

# Основы глубинного обучения

Лекция 5

Оптимизация в глубинном обучении. Свёрточные архитектуры.

Евгений Соколов

[esokolov@hse.ru](mailto:esokolov@hse.ru)

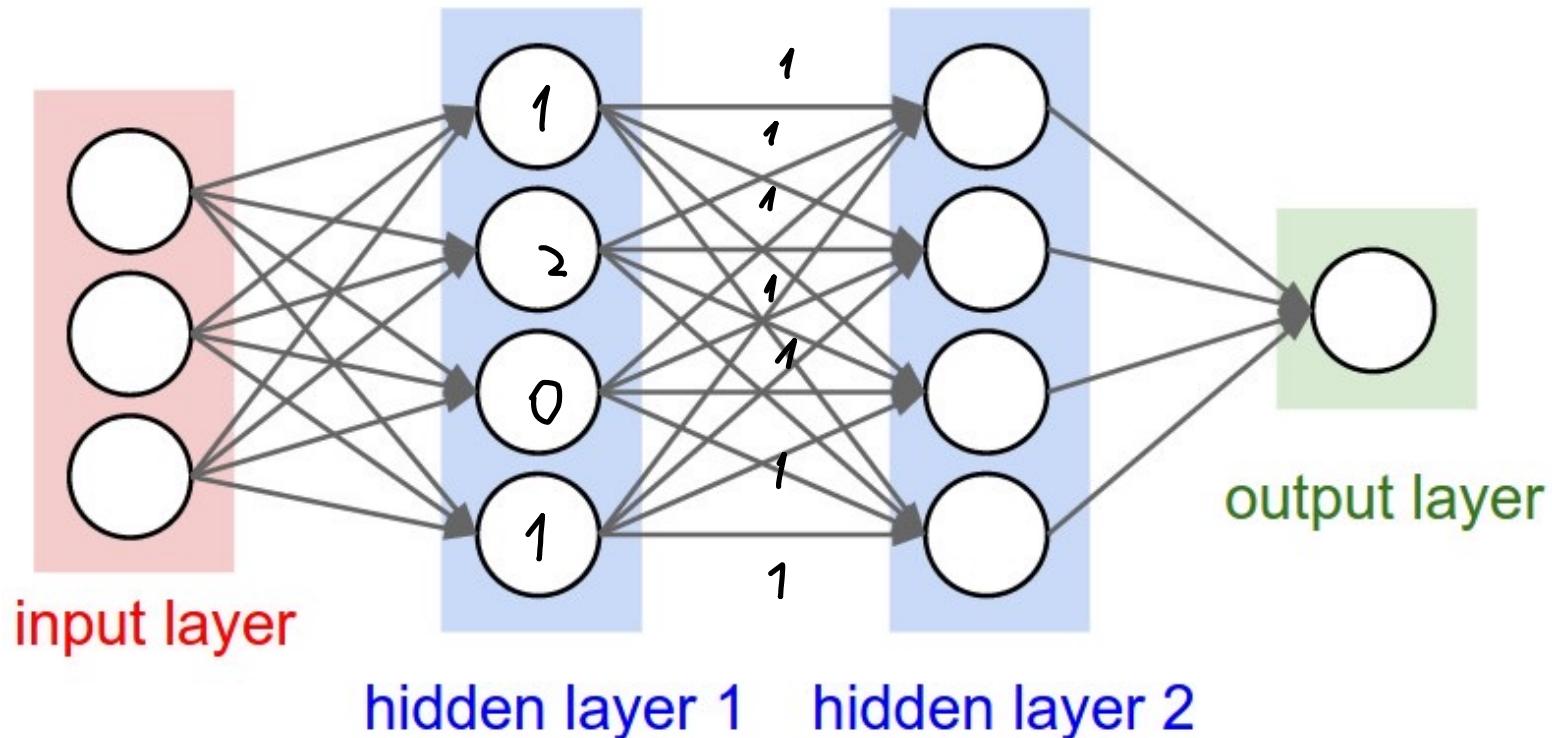
НИУ ВШЭ, 2023

# Нормализации

# Internal covariate shift

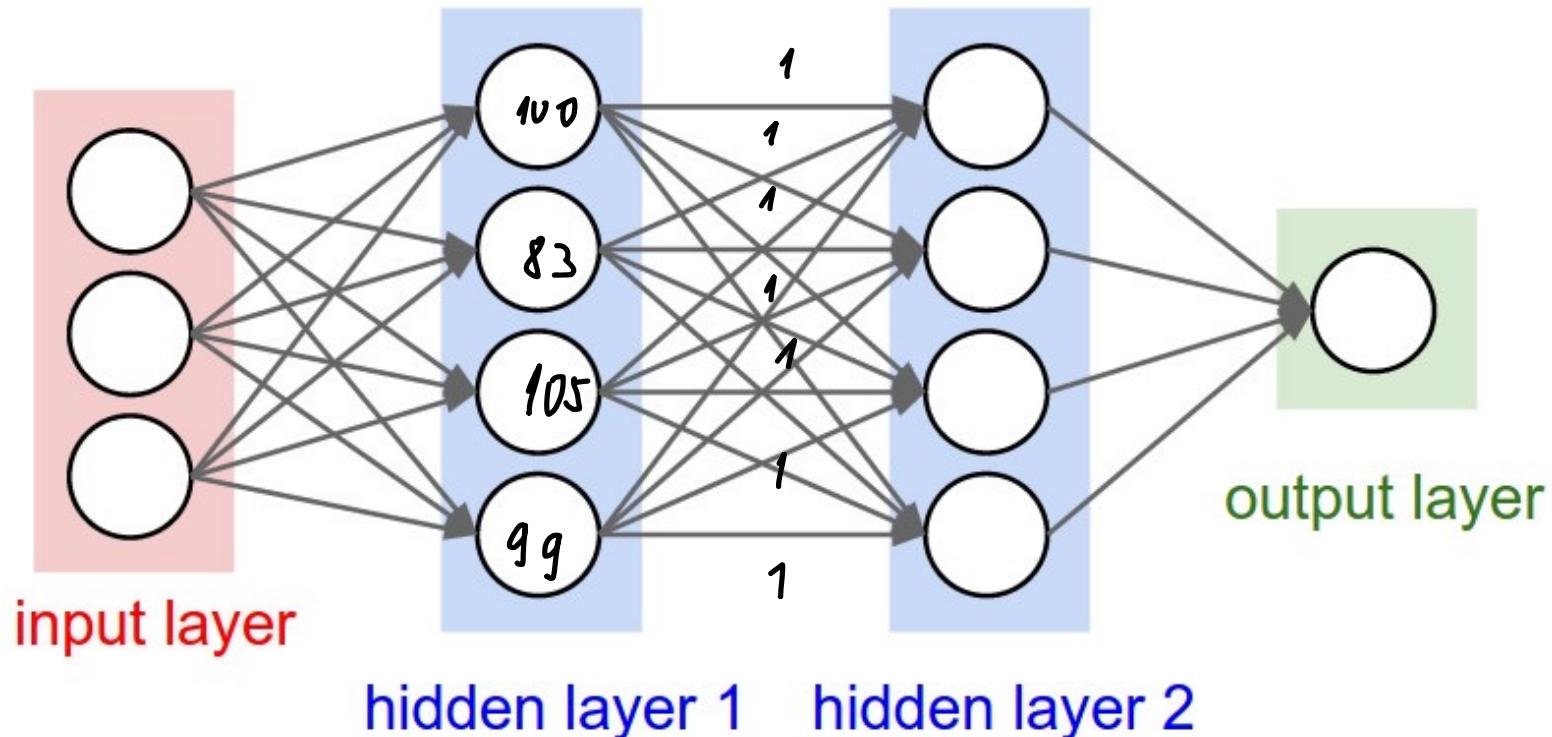
- В нейронной сети каждый слой обучается на выходах предыдущих слоёв
- Если слой в начале сильно меняется, то все следующие слои надо переделывать

