академия больших данных

mail.ru group

made

Нейросети на мобильных устройствах

Фёдор Киташов

Программист-исследователь в команде компьютерного зрения

План лекции

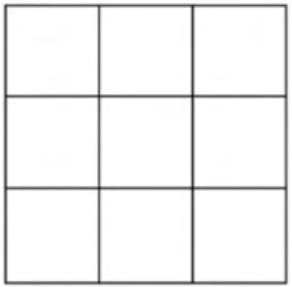
- Повторение: свёртка
- Архитектуры нейросетей для мобильных телефонов
  - Mobilenet v1/v2
    - 1x1 Convs
    - Depthwise separaey convolution
- Квантизация нейронных сетей
  - Статическая квантизация
  - Динамическая квантизация
  - Обучение с учётом квантизации
- Деплой нейронных сетей
  - Pytorch JIT
    - Torchscript
    - JIT compiler
  - ONNX и ONNX Runtime

Свёртка: одномерный пример

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

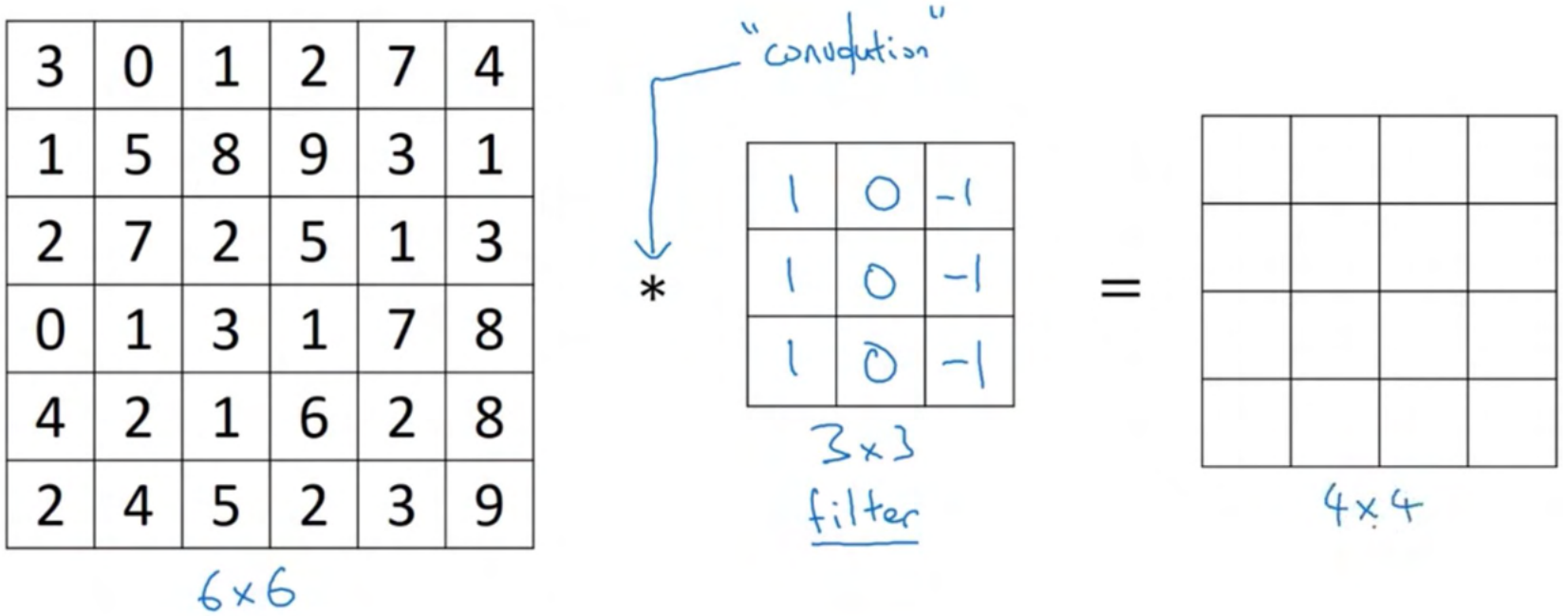
6×6

*



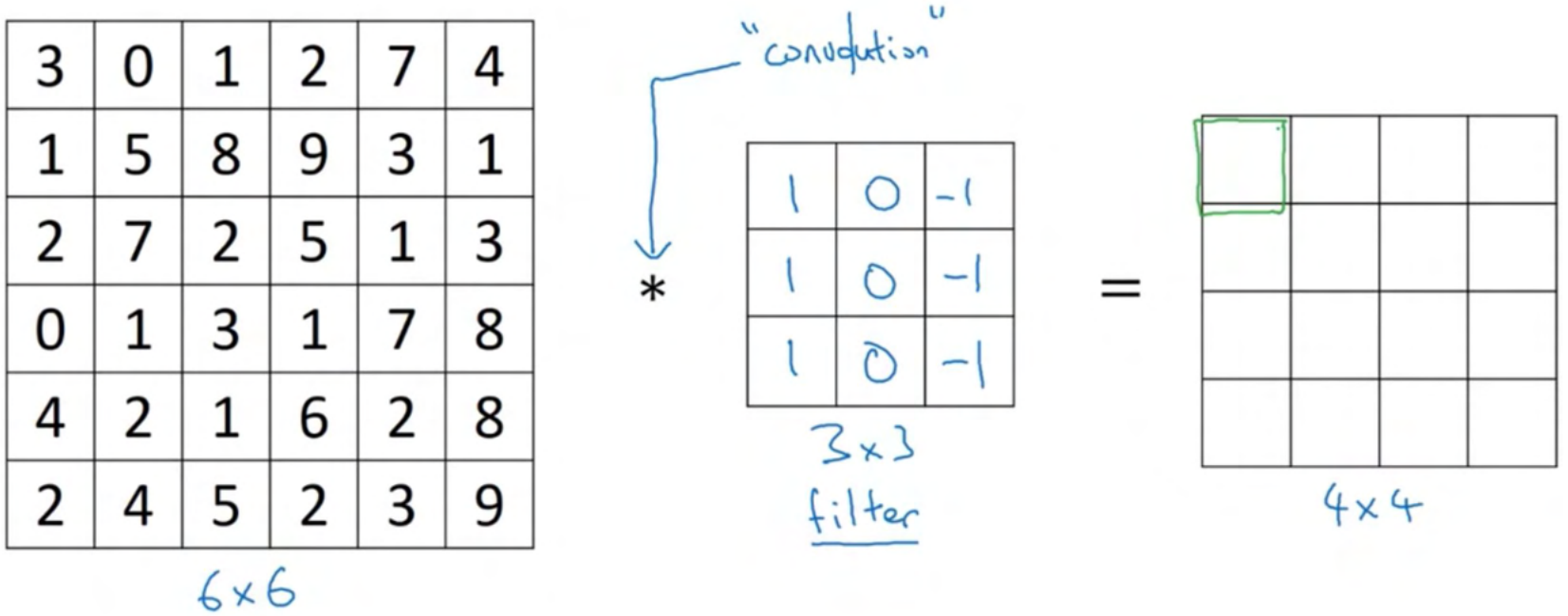
https://youtu.be/XuD4C8vJzEQ: Edge detection example (C4W1L02 Deep Learning)

Свёртка: одномерный пример



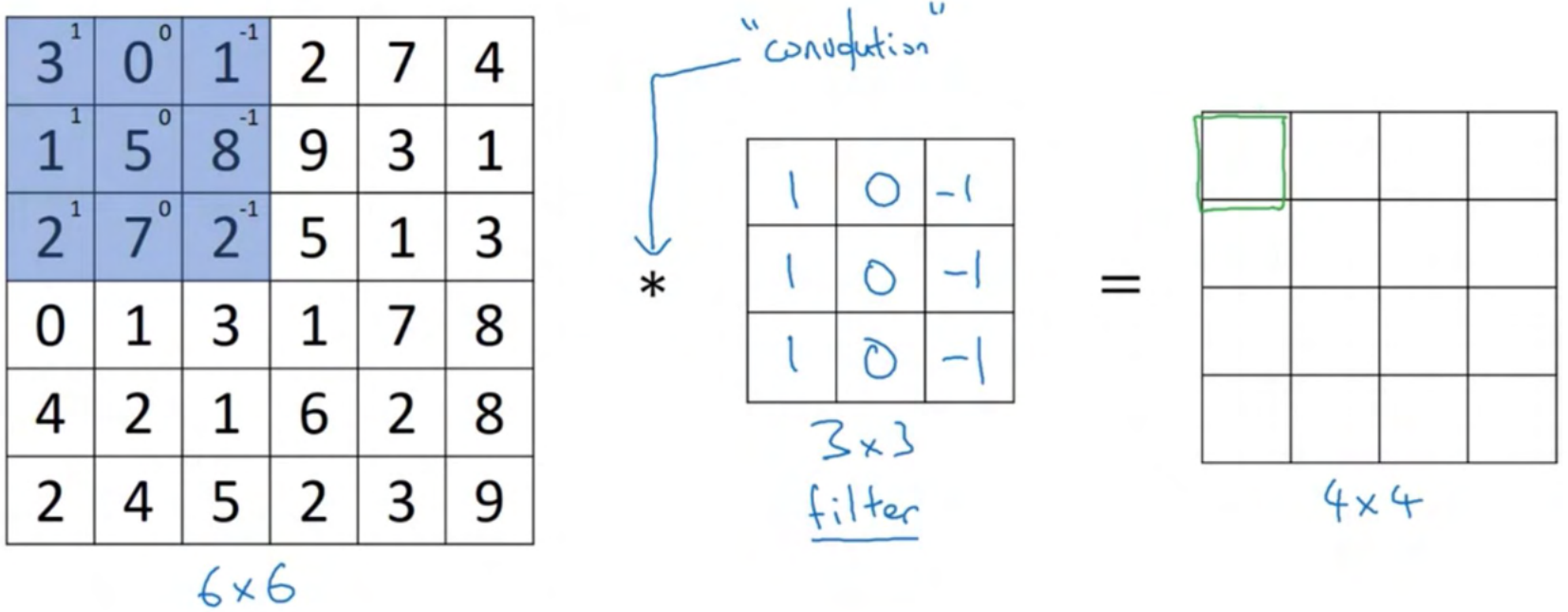
https://youtu.be/XuD4C8vJzEQ: Edge detection example (C4W1L02 Deep Learning)

