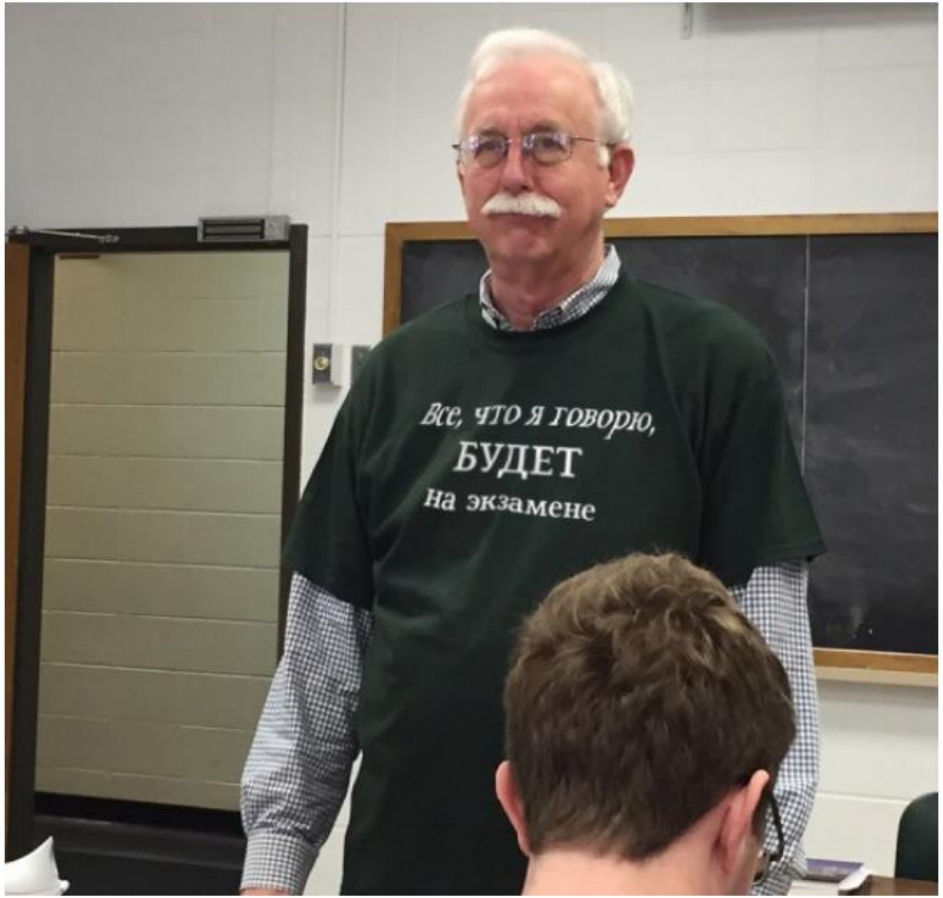
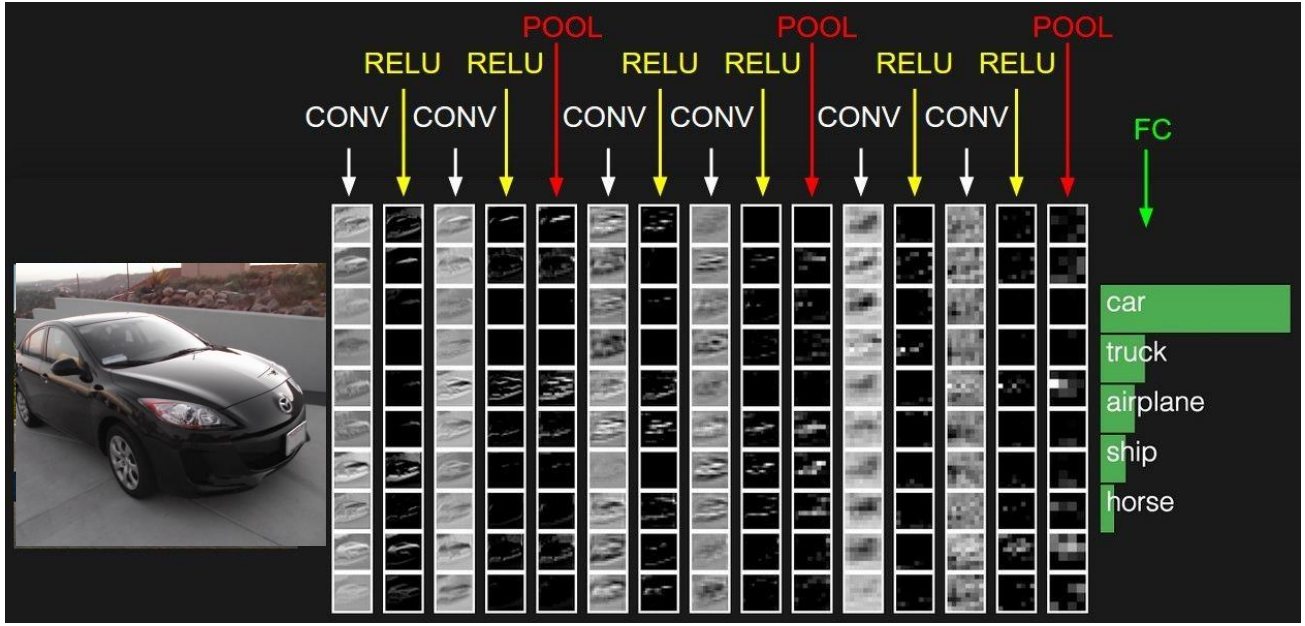
Лекция 9: Архитектуры CNC

