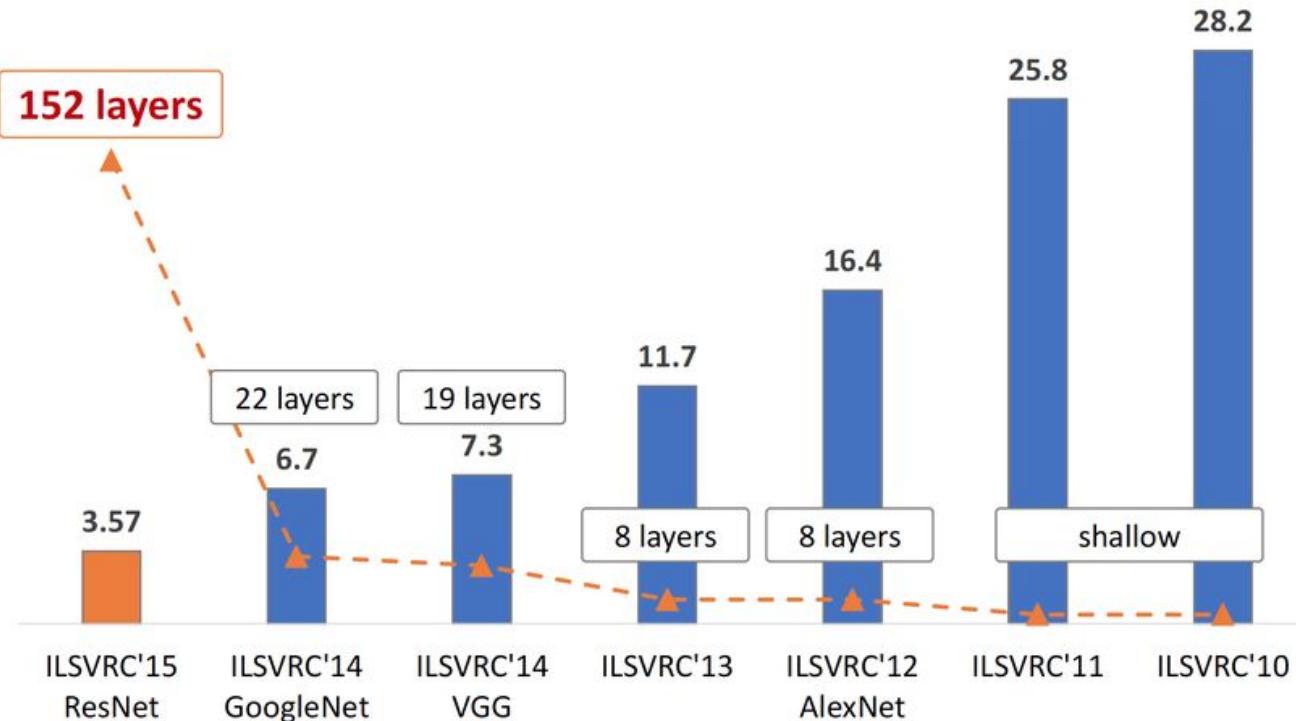


Компьютерное зрение

Лекция 7. Архитектуры CNN, аугментация, тюнинг

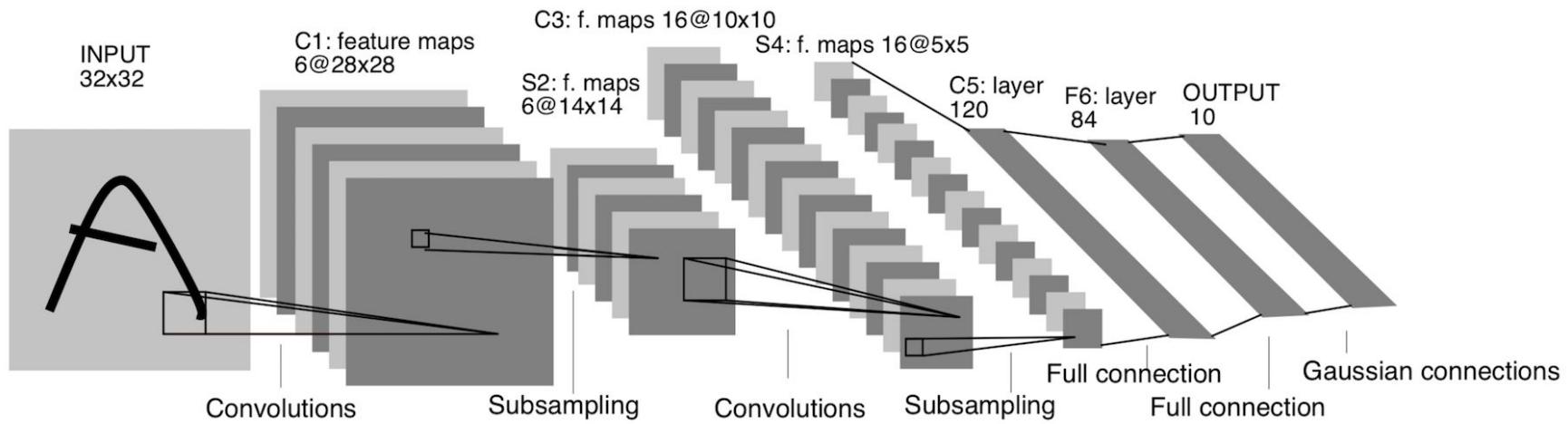
20.06.2020
Руслан Алиев

ImageNet



Архитектуры сверточных сетей

LeNet (1998)



AlexNet (2012)

Architecture:

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

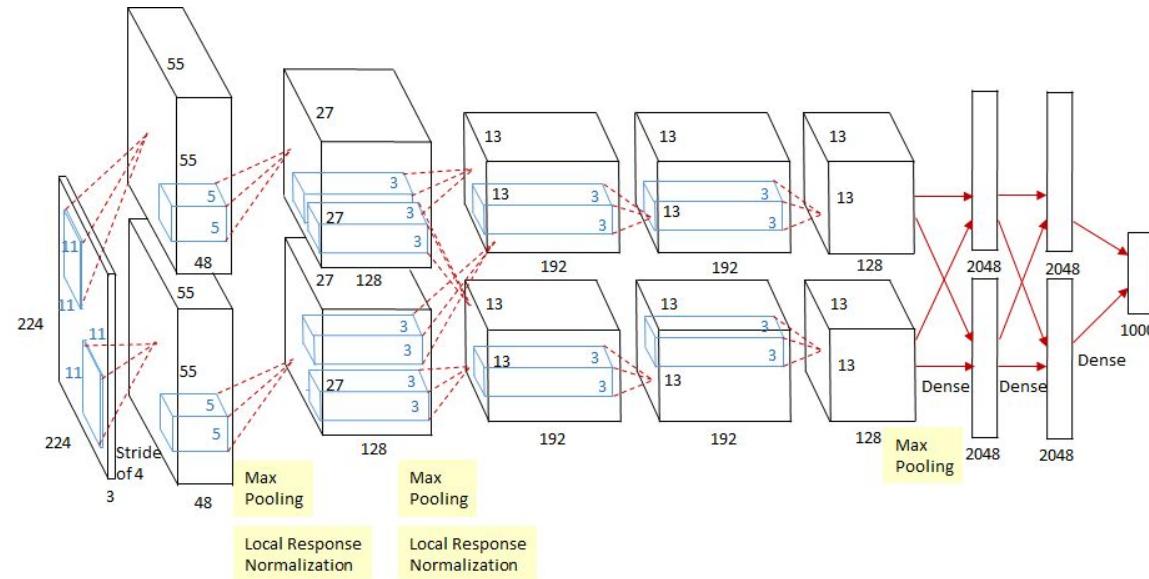
CONV5

Max POOL3

FC6

FC7

FC8



AlexNet (2012)

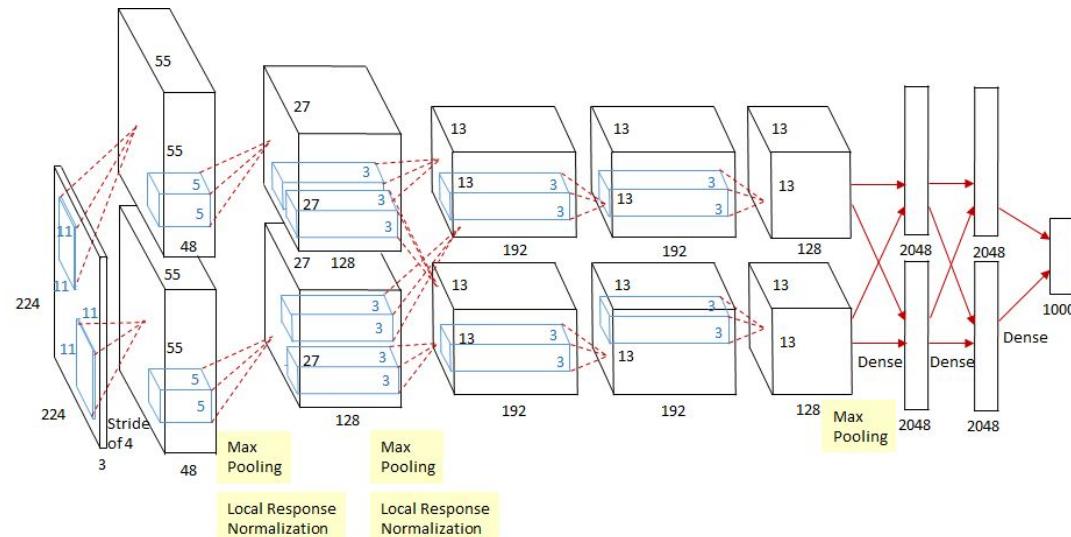
$$m = \left\lfloor \frac{n + 2p - f}{s} \right\rfloor + 1$$

Входной размер: 3x227x227

Первый слой:

96 11x11 фильтров, stride = 4

Какая размерность выхода?



AlexNet (2012)

$$m = \left\lfloor \frac{n + 2p - f}{s} \right\rfloor + 1$$

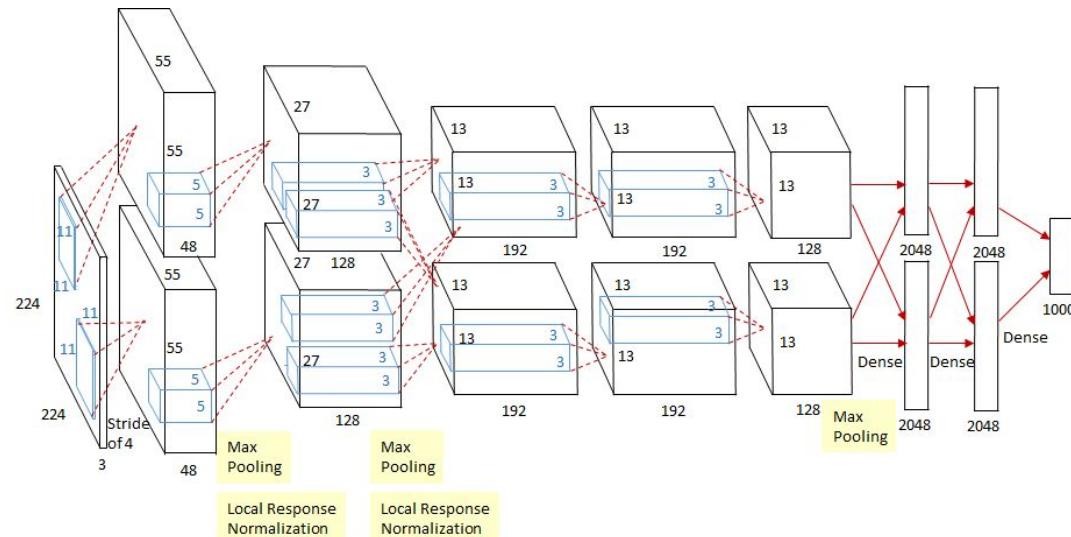
Входной размер: 3x227x227

Первый слой:

96 11x11 фильтров, stride = 4

Какая размерность выхода?

96x55x55



AlexNet (2012)

Входной размер: 3x227x227

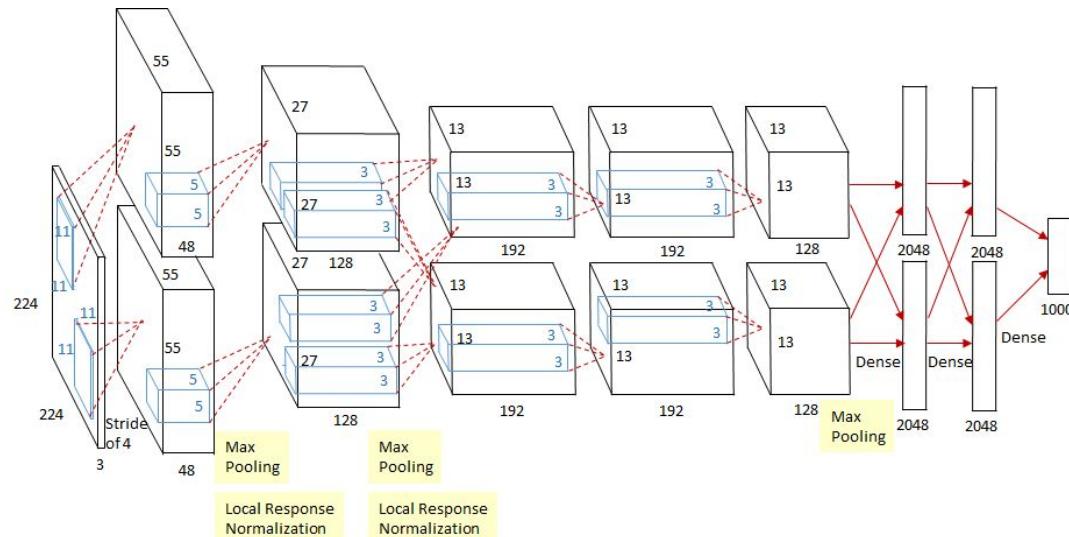
Первый слой:

96 11x11 фильтров, stride = 4

Какая размерность выхода?

96x55x55

Сколько параметров?



AlexNet (2012)

Входной размер: 3x227x227

Первый слой:

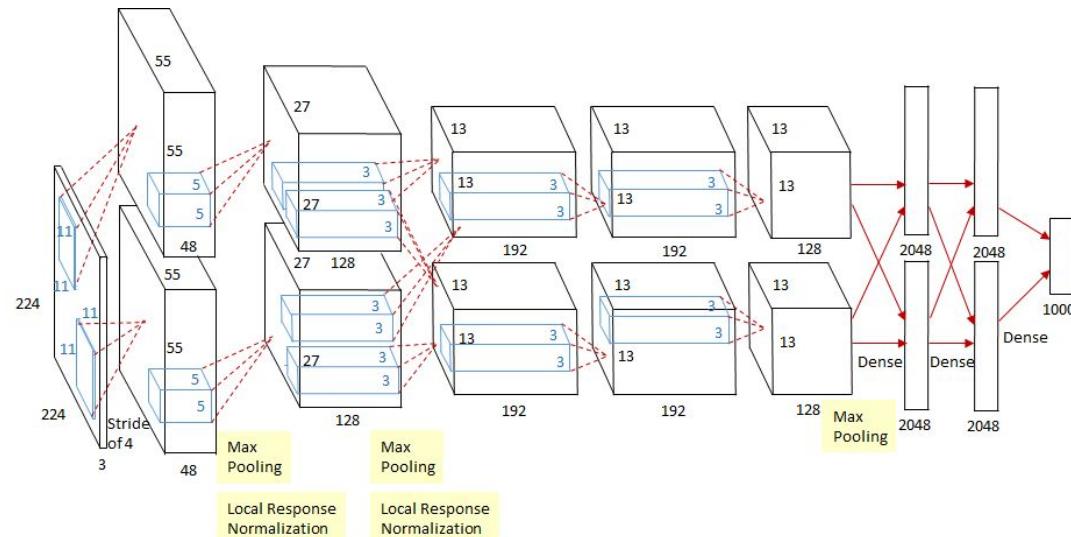
96 11x11 фильтров, stride = 4

Какая размерность выхода?

