

# Лекция 5

Архитектуры CNN, аугментация, тюнинг

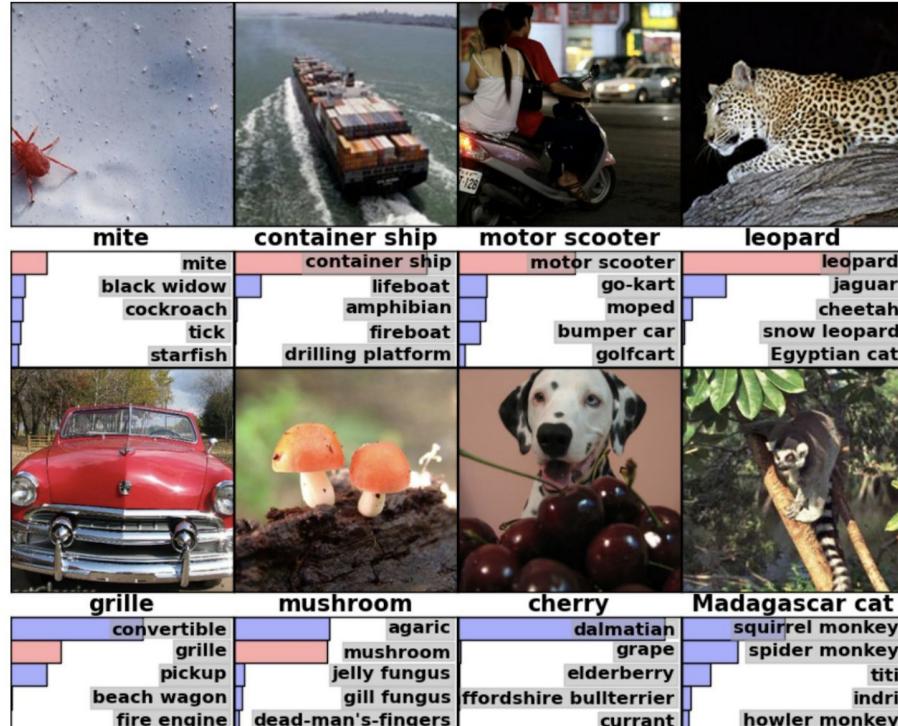
# План

- ImageNet
- Известные архитектуры
- Обучение сверточных сетей на практике

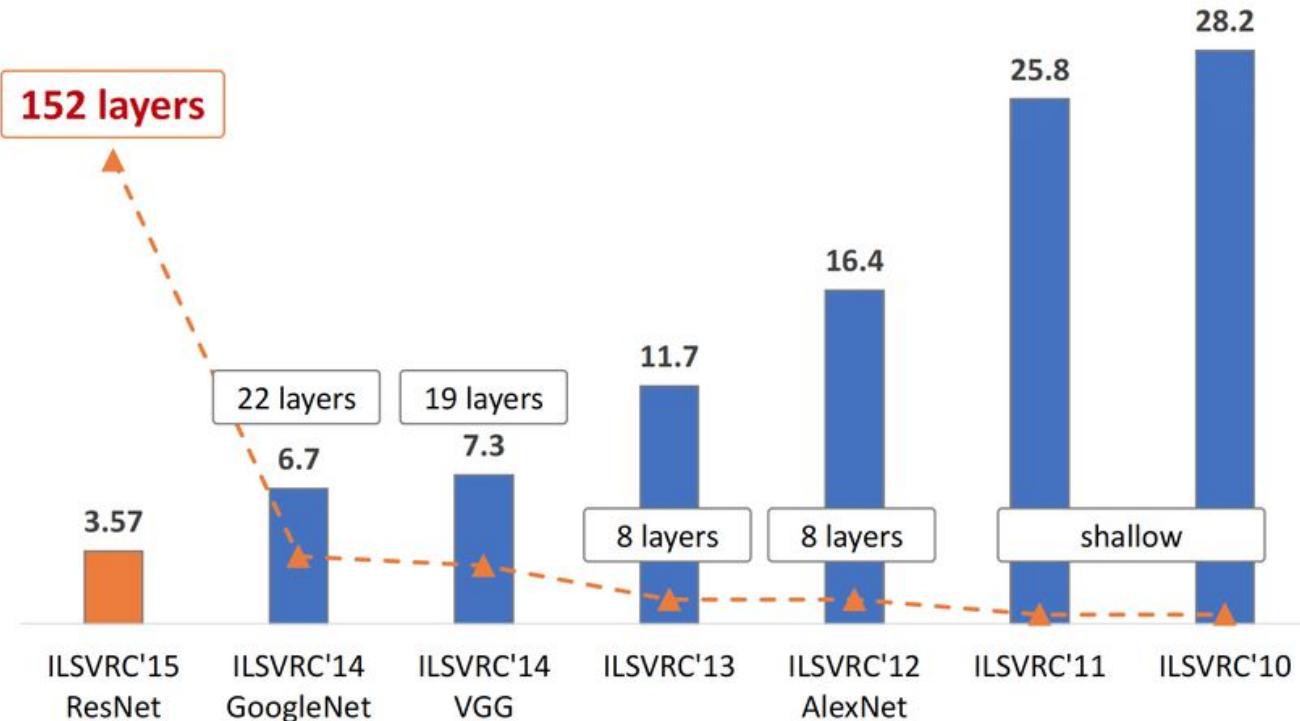
# ImageNet Challenge

IMAGENET

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.