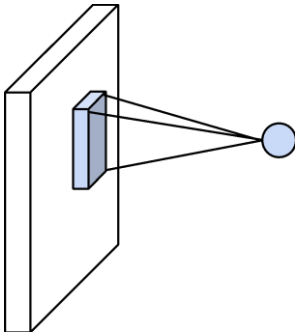


Напомним: Convolutional Neural Networks

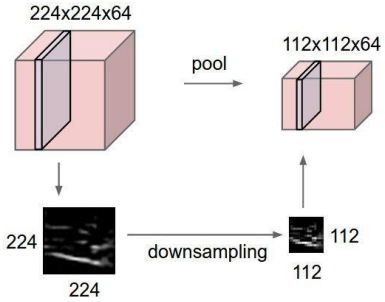


Components of CNNs

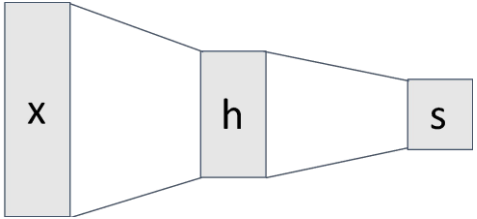
Convolution Layers



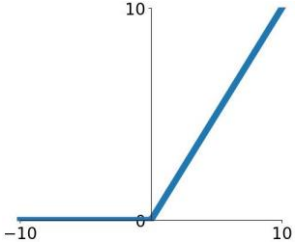
Pooling Layers



Fully-Connected Layers



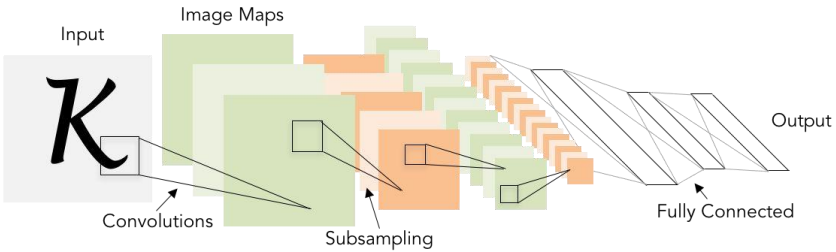
Activation Function

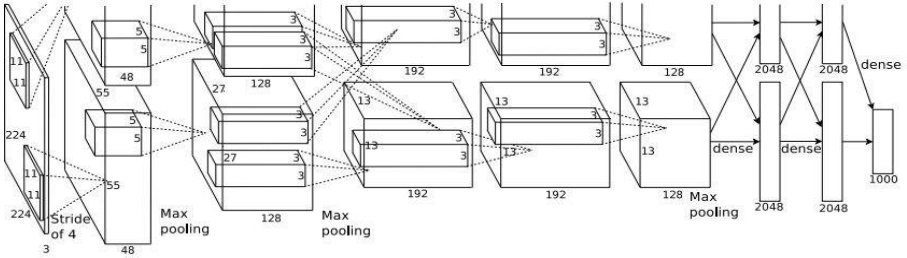


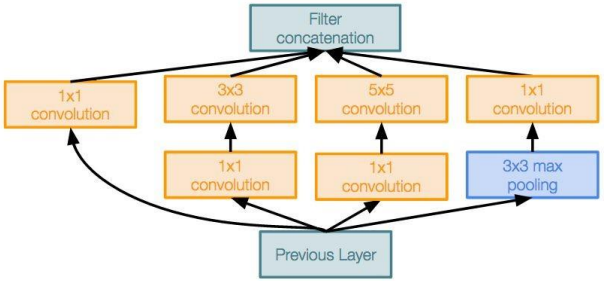
Normalization

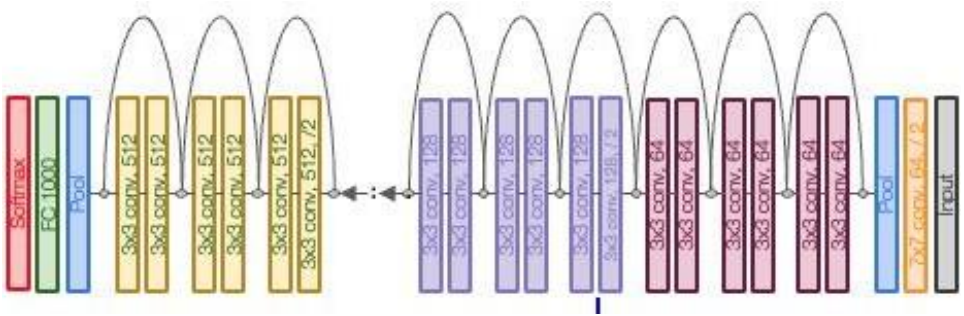
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Архитектуры CHC



