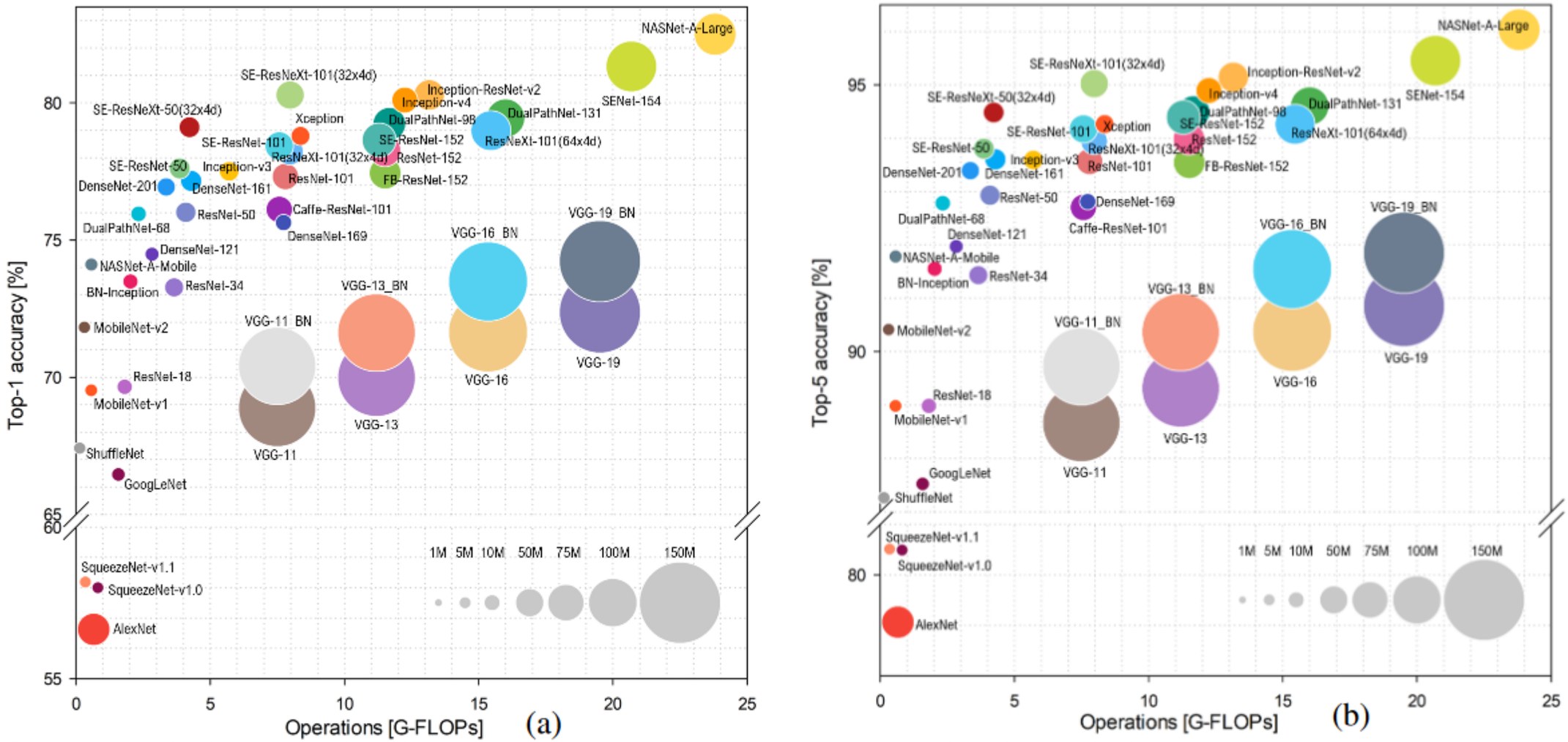
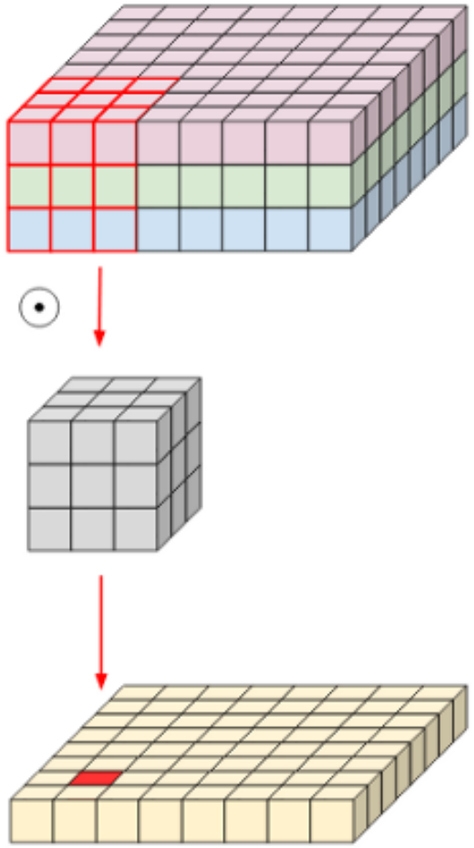