96x55x55

Сколько параметров?

$(11*11*3)*96=35\text{k}$



AlexNet (2012)

Входной размер: 3x227x227

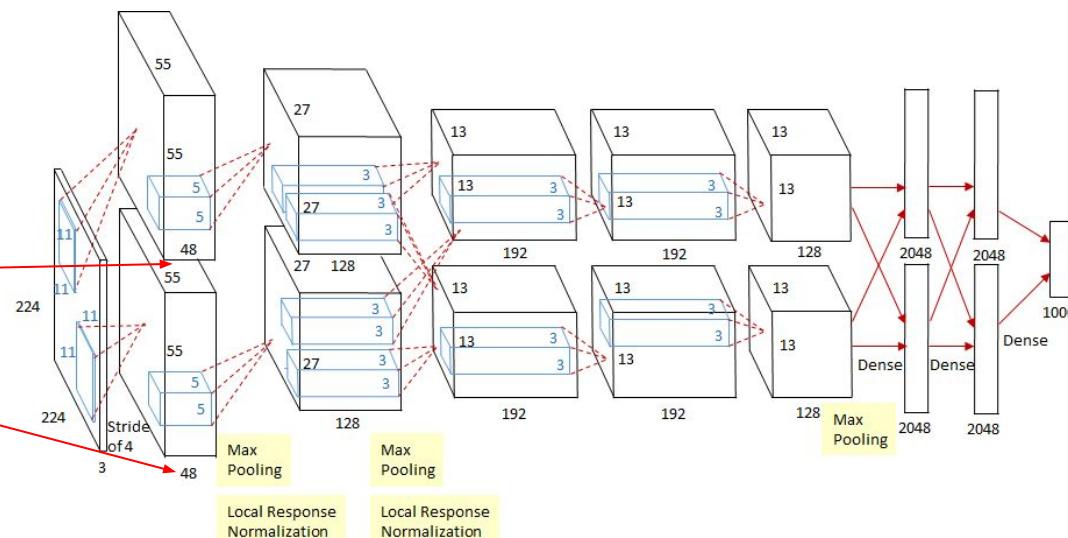
Первый слой:

96 11x11 фильтров, stride = 4

Какая размерность выхода?

96x55x55

Сколько параметров?
 $(11*11*3)*96=35\text{k}$



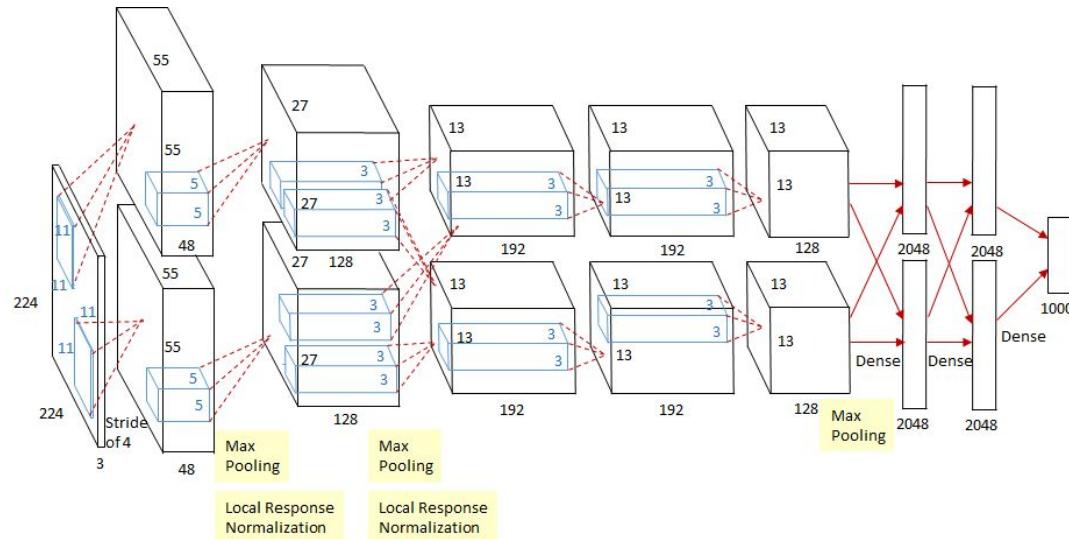
AlexNet (2012)

$$m = \left\lfloor \frac{n + 2p - f}{s} \right\rfloor + 1$$

Входной размер: 96x55x55

Второй слой:
пулинг 3x3 фильтры, stride=2

Какая размерность выхода?



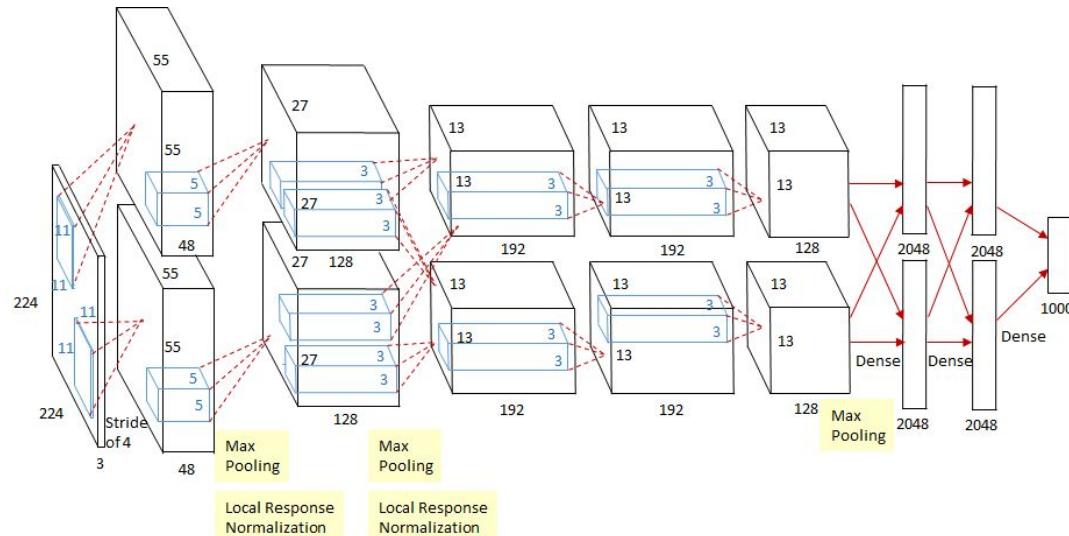
AlexNet (2012)

$$m = \left\lfloor \frac{n + 2p - f}{s} \right\rfloor + 1$$

Входной размер: 96x55x55

Второй слой:
пулинг 3x3 фильтры, stride=2

Какая размерность выхода?
96x27x27



AlexNet (2012)

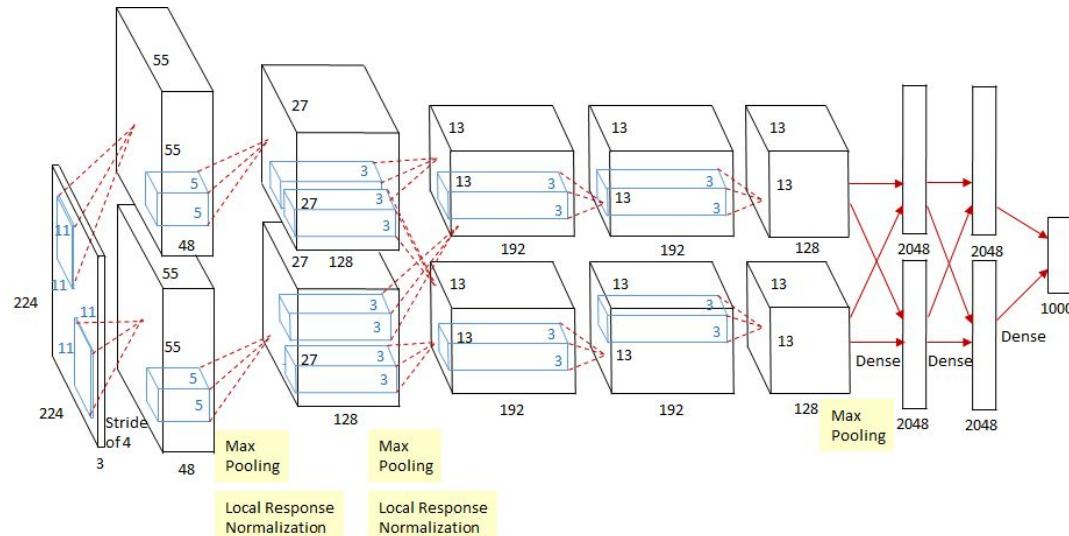
$$m = \left\lfloor \frac{n + 2p - f}{s} \right\rfloor + 1$$

Входной размер: 96x55x55

Второй слой:
пулинг 3x3 фильтры, stride=2

Какая размерность выхода?
96x27x27

Сколько параметров?



AlexNet (2012)

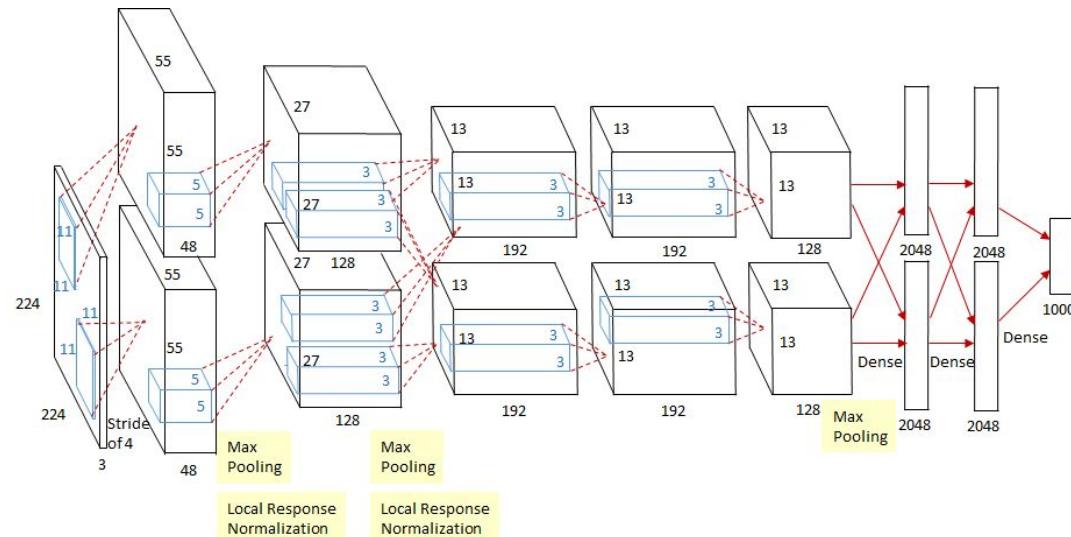
$$m = \left\lfloor \frac{n + 2p - f}{s} \right\rfloor + 1$$

Входной размер: 96x55x55

Второй слой:
пулинг 3x3 фильтры, stride=2

Какая размерность выхода?
96x27x27

Сколько параметров?
0!

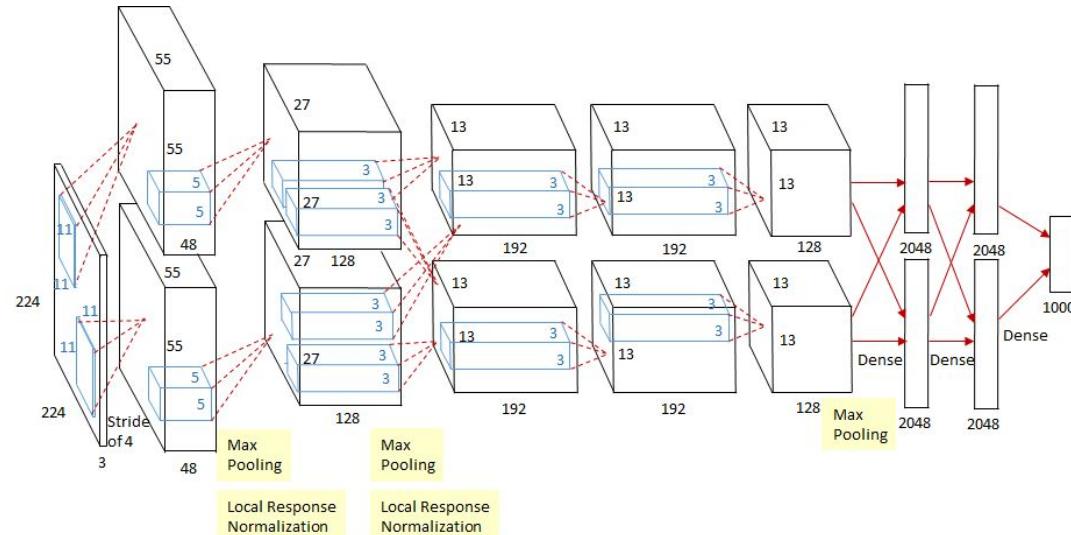


AlexNet (2012)

$$m = \left\lfloor \frac{n + 2p - f}{s} \right\rfloor + 1$$

Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

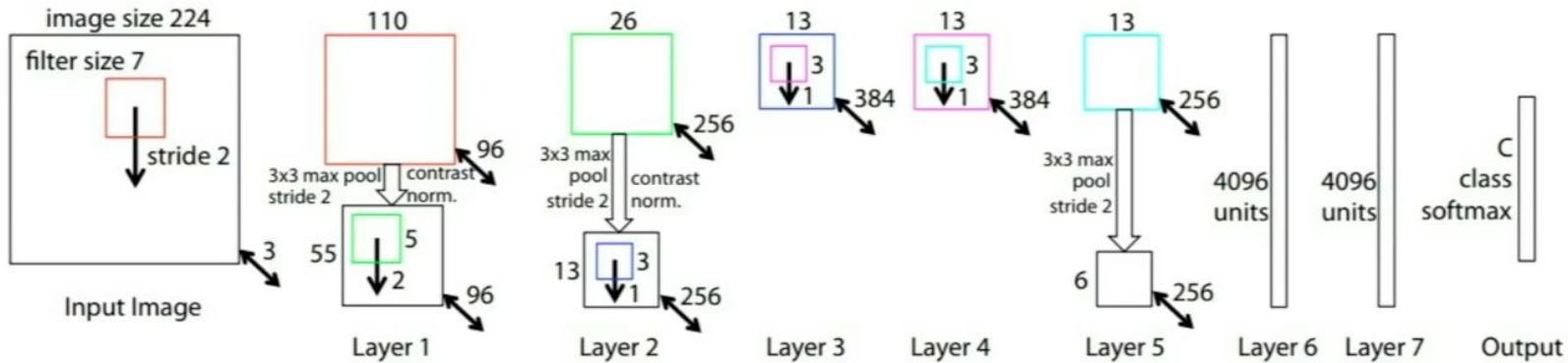


AlexNet (2012)