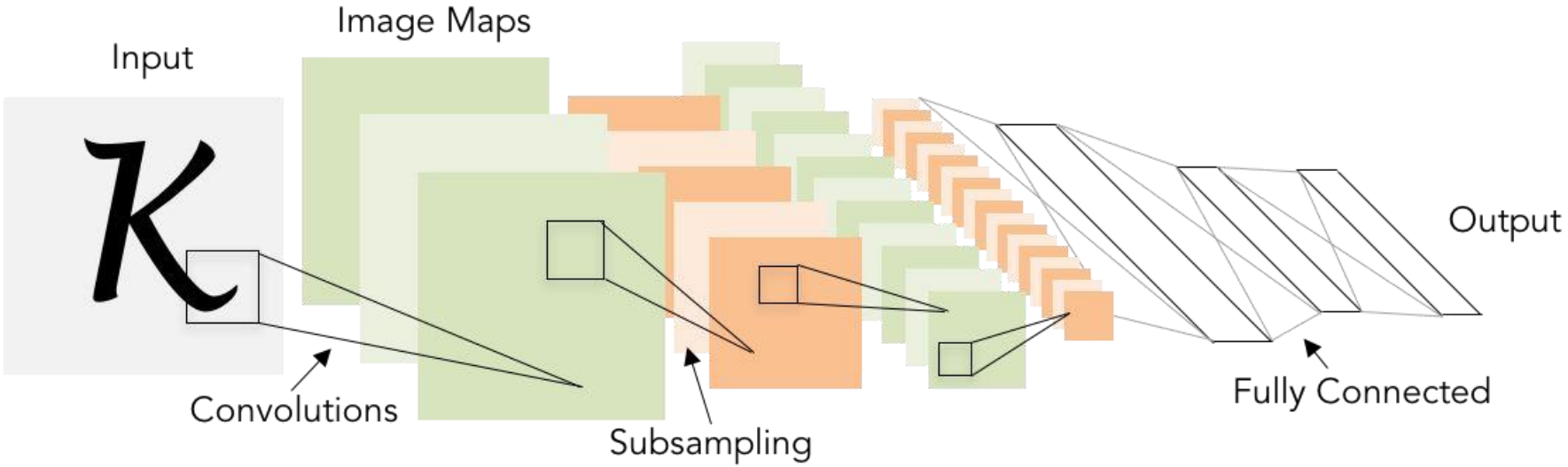






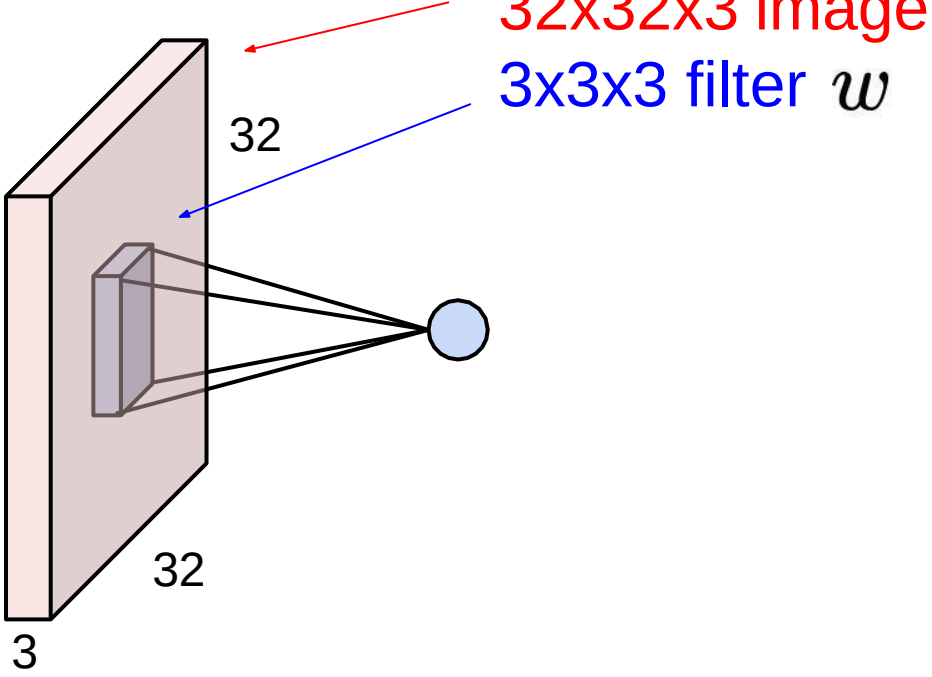
Ревью: LeNet-5

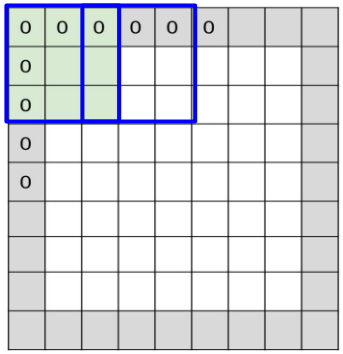
[LeCun et al., 1998]



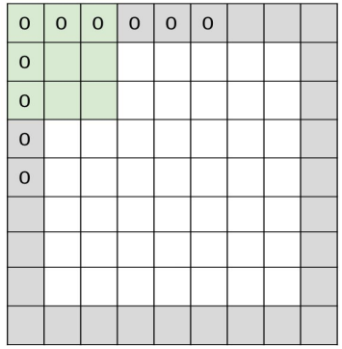
Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

