

# Машинное обучение

## на примере глубокого обучения в компьютерного зрения

### Занятие 5

## Тренировка нейронных сетей часть 2

### Архитектуры CNN

Дмитрий Яшунин, к.ф.-м.н  
IntelliVision

e-mail: [yashuninda@yandex.ru](mailto:yashuninda@yandex.ru)

# Архитектуры нейронных сетей

# Классификация изображений

Ключевая задача компьютерного зрения



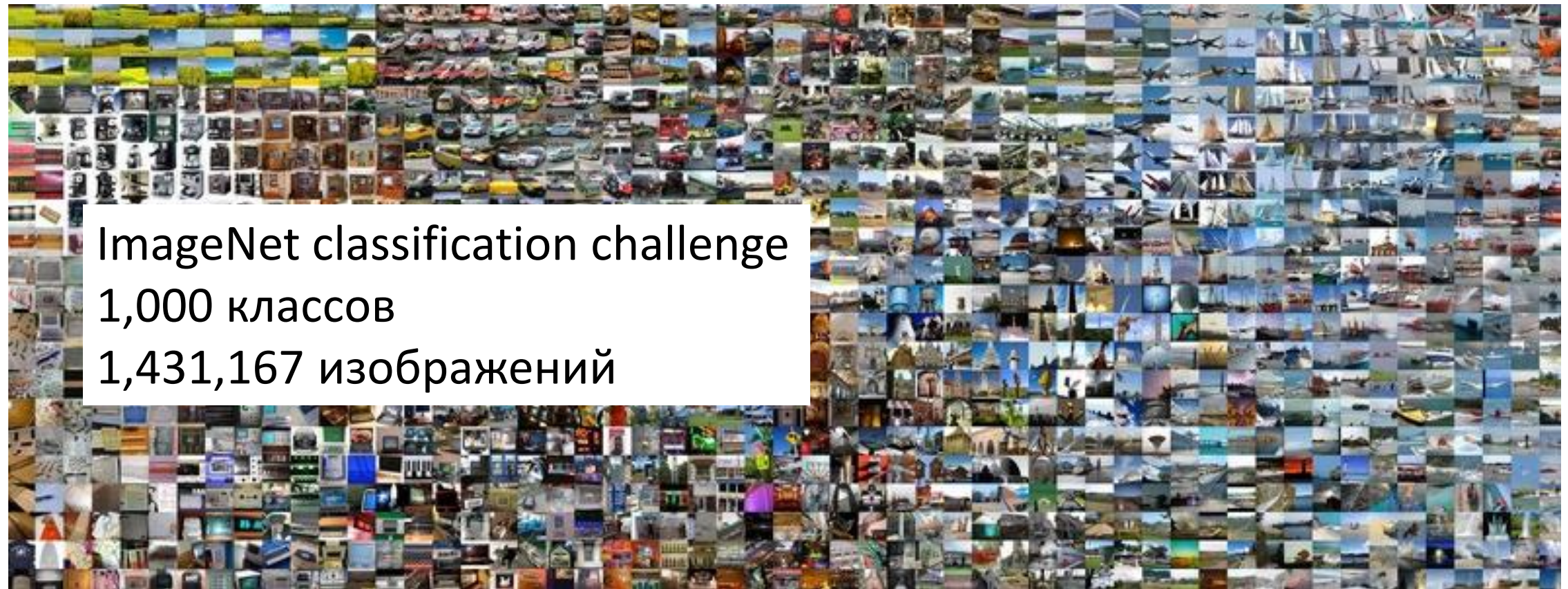
К какому классу принадлежит изображение?  
классы: человек, животное, автомобиль ...



КОТ



# Large Scale Visual Recognition Challenge (ILSVRC)

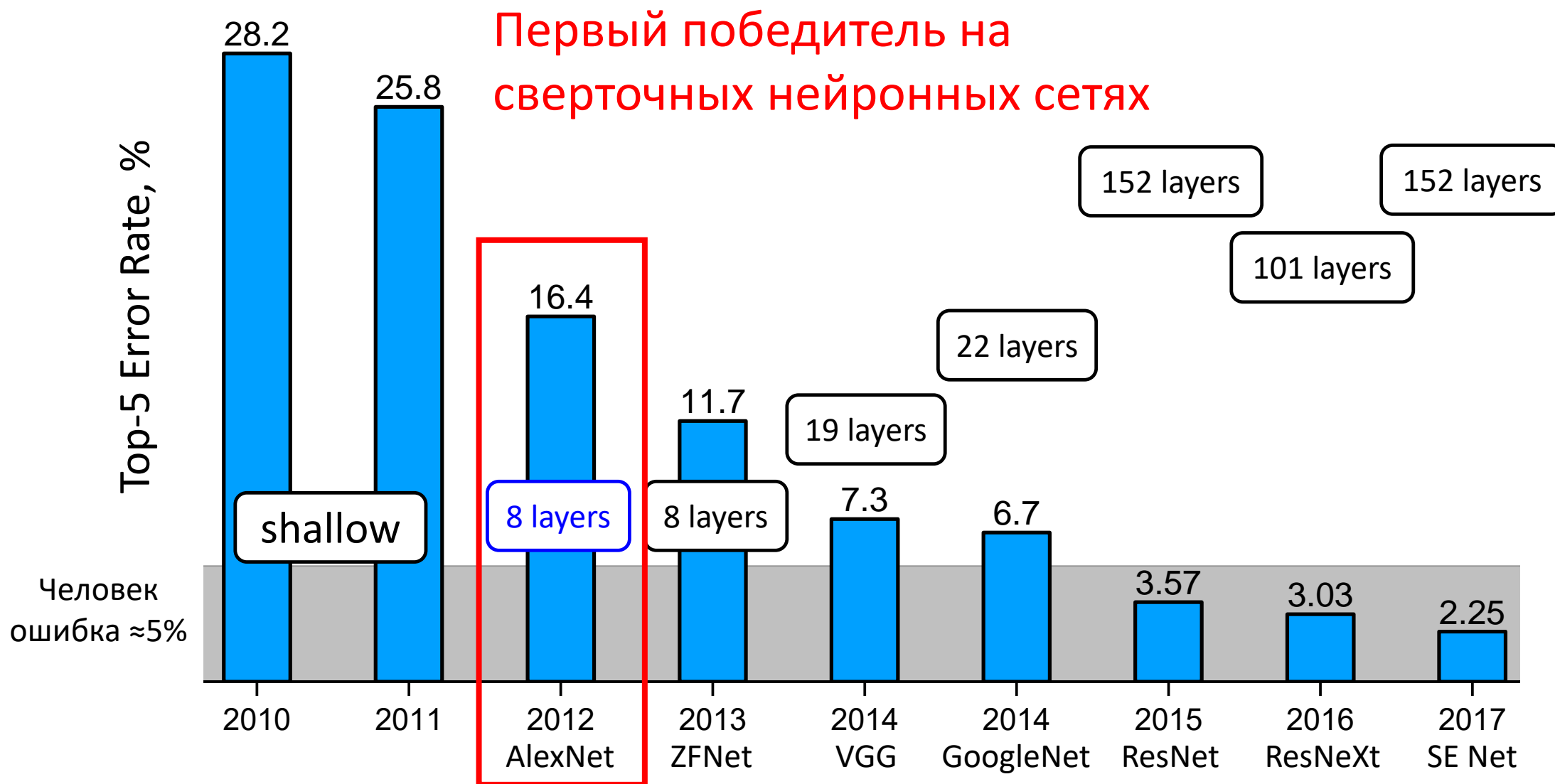


ImageNet classification challenge  
1,000 классов  
1,431,167 изображений

<http://image-net.org/>



# Large Scale Visual Recognition Challenge (ILSVRC)



# AlexNet

(Krizhevsky et al. 2012)

Архитектура:

CONV1

MAX POOL1

LRN1

CONV2

MAX POOL2

LRN2

CONV3

CONV4

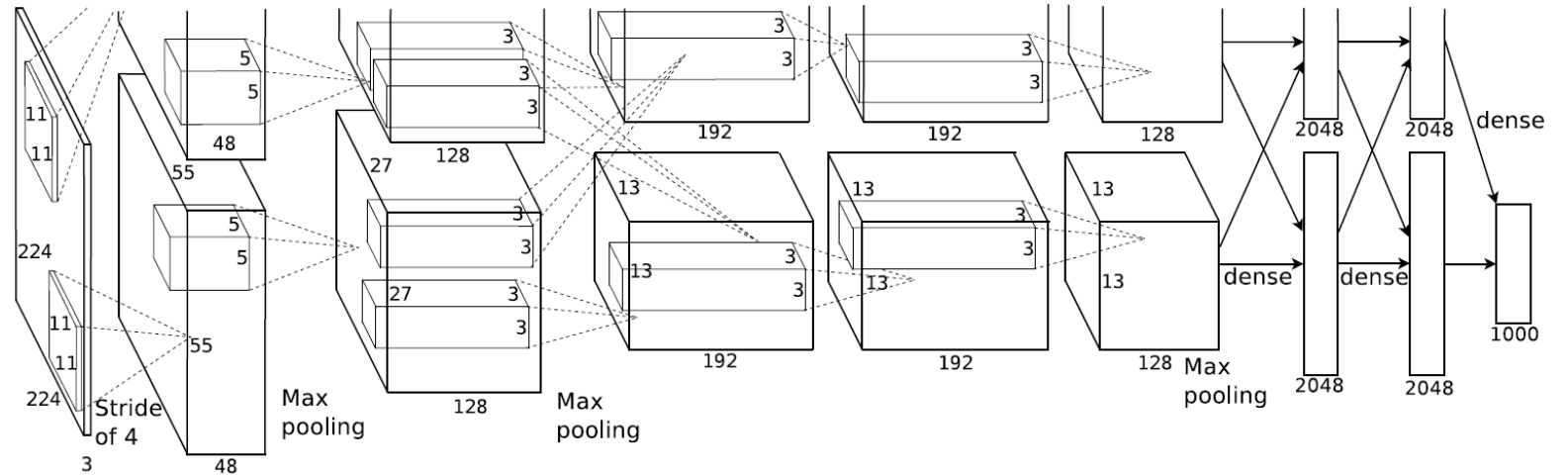
CONV5

Max POOL3

FC6

FC7

FC8

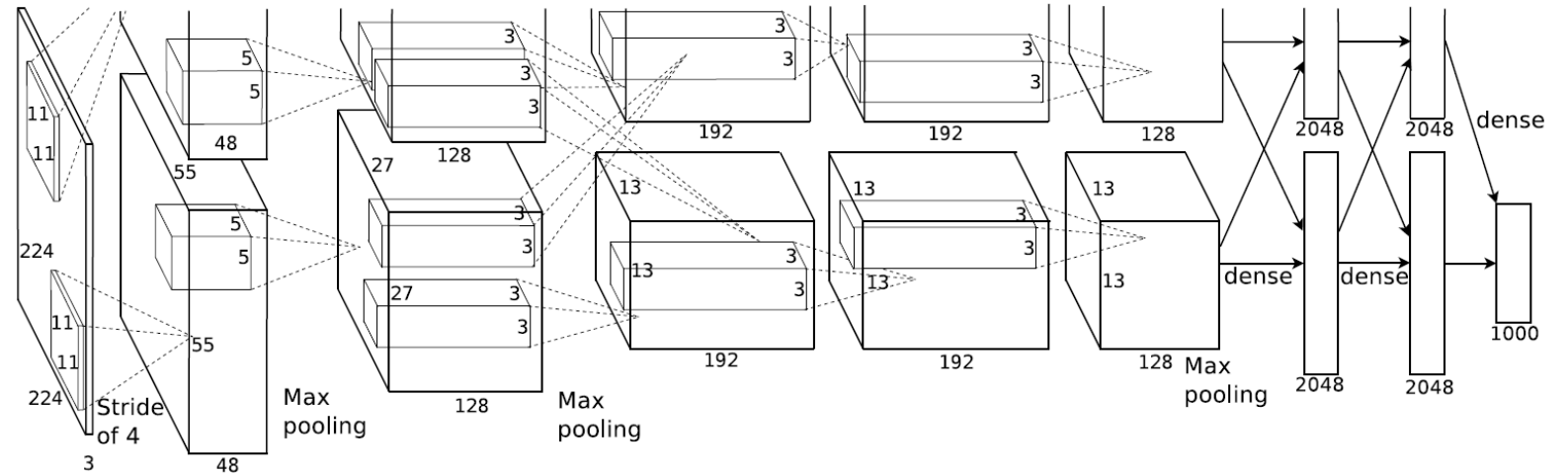


ImageNet top 5 error: 25.8% -> 16.4%



# AlexNet

(Krizhevsky et al. 2012)



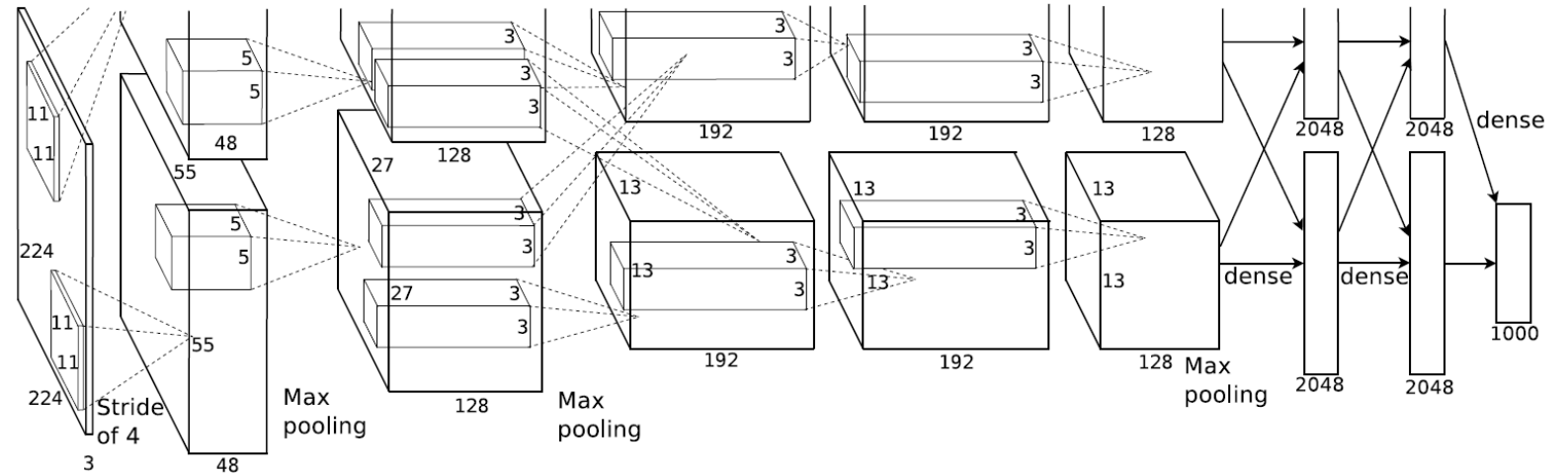
Размер входного изображения: 227x227x3

**Первый слой (CONV1):** 96 11x11 фильтров (filters) с шагом (stride) 4 отступом (padding) 0

Вопрос: какой размер карты активаций на выходе? Подсказка:  $(227-11)/4+1 = 55$

# AlexNet

(Krizhevsky et al. 2012)



Размер входного изображения: 227x227x3

**Первый слой (CONV1):** 96 11x11 фильтров (filters) с шагом (stride) 4

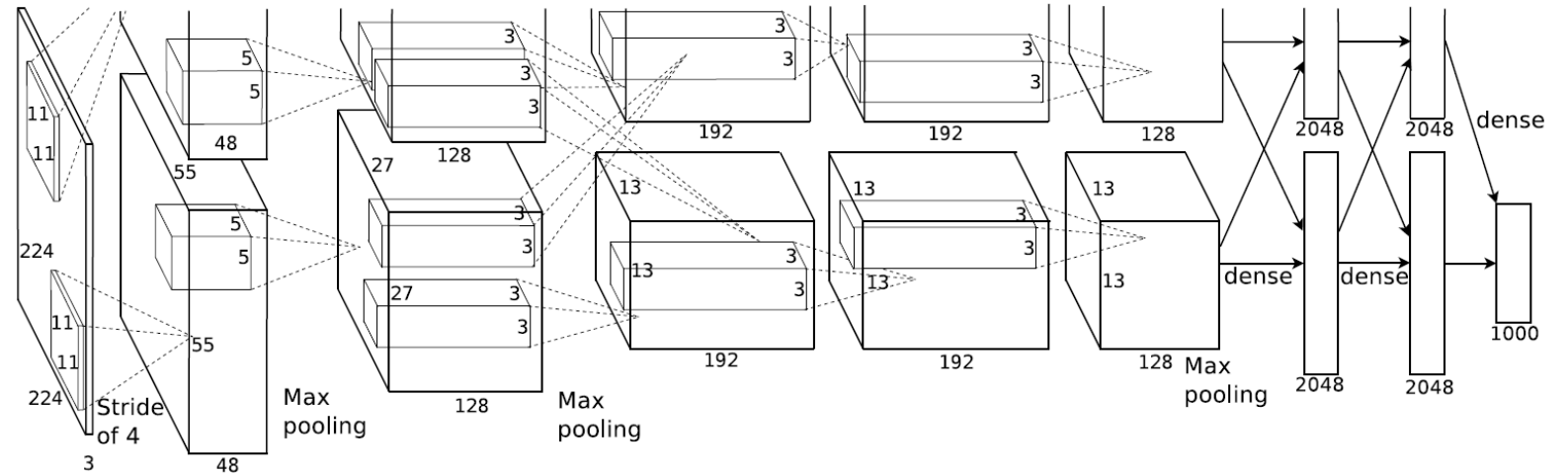
Размер карты активаций **55x55x96**

Вопрос: какое число параметров в слое?



# AlexNet

*(Krizhevsky et al. 2012)*



Размер входного изображения: 227x227x3

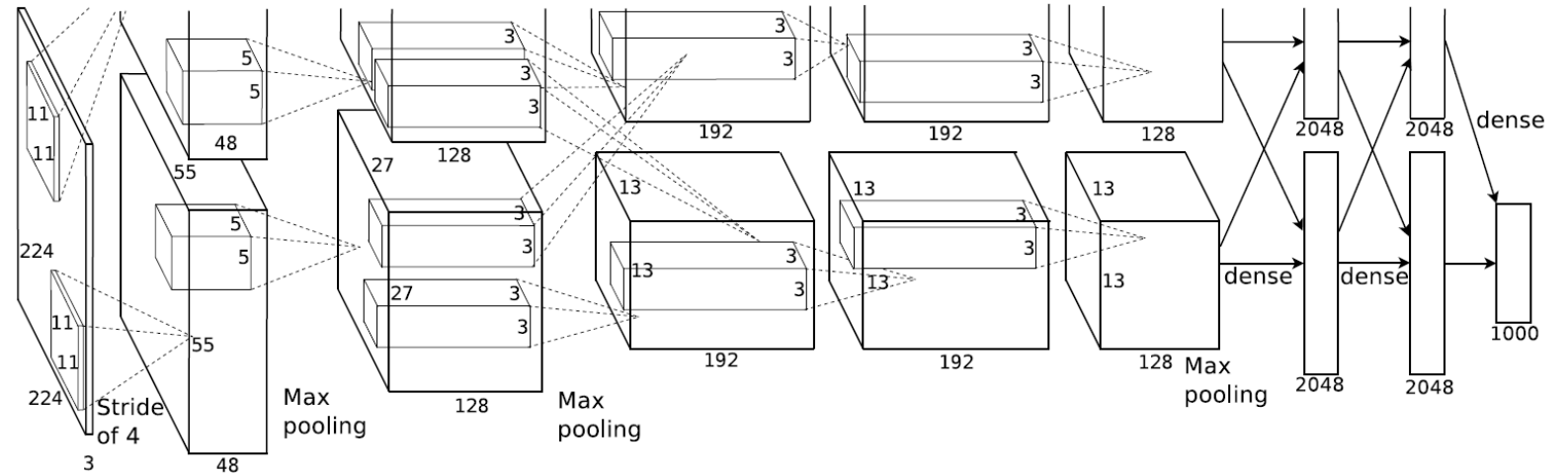
**Первый слой (CONV1):** 96 11x11 фильтров (filters) с шагом (stride) 4

Размер карты активаций **55x55x96**

Число параметров:  $(11*11*3)*96 = 35K$

# AlexNet

*(Krizhevsky et al. 2012)*



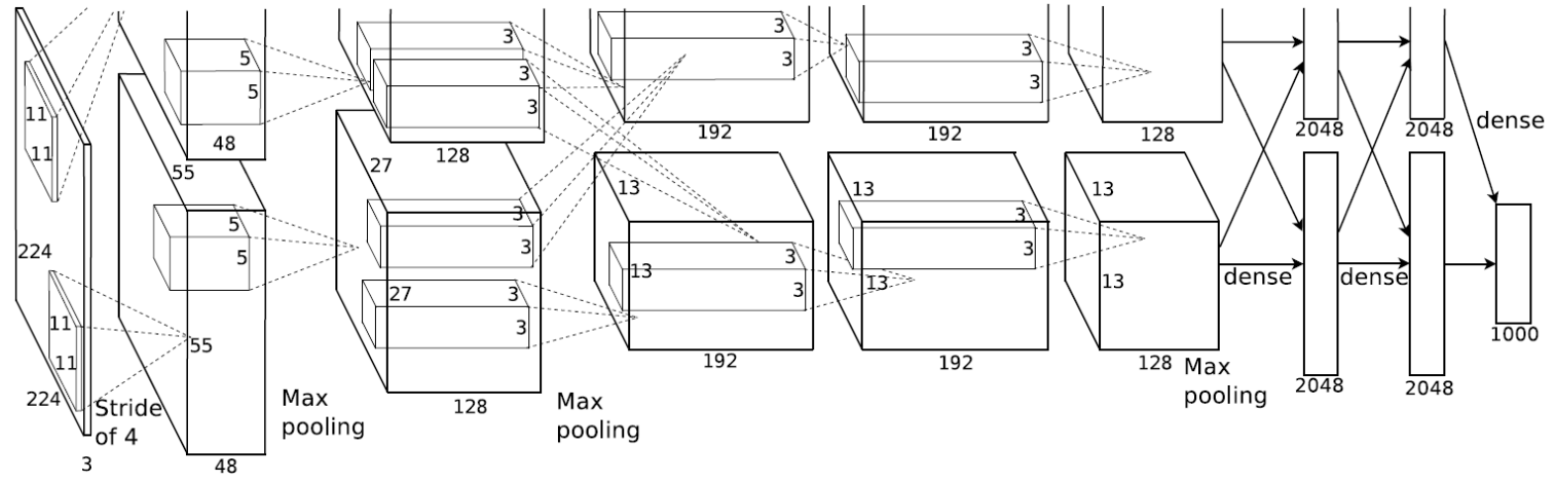
Размер входного изображения: 227x227x3  
после CONV1: 55x55x96

**Второй слой (MAX POOL1):** фильтры 3x3 с шагом (stride) 2

Вопрос: какой размер на выходе? Подсказка:  $(55-3)/2+1 = 27$

# AlexNet

(Krizhevsky et al. 2012)



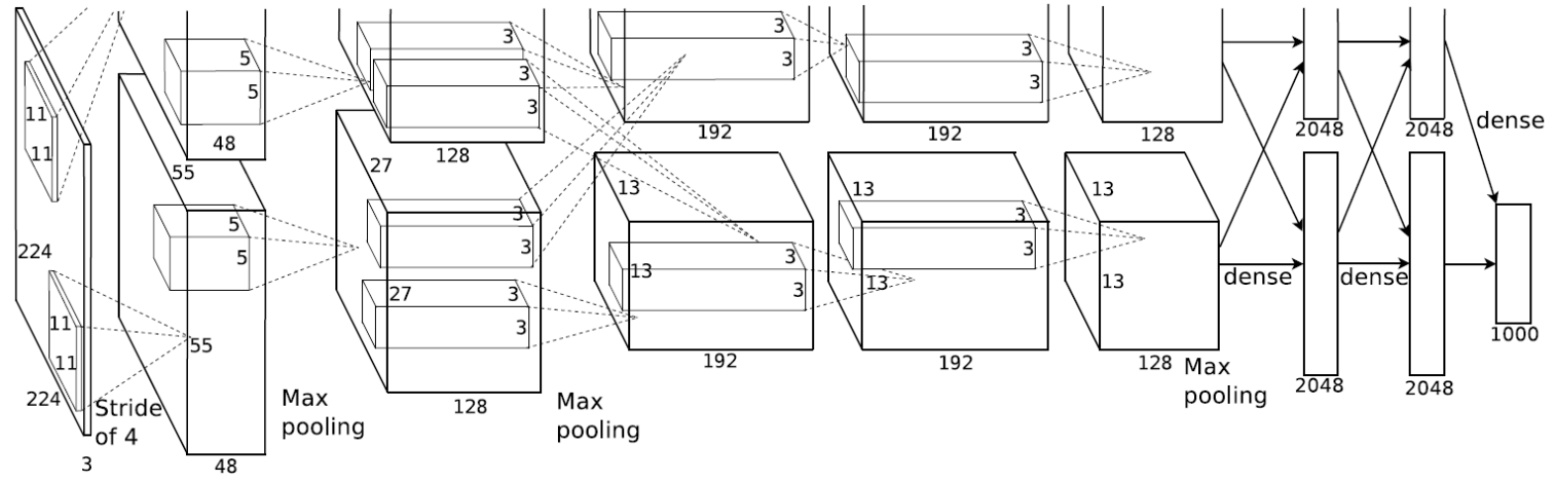
Размер входного изображения: 227x227x3  
после CONV1: 55x55x96

**Второй слой (MAX POOL1):** фильтры 3x3 с шагом (stride) 2  
Размер на выходе: **27x27x96**

Вопрос: какое число параметров в слое?

# AlexNet

*(Krizhevsky et al. 2012)*



Размер входного изображения: 227x227x3  
после CONV1: 55x55x96

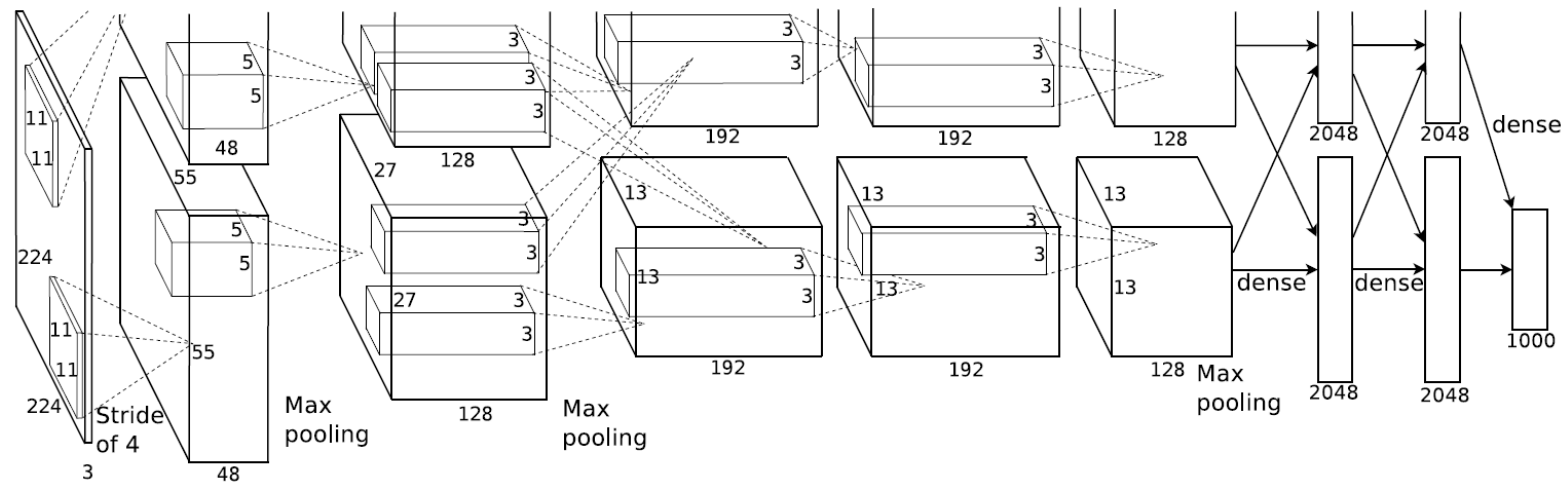
**Второй слой (MAX POOL1):** фильтры 3x3 с шагом (stride) 2

Размер на выходе: **27x27x96**

Число параметров: **0**

# AlexNet

*(Krizhevsky et al. 2012)*



Размер входного изображения: 227x227x3

после CONV1: 55x55x96

после POOL1: 27x27x96

...

# AlexNet

(Krizhevsky et al. 2012)

## Архитектура AlexNet:

[227x227x3] INPUT

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

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

[27x27x96] **LRN1**: Normalization layer

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

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

[13x13x256] **LRN2**: Normalization layer

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

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

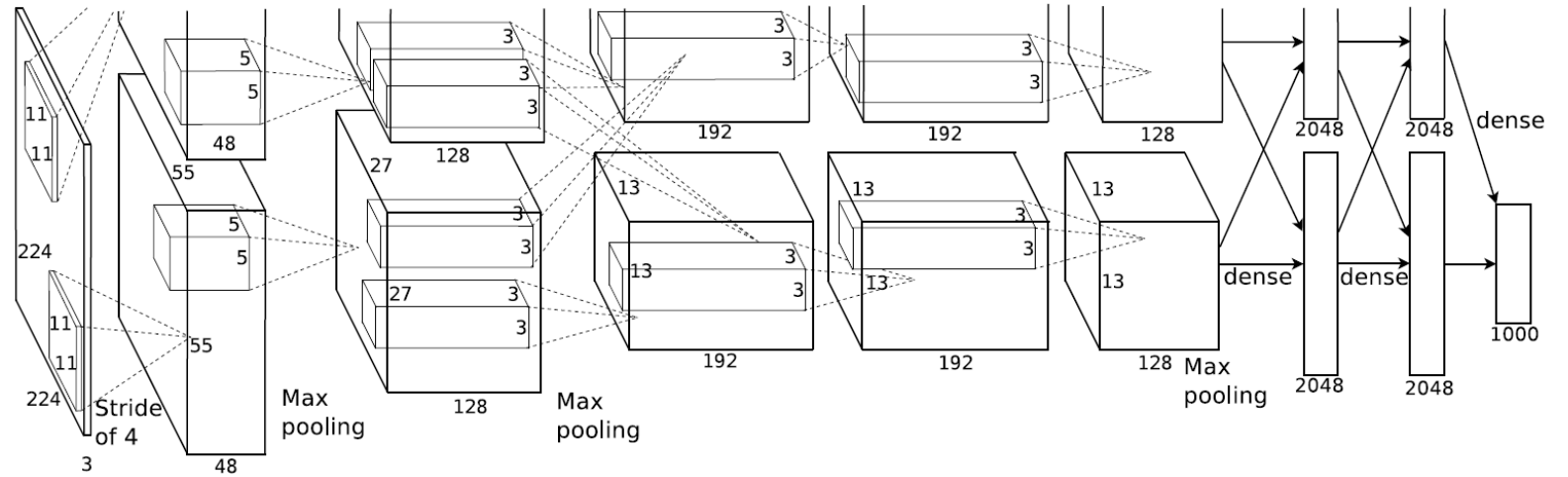
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

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

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)





# AlexNet

(Krizhevsky et al. 2012)

## Архитектура AlexNet:

[227x227x3] INPUT

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

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

[27x27x96] **LRN1**: Normalization layer

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

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

[13x13x256] **LRN2**: Normalization layer

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

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

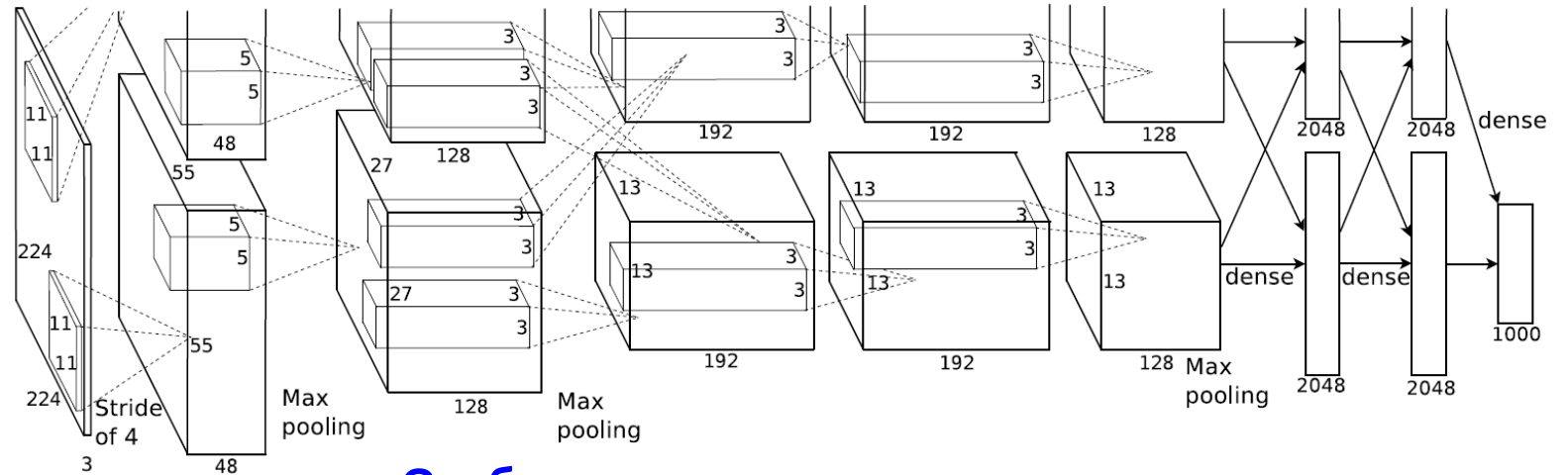
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

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

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)



## Особенности:

- впервые применили ReLU
- использовали Local Response Normalization слои (теперь не применяются)
- сильная аугментация данных
- dropout 0.5
- batch size 128, SGD Momentum 0.9
- learning rate 1e-2, когда точность на валидационном наборе переставала расти уменьшали в 10 раз
- L2 weight decay (регуляризация) 5e-4
- 7 CNN ансамбль моделей (ensemble): 18.2% -> 15.4% (на валидации)
- ≈ 60 млн. параметров, ≈ 230 Мбайт caffemodel

# AlexNet

(Krizhevsky et al. 2012)

## Архитектура AlexNet:

[227x227x3] INPUT

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

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

[27x27x96] **LRN1**: Normalization layer

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

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

[13x13x256] **LRN2**: Normalization layer

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

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

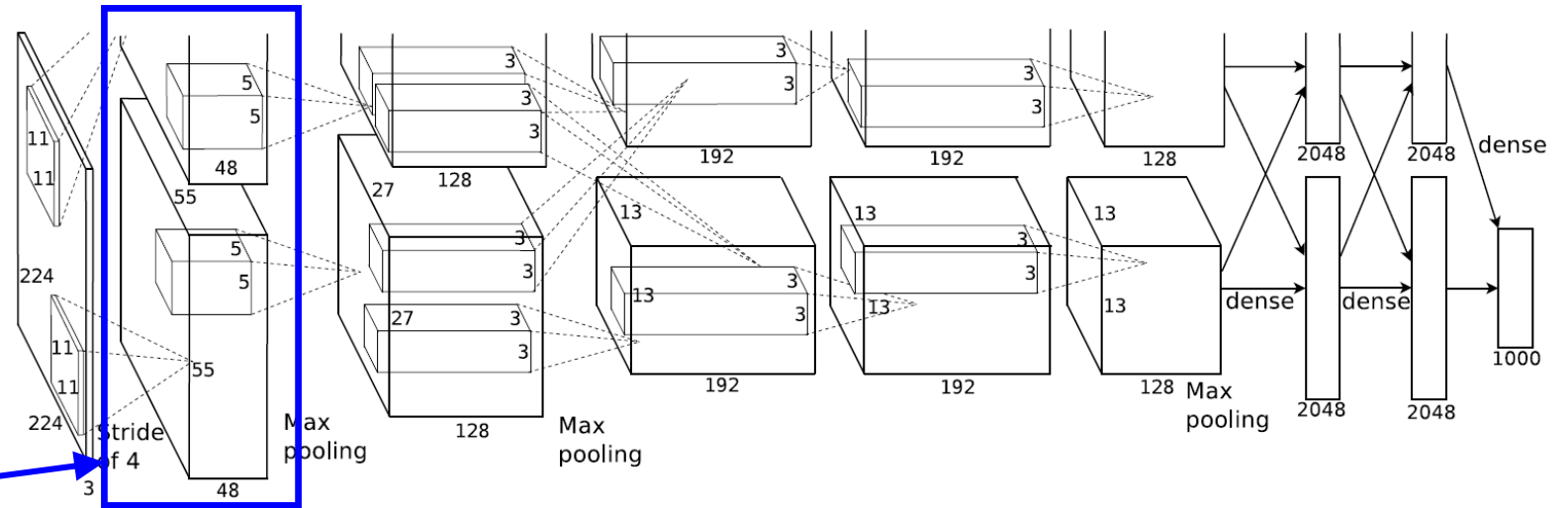
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

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

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)



[55x55x48]x2

Сеть тренировали на двух GTX 580 3GB GPUs.