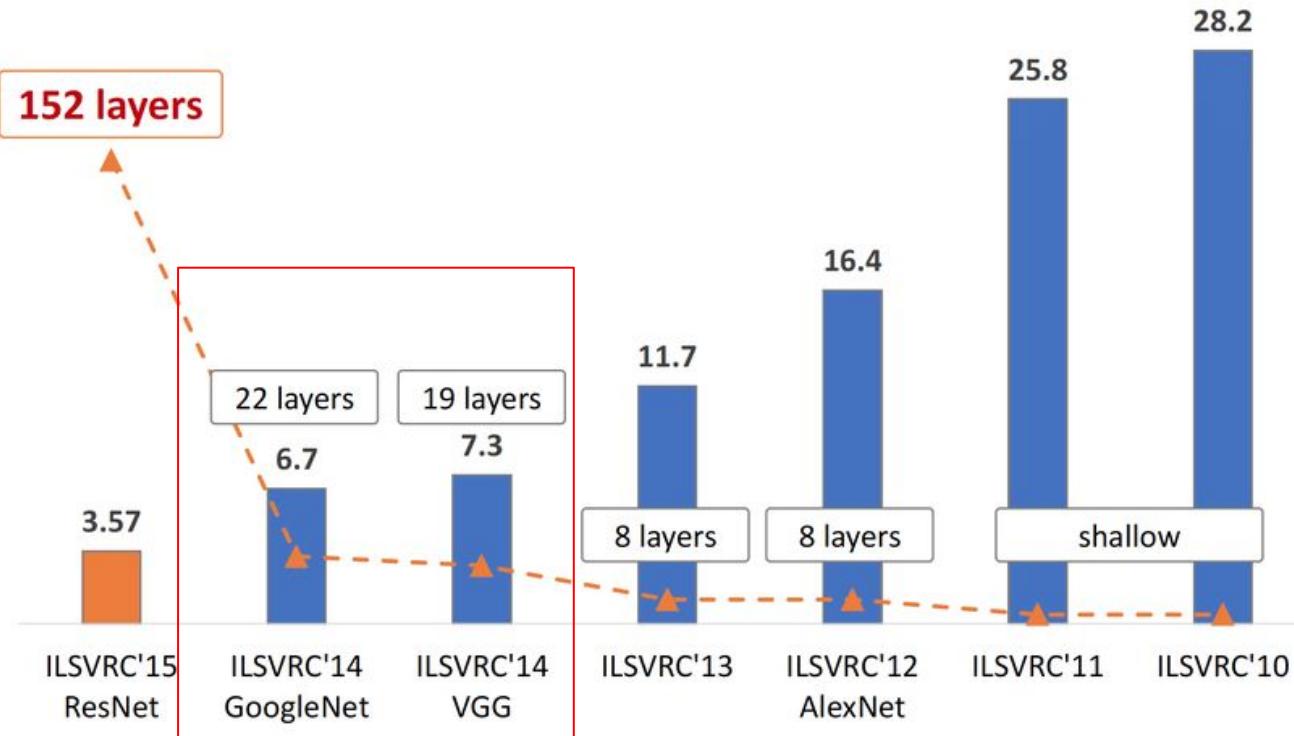
- параллельная архитектура
- использование ReLU в качестве функций активации
- для регуляризации использование Dropout перед полносвязными слоями
- число параметров ~ 60M

ZFNet

[Zeiler and Fergus, 2013]

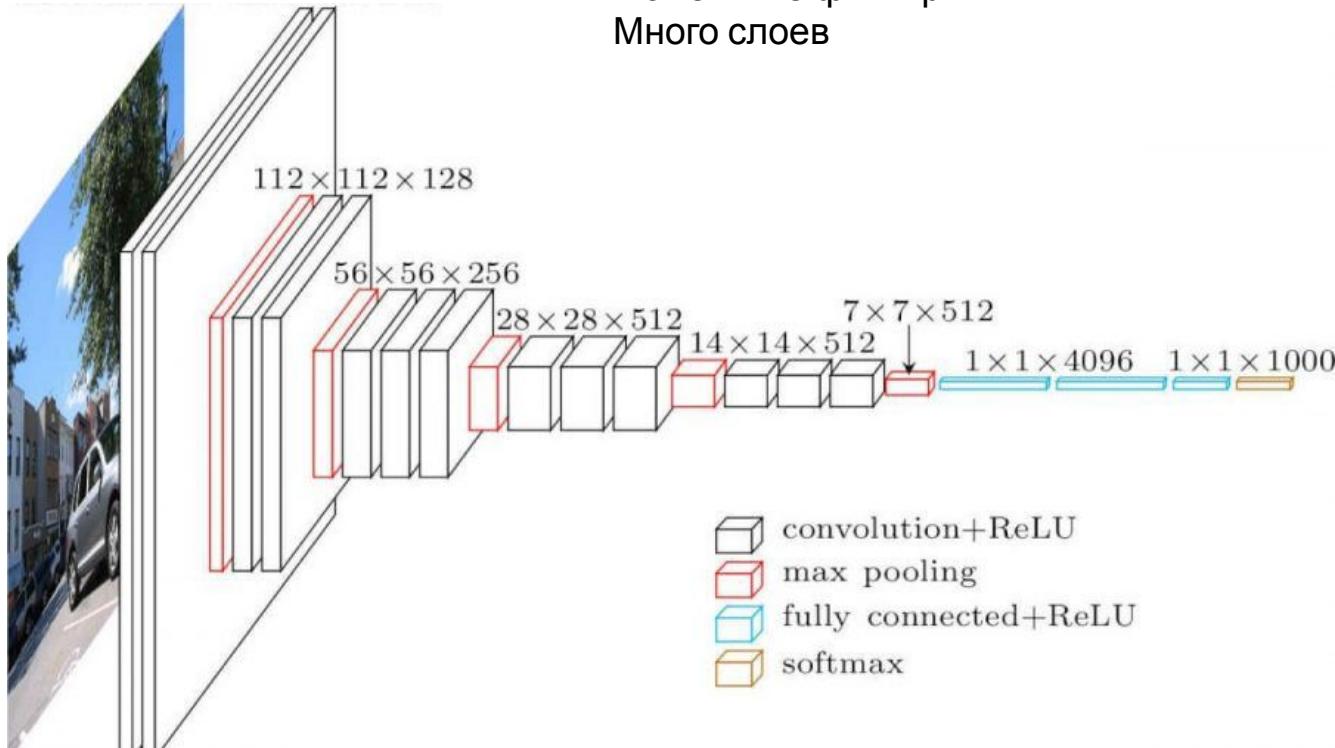


ImageNet



VGG (2014)

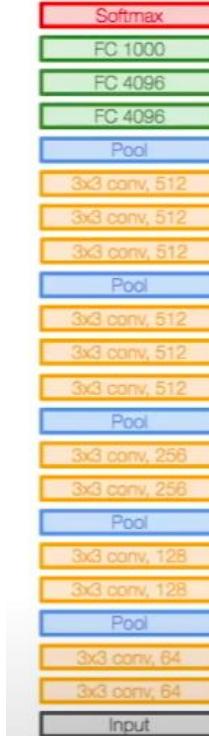
Маленькие фильтры
Много слоев



VGG (2014)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

Форвард пасс: 96 мб / одно изображение,
Бэквард: в два раза больше
138 миллионов параметров



VGG16

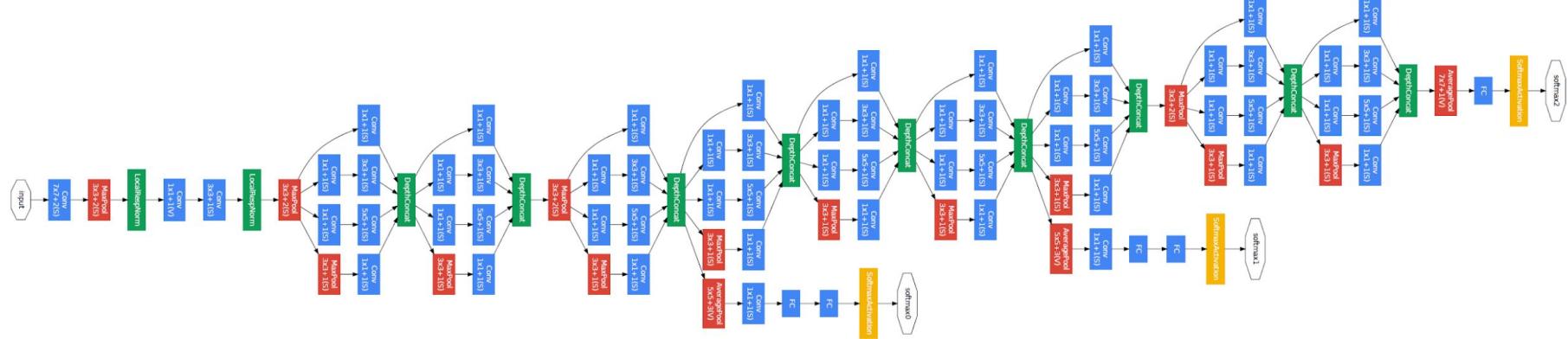
VGG (2014)

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

VGG (2014)

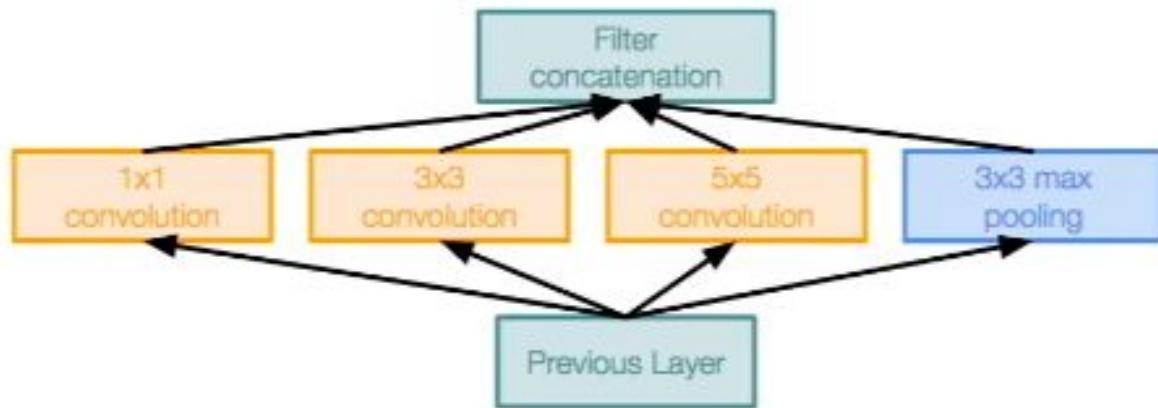
- последовательно применяются свертки фильтров с небольшим размером ядра
- большой объем данных на выходе каждого слоя требует большого количества памяти
- Используются ансамбли
- число параметров $\sim 130M$
- FC фичи хорошо обобщаются на другие задачи

GoogleNet (Inception) (2014)



Inception Block (naive)

Сетка внутри сетки



Inception Block (naive)

Сетка внутри сетки

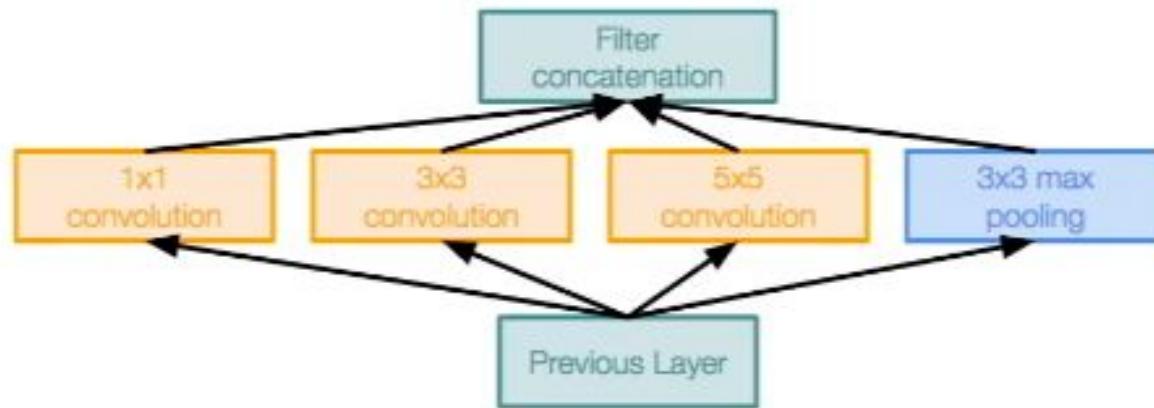
Conv Ops:

[1x1 conv, 128] $28 \times 28 \times 128 \times 1 \times 1 \times 256$

[3x3 conv, 192] $28 \times 28 \times 192 \times 3 \times 3 \times 256$

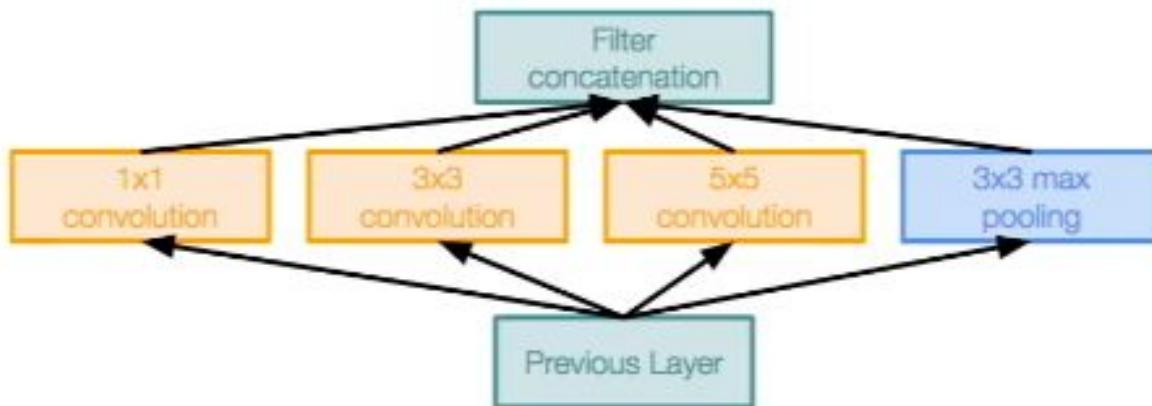
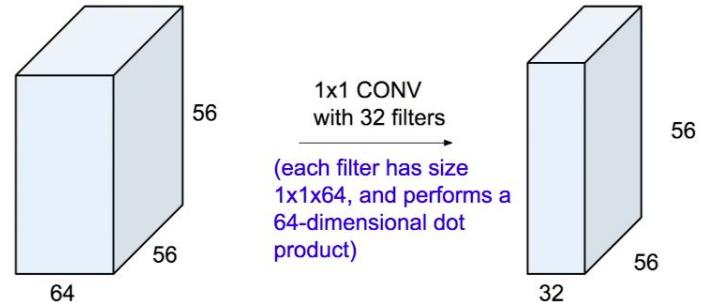
[5x5 conv, 96] $28 \times 28 \times 96 \times 5 \times 5 \times 256$