Свёртка: одномерный пример



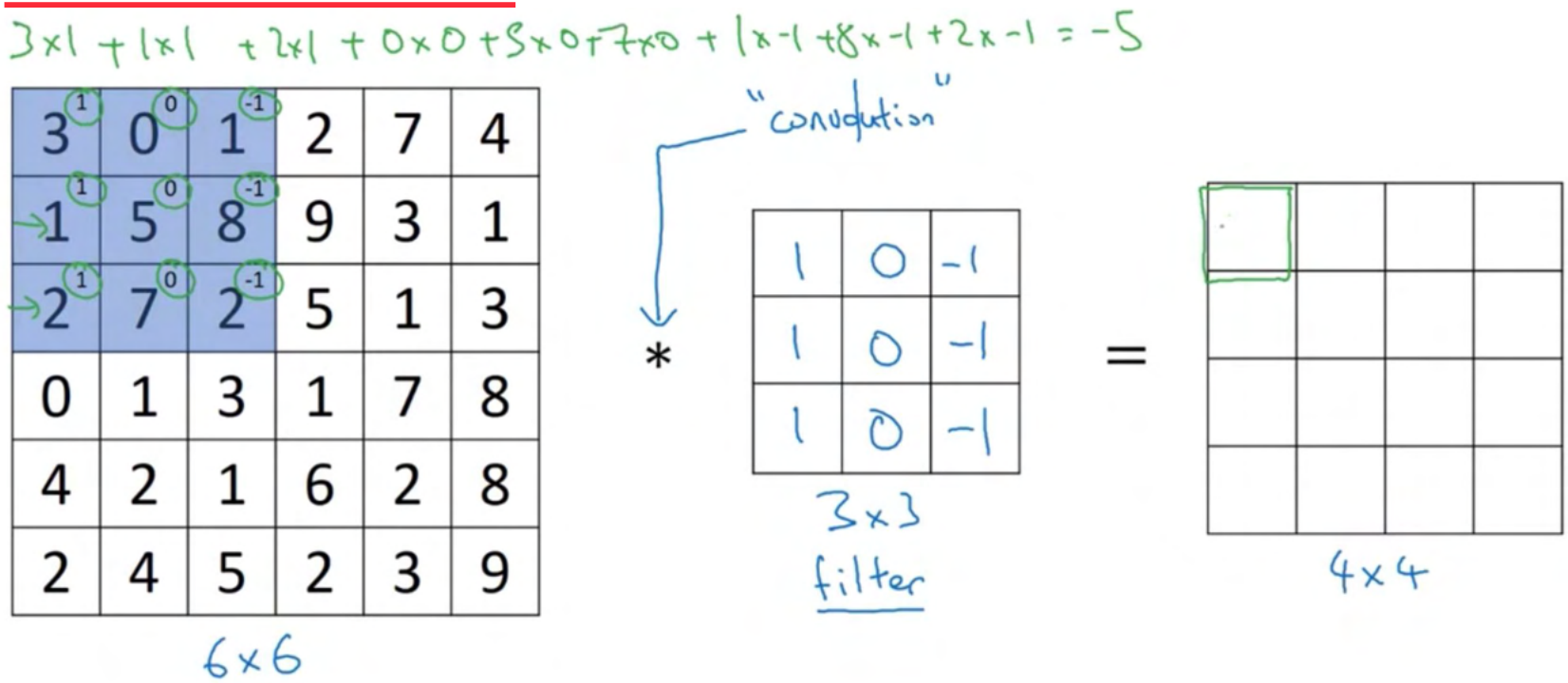
https://youtu.be/XuD4C8vJzEQ: Edge detection example (C4W1L02 Deep Learning)

Свёртка: одномерный пример



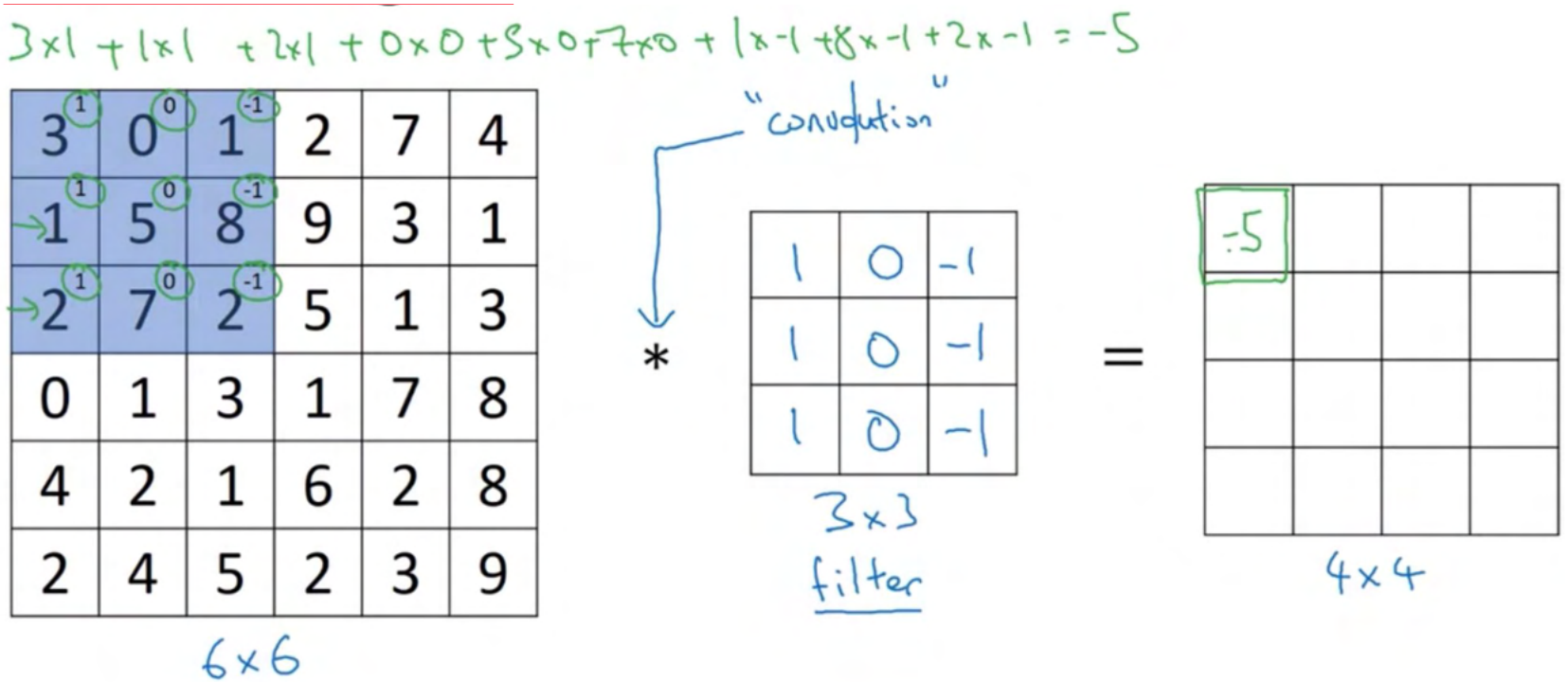
https://youtu.be/XuD4C8vJzEQ: Edge detection example (C4W1L02 Deep Learning)

Свёртка: одномерный пример



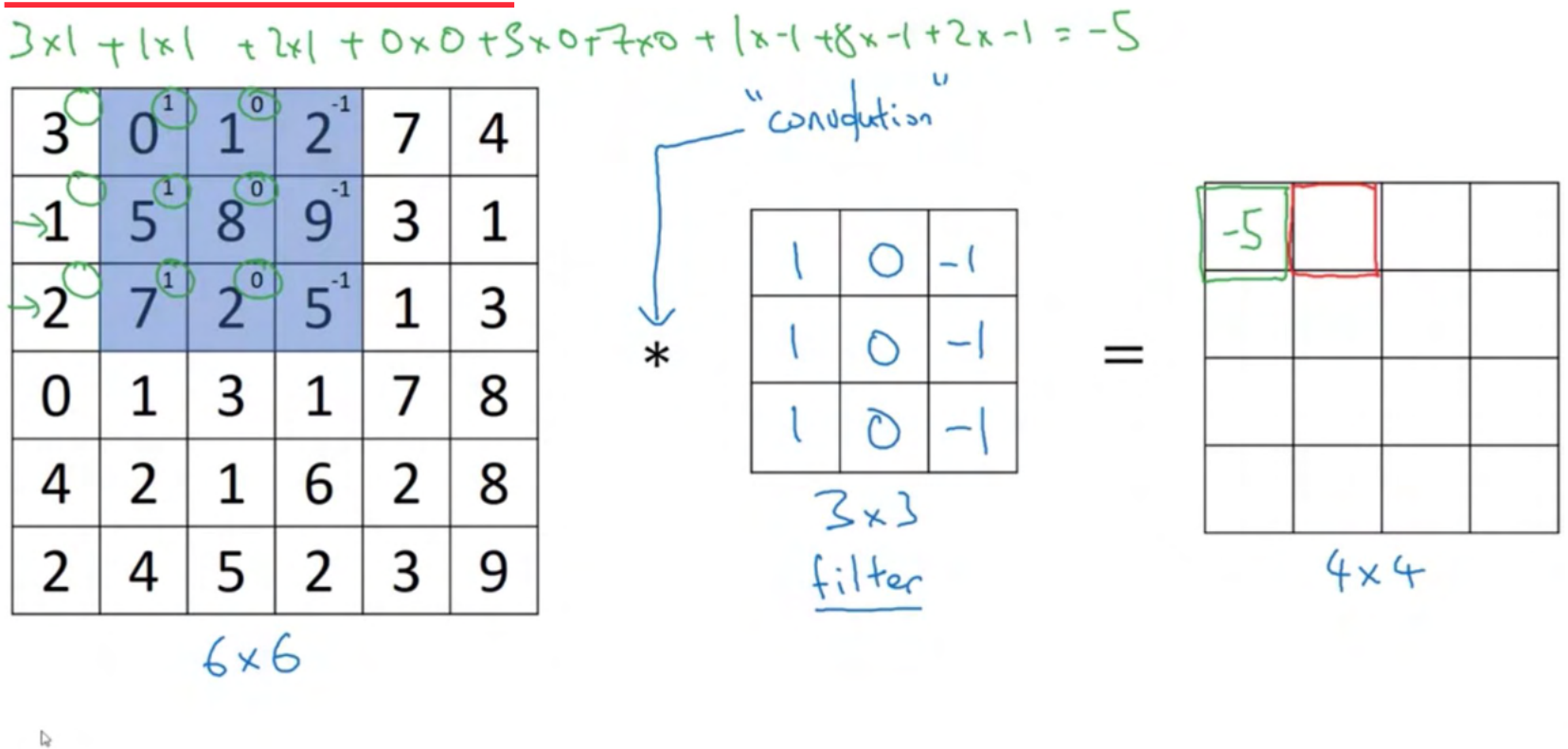
https://youtu.be/XuD4C8vJzEQ: Edge detection example (C4W1L02 Deep Learning)

Свёртка: одномерный пример



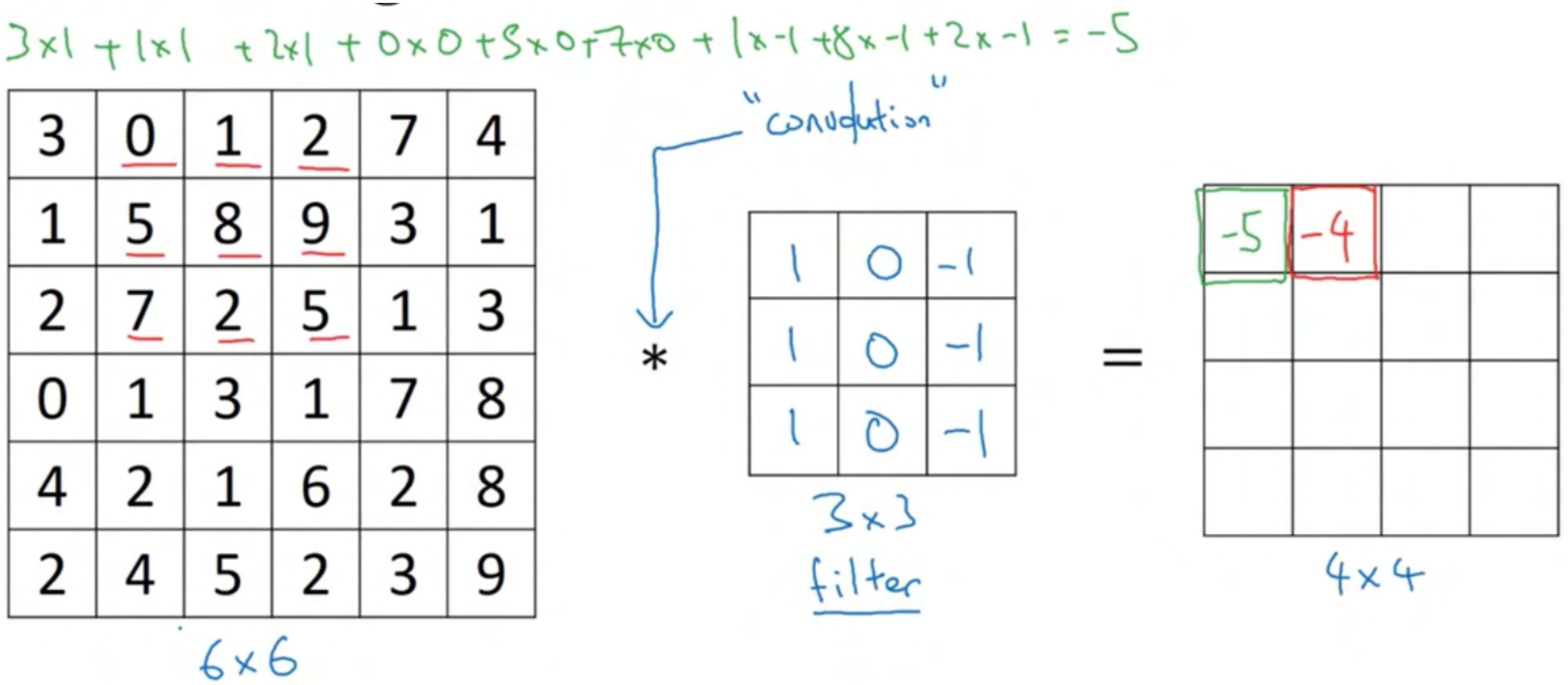
https://youtu.be/XuD4C8vJzEQ: Edge detection example (C4W1L02 Deep Learning)