Сеть делилась между двумя видеокартами. Половина карт активации на одной GPU, половина на другой.

# AlexNet

(Krizhevsky et al. 2012)

## Архитектура AlexNet:

[227x227x3] INPUT

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

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

[27x27x96] **LRN1**: Normalization layer

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

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

[13x13x256] **LRN2**: Normalization layer

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

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

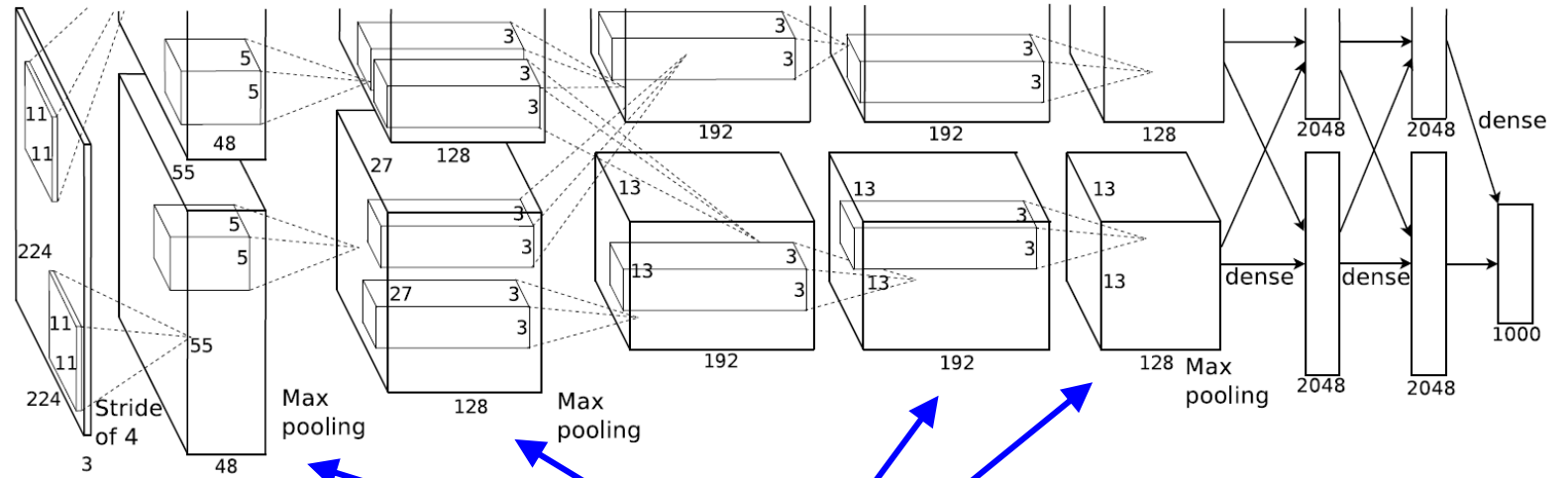
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

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

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)



CONV1, CONV2, CONV4, CONV5:  
Connections only with feature maps  
on same GPU

# AlexNet

(Krizhevsky et al. 2012)

## Архитектура AlexNet:

[227x227x3] INPUT

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

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

[27x27x96] **LRN1**: Normalization layer

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

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

[13x13x256] **LRN2**: Normalization layer

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

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

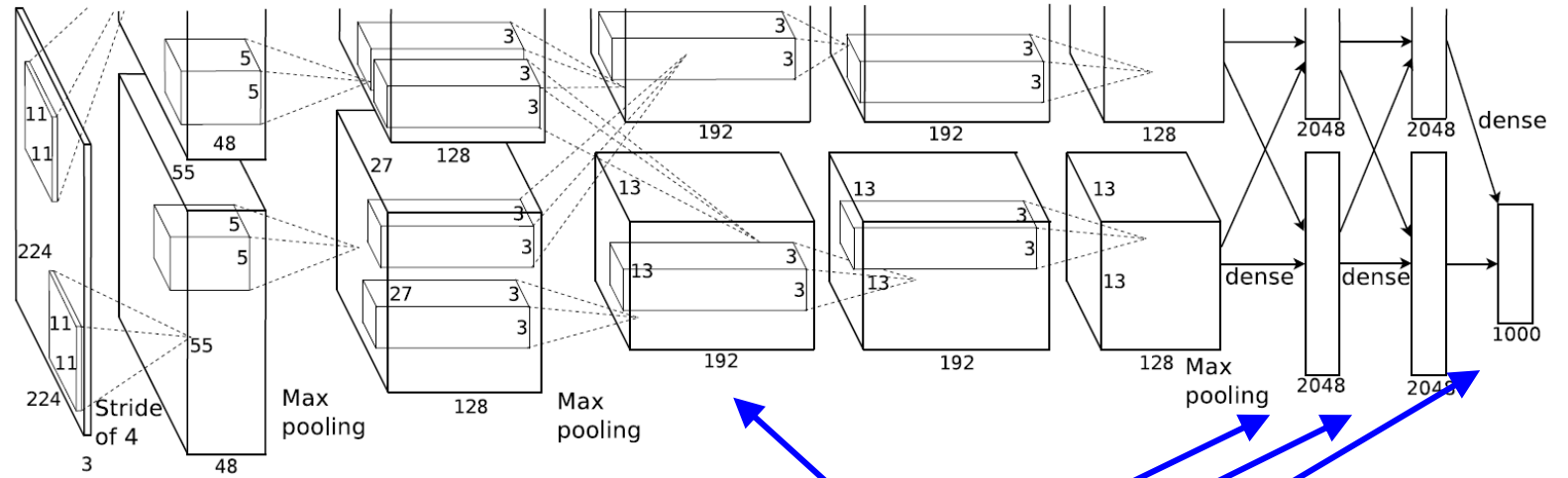
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

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

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

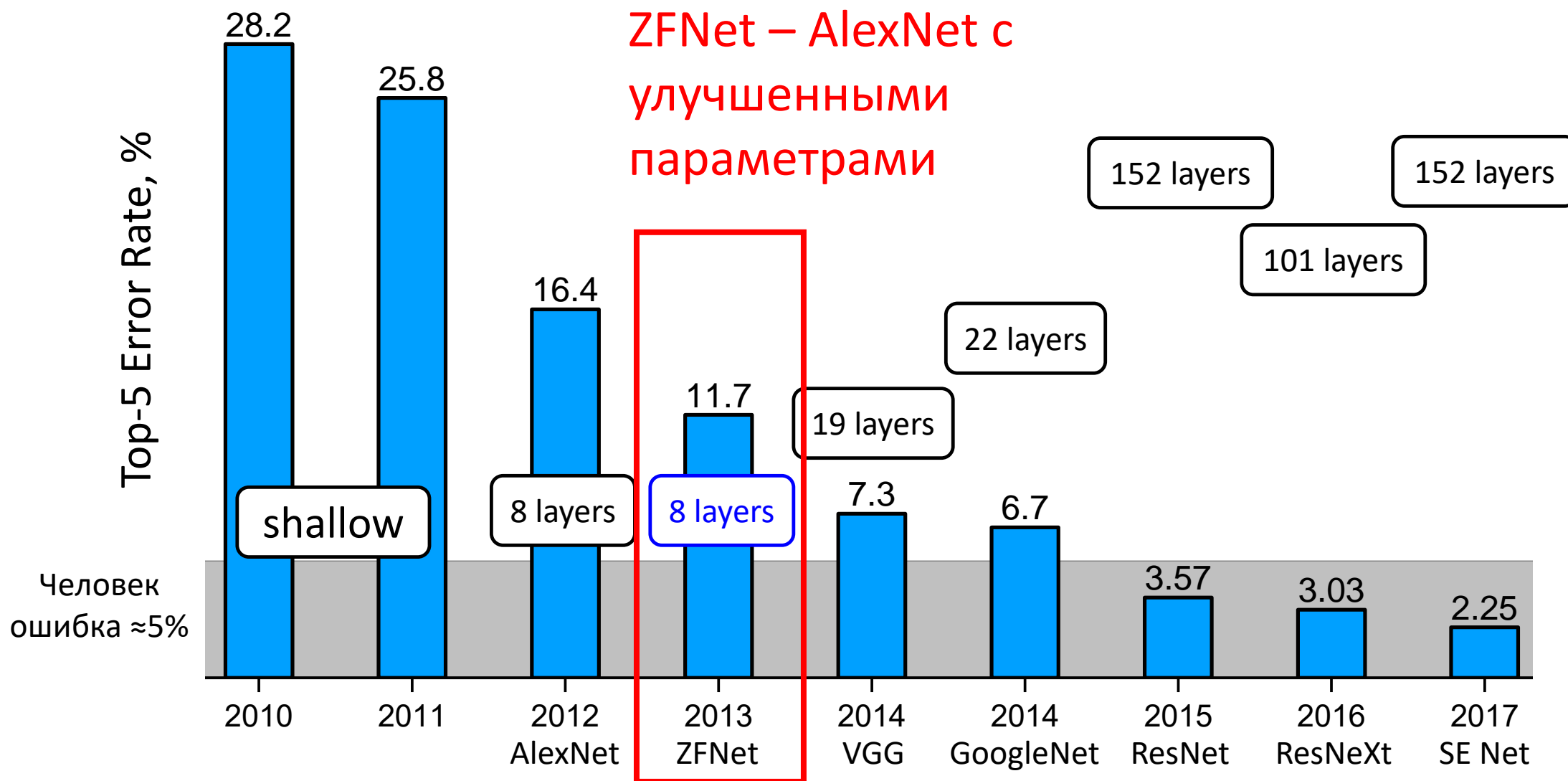
[1000] **FC8**: 1000 neurons (class scores)



**CONV3, FC6, FC7, FC8:**  
Connections with all feature maps in preceding layer, communication across GPUs

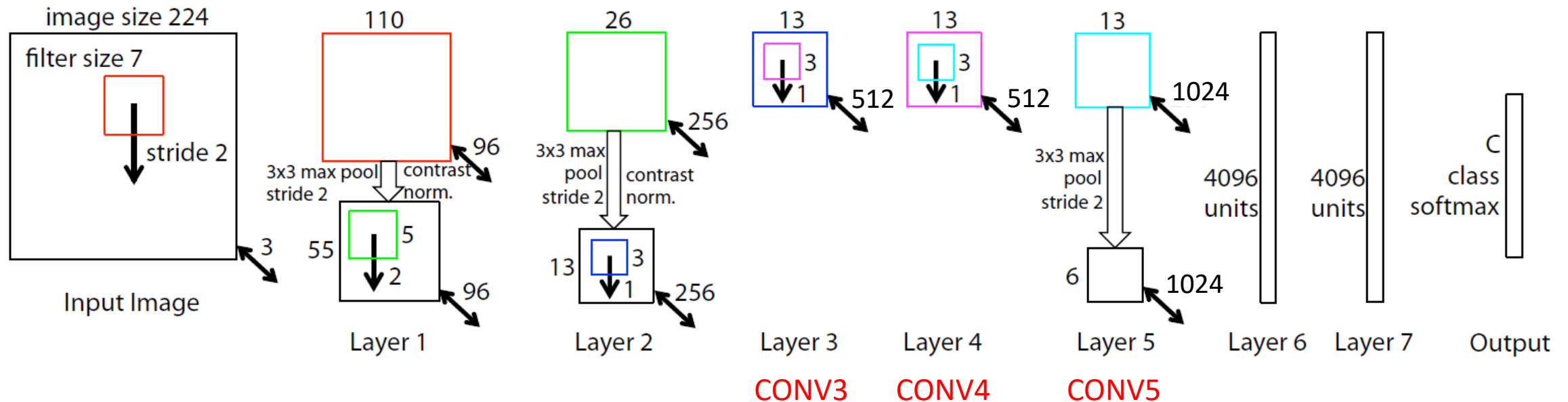


# Large Scale Visual Recognition Challenge (ILSVRC)



# ZFNet

(Zeiler and Fergus, 2013)



Архитектура ZFNet почти такая же как у AlexNet:

**CONV1:** (11x11 stride 4) -> (7x7 stride 2)

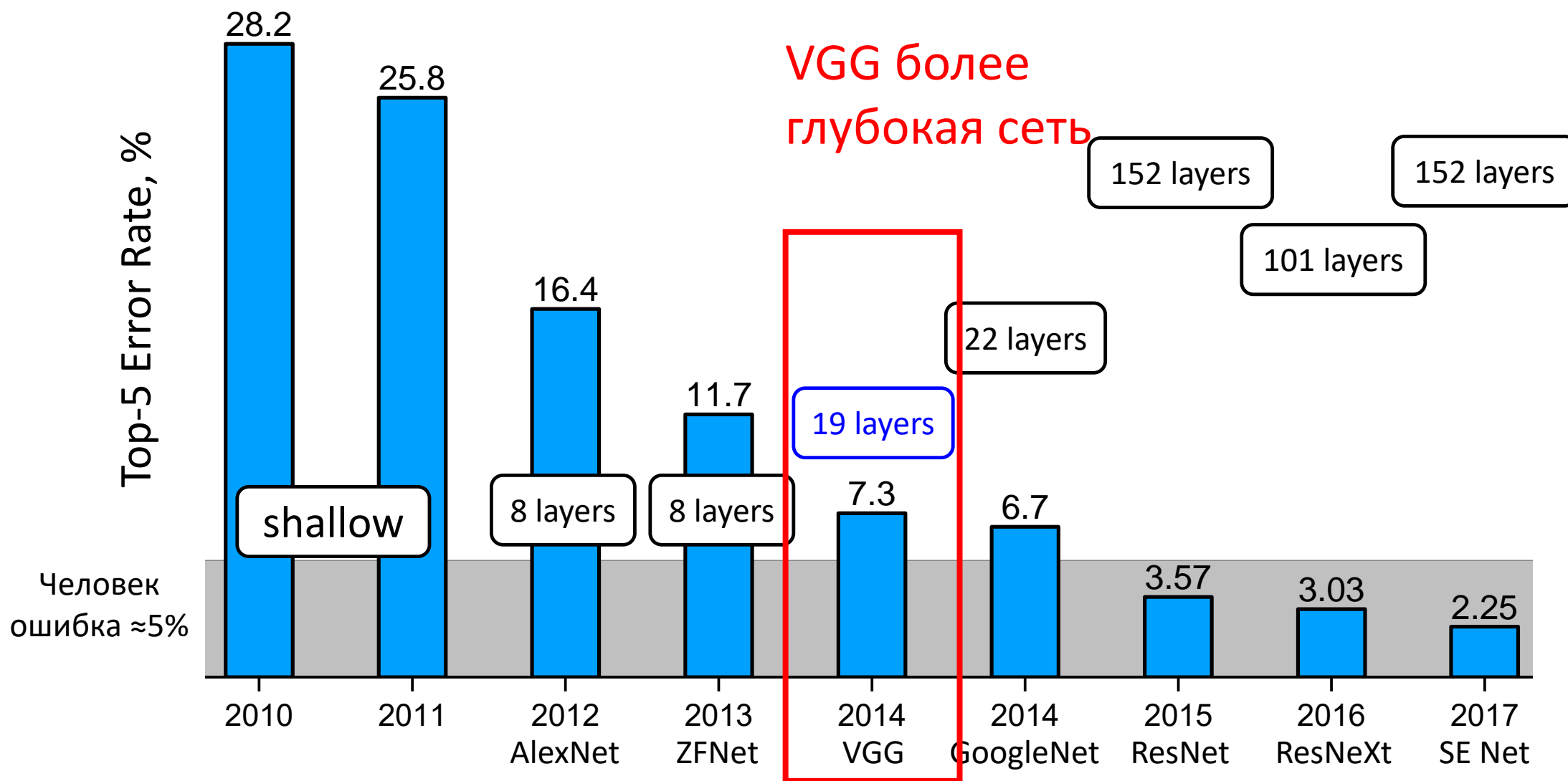
**CONV3,4,5:** 384, 384, 256 filters -> 512, 1024, 512 filters

ImageNet top 5 error: 16.4% (AlexNet) -> 11.7%





# Large Scale Visual Recognition Challenge (ILSVRC)



# VGG

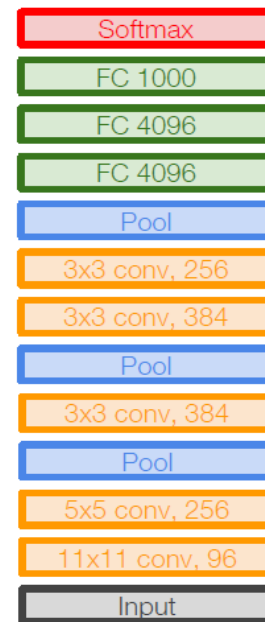
(Simonyan and Zisserman, 2014)

Глубже сеть, меньше свертки (3x3)

AlexNet 8 слоев -> VGG 16 - 19 слоев

Свертки только 3x3 stride 1, pad 1  
и MAX POOL 2x2 stride 2

ImageNet top 5 error: 11.7% (ZFNet) -> 7.3%



AlexNet



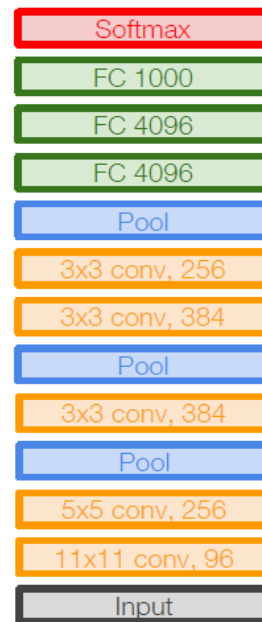
VGG16

VGG19

# VGG

(Simonyan and Zisserman, 2014)

Вопрос: почему авторы используют  
небольшие свертки (3x3)?



AlexNet

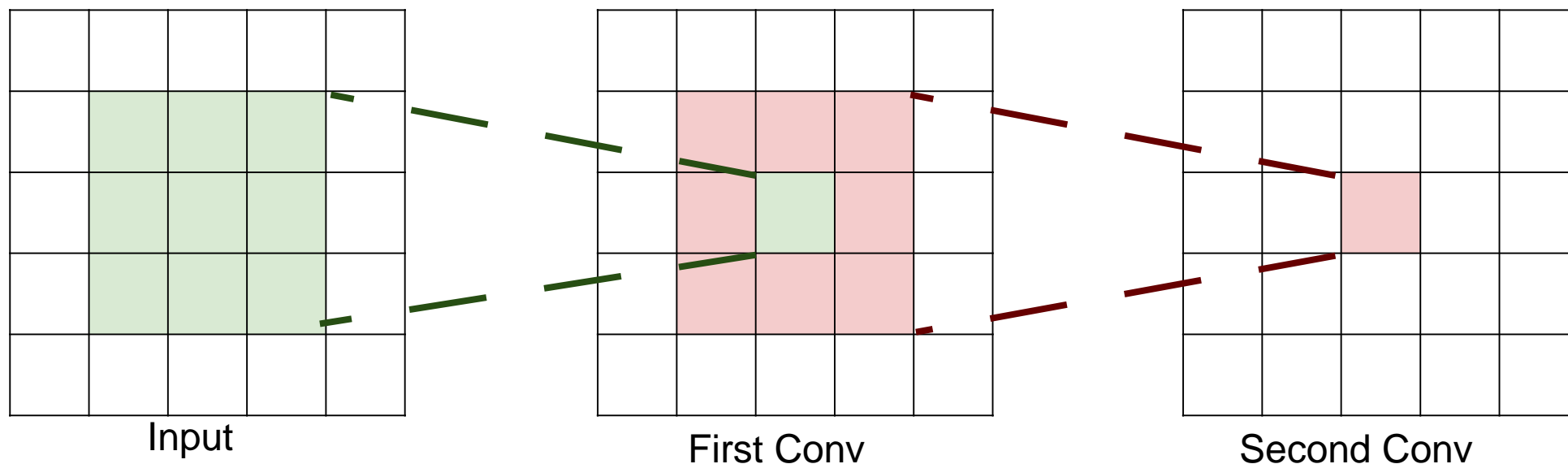


VGG16

VGG19

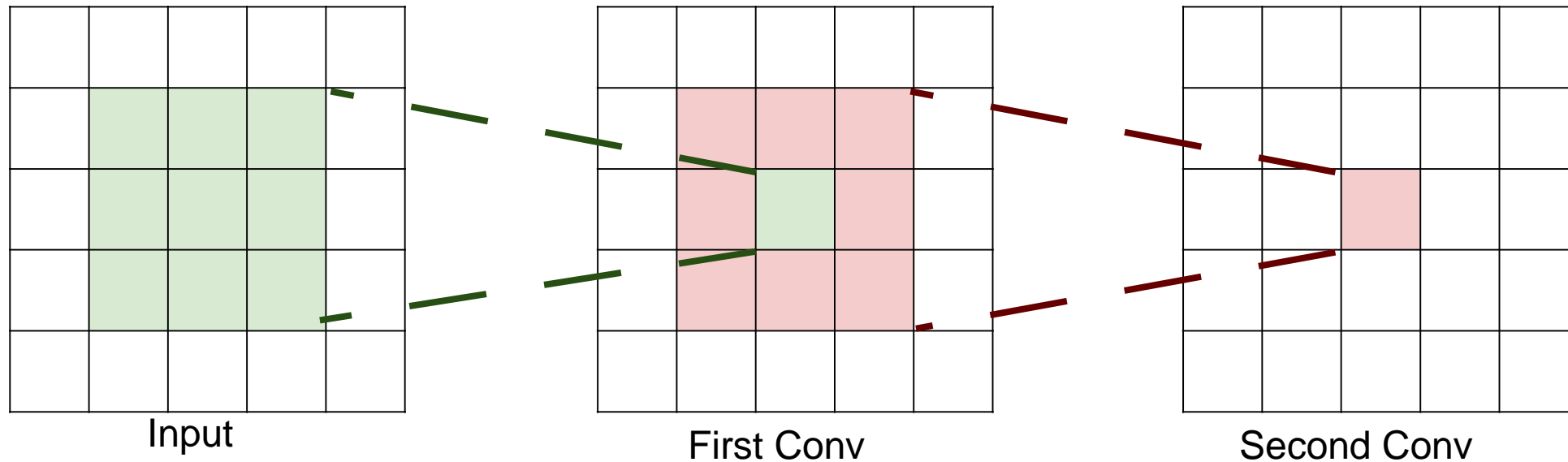
# Преимущества маленьких сверток

Пусть мы применяем последовательно две свертки 3x3 stride 1  
Каждый нейрон видит область 3x3 на предыдущей карте активаций



# Преимущества маленьких сверток

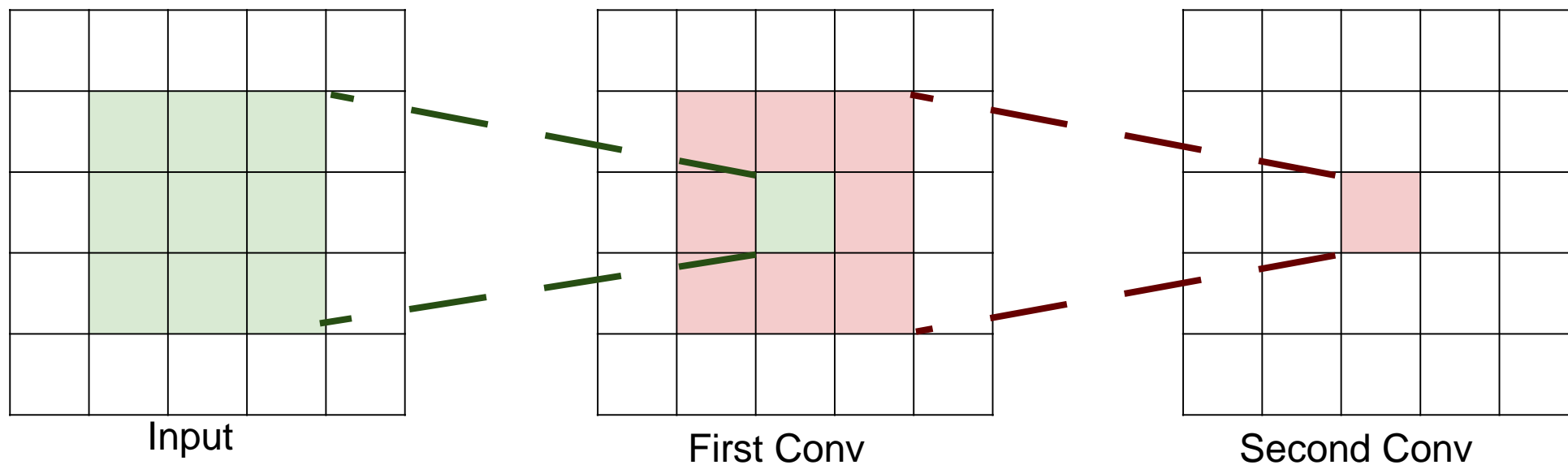
Вопрос: какую область входа видит нейрон во втором сверточном слое?



# Преимущества маленьких сверток

Вопрос: какую область входа видит нейрон во втором сверточном слое?

Ответ: 5x5





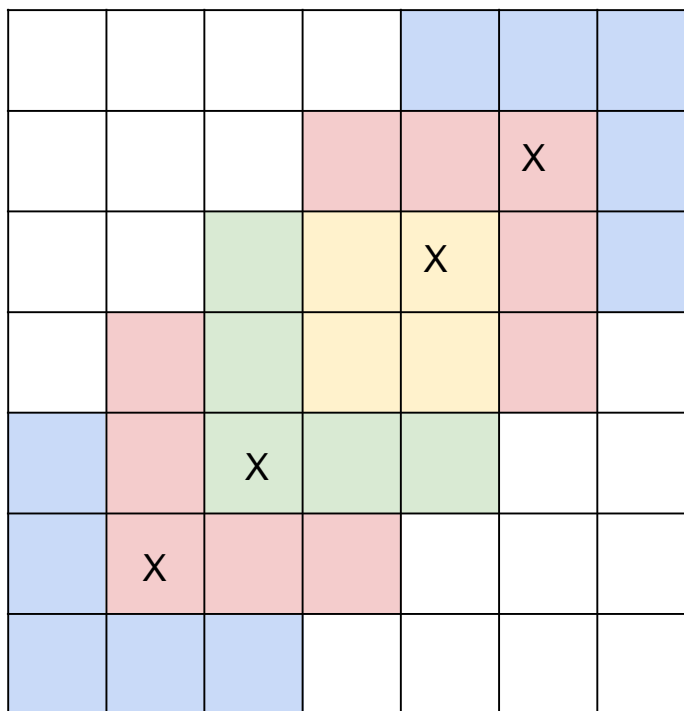
# Преимущества маленьких сверток

Вопрос: если бы мы применили **три** свертки 3x3 stride 1, то какую бы область входа видел нейрон из последнего сверточного слоя?

# Преимущества маленьких сверток

Вопрос: если бы мы применили **три** свертки  $3 \times 3$  stride 1, то какую бы область входа видел нейрон из третьего сверточного слоя?

Ответ:  $7 \times 7$



Три свертки  $3 \times 3$   
аналогичны свертке  $7 \times 7$

# Преимущества маленьких сверток

Пусть размер входа  $H \times W \times C$  и мы применяем  $C$  сверток для сохранения глубины (stride 1, padding такой, чтобы сохранить размеры  $H, W$ )

1 CONV with  $7 \times 7$  filters

3 CONV with  $3 \times 3$  filters

# Преимущества маленьких сверток

Пусть размер входа  $H \times W \times C$  и мы применяем  $C$  сверток для сохранения глубины (stride 1, padding такой, чтобы сохранить размеры  $H, W$ )

1 CONV with 7 x 7 filters

Число параметров:

$$= C \times (7 \times 7 \times C) = 49 C^2$$

3 CONV with 3 x 3 filters

Число параметров:

$$= 3 \times C \times (3 \times 3 \times C) = 27 C^2$$

# Преимущества маленьких сверток

Пусть размер входа  $H \times W \times C$  и мы применяем  $C$  сверток для сохранения глубины (stride 1, padding такой, чтобы сохранить размеры  $H, W$ )

1 CONV with 7 x 7 filters

Число параметров:

$$= C \times (7 \times 7 \times C) = 49 C^2$$

3 CONV with 3 x 3 filters

Число параметров:

$$= 3 \times C \times (3 \times 3 \times C) = 27 C^2$$



Меньше параметров, выше нелинейность!

