

# DEEP LEARNING

## Неделя 5

---

Святослав Елизаров, Борис Коваленко, Артем Грачев

28 октября 2017

Высшая школа экономики

# СВЁРТОЧНЫЕ СЕТИ

---

# КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ

Рассмотрим задачу классификации изображений.



# КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ

Каждое изображение представлено тремя матрицами, по одной на каждый цветовой канал (RGB):



(a) Red

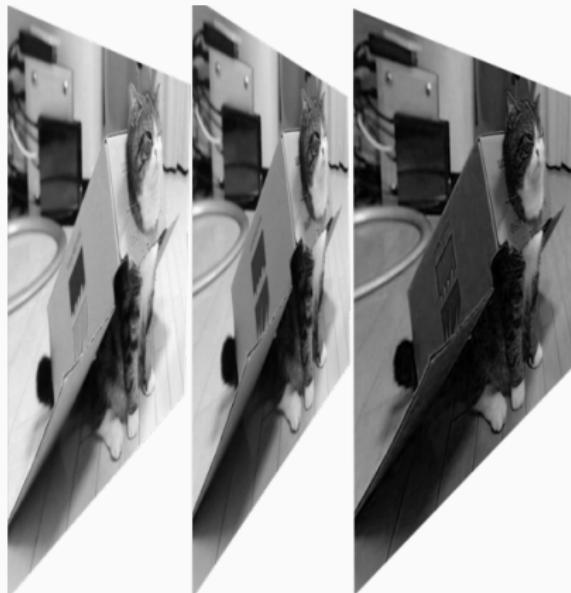
(b) Green

(c) Blue

Элементы матриц принимают значения от 0 до 255 и обозначают интенсивность одного из трёх цветов.

# КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ

Для удобства мы будем хранить все три матрицы вместе в одном трёхмерном массиве, где третий индекс отвечает за номер канала. Такая структура является частным случаем **тензора**.



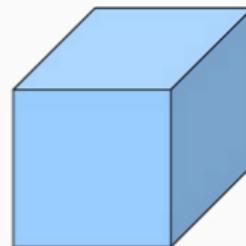
# КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ



1d-tensor



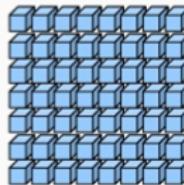
2d-tensor



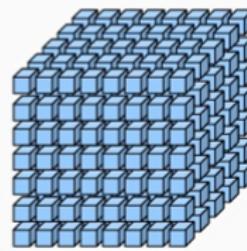
3d-tensor



4d-tensor



5d-tensor

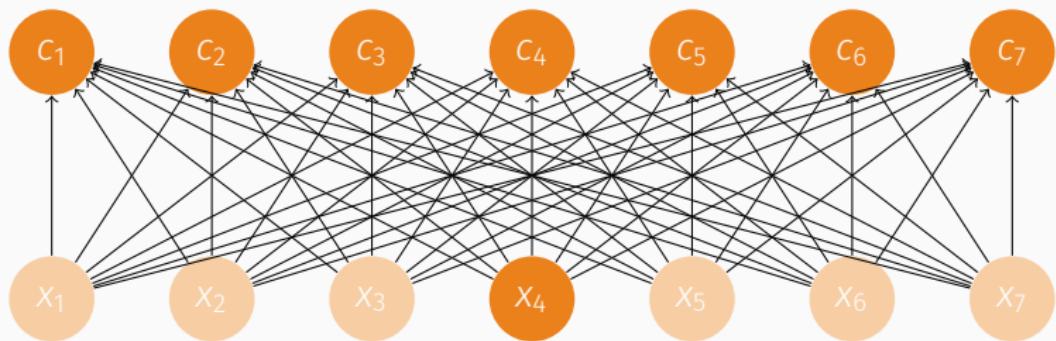


6d-tensor

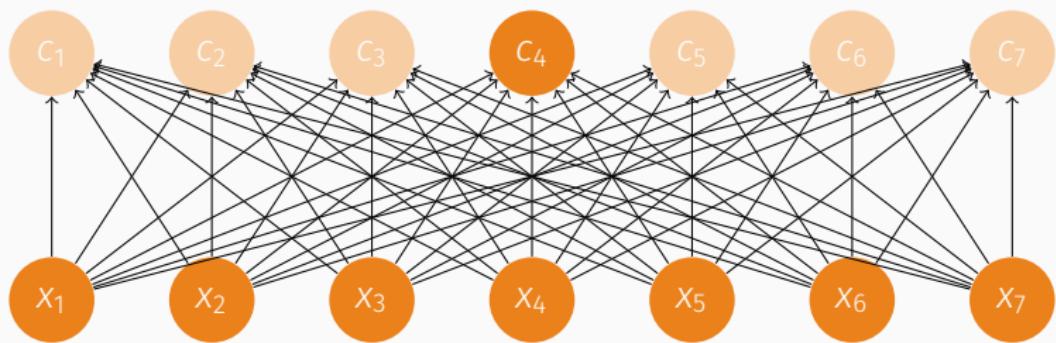
# КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ

- Как нам работать с такими данными?
- Будет ли эффективна полносвязная сеть?
- Почему?

# ПОЛНОСВЯЗНЫЙ СЛОЙ



# ПОЛНОСВЯЗНЫЙ СЛОЙ



# КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ

- Каждый вход влияет на каждый узел слоя и наоборот
- Зависит от положения объектов в кадре
- Большое количество параметров. Например для изображения  $800 \times 600$  только одно измерение матрицы весов составит 480000

# СВЁРТКИ

$$(f * K)(x) = \sum_{\tau} f(\tau)K(x - \tau) = \sum_{\tau} f(x - \tau)K(\tau)$$

Где  $f$  называется *оригиналом*, а  $K$  **ядром** свёртки. Можно сказать, что ядро присваивает вес каждому значению функции.