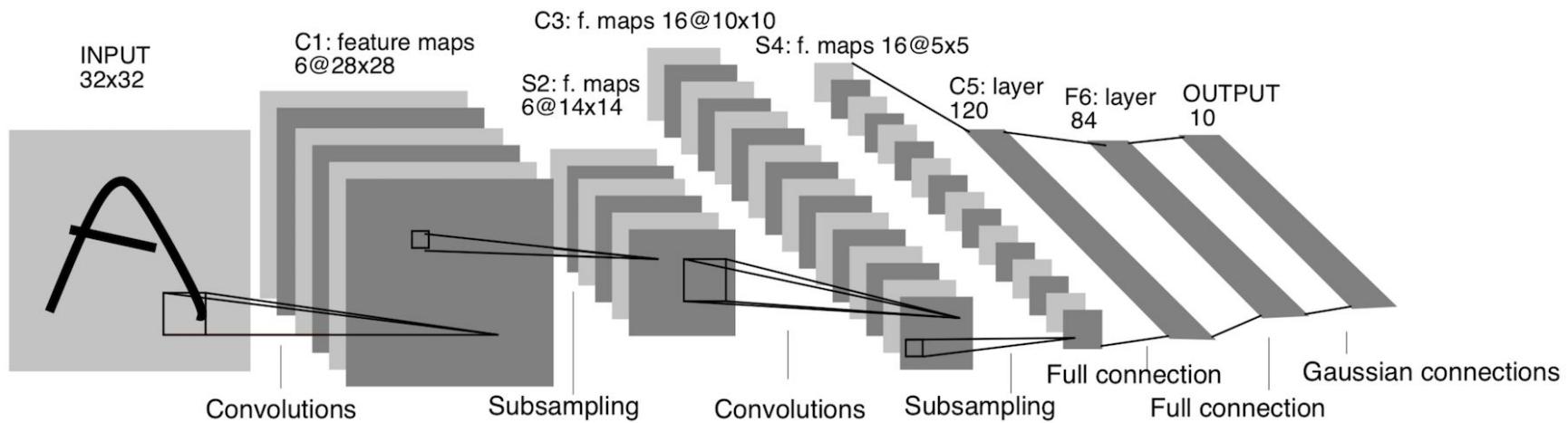


# ImageNet

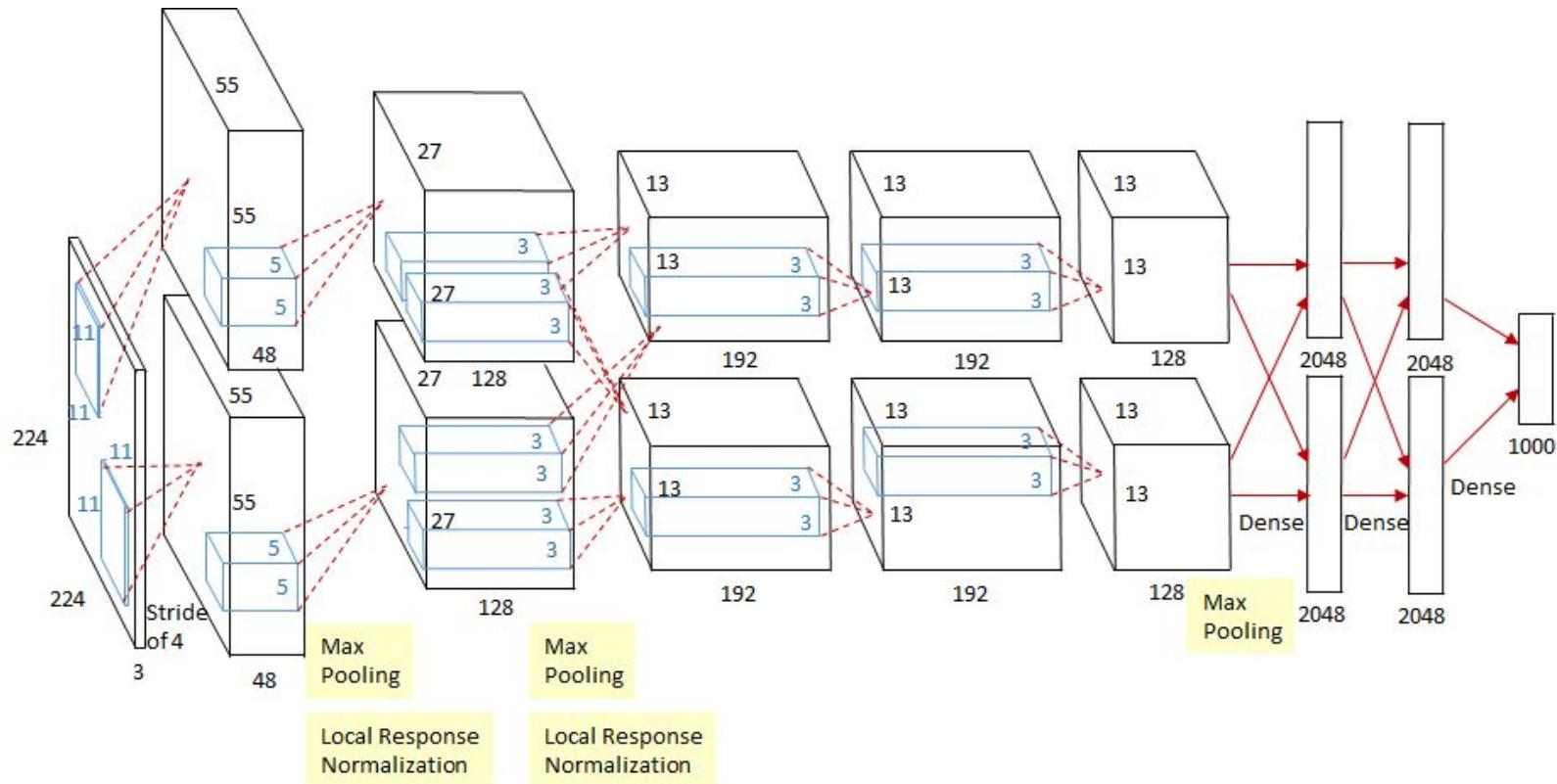


# Архитектуры сверточных сетей

# LeNet (1998)



# AlexNet (2012)



# AlexNet (2012)

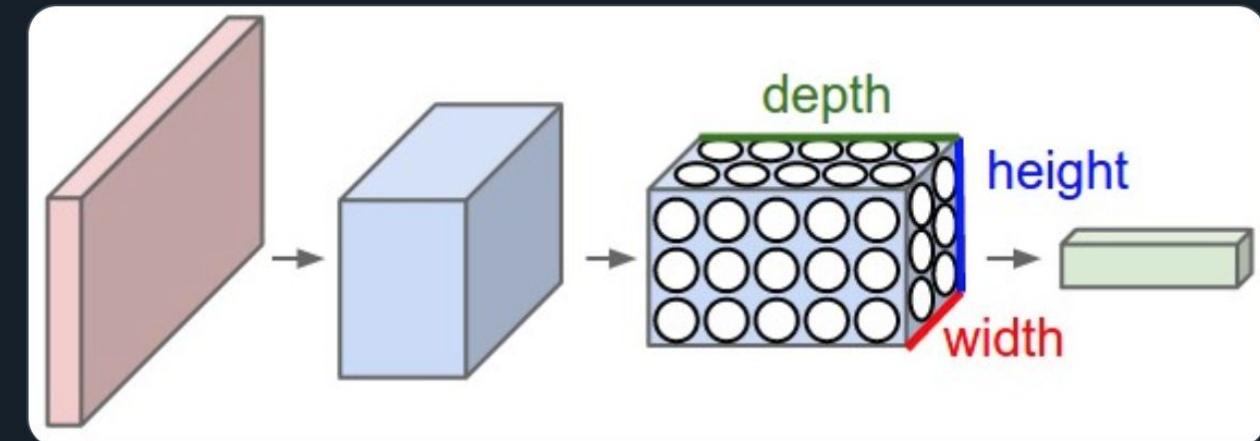
- параллельная архитектура
- использование ReLU в качестве функций активации
- для регуляризации использование Dropout перед полносвязными слоями
- число параметров ~ 60M



Andrej Karpathy ✅  
@karpathy

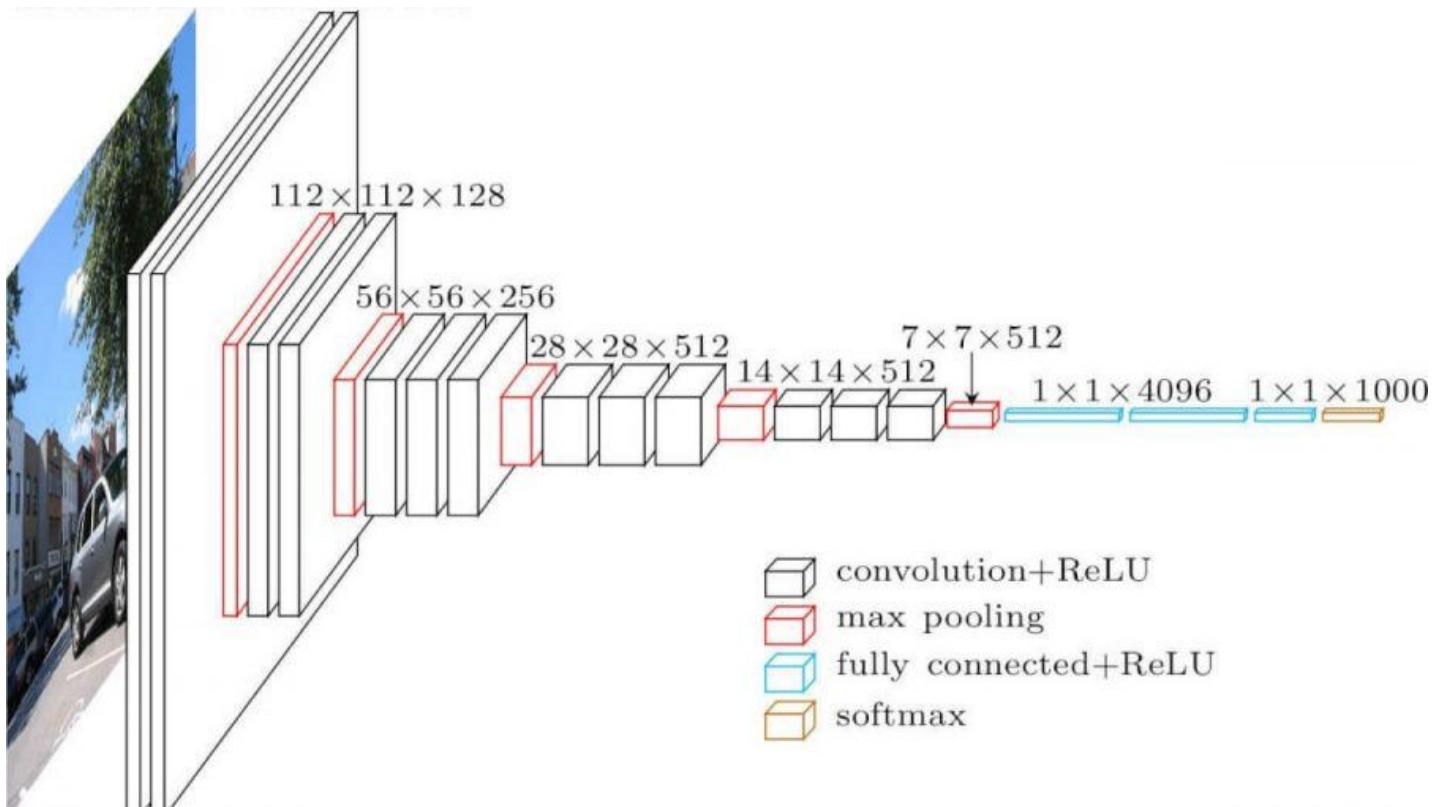
▼

This diagram, which I hastily sketched out in Google slides one day a few years ago at 3am, has become the most popular ugliest diagram of a ConvNet and, I regret to see, the #2 result in Google image search for "ConvNet". I am sorry 😞



9:49 AM · Jun 12, 2019 · Twitter Web Client

# VGG (2014)



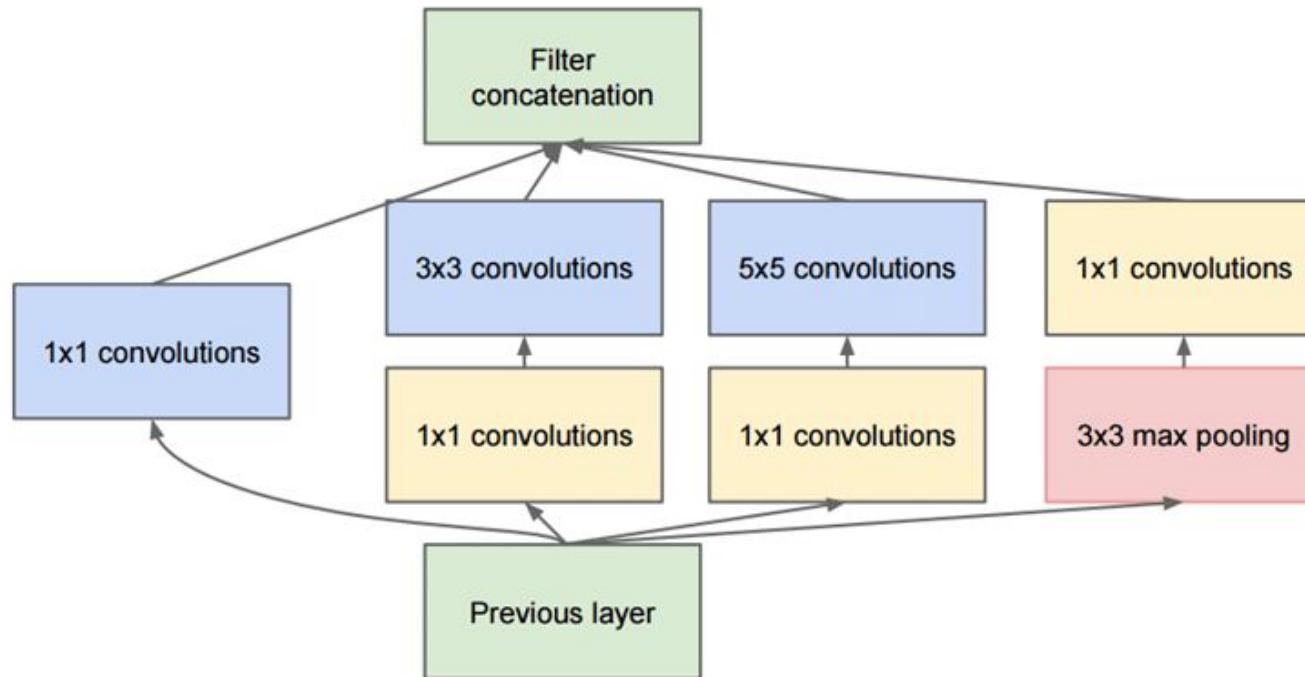
# VGG (2014)