Total: 854M ops



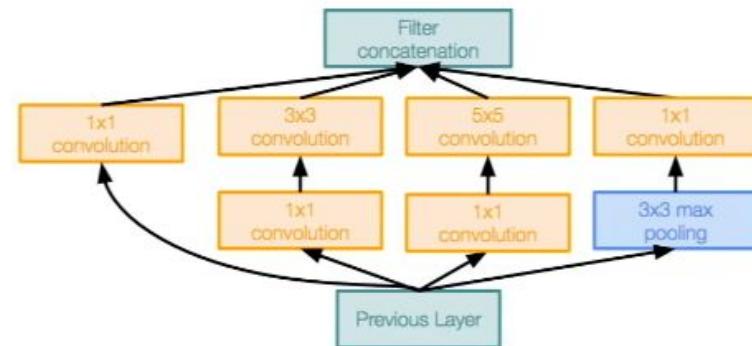
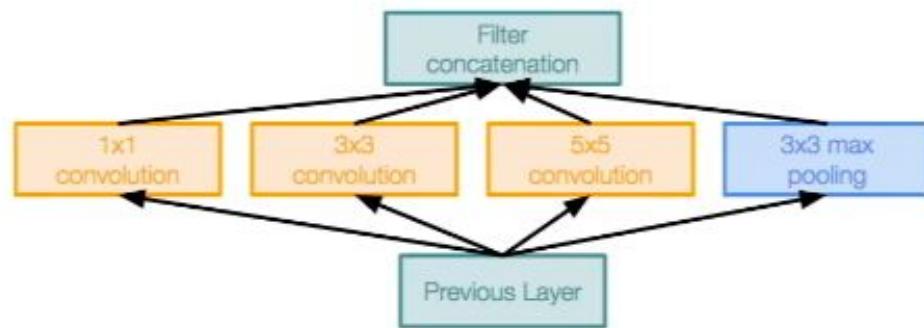
Inception Block (naive)

Сетка внутри сетки

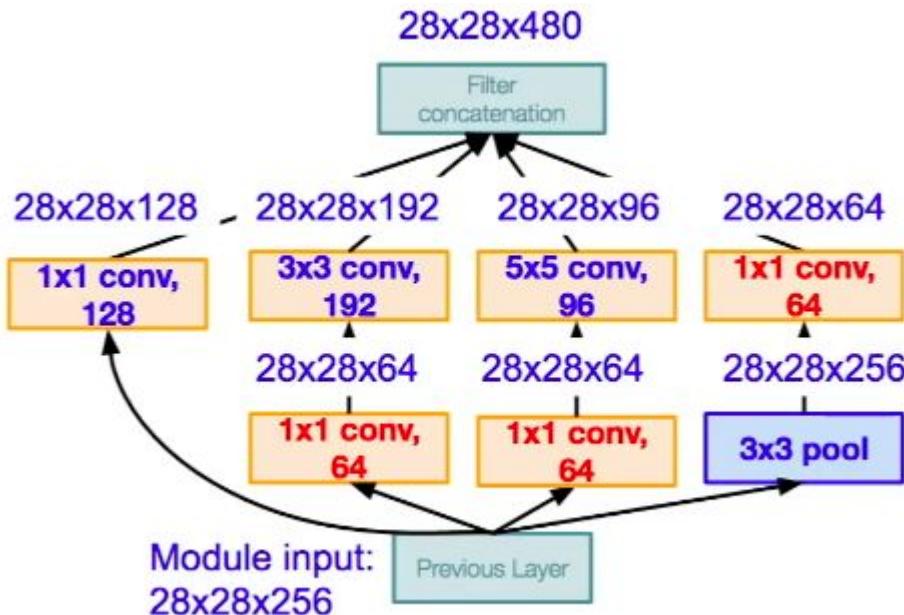


Inception Block

Сетка внутри сетки



Inception Block

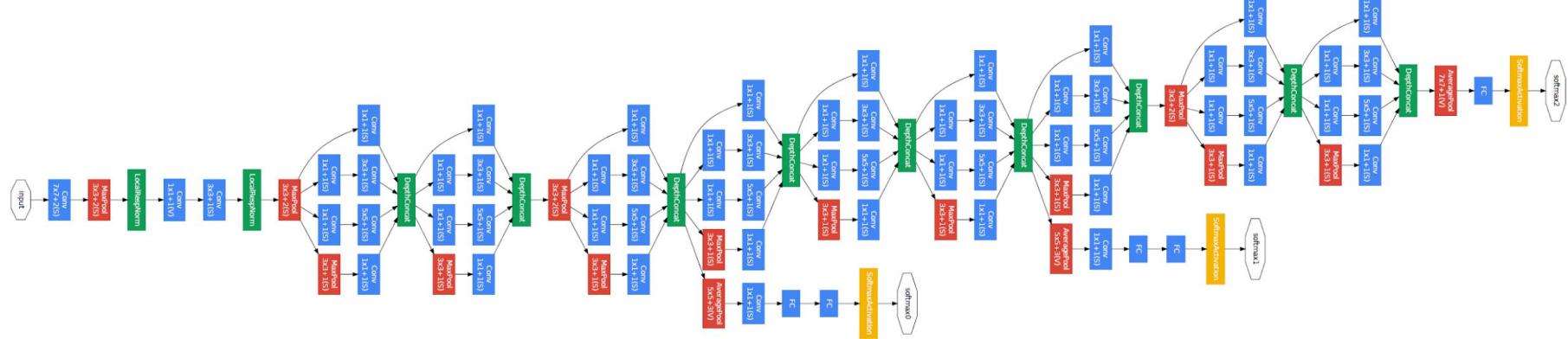


Conv Ops:

- [1x1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$
- [1x1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$
- [1x1 conv, 128] $28 \times 28 \times 128 \times 1 \times 1 \times 256$
- [3x3 conv, 192] $28 \times 28 \times 192 \times 3 \times 3 \times 64$
- [5x5 conv, 96] $28 \times 28 \times 96 \times 5 \times 5 \times 64$
- [1x1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$

Total: 358M ops

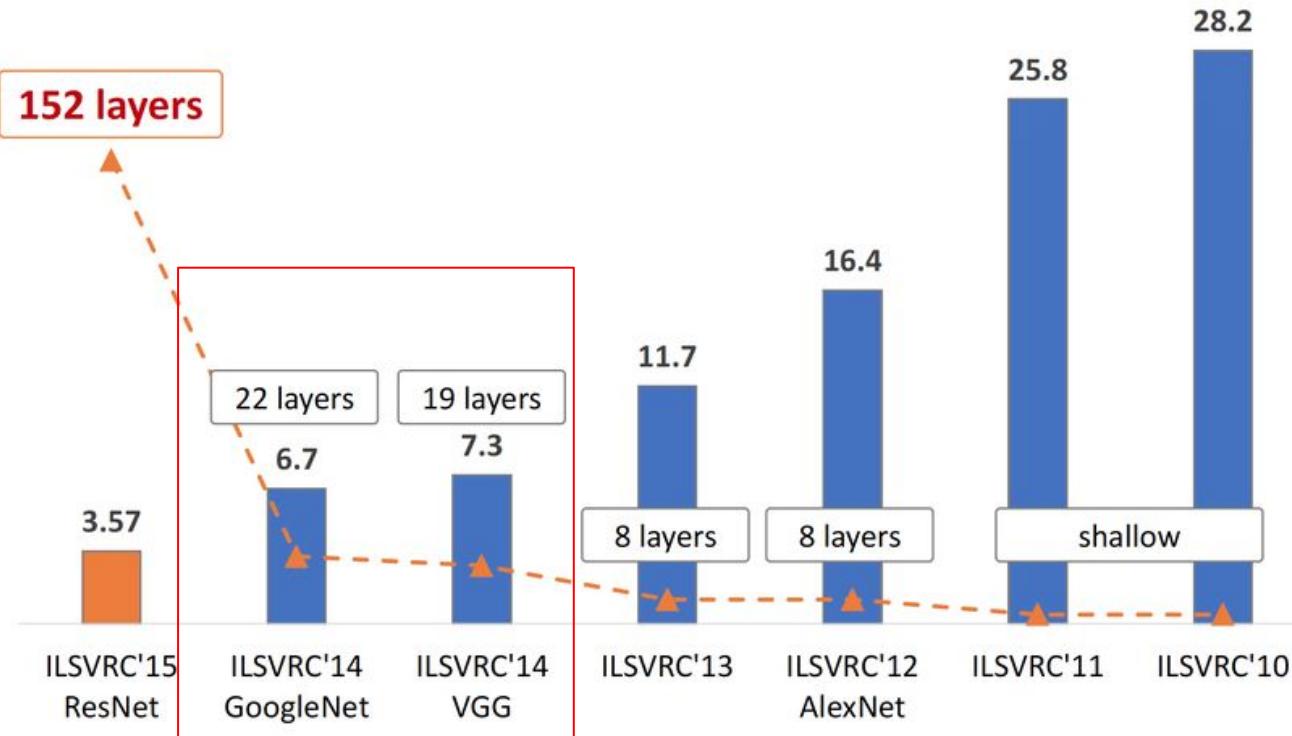
GoogleNet (Inception) (2014)



GoogleNet (Inception) (2014)

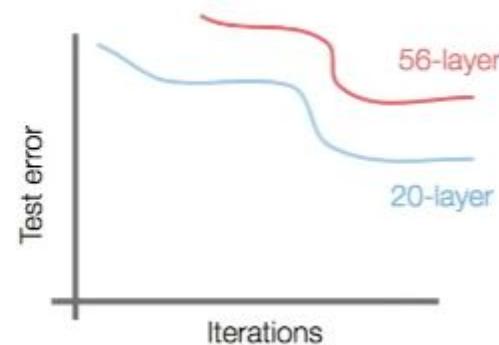
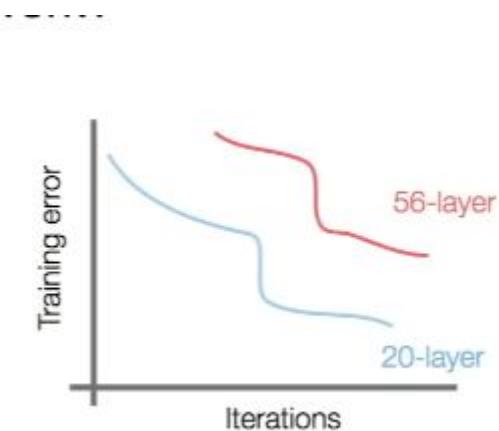
- применяются фильтры разного размера к одним и тем же данным
- уменьшение глубины выхода (числа каналов) за счет свертки 1×1
- в результате уменьшения глубины получаем ускорение сверток 3×3 и 5×5
- число параметров $\sim 5M$

ImageNet

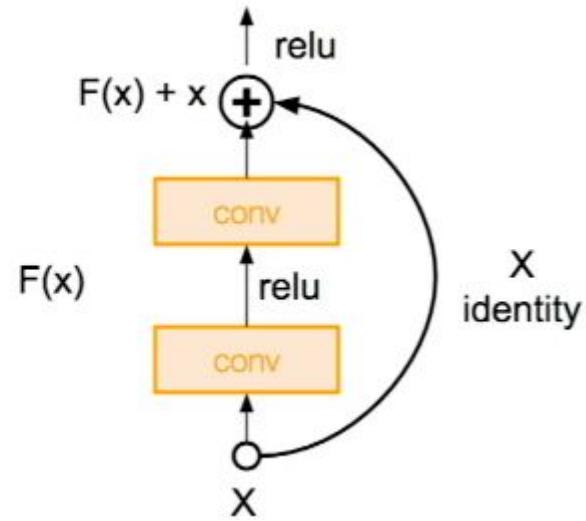
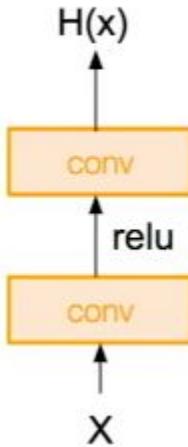


ResNet (2015)

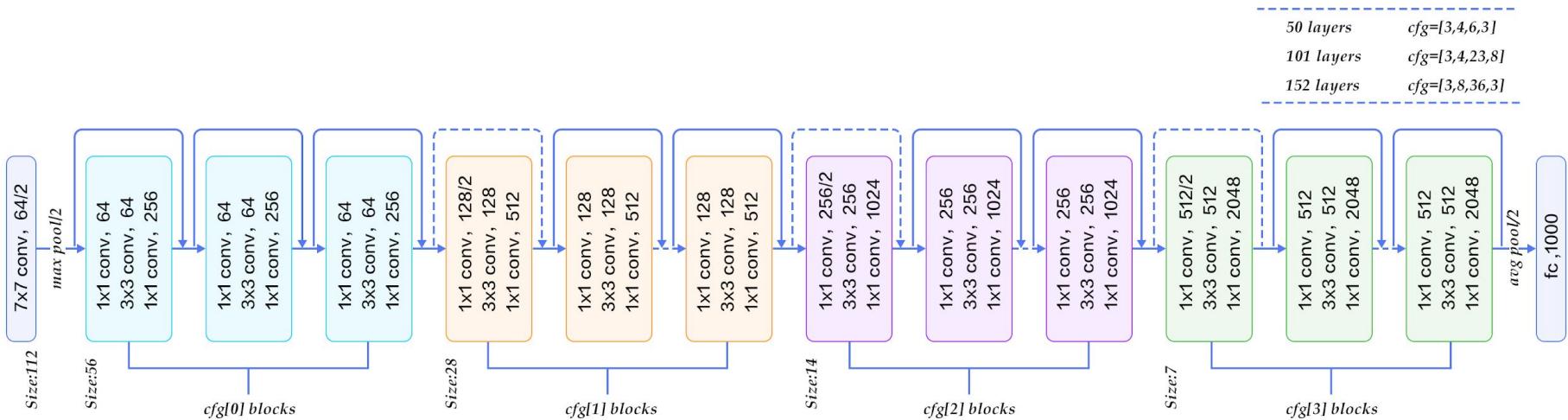
Классическая сетка с последовательными блоками