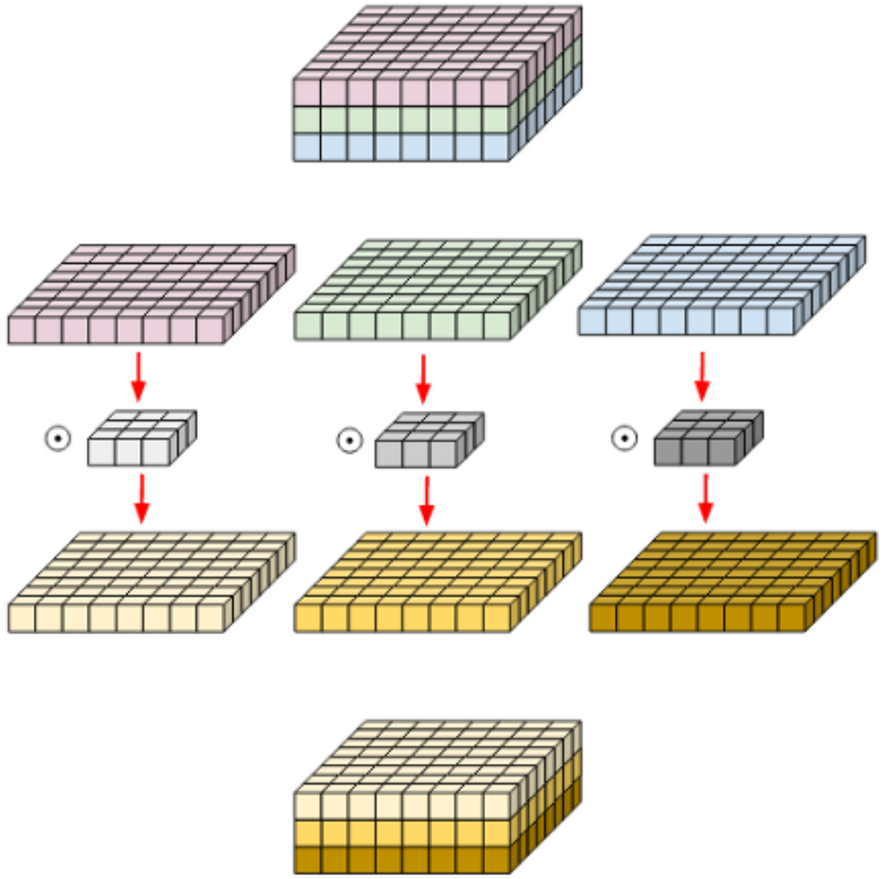
FIGURE 1: Ball chart reporting the Top-1 and Top-5 accuracy *vs.* computational complexity. Top-1 and Top-5 accuracy using only the center crop versus floating-point operations (FLOPs) required for a single forward pass are reported. The size of each ball corresponds to the model complexity. (a) Top-1; (b) Top-5.