- последовательно применяются свертки фильтров с небольшим размером ядра
- большой объем данных на выходе каждого слоя требует большого количества памяти
- число параметров  $\sim 130M$

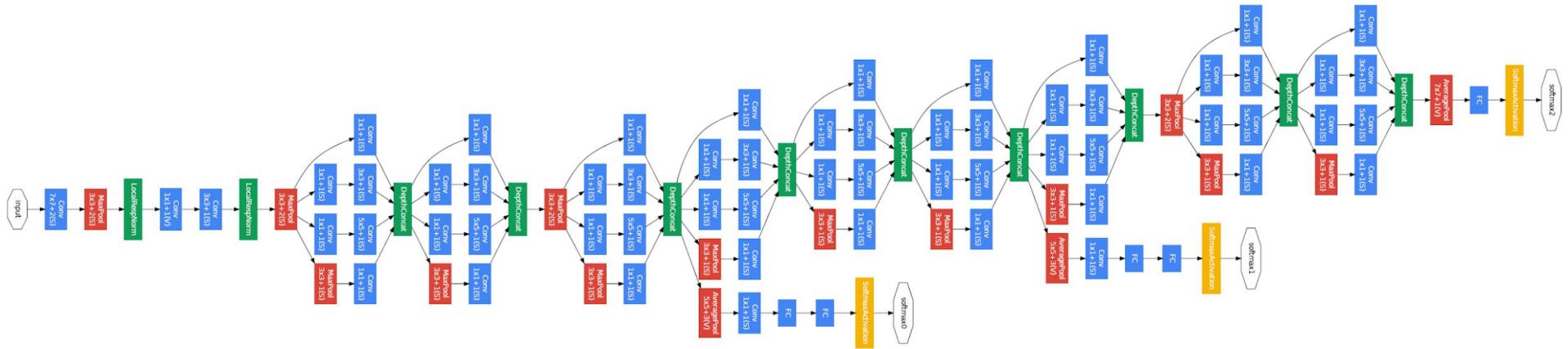
# GoogleNet (Inception) (2014)



# GoogleNet (Inception) (2014)



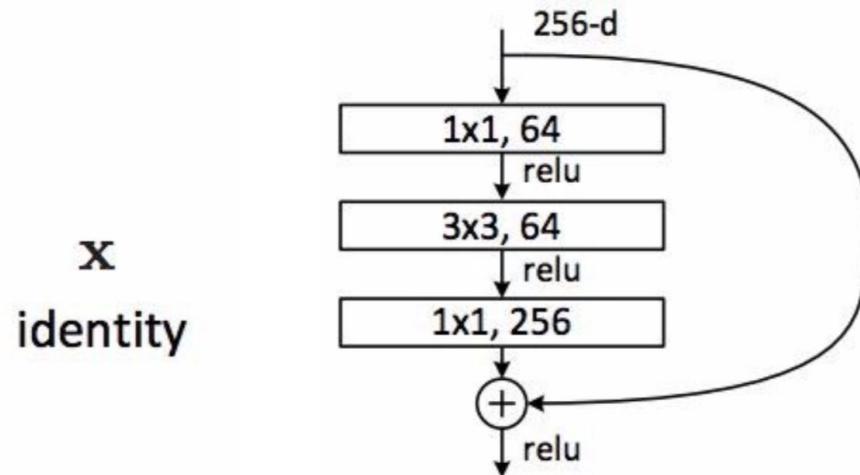
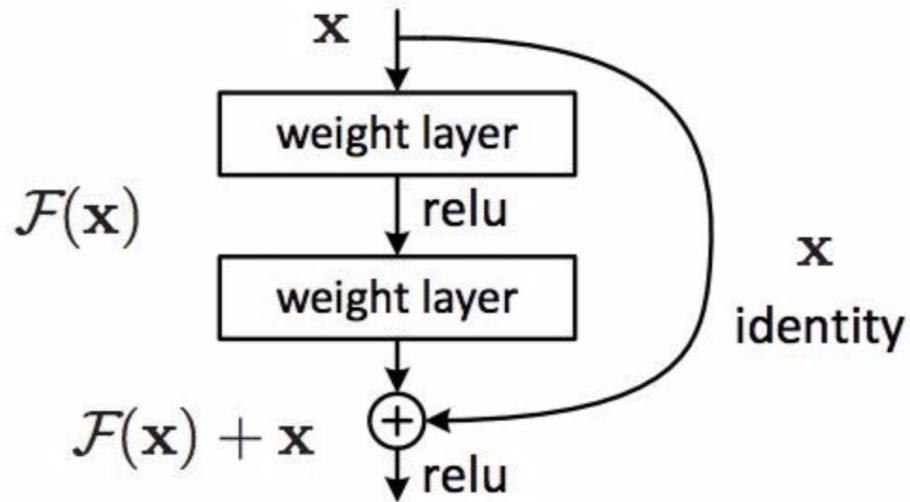
# GoogleNet (Inception) (2014)



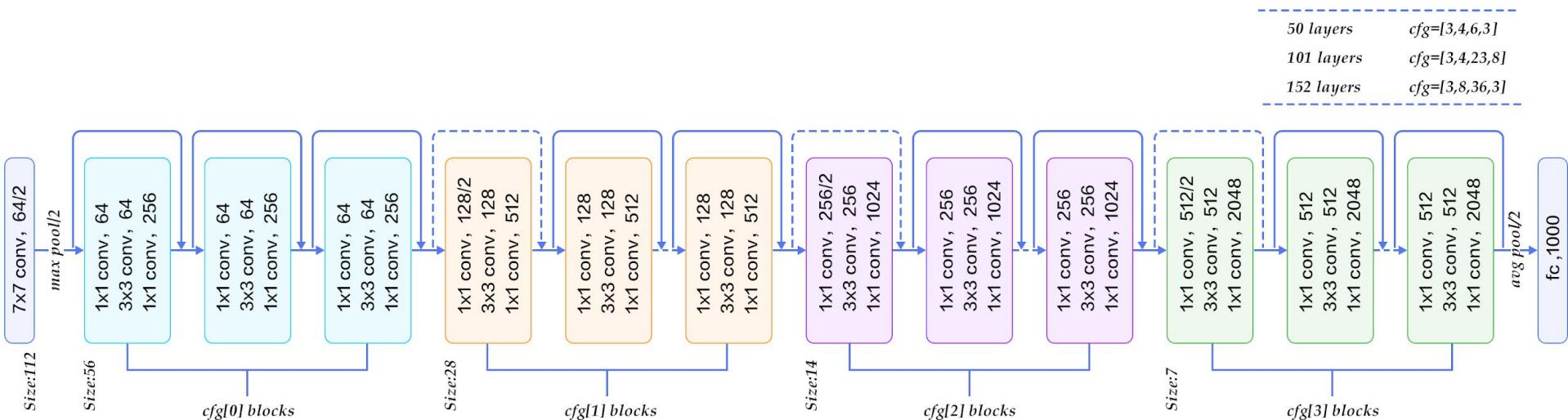
# GoogleNet (Inception) (2014)

- применяются фильтры разного размера к одним и тем же данным
- уменьшение размерности выхода (глубины) за счет свертки  $1 \times 1$
- в результате уменьшения глубины получаем ускорение сверточных ячеек  $3 \times 3$  и  $5 \times 5$
- число параметров  $\sim 7M$