# Internal covariate shift



# Internal covariate shift

Допустим, веса первого слоя сильно поменялись после градиентного шага



# Internal covariate shift

- Идея: преобразовывать выходы слоёв так, чтобы они гарантированно имели фиксированное распределение

# Batch Normalization

- Реализуется как отдельный слой
- Вычисляется для текущего батча
- Оценим среднее и дисперсию каждой компоненты входного вектора:

$$\mu_B = \frac{1}{n} \sum_{j=1}^n x_{B,j}$$

покоординатно

$$\sigma_B^2 = \frac{1}{n} \sum_{j=1}^n (x_{B,j} - \mu_B)^2$$

$x_{B,j}$  —  $j$ -й объект в батче  $B$

# Batch Normalization

- Отмасштабируем все выходы:

$$\tilde{x}_{B,j} = \frac{x_{B,j} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

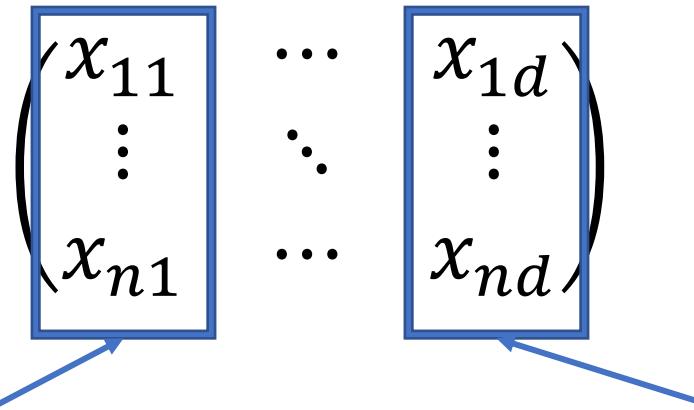
- Зададим нужные нам среднее и дисперсию:

$$z_{B,j} = \gamma \circ \tilde{x}_{B,j} + \beta$$



обучаемые параметры (векторы, размерность  
равна размерности входных векторов)

# Batch Normalization



Приводим среднее и  
дисперсию к  $\beta_1$  и  $\gamma_1$

Приводим среднее и  
дисперсию к  $\beta_d$  и  $\gamma_d$

- $n$  — размер батча
- $d$  — размерность входного вектора

# Batch Normalization

Важно: после BatchNorm среднее и дисперсия каждого выхода зависят только от параметров нормализации, но не от параметров прошлых слоёв!

# Batch Normalization

Во время применения нейронной сети:

- Те же самые формулы, но вместо  $\mu_B$  и  $\sigma_B^2$  используем их средние значения по всем батчам

# Batch Normalization

- Обычно вставляется между полносвязным/свёрточным слоём и нелинейностью
- Позволяет увеличить длину шага в градиентном спуске
- Не факт, что действительно устраняет covariance shift

# В чём польза от BatchNorm?

---

## How Does Batch Normalization Help Optimization?

---

**Shibani Santurkar\***  
MIT  
[shibani@mit.edu](mailto:shibani@mit.edu)

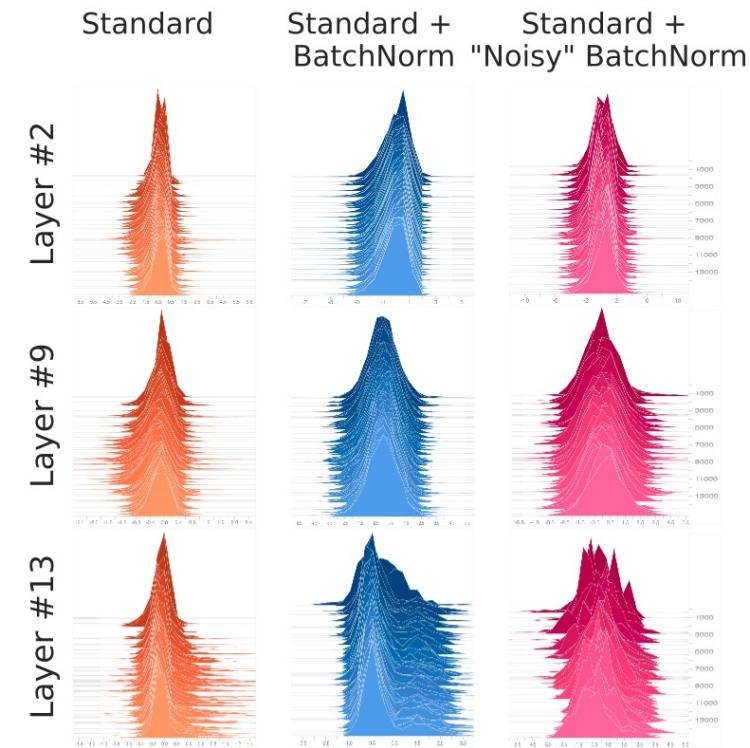
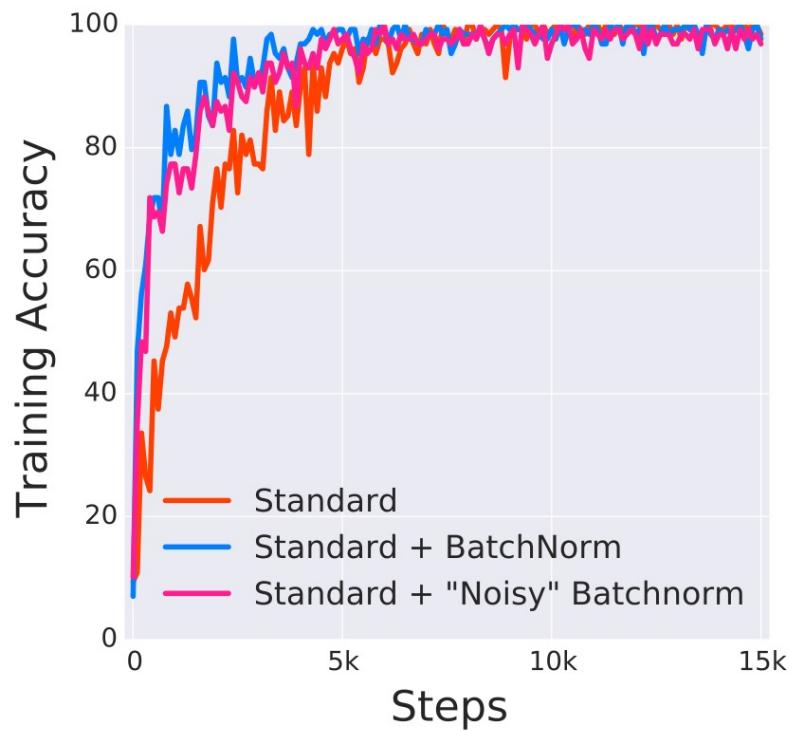
**Dimitris Tsipras\***  
MIT  
[tsipras@mit.edu](mailto:tsipras@mit.edu)