Ревью: Convolution



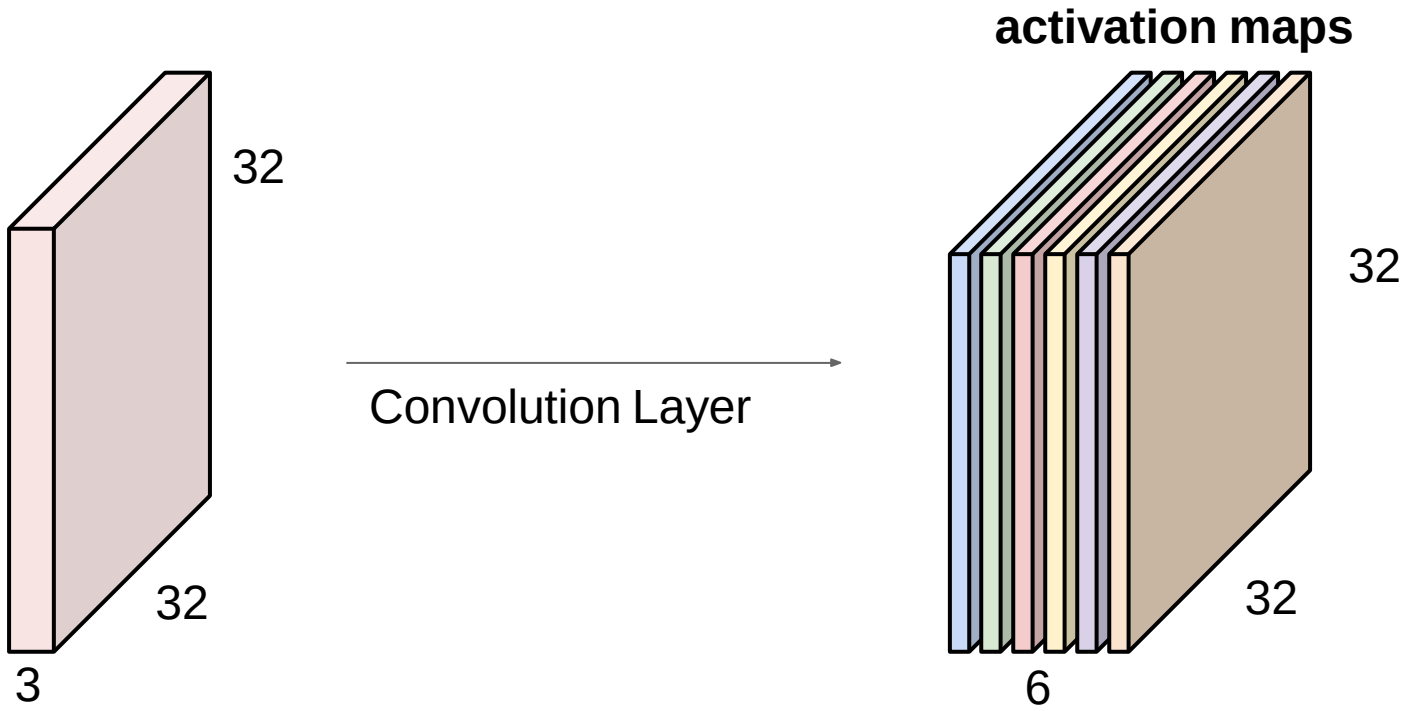


Stride:
Downsample
output activations