# СВЁРТКИ

Пример:

- пусть  $f(i)$  возвращает значения функции потерь на  $i$ -м батче при тренировке нейронной сети
- Из-за шума график из таких значений выглядит не очень красиво
- Необходимо применить сглаживание

# СВЁРТКИ

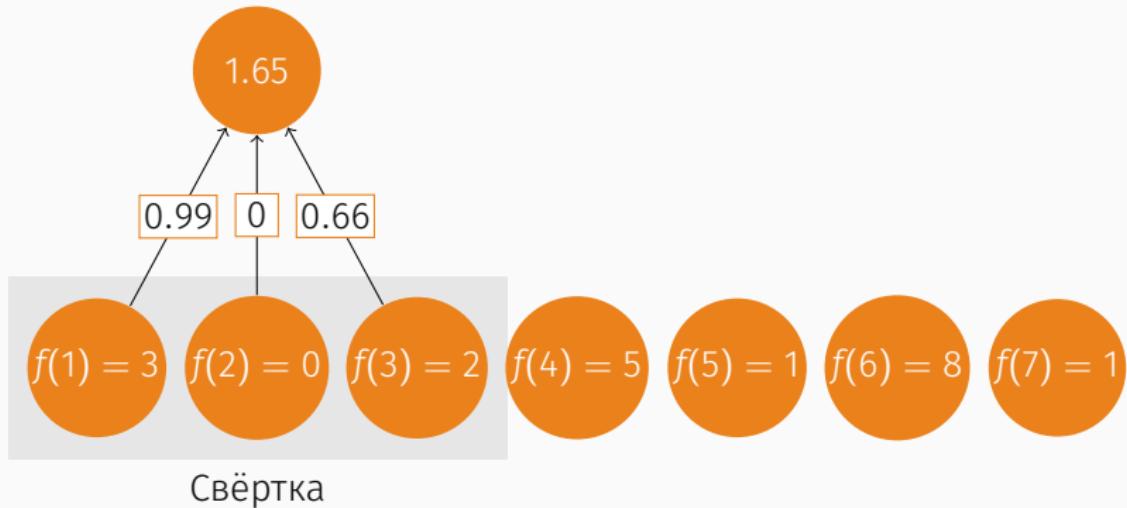
Пример:

- Ядро  $K(i)$  возвращает значения 0.33 при значениях  $i \in \{-1, 0, 1\}$
- И 0 во всех остальных случаях

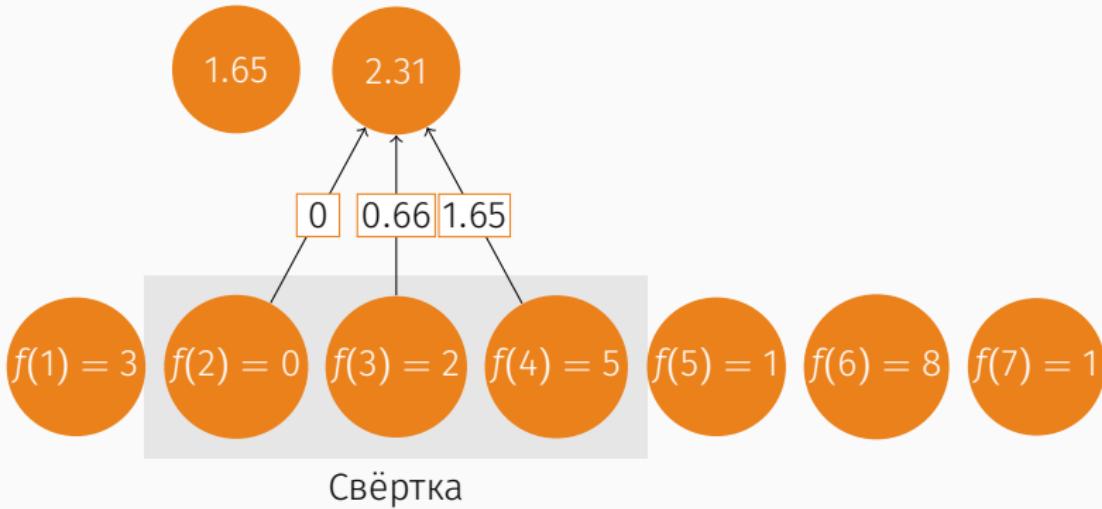
Таким образом можно представить операцию свёртки, как скользящее окно длины 3, на значениях функции оригинала  $f$ .

Визуализируем это:

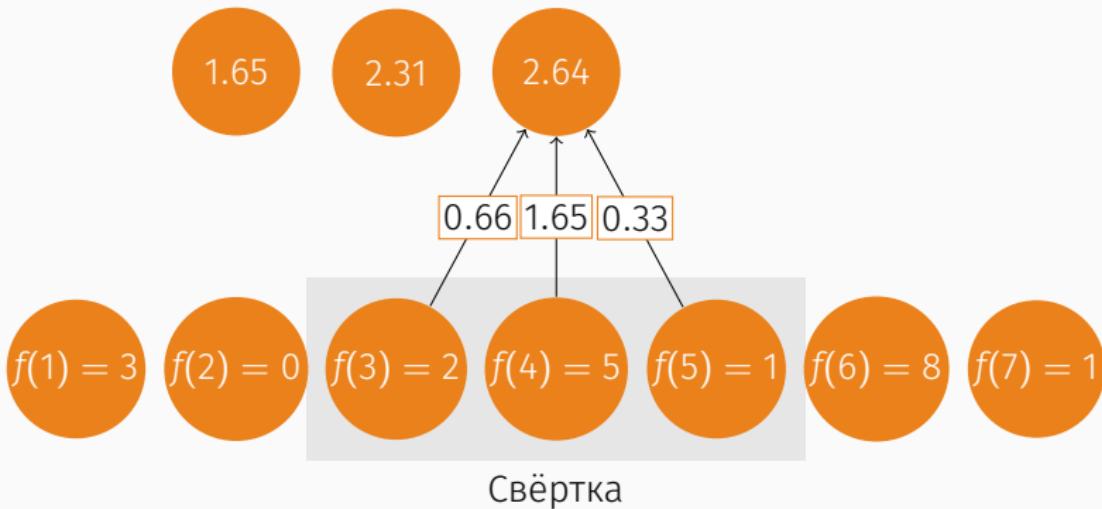
# СВЁРТКИ



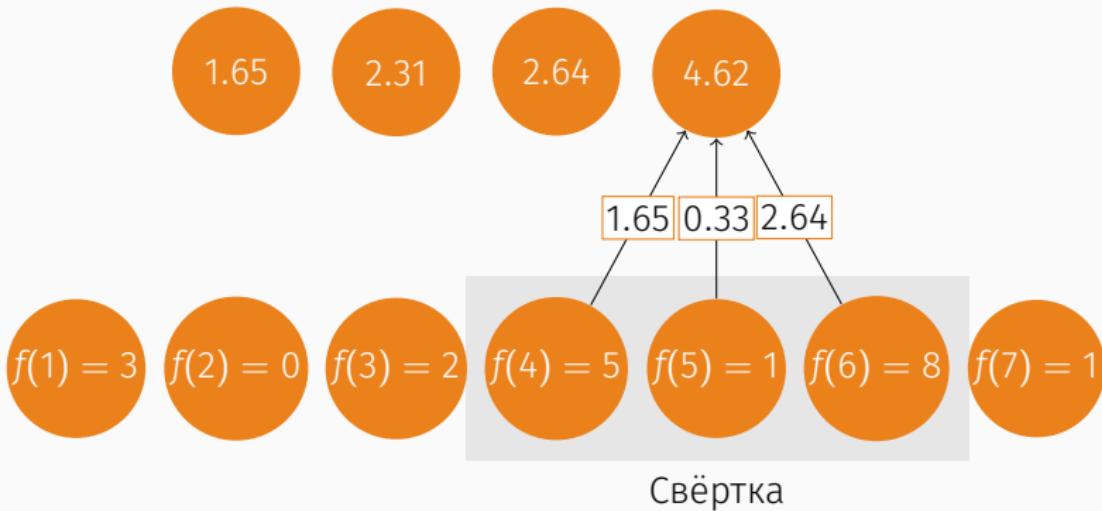
# СВЁРТКИ



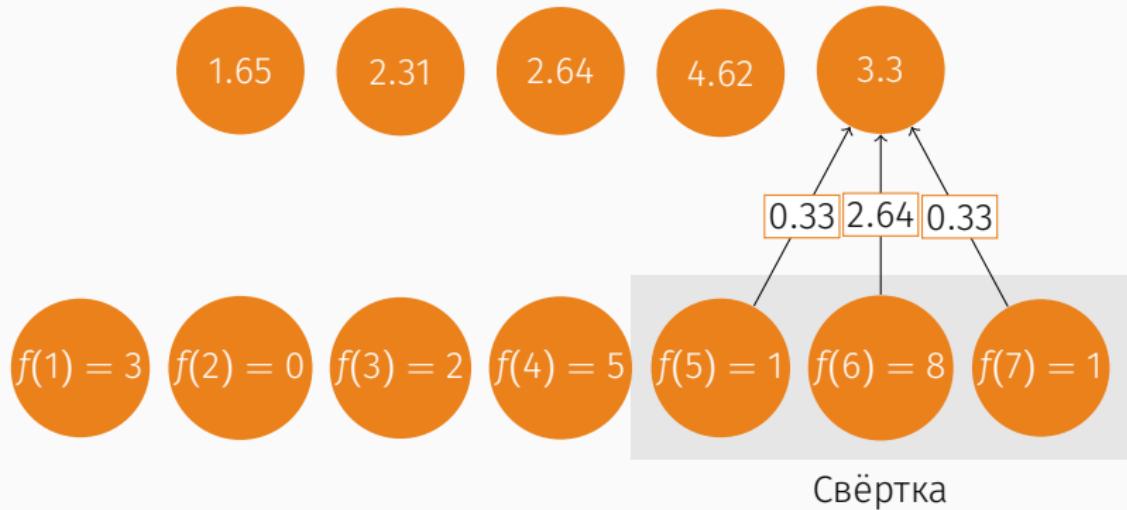
# СВЁРТКИ



# СВЁРТКИ



# СВЁРТКИ



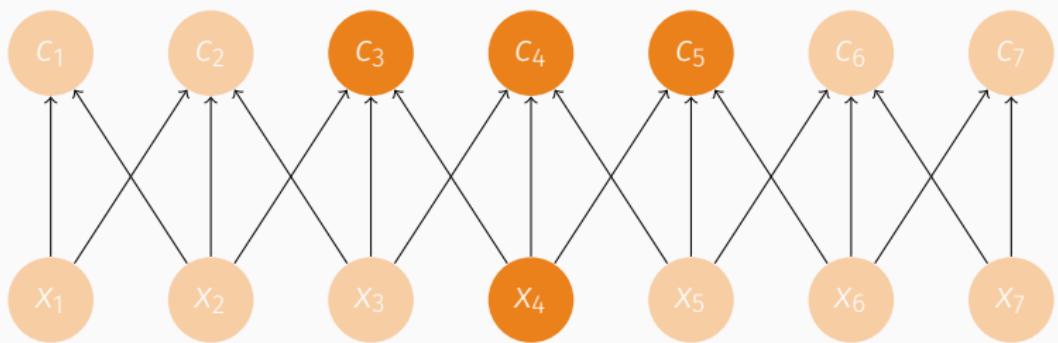
# СВЁРТКИ

- Ядро в данном случае задаёт обычную линейную комбинацию и полностью определено вектором  $(0.33, 0.33, 0.33)$
- Среди значений свёртки не хватает двух элементов. Это вызвано тем, что окно не выходит за пределы имеющегося ряда. Такой тип свёрток обычно называют **valid convolution**