**Andrew Ilyas\***  
MIT  
[ailyas@mit.edu](mailto:ailyas@mit.edu)

**Aleksander Mądry**  
MIT  
[madry@mit.edu](mailto:madry@mit.edu)

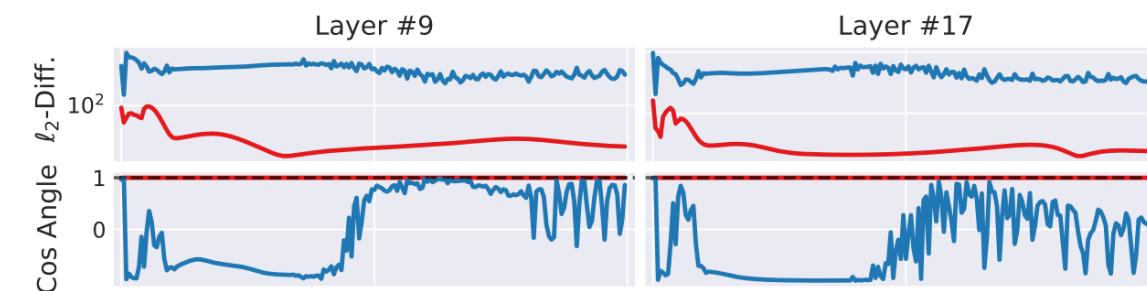
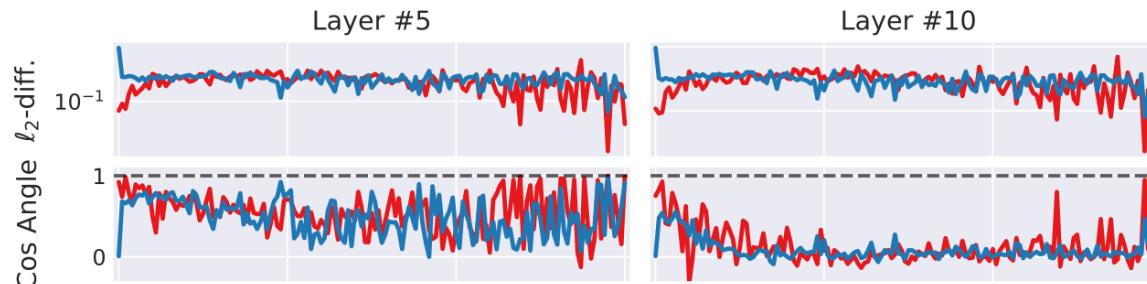
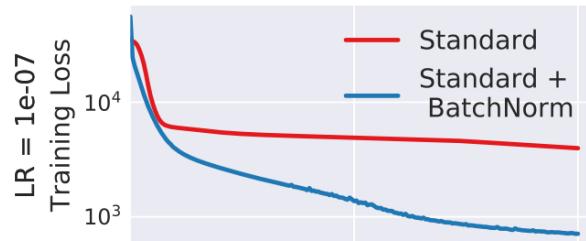
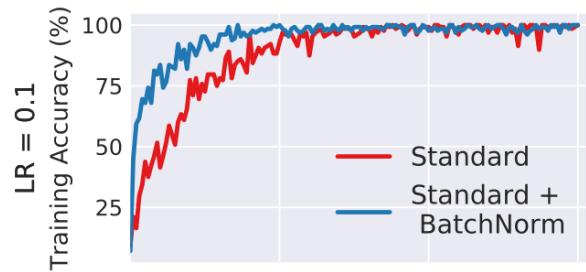
# В чём польза от BatchNorm?

- Добавим шум после нормализации — хуже не становится!



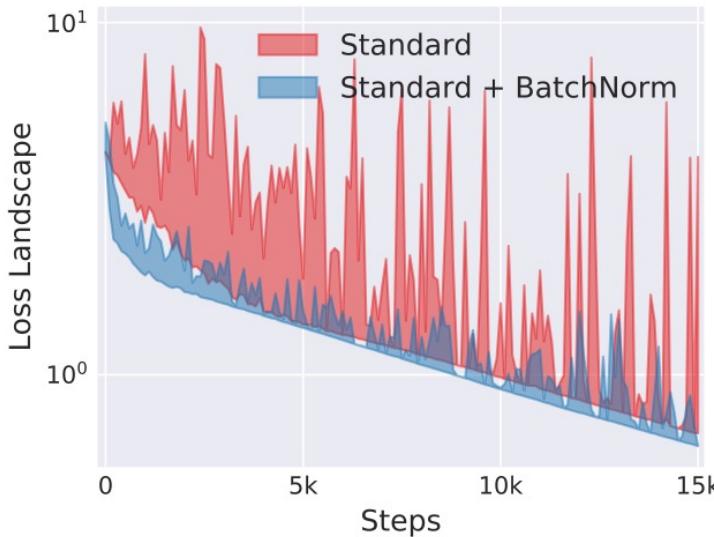
# В чём польза от BatchNorm?

- Как связаны градиенты на соседних итерациях?

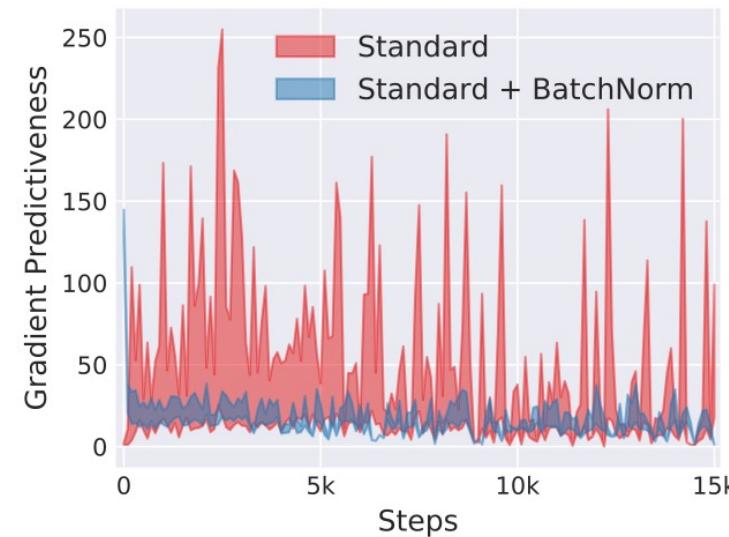


# В чём польза от BatchNorm?

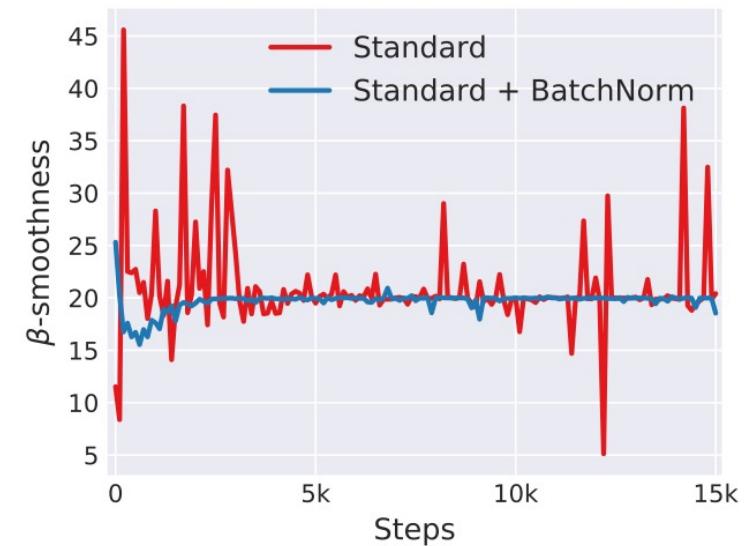
- Функционал ошибки становится более «гладким»!