# ResNet (2015)



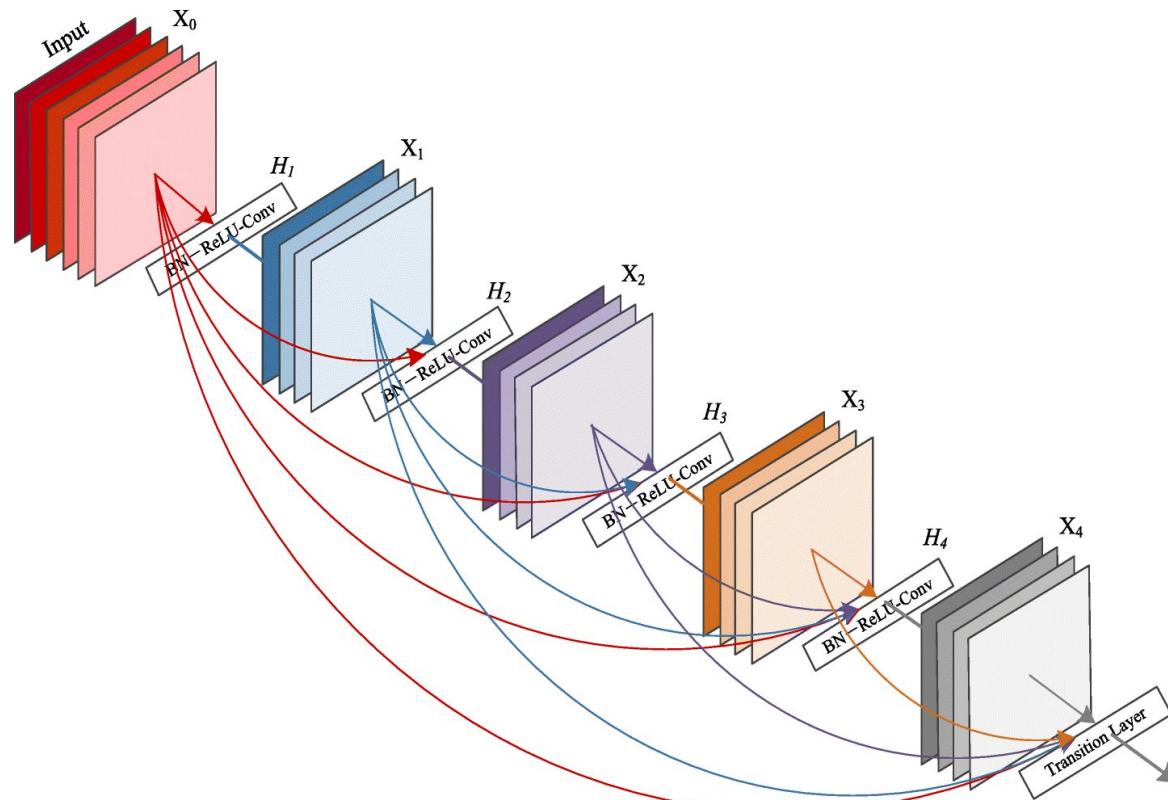
# ResNet (2015)



# ResNet (2015)

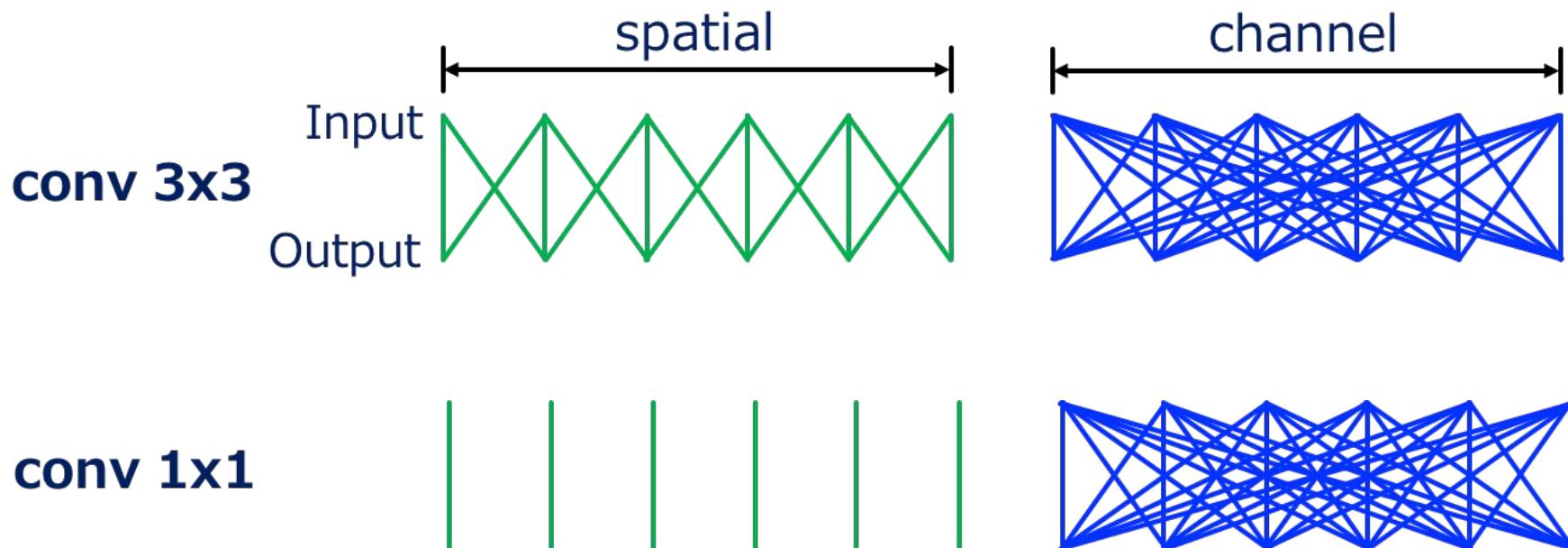
- добавлена параллельная ветка с исходными данными
- первая архитектура, у которой более 100 слоев
- число параметров ResNet 50 ~ 25M

# DenseNet (2016)



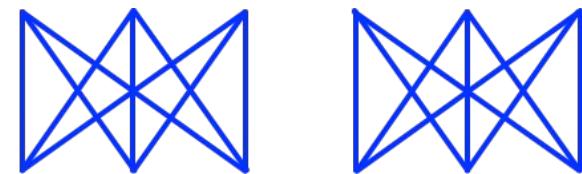
# Эффективные модели

# Convolutions

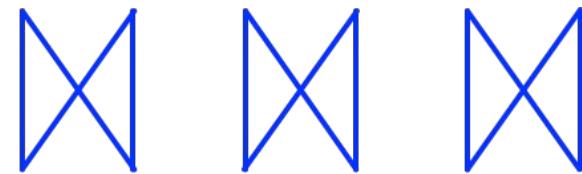
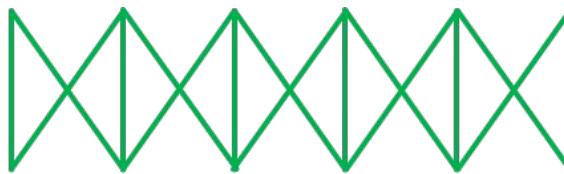


# Grouped convolutions

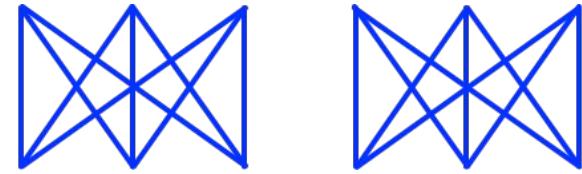
**gconv 3x3 (g=2)**



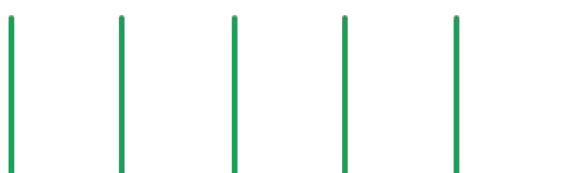
**gconv 3x3 (g=3)**



**gconv 1x1 (g=2)**



**gconv 1x1 (g=3)**



# Depthwise convolution

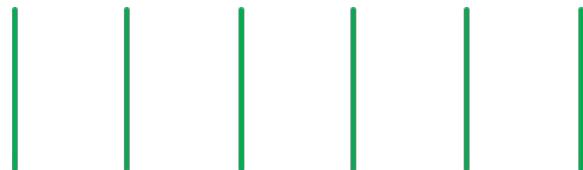
- Групповая свертка с числом групп = числу входных и выходных каналов

**depthwise conv**

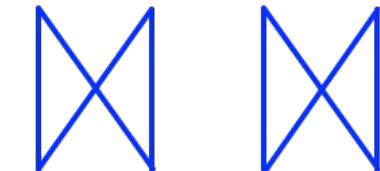


# ResNeXt (2016)

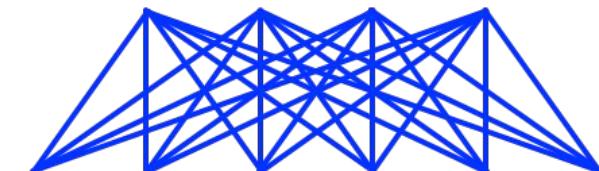
**conv 1x1**



**gconv 3x3**



**conv 1x1**

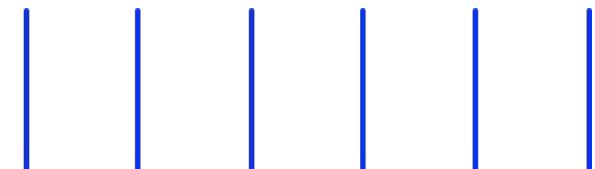
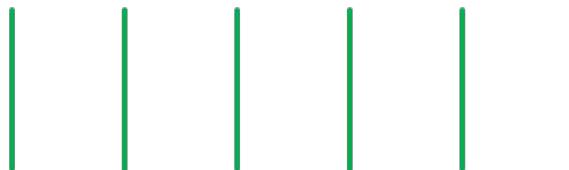