# Преимущества маленьких сверток

Пусть размер входа  $H \times W \times C$  и мы применяем  $C$  сверток для сохранения глубины (stride 1, padding такой, чтобы сохранить размеры  $H, W$ )

1 CONV with 7 x 7 filters

Число параметров:

$$= C \times (7 \times 7 \times C) = 49 C^2$$

Число умножений и сложений:

3 CONV with 3 x 3 filters

Число параметров:

$$= 3 \times C \times (3 \times 3 \times C) = 27 C^2$$

Число сложений и умножений:



# Преимущества маленьких сверток

Пусть размер входа  $H \times W \times C$  и мы применяем  $C$  сверток для сохранения глубины (stride 1, padding такой, чтобы сохранить размеры  $H, W$ )

1 CONV with 7 x 7 filters

Число параметров:

$$= C \times (7 \times 7 \times C) = 49 C^2$$

Число умножений и сложений:

$$= (H \times W \times C) \times (7 \times 7 \times C)$$

$$= 49 HWC^2$$

3 CONV with 3 x 3 filters

Число параметров:

$$= 3 \times C \times (3 \times 3 \times C) = 27 C^2$$

Число сложений и умножений:

$$= 3 \times (H \times W \times C) \times (3 \times 3 \times C)$$

$$= 27 HWC^2$$

# Преимущества маленьких сверток

Пусть размер входа  $H \times W \times C$  и мы применяем  $C$  сверток для сохранения глубины (stride 1, padding такой, чтобы сохранить размеры  $H, W$ )

1 CONV with 7 x 7 filters

Число параметров :

$$= C \times (7 \times 7 \times C) = 49 C^2$$

Число умножений и сложений:

$$= (H \times W \times C) \times (7 \times 7 \times C)$$

$$= 49 HWC^2$$

3 CONV with 3 x 3 filters

Число параметров :

$$= 3 \times C \times (3 \times 3 \times C) = 27 C^2$$

Число сложений и умножений:

$$= 3 \times (H \times W \times C) \times (3 \times 3 \times C)$$

$$= 27 HWC^2$$



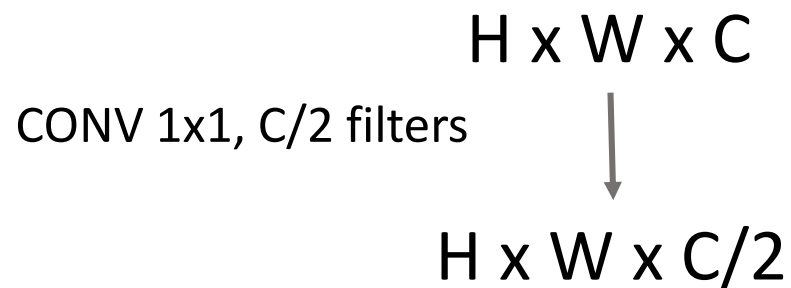
Меньше вычислений, выше нелинейность!

# Преимущества маленьких сверток

Как использовать свертки 1x1?

# Преимущества маленьких сверток

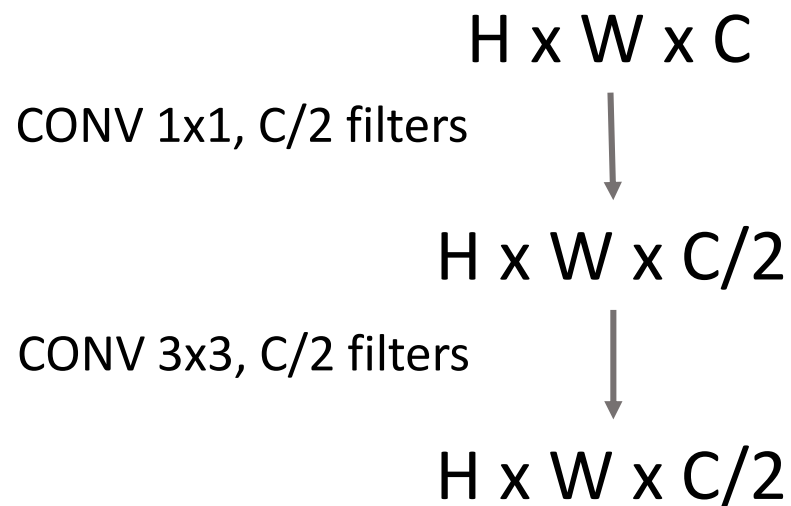
Как использовать свертки 1x1?



1. уменьшаем число каналов с помощью “bottleneck” 1 x 1 conv

# Преимущества маленьких сверток

Как использовать свертки 1x1?

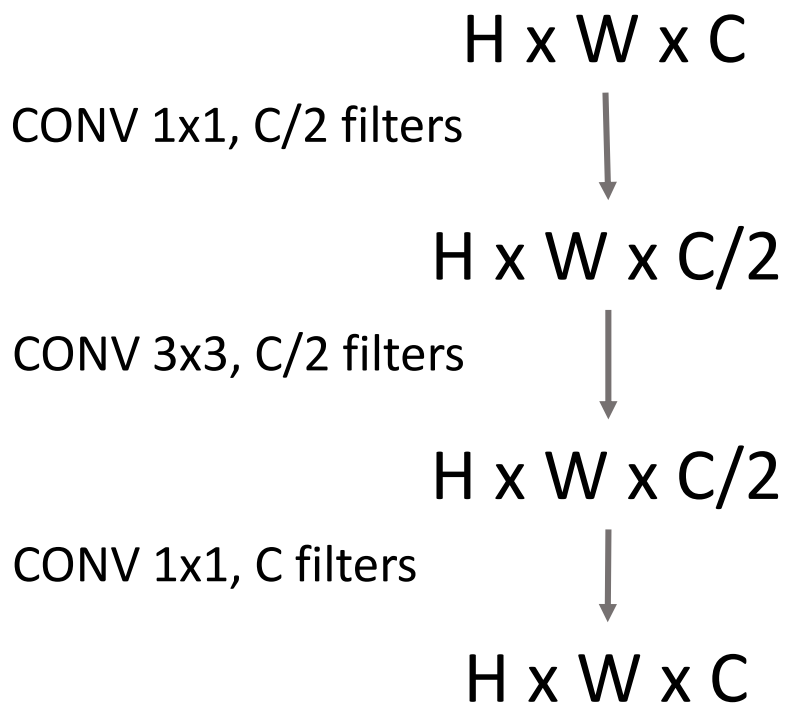


1. уменьшаем число каналов с помощью “bottleneck” 1 x 1 conv

2. применяем 3 x 3 conv

# Преимущества маленьких сверток

Как использовать свертки 1x1?



1. уменьшаем число каналов с помощью “bottleneck” 1 x 1 conv

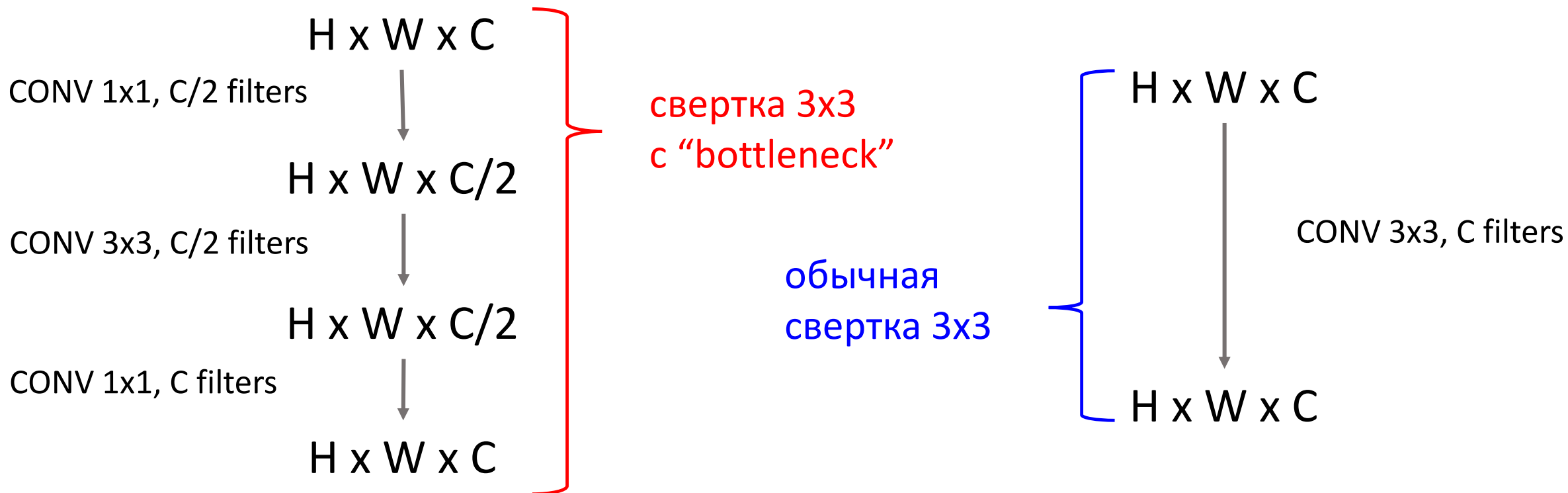
2. применяем 3 x 3 conv

3. восстанавливаем число каналов с помощью 1 x 1 conv

Подобный подход использовался в GoogleNet, ResNet и подобным им сетям, например, DenseNet

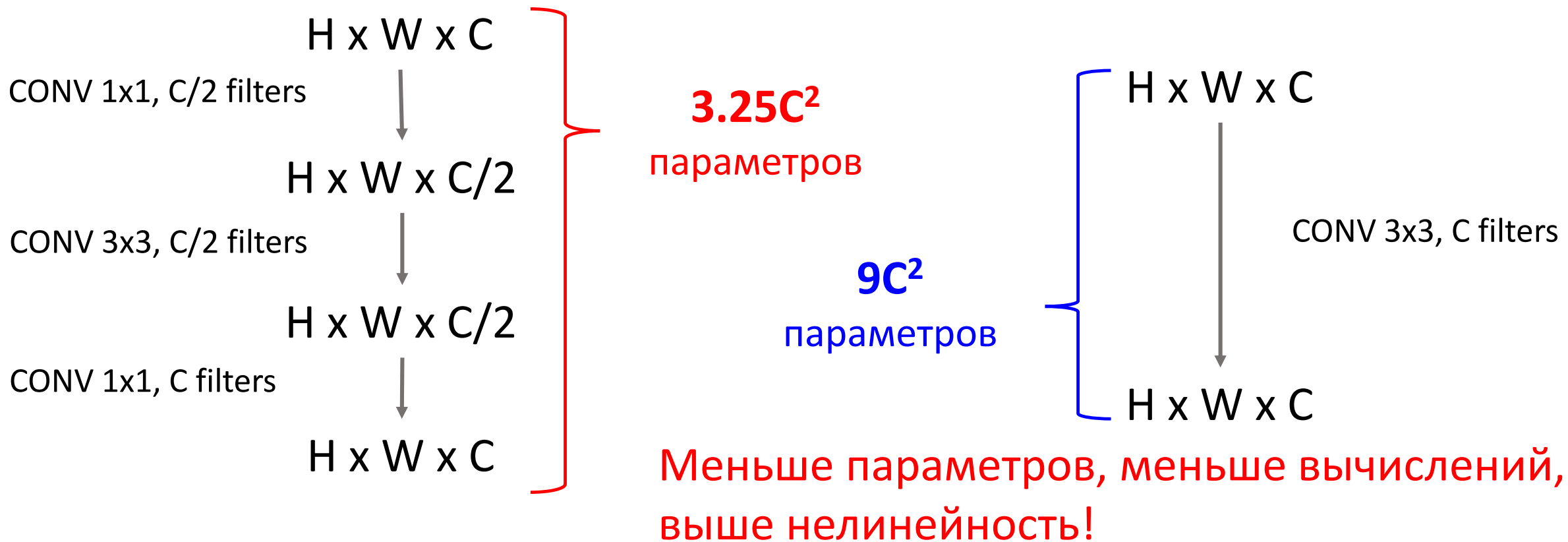
# Преимущества маленьких сверток

Как использовать свертки 1x1?



# Преимущества маленьких сверток

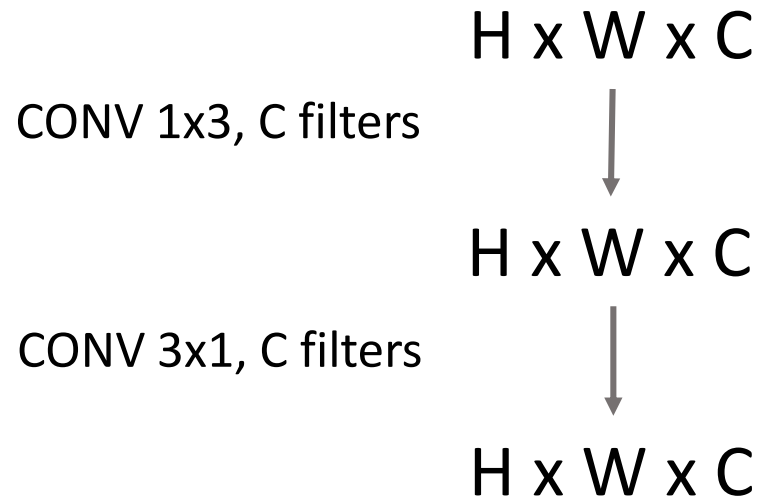
Как использовать свертки 1x1?





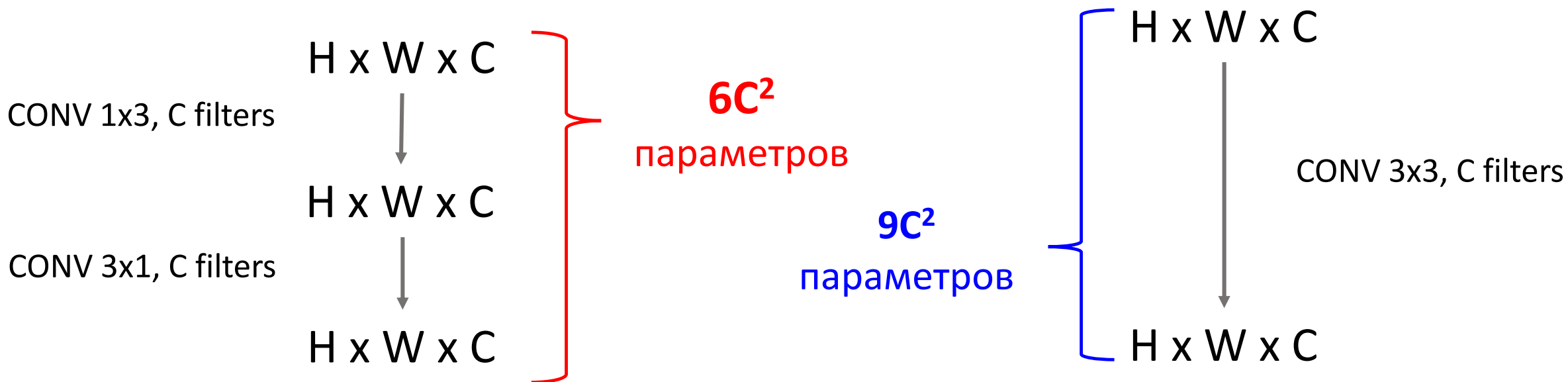
# Преимущества маленьких сверток

Совсем избавиться от сверток  $3 \times 3$  мы не можем.  
Можем ли мы их упростить?



# Преимущества маленьких сверток

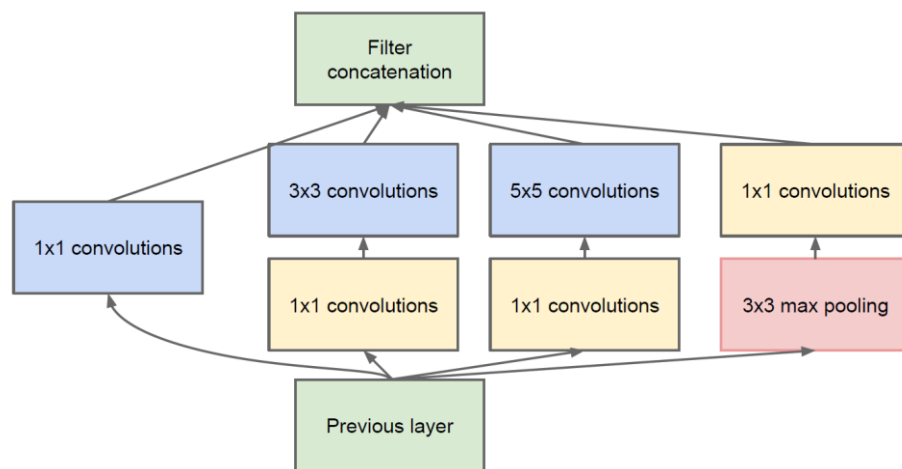
Совсем избавиться от сверток 3x3 мы не можем.  
Можем ли мы их упростить?



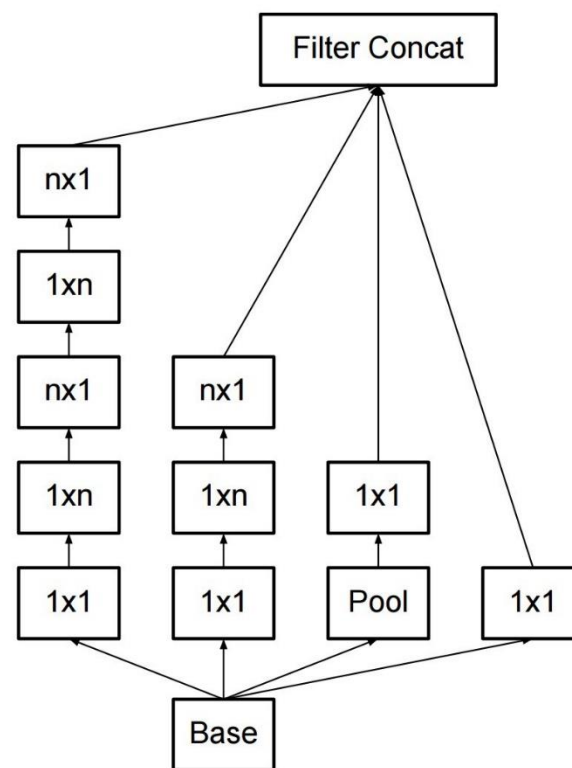
Меньше параметров, меньше  
вычислений, выше нелинейность!

# Преимущества маленьких сверток

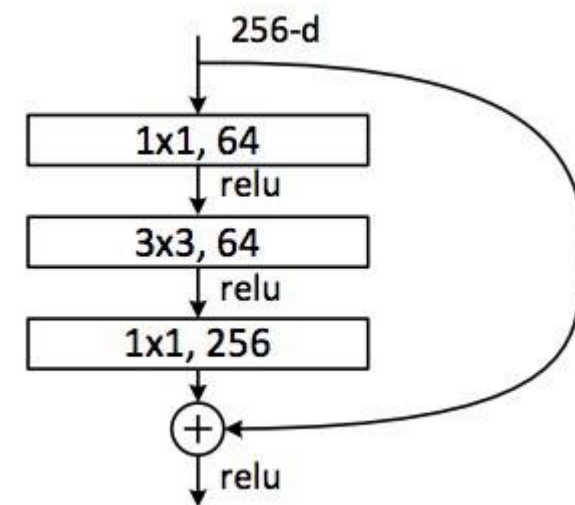
Inception v1  
(GoogleNet)



Inception v3



ResNet



# VGG

(Simonyan and Zisserman, 2014)

INPUT: [224x224x3] memory:  $224*224*3=150K$  params: 0

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

...

...

CONV3-512: [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

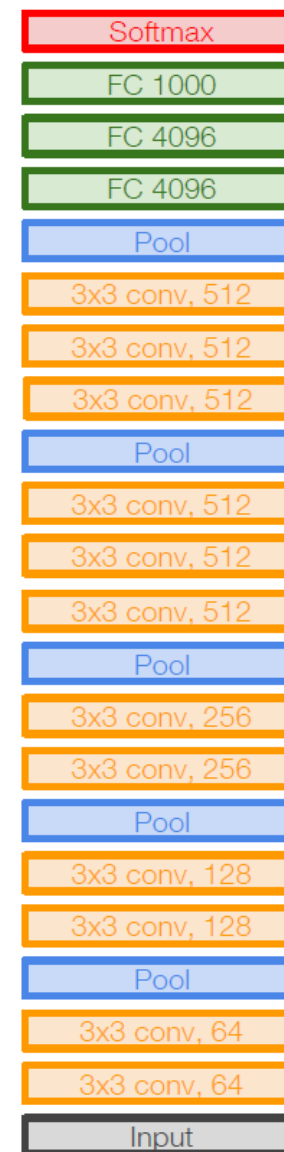
FC: [1x1x4096] memory: 4096 params:  $7*7*512*4096 = 102,760,448$

FC: [1x1x4096] memory: 4096 params:  $4096*4096 = 16,777,216$

FC: [1x1x1000] memory: 1000 params:  $4096*1000 = 4,096,000$

**TOTAL memory: 24M \* 4 bytes  $\approx$  96MB / image** (only forward!  $\approx$ x2 for backward)

**TOTAL params: 138M parameters**



VGG16

# VGG

(Simonyan and Zisserman, 2014)

INPUT: [224x224x3] memory:  $224*224*3=150\text{K}$  params: 0

CONV3-64: [224x224x64] memory:  $224*224*64=3.2\text{M}$  params:  $(3*3*3)*64 = 1,728$

CONV3-64: [224x224x64] memory:  $224*224*64=3.2\text{M}$  params:  $(3*3*64)*64 = 36,864$

POOL2: [112x112x64] memory:  $112*112*64=800\text{K}$  params: 0

CONV3-128: [112x112x128] memory:  $112*112*128=1.6\text{M}$  params:  $(3*3*64)*128 = 73,728$

CONV3-128: [112x112x128] memory:  $112*112*128=1.6\text{M}$  params:  $(3*3*128)*128 = 147,456$

POOL2: [56x56x128] memory:  $56*56*128=400\text{K}$  params: 0

...

...

CONV3-512: [14x14x512] memory:  $14*14*512=100\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory:  $14*14*512=100\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory:  $14*14*512=100\text{K}$  params:  $(3*3*512)*512 = 2,359,296$

POOL2: [7x7x512] memory:  $7*7*512=25\text{K}$  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:  $4096*1000 = 4,096,000$

**TOTAL memory:**  $24\text{M} * 4 \text{ bytes} \approx 96\text{MB}$  / image (only forward!  $\approx x2$  for backward)

**TOTAL params:** 138M parameters

Самые большие расходы  
памяти в первых сверточных  
слоях

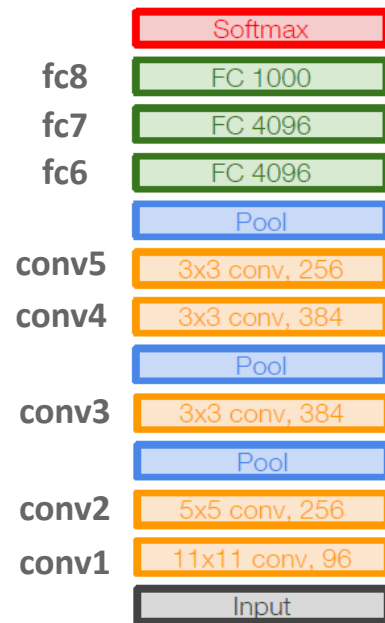
Больше всего параметров в  
выходных полно-связанных  
слоях

# VGG

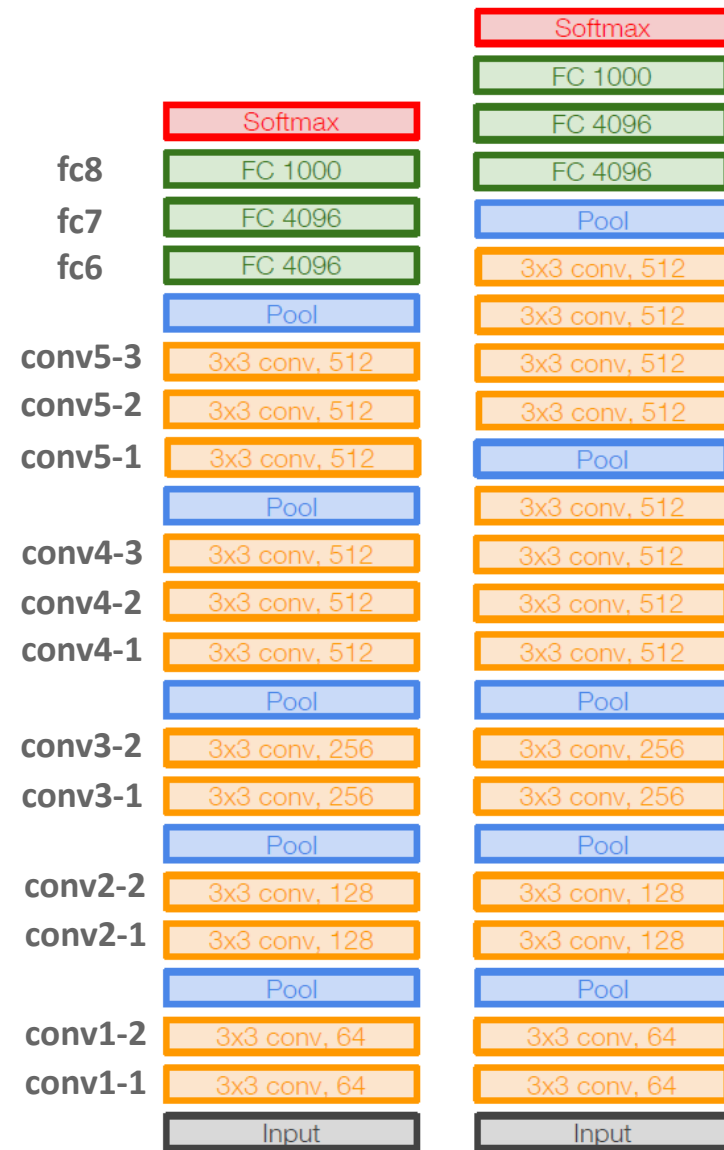
(Simonyan and Zisserman, 2014)

## Особенности:

- ImageNet'14 в классификации 2-е место, в локализации 1-е
- процедура тренировки аналогична AlexNet
- признаки из fc7 слоя хорошо подходят для решения задач компьютерного зрения
- VGG16,  $\approx 140$  млн. параметров,  $\approx 530$  Мбайт caffemodel



AlexNet

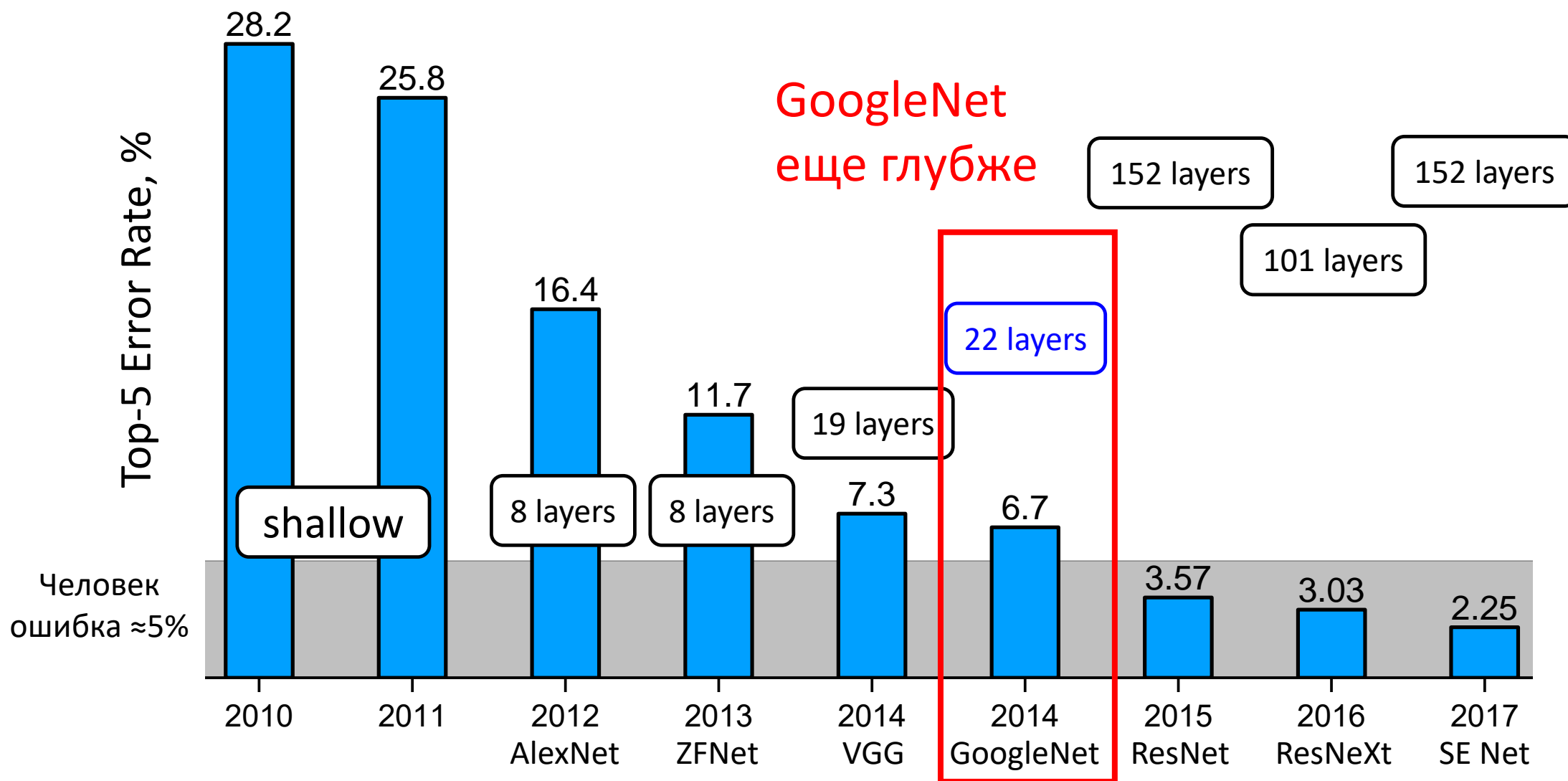


VGG16

VGG19



# Large Scale Visual Recognition Challenge (ILSVRC)



# GoogleNet

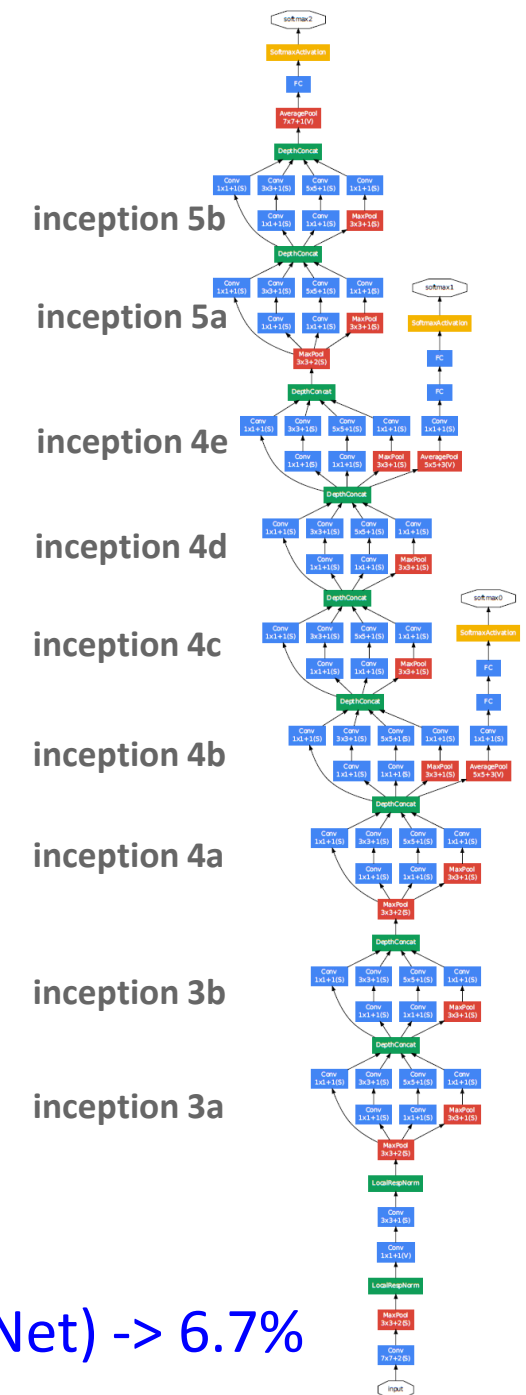
(Szegedy et al., 2014)

Еще глубже, оптимизированная архитектура для высокой скорости вычислений

## Особенности:

- победитель ImageNet'14 в классификации
- 22 слоя
- 9 Inception модулей
- нет полно-связанных слоев (AVE POOL)
- $\approx 5$  млн. параметров, в 12 раз меньше, чем в AlexNet и в 2 раза меньше вычислений!
- $\approx 50$  Мбайт caffemodel

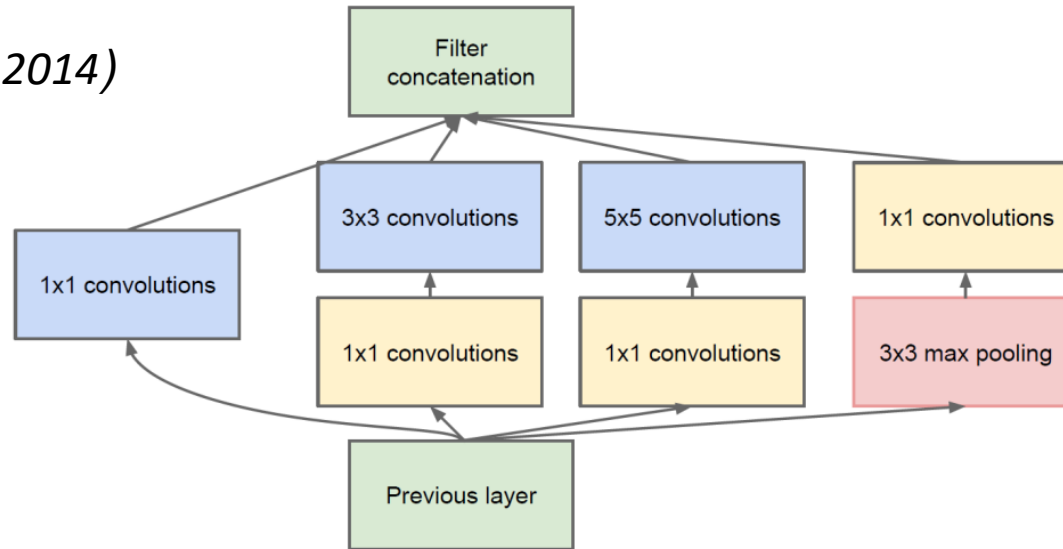
ImageNet top 5 error: 11.7% (ZFNet) -> 6.7%