Свёртка: одномерный пример



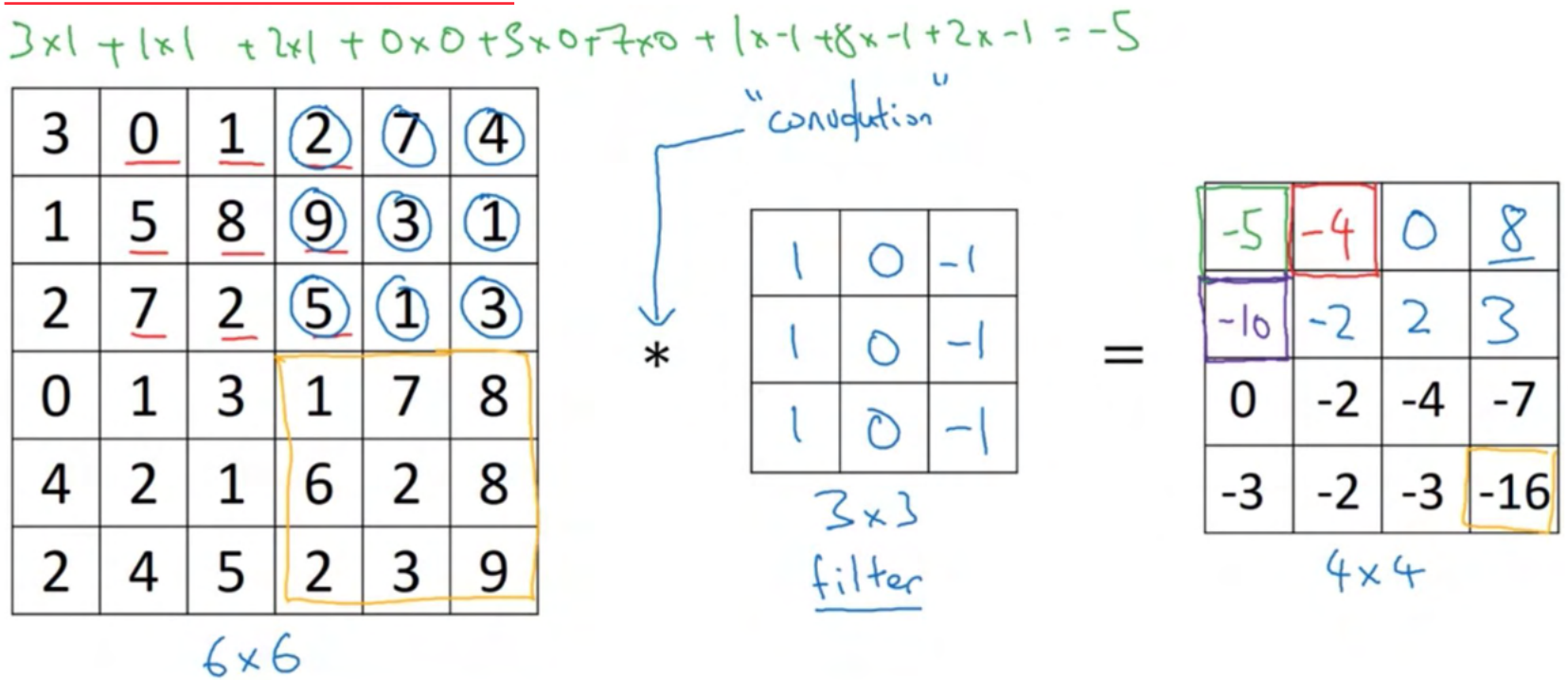
https://youtu.be/XuD4C8vJzEQ: Edge detection example (C4W1L02 Deep Learning)

Свёртка: одномерный пример



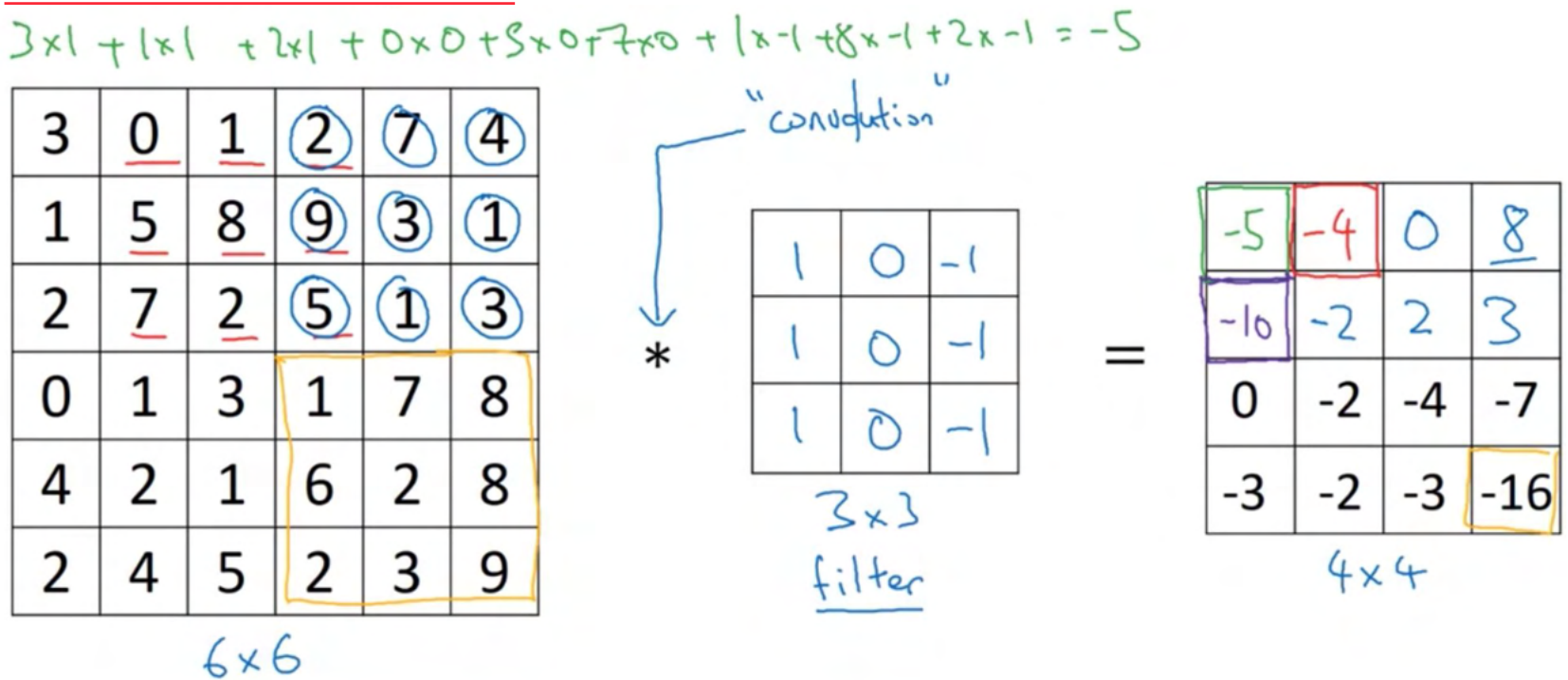
https://youtu.be/XuD4C8vJzEQ: Edge detection example (C4W1L02 Deep Learning)

Свёртка: одномерный пример



https://youtu.be/XuD4C8vJzEQ: Edge detection example (C4W1L02 Deep Learning)

Свёртка: Pytorch Conv2d



https://youtu.be/XuD4C8vJzEQ: Edge detection example (C4W1L02 Deep Learning)

Свёртка: Pytorch Conv2d

```
input_tensor = torch.tensor([[[[3, 0, 1, 2, 7, 4],
                               [1, 5, 8, 9, 3, 1],
                               [2, 7, 2, 5, 1, 3],
                               [0, 1, 3, 1, 7, 8],
                               [4, 2, 1, 6, 2, 8],
                               [2, 4, 5, 2, 3, 9]]]])
```

```
input_tensor.shape
```

```
torch.Size([1, 1, 6, 6])
```

https://youtu.be/XuD4C8vJzEQ: Edge detection example (C4W1L02 Deep Learning)

Свёртка: Pytorch Conv2d

```
input_tensor = torch.tensor([[[[3, 0, 1, 2, 7, 4],
                               [1, 5, 8, 9, 3, 1],
                               [2, 7, 2, 5, 1, 3],
                               [0, 1, 3, 1, 7, 8],
                               [4, 2, 1, 6, 2, 8],
                               [2, 4, 5, 2, 3, 9]]]])
```

```
input_tensor.shape
```

```
torch.Size([1, 1, 6, 6])
```

```
layer = torch.nn.Conv2d(1, 1, kernel_size=3)

# вес доступен через .weight.data
layer.weight.data = torch.tensor([[[[1, 0, -1],
                                    [1, 0, -1],
                                    [1, 0, -1]]]]).float()

layer.bias.data = torch.zeros_like(layer.bias.data)
```

```
layer.weight
```

```
Parameter containing:
tensor([[[[ 1.,  0., -1.],
          [ 1.,  0., -1.],
          [ 1.,  0., -1.]]]], requires_grad=True)
```

https://youtu.be/XuD4C8vJzEQ: Edge detection example (C4W1L02 Deep Learning)

Свёртка: Pytorch Conv2d

```
input_tensor = torch.tensor([[[[3, 0, 1, 2, 7, 4],
                               [1, 5, 8, 9, 3, 1],
                               [2, 7, 2, 5, 1, 3],
                               [0, 1, 3, 1, 7, 8],
                               [4, 2, 1, 6, 2, 8],
                               [2, 4, 5, 2, 3, 9]]]])
```

```
input_tensor.shape
```

```
torch.Size([1, 1, 6, 6])
```

```
layer = torch.nn.Conv2d(1, 1, kernel_size=3)

# вес доступен через .weight.data
layer.weight.data = torch.tensor([[[[1, 0, -1],
                                    [1, 0, -1],
                                    [1, 0, -1]]]]).float()

layer.bias.data = torch.zeros_like(layer.bias.data)
```

```
layer.weight
```