(a) loss landscape

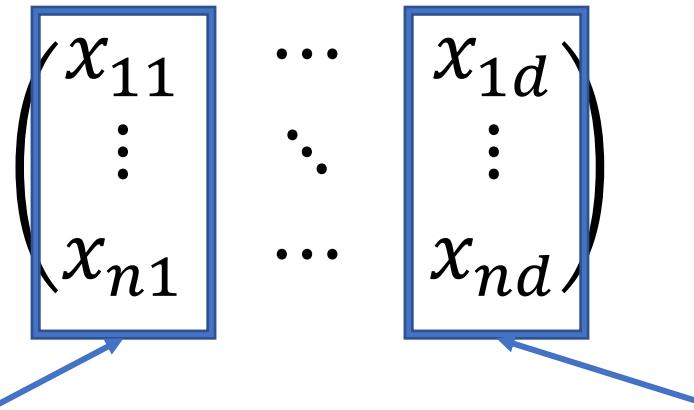


(b) gradient predictiveness



(c) “effective”  $\beta$ -smoothness

# Batch Normalization

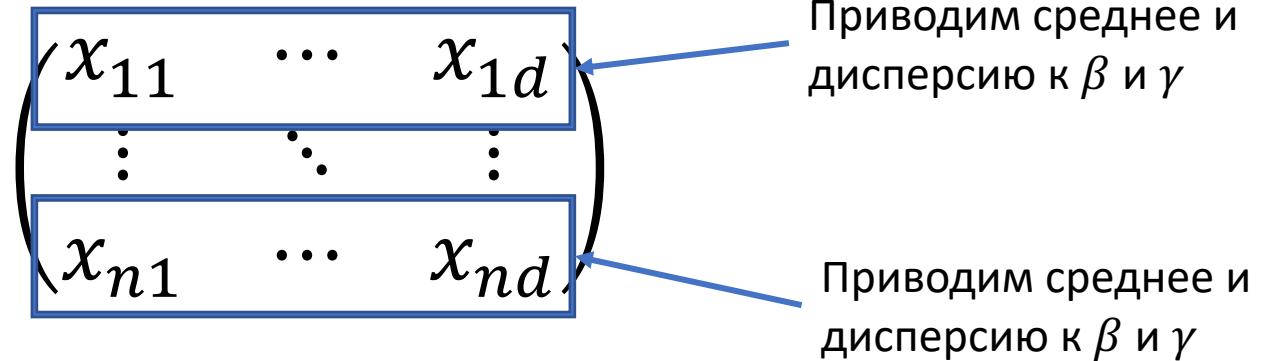


Приводим среднее и  
дисперсию к  $\beta_1$  и  $\gamma_1$

Приводим среднее и  
дисперсию к  $\beta_d$  и  $\gamma_d$

- $n$  — размер батча
- $d$  — размерность входного вектора

# Layer Normalization



- $n$  — размер батча
- $d$  — размерность входного вектора

# Layer Normalization

- Нормализуем распределение «признаков» одного объекта

$$\mu_i = \frac{1}{d} \sum_{j=1}^d x_{ij}$$

$$\sigma_i^2 = \frac{1}{d} \sum_{j=1}^d (x_{ij} - \mu)^2$$

$x_{ij}$  —  $j$ -й признак  $i$ -го объекта

# Layer Normalization

- Отмасштабируем все выходы:

$$\tilde{x}_{ij} = \frac{x_{ij} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

- Зададим нужные нам среднее и дисперсию:

$$z_{ij} = \gamma \circ \tilde{x}_{ij} + \beta$$

↑                      ↑  
обучаемые параметры (скаляры)

# Инициализации

# Инициализация весов

- Не должно быть симметрий (плохо инициализировать всё одним числом)
- Хороший вариант:

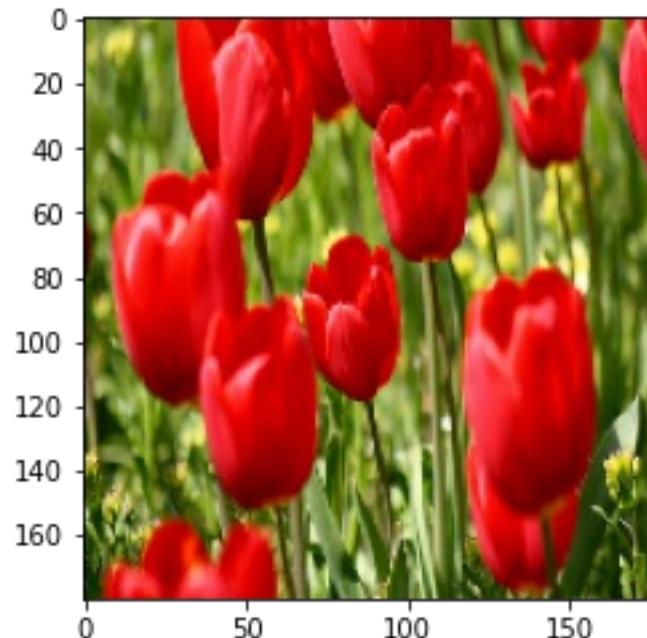
$$w_j \sim \frac{2}{\sqrt{n}} \mathcal{N}(0, 1)$$

$n$  — число входов

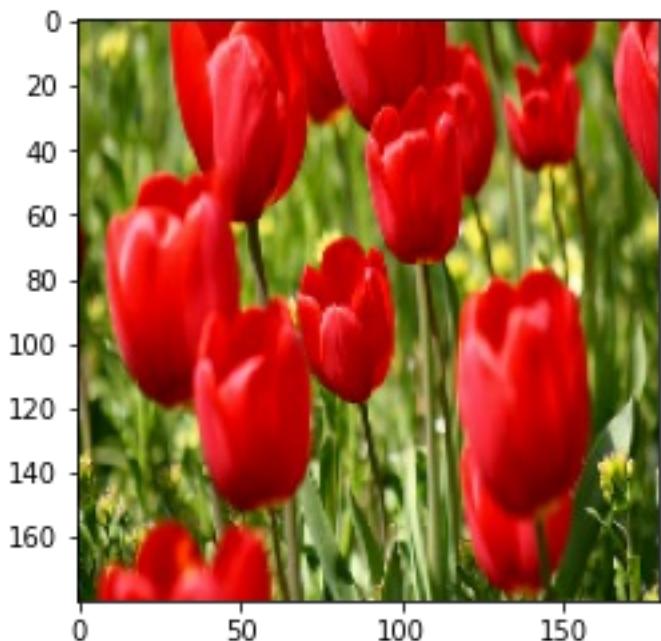
- Пытаемся сделать так, чтобы масштаб всех выходов был примерно одинаковым

# Аугментация

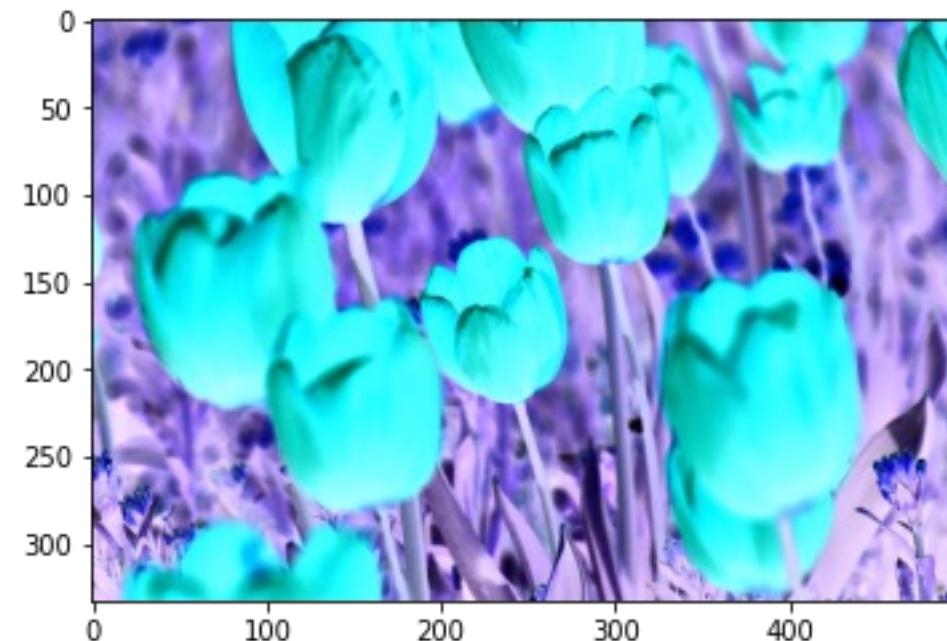
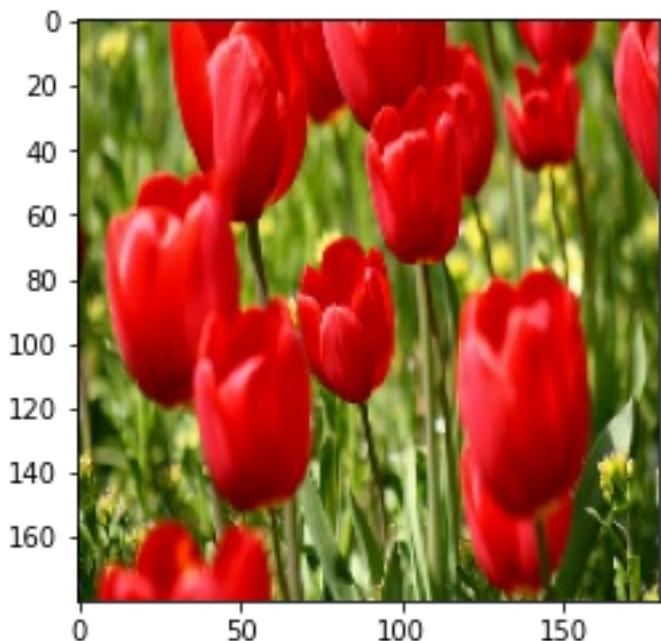
# Аугментация