- Так же часто используются **same convolution**. В этом случае "недостающие" элементы заполняются нулями, и результирующий ряд имеет такое же количество элементов.
- В данном примере использовался единичный шаг окна, однако этот параметр может меняться. Шаг обычно обозначается **stride**.

# СВЁРТКИ

Так зачем это нужно? Чем это лучше полносвязного слоя?

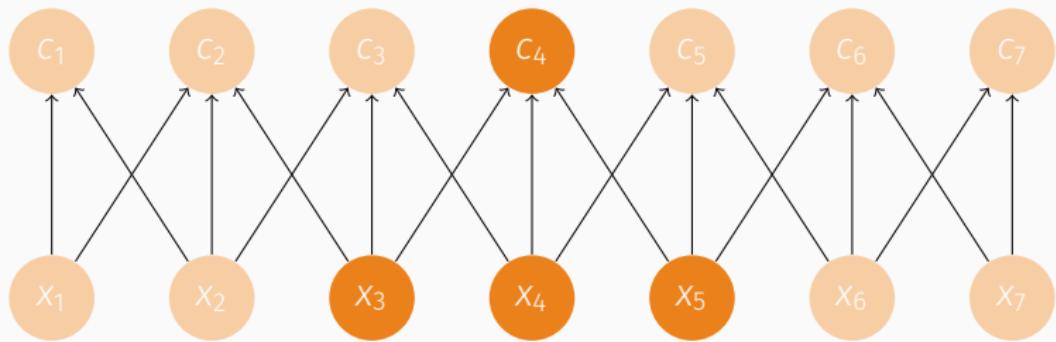
# ОБЩИЕ ВЕСА



## ОБЩИЕ ВЕСА

- Каждый вход влияет только на три узла (зависит от размера ядра свёртки)
- Притом веса используются повторно. Например, вес "соединяющий"  $x_4$  и  $x_5$  равен весу связи  $x_6$  и  $x_7$ .
- Свёртки действуют локально