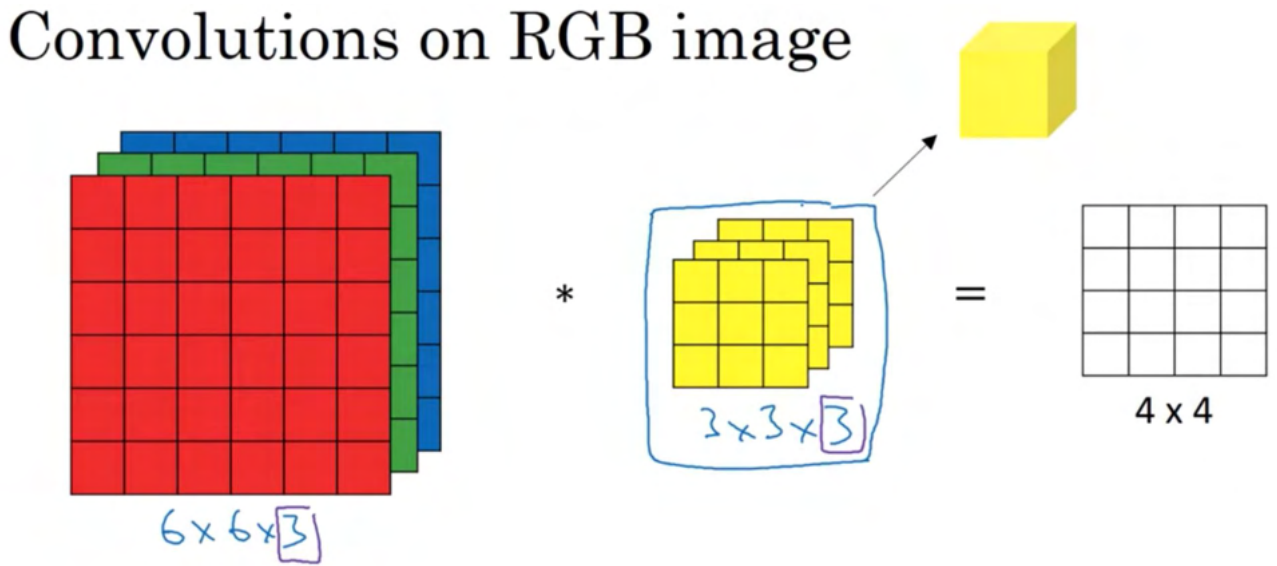
```
Parameter containing:
tensor([[[[ 1.,  0., -1.],
          [ 1.,  0., -1.],
          [ 1.,  0., -1.]]]], requires_grad=True)
```

```
output_tensor = layer(input_tensor.float())
```

```
output_tensor
```

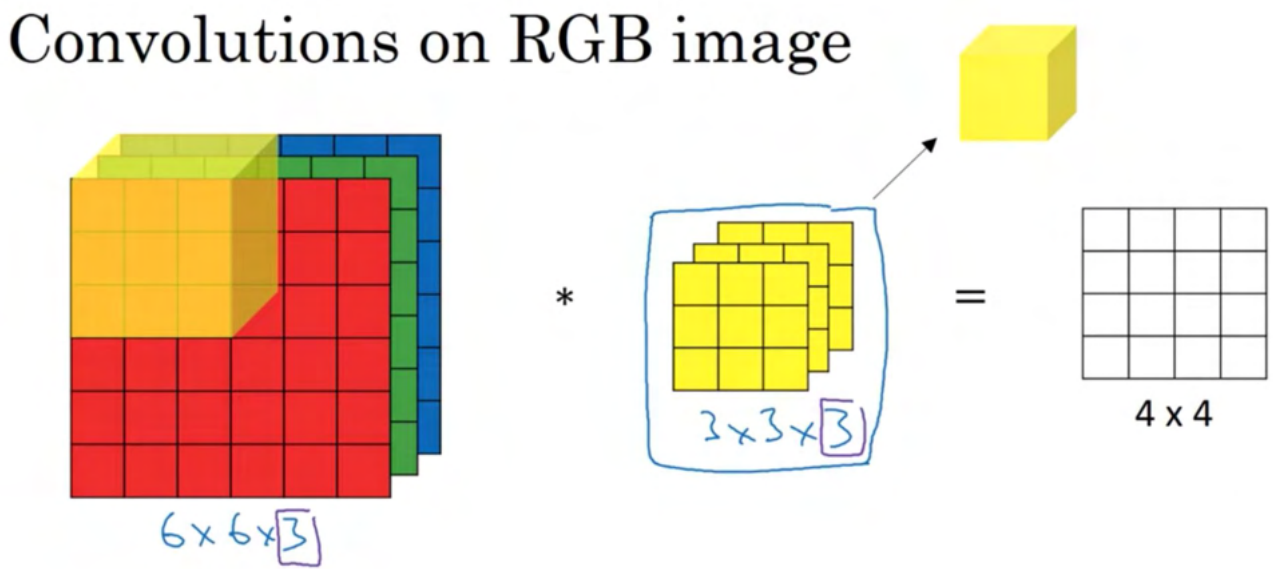
```
tensor([[[[ -5.,  -4.,   0.,   8.],
          [-10.,  -2.,   2.,   3.],
          [  0.,  -2.,  -4.,  -7.],
          [ -3.,  -2.,  -3., -16.]]]], grad_fn=<ThnnConv2DBackward>)
```

Свёртка: rgb пример



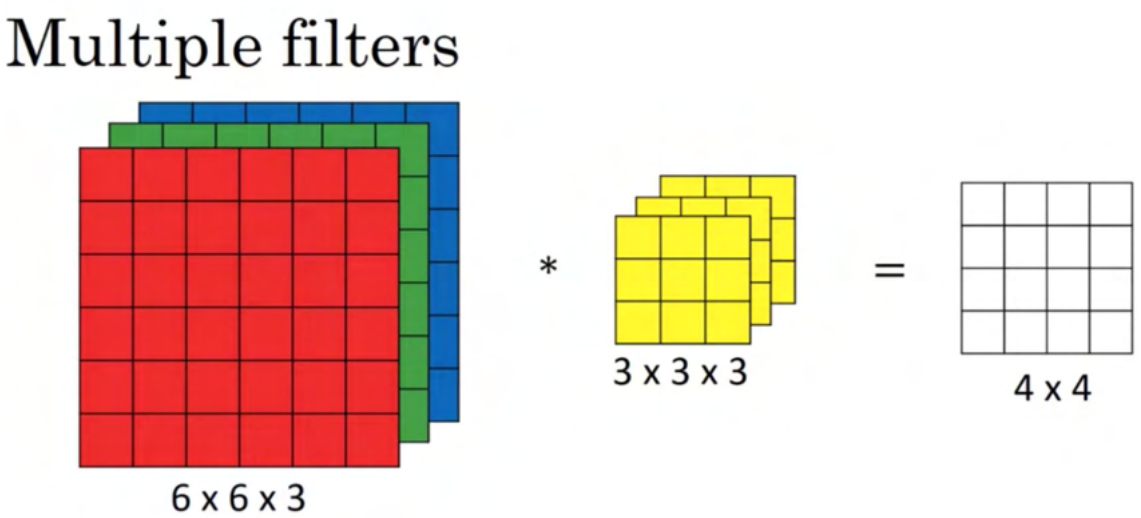
https://youtu.be/KTB_OFoAQcc: Convolutions over volumes (C4W1L06 Deep Learning)

Свёртка: rgb пример



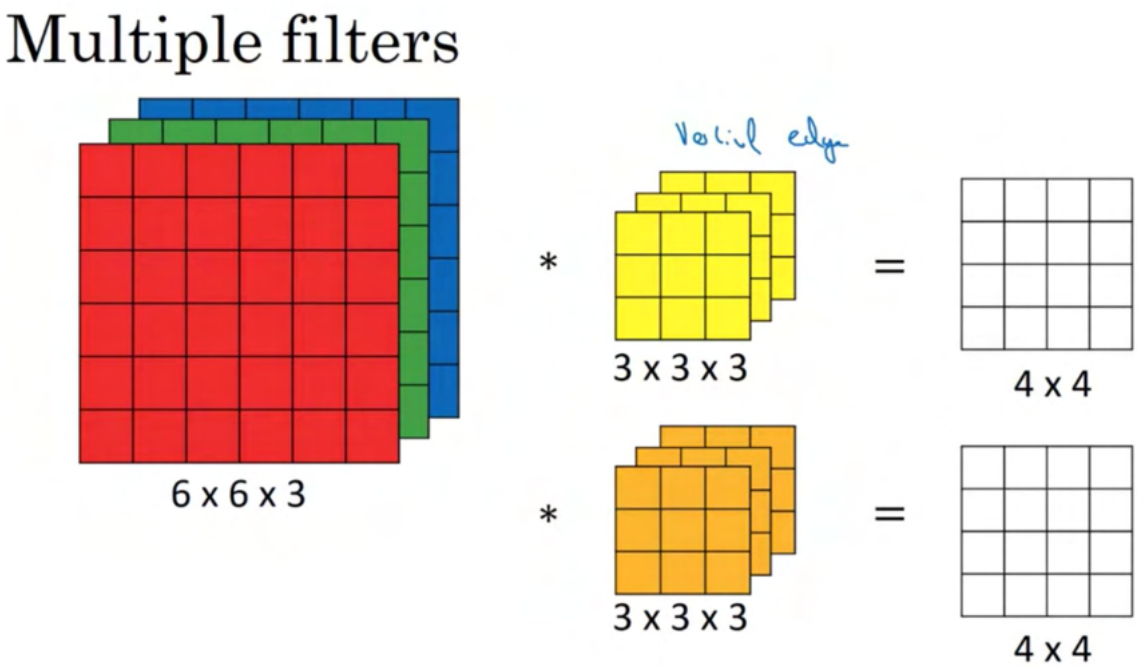
https://youtu.be/KTB_OFoAQcc: Convolutions over volumes (C4W1L06 Deep Learning)

Свёртка: rgb пример



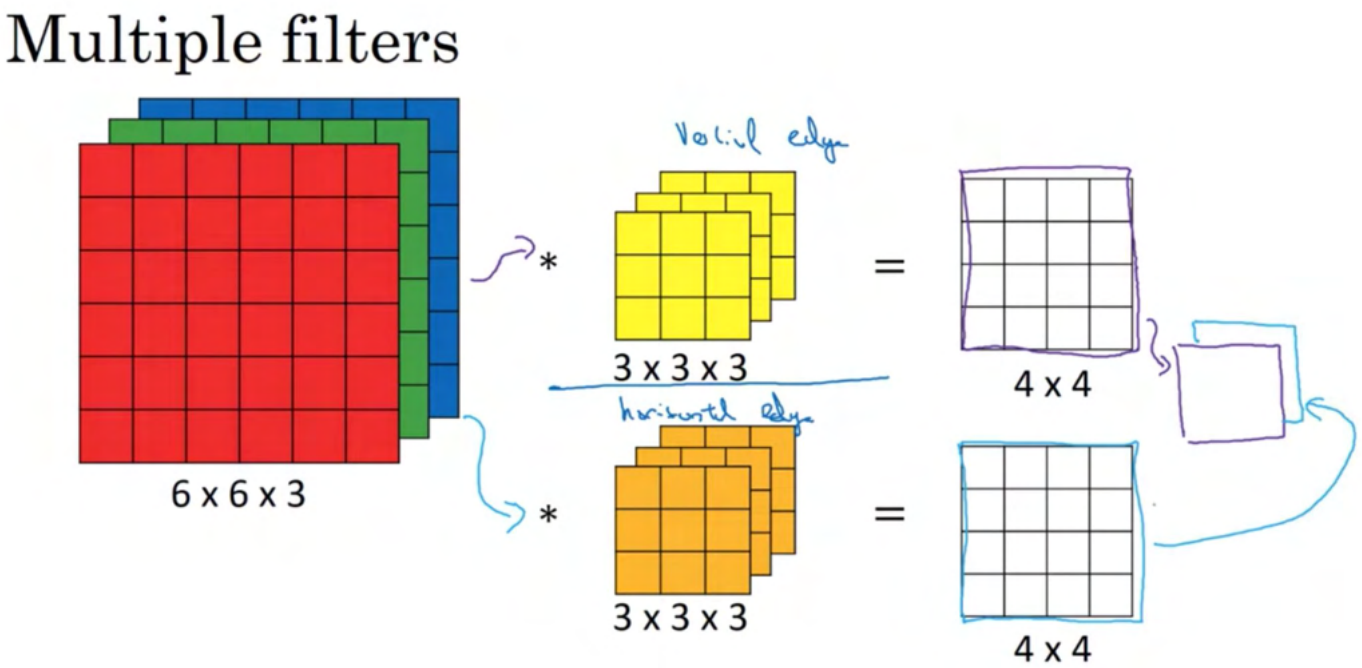
https://youtu.be/KTB_OFoAQcc: Convolutions over volumes (C4W1L06 Deep Learning)