# MobileNet (Separable conv) (2017)

**depthwise conv**

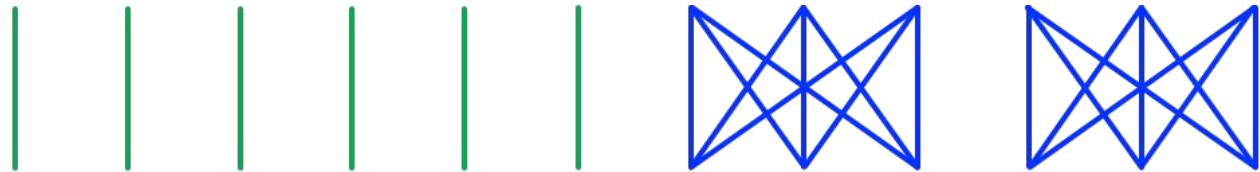


**conv 1x1**

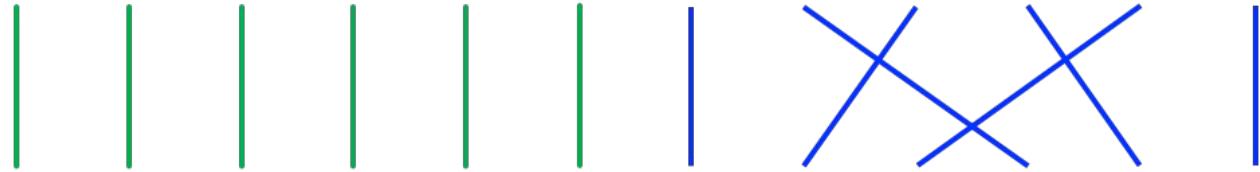


# ShuffleNet (2017)

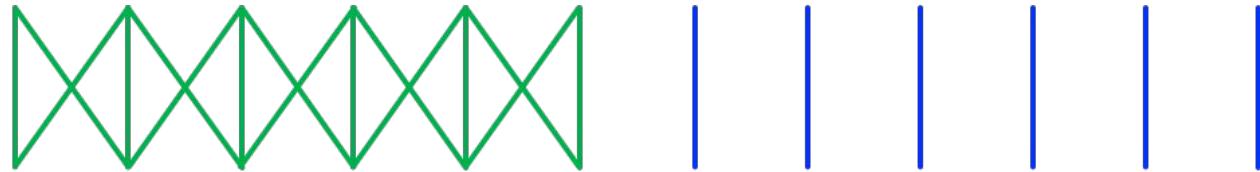
**gconv 1x1**



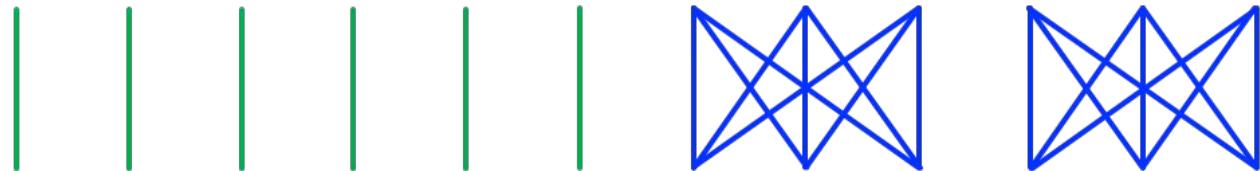
**c shuffle**



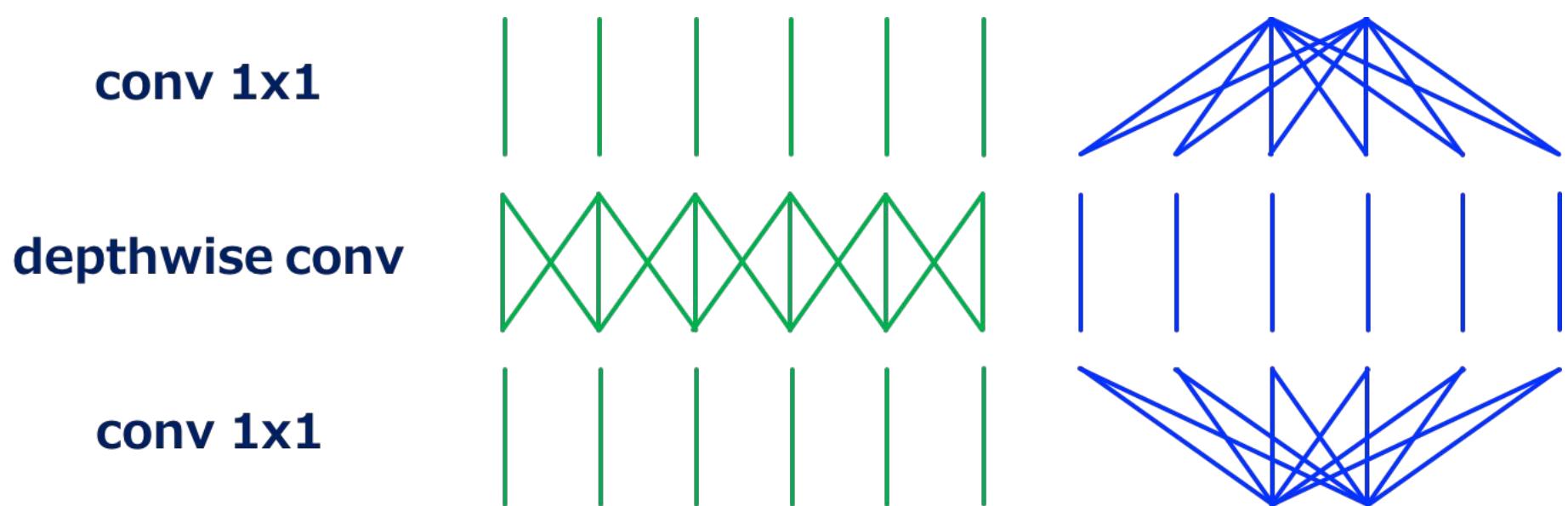
**depthwise conv**



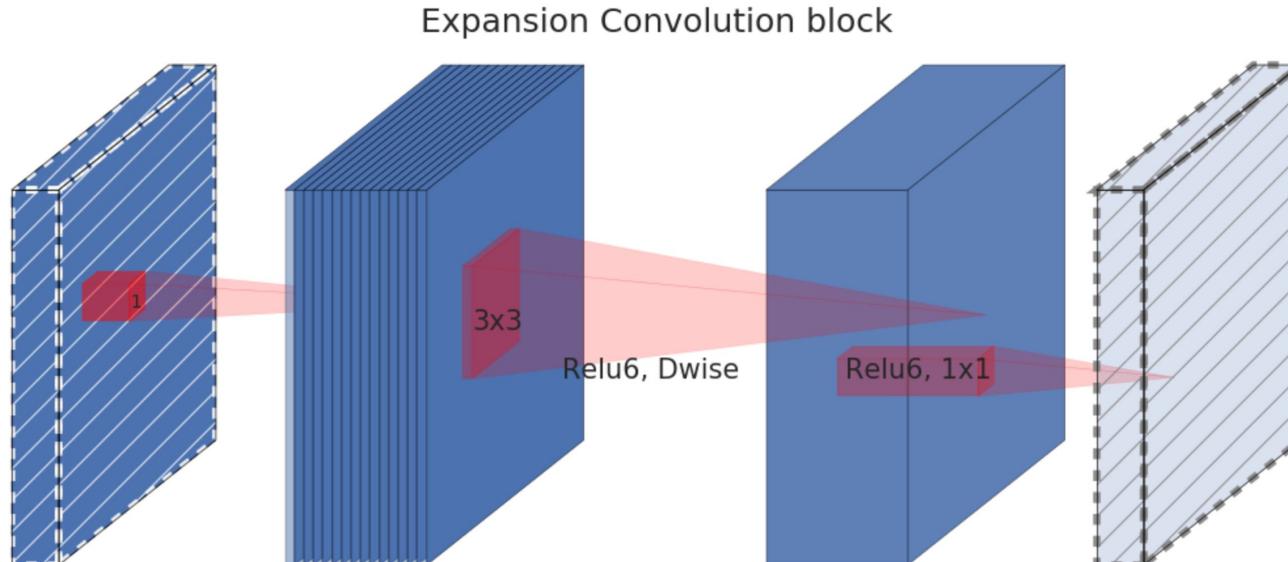
**gconv 1x1**



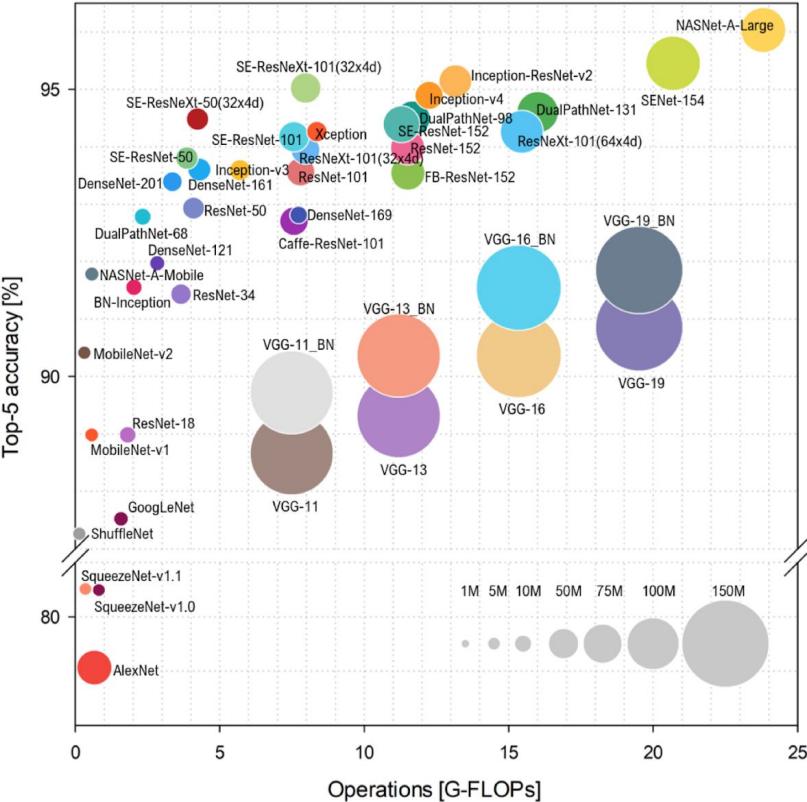
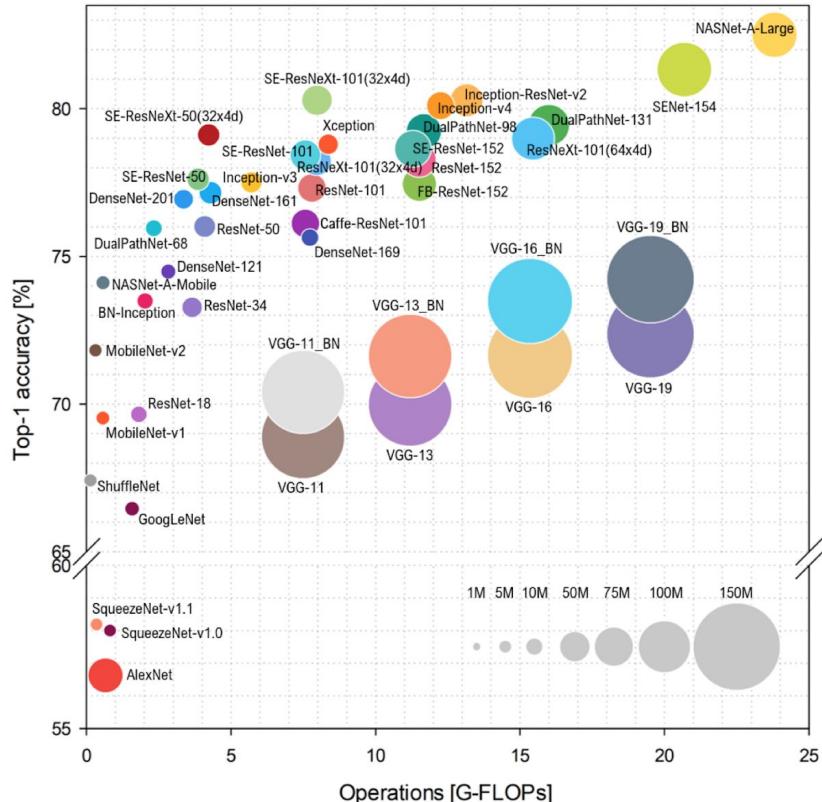
# MobileNet-v2 (2018)



# MobileNet-v2 (2018)



# Сравнение: качество vs скорость



# 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

	Задача похожа	Задача сильно отличается
Данных мало	Обучаем линейный классификатор на признаках с последнего скрытого слоя	Тут проблемы :) Обучаем классификатор на признаках с разных слоев
Данных много	Дообучаем несколько последних внутренних слоев	Фиксируем первые слои, остальные слои дообучаем

# Обучение CNN на практике. Баланс классов

# Баланс классов

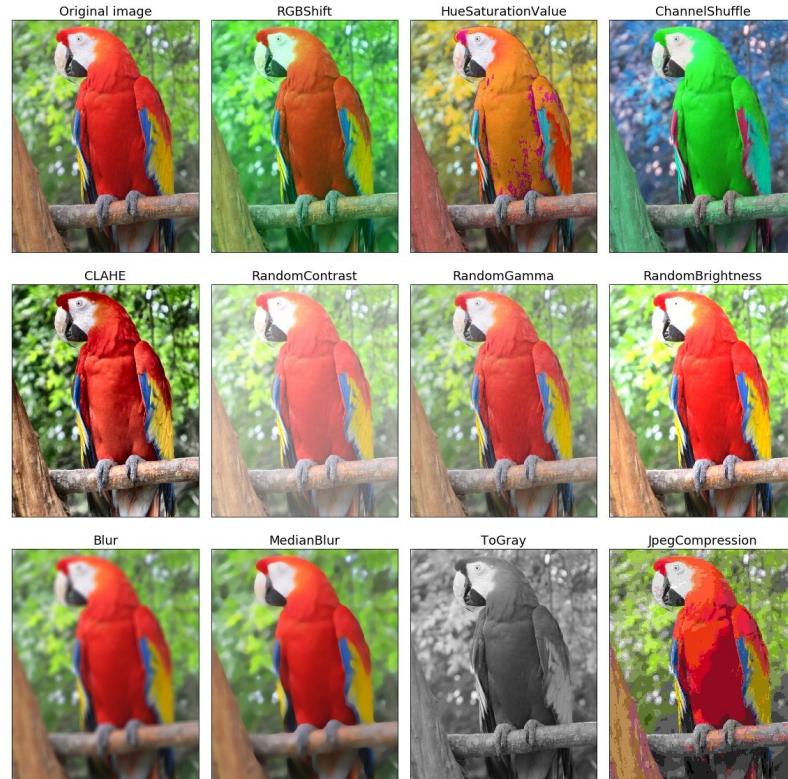
- часто на практике выборка не сбалансирована