Какие бывают свёртки

convolution



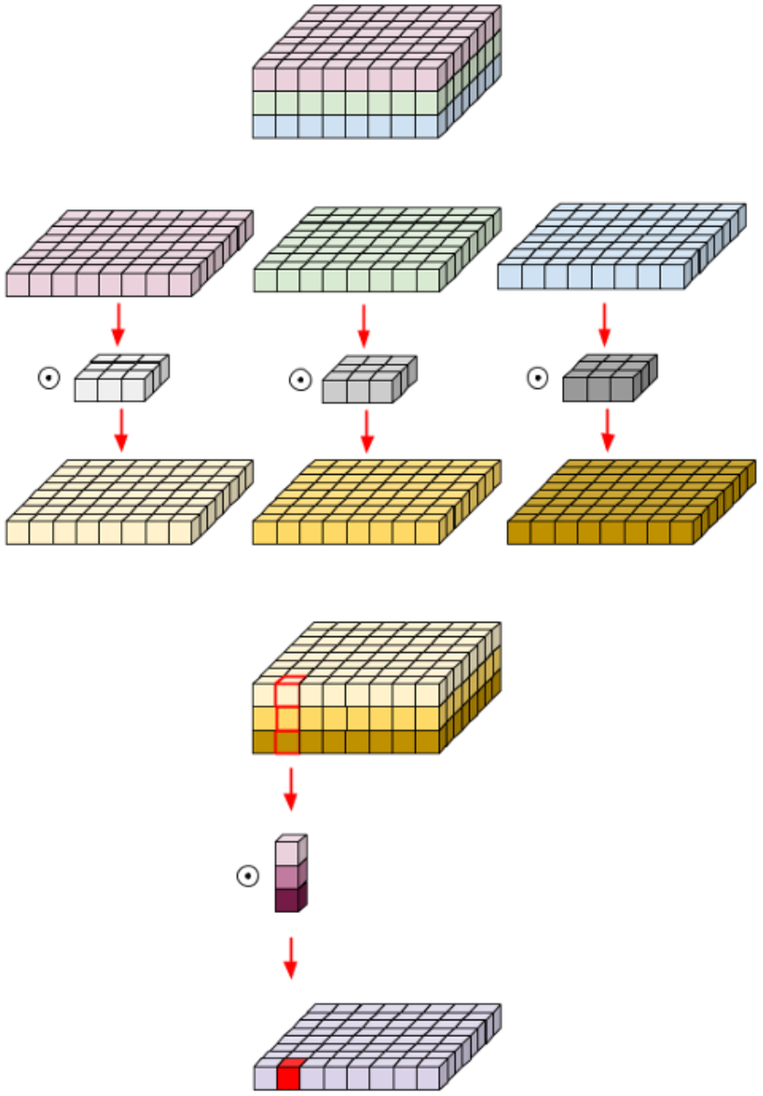
depth-wise convolution



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

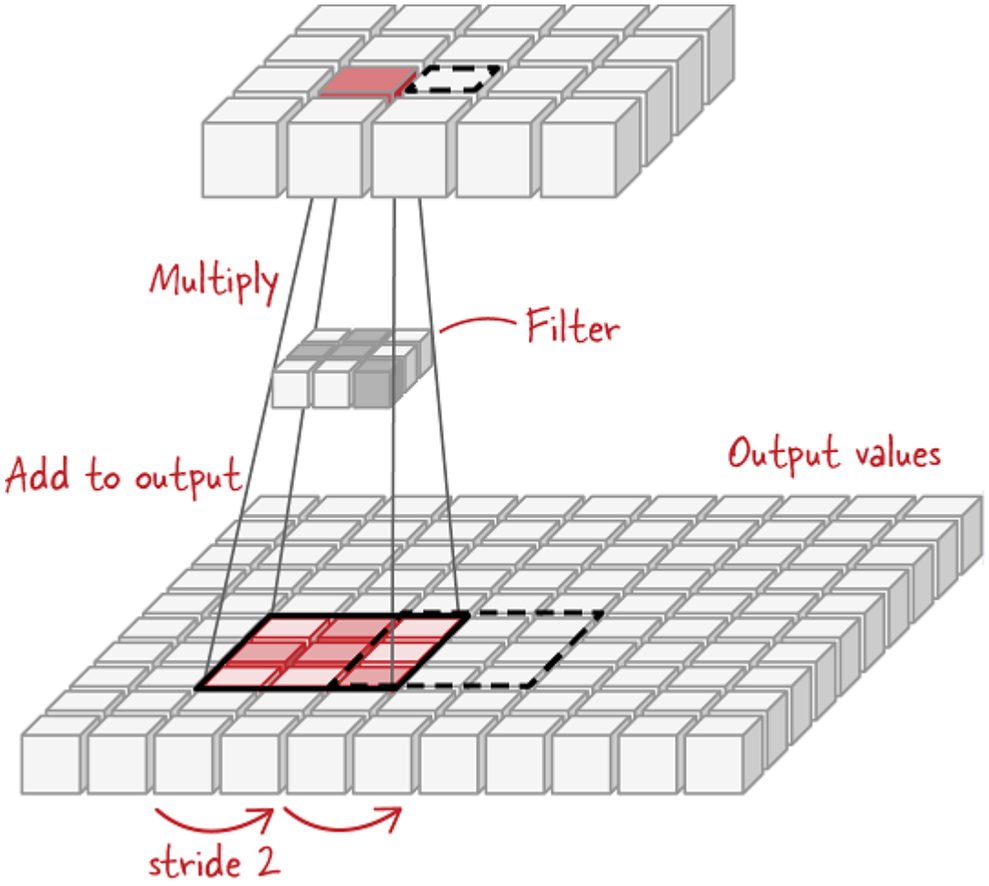
```
S=128, F=3, inC=3, outC=16

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…

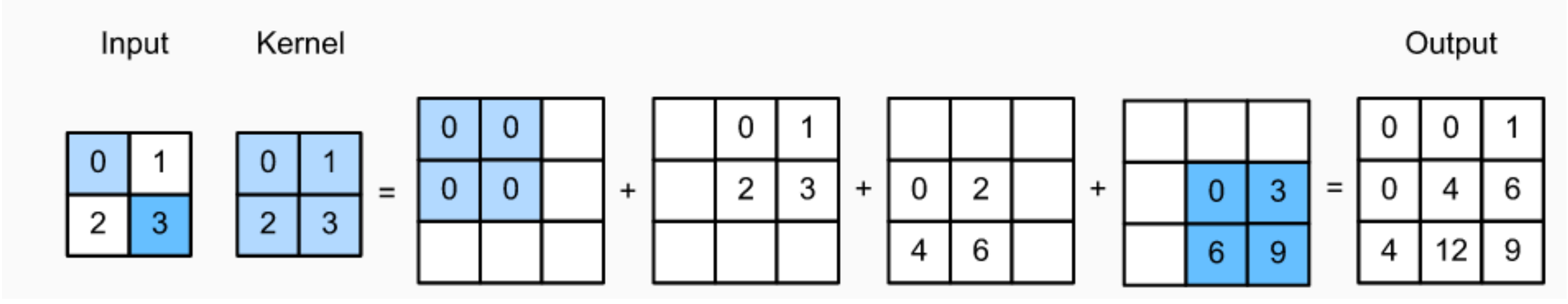
Transposed convolution

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

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

«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])
```