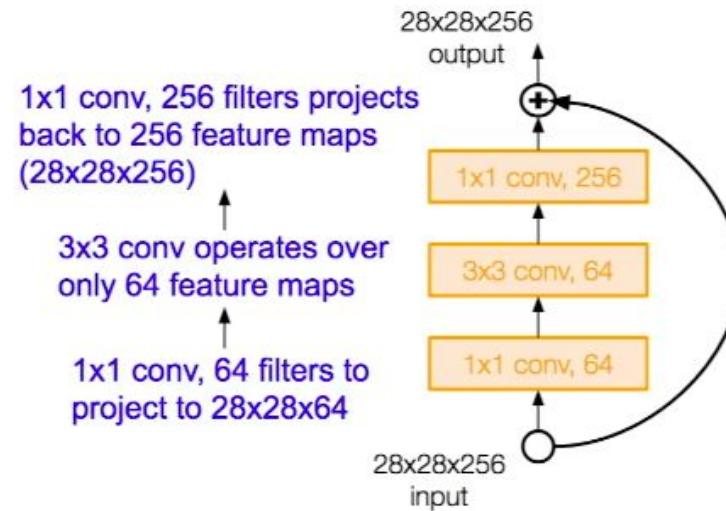
ResNet (2015)



ResNet (2015)



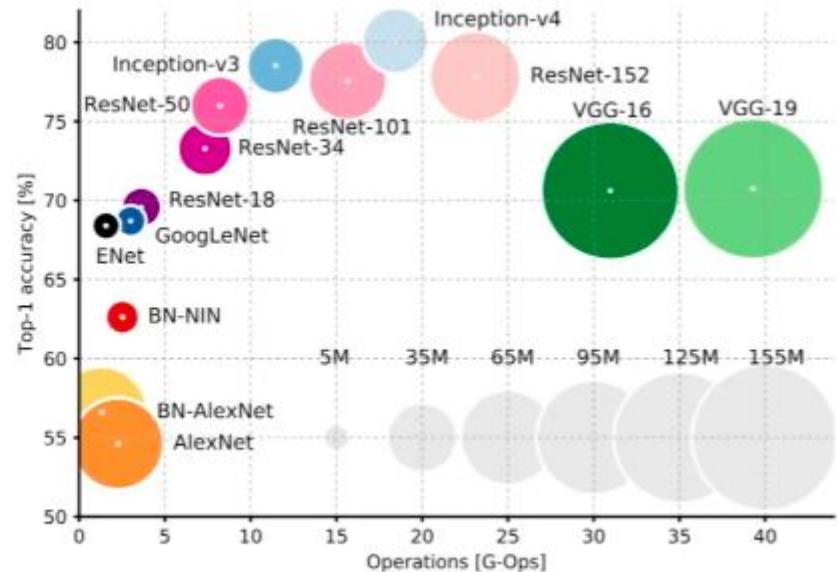
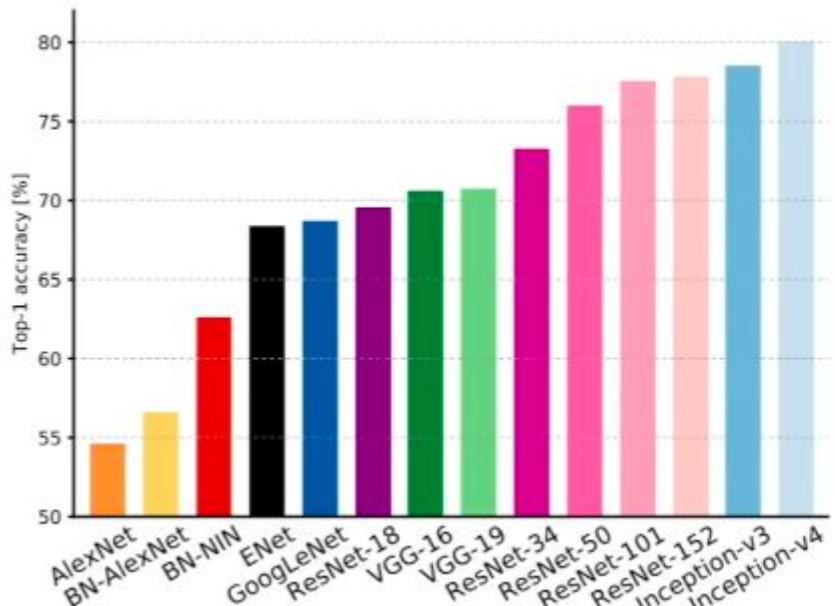
ResNet (2015)



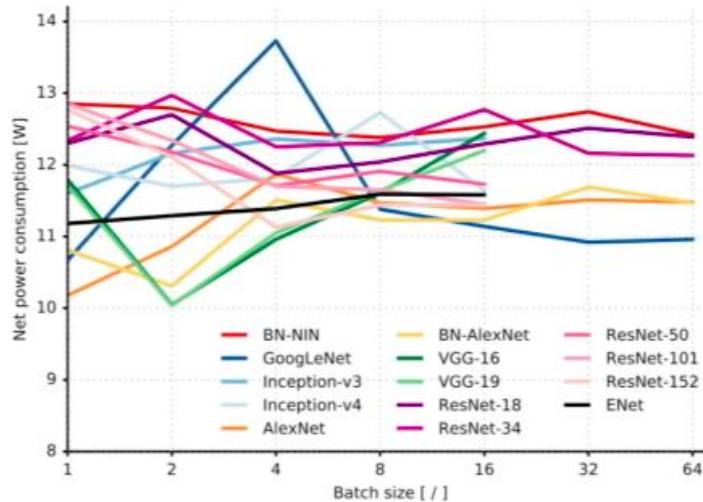
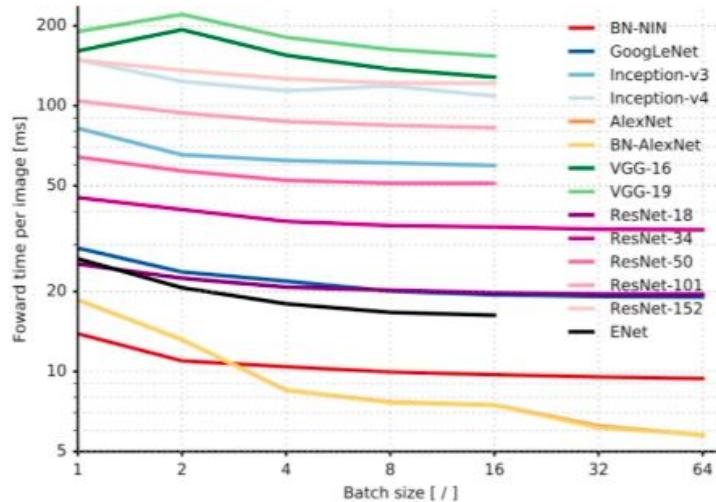
ResNet (2015)

- добавлена параллельное соединение с исходными данными
- Много слоев (150!!)
- число параметров ResNet 50 ~ 25M
- Первые опередили “human performance” (top-1 error 5%)

Comparison

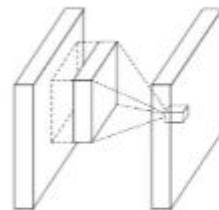


Comparison (forward pass time)

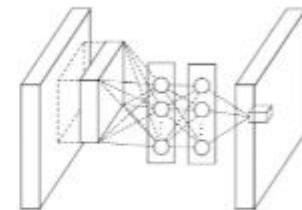


Другие модели

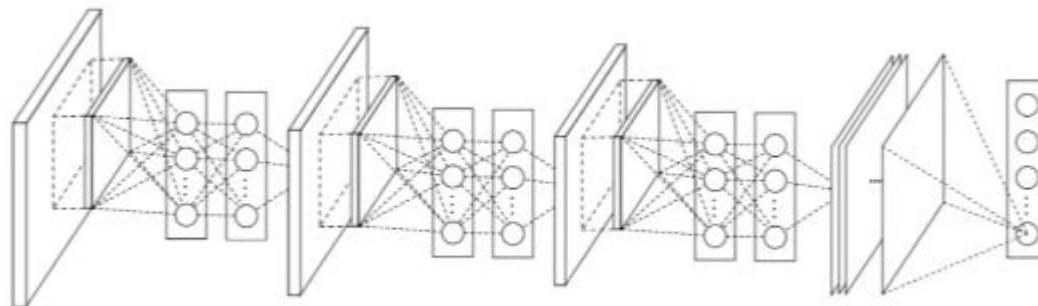
Network in Network (NiN), 2014



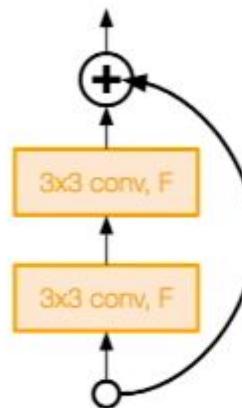
(a) Linear convolution layer



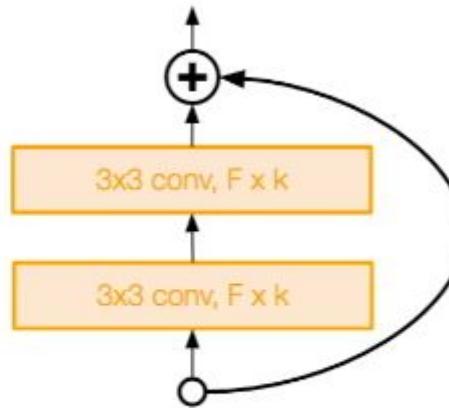
(b) Mlpconv layer



Wide ResNET

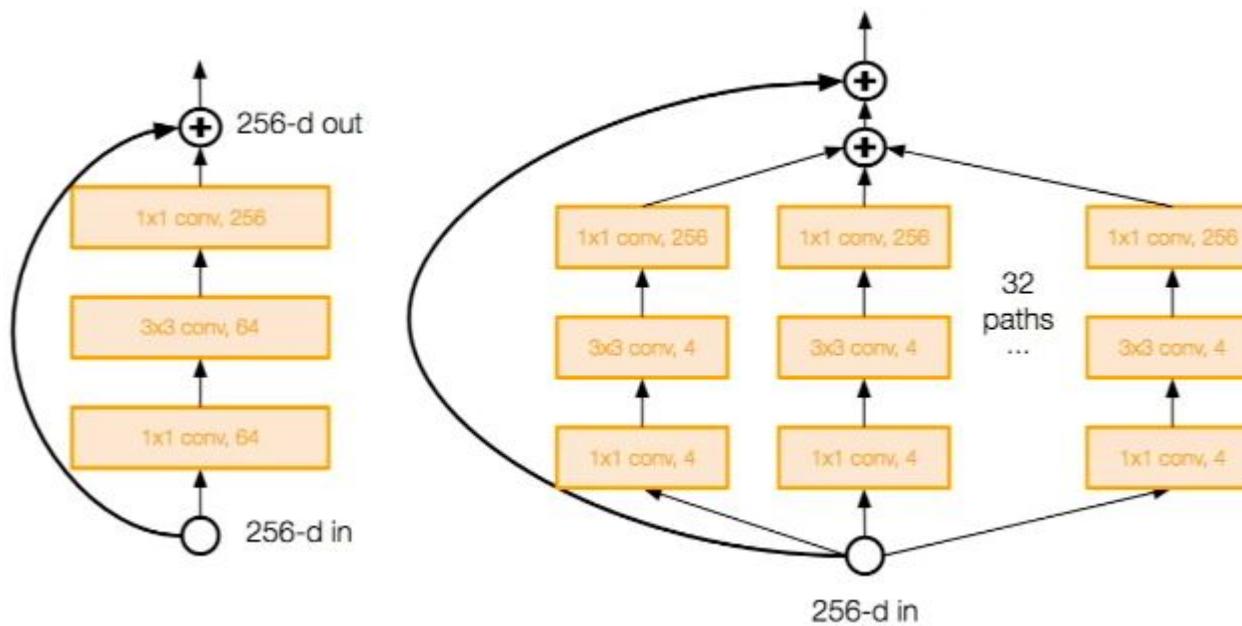


Basic residual block



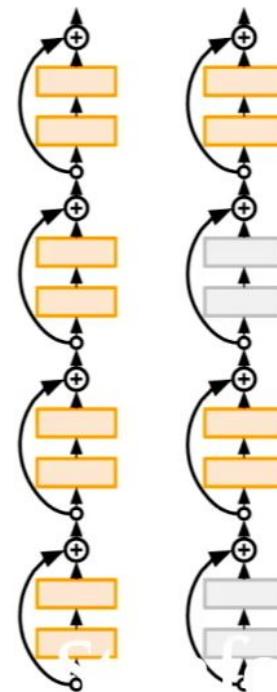
Wide residual block

ResNeXt

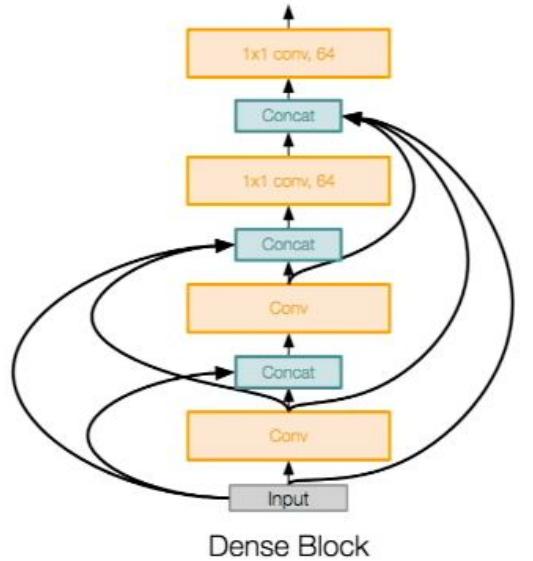
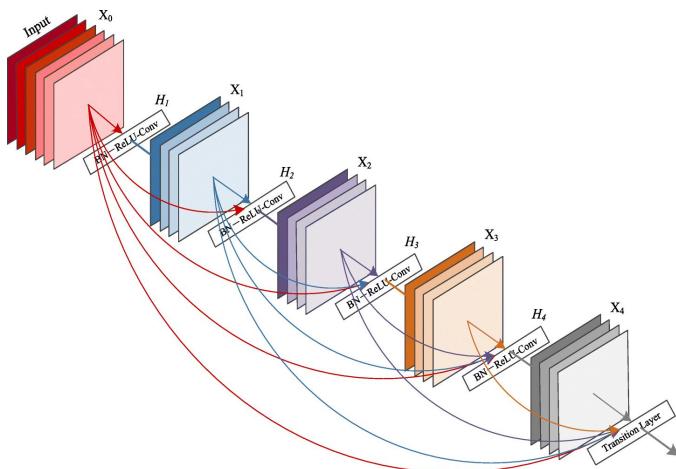


ResNets with stochastic Depth

Во время трейна случайным образом выключать residual блоки (своего рода дропаут)



DenseNet

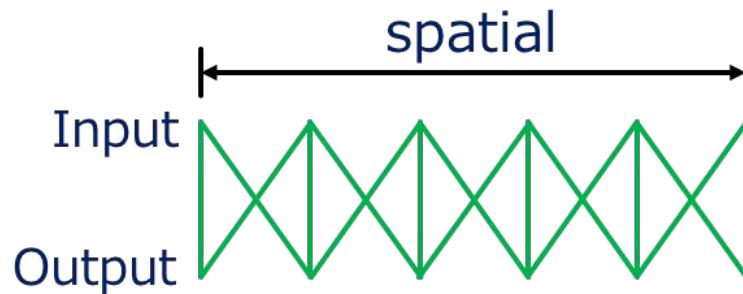


Softmax
FC
Pool
Dense Block 3
Conv
Pool
Conv
Dense Block 2
Conv
Pool
Conv
Dense Block 1
Conv
Input