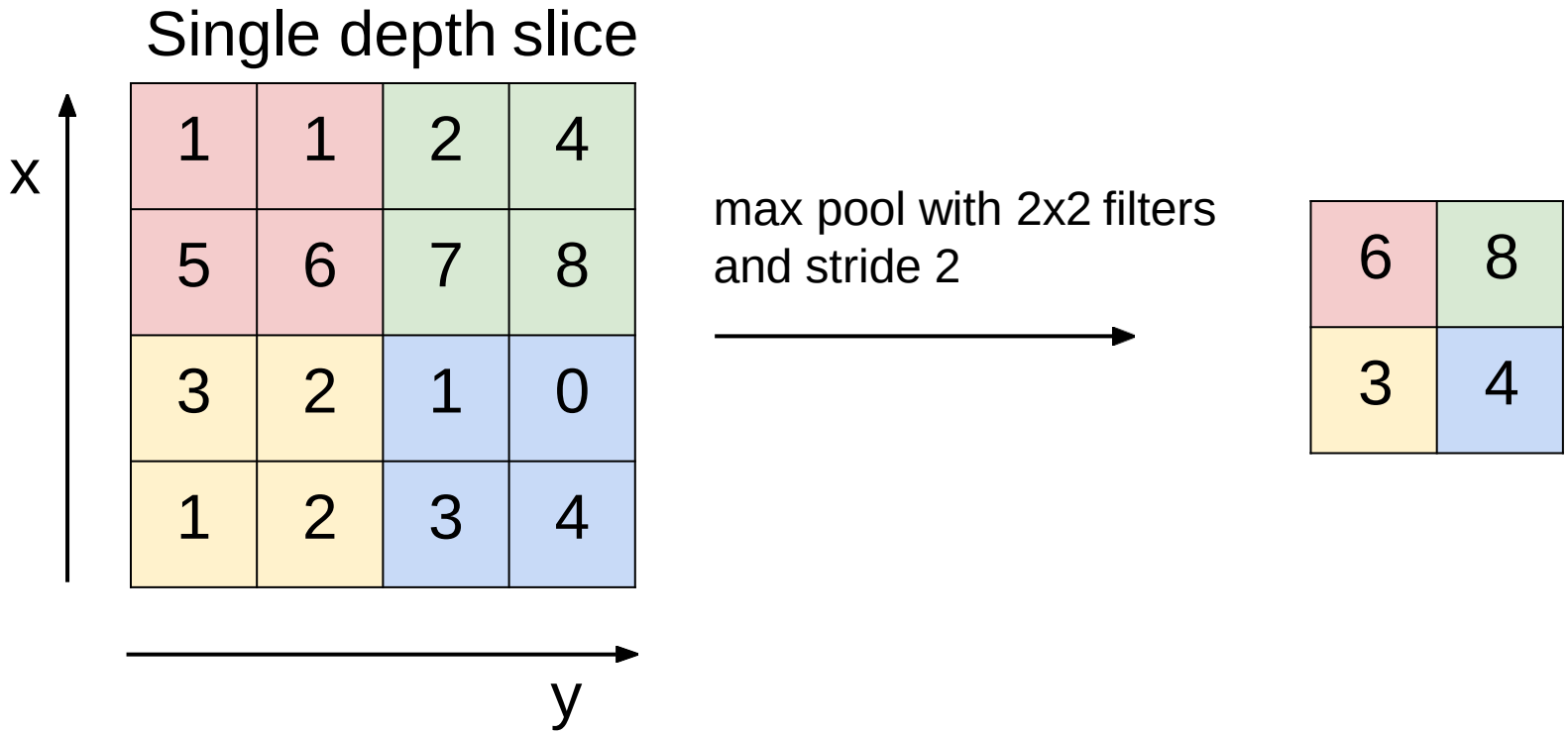
Padding:
Preserve
input spatial
dimensions in
output activations