# GoogleNet

(Szegedy et al., 2014)



## Inception модуль

- 4 канала: свертки 1x1, 3x3, 5x5 и MAX POOL
- свертки 1x1, 3x3, 5x5
  - кластеризуют нейроны с коррелирующими активациями, выше разреженность активаций
  - обработка разных пространственных масштабов
- “bottleneck” уменьшает сложность вычислений

inception 5b

inception 5a

inception 4e

inception 4d

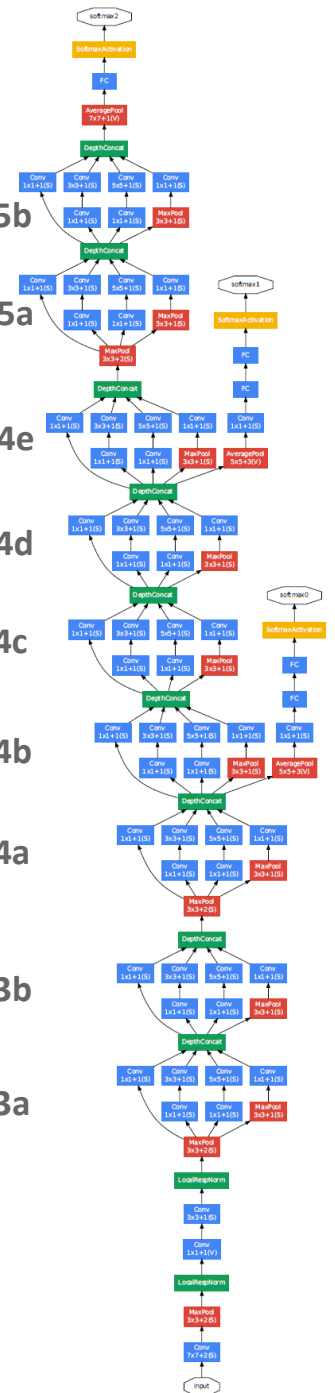
inception 4c

inception 4b

inception 4a

inception 3b

inception 3a



# GoogleNet

(Szegedy et al., 2014)

## Архитектура GoogleNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

inception 5b

inception 5a

inception 4e

inception 4d

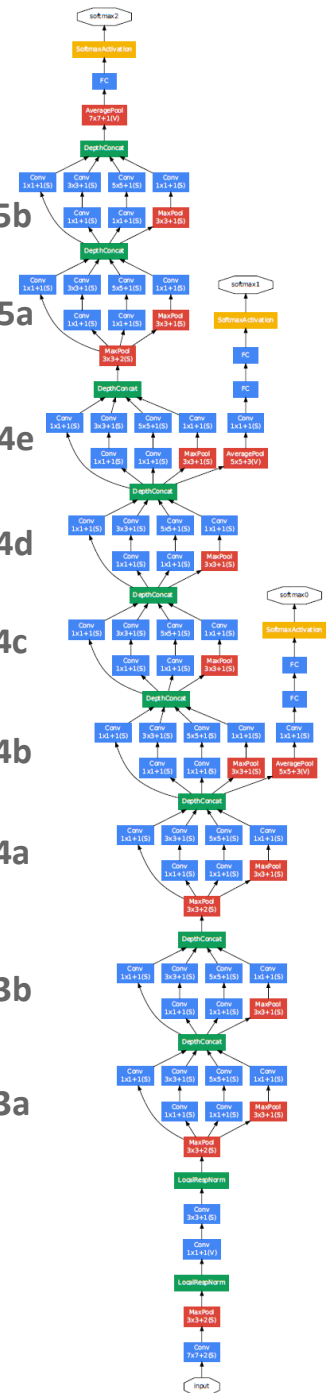
inception 4c

inception 4b

inception 4a

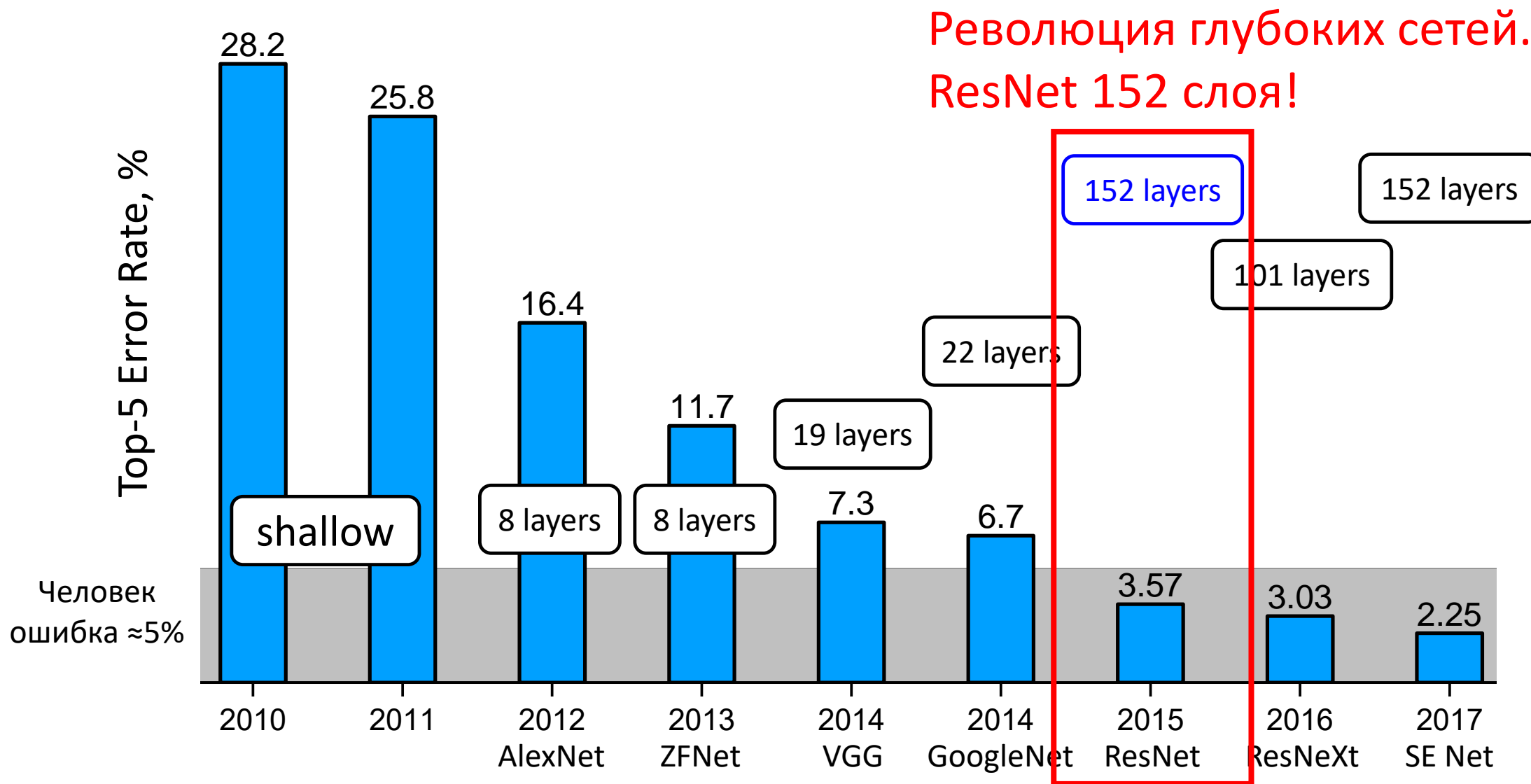
inception 3b

inception 3a





# Large Scale Visual Recognition Challenge (ILSVRC)



# ResNet

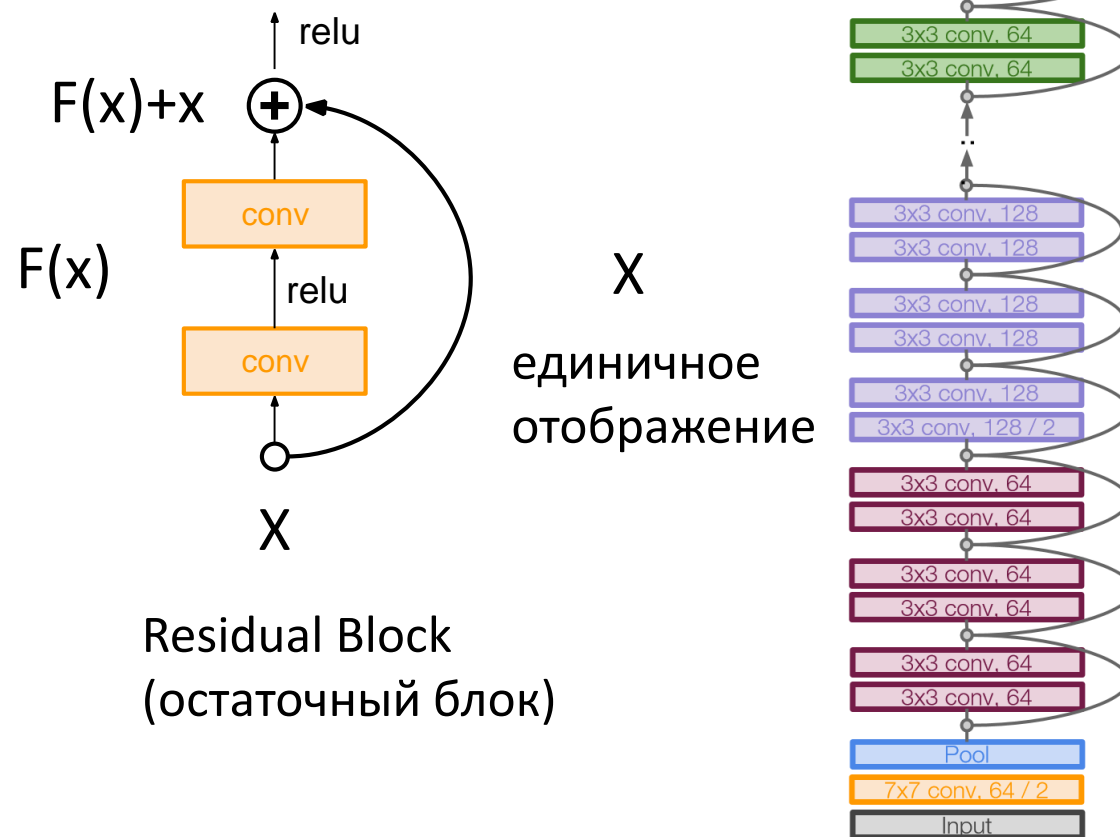
(He et al., 2015)

Очень глубокая сеть за счет  
остаточных соединений  
(residual connections)

152 слоя в моделях для ImageNet'15

Победитель в классификации,  
детекции и сегментации в конкурсах  
ImageNet'15 и COCO'15

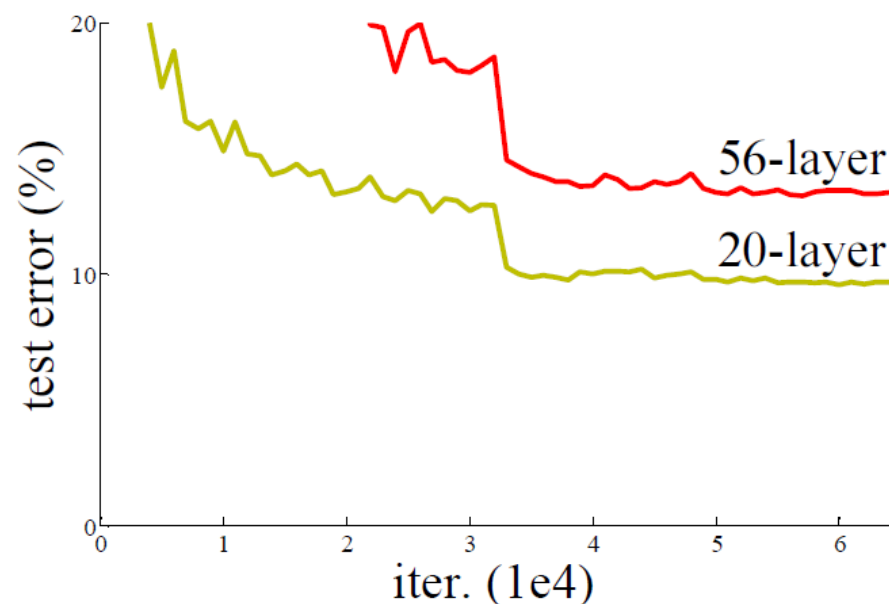
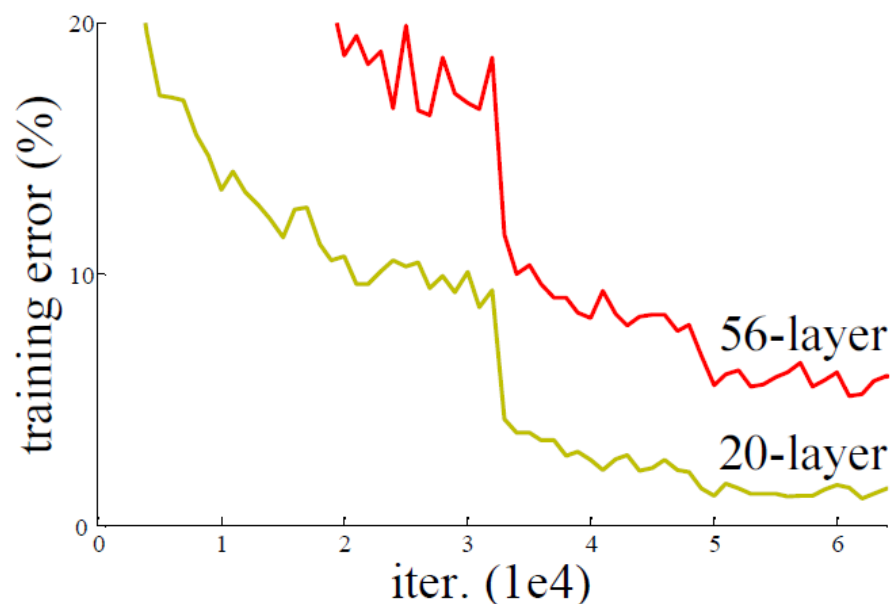
ImageNet top 5 error: 6.7% (GoogleNet) -> 3.57% - **точнее человека (≈5%)!**



# ResNet

(He et al., 2015)

Графики ошибки на CIFAR-10 в зависимости от глубины обычной нейронной сети  
[ [CONV-ReLu]xN - [POOL] ]xM

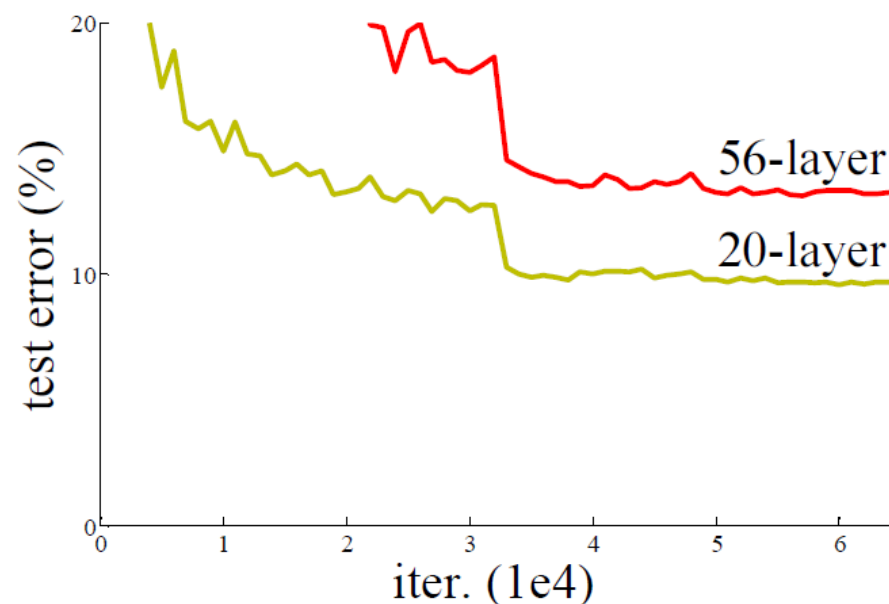
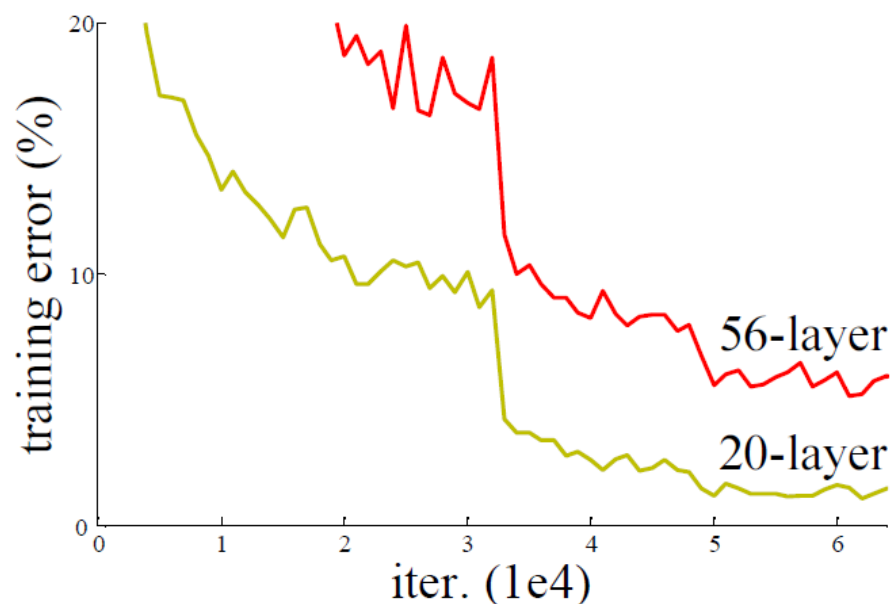


Вопрос: что странного в этих графиках ошибки на Train и Test данных?

# ResNet

(He et al., 2015)

Графики ошибки на CIFAR-10 в зависимости от глубины обычной нейронной сети  
[ [CONV-ReLu]xN - [POOL] ]xM



Вопрос: что странного в этих графиках ошибки на Train и Test данных?  
Точность падает с ростом глубины, и это не связано с переобучением!

# ResNet

*(He et al., 2015)*

Идея: проблема падения точности с ростом глубины сетей –  
проблема *оптимизации*

# ResNet

*(He et al., 2015)*

Идея: проблема падения точности с ростом глубины сетей –  
проблема *оптимизации*

Мы хотим, чтобы более глубокие сети имели точность не хуже, чем неглубокие.

Как построить такую глубокую сеть?

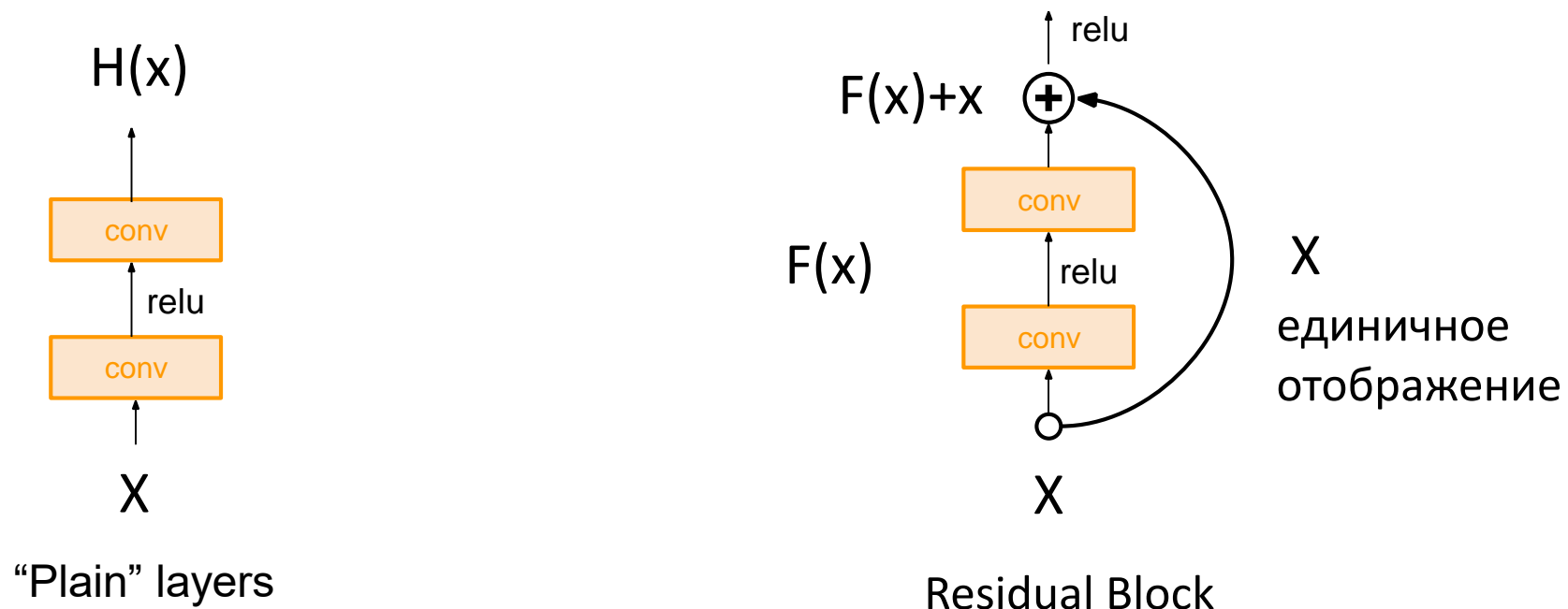
Очевидное решение методом построения: взять неглубокую сеть и добавить слои, которые копируют вход (identity mapping)



# ResNet

(He et al., 2015)

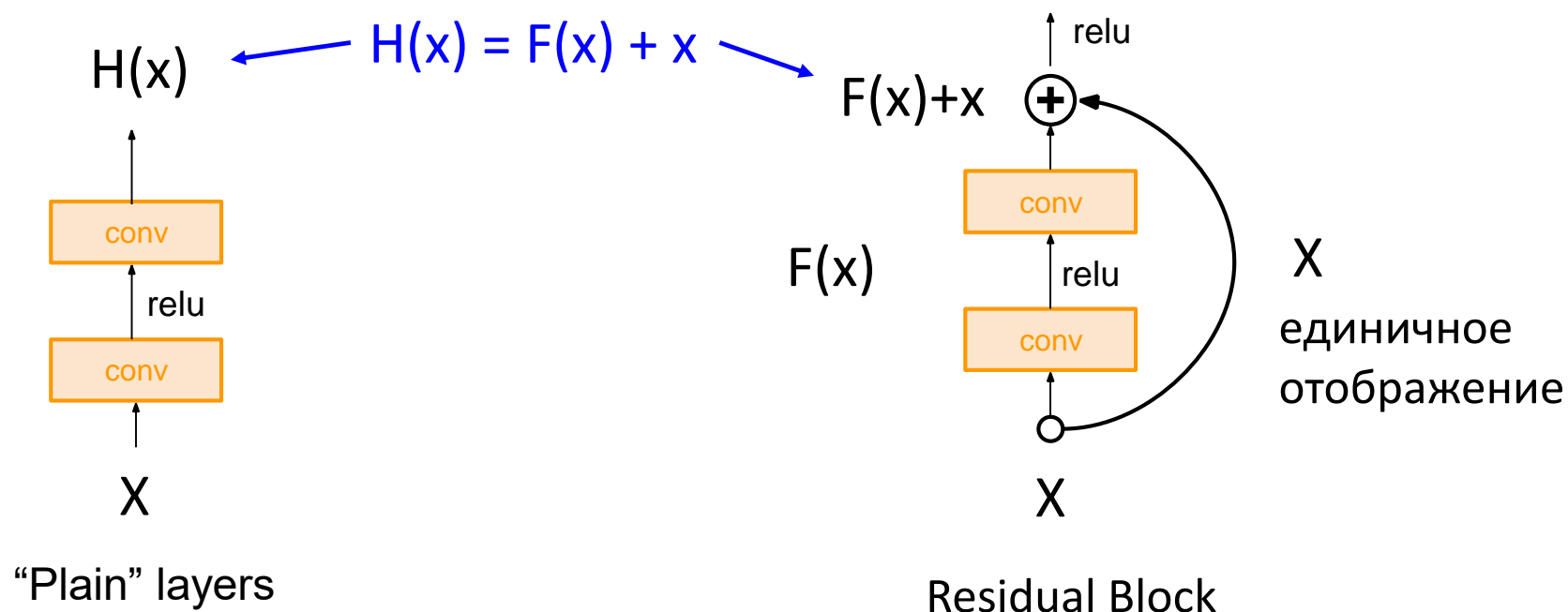
Решение: учим нейронную сеть для нахождения добавки/остатка к единичному отображению, вместо вычисления самого отображения



# ResNet

(He et al., 2015)

Решение: учим нейронную сеть для нахождения добавки/остатка к единичному отображению, вместо вычисления самого отображения



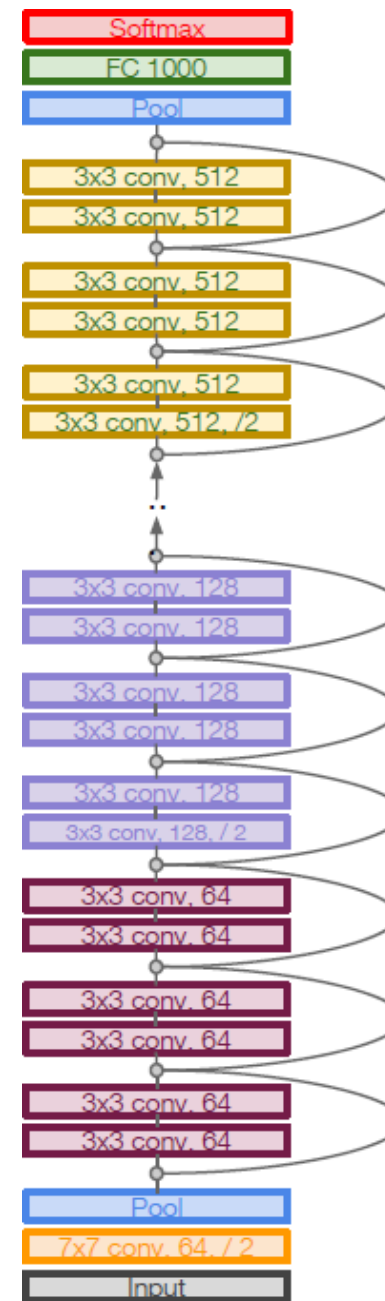
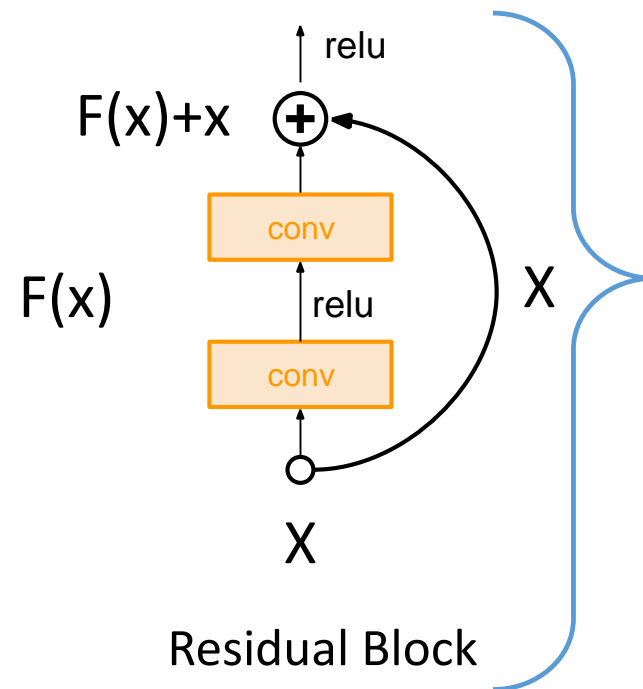
Используем слои  
сети для  
определения  
 $F(x) = H(x) - x$   
вместо  
 $H(x)$

# ResNet

(He et al., 2015)

Пример архитектуры ResNet-34

- Стэк из residual blocks (2 CONV 3x3)

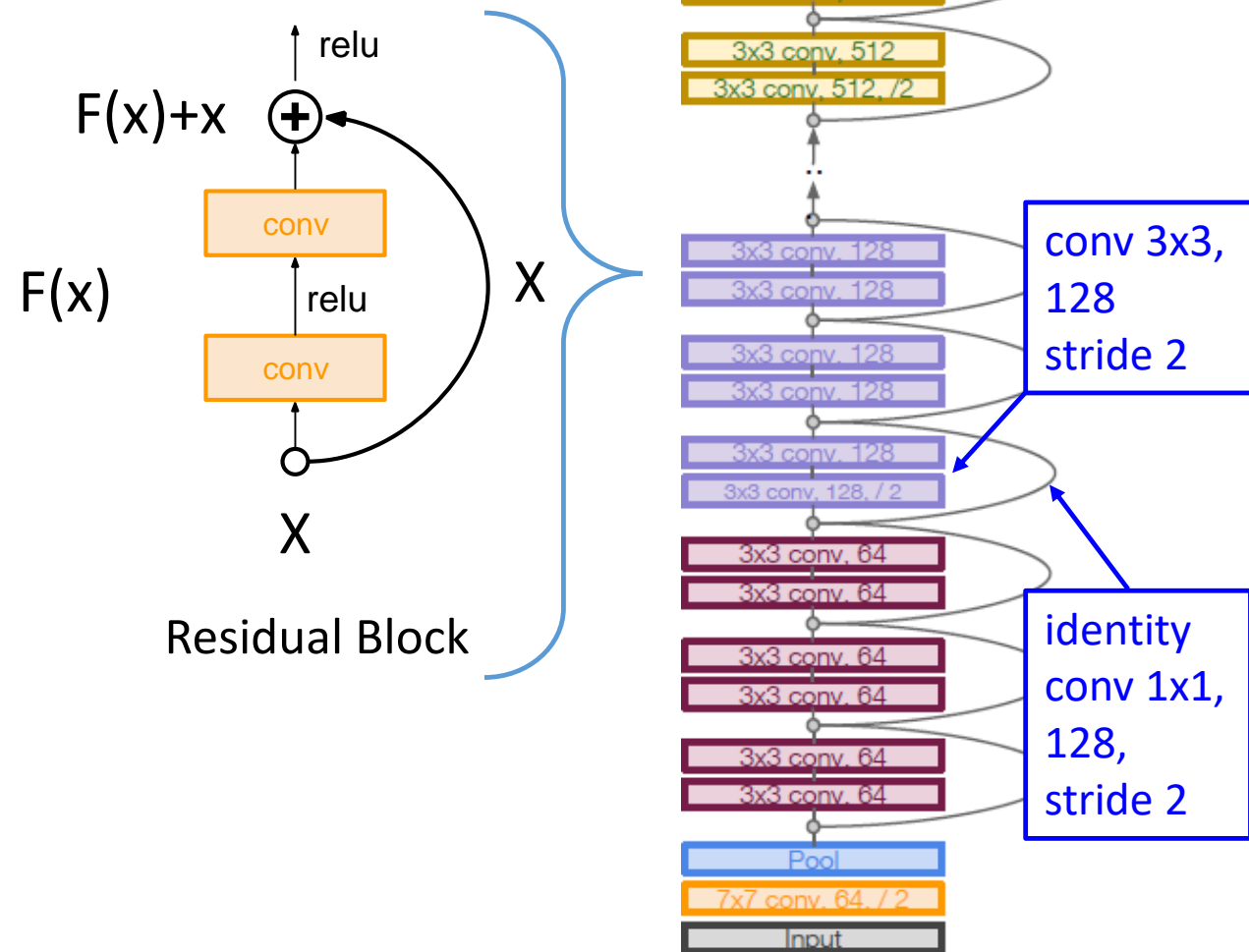


# ResNet

(He et al., 2015)

Пример архитектуры ResNet-34

- Стэк из residual blocks (2 CONV 3x3)
- Число фильтров удваивается, когда пространственное разрешение уменьшается в 2 раза (stride 2, в identity CONV 1x1)