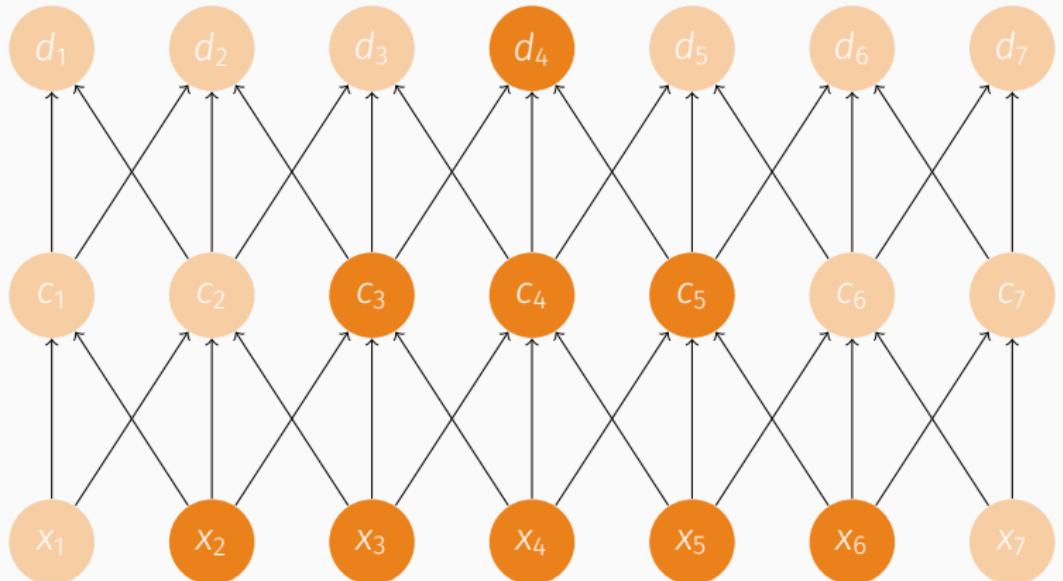
# ЗОНА ВИДИМОСТИ



## ЗОНА ВИДИМОСТИ

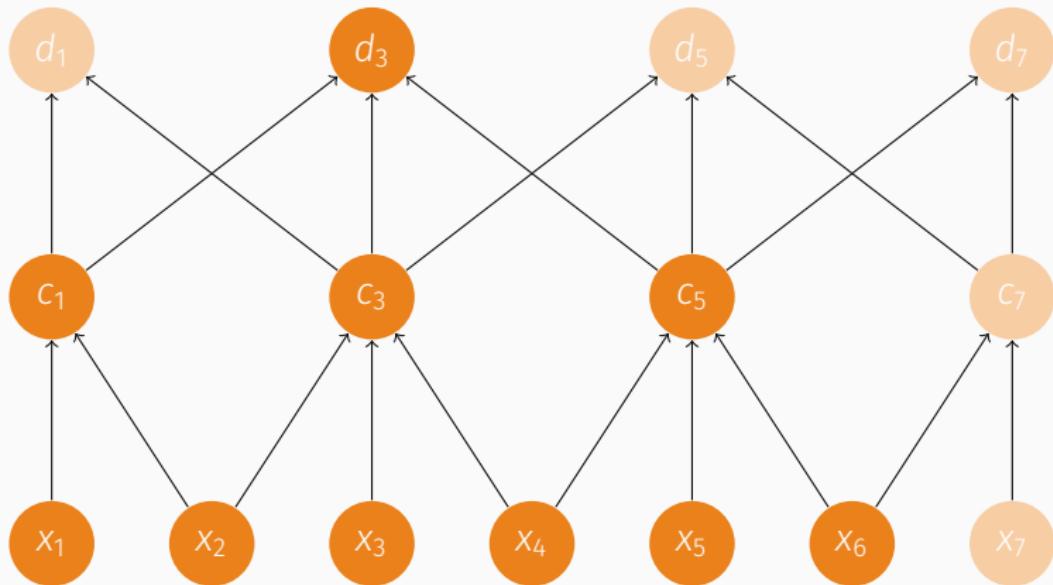
- Каждый узел изменяется под действием только трёх входов (зависит от размера ядра свёртки)
- Входы  $x_3, x_4, x_5$  называются **зоной видимости или рецептивным полем** (receptive field) узла  $c_4$
- Зона видимости зависит от глубины

# ЗОНА ВИДИМОСТИ



# ЗОНА ВИДИМОСТИ

Зона видимости так же зависит от размера шага. Чем больше шаг, тем больше зона.



# СВЁРТКИ

Свёртки легко обобщить на двухмерный ( $n$ -мерный) случай:

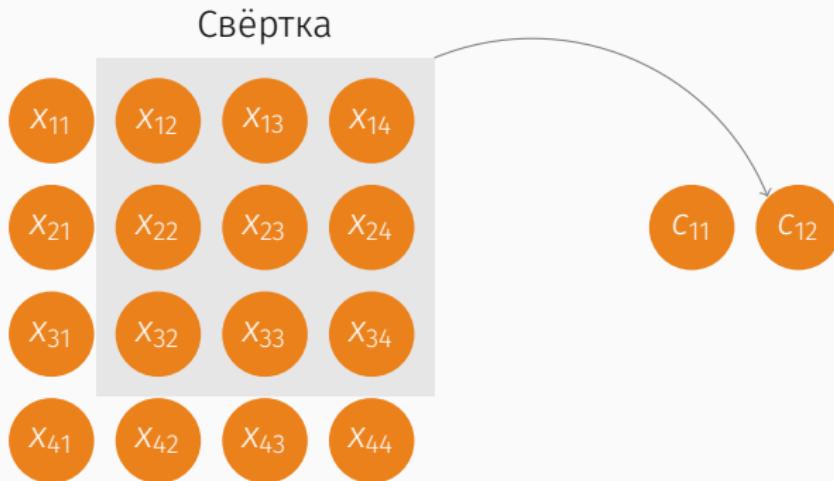
$$(f * K)(x, y) = \sum_{\tau, \eta} f(\tau, \eta)K(x - \tau, y - \eta) = \sum_{\tau, \eta} f(x - \tau, y - \eta)K(\tau, \eta)$$

Теперь ядро свёртки задаётся матрицей ( $n$ -мерным тензором).  
Рассмотрим пример.