- это приводит к тому, что модель переобучается на классы с большим числом примеров и дает смещеннное предсказание

# Баланс классов

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

# Обучение CNN на практике. Аугментация данных

# Аугментация данных



# Аугментация данных

Original



VerticalFlip



HorizontalFlip



RandomRotate90



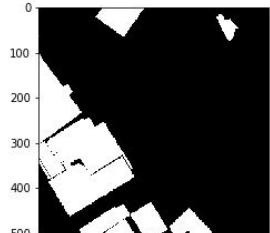
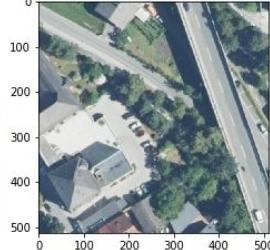
Transpose



ShiftScaleRotate



RandomSizedCrop



# Аугментация данных

- зеркальное отражение по горизонтали
- 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

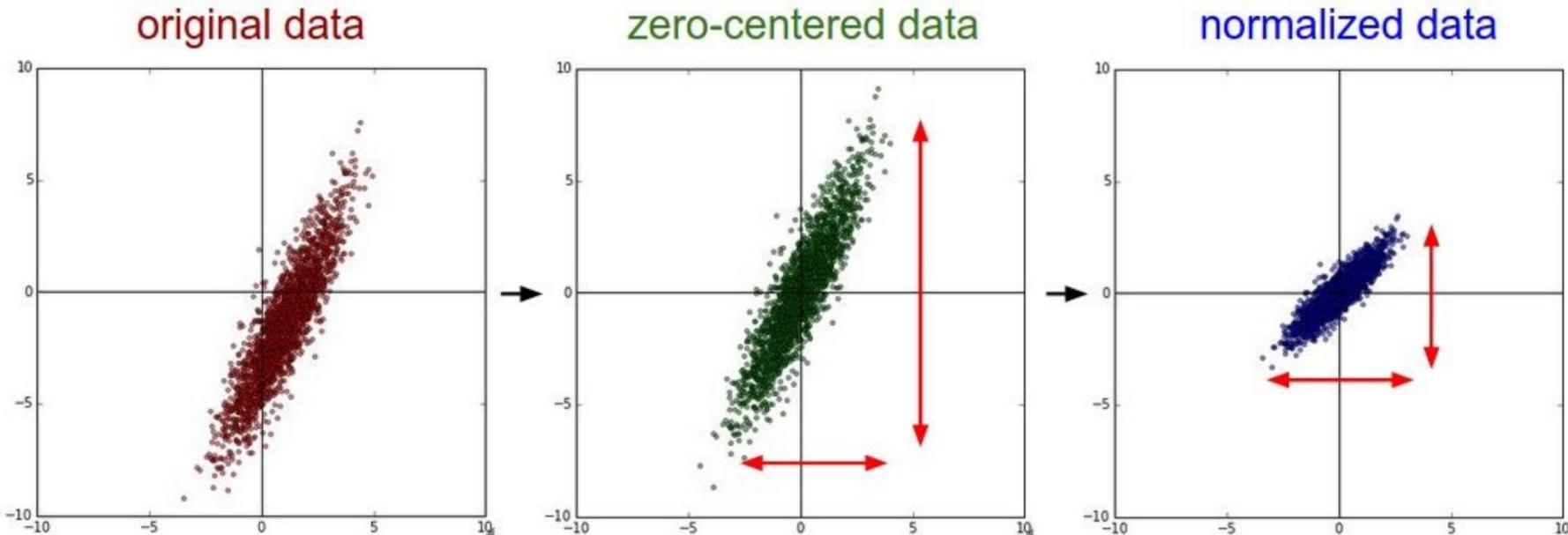
- на практике аугментированные изображения сильно увеличивают размер выборки
- хранить на диске аугментированные копии нецелесообразно
- процесс аугментации запускают на лету параллельно с обучением
- аугментация выполняется на CPU и существенно не влияет на скорость при обучении на GPU

# Библиотеки аугментации изображений

- Torchvision - легко в использовании
- [Albumentations](#) (мощная и популярная среди kaggleров библиотека)