Свёртка: rgb пример



https://youtu.be/KTB_OFoAQcc: Convolutions over volumes (C4W1L06 Deep Learning)

Свёртка: rgb пример



https://youtu.be/KTB_OFoAQcc: Convolutions over volumes (C4W1L06 Deep Learning)

Свёртка: rgb пример

RGB example

```
input_tensor = torch.randn((1, 3, 6, 6))
```

```
layer = torch.nn.Conv2d(in_channels=3, out_channels=2, kernel_size=3)
```

```
output_tensor = layer(input_tensor.float())
```

```
output_tensor.shape
```

```
torch.Size([1, 2, 4, 4])
```

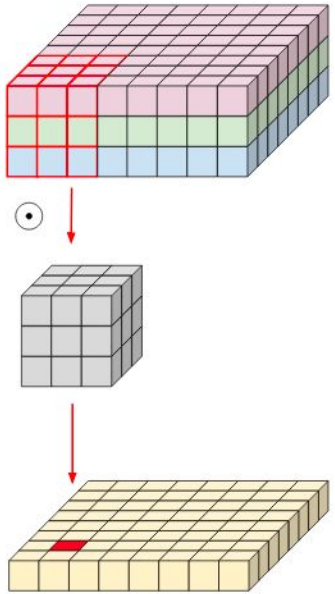
https://youtu.be/KTB_OFoAQcc: Convolutions over volumes (C4W1L06 Deep Learning)

Building blocks

- Convolution (Conv2d)
- 1x1 convolution
- Depthwise separate convolution

Simple convolution

- В обычном сверточном слое 1 фильтр выполняет одновременно
  - свертку по ширине/высоте
  - смешивание различных каналов

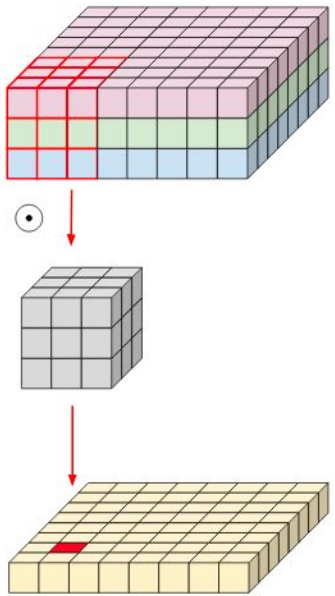


https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

Simple convolution

- В обычном сверточном слое 1 фильтр выполняет одновременно
  - свертку по ширине/высоте
  - смешивание различных каналов

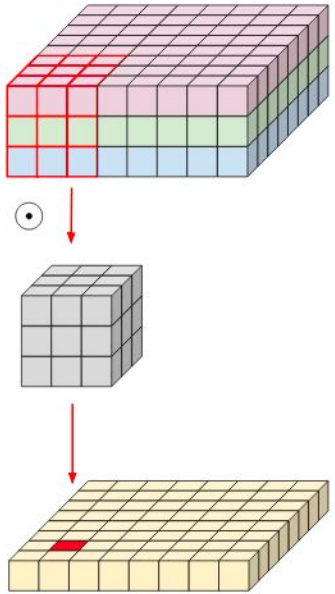
Вход размера HxWxD, ядро KxK, всего N фильтров
Сколько операций требуется для вычисления?



https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

Simple convolution

- В обычном сверточном слое 1 фильтр выполняет одновременно
  - свертку по ширине/высоте
  - смешивание различных каналов



Вход размера HxWxD, ядро KxK, всего N фильтров

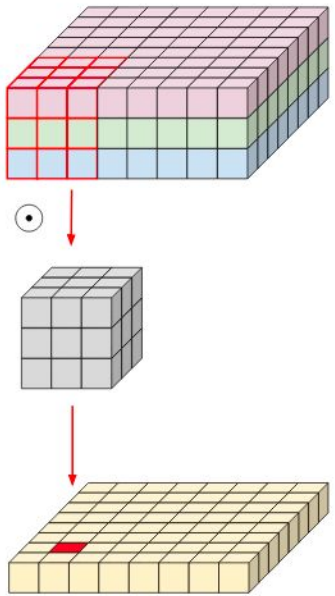
Сколько операций требуется для вычисления?

$$(K \times K \times D) \times (H \times W) \times N$$

https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

Simple convolution

- В обычном сверточном слое 1 фильтр выполняет одновременно
  - свертку по ширине/высоте
  - смешивание различных каналов



Вход размера HxWxD, ядро KxK, всего N фильтров

Сколько операций требуется для вычисления?

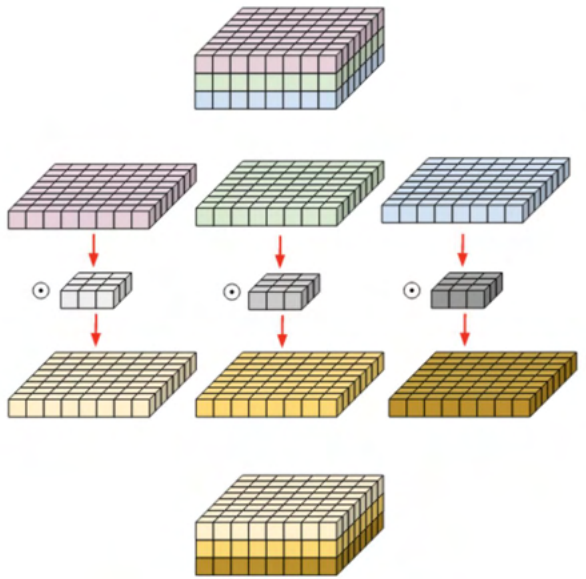
$$(K \times K \times D) \times (H \times W) \times N$$

При K = 3, H = W = 64, D = 256, N = 256: 2.4E09

https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

Depthwise convolution

- Позволим 1 фильтру видеть только 1 канал входного тенора

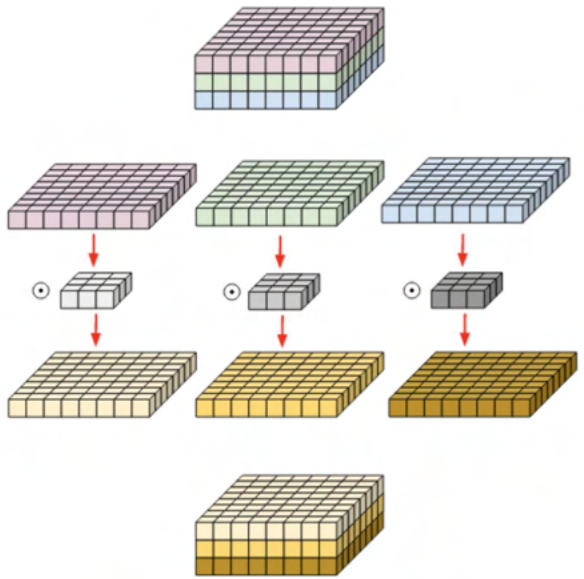


https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

Depthwise convolution

- Позволим 1 фильтру видеть только 1 канал входного тенора

Вход размера HxWxD, ядро KxK, всего N фильтров
Сколько операций требуется для вычисления?



https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

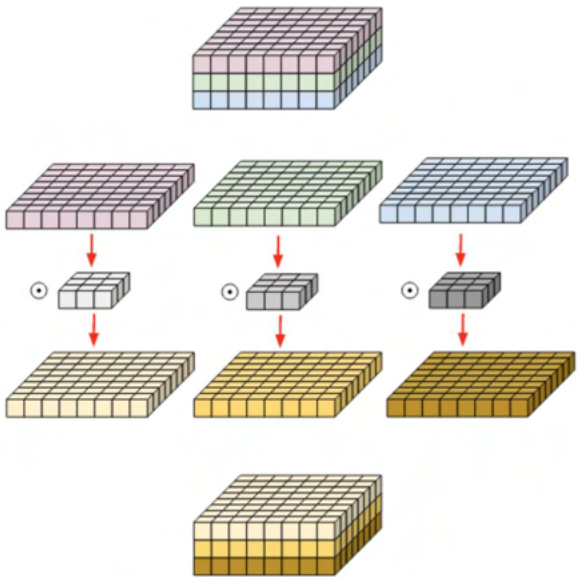
Depthwise convolution

- Позволим 1 фильтру видеть только 1 канал входного тенора

Вход размера HxWxD, ядро KxK, всего N фильтров
Сколько операций требуется для вычисления?

$$(K \times K \times 1) \times (H \times W) \times N$$

- Число операций уменьшилось в D раз



https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

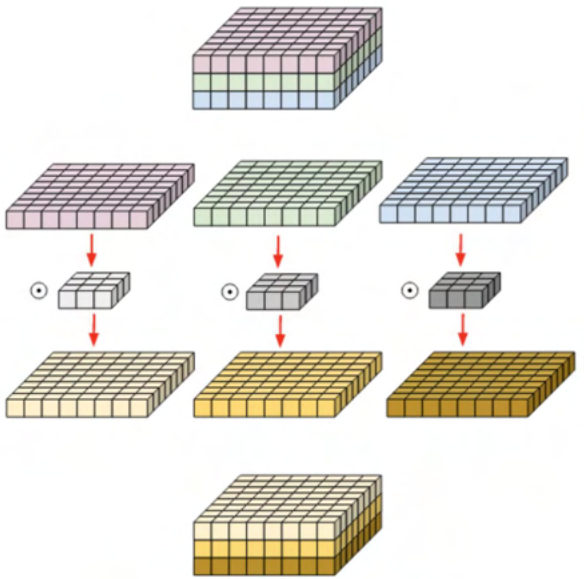
Depthwise convolution

- Позволим 1 фильтру видеть только 1 канал входного тенора

Вход размера HxWxD, ядро KxK, всего N фильтров
Сколько операций требуется для вычисления?

$$(K \times K \times 1) \times (H \times W) \times N$$

- Число операций уменьшилось в D раз
- Проблема: каналы остались независимыми



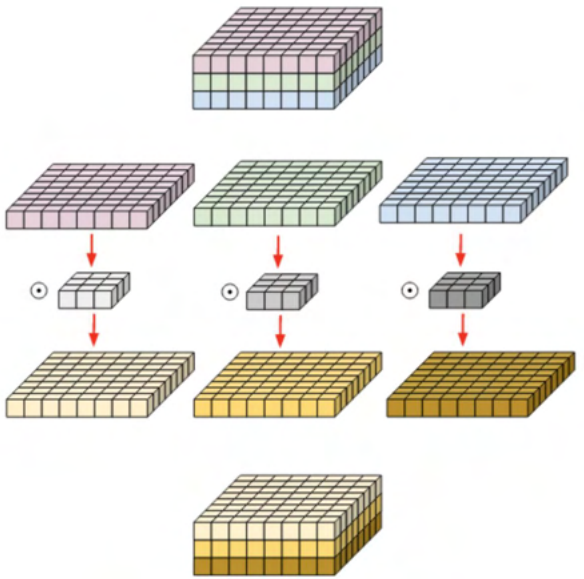
https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