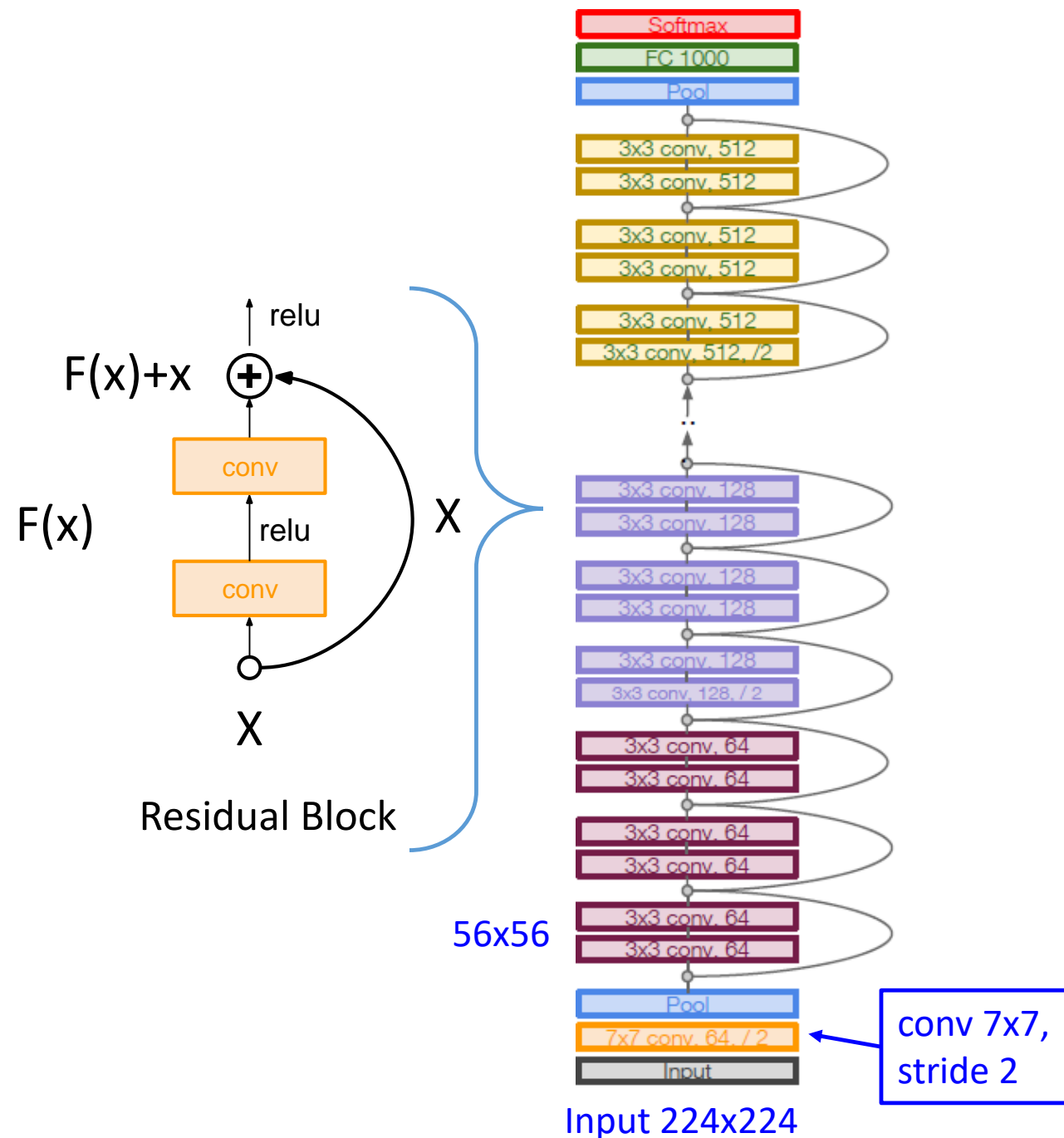


# ResNet

(He et al., 2015)

## Пример архитектуры ResNet-34

- Стэк из residual blocks (2 CONV 3x3)
- Число фильтров удваивается, когда пространственное разрешение уменьшается в 2 раза (stride 2, в identity CONV 1x1)
- Сразу после входа CONV 7x7 stride 2 + MAX POOL 3x3 stride 2

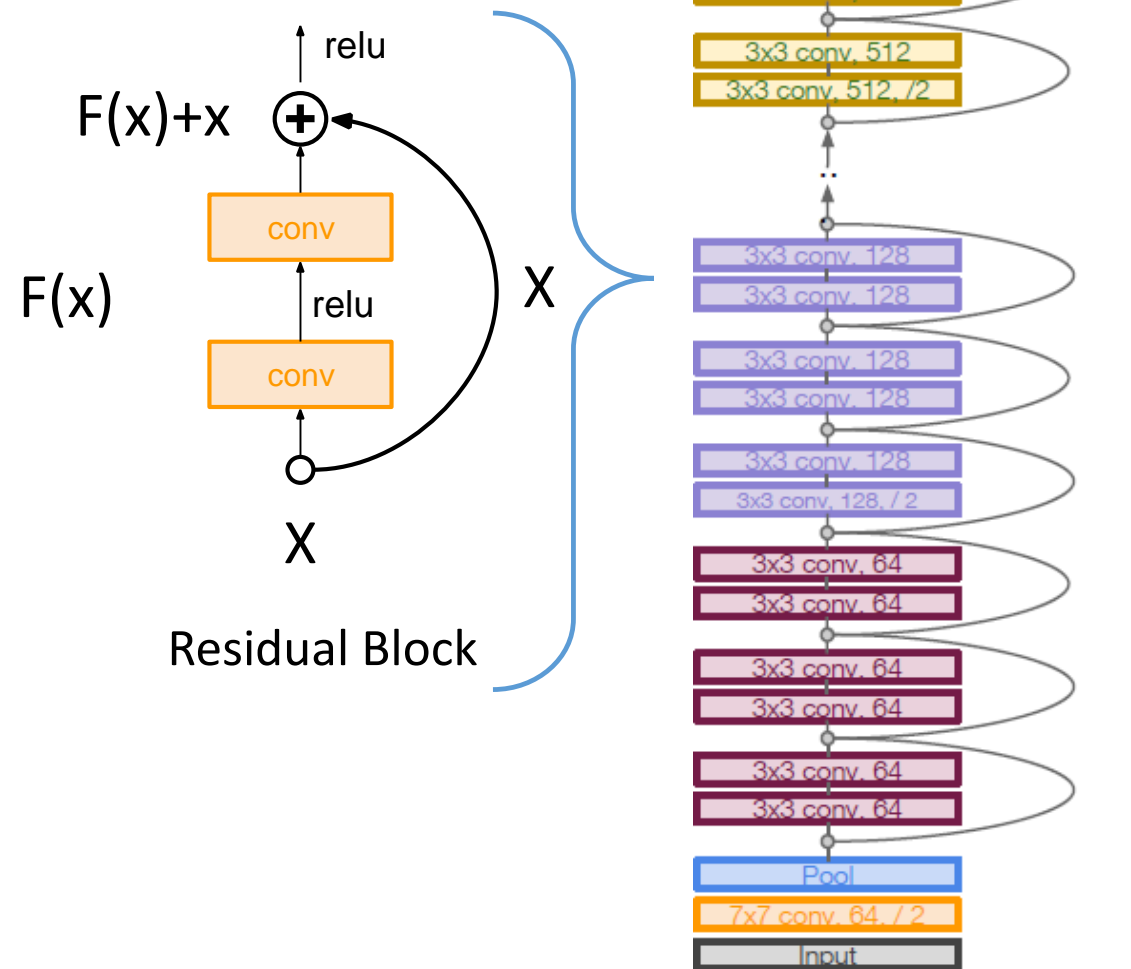


# ResNet

(He et al., 2015)

Пример архитектуры ResNet-34

- Стэк из residual blocks (2 CONV 3x3)
- Число фильтров удваивается, когда пространственное разрешение уменьшается в 2 раза (stride 2, в identity CONV 1x1)
- Сразу после входа CONV 7x7 stride 2 + MAX POOL 3x3 stride 2
- Полно-связанный слой только на выходе сети (1000 классов) после AVE POOL

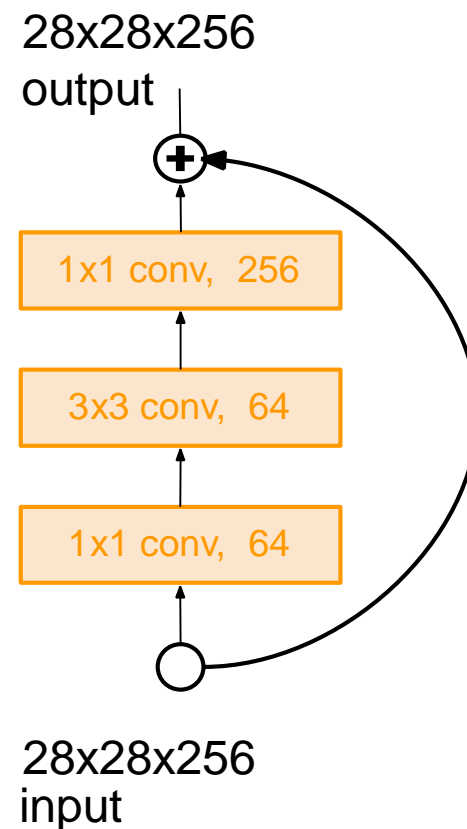


# ResNet

(He et al., 2015)

Глубина сетей ResNet 34, 50, 101  
или 152 слоя

Для глубоких сетей  
(ResNet-50+), авторы  
использовали “bottleneck” для  
уменьшения сложности  
вычислений  
(аналогично GoogleNet)

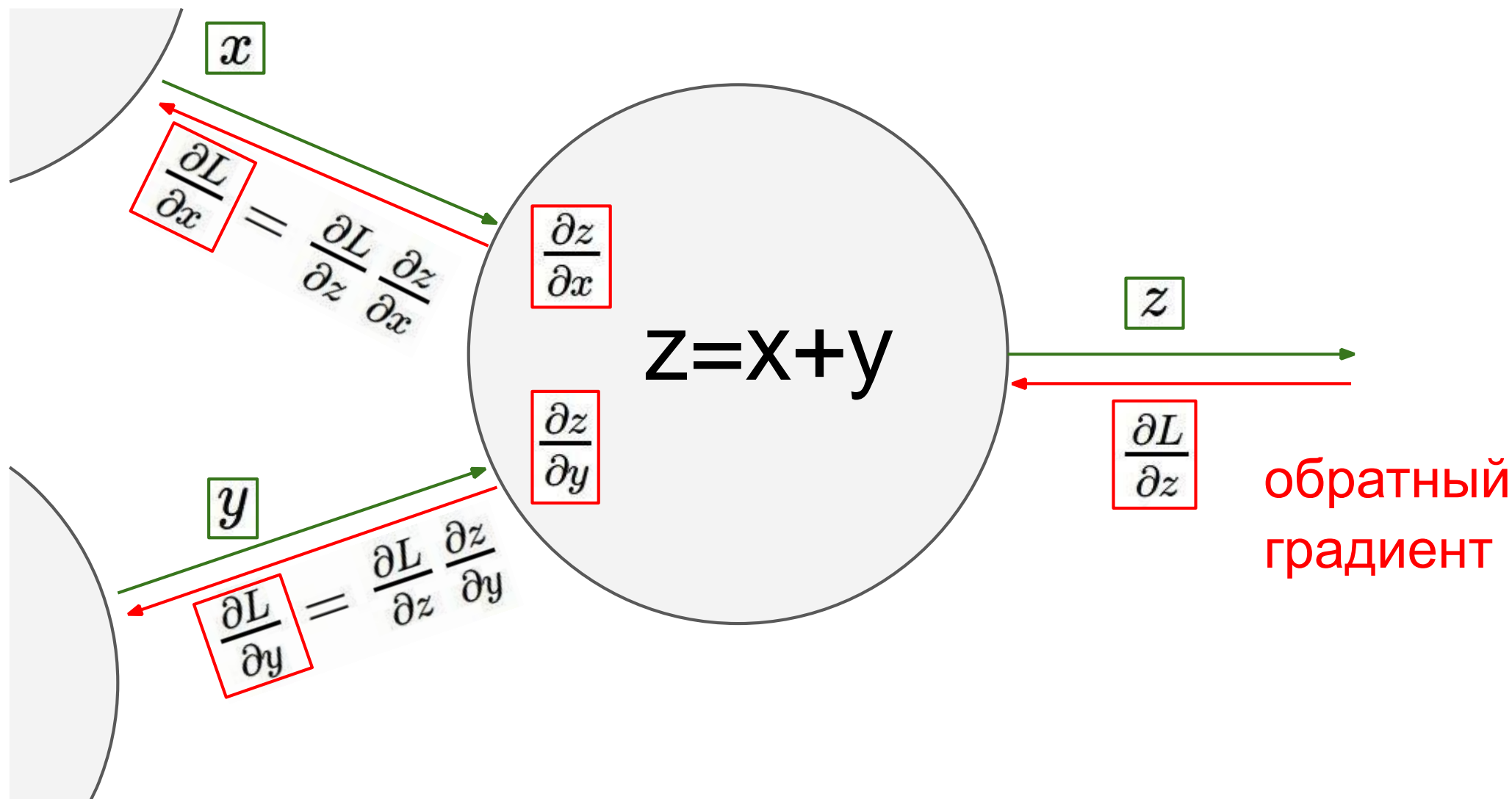


# Почему ResNet проще обучать?

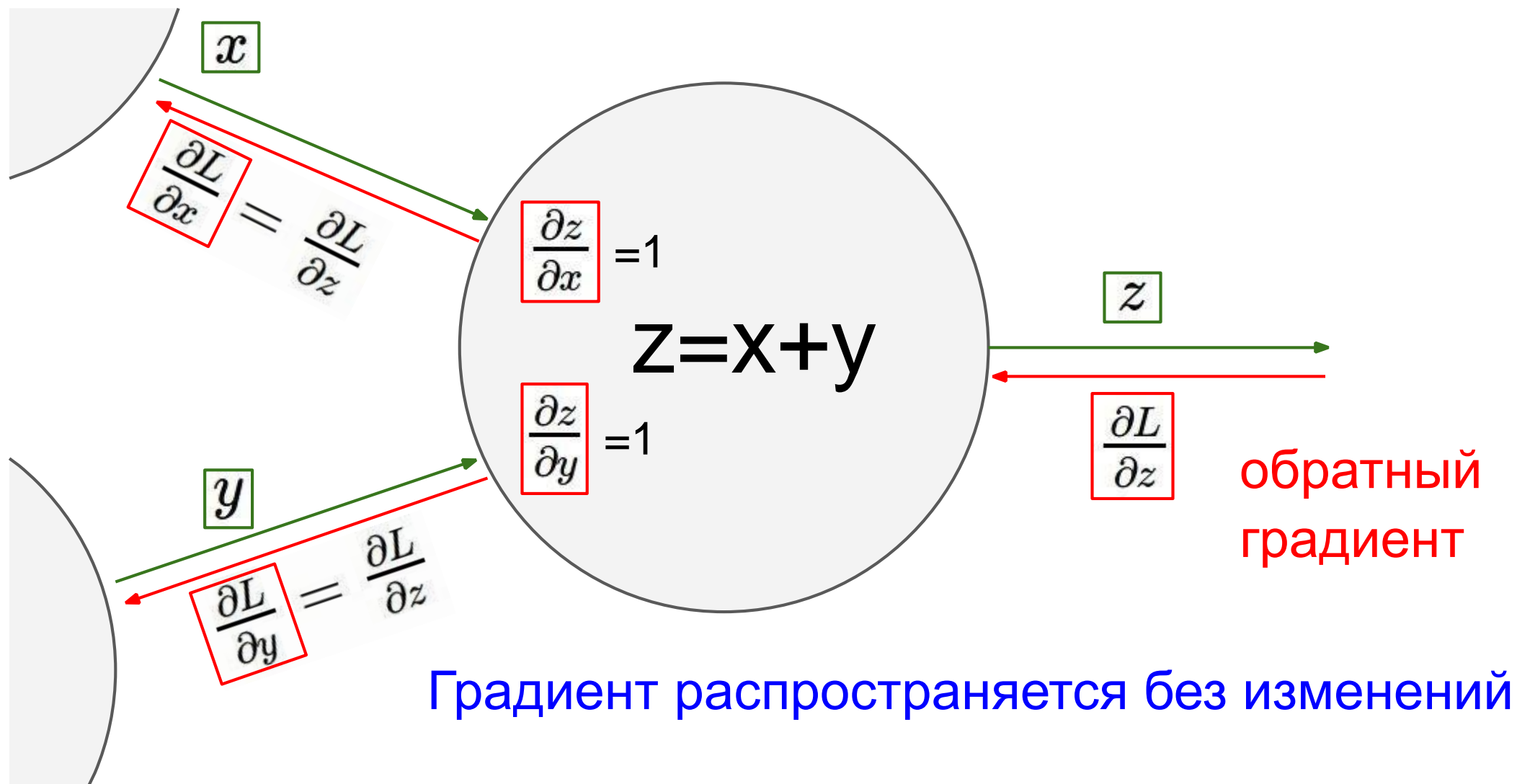
- Градиент ошибки легко распространяется назад



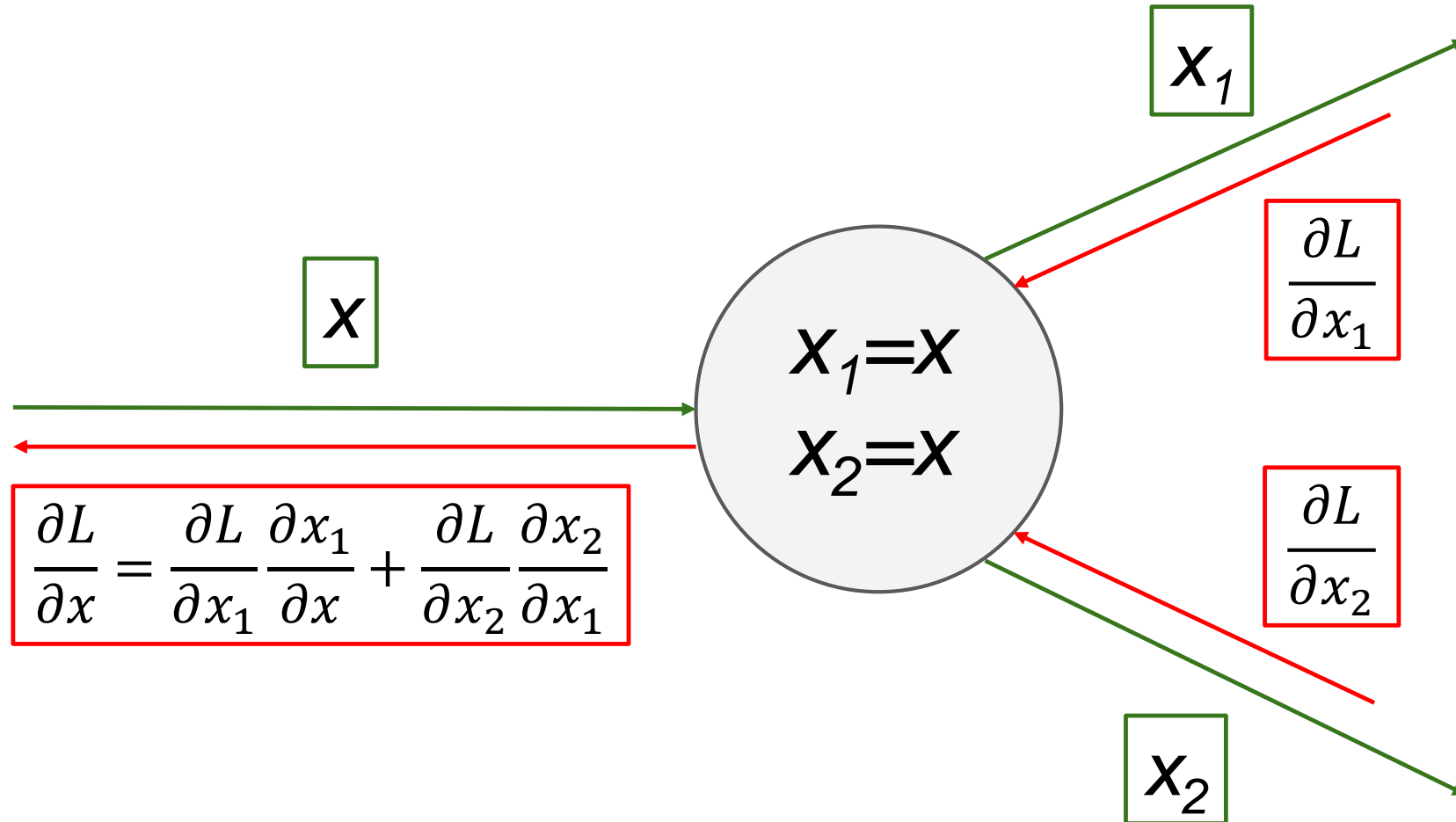
# Обратный градиент при суммировании



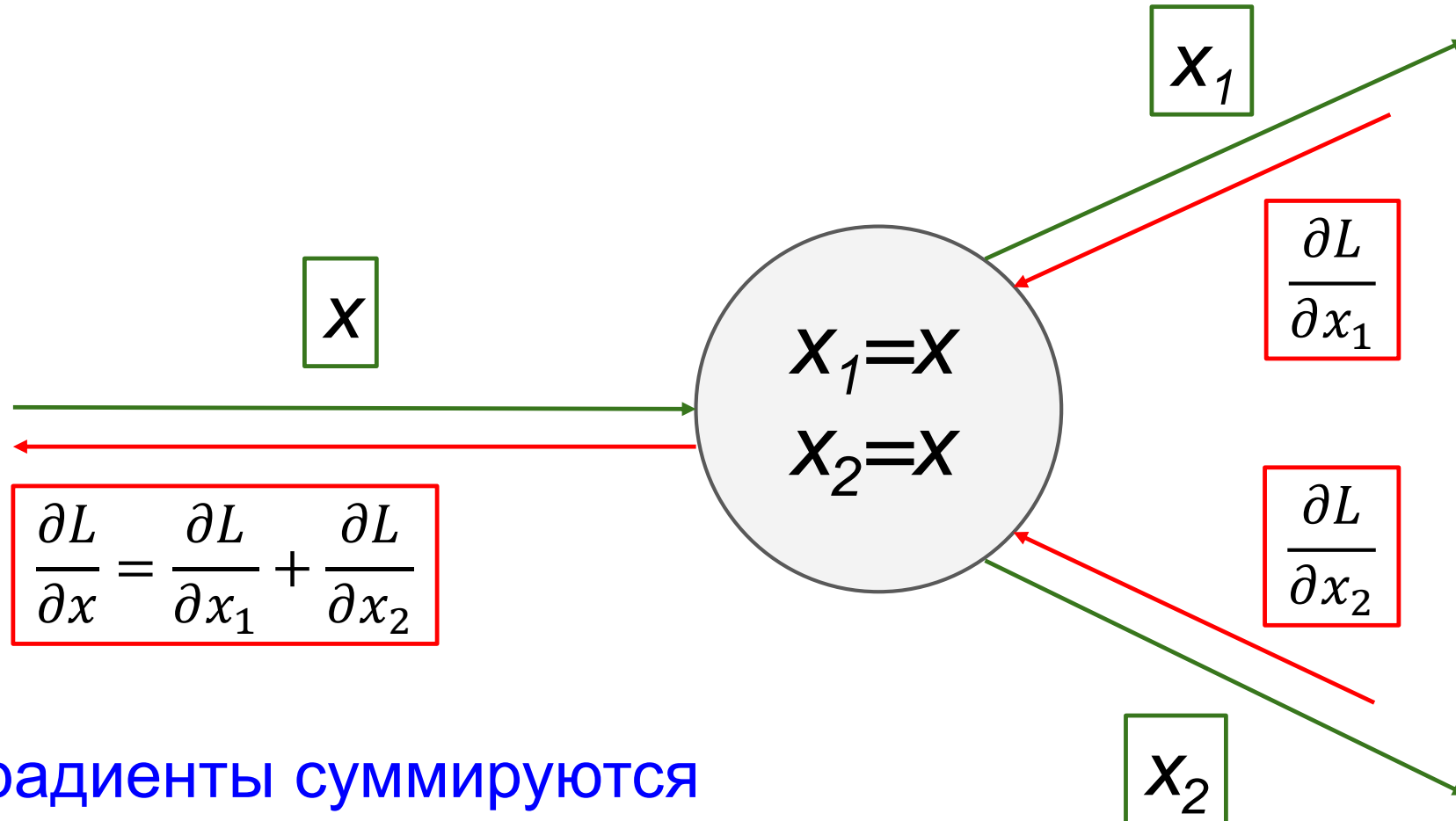
# Обратный градиент при суммировании



# Обратный градиент при переиспользовании переменной

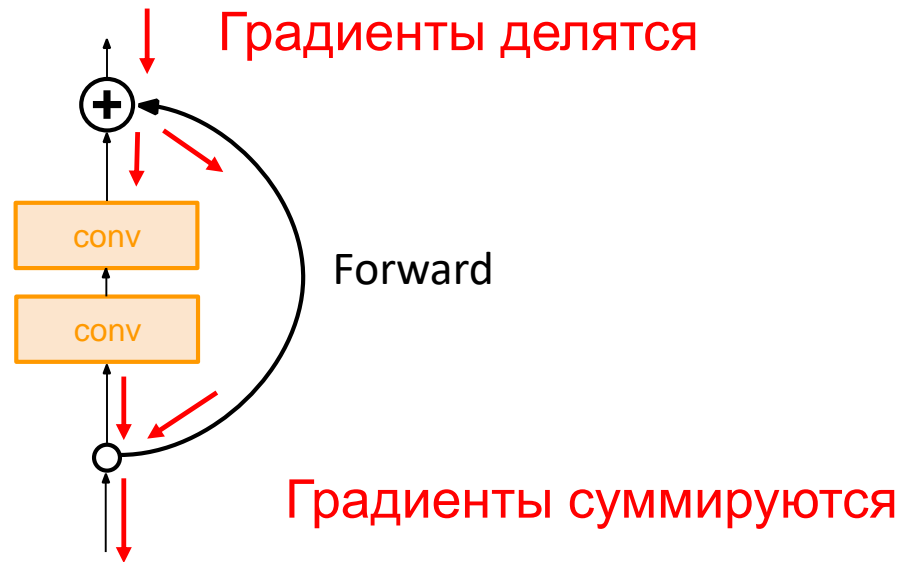


# Обратный градиент при переиспользовании переменной



# Почему ResNet проще обучать?

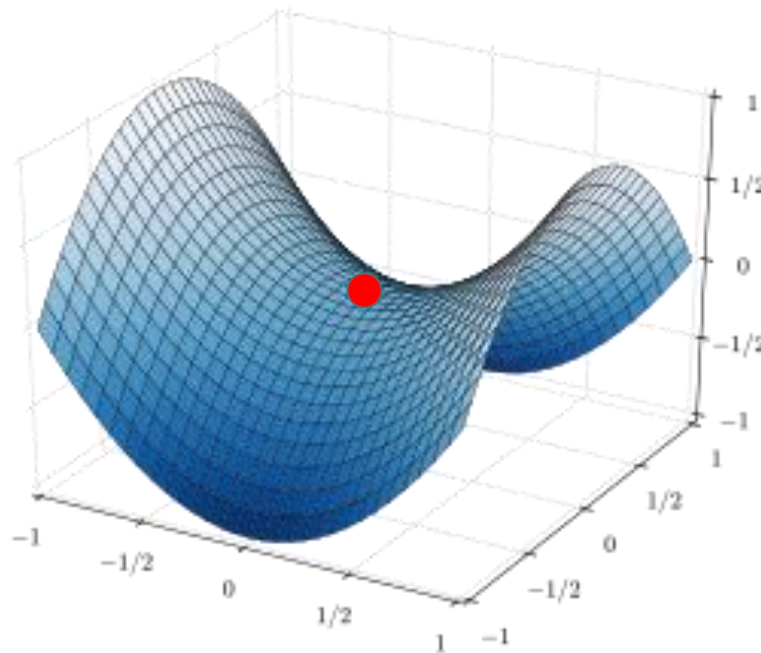
- Градиент ошибки легко распространяется назад



# Почему ResNet проще обучать?

- Градиент ошибки легко распространяется назад
- Сложения в residual block нарушают симметрию => меньше седловых точек => проще оптимизировать с помощью backpropagation

*(Orhan et al., Skip Connections Eliminate Singularities, 2017)*



# ResNet

(He et al., 2015)

## Особенности:

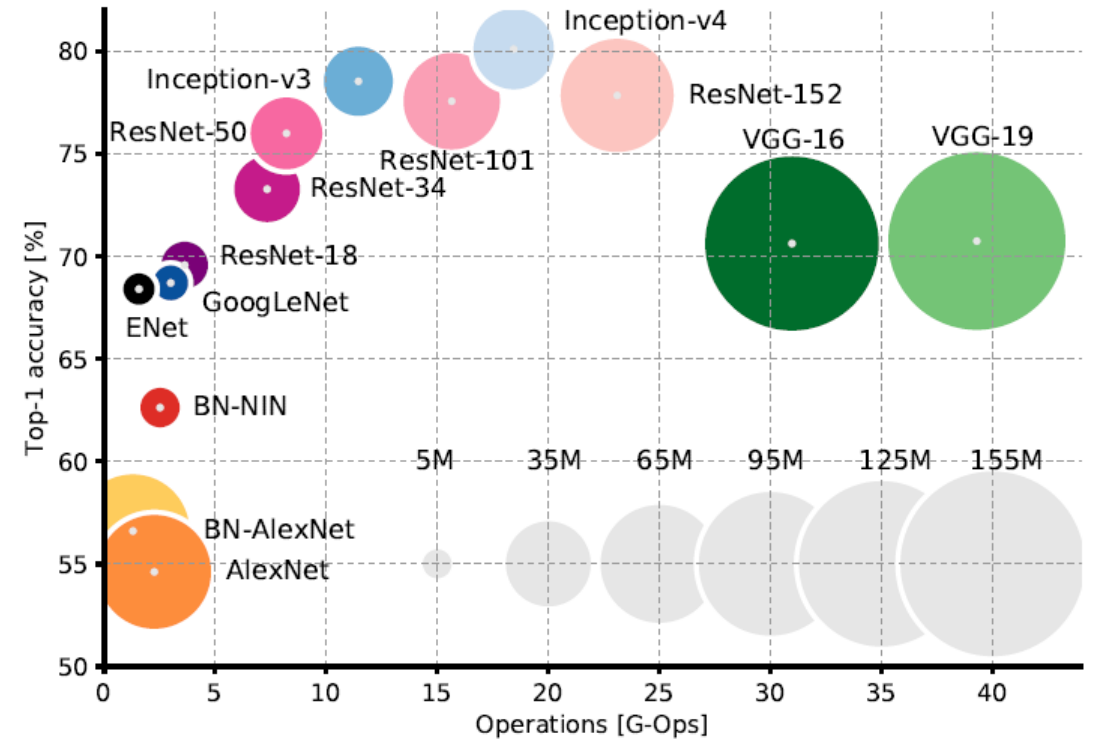
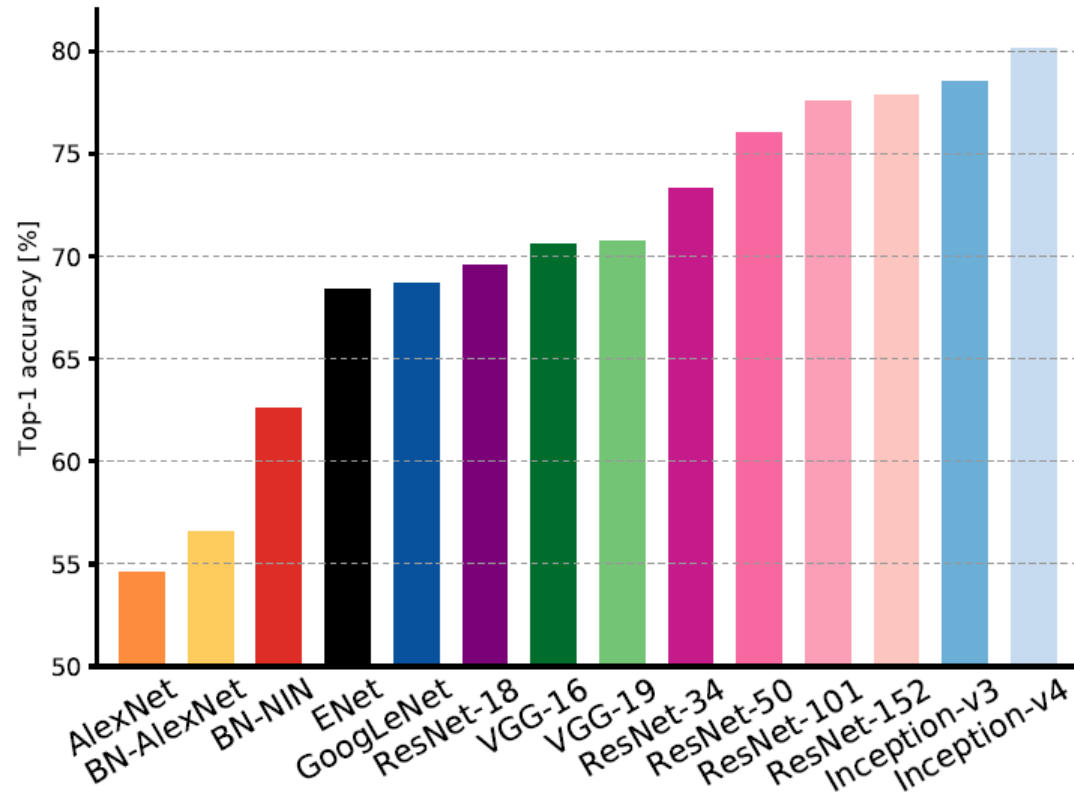
- 1-е место во всех основных конкурсах ImageNet'15 и COCO'15
- residual connections позволили тренировать глубокие сети без уменьшения точности (152 слоя ImageNet)
- Batch Norm после каждого CONV слоя

## MSRA @ ILSVRC & COCO 2015 Competitions

### • 1st places in all five main tracks

- ImageNet Classification: “Ultra-deep” (quote Yann) 152-layer nets
- ImageNet Detection: 16% better than 2nd
- ImageNet Localization: 27% better than 2nd
- COCO Detection: 11% better than 2nd
- COCO Segmentation: 12% better than 2nd