Ревью: Convolution



Each conv filter outputs a “slice” in the activation

Ревью: Pooling



Архитектуры CHC, план

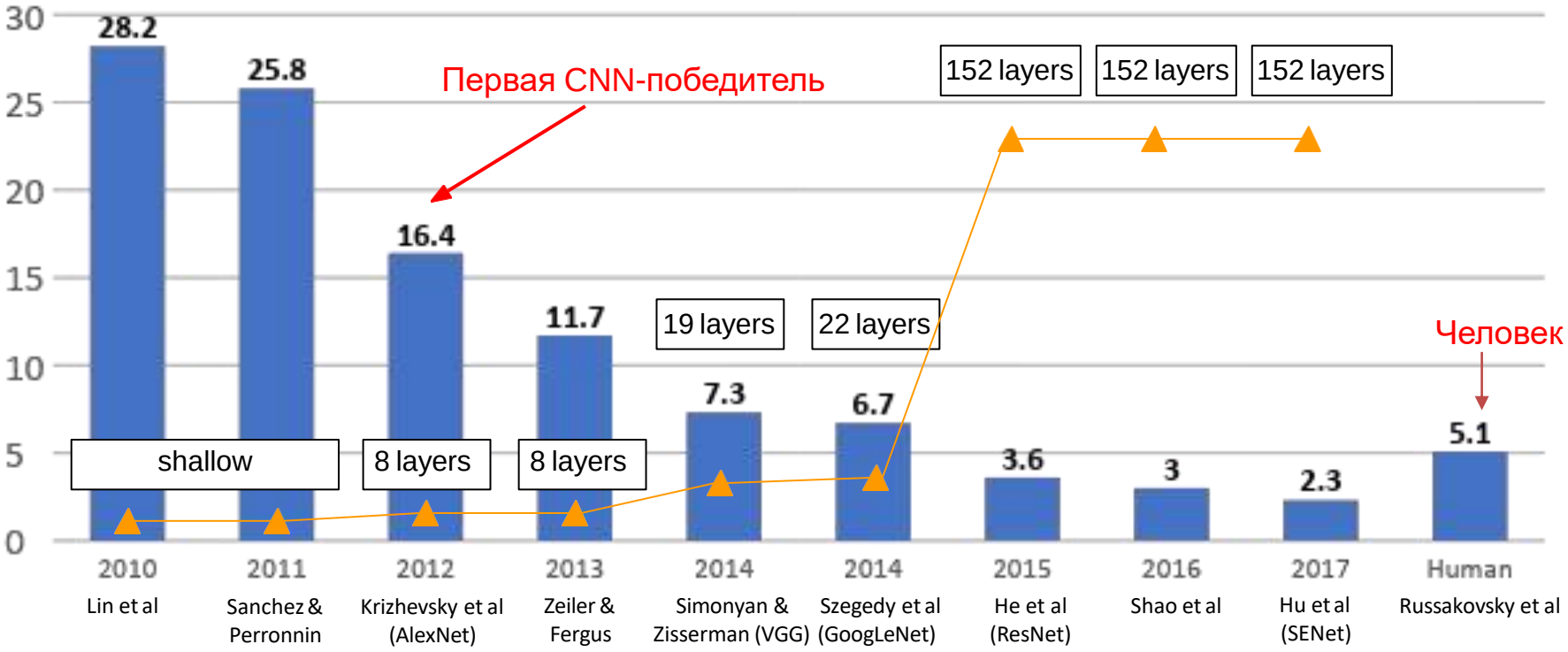
Кейсы

- AlexNet
- VGG
- GoogLeNet
- ResNet

Также....