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))
```

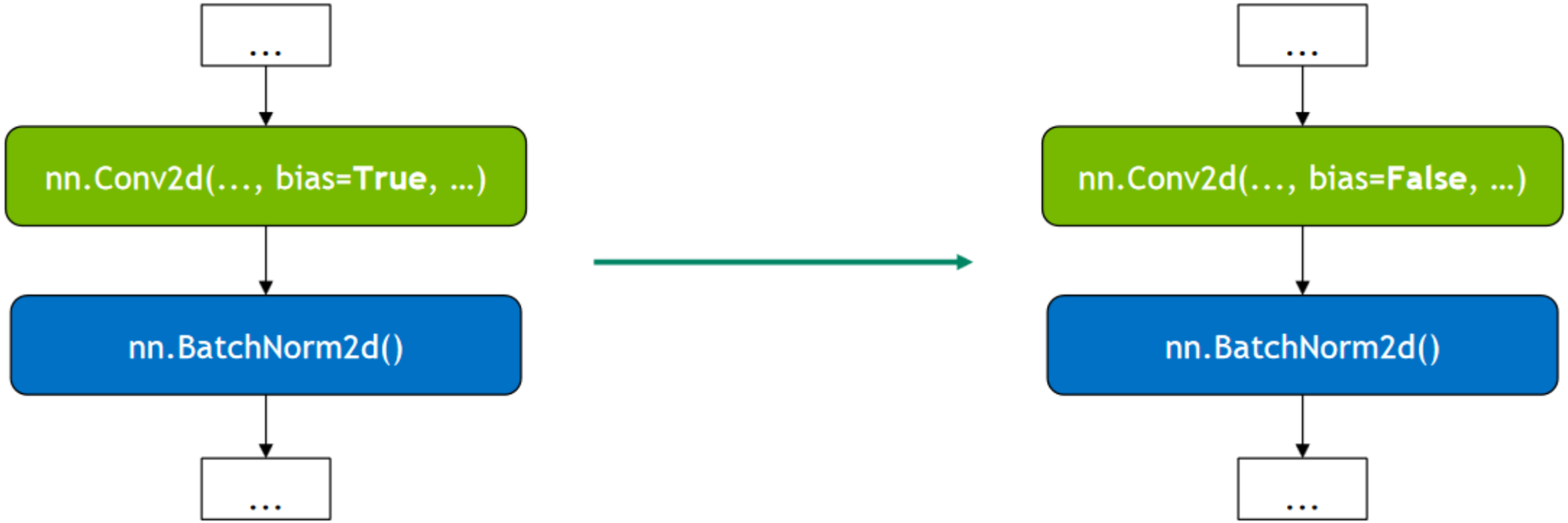
```
tensor([[[[ 0.,  0.],
          [ 0.,  0.]],

         [[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.

P.S. BN при свёртках



https://nvlabs.github.io/eccv2020-mixed-precision-tutorial/files/szymon_migacz-pytorch-performance-tuning-guide.pdf

BN при свёртках

FCN

$$x \sim N \times D$$

$$\mu, \sigma \sim 1 \times D$$

CCN

$$x \sim N \times C \times H \times W$$

$$\mu, \sigma \sim 1 \times C \times 1 \times 1$$

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)

Итог

Есть много стандартных приёмов:

- **каскад свёрток**
- **факторизация свёрток / параметров**
- **1×1-свёртки**
- **узкое горло**
- **прокидывание связей**

Если задача как-то связана с изображениями,
часто берут архитектуру проверенную на ImageNet-е
(детектирование, сегментация, определение позы/действия, img → text)

И даже, если не связана с изображениями…
Volumetric Brain Segmentation (VoxResNet)
City-Wide Crowd Flow Prediction (ST-ResNet)
Generating Realistic Voices (WaveNet)

Литература

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