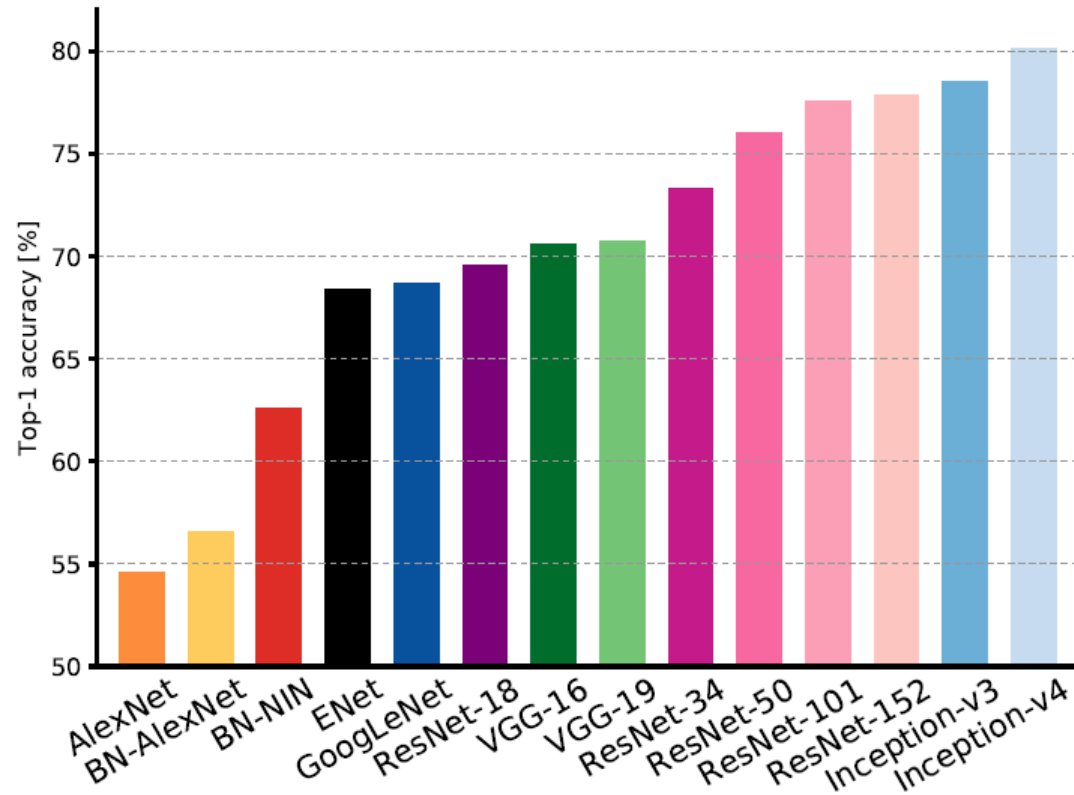
# Сравнение моделей



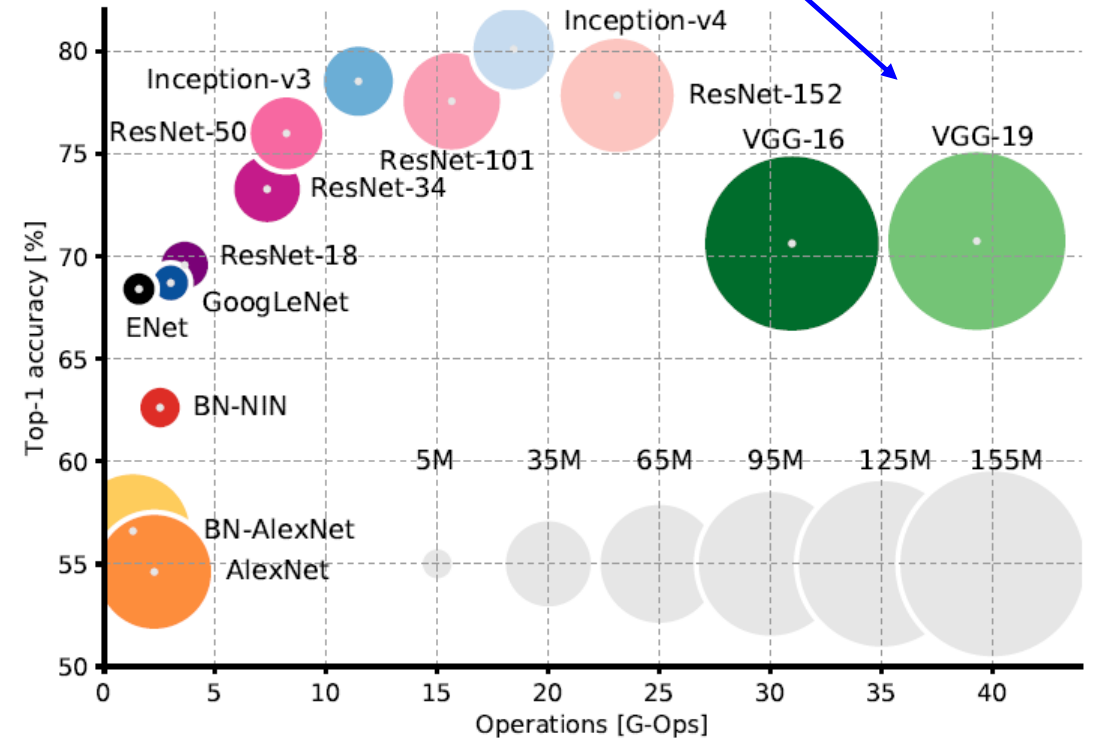
(A Canziani et al., An Analysis Of Deep Neural Network Models For Practical Applications, 2017)



# Сравнение моделей

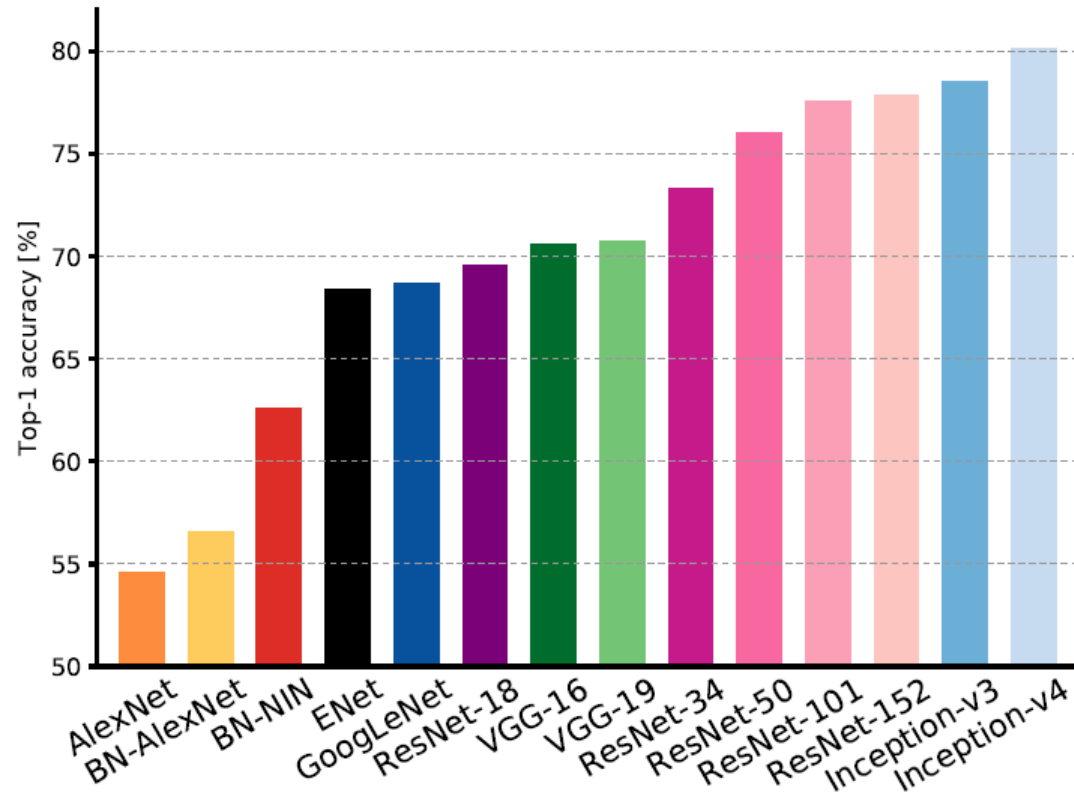


VGG: самая неэффективная  
много памяти, много вычислений

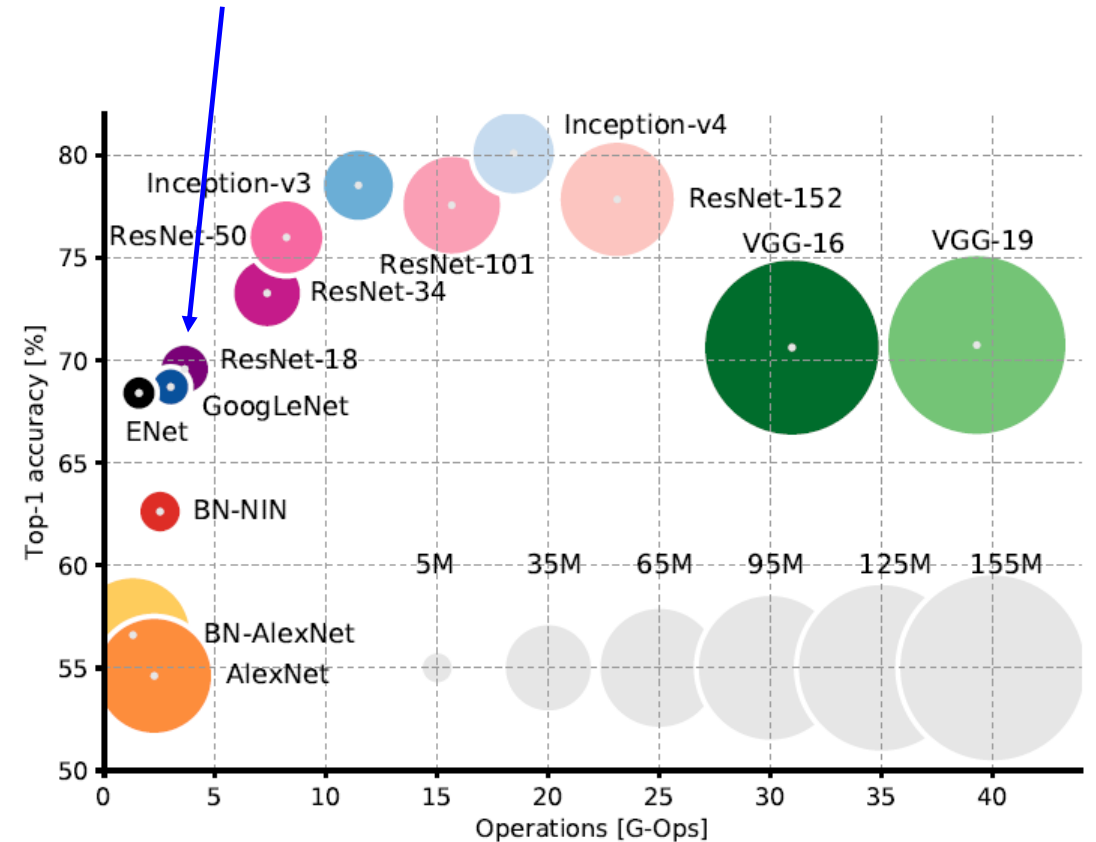


(A Canziani et al., An Analysis Of Deep Neural Network Models For Practical Applications, 2017)

# Сравнение моделей

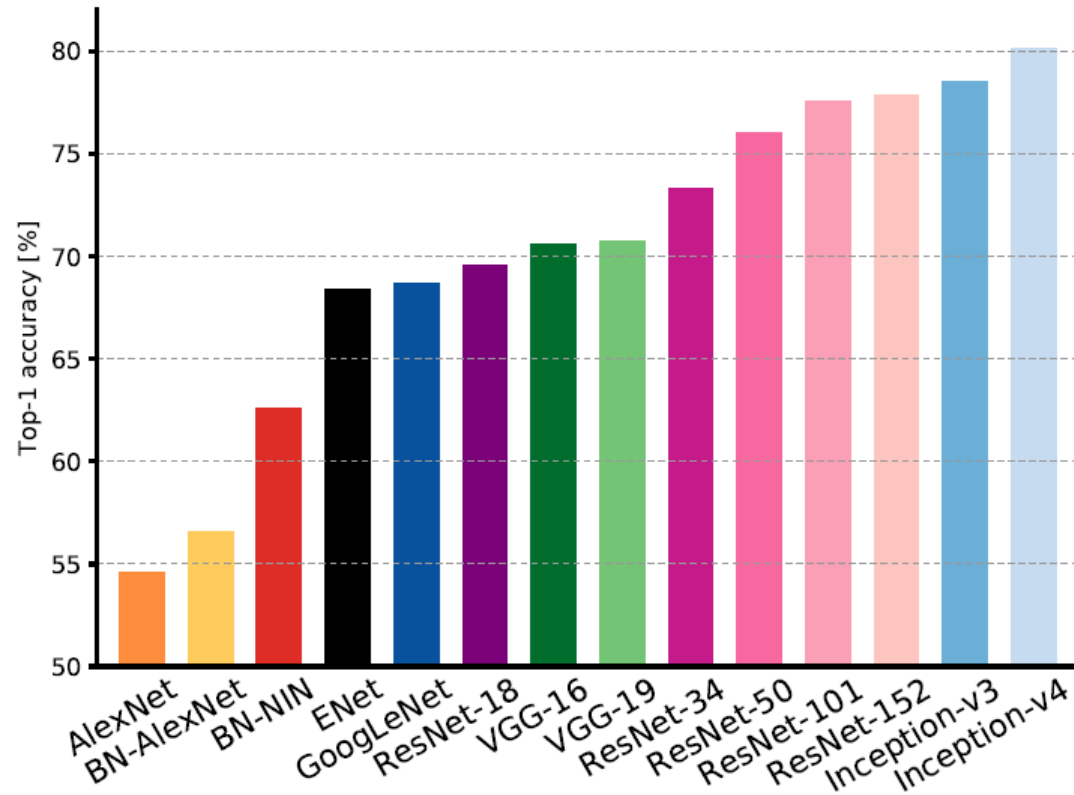


GoogLeNet и ResNet-18  
самые эффективные

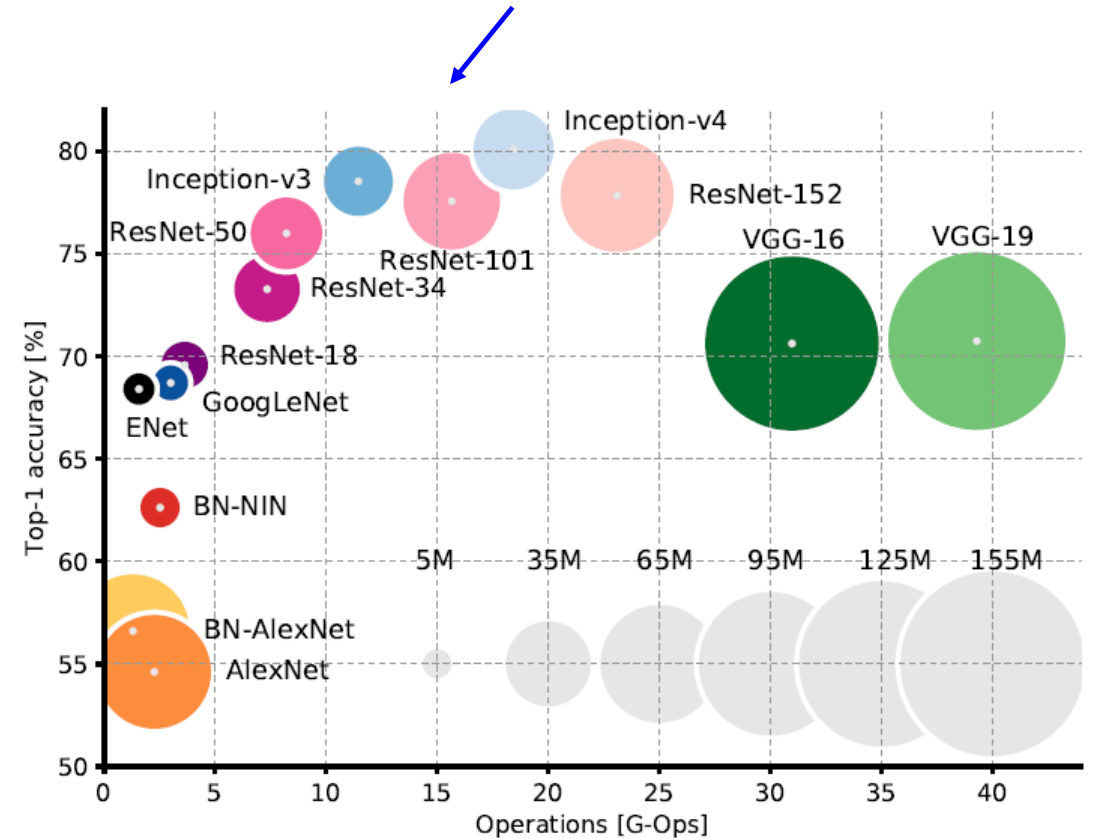


(A Canziani et al., An Analysis Of Deep Neural Network Models For Practical Applications, 2017)

# Сравнение моделей

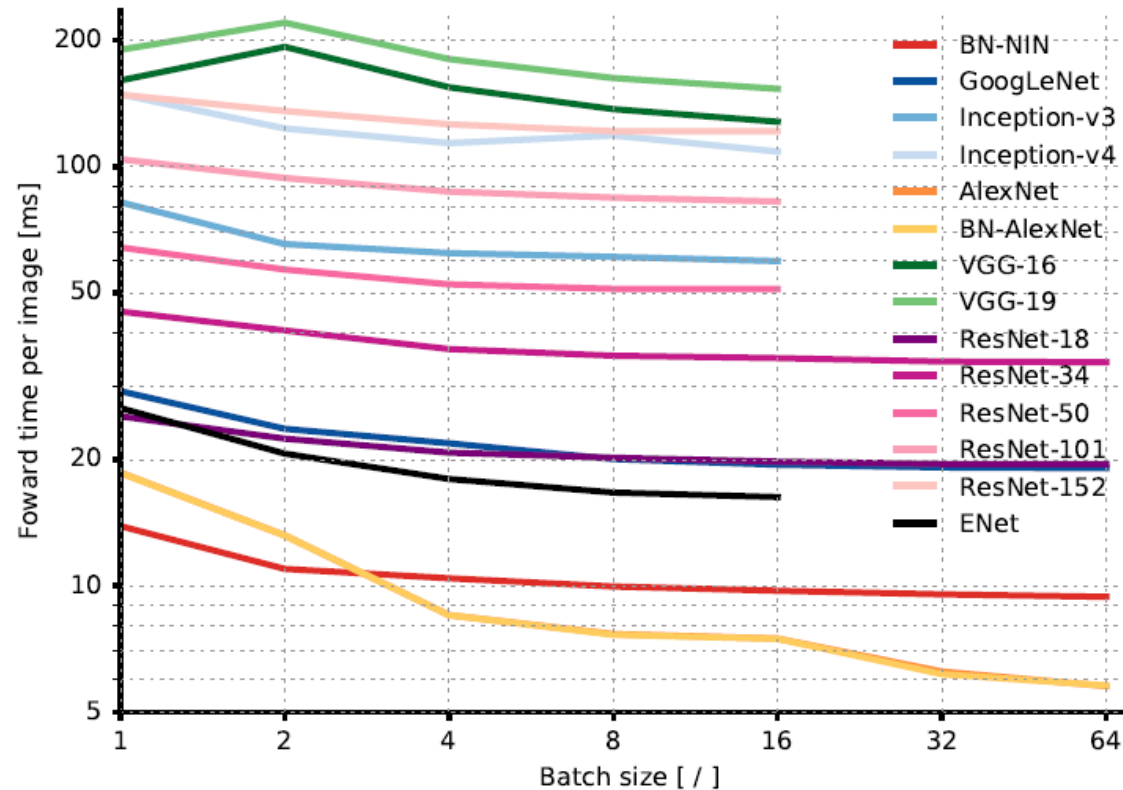


Inception v3/4 и ResNet-101/152:  
средняя эффективность, высокая  
точность, но много вычислений



(A Canziani et al., An Analysis Of Deep Neural Network Models For Practical Applications, 2017)

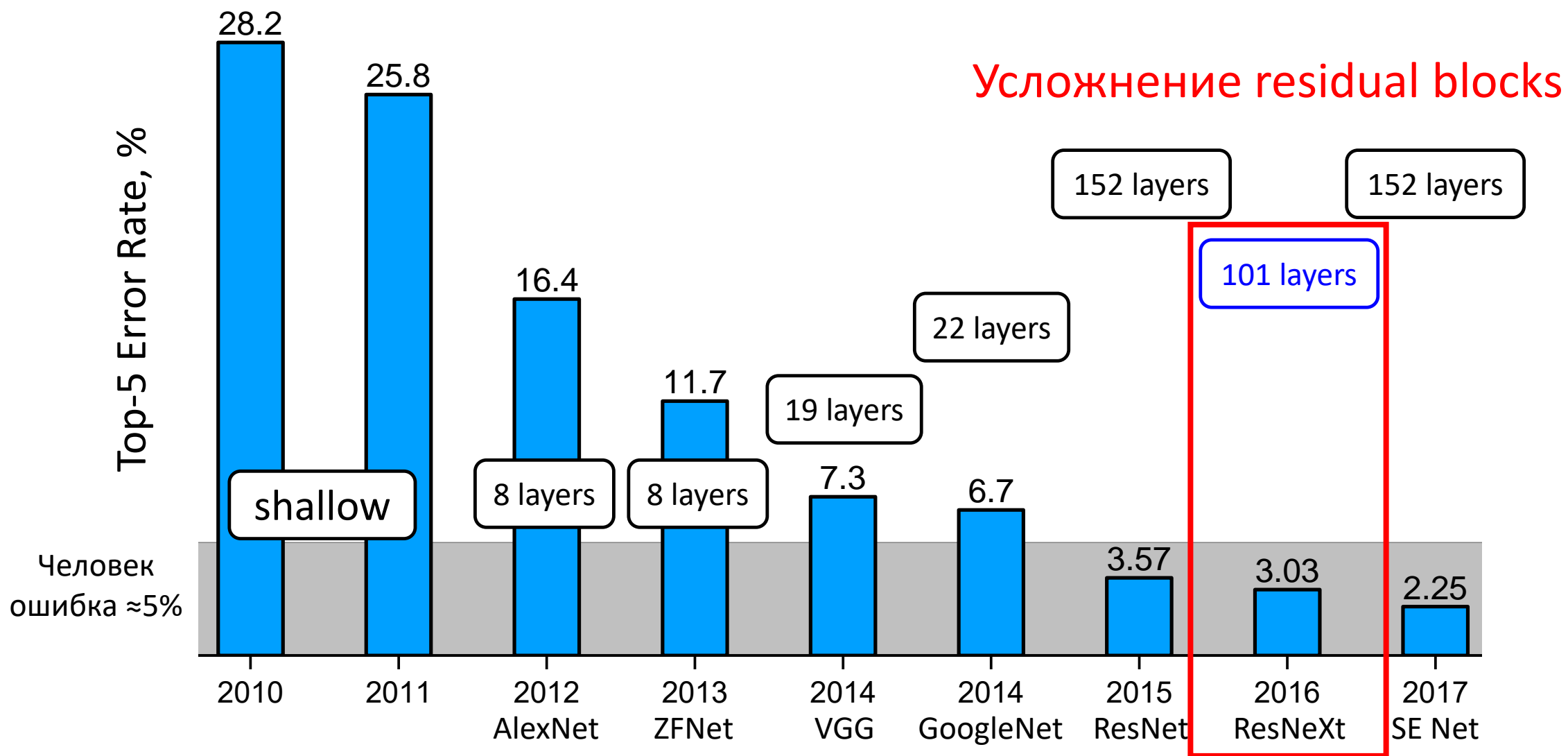
# Сравнение моделей



(A Canziani et al., An Analysis Of Deep Neural Network Models For Practical Applications, 2017)



# Large Scale Visual Recognition Challenge (ILSVRC)



# ResNeXt

(Xie et al., 2016)

ImageNet'16 2-е место в  
классификации  
(top 5 error 3.03%, у первого  
места 2.99% 😊)

Новые residual блоки похожи  
на Inception блоки, но с  
одинаковыми каналами

Представили архитектуры  
ResNeXt-50 и ResNeXt-101

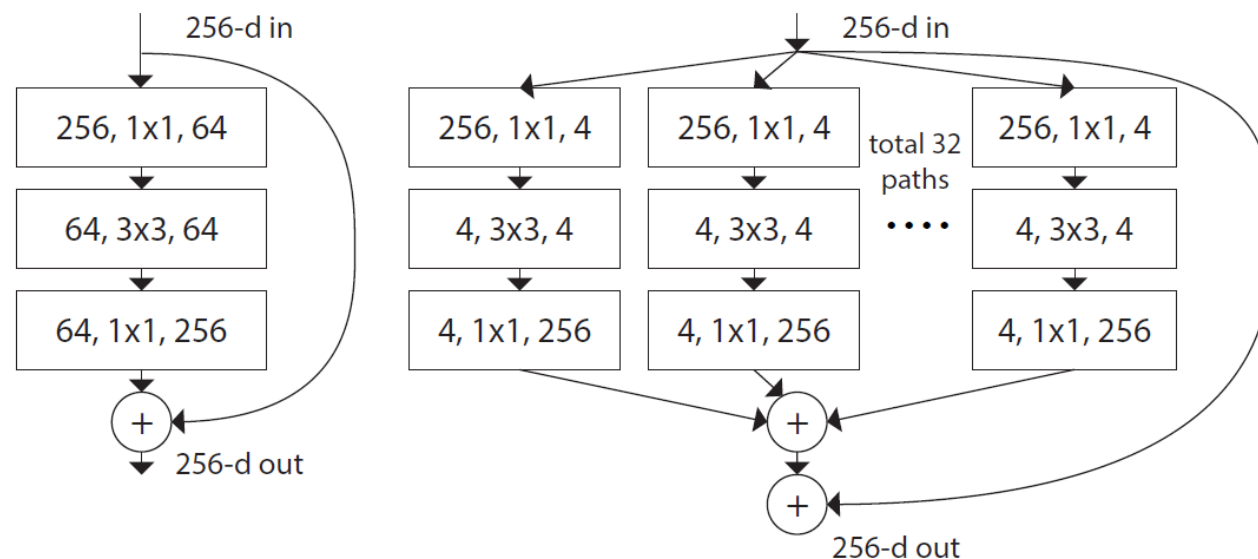


Figure 1. **Left:** A block of ResNet [14]. **Right:** A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

ImageNet top 5 error: 3.57% (ResNet) -> 3.03

# ResNeXt

(Xie et al., 2016)

## Переформулировки residual block

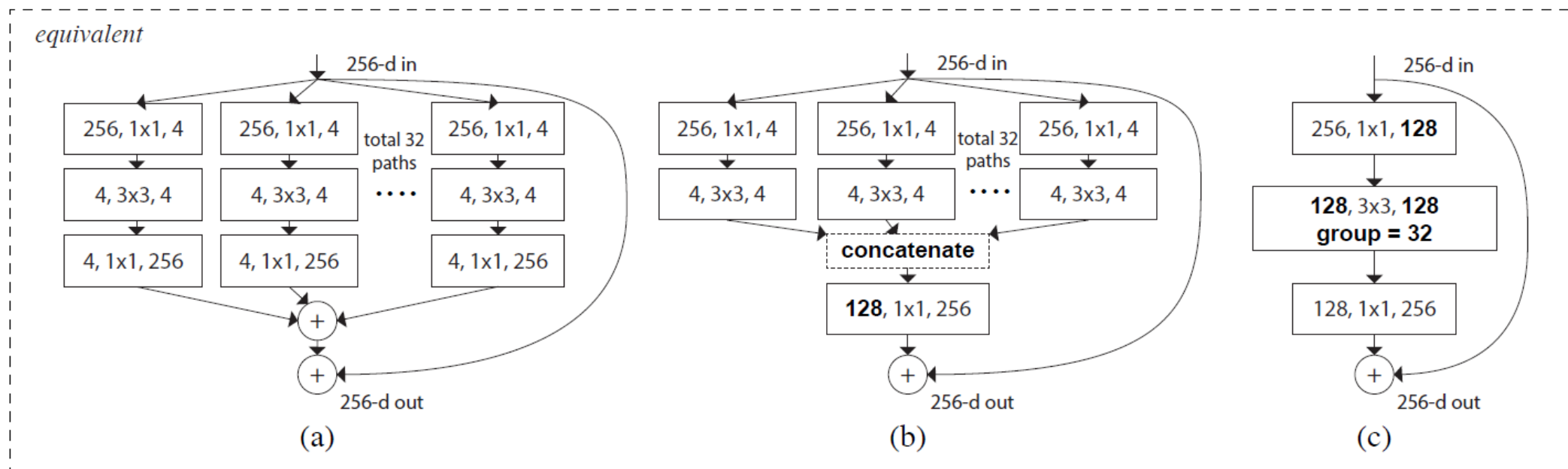


Figure 3. Equivalent building blocks of ResNeXt. **(a)**: Aggregated residual transformations, the same as Fig. 1 right. **(b)**: A block equivalent to (a), implemented as early concatenation. **(c)**: A block equivalent to (a,b), implemented as grouped convolutions [24]. Notations in **bold** text highlight the reformulation changes. A layer is denoted as (# input channels, filter size, # output channels).