- SENet
- Wide ResNet
- ResNeXT
- DenseNet
- MobileNets
- NASNet
- EfficientNet

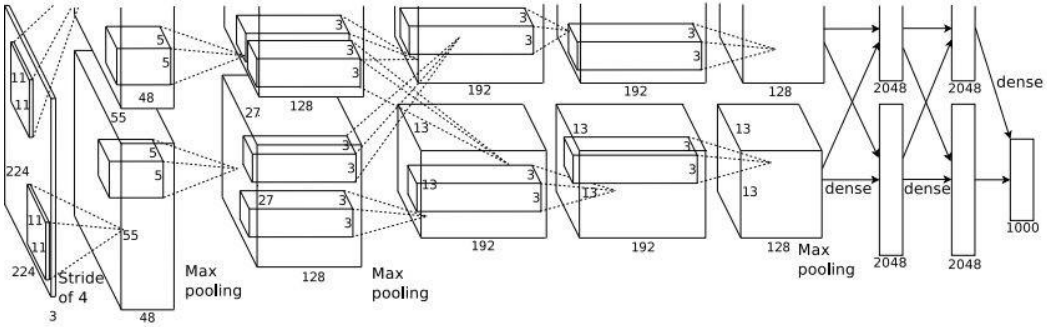
Победители ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



Кейс: AlexNet

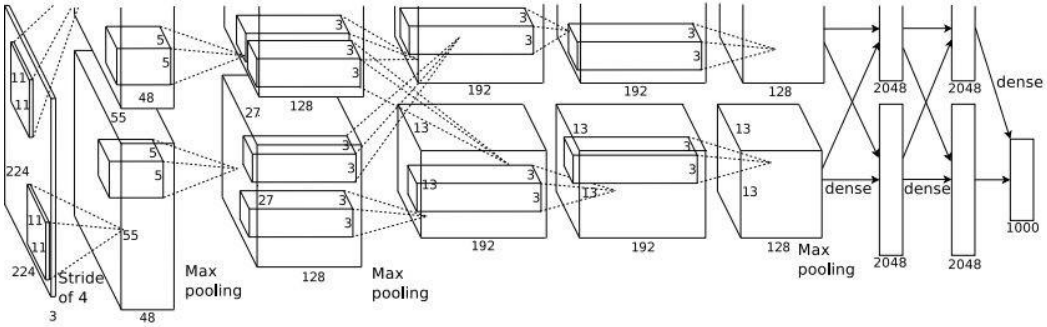
[Krizhevsky et al. 2012]

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8



Кейс: AlexNet

[Krizhevsky et al. 2012]



Вход изображения: 227x227x3

Первый слой (CONV1): 96 11x11 фильтры с шагом 4

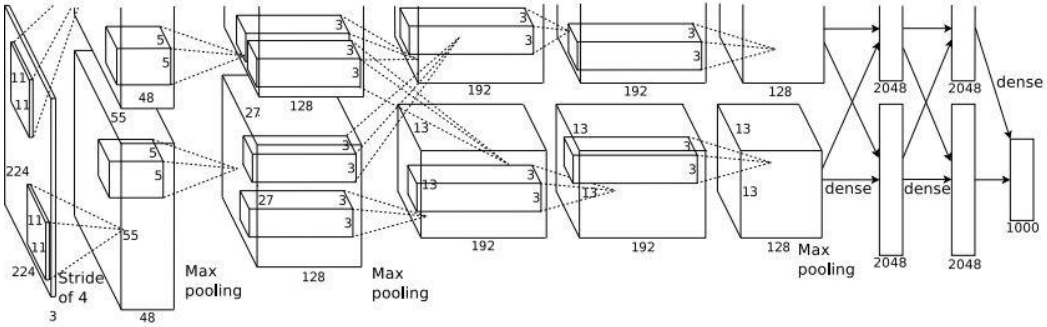
=>