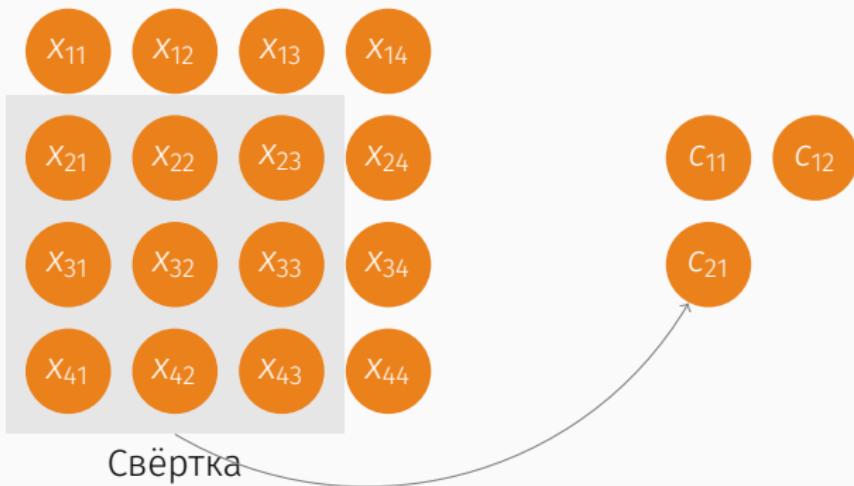
# СВЁРТКИ



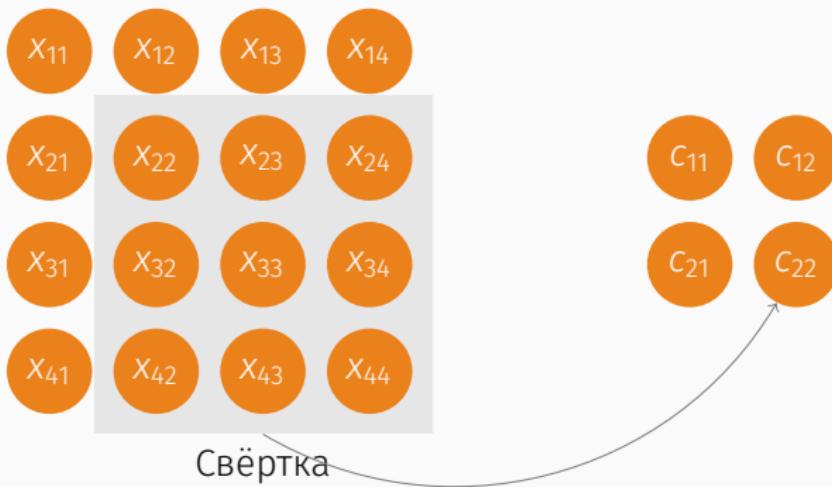
# СВЁРТКИ



# СВЁРТКИ



# СВЁРТКИ



# СВЁРТКИ

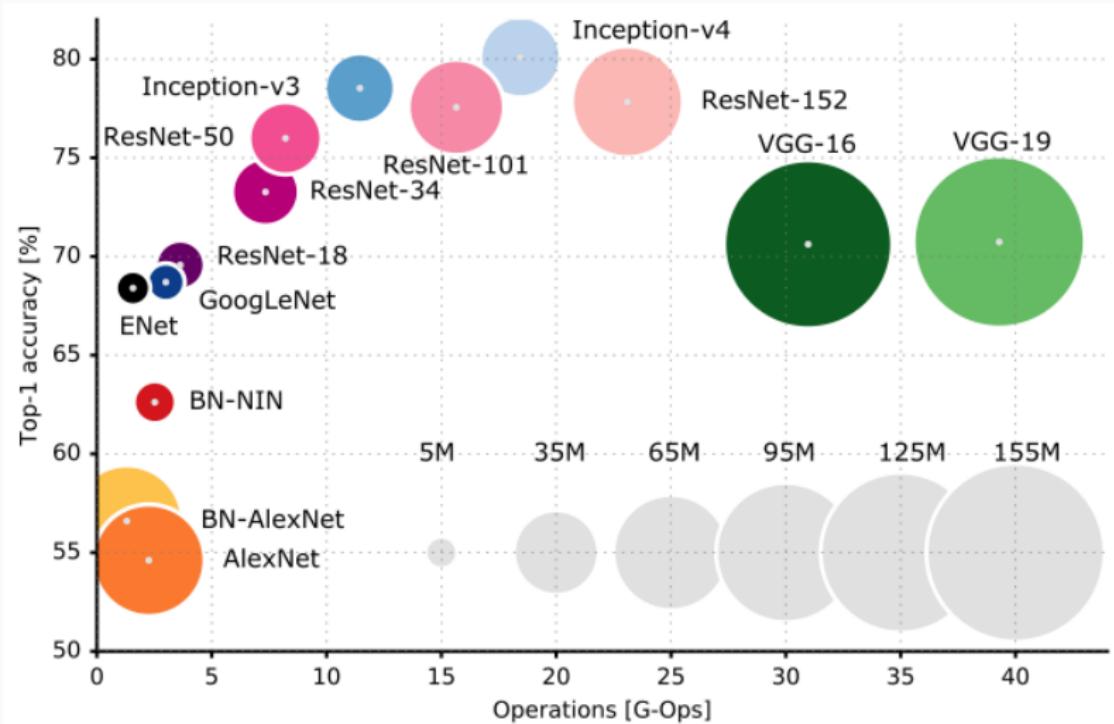
Как рассчитать размер результирующего слоя?

$$width_{result} = (width_{input} - width_{kernel} + 2padding)/stride + 1$$

$$height_{result} = (height_{input} - height_{kernel} + 2padding)/stride + 1$$

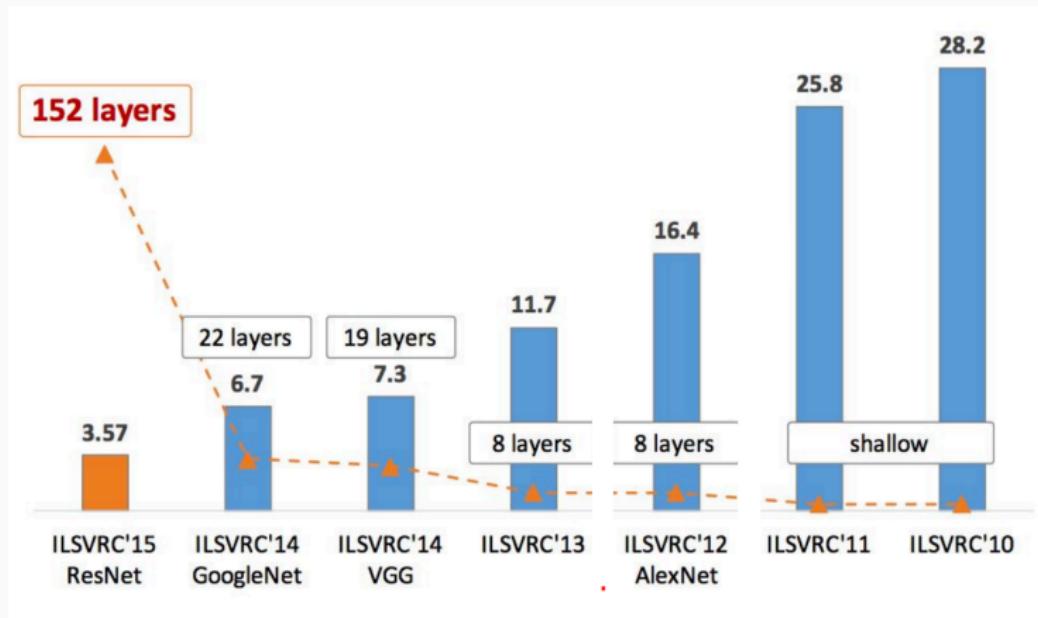
# ИСТОРИЧЕСКИЙ ОБЗОР

ImageNet



# ИСТОРИЧЕСКИЙ ОБЗОР

ImageNet



# ALEXNET

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)