# ResNeXt

(Xie et al., 2016)

## Пример архитектуры ResNeXt-50

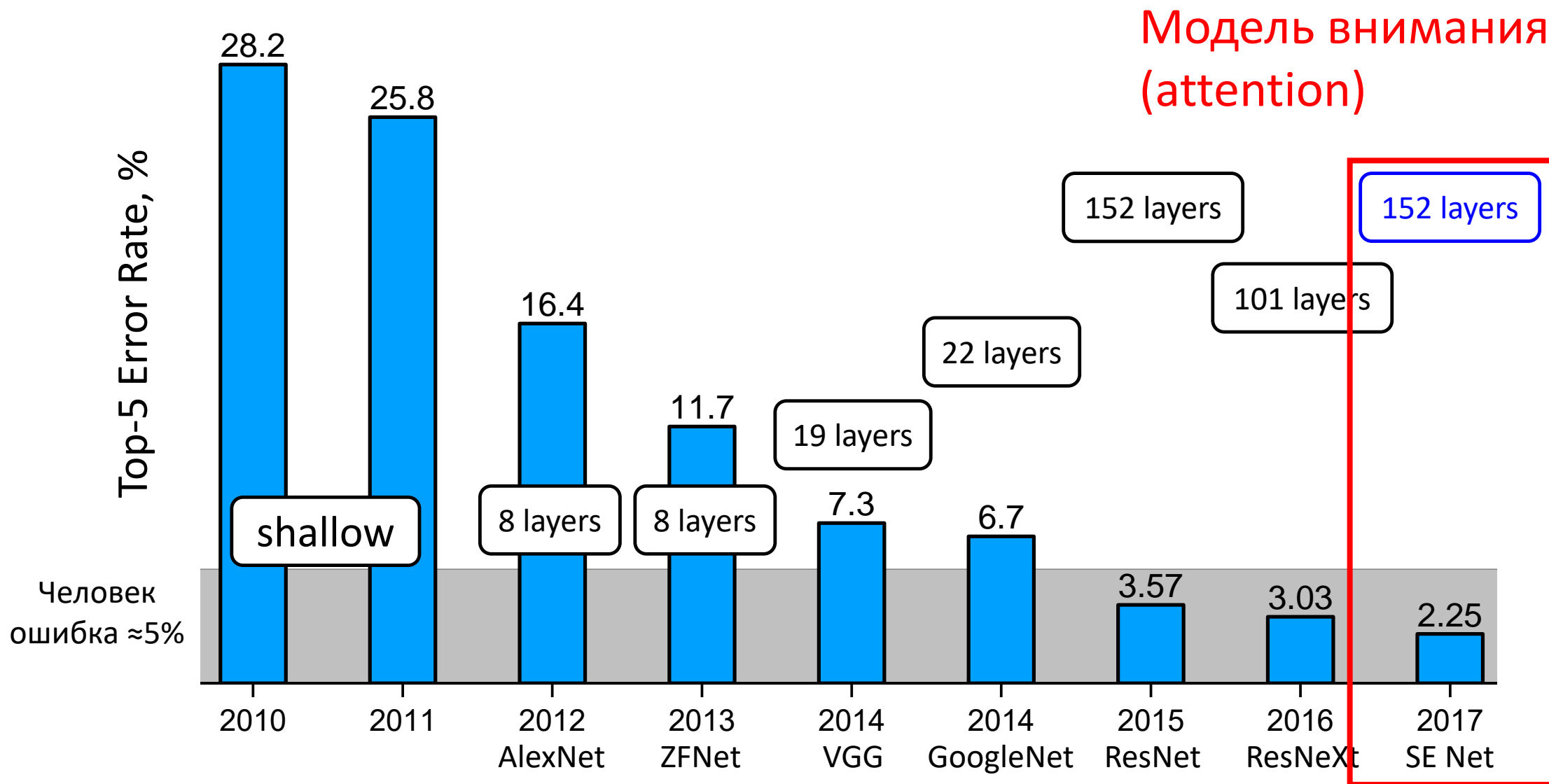
stage	output	ResNet-50	ResNeXt-50 ( $32\times 4d$ )
conv1	$112\times 112$	$7\times 7$ , 64, stride 2	$7\times 7$ , 64, stride 2
conv2	$56\times 56$	$3\times 3$ max pool, stride 2	$3\times 3$ max pool, stride 2
		$\begin{bmatrix} 1\times 1, 64 \\ 3\times 3, 64 \\ 1\times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 128 \\ 3\times 3, 128, C=32 \\ 1\times 1, 256 \end{bmatrix} \times 3$
conv3	$28\times 28$	$\begin{bmatrix} 1\times 1, 128 \\ 3\times 3, 128 \\ 1\times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times 1, 256 \\ 3\times 3, 256, C=32 \\ 1\times 1, 512 \end{bmatrix} \times 4$
conv4	$14\times 14$	$\begin{bmatrix} 1\times 1, 256 \\ 3\times 3, 256 \\ 1\times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times 1, 512 \\ 3\times 3, 512, C=32 \\ 1\times 1, 1024 \end{bmatrix} \times 6$
conv5	$7\times 7$	$\begin{bmatrix} 1\times 1, 512 \\ 3\times 3, 512 \\ 1\times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 1024 \\ 3\times 3, 1024, C=32 \\ 1\times 1, 2048 \end{bmatrix} \times 3$
	$1\times 1$	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		$25.5\times 10^6$	$25.0\times 10^6$
FLOPs		$4.1\times 10^9$	$4.2\times 10^9$

Table 1. **(Left)** ResNet-50. **(Right)** ResNeXt-50 with a  $32\times 4d$  template (using the reformulation in Fig. 3(c)). Inside the brackets are the shape of a residual block, and outside the brackets is the number of stacked blocks on a stage. “ $C=32$ ” suggests grouped convolutions [24] with 32 groups. *The numbers of parameters and FLOPs are similar between these two models.*





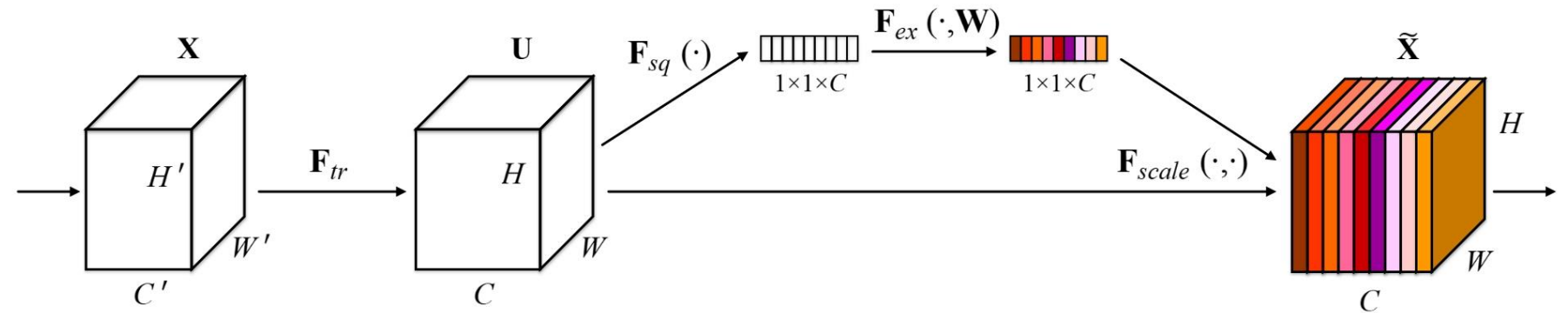
# Large Scale Visual Recognition Challenge (ILSVRC)



# Squeeze-and-Excitation Networks

(Hu et al., 2017)

ImageNet'17 1-е место в классификации



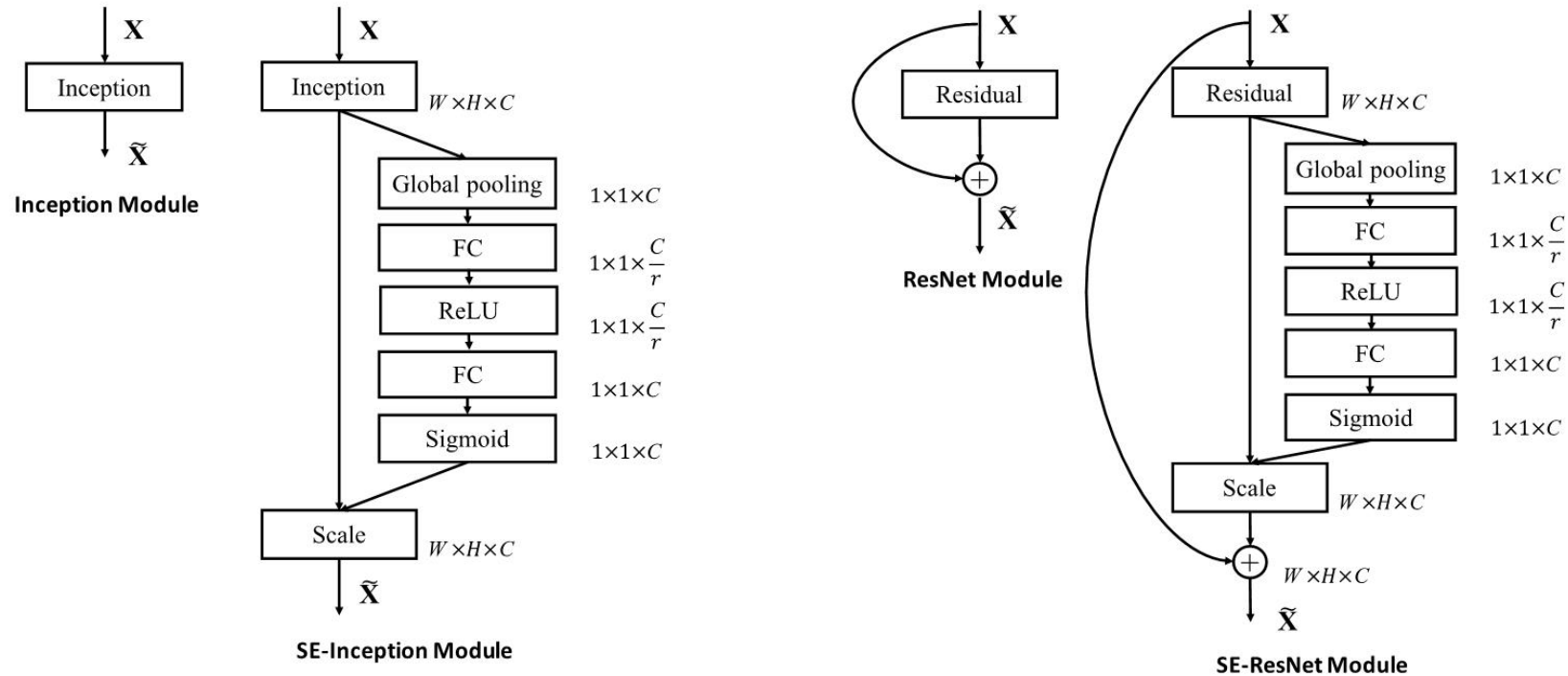
Модель внимания. Squeeze-and-Excitation блоки (SE blocks) нормируют каналы и помогают использовать глобальную информацию

Новые SE block очень быстрые и практически не замедляют сеть

ImageNet top 5 error: 3.03% (ResNeXt) -> **2.25**

# Squeeze-and-Excitation Networks

(Hu et al., 2017)



Schema of SE-Inception and SE-ResNet modules.  $r=16$  in all models.

# Squeeze-and-Excitation Networks

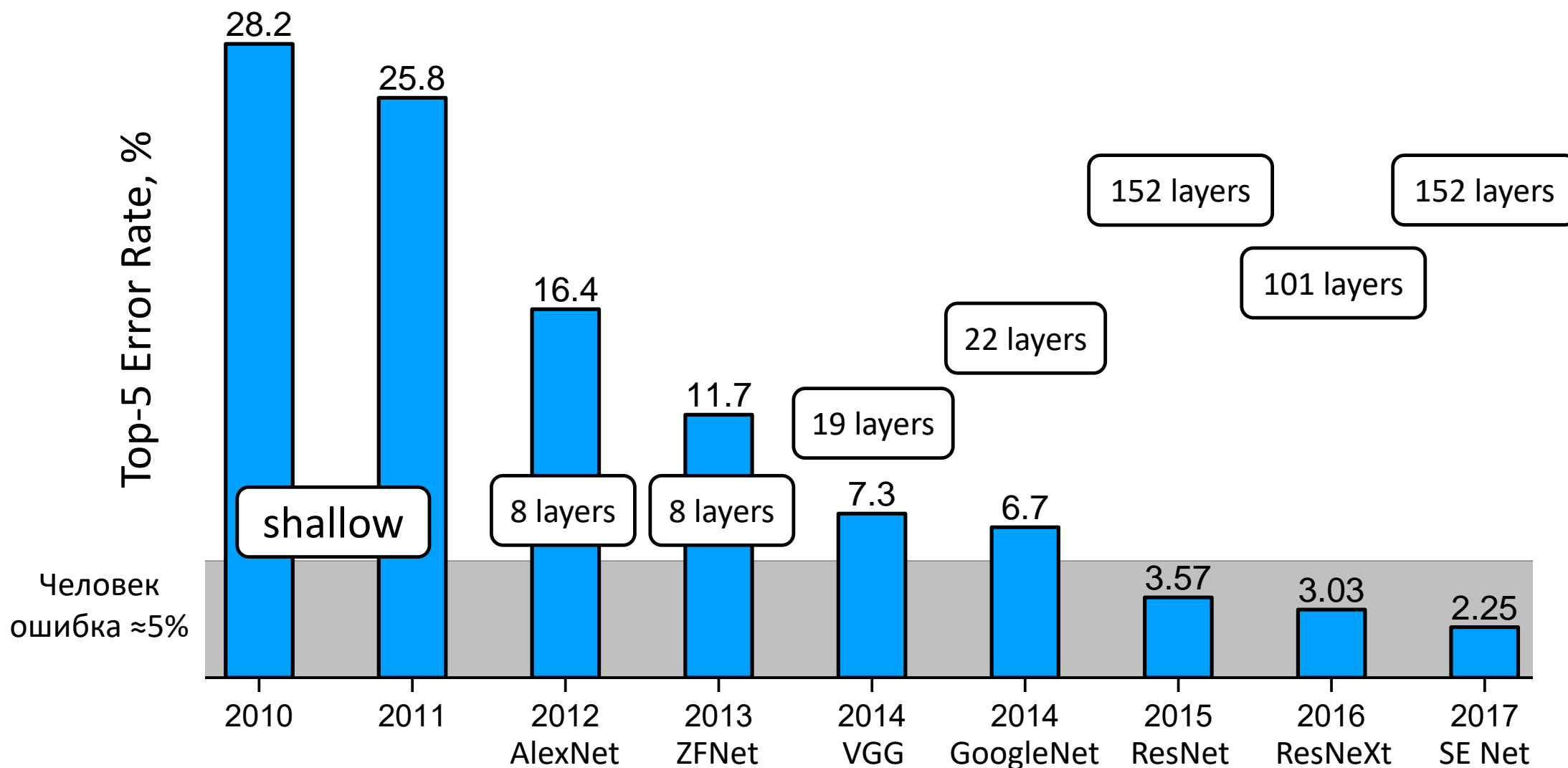
*(Hu et al., 2017)*

Особенности победителя ImageNet:

- ResNeXt-152 + SE blocks
  - 7x7 CONV => 3 3x3 CONV (like VGG)
  - Число каналов в 1x1 “bottleneck” уменьшено в 2 раза, для увеличения скорости вычислений
  - 1x1 CONV stride 2 => 3x3 CONV stride 2 для уменьшения пространственного разрешения
  - Тренировка: Label smoothing regularization  
с небольшой вероятностью реальные метки заменяются на распределение меток из train набора
- (Szegedy et al., Rethinking the inception architecture for computer vision, 2016)*



# Large Scale Visual Recognition Challenge (ILSVRC)

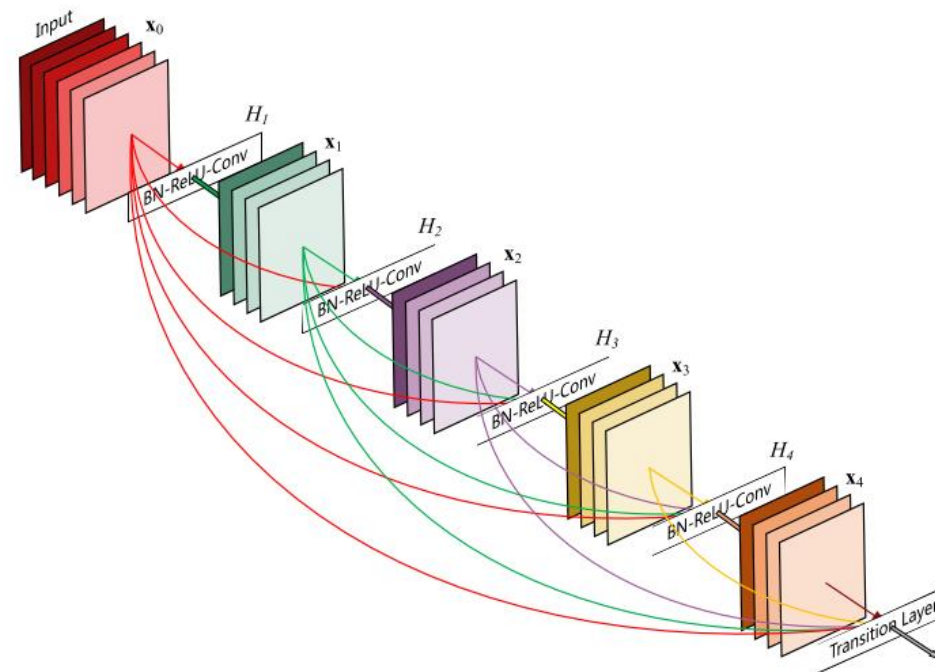


# DenseNet

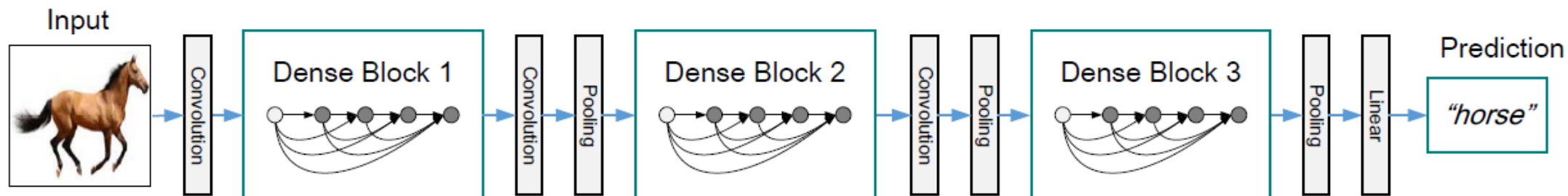
(Huang et al., 2017)

Улучшает распространение информации между слоями, и уменьшает число параметров

Dense blocks: каждый слой переиспользует признаки всех предыдущих



A 5-layer ( $l=5$ ) dense block with a growth rate of  $k = 4$ . Each layer takes all preceding feature-maps as input.



# DenseNet

(Huang et al., 2017)

Набор dense блоков,  
разделенных  
transition слоями

## Архитектуры для ImageNet

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	$112 \times 112$	$7 \times 7$ conv, stride 2			
Pooling	$56 \times 56$	$3 \times 3$ max pool, stride 2			
Dense Block (1)	$56 \times 56$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	$56 \times 56$	$1 \times 1$ conv			
	$28 \times 28$	$2 \times 2$ average pool, stride 2			
Dense Block (2)	$28 \times 28$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	$28 \times 28$	$1 \times 1$ conv			
	$14 \times 14$	$2 \times 2$ average pool, stride 2			
Dense Block (3)	$14 \times 14$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	$14 \times 14$	$1 \times 1$ conv			
	$7 \times 7$	$2 \times 2$ average pool, stride 2			
Dense Block (4)	$7 \times 7$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	$1 \times 1$	$7 \times 7$ global average pool			
		1000D fully-connected, softmax			

**Table 1:** DenseNet architectures for ImageNet. The growth rate for all the networks is  $k = 32$ . Note that each “conv” layer shown in the table corresponds the sequence BN-ReLU-Conv.

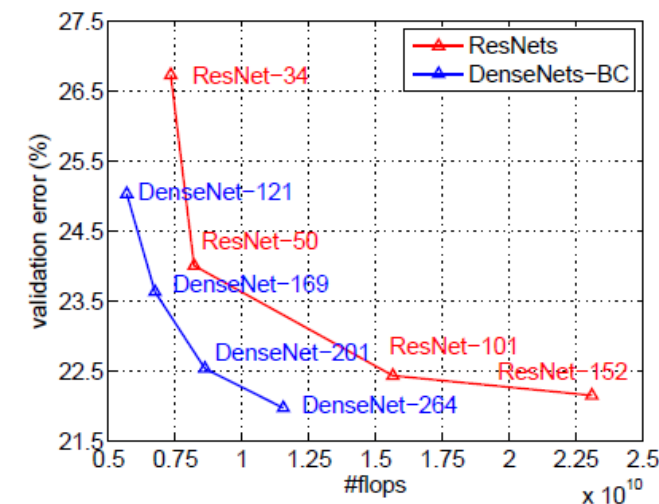
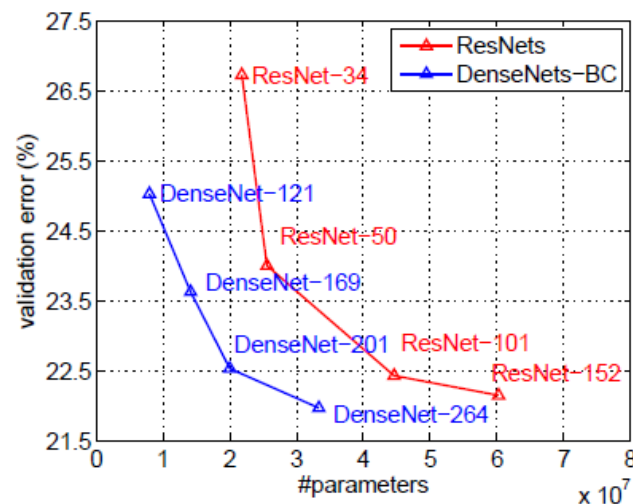
# DenseNet

(Huang et al., 2017)

## Результаты на ImageNet

Model	top-1	top-5
DenseNet-121	25.02 / 23.61	7.71 / 6.66
DenseNet-169	23.80 / 22.08	6.85 / 5.92
DenseNet-201	22.58 / 21.46	6.34 / 5.54
DenseNet-264	22.15 / 20.80	6.12 / 5.29

**Table 3:** The top-1 and top-5 error rates on the ImageNet validation set, with single-crop / 10-crop testing.



**Figure 3:** Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

DenseNet-BC = DenseNet + bottleneck + channel reduction  $\times 0.5$  in transition layers



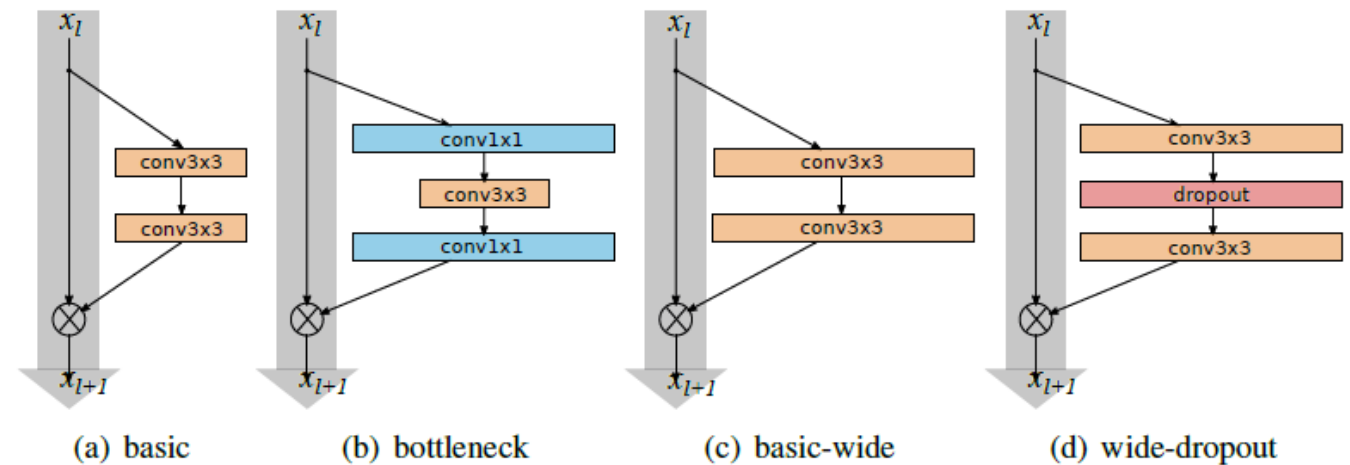
# Wide Residual Networks

(Zagoruyko et al., 2016)

Ширина сети также важна как глубина

50 слойная wide ResNet точнее, чем 152 слойная обычная ResNet

Увеличение ширины сети по сравнению с глубиной позволяет достичь большей скорости за счет распараллеливания матричных операций на GPU



## Архитектура для CIFAR-10

group name	output size	block type = $B(3, 3)$
conv1	$32 \times 32$	$[3 \times 3, 16]$
conv2	$32 \times 32$	$\begin{bmatrix} 3 \times 3, 16 \times k \\ 3 \times 3, 16 \times k \end{bmatrix} \times N$
conv3	$16 \times 16$	$\begin{bmatrix} 3 \times 3, 32 \times k \\ 3 \times 3, 32 \times k \end{bmatrix} \times N$
conv4	$8 \times 8$	$\begin{bmatrix} 3 \times 3, 64 \times k \\ 3 \times 3, 64 \times k \end{bmatrix} \times N$
avg-pool	$1 \times 1$	$[8 \times 8]$

Baseline ResNet  
 $k=1$

Wide ResNet  
 $k=2..10$   
 $N=3..10$

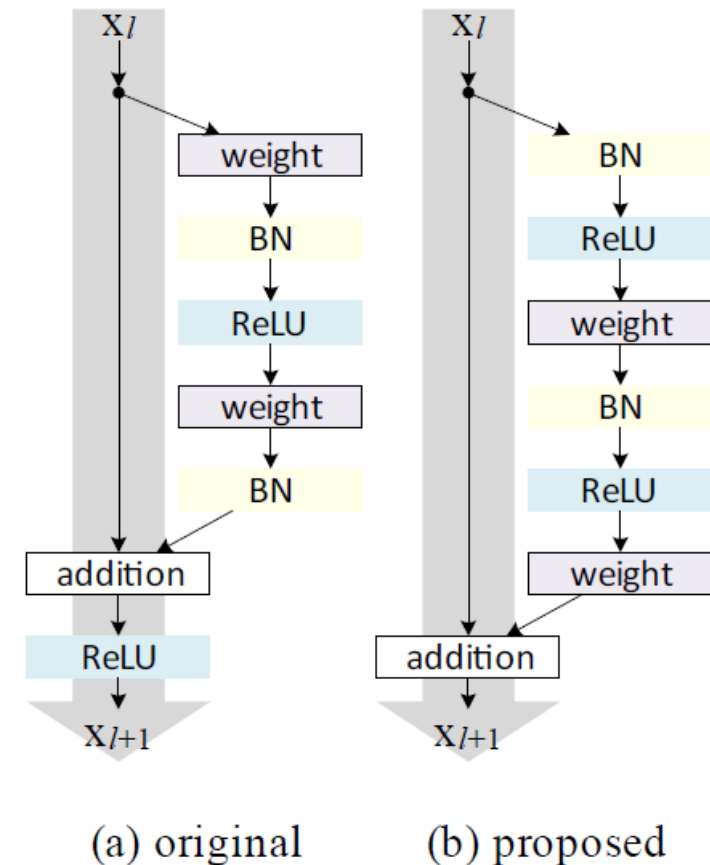
# Identity Mappings in Deep Residual Networks

(He et al., 2016)

Существенно улучшили residual block

Сделали прямой путь для распространения информации через всю сеть  
(активационные функции внесли в residual connection + переставили Batch Norm)

Стало возможным тренировать глубокие ResNet сети без потери точности (ранее на CIFAR-10 ResNet-1202 давала худшую точность, чем ResNet-110)

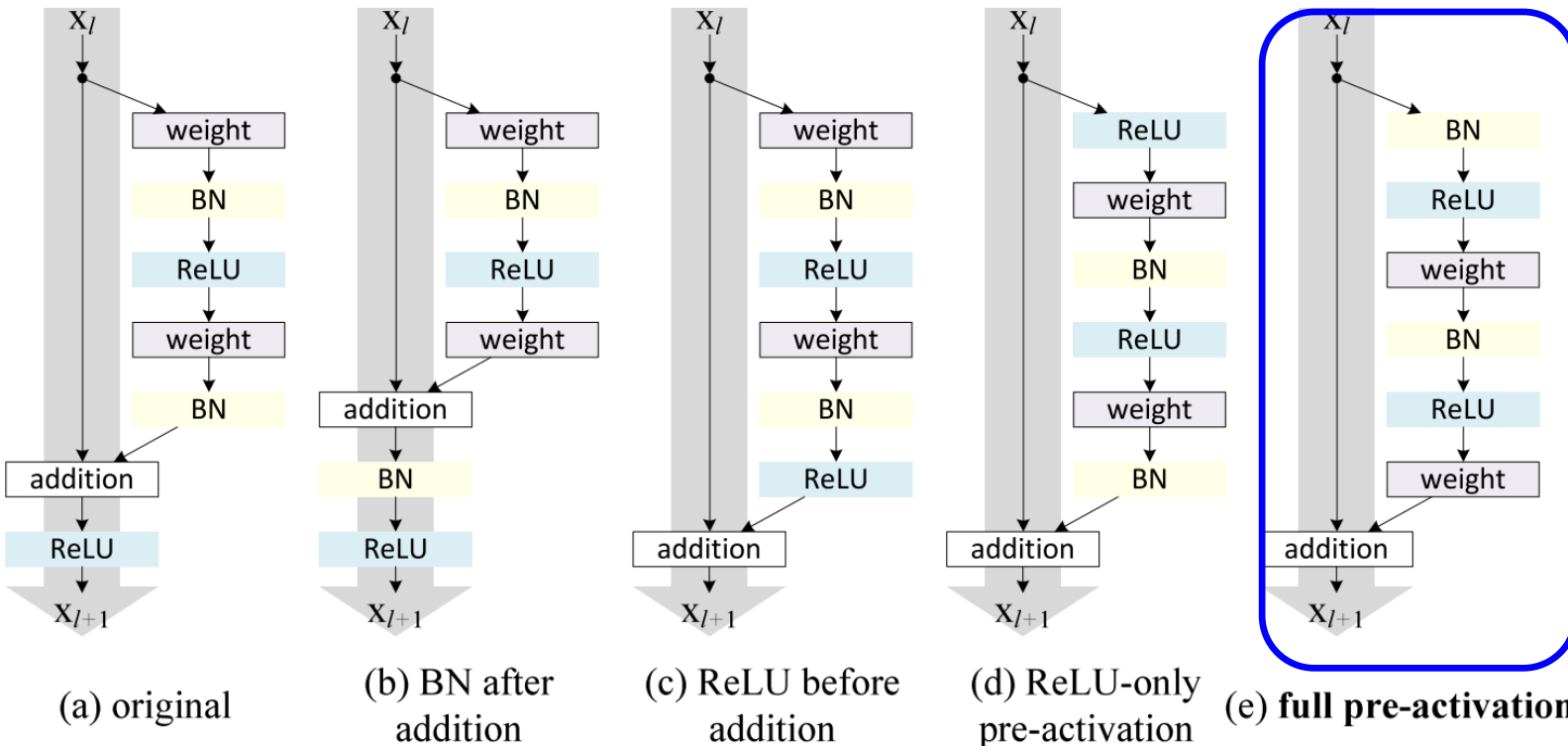


# Identity Mappings in Deep Residual Networks

(He et al., 2016)

Classification error (%) on the CIFAR-10

case	ResNet-110	ResNet-164
original ResNet unit (a)	6.61	5.93
BN after addition (b)	8.17	6.50
ReLU before addition (c)	7.84	6.14
ReLU-only pre-activation (d)	6.71	5.91
<b>full pre-activation (e)</b>	<b>6.37</b>	<b>5.46</b>



ReLU и BatchNorm в потоке информации.  
Плохо!

Положительная добавка.  
Плохо!

Добавка с единичным распределением.  
Плохо!

Добавка может иметь произвольное распределение.  
**Хорошо!**

# Identity Mappings in Deep Residual Networks

(He et al., 2016)

## ImageNet результаты

**Table 5.** Comparisons of single-crop error on the ILSVRC 2012 validation set. All ResNets are trained using the same hyper-parameters and implementations as [1]). Our Residual Units are the full pre-activation version (Fig. 4(e)). <sup>†</sup>: code/model available at <https://github.com/facebook/fb.resnet.torch/tree/master/pretrained>, using scale and aspect ratio augmentation in [20].

method	augmentation	train crop	test crop	top-1	top-5
ResNet-152, original Residual Unit [1]	scale	224×224	224×224	23.0	6.7
ResNet-152, original Residual Unit [1]	scale	224×224	320×320	21.3	5.5
ResNet-200, original Residual Unit [1]	scale	224×224	320×320	21.8	6.0

ResNet-200, pre-act Residual Unit	scale	224×224	320×320	20.7	5.3
-----------------------------------	-------	---------	---------	------	-----

ResNet-200 + original  
Residual Unit дает худшую  
точность, чем ResNet-152

ResNet-200 + pre-act  
Residual Unit точнее, чем  
ResNet-152

# Deep Networks with Stochastic Depth

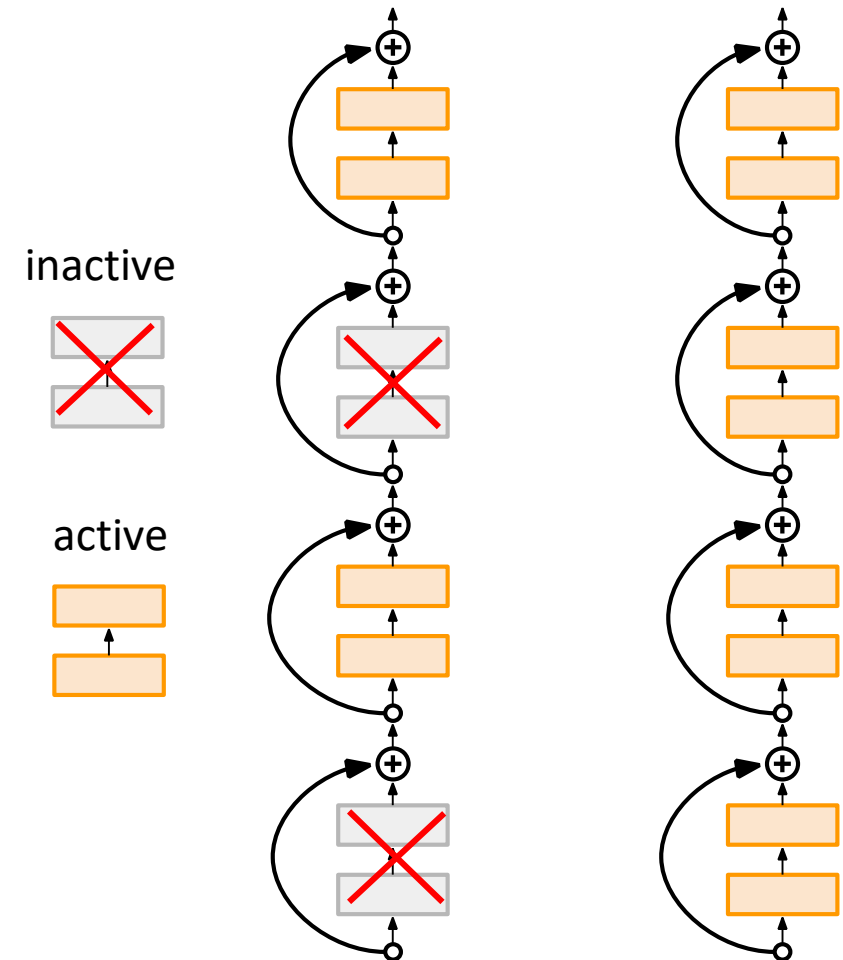
(Huang et al., 2016)

ResNet-1202 со случайной глубиной

Идея: уменьшаем время тренировки и  
улучшаем распространение градиентов  
за счет уменьшения глубины сети во  
время тренировки

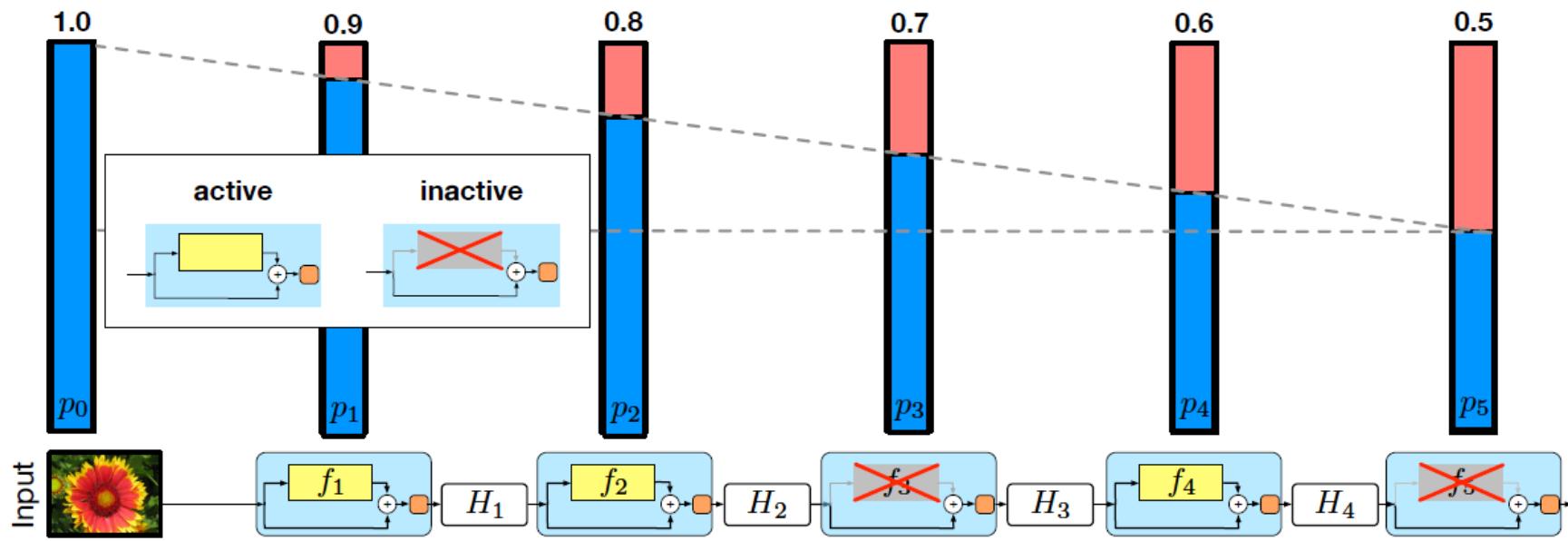
Случайно выкидываем residual  
connections во время тренировки  
(регуляризация  $\approx$  ансамбль моделей)

Во время теста запускаем всю сеть



# Deep Networks with Stochastic Depth

(Huang et al., 2016)



**Fig. 2.** The linear decay of  $p_l$  illustrated on a ResNet with stochastic depth for  $p_0 = 1$  and  $p_L = 0.5$ . Conceptually, we treat the input to the first ResBlock as  $H_0$ , which is always active.