# Аугментация



# Аугментация



[https://www.tensorflow.org/tutorials/images/data\\_augmentation](https://www.tensorflow.org/tutorials/images/data_augmentation)

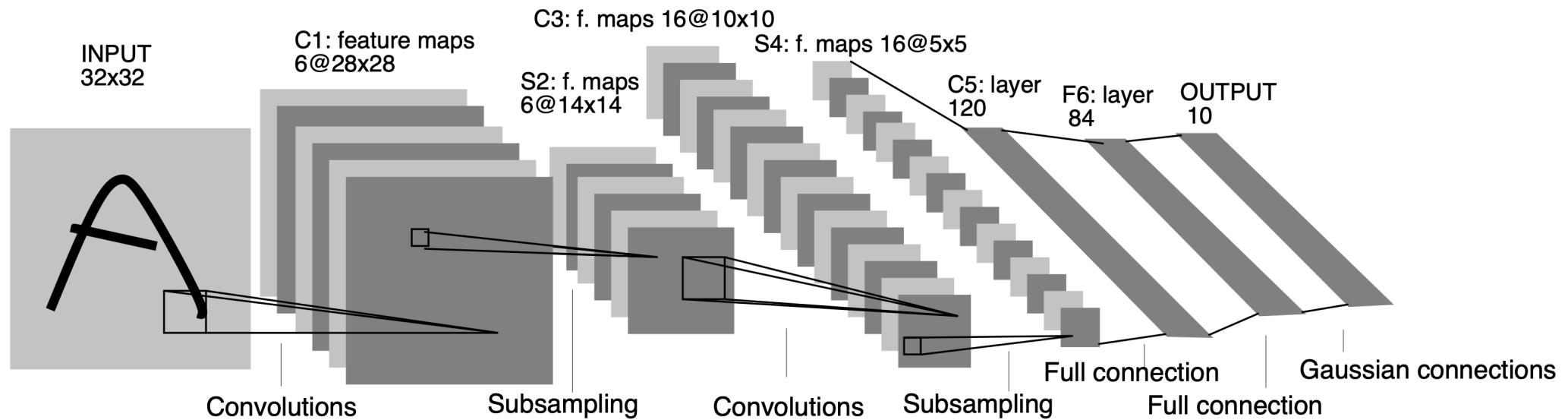


# Аугментация

- Много разных вариантов
- «Бесплатное» расширение обучающей выборки
- В некотором смысле регуляризация модели
  
- Обычно аугментации случайно применяют к картинкам из текущего батча
- На этапе применения можно сделать несколько аугментаций картинки, применить сеть к каждой, усреднить предсказания

# Архитектуры свёрточных сетей

# LeNet (1998)



# LeNet (1998)

- Для данных MNIST
- Идея end-to-end обучения
- Использовали аугментацию
- Около 60.000 параметров
- Доля ошибок на тесте 0.8%

# ImageNet



- ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
- Около 1.000.000 изображений
- 1000 классов
- Обычно качество измерялось на основе лучшей гипотезы модели

# AlexNet (2012)

---

## **ImageNet Classification with Deep Convolutional Neural Networks**

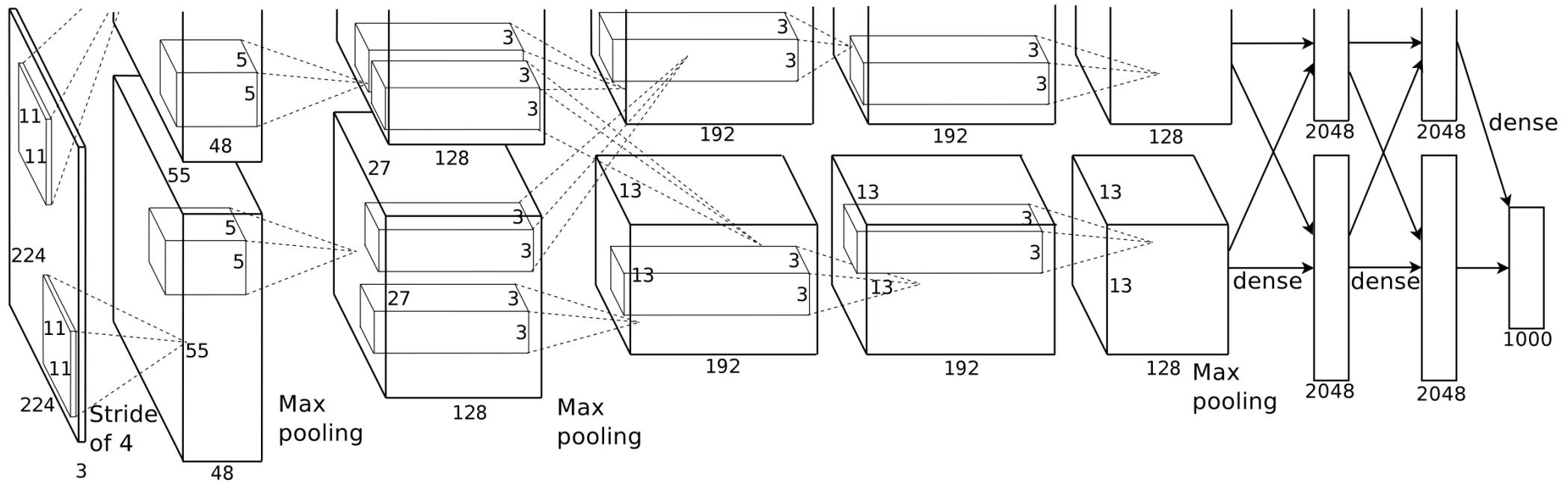
---

**Alex Krizhevsky**  
University of Toronto  
kriz@cs.utoronto.ca

**Ilya Sutskever**  
University of Toronto  
ilya@cs.utoronto.ca

**Geoffrey E. Hinton**  
University of Toronto  
hinton@cs.utoronto.ca

# AlexNet (2012)



# AlexNet (2012)

- Используют ReLU, аугментацию, dropout
- Градиентный спуск с инерцией (momentum)
- Обучение на двух GPU (5-6 суток)
- Около 60 миллионов параметров
- Ошибка около 17%

# VGG (2014)

## VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

**Karen Simonyan\*** & **Andrew Zisserman<sup>+</sup>**

Visual Geometry Group, Department of Engineering Science, University of Oxford  
`{karen,az}@robots.ox.ac.uk`

# 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 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	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 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

# VGG (2014)

**Table 2: Number of parameters (in millions).**

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

# VGG (2014)

- Только маленькие свёртки
  - Меньше параметров
  - Больше нелинейностей (т.к. больше свёрточных слоёв)
- Градиентный спуск с инерцией
- Dropout для двух первых полносвязных слоёв
- Хитрая инициализация (сначала обучается вариант A со случайными начальными весами, потом им инициализируются более глубокие сети)

# VGG (2014)

Table 3: ConvNet performance at a single test scale.