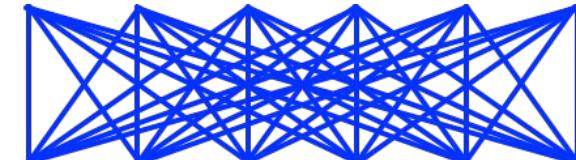
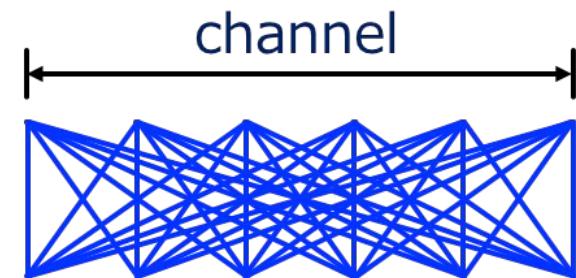
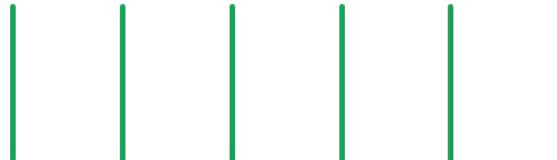
Efficient models

Convolutions

conv 3x3

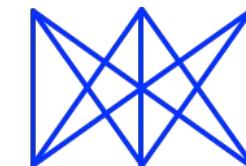
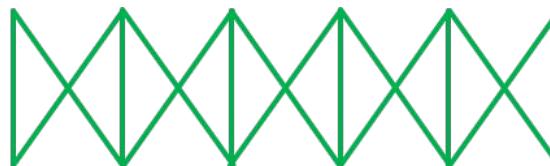


conv 1x1

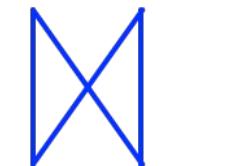
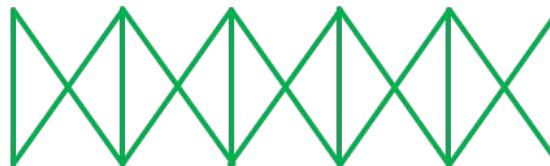


Grouped convolutions

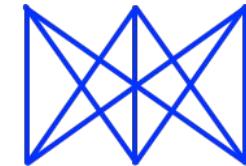
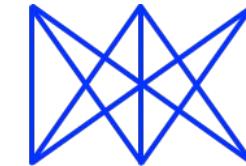
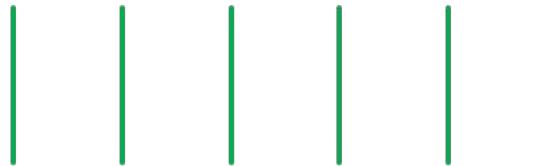
gconv 3x3 (g=2)



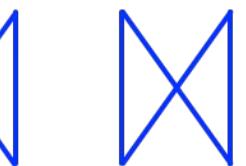
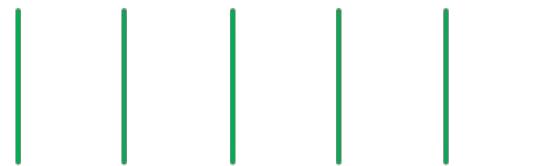
gconv 3x3 (g=3)



gconv 1x1 (g=2)



gconv 1x1 (g=3)



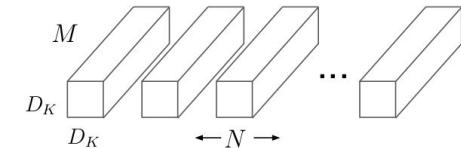
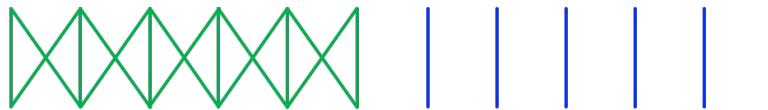
Grouped convolutions

- Основная идея - выходные каналы у свертки зависят не от всех каналов на входе, а только от определенной части
- Число параметров уменьшается:
 - Число параметров:
weight: (cout, cin / groups, kh, kw)
Bias: (cout,)
 - Обычная свертка = групповая, если число групп = 1

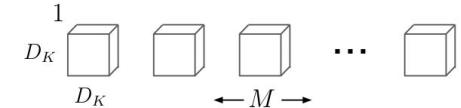
Depthwise convolution

- Особый случай групповой свертки с числом групп, равным числу входных и выходных каналов

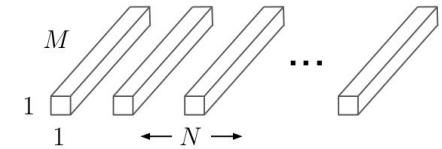
depthwise conv



(a) Standard Convolution Filters



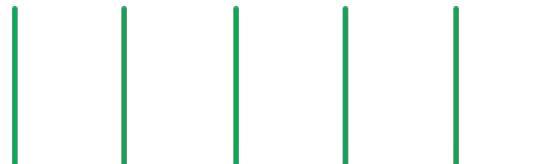
(b) Depthwise Convolutional Filters



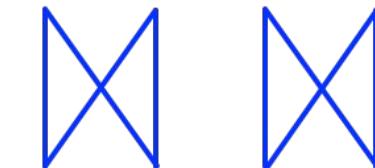
(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

ResNeXt (2016)

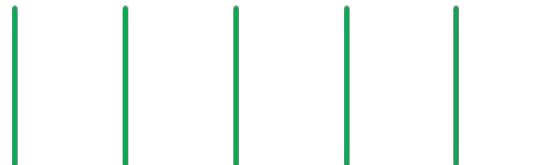
conv 1x1



gconv 3x3

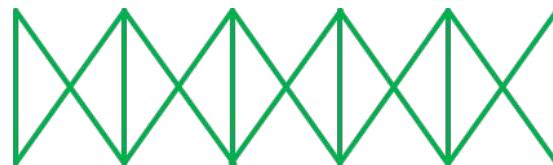


conv 1x1

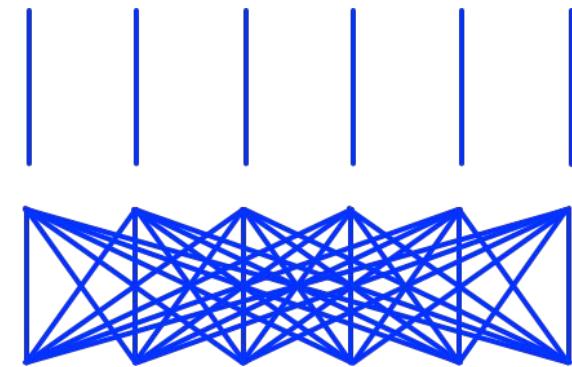
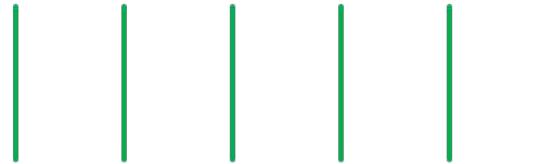


MobileNet (Separable conv) (2017)

depthwise conv

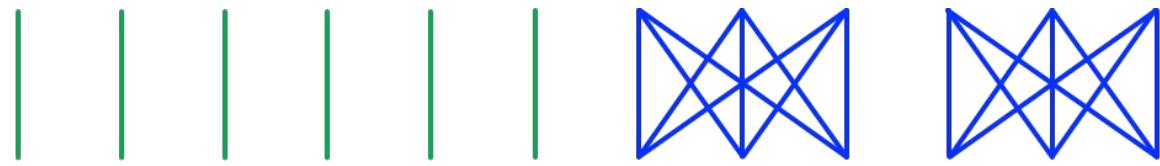


conv 1x1



ShuffleNet (2017)

gconv 1x1



c shuffle



depthwise conv

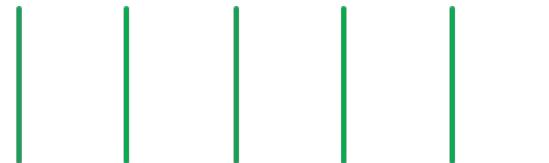


gconv 1x1

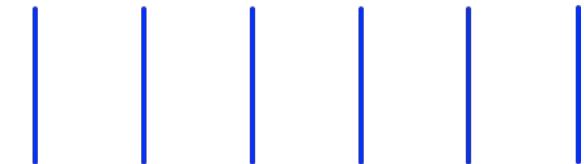
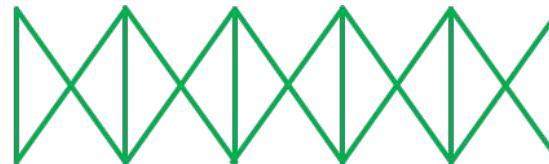


MobileNet-v2 (2018)

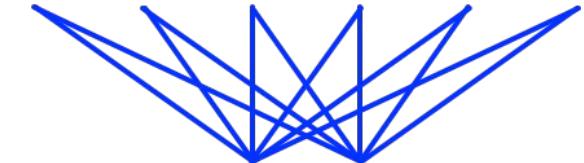
conv 1x1



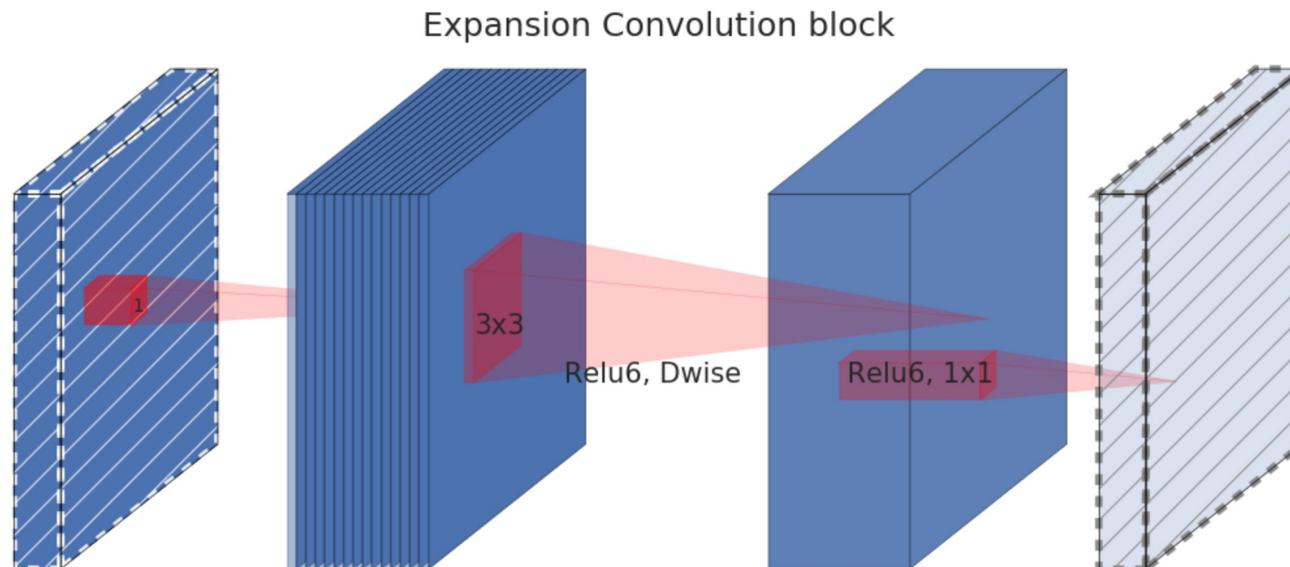
depthwise conv



conv 1x1



MobileNet-v2 (2018)





Обучение CNN на практике. Transfer learning



Transfer learning

- на практике очень редко обучаются сверточные сети с нуля
- это связано как правило с ограниченным объемом доступных данных и ограниченными вычислительными ресурсами
- как правило, существующие предобученные сети адаптируют под конкретную задачу

Transfer learning

- как правило выходной слой предобученной сети требует изменения для каждой задачи (разное число классов)
- копируем архитектуру и веса предобученной сети и заменяем последний слой
- дообучаем новый выходной слой на данных задачи

Transfer learning

Как будем обучать?

Вариант 1:

1. Инициализируем новые слои нулями (это важно!)
2. Обучаем целиком с небольшим learning rate

Вариант 2 (предпочтительнее):

1. Замораживаем веса (`param.requires_grad = False`) всех слоев, кроме тех что заменили
2. Обучаем сеть
3. (Опционально) Размораживаем веса всех слоев или **k-последних** и обучаем сеть целиком с меньшим learning rate. С этим нужно аккуратно так как можем переобучиться.

Transfer learning

Сколько последних слоев обучать? (число k с прошлого слайда)

	Задача похожа	Задача сильно отличается
Данных мало	$k = 1$ т.е. обучаем линейный классификатор на признаках с последнего скрытого слоя	Тут проблемы :) Либо подбираем $k \gg 1$, либо обучаем с нуля
Данных много	$k > 1$	$k \gg 1$



Обучение CNN на практике. Дисбаланс классов

Дисбаланс классов

- часто на практике выборка не сбалансирована
- это приводит к тому, что модель переобучается на классы с большим числом примеров и дает смещеннное предсказание

Дисбаланс классов

Как бороться?