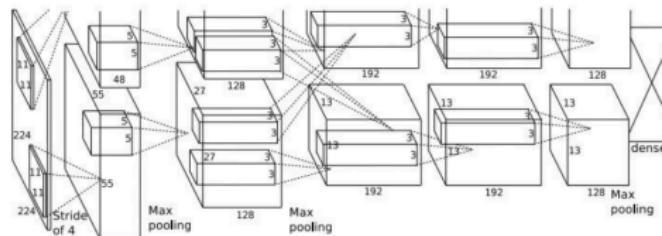
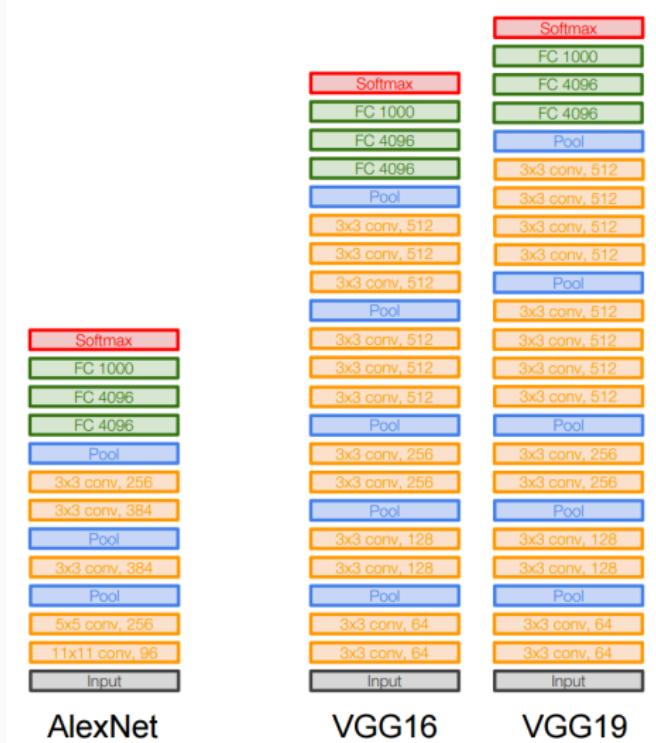


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced

# VGG



# ГЛУБИНА

- Во многих задачах предпочтительнее использовать более глубокую сеть. Например, чтобы иметь более широкое поле обзора.
- Теоретически доказано, что глубина сети влияет на обобщающую способность. [Eldan, Shamir, 2016 The Power of Depth for Feedforward Neural Networks] и [Matus Telgarsky. 2016 Benefits of depth in neural networks]
- Однако “наивный” способ добавления слоёв может не сработать. **Почему?**

# RESNET

- Если градиент исчезает, то давайте “пробросим” связь от входа до выхода (shortcut)
- Таким образом слои сети будут учить разницу между входом и выходом.

# RESNET

$$\psi(x, W) = \phi(x, W) + x$$

Где  $x$  – вход,  $W$  – тензор параметров,  $\phi$  – слой или несколько слоёв нейронной сети (например свёрточный слой).

Такой тип архитектуры называется *разностным* (residual network) или просто resnet

[He et al. 2015 Deep Residual Learning for Image Recognition]



Составной блок разностной сети

# RESNET

- Из таких блоков могут быть построены сети с сотнями (и даже тысячами) слоёв
- Теоретически обосновано, что наиболее эффективным является shortcut длины 2. [Le et al. 2017 Demystifying ResNet]

Для обучения очень глубоких resnet-сетей используется **метод стохастической глубины** (stochastic depth) описанный в работе [Huang et al. 2016 Deep Networks with Stochastic Depth]

Метод напоминает dropout, однако работает на уровне “блоков” сети. С заданной вероятностью произвольные resnet-блоки заменяются тождественной связью во время тренировки. Вывод, как и в случае с dropout, производится на полностью активированной сети.

# HIGHWAY NETWORKS

- Resnet является частным случаем Highway Networks
- Вход не просто прибавляется к выходу, а смешивается с ним в пропорции, которая обучается автоматически.

$$\psi(x, W_{layer}, W_\sigma) = \sigma(x, W_\sigma)\phi(x, W) + (1 - \sigma(x, W_\sigma))x$$

Где  $\sigma$  – функция, значения которой лежат от 0 до 1. Эта функция называется *воротами*.

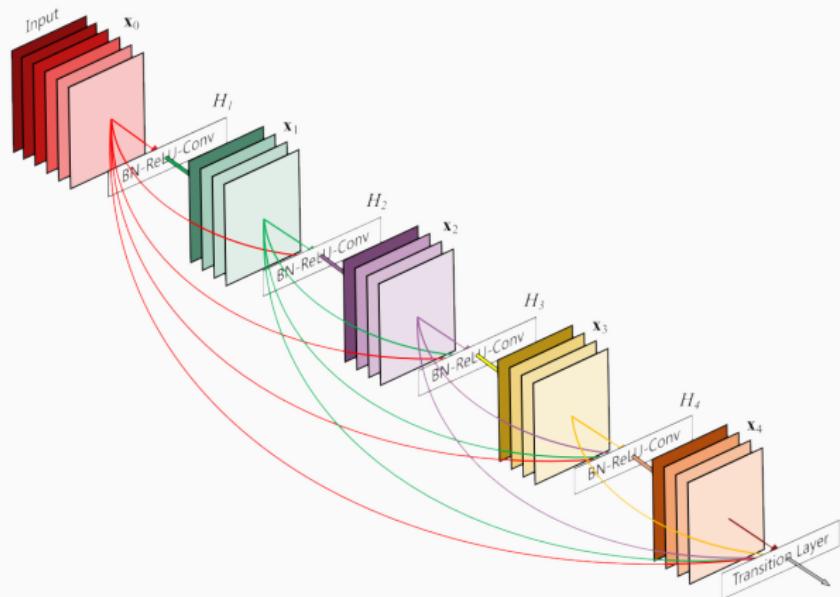
[Srivastava, Greff, Schmidhuber. 2015 Training Very Deep Networks]