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train ( $S$ )	test ( $Q$ )		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	<b>25.5</b>	<b>8.0</b>

# GoogLeNet (2014)

## Going Deeper with Convolutions

Christian Szegedy<sup>1</sup>, Wei Liu<sup>2</sup>, Yangqing Jia<sup>1</sup>, Pierre Sermanet<sup>1</sup>, Scott Reed<sup>3</sup>,  
Dragomir Anguelov<sup>1</sup>, Dumitru Erhan<sup>1</sup>, Vincent Vanhoucke<sup>1</sup>, Andrew Rabinovich<sup>4</sup>

<sup>1</sup>Google Inc. <sup>2</sup>University of North Carolina, Chapel Hill

<sup>3</sup>University of Michigan, Ann Arbor <sup>4</sup>Magic Leap Inc.

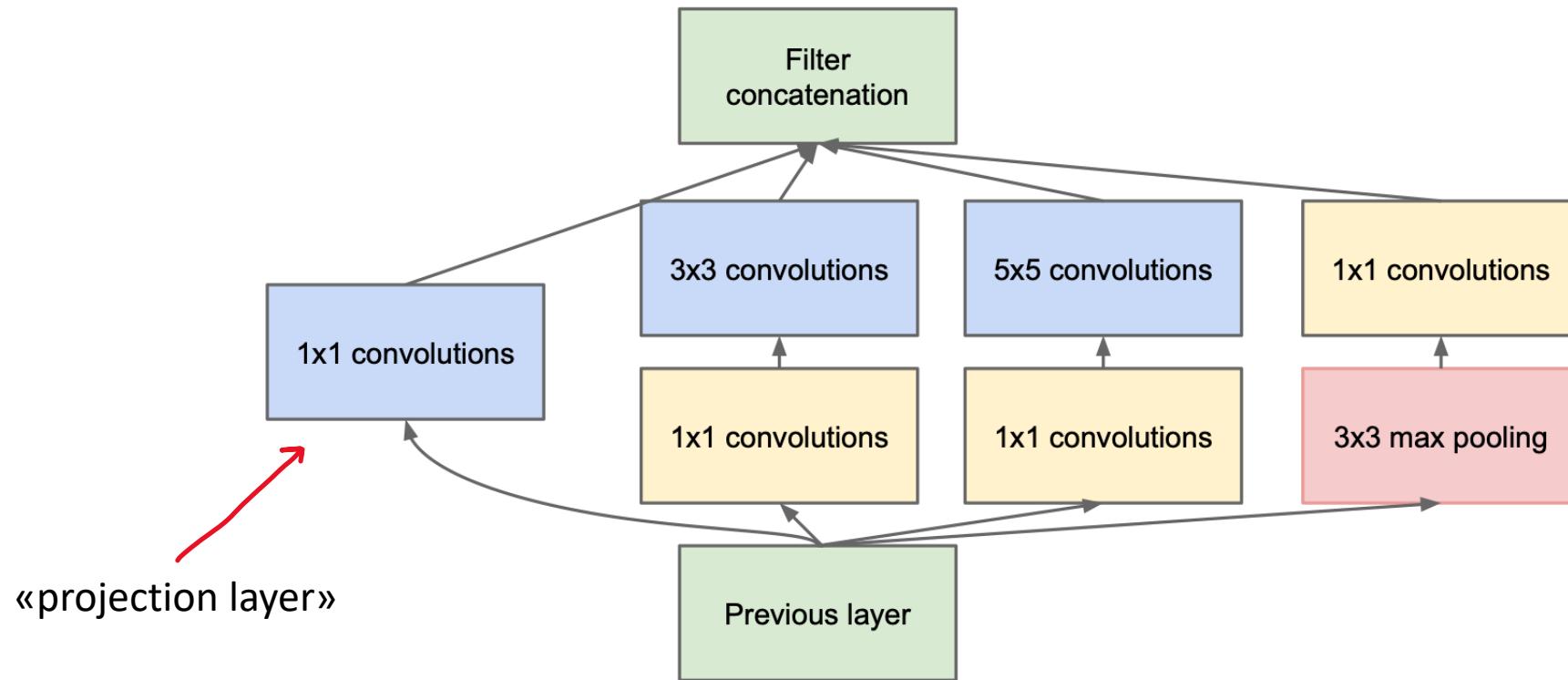
<sup>1</sup>{szegedy,jiayq,sermanet,dragomir,dumitru,vanhoucke}@google.com

<sup>2</sup>wliu@cs.unc.edu, <sup>3</sup>reedscott@umich.edu, <sup>4</sup>arabinovich@magineleap.com



Figure 3: GoogLeNet network with all the bells and whistles.

# GoogLeNet (2014)



(b) Inception module with dimensionality reduction

свёртки делаются с паддингом!

<http://arxiv.org/abs/1409.4842>

# GoogLeNet (2014)

- Снижается число каналов перед «тяжёлыми» свёртками
- Несколько выходных слоёв для улучшения обучаемости
- Обучается градиентным спуском с инерцией
- Ошибка 6.67% на ImageNet

# ResNet (2015)

## **Deep Residual Learning for Image Recognition**

Kaiming He

Xiangyu Zhang

Shaoqing Ren

Jian Sun

Microsoft Research

{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

# ResNet (2015)

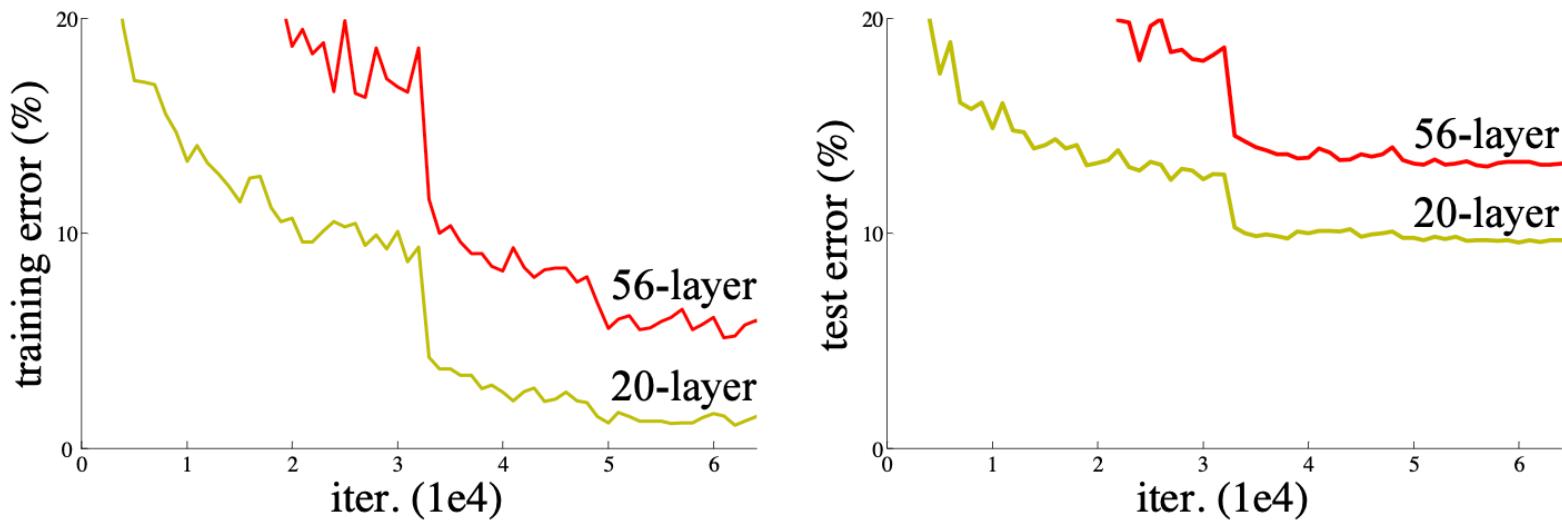
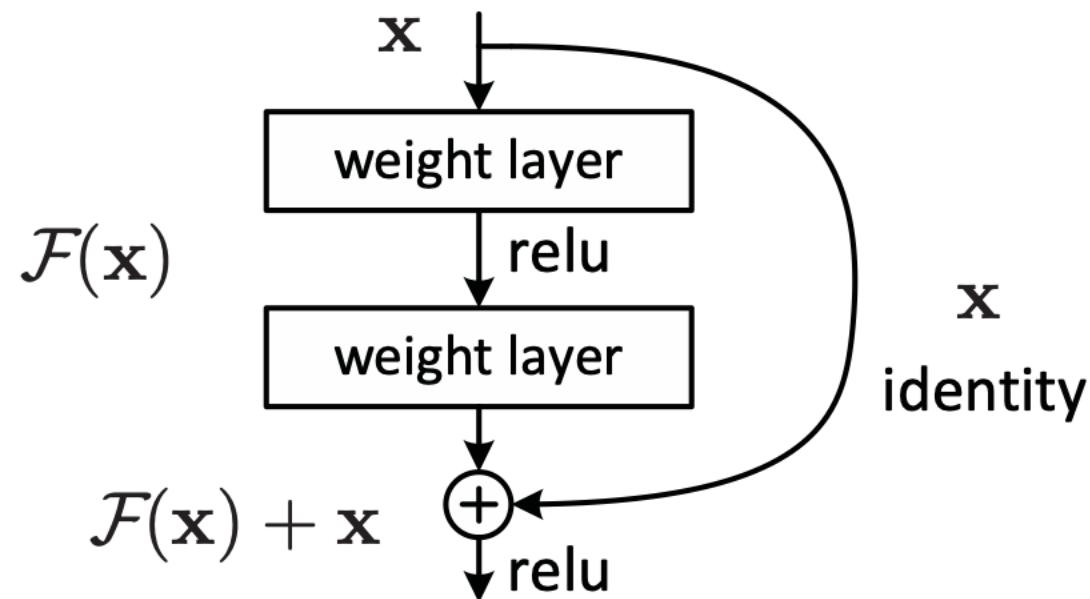