Depthwise convolution

- Позволим 1 фильтру видеть только 1 канал входного тенора

Вход размера HxWxD, ядро KxK, всего N фильтров
Сколько операций требуется для вычисления?

$$(K \times K \times 1) \times (H \times W) \times N$$

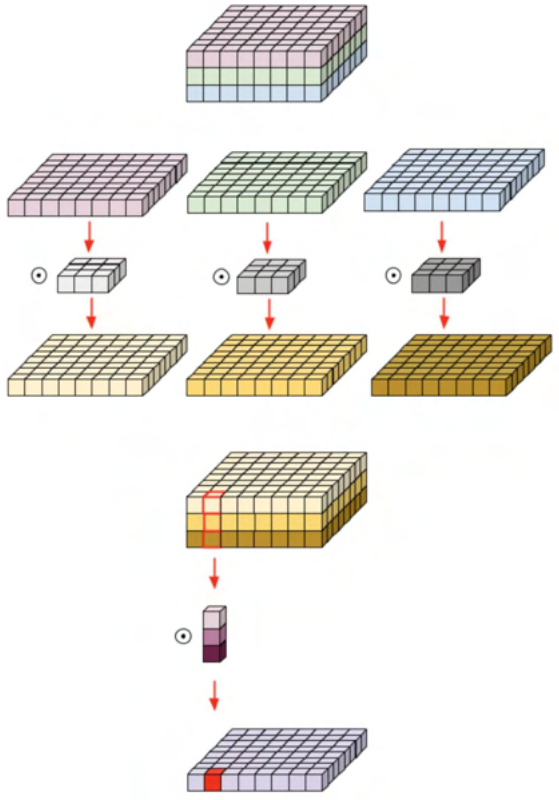


- Число операций уменьшилось в D раз
- Проблема: каналы остались независимыми
- Решение: свертка 1x1 (pointwise convolution)

https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

Depthwise + pointwise = depthwise-separable

- Сначала свертка depthwise (KxK)
- Затем свертка pointwise (1x1)



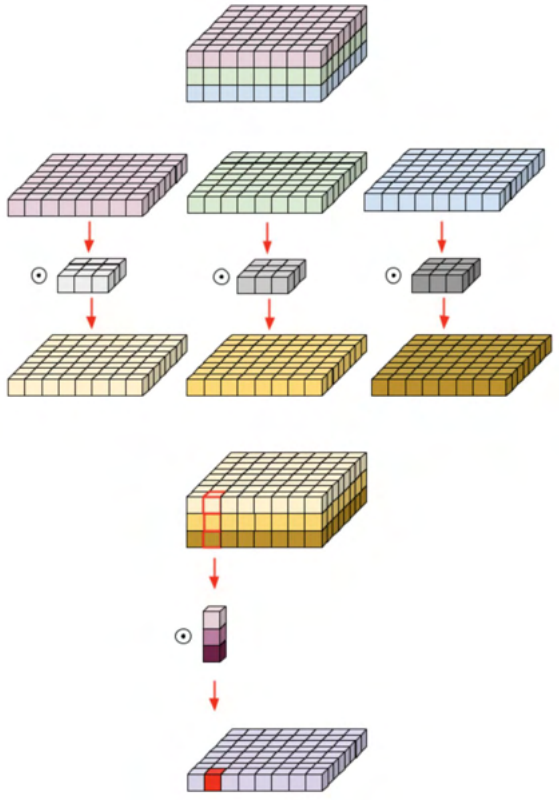
https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

Depthwise + pointwise = depthwise-separable

- Сначала свертка depthwise (KxK)
- Затем свертка pointwise (1x1)

Вход размера HxWxD, ядро KxK, всего N фильтров

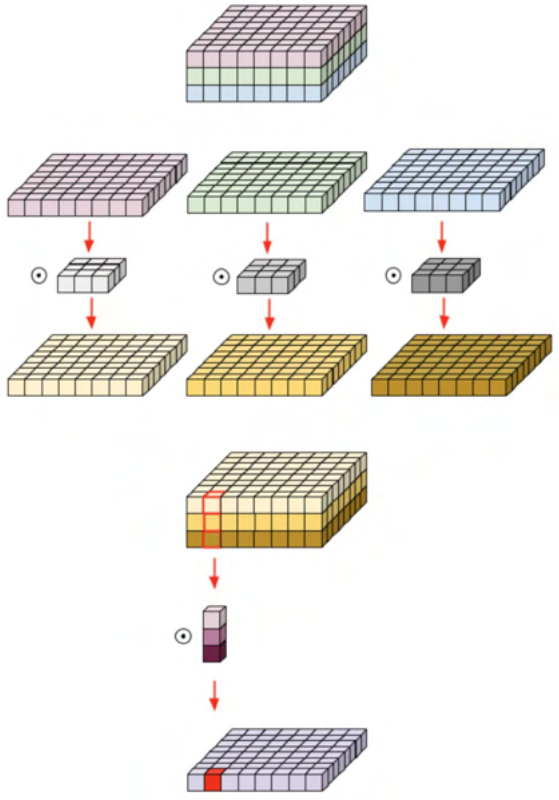
Сколько операций требуется для вычисления?



https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

Depthwise + pointwise = depthwise-separable

- Сначала свертка depthwise (KxK)
- Затем свертка pointwise (1x1)



Вход размера HxWxD, ядро KxK, всего N фильтров

Сколько операций требуется для вычисления?

Depthwise: (K x K x 1) x (H x W) x N

https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

Depthwise + pointwise = depthwise-separable

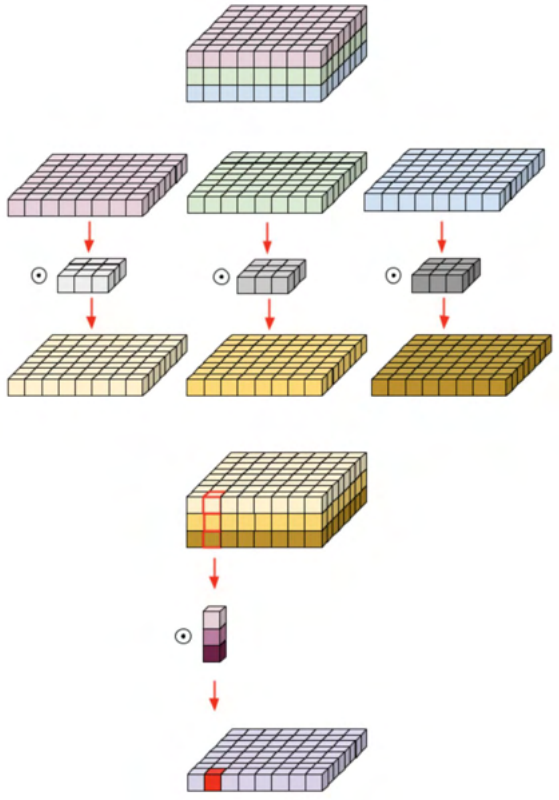
- Сначала свертка depthwise (KxK)
- Затем свертка pointwise (1x1)

Вход размера HxWxD, ядро KxK, всего N фильтров

Сколько операций требуется для вычисления?

Depthwise: (K x K x 1) x (H x W) x N

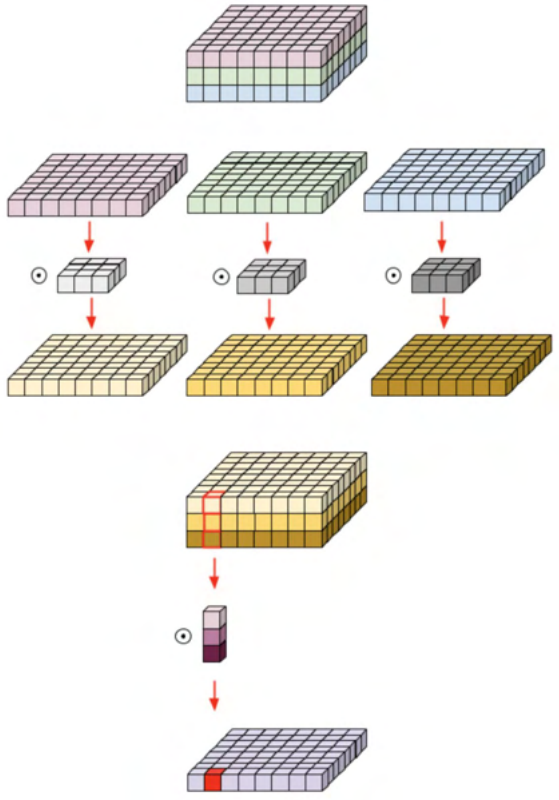
Pointwise: (1 x 1 x D) x (H x W) x N



https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

Depthwise + pointwise = depthwise-separable

- Сначала свертка depthwise (KxK)
- Затем свертка pointwise (1x1)



Вход размера HxWxD, ядро KxK, всего N фильтров

Сколько операций требуется для вычисления?

Depthwise: (K x K x 1) x (H x W) x N

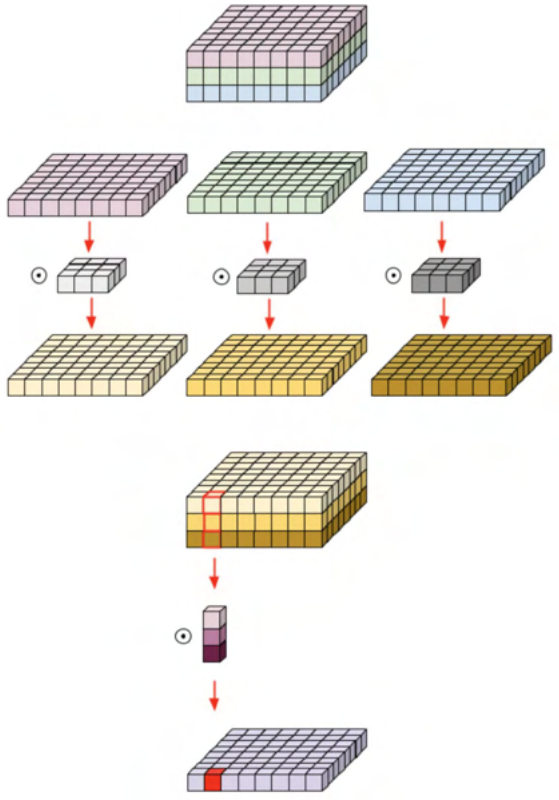
Pointwise: (1 x 1 x D) x (H x W) x N

Total: (K x K + D) x (H x W) x N

https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

Depthwise + pointwise = depthwise-separable

- Сначала свертка depthwise (KxK)
- Затем свертка pointwise (1x1)



Вход размера HxWxD, ядро KxK, всего N фильтров

Сколько операций требуется для вычисления?