## DENSE NETWORKS

- Другой интересной архитектурой для тренировки глубоких сетей являются **плотные сети** (dense networks).
- Основная идея заключается в том, что вход каждого слоя связывается со всеми выходами промежуточных слоёв
- В отличие от resnet в связи используется не суммирование, а конкатенация

[Huang, Liu, Weinberger. 2016 Densely Connected Convolutional Networks]

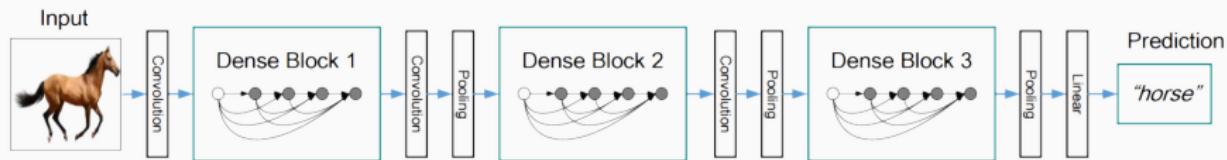
# DENSE NETWORKS



Составной блок плотной сети

- Градиент легко передаётся на нижние слои
- Каждый слой может использовать признаки, полученные на предыдущих, что может быть полезно
- Из-за того, что выходы приклеиваются ко входам ширина слоёв растёт.

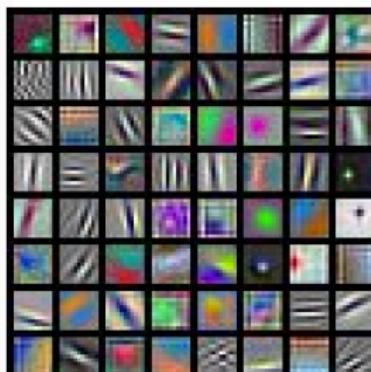
# DENSE NETWORKS



Чтобы сеть не расширялась не используют блоки глубины  
больше 5

# ВИЗУАЛИЗАЦИЯ

## Визуализация фильтров 1 слоя



AlexNet:  
 $64 \times 3 \times 11 \times 11$



ResNet-18:  
 $64 \times 3 \times 7 \times 7$



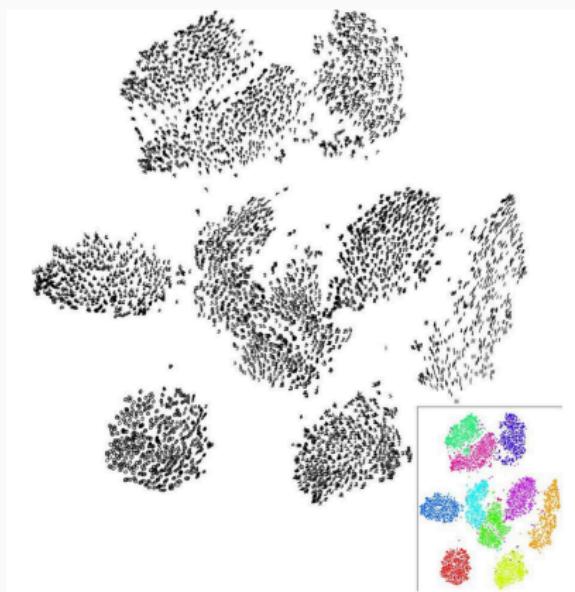
ResNet-101:  
 $64 \times 3 \times 7 \times 7$



DenseNet-121:  
 $64 \times 3 \times 7 \times 7$

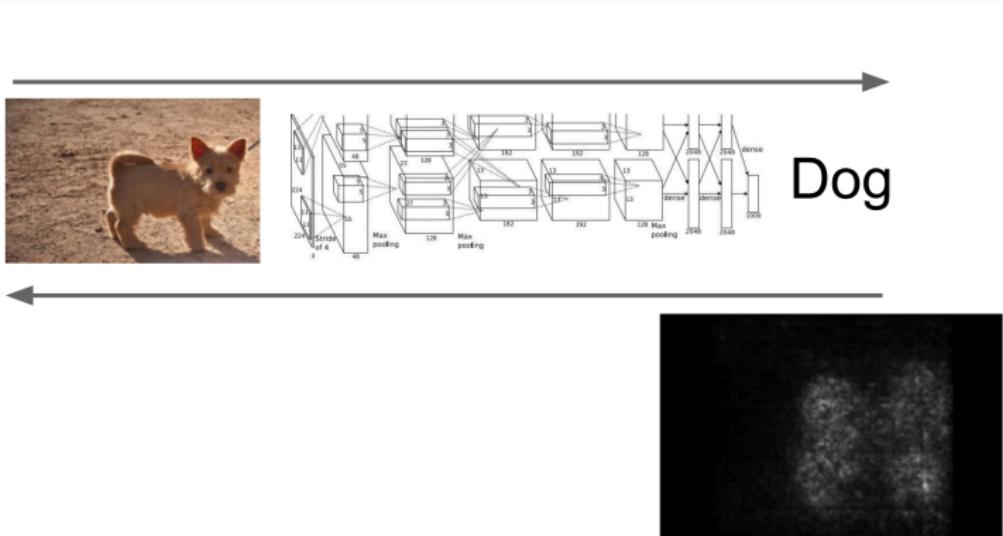
# ВИЗУАЛИЗАЦИЯ

Визуализация извлеченных признаков



# ВИЗУАЛИЗАЦИЯ

Saliency maps



# ВИЗУАЛИЗАЦИЯ

Saliency maps