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

# ResNet (2015)

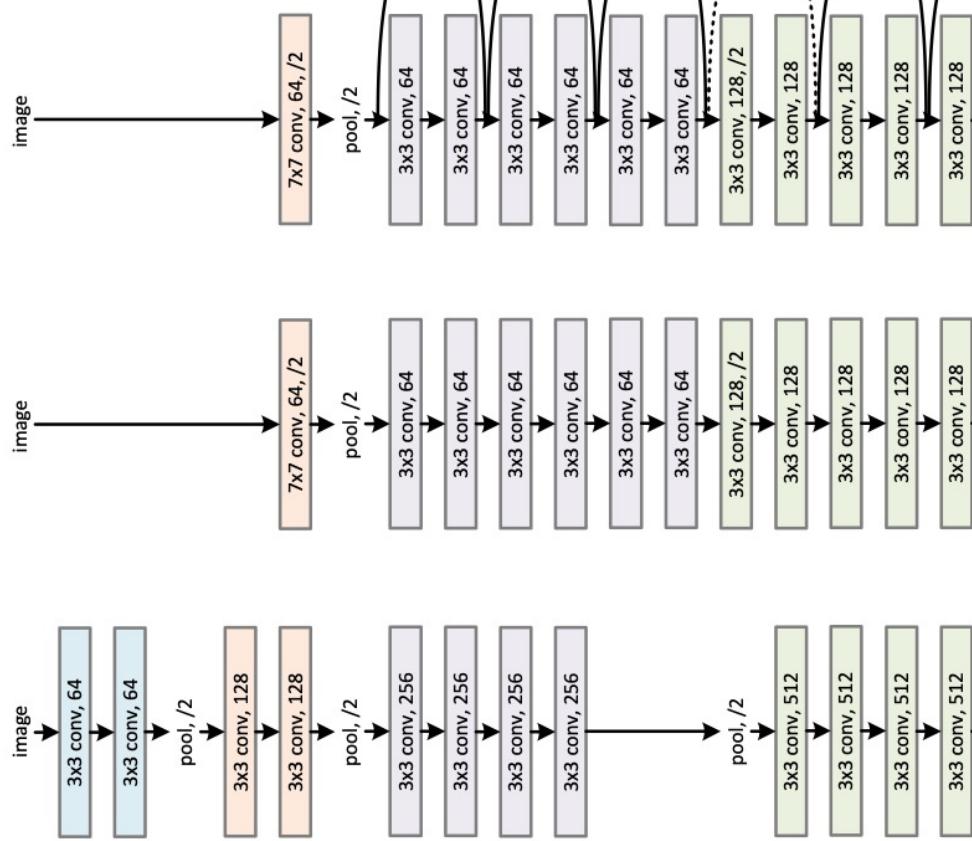
- Добавление слоёв в свёрточную сеть ухудшает качество даже на обучении
- Хотя возможностей для переобучения больше, сеть почему-то не может ими воспользоваться