# Обучение CNN на практике. Предобработка данных

# Центрирование и нормализация



# Обучение CNN на практике. Инициализация

# Инициализация весов

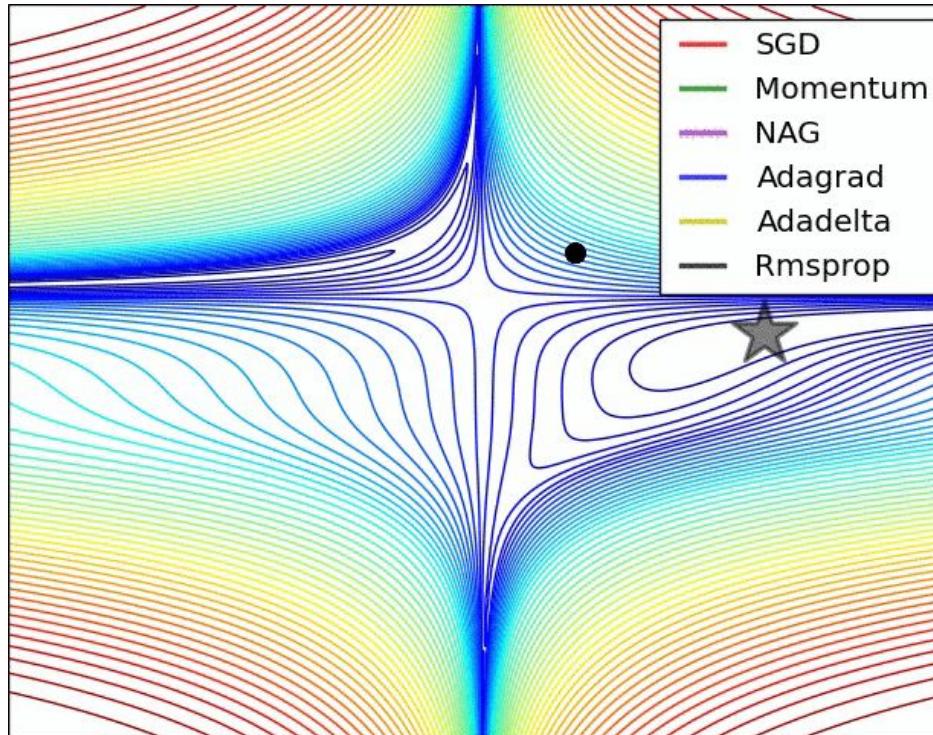
- инициализация различными весами (если нейроны возвращают одинаковые значения, значит и градиенты для них будут одинаковыми)
- инициализация случайными значениями с небольшой дисперсией (гаусс или нормальное - не существенно важно)
- [torch.nn.init](#)

# Обучение CNN на практике. Градиентный спуск

# Оптимизация (градиентный спуск)

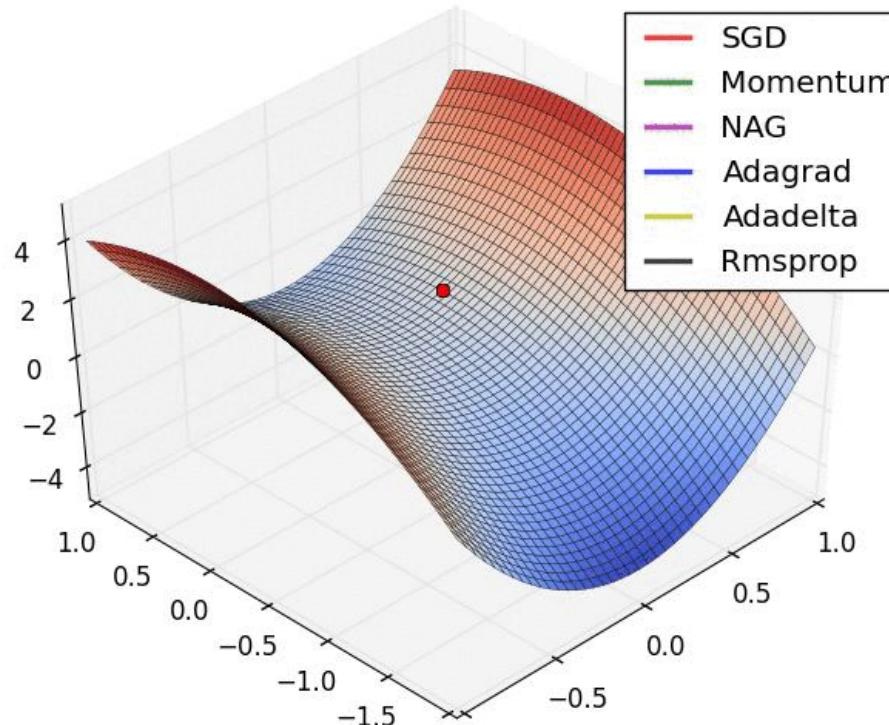
- при увеличении размера батча оценка градиента получается стабильнее, что позволяет увеличить learning rate
- в случае, если не происходит изменение метрики на валидации, стоит уменьшить значение learning rate ReduceLROnPlateau
- наиболее популярные оптимизаторы: SGD, Adam
- [torch.optim](#)

# Оптимизация (градиентный спуск)



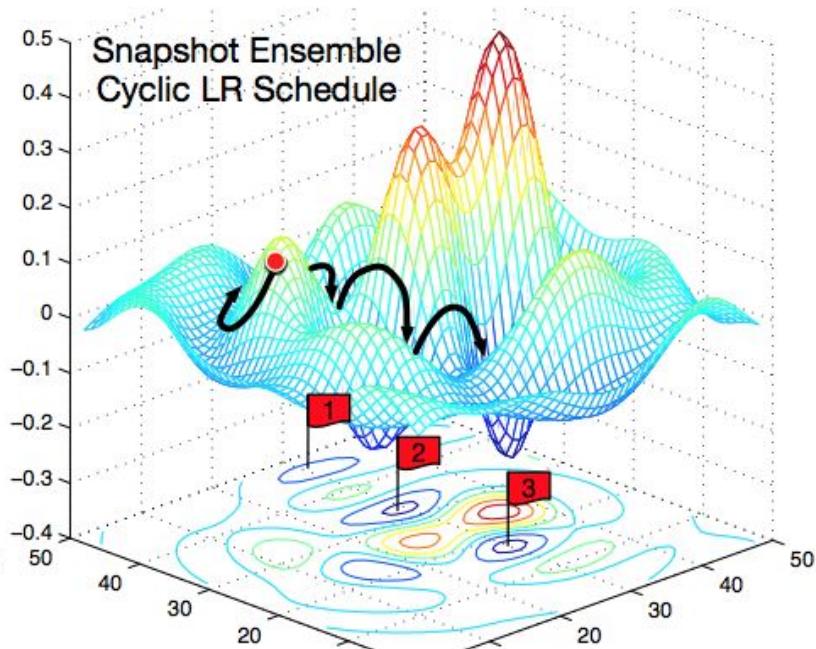
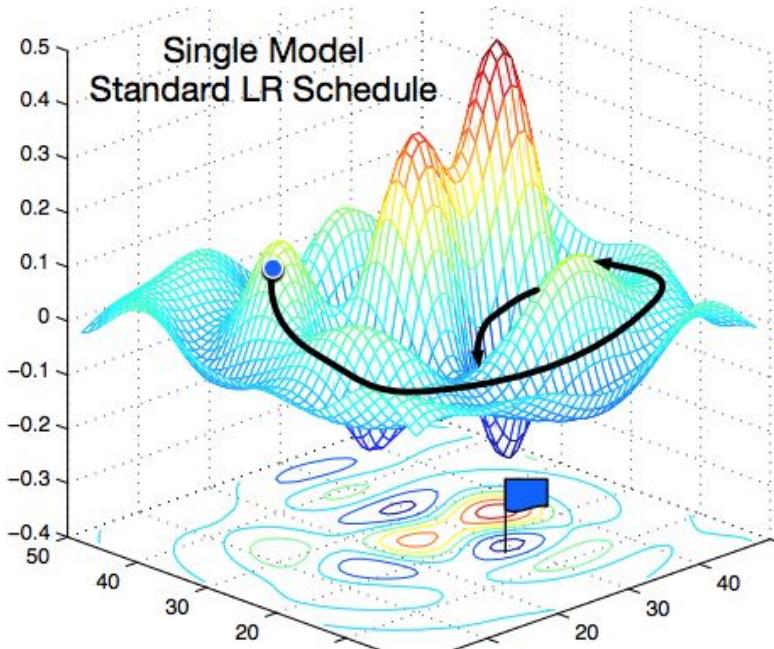
<http://ruder.io/optimizing-gradient-descent/>

# Оптимизация - седловая точка



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

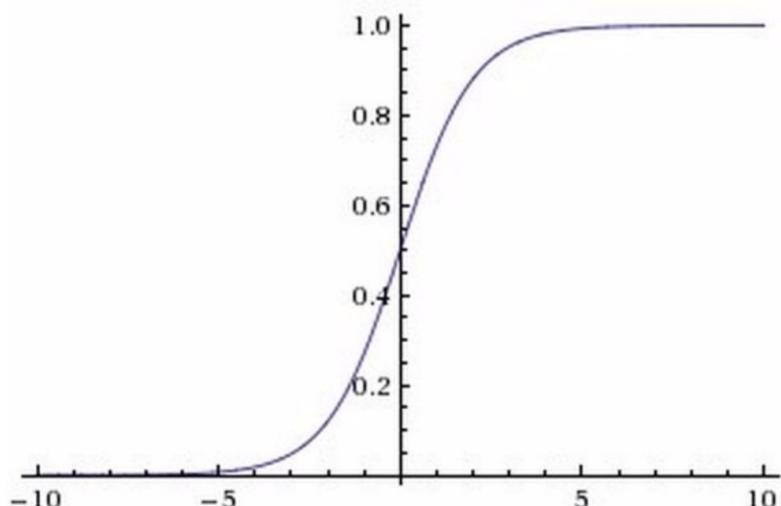
# Изменение LR по расписанию



# Обучение CNN на практике. Функции активации

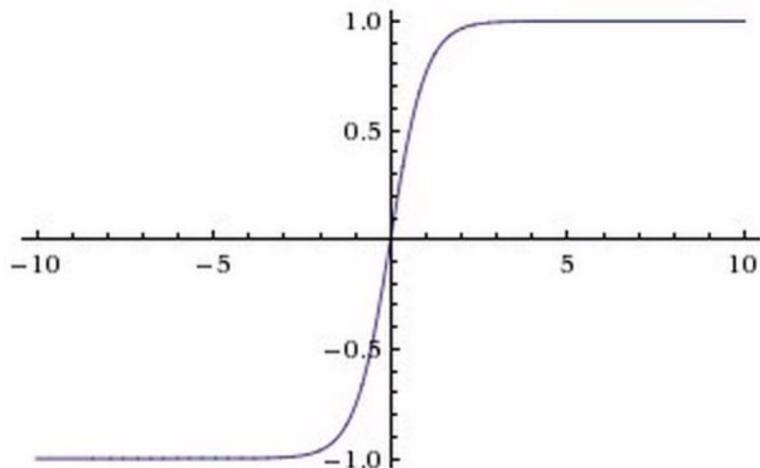
# Функции активации - sigmoid

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



- убивает градиент
- смещена относительно нуля - проблема для обучения
- используется на выходном слое