Depthwise: (K x K x 1) x (H x W) x N

Pointwise: (1 x 1 x D) x (H x W) x N

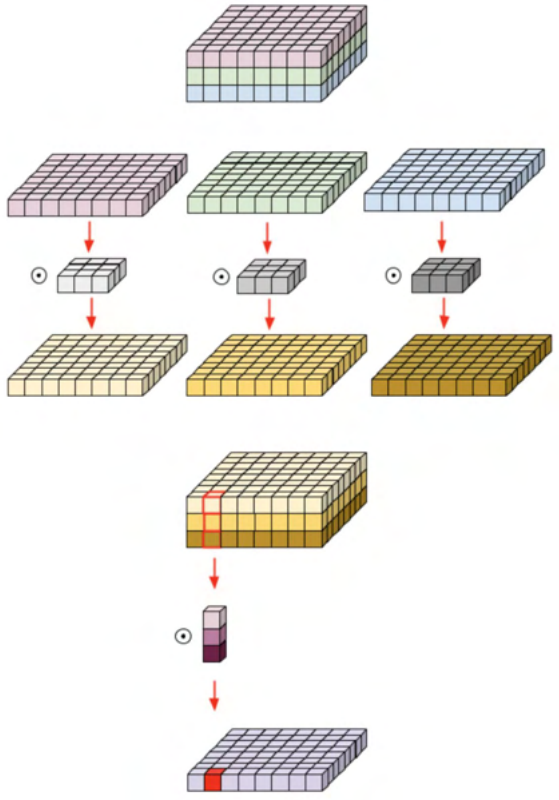
Total: (K x K + D) x (H x W) x N

При K = 3, H = W = 64, D = 256, N = 256: 2.8E08

https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

Depthwise + pointwise = depthwise-separable

- Сначала свертка depthwise (KxK)
- Затем свертка pointwise (1x1)



Вход размера HxWxD, ядро KxK, всего N фильтров

Сколько операций требуется для вычисления?

Depthwise: (K x K x 1) x (H x W) x N

Pointwise: (1 x 1 x D) x (H x W) x N

Total: (K x K + D) x (H x W) x N

При K = 3, H = W = 64, D = 256, N = 256: 2.8E08

С обычной свёрткой было 2.4E09!

https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

MobileNet (2017-...)

- Семейство (из нескольких поколений!) легких моделей, предназначенных для работы на мобильных устройствах
- V1 (2017): MobileNets: Efficient CNNs for Mobile Vision Applications
  - Разделение вычислений по HW и по D (depthwise-separable convolutions)
- V2 (2018): MobileNetV2: Inverted Residuals and Linear Bottlenecks
- V3 (2019): Searching for MobileNetV3

MobileNet

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

MobileNet

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

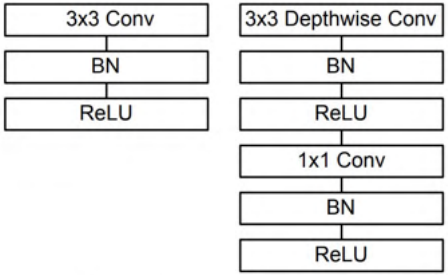


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

MobileNet

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

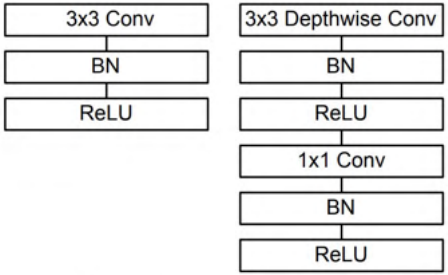


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

MobileNet

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

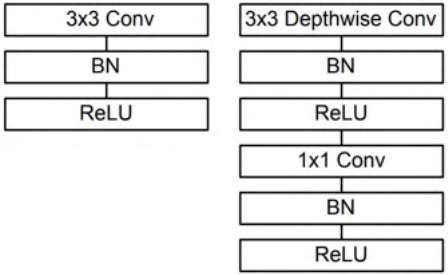


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Table 2. Resource Per Layer Type

Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

MobileNet v2

Table 2. Resource Per Layer Type

Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

MobileNet v2

Table 2. Resource Per Layer Type

Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

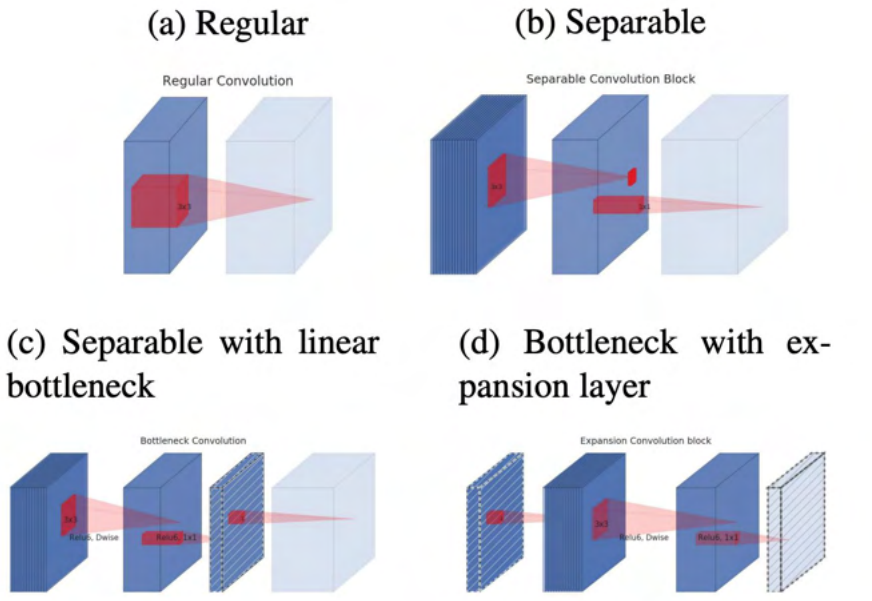
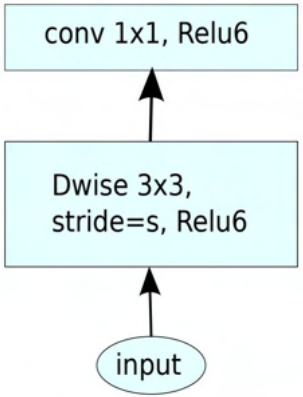


Figure 2: Evolution of separable convolution blocks. The diagonally hatched texture indicates layers that do not contain non-linearities. The last (lightly colored) layer indicates the beginning of the next block. Note: 2d and 2c are equivalent blocks when stacked. Best viewed in color.

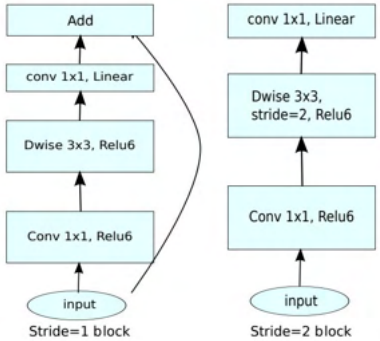
MobileNet v2

Table 2. Resource Per Layer Type

Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%



(b) MobileNet[27]



(d) Mobilenet V2

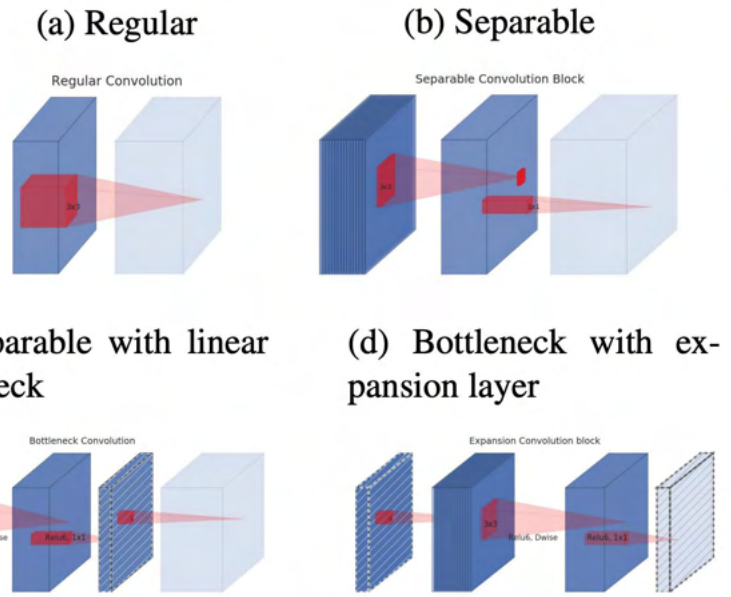


Figure 2: Evolution of separable convolution blocks. The diagonally hatched texture indicates layers that do not contain non-linearities. The last (lightly colored) layer indicates the beginning of the next block. Note: 2d and 2c are equivalent blocks when stacked. Best viewed in color.

MobileNet против MobileNetV2

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

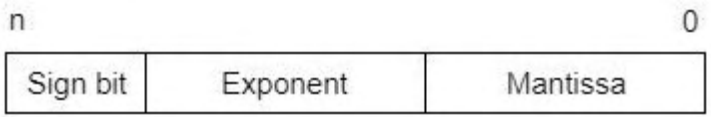
Table 2: MobileNetV2 : Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated n times. All layers in the same sequence have the same number c of output channels. The first layer of each sequence has a stride s and all others use stride 1. All spatial convolutions use 3×3 kernels. The expansion factor t is always applied to the input size as described in Table 1.

Квантование нейронных сетей

- Dynamic quantization
- Static Quantization
- Post-training quantization

https://pytorch.org/docs/stable/quantization.html

Арифметика чисел с плавающей точкой



Number Representation

		Range	Accuracy
FP32	1 (S) 8 (E) 23 (M)	10^{-38} - 10^{38}	.000006%
FP16	1 (S) 5 (E) 10 (M)	$6x10^{-5}$ - $6x10^{4}$	.05%
Int32	1 (S) 31 (M)	0 – $2x10^{9}$	½
Int16	1 (S) 15 (M)	0 – $6x10^{4}$	½
Int8	1 (S) 7 (M)	0 – 127	½

A number is represented by the formula `(-1)**s (1+m)*(2**(e-bias))`, where `s` is the sign bit, `m` is the mantissa, `e` is the exponent value, and `bias` is the bias number.

https://heartbeat.fritz.ai/quantization-arithmetic-421e66afd842

Динамическая квантизация

- Перевели веса сети в int8, активации переводим в int8 на лету во время forward pass
- Работает для nn.Linear, LSTM, GRU
- Не работает для свёрток

Diagram:

```
# original model
# all tensors and computations are in floating point
previous_layer_fp32 -- linear_fp32 -- activation_fp32 -- next_layer_fp32
                    /
linear_weight_fp32

# dynamically quantized model
# linear and LSTM weights are in int8
previous_layer_fp32 -- linear_int8_w_fp32_inp -- activation_fp32 -- next_layer_fp32
                     /
   linear_weight_int8
```

https://github.com/pytorch/tutorials/issues/899

Статическая квантизация

- Переводим веса и активации сети в int8 с помощью линейного преобразования. Веса линейного преобразования оцениваем на небольшом датасете

Diagram:

```
# original model
# all tensors and computations are in floating point
previous_layer_fp32 -- linear_fp32 -- activation_fp32 -- next_layer_fp32
                    /
linear_weight_fp32

# dynamically quantized model
# linear and LSTM weights are in int8
previous_layer_fp32 -- linear_int8_w_fp32_inp -- activation_fp32 -- next_layer_fp32
                     /
   linear_weight_int8
```

https://pytorch.org/docs/stable/quantization.html

Quantization aware training

Diagram:

```
# original model
# all tensors and computations are in floating point
previous_layer_fp32 -- linear_fp32 -- activation_fp32 -- next_layer_fp32
                      /
    linear_weight_fp32

# model with fake_quants for modeling quantization numerics during training
previous_layer_fp32 -- fq -- linear_fp32 -- activation_fp32 -- fq -- next_layer_fp32
                           /
   linear_weight_fp32 -- fq

# quantized model
# weights and activations are in int8
previous_layer_int8 -- linear_with_activation_int8 -- next_layer_int8
                      /
   linear_weight_int8
```

https://pytorch.org/docs/stable/quantization.html

Квантизация

WORKFLOWS

	Quantization	Dataset Requirements	Works Best For	Accuracy	Notes
Dynamic Quantization	weights only (both fp16 and int8)	None	LSTMs, MLPs, Transformers	good	Suitable for dynamic models (LSTMs), Close to static post training quant when performance is compute bound or memory bound due to weights.
Static Post Training Quantization	weights and activations (8 bit)	calibration	CNNs	good	Suitable for static models, provides best perf
Static Quantization-Aware Training	weights and activations (8 bit)	fine-tuning	CNNs	best	Requires fine tuning of model, currently supported only for static quantization.