# ResNet (2015)

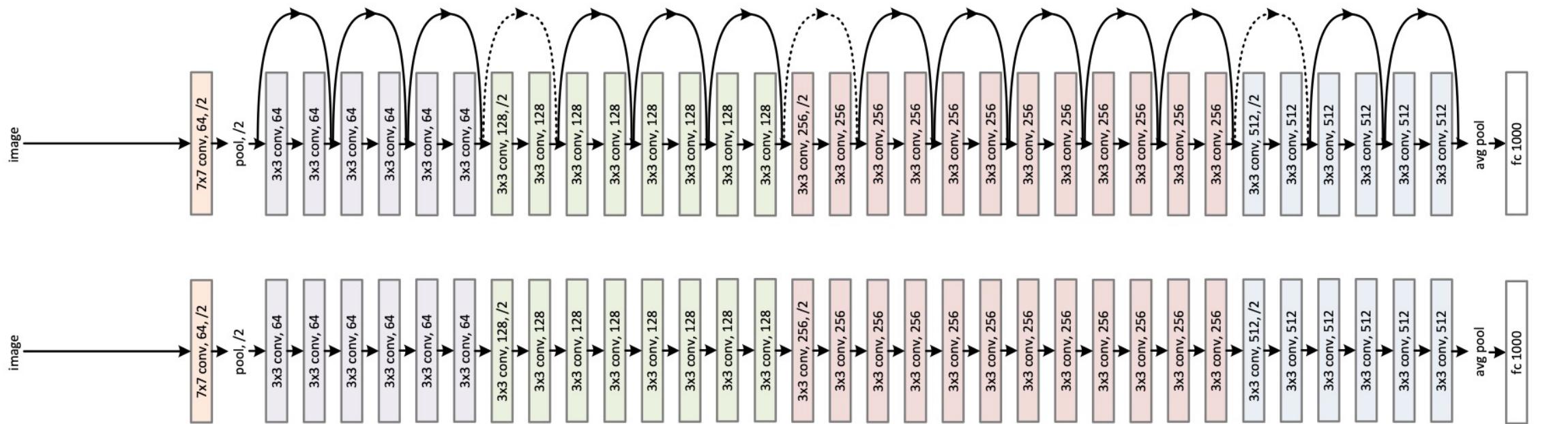


# ResNet (2015)

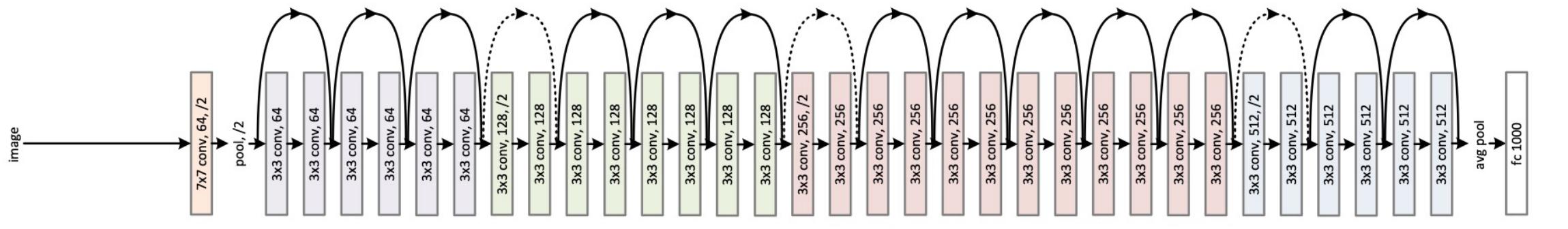
VGG-19



34-layer plain



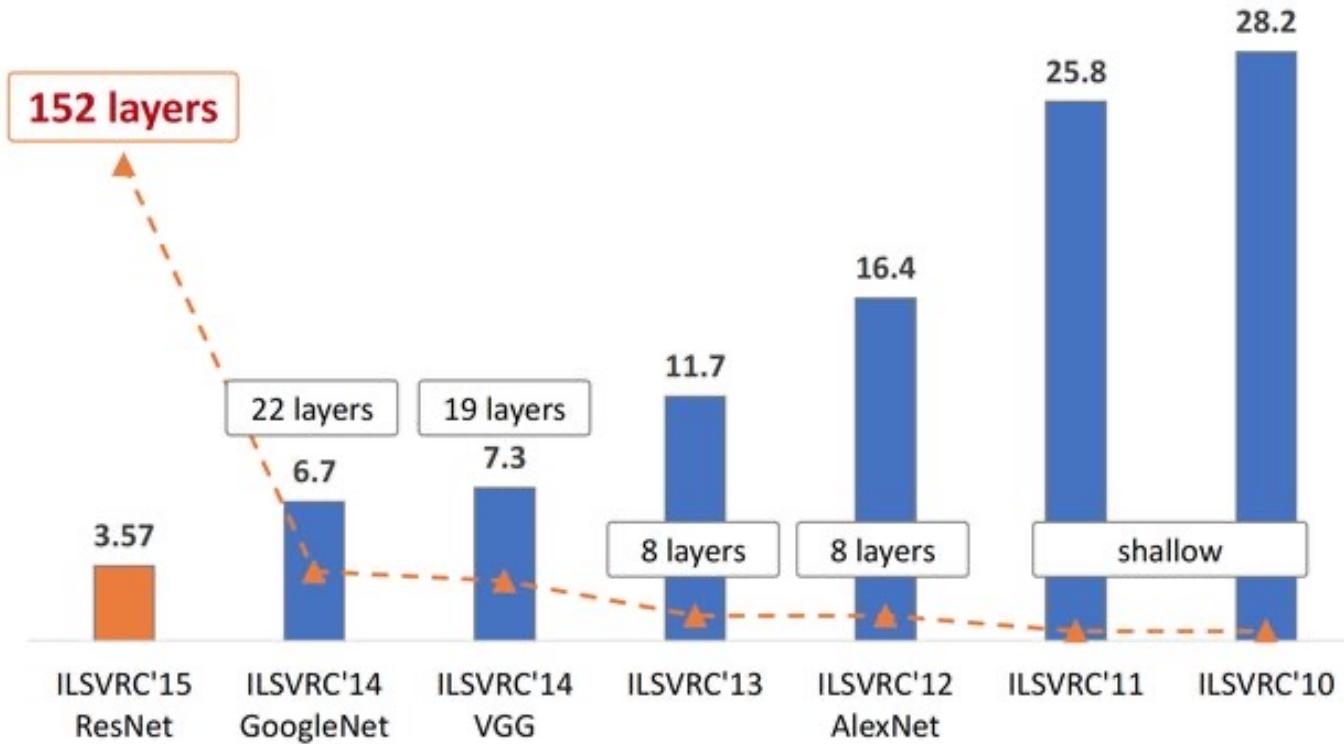
34-layer residual



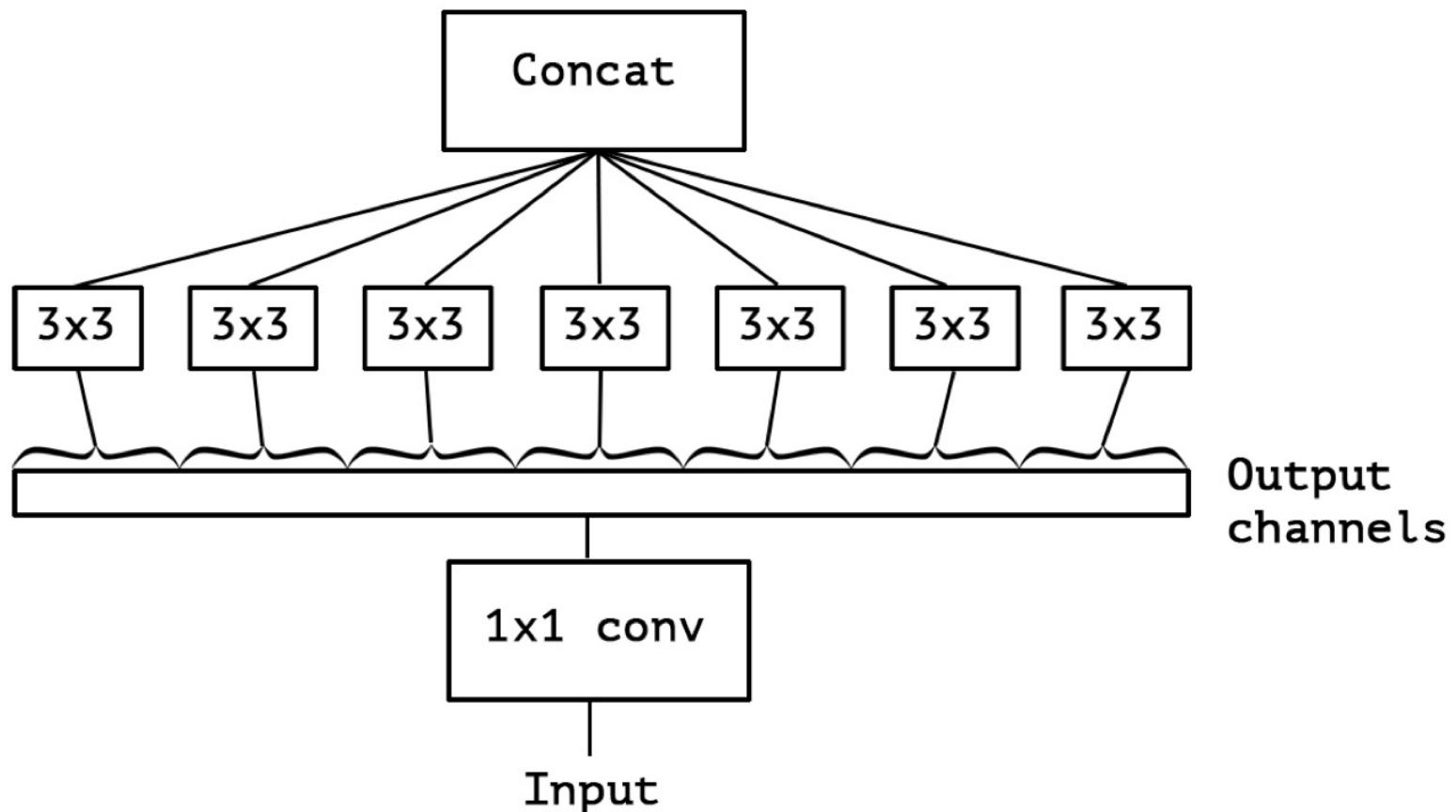
# ResNet (2015)

- Даёт низкую ошибку на обучении даже с 1000 слоёв (но там плохо на тестовой выборке)
- Обучается градиентным спуском с инерцией со случайной инициализацией
- Ошибка 4.49% на ImageNet

# Эволюция архитектур



# Xception



# Xception

- Разделяется роль свёрток: либо по каналам, либо по пространству
- Более эффективное использование параметров

# Что ещё?

- Highway networks
- Inception-ResNet
- Squeeze and Excitation Network
- MobileNet
- EfficientNet
- ...