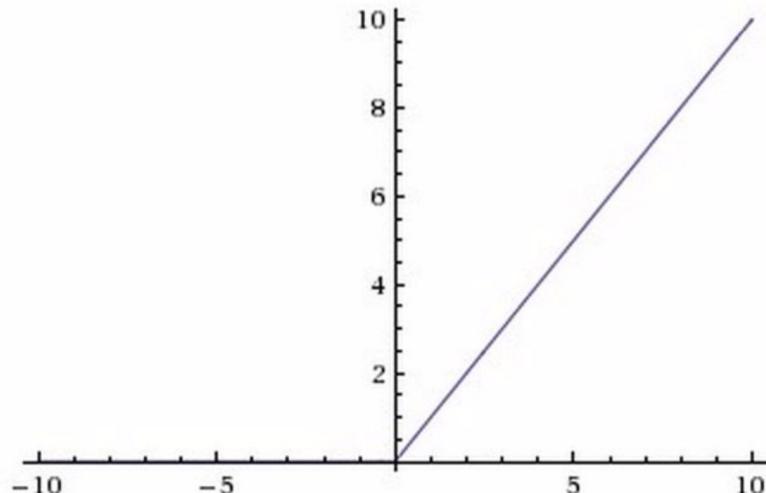
# Функции активации - $\tanh$



- остается проблема с градиентами
- решена проблема с центрированием

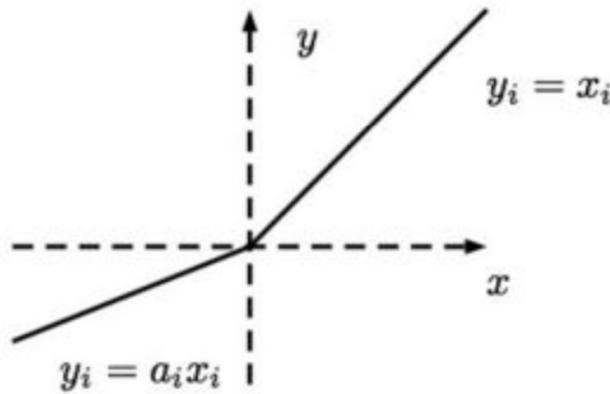
# Функции активации - ReLU

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

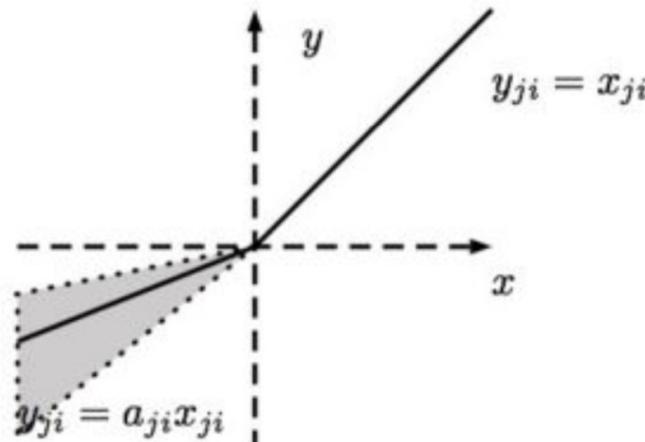


- простая в реализации
- отсутствие насыщения ускоряет процесс сходимости
- при большом значении градиента значение может уйти в минус и не вернуться dying ReLU

# Функции активации - Parametric LU



Leaky ReLU/PReLU

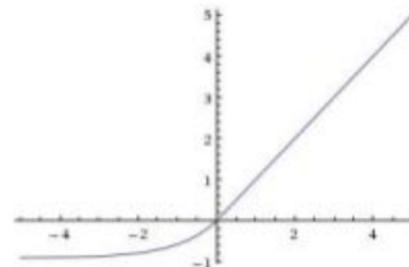
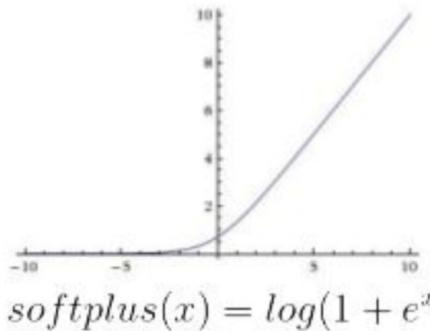


Randomized Leaky ReLU

- угол наклона является обучаемым параметром
- угол наклона изменяется случайным образом

# Функции активации - Exponential LU

Softplus and Exponential Linear Unit (ELU)



- решена неоднозначность с градиентом в области нуля

# 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	<b>0.4042</b>
PReLU	0.0633	0.4163
RReLU	0.1141	<b>0.4025</b>

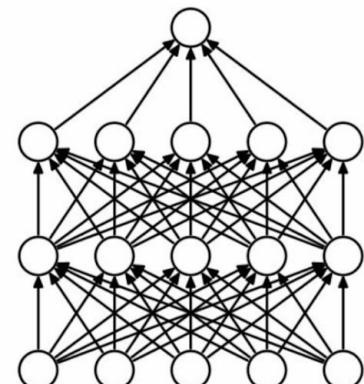
Table 4. Error rate of CIFAR-100 Network in Network with different activation function

# Обучение CNN на практике. Регуляризация

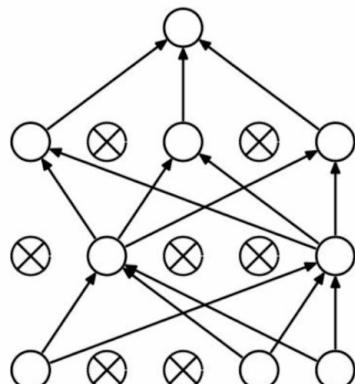
# Регуляризация

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

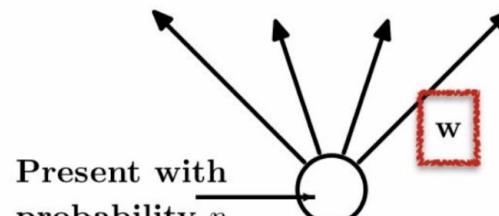
# Регуляризация - Dropout



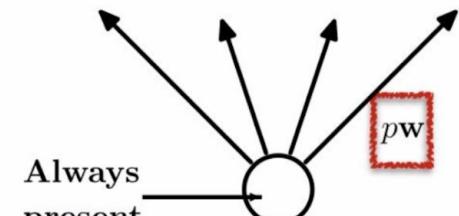
(a) Standard Neural Net



(b) After applying dropout.



(c) At training time



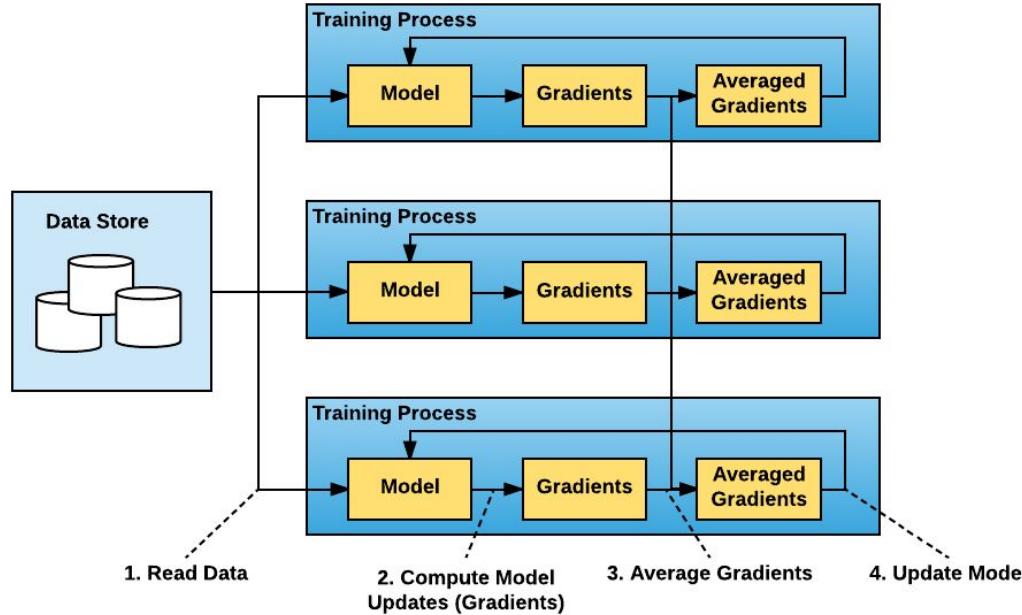
(d) At test time

# Обучение CNN на практике. Оптимизация вычислений

# Параметры фильтров

- размеры изображения кратны числу сэмплирующих слоев (Pooling)
- 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



[https://pytorch.org/tutorials/intermediate/model\\_parallel\\_tutorial.html](https://pytorch.org/tutorials/intermediate/model_parallel_tutorial.html)

# Резюме

- 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, использование ReLU в качестве функции активации