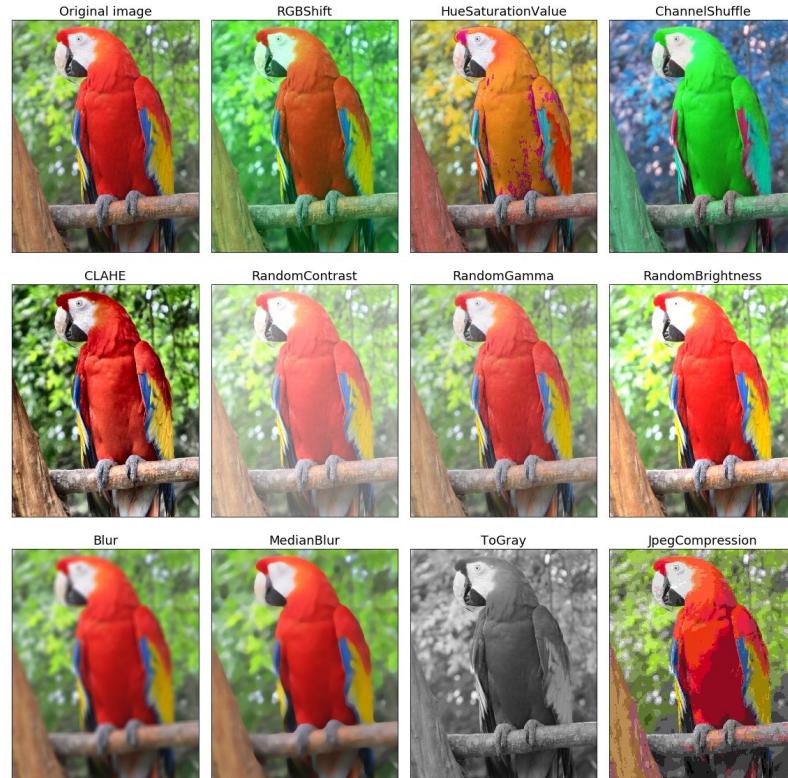
1. выравниваем градиенты, взвешивая ошибку обратно пропорционально числу классов
2. выравниваем баланс классов на уровне батча
3. добавляем число примеров редких классов за счет аугментации



Обучение CNN на практике. Аугментация данных



Аугментация данных



Аугментация данных

Original



VerticalFlip



HorizontalFlip



RandomRotate90



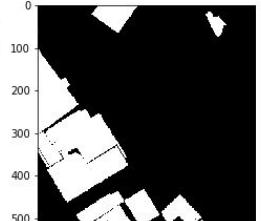
Transpose



ShiftScaleRotate



RandomSizedCrop



Аугментация данных

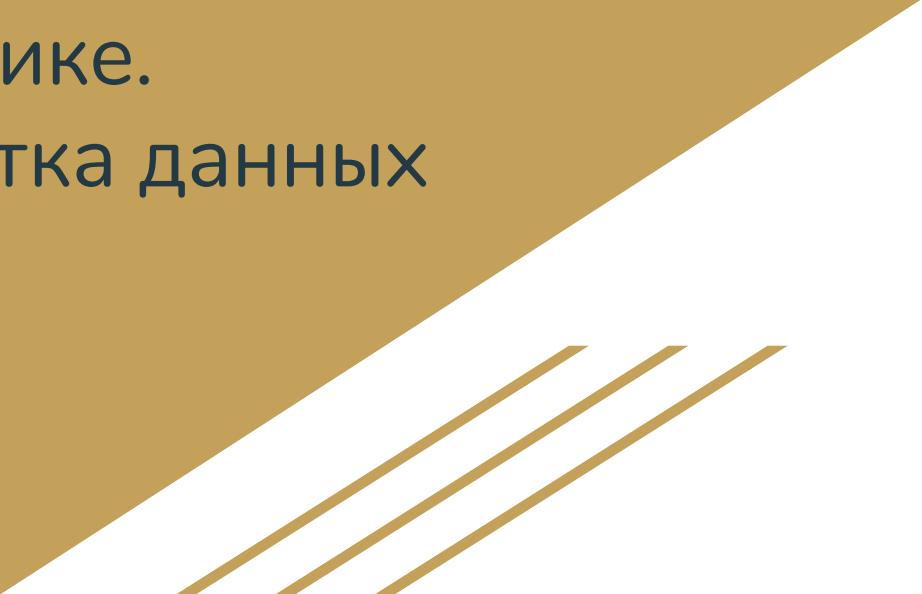
- зеркальное отражение по горизонтали (horizontal flip)
- вырезаем случайную часть из изображения (crop) и масштабируем до исходного
- аугментация освещенности (в пространстве HSV)
- аугментация цвета (случайный шум по каналам)
- и многие другие..

Реализация

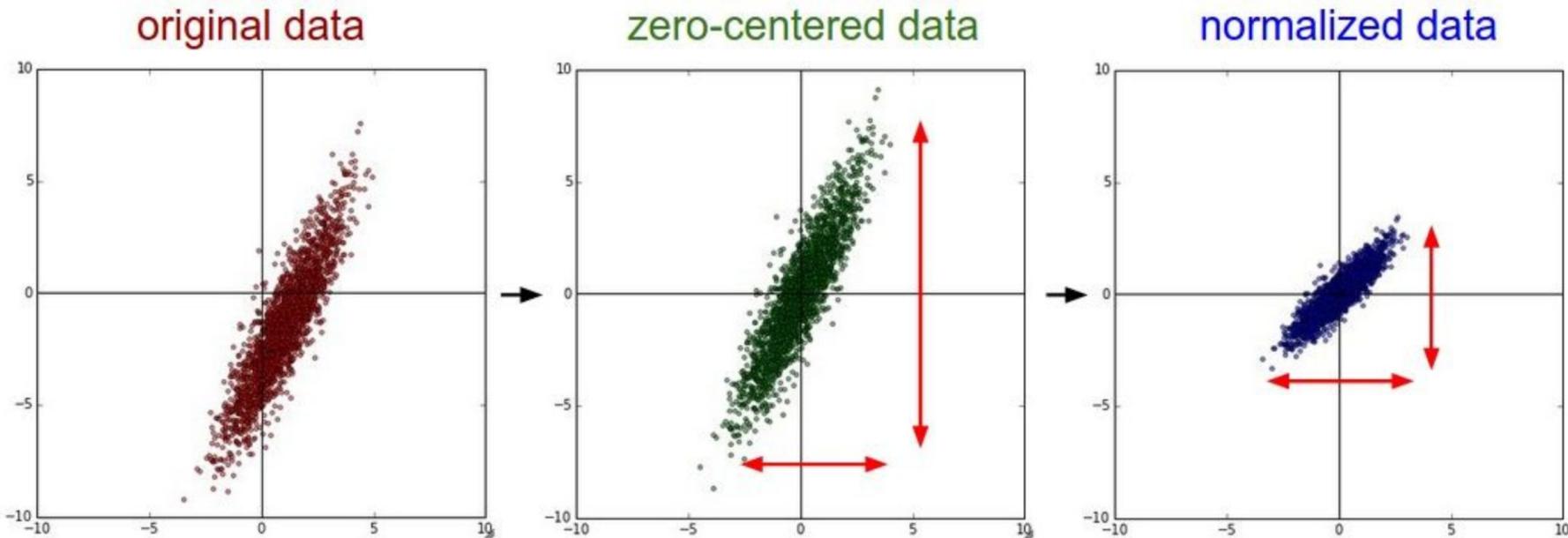
- на практике аугментированные изображения сильно увеличивают размер выборки
- хранить на диске аугментированные копии нецелесообразно
- процесс аугментации запускают на лету параллельно с обучением
- аугментация может выполняться как на CPU, так и на GPU



Обучение CNN на практике. Предобработка данных



Центрирование и нормализация





Обучение CNN на практике. Инициализация



Инициализация весов

- инициализация различными весами (если нейроны возвращают одинаковые значения, значит и градиенты для них будут одинаковыми)
- инициализация случайными значениями с небольшой дисперсией
- [torch.nn.init](#)



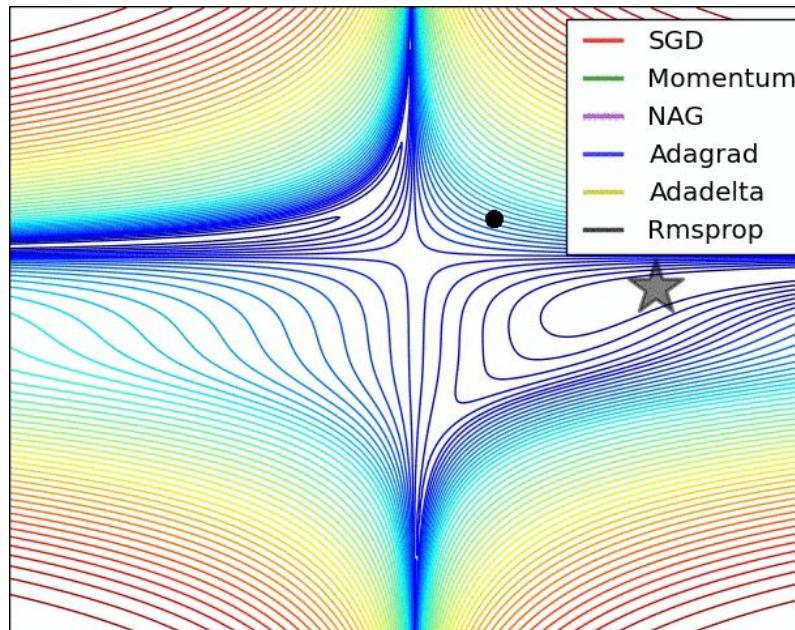
Обучение CNN на практике. Градиентный спуск



Оптимизация (градиентный спуск)

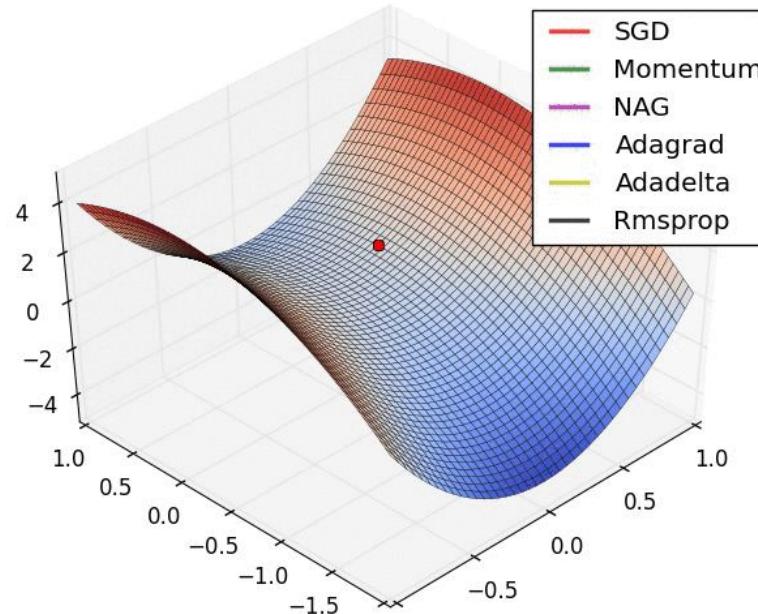
- при увеличении размера батча оценка градиента получается стабильнее, что позволяет увеличить learning rate
- в случае, если не происходит изменение метрики на валидации, стоит уменьшить значение learning rate ReduceLROnPlateau
- наиболее популярные оптимизаторы: SGD, Adam
- [torch.optim](#)

Оптимизация (градиентный спуск)



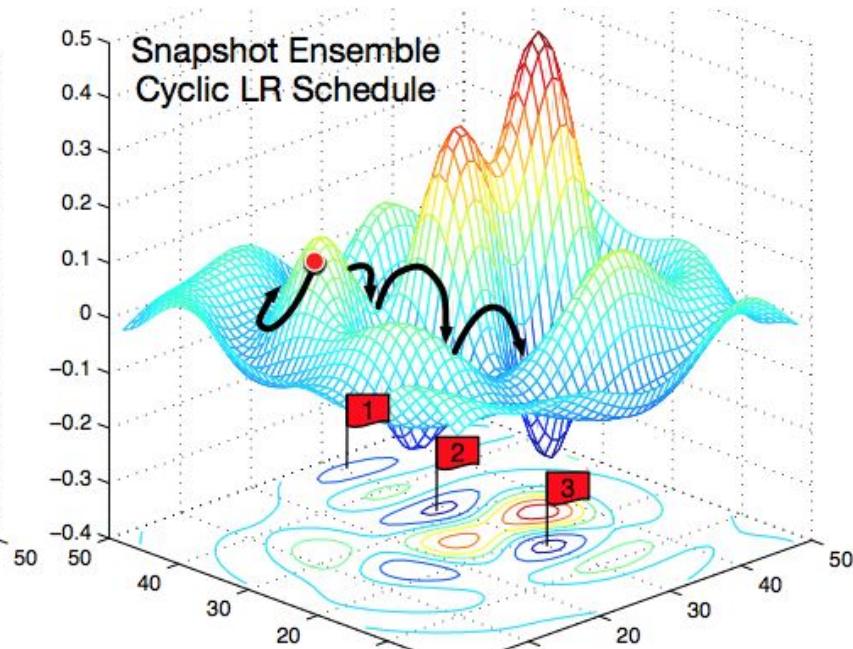
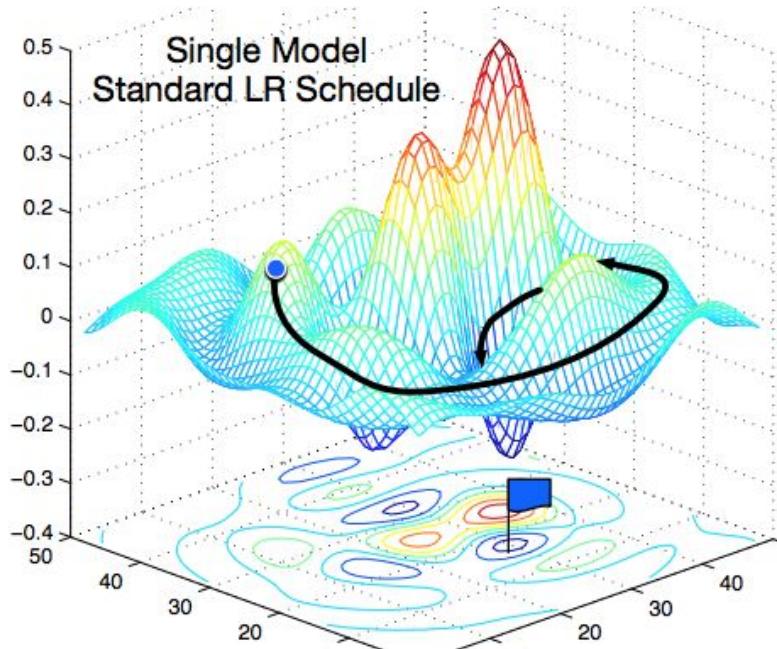
<http://ruder.io/optimizing-gradient-descent/>

Оптимизация - седловая точка



<http://ruder.io/optimizing-gradient-descent/>

Изменение LR по расписанию



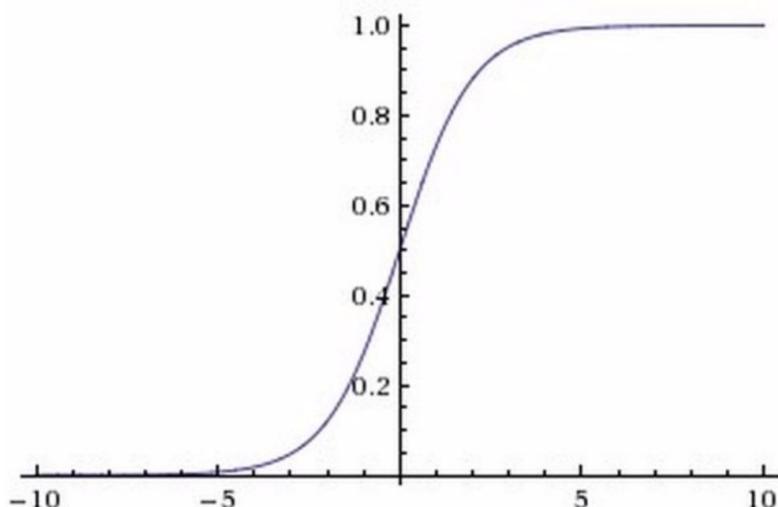


Обучение CNN на практике. Функции активации



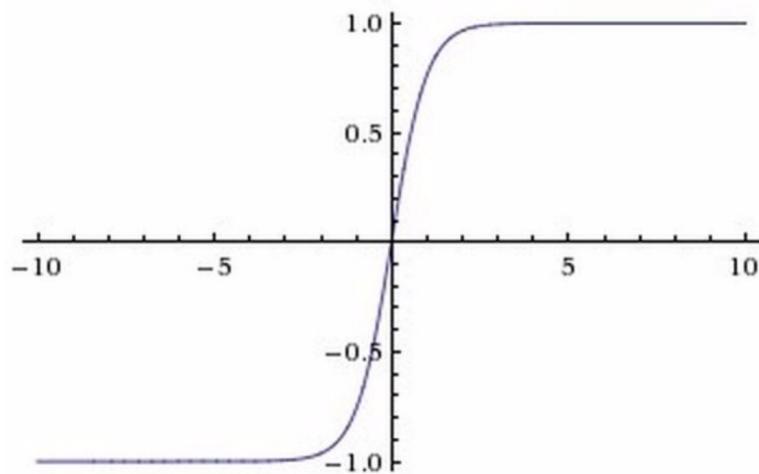
Функции активации - sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$



- убивает градиент
- смещена относительно нуля - проблема для обучения
- обычно используется на выходном слое в случае multilabel (not multiclass!), binary classification и в различных “gating” механизмах

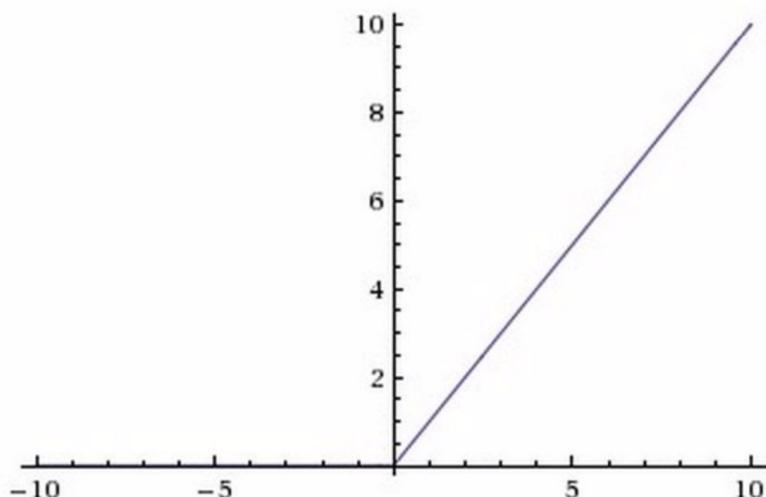
Функции активации - \tanh



- остается проблема с градиентами
- решена проблема с центрированием

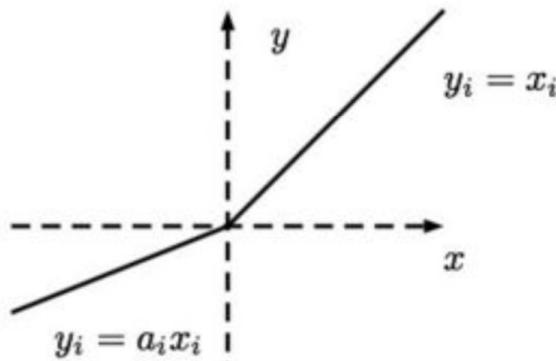
Функции активации - ReLU

$$f(x) = \max(0, x)$$

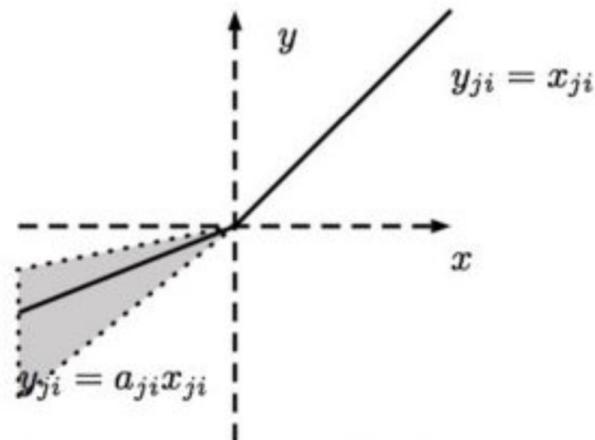


- простая в реализации
- отсутствие насыщения ускоряет процесс сходимости
- при большом значении градиента значения до активаций могут уйти далеко в минус и не вернуться:
dying ReLU

Функции активации - LeakyReLU



Leaky ReLU/PReLU

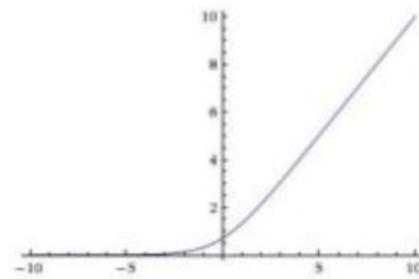


Randomized Leaky ReLU

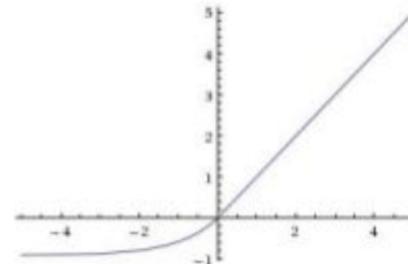
- угол наклона задается (обычно 0.01, 0.1, 0.2)
- либо угол наклона является обучаемым параметром
- либо угол наклона изменяется случайным образом

Функции активации - Exponential LU

Softplus and Exponential Linear Unit (ELU)



$$\text{softplus}(x) = \log(1 + e^x)$$



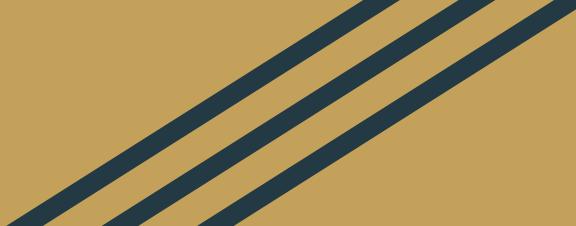
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

- решена неоднозначность с градиентом в области нуля
- долго вычисляется

Функции активации ReLU

Activation	Training Error	Test Error
ReLU	0.1356	0.429
Leaky ReLU, $a = 100$	0.11552	0.4205
Leaky ReLU, $a = 5.5$	0.08536	0.4042
PReLU	0.0633	0.4163
RReLU	0.1141	0.4025

Table 4. Error rate of CIFAR-100 Network in Network with different activation function



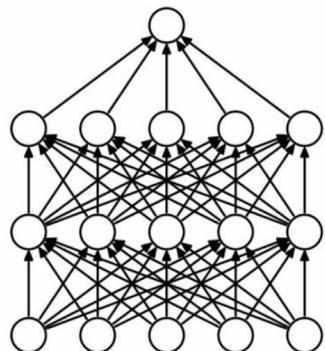
Обучение CNN на практике. Регуляризация



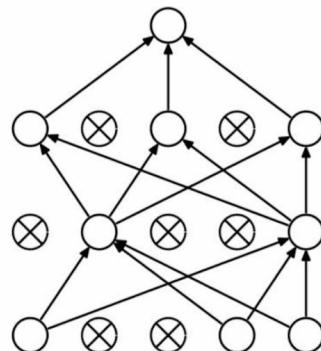
Регуляризация

- L1, L2 - добавка в функцию потерь модуль, при добавлении обоих - называется elastic net
- регуляризация заключается в добавлении соответствующего слагаемого в функцию потерь

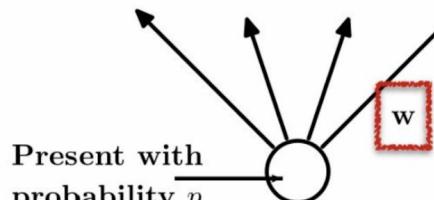
Регуляризация - Dropout



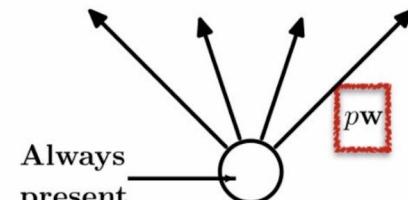
(a) Standard Neural Net



(b) After applying dropout.



(c) At training time



(d) At test time



Обучение CNN на
практике.

Оптимизация вычислений



Параметры фильтров

- размеры изображения кратны числу pooling слоев
- чем меньше размер фильтра, тем меньше операций необходимо выполнить для вычисления свертки
- отступы (padding) необходимо добавлять при свертке для сохранения информации на краях картинки (актуально с увеличением глубины)

Распределенное обучение

Существует два типа параллелизма:

- **Data Parallel** - модель копируется на каждый из доступных гпу, входные данные делятся на равные части, каждая из частей посыпается на свой гру, результат далее объединяется
- **Model parallel** - модель делится на части, и каждая часть работает на отдельном гру

Чаще приходится сталкиваться с Data Parallel, а не с Model parallel

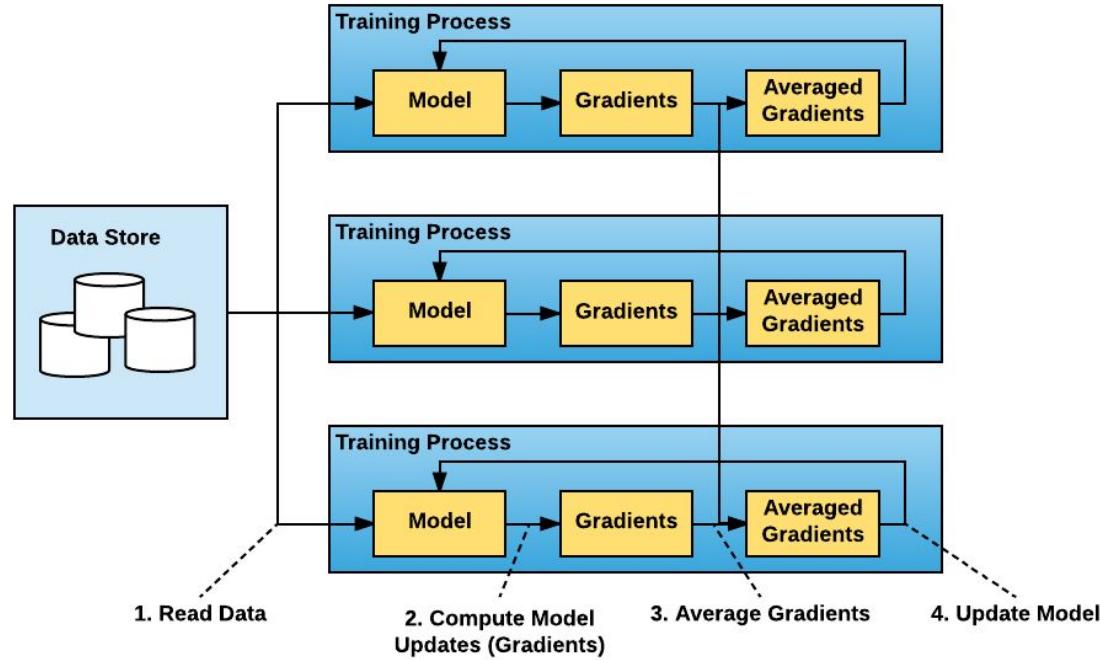
Распределенное обучение в PyTorch

Взято из документации:

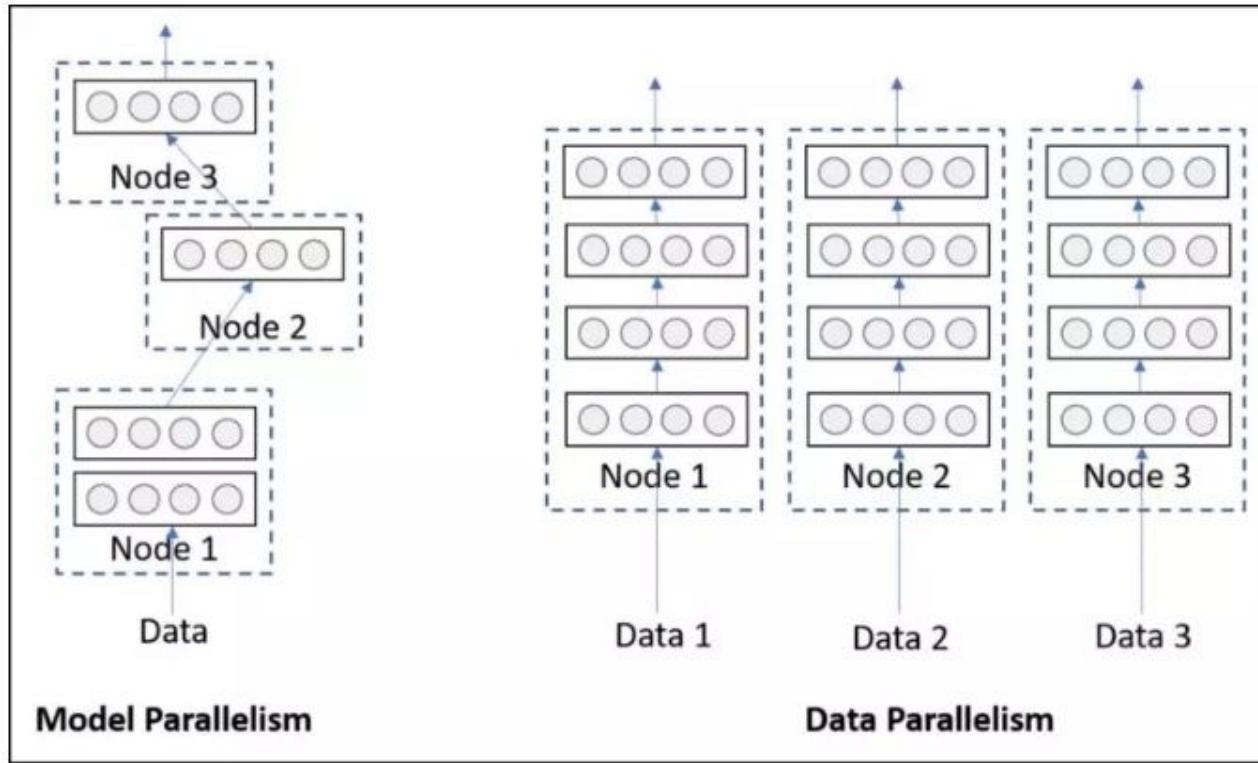
- [DataParallel](#) is single-process, multi-thread, and only works on a single machine; does not work with [model parallel](#) at this time.
- [DistributedDataParallel](#) is multi-process and works for both single-and multi-machine training; works with model parallel

Добавить DataParallel проще, но DistributedDataParallel работает быстрее

DataParallel



DataParallel vs ModelParallel



Recap

- качество различных нейросетевых архитектур сравнивают на открытых датасетах
- предобученные на открытых датасетах модели доступны для скачивания
- для решения практической задачи как правило адаптируют готовую архитектуру, предобученную на открытом датасете
- существует набор подходов, позволяющий повысить качество модели: аугментация, предобработка и нормализация входных данных, регуляризация модели, выбор функции активации

Полезные материалы

- [PyTorch Tutorials](#)
- [Torchvision - popular datasets, pertained architectures, and common image transformations](#)
- [Albumentations - one of the most popular libraries for augmentations](#)
- [150 line script that trains a ResNet-18 to ~94% accuracy on CIFAR-10](#)
- [Pretrained state-of-the-art classification models in PyTorch](#)
- [\(Generic\) EfficientNets for PyTorch](#)
- [ShakeDrop Regularization for Deep Residual Learning](#) - аугментации бывают не только на уровне входных данных