https://www.youtube.com/watch?v=c3MT2qV5f9w

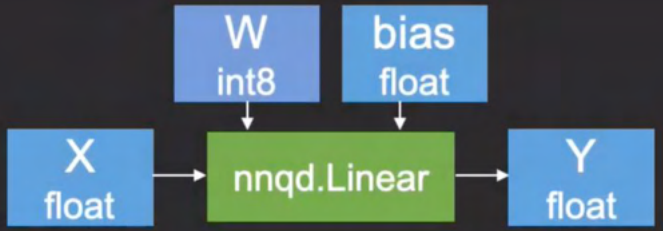
WORKFLOW:
POST TRAINING DYNAMIC QUANTIZATION

How: **tweak model**, one line API

What: quantize weights once, activations at runtime

Good for: LSTMs/Transformers and MLPs with small batch size

Savings: 2x faster compute, 4x less memory




```
# load or train your model
model = WordLanguageModel()
model.load_state_dict(torch.load("model.pt"))

# quantize
qmodel = quantize_dynamic(model,
                          dtype=torch.quint8)

# use or deploy for C++ inference
output = qmodel(input)
torch.jit.script(qmodel).save("scripted.pt")
```

https://www.youtube.com/watch?v=c3MT2qV5f9w

WORKFLOW:
POST TRAINING STATIC QUANTIZATION
2. PREPARE AND CALIBRATE

How:

1. Specify which parts of the model need to be quantized
2. Specify how to collect statistics (Observers)

```
# load or train your model
model = ResNet50()
model.load_state_dict(torch.load("model.pt"))

# tweak model for best results
# change code directly or use manipulation APIs
model.eval()
model = quantization.fuse_modules(model,
  [["conv1", "bn1", "relu1"]])

# specify which part to quantize and how
model.qconfig =
quantization.get_default_qconfig('fbgemm')
# configurable!

qmodel = quantization.prepare(model,
inplace=False)

# collect calibration statistics
qmodel.eval()
for batch, target in data_loader:
        model(batch)

print(model.conv1)

  ConvReLU2d(3, 64, kernel_size=(7, 7), ...
    (observer): MinMaxObserver(
      min_val=0.0, max_val=4.55)
  )
```

WORKFLOW:
POST TRAINING STATIC QUANTIZATION
3. CONVERT

How: call torch.quantization.convert()

What:

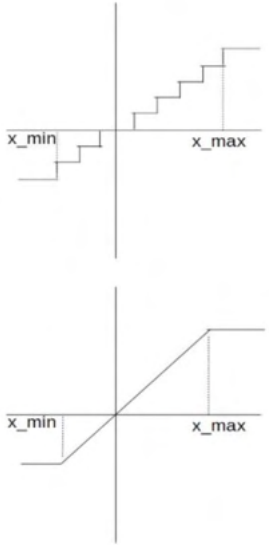
Converts operations from fp32 to int8 arithmetic

```
# load or train your model
model = ResNet50()
model.load_state_dict(torch.load("model.pt"))

# tweak model for best results
# change code directly or use manipulation APIs
model.eval()
model = quantization.fuse_modules(model,
  [["conv1", "bn1", "relu1"]])

# specify which part to quantize and how
model.qconfig =
quantization.get_default_qconfig('fbgemm')
# configurable!

# collect calibration statistics
qmodel.eval()
for batch, target in data_loader:
        model(batch)

# get the quantized model
quantization.convert(qmodel)

print(qmodel.conv1)

  QuantizedConvReLU2d(3, 64,
    scale=0.035, zero_point=0,
    kernel_size=(7, 7), ...)
```

QUANTIZATION AWARE TRAINING

- Emulate quantization by quantizing and de-quantizing in succession
  - Values are still in floating point, but with reduced precision

- $x\_out = FakeQuant(x)$
  $= DeQuant(Quant(x))$
  $= s * (Clamp(round(x/s) - z) + z)$



Fake Quantizer (top), showing the quantization of output values. Approximation for purposes of derivative calculation (bottom).

Квантизация

EXAMPLE MODELS

	fp32 accuracy	int8 accuracy change	Technique	CPU inference speed up
ResNet50	76.1 Imagenet	-0.2 75.9	Post Training	2x 214ms →102ms, Intel Skylake-DE
MobileNetV2	71.9 Imagenet	-0.4 71.5	Quantization-Aware Training	4x 75ms →18ms OnePlus 5, Snapdragon 835
BERT	90.2 F1 (GLUE MRPC)	0.0 90.2	Dynamic Quantization with per-channel quantization	1.6x 581ms →313ms, Intel Skylake-DE, Batch size=1

https://www.youtube.com/watch?v=c3MT2qV5f9w

QUANTIZING MOBILENETS

	fp32 accuracy	int8 accuracy change	Technique
MobileNetV2	71.9 Imagenet	-6.3 65.6	Post Training: Per Tensor quantization
MobileNetV2	71.9 Imagenet	-4.8 67.1	Post-Training: Per-channel quantization
MobileNetV2	71.9 Imagenet	-0.4 71.5	Quantization-Aware Training

https://www.youtube.com/watch?v=c3MT2qV5f9w

Деплой нейронных сетей

- Pytorch JIT
  - Torchscript
  - JIT compiler
- ONNX и ONNX Runtime
- Pytorch mobile vs TF lite
- CoreML

Pytorch JIT

- Позволяет гонять в продакшне сети без использования python

https://www.youtube.com/watch?v=St3gdHJzic0

Pytorch JIT

- Позволяет гонять в продакшне сети без использования python
- Состоит из двух частей:
  - Torchscript - сабсет языка Python
  - Jit compiler - помогает оптимизировать инференс

https://www.youtube.com/watch?v=St3gdHJzic0

TORCHSCRIPT

A static, high-performance subset of Python.

1. Prototype your model with PyTorch
2. Control flow is preserved
3. First-class support for lists, dicts, etc.

```python
import torch
class MyModule(torch.nn.Module):
    def __init__(self, N, M, state: List[Tensor]):
        super(MyModule, self).__init__()
        self.weight = torch.nn.Parameter(torch.rand(N, M))
        self.state = state

    def forward(self, input):
        self.state.append(input)
        if input.sum() > 0:
            output = self.weight.mv(input)
        else:
            output = self.weight + input
        return output

# Compile the model code to a static representation
my_module = MyModule(3, 4, [torch.rand(3, 4)])
my_script_module = torch.jit.script(my_module)

# Save the compiled code and model data
# so it can be loaded elsewhere
my_script_module.save("my_script_module.pt")
```

PYTORCH JIT

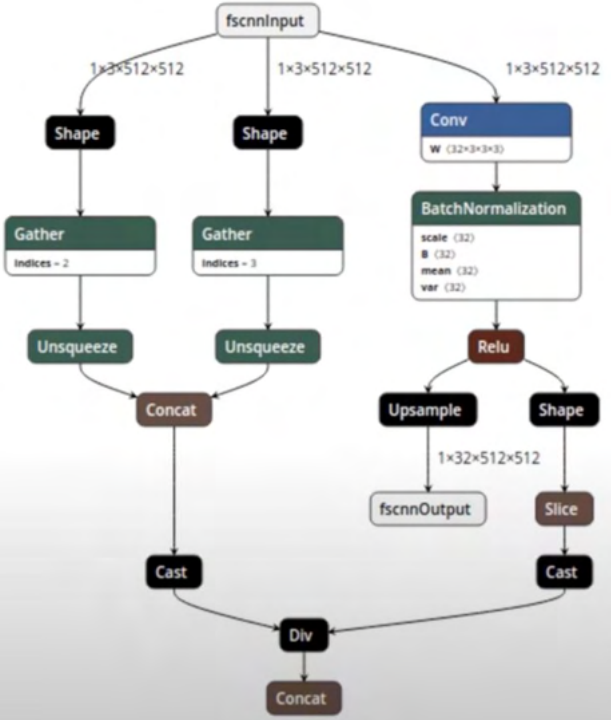
An optimizing just-in-time compiler
for PyTorch programs.

1. Lightweight, thread-safe interpreter
2. Easy to write custom transformations
3. Not just for inference! Autodiff support.

```
graph(%self : ClassType<MyModule>,
      %input.1 : Tensor):
  %16 : int = prim::Constant[value=1]()
  %6 : None = prim::Constant()
  %8 : int = prim::Constant[value=0]()
  %2 : Tensor[] = prim::GetAttr[name="state"](%self)
  %4 : Tensor[] = aten::append(%2, %input.1)
  %7 : Tensor = aten::sum(%input.1, %6)
  %9 : Tensor = aten::gt(%7, %8)
  %10 : bool = aten::Bool(%9)
  %output : Tensor = prim::If(%10)
    block0():
      %11 : Tensor = prim::GetAttr[name="weight"](%self)
      %output.1 : Tensor = aten::mv(%11, %input.1)
      -> (%output.1)
    block1():
      %14 : Tensor = prim::GetAttr[name="weight"](%self)
      %output.2 : Tensor = aten::add(%14, %input.1, %16)
      -> (%output.2)
  return (%output)
```

ONNX File Format

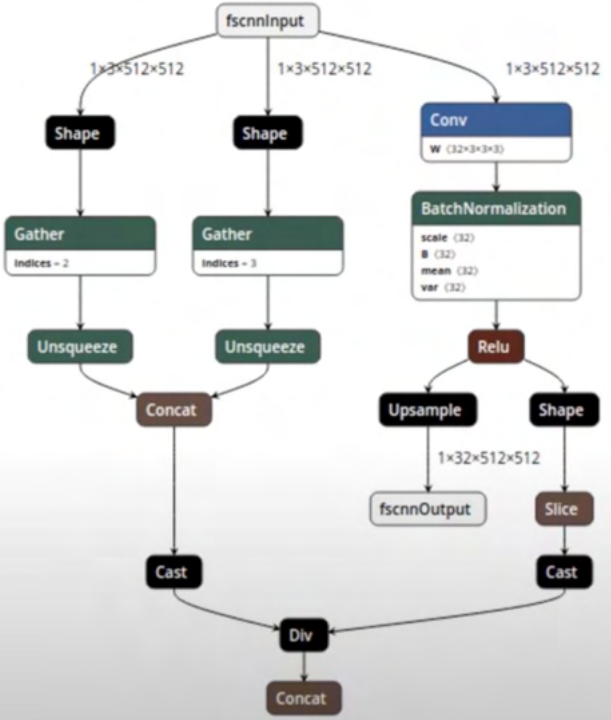
- **Model**
  - Version info
  - Metadata
  - Acyclic computation dataflow graph
- **Graph**
  - Inputs and outputs
  - List of computation nodes
  - Graph name
- **Computation Node**
  - Zero or more inputs of defined types
  - One or more outputs of defined types
  - Operator
  - Operator parameters



https://www.youtube.com/watch?v=cK5AyawZSUI

ONNX File Format

- **Model**
  - Version info
  - Metadata
  - Acyclic computation dataflow graph
- **Graph**
  - Inputs and outputs
  - List of computation nodes
  - Graph name
- **Computation Node**
  - Zero or more inputs of defined types
  - One or more outputs of defined types
  - Operator
  - Operator parameters



https://www.youtube.com/watch?v=cK5AyawZSUI