Лучшие результаты с  $p_L=0.5$

# SqueezeNet

(Iandola et al., 2017)

Сфокусированы на уменьшении числа параметров

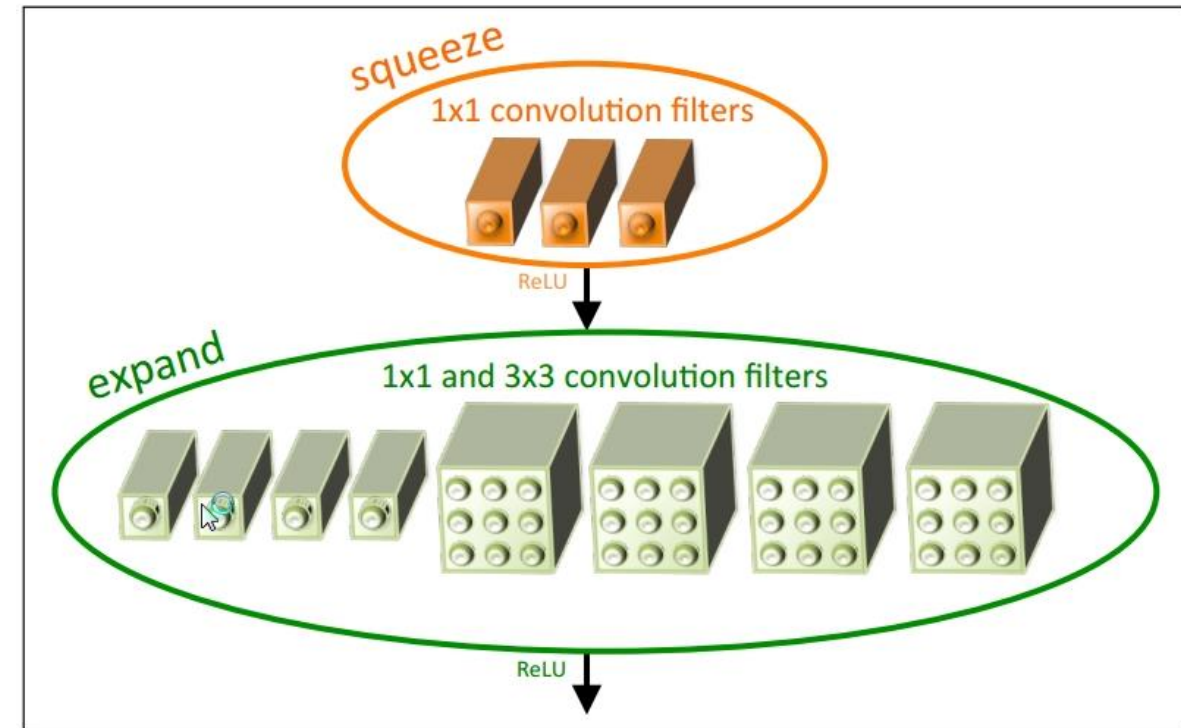
Fire block:

- squeeze layer 1x1 CONV
- expand layer 1x1 и 3x3 CONV

Точность как у AlexNet на ImageNet, но параметров в x50 меньше

Используя Deep Compression

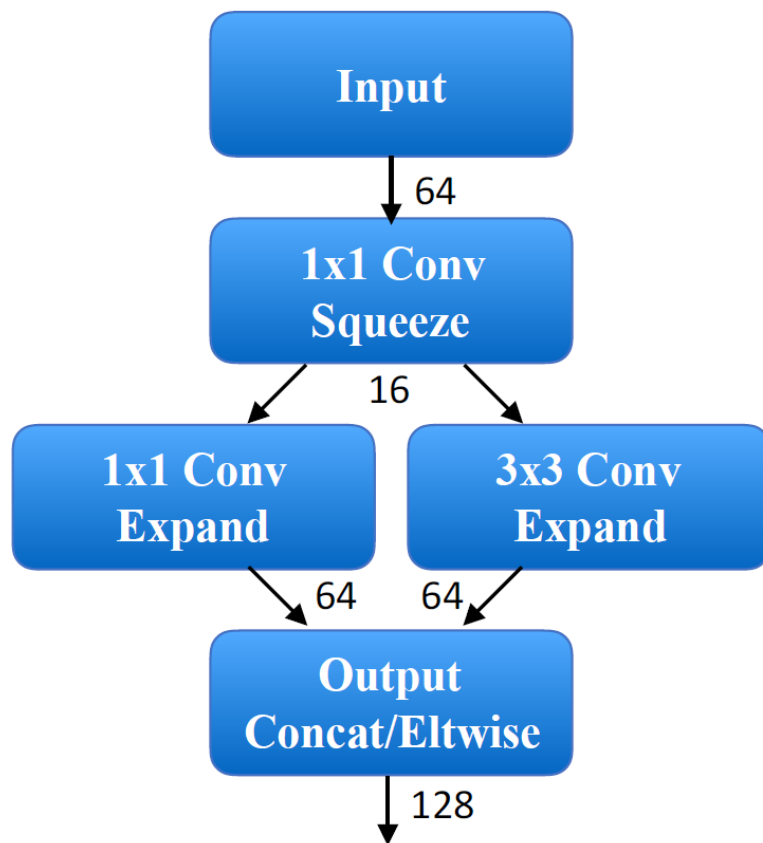
SqueezeNet можно сжать до размера в x510 меньше, чем AlexNet (0.5Mb)



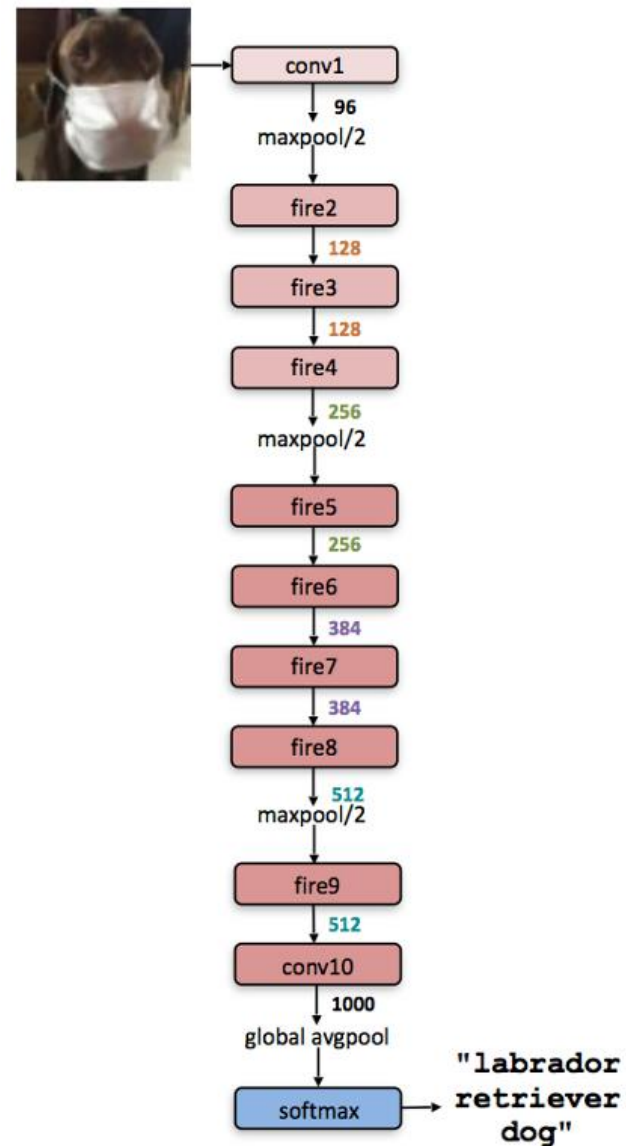
# SqueezeNet

(Iandola et al., 2017)

Fire block



## Vanilla SqueezeNet





# SqueezeNet

(Iandola et al., 2017)

Table 2: Comparing SqueezeNet to model compression approaches. By *model size*, we mean the number of bytes required to store all of the parameters in the trained model.

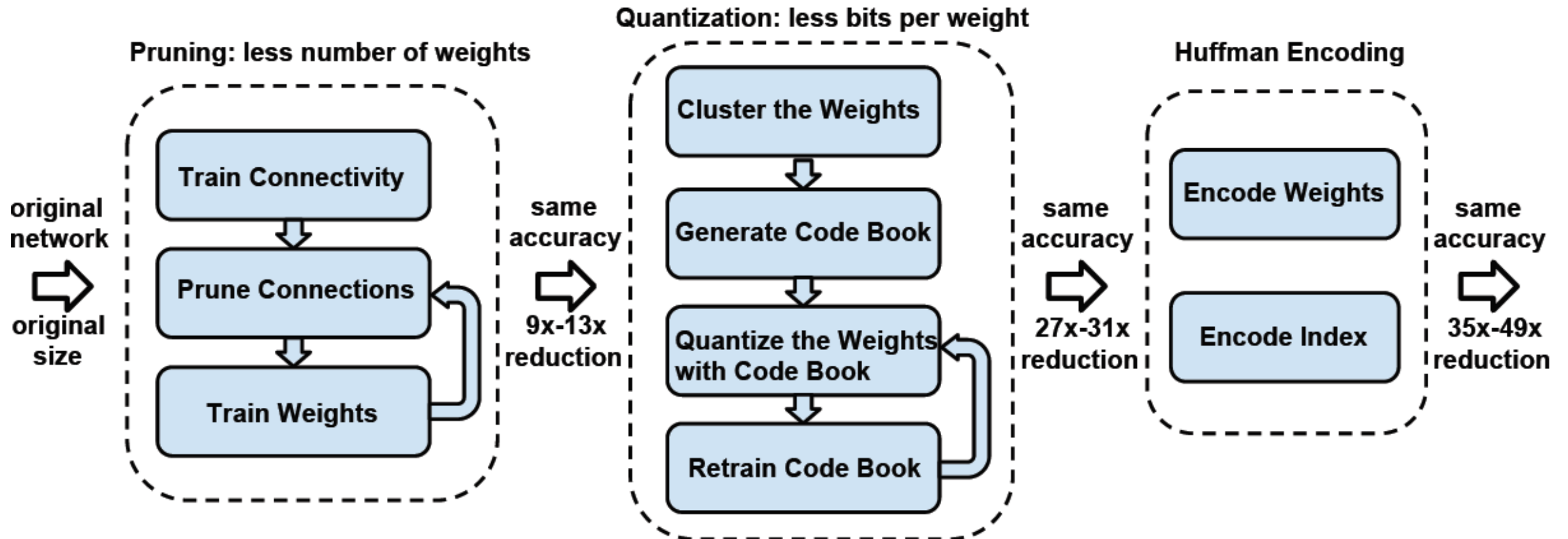
CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network Pruning (Han et al., 2015b)	32 bit	240MB → 27MB	9x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	None	32 bit	4.8MB	50x	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	363x	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	510x	57.5%	80.3%

x510 сжатие, по сравнению с AlexNet  
240MB -> 0.5MB!

# SqueezeNet

(Iandola et al., 2017)

## Summary of Deep Compression



# MobileNet v1

(Howard et al., 2017)

Сфокусированы на уменьшении сложности вычислений и числа параметров

Depthwise separable convolutions:

- 3x3 CONV (single filter per each input channel)
- 1x1 CONV (to create a linear combination)

Оптимизация скорости и точности за счет изменения разрешения входного изображения и числа каналов

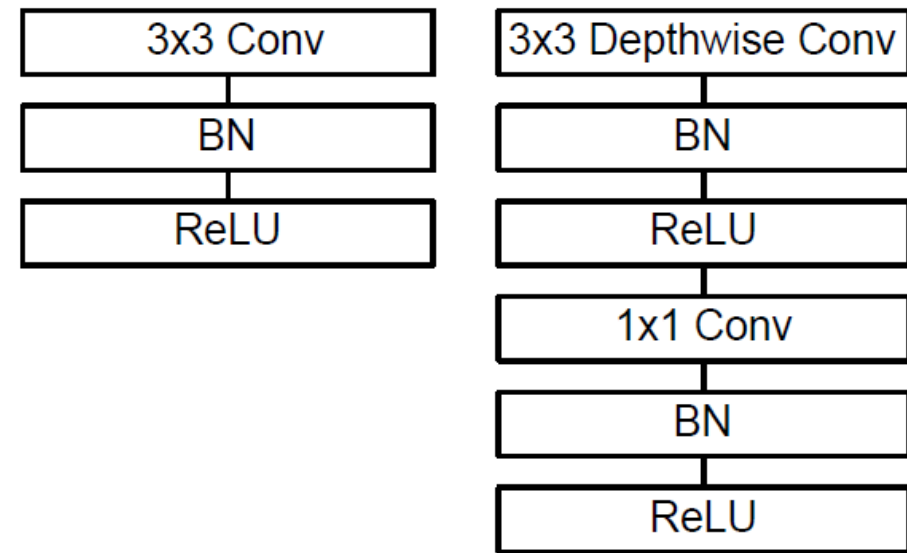
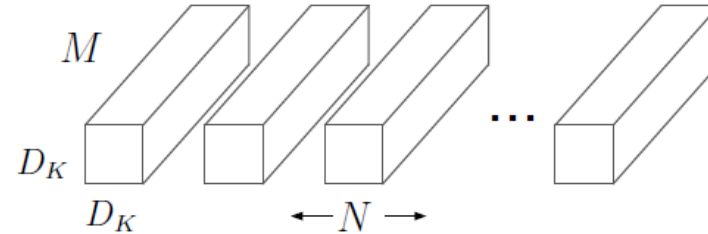


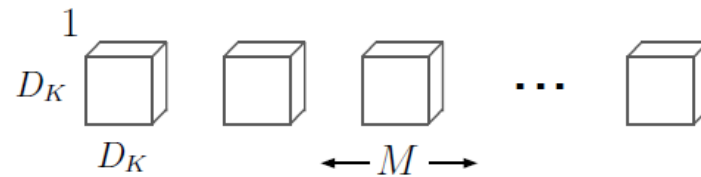
Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

# MobileNet v1

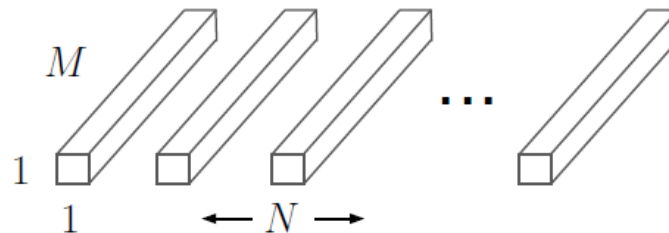
(Howard et al., 2017)



(a) Standard Convolution Filters



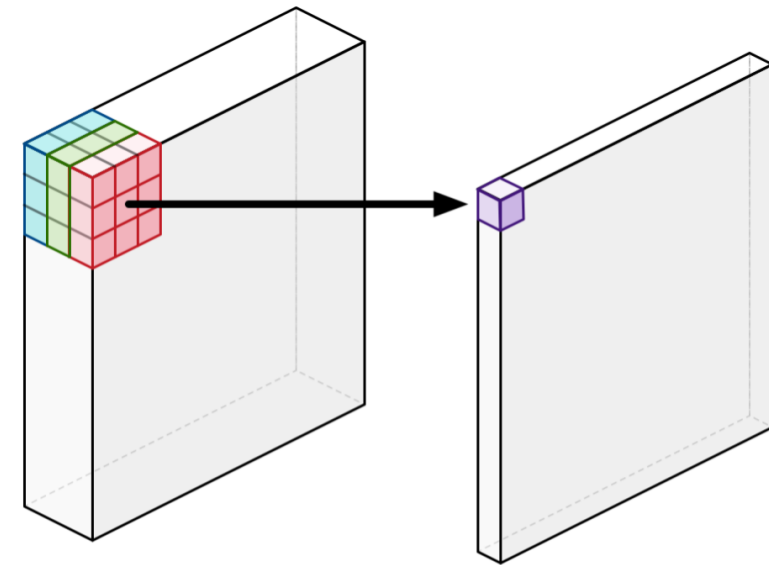
(b) Depthwise Convolutional Filters



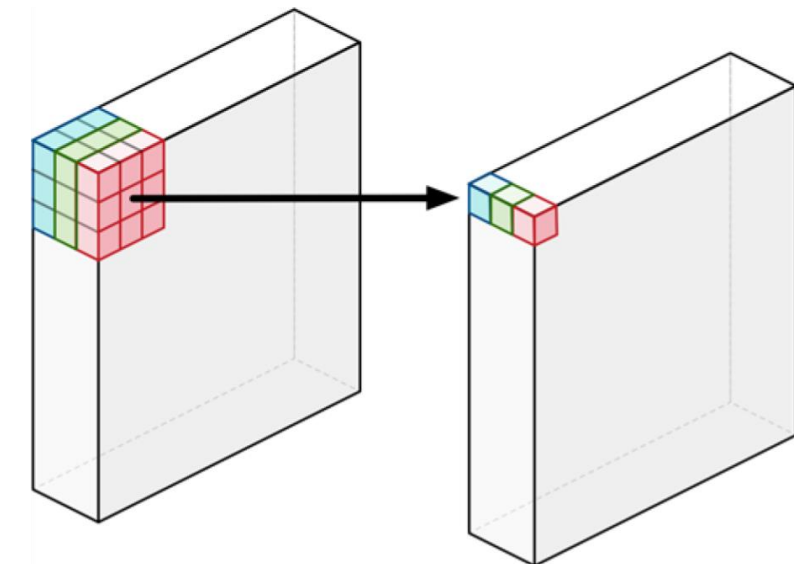
(c)  $1 \times 1$  Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Depthwise separable convolutions:

- 3x3 CONV (single filter per each input channel)
- 1x1 CONV (to create a linear combination)



convolution



depthwise convolution

# MobileNet v1

(Howard et al., 2017)

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×	Conv dw / s1 $3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
	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 $7 \times 7$	$7 \times 7 \times 1024$
FC / s1	$1024 \times 1000$	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Table 2. Resource Per Layer Type

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

Основное число операций  
и параметров в CONV 1x1!

# MobileNet v1

(Howard et al., 2017)

Depthwise CONV уменьшают  
число параметров и операций  
умножения-сложения

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

Оптимизация числа каналов

Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

Оптимизация разрешения  
входного изображения

# MobileNet v1

*(Howard et al., 2017)*

Особенности тренировки простых  
MobileNets:

- слабая регуляризация и аугментация данных
- иногда авторы отключали l2 decay для Depthwise CONV

# MobileNet v2

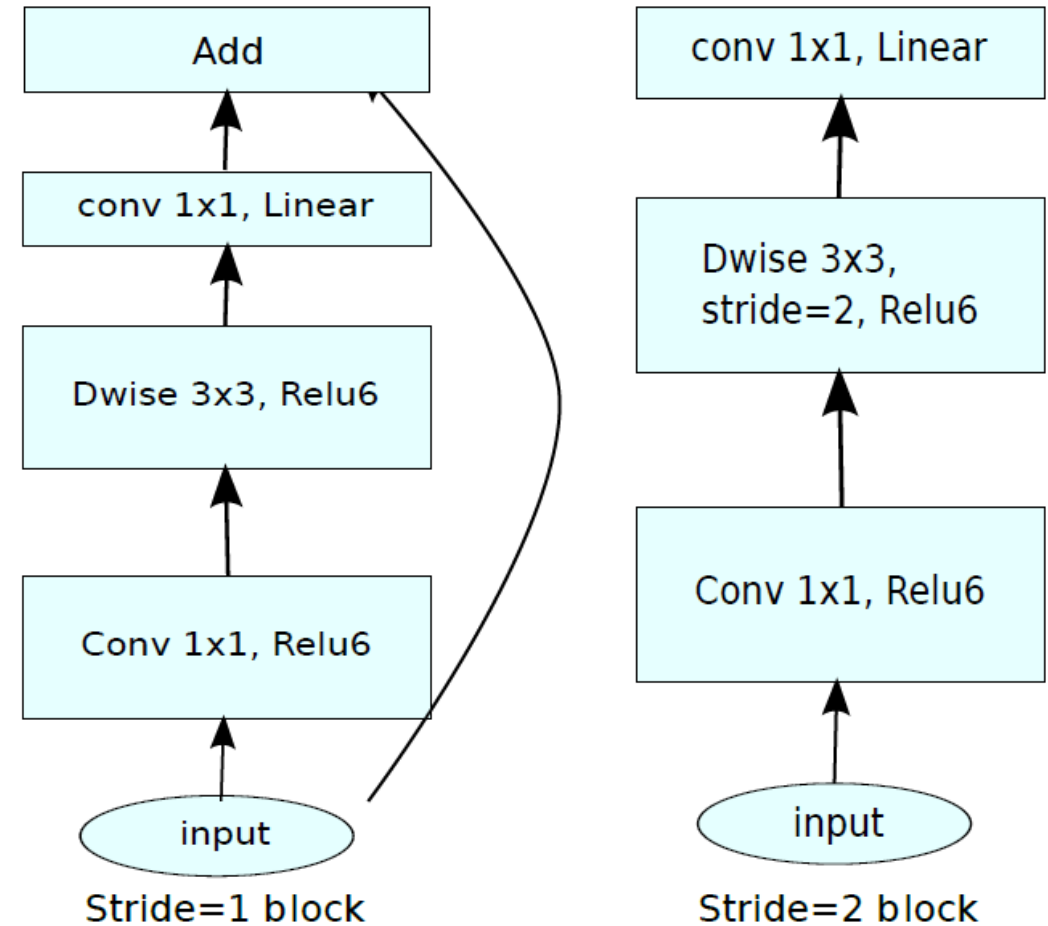
(Sandler et al., 2018)

Дальнейшее уменьшение числа операций и параметров

**Inverted residual unit:**

- 1x1 CONV ReLU with **expansion**
- 3x3 CONV Depthwise ReLU
- 1x1 CONV **linear (no activation func)**

Оптимизация скорости и точности за счет сохранения информации в inverted residual unit





# MobileNet v2

(Sandler et al., 2018)

## ImageNet результаты

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	<b>3.4M</b>	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	<b>72.0</b>	<b>3.4M</b>	<b>300M</b>	<b>75ms</b>
MobileNetV2 (1.4)	<b>74.7</b>	6.9M	585M	<b>143ms</b>

**v2 > v1**

## Архитектура

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	-	-

$t$  - expansion factor

$c$  - number of output channels

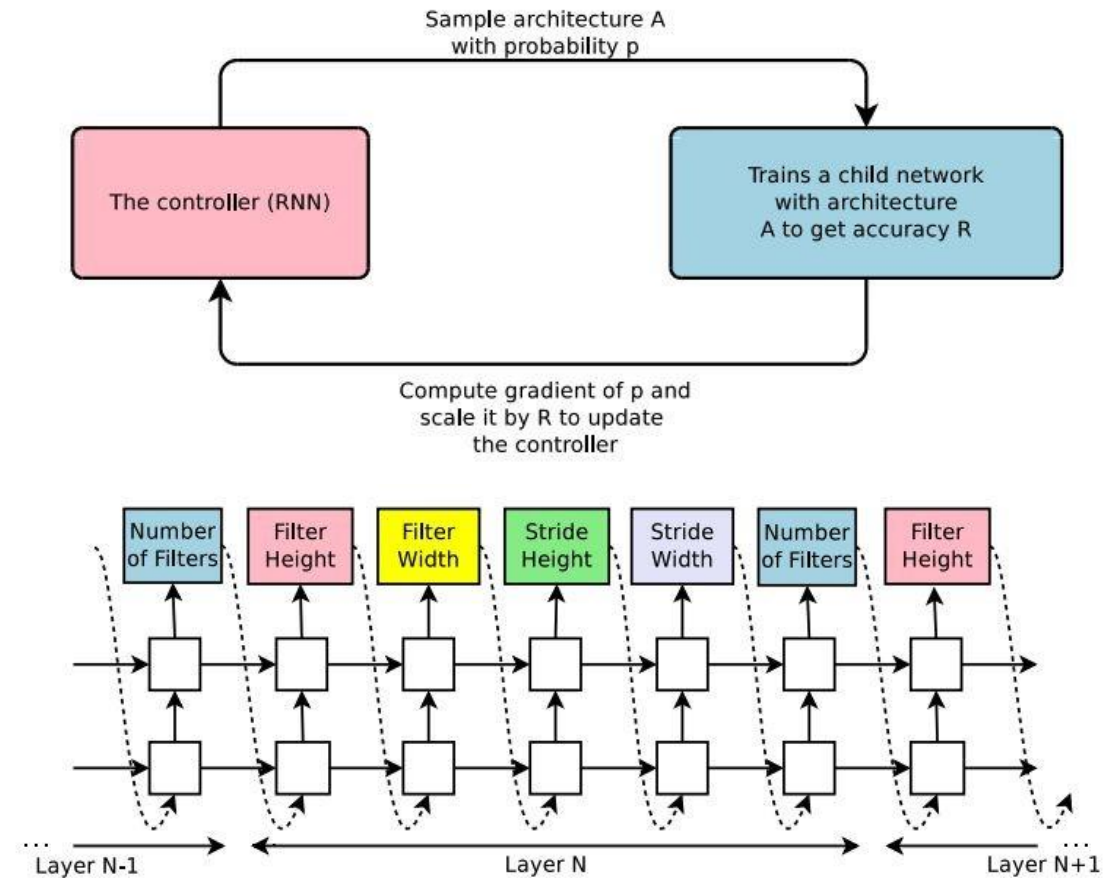
$n$  - number of repetitions

$s$  - stride

# Neural Architecture Search:

## Neural Architecture Search with Reinforcement Learning (NAS) [Zoph et al. 2016]

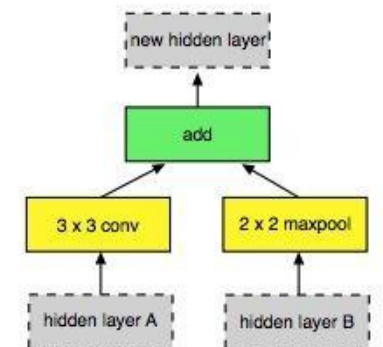
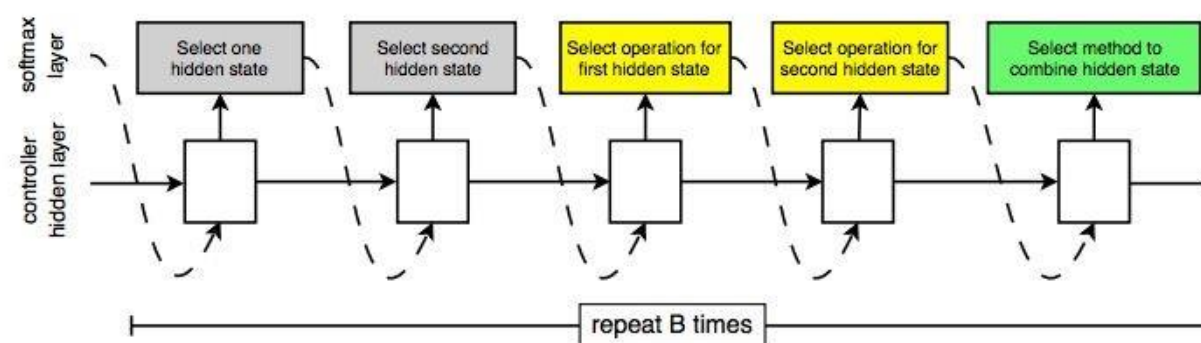
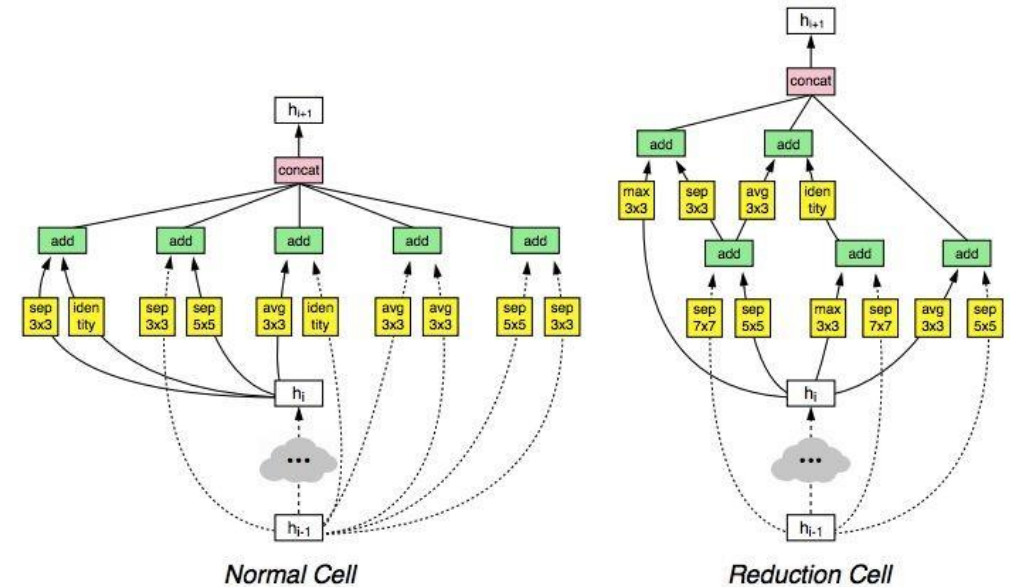
- “Controller” network that learns to design a good network architecture (output a string corresponding to network design)
- Iterate:
  - 1) Sample an architecture from search space
  - 2) Train the architecture to get a “reward”  $R$  corresponding to accuracy
  - 3) Compute gradient of sample probability, and scale by  $R$  to perform controller parameter update (i.e. increase likelihood of good architecture being sampled, decrease likelihood of bad architecture)



# Neural Architecture Search:

## Learning Transferable Architectures for Scalable Image Recognition *[Zoph et al. 2017]*

- Applying neural architecture search (NAS) to a large dataset like ImageNet is expensive
- Design a search space of building blocks (“cells”) that can be flexibly stacked
- NASNet: Use NAS to find best cell structure on smaller CIFAR-10 dataset, then transfer architecture to ImageNet



# Резюме: модели нейронных сетей

Мы разобрали основные модели

- AlexNet
- VGG
- GoogleNet
- ResNet

Также посмотрели

- ResNeXt
- SE Nets
- Wide ResNet
- ResNet with Stochastic Depth
- MobileNets
- SqueezeNet
- DenseNet

# Резюме: модели нейронных сетей

- Широко используются VGG, GoogLeNet, ResNet
- Последние улучшения архитектур:
  - увеличение глубины сетей
  - применение residual blocks = identity mappings + residual connections
  - усложнение residual blocks (attention)
  - **NAS**
- Рекомендую модели: GoogLeNet, ResNet, DenseNet
- На следующем занятии  
Caffe, NVIDIA DIGITS и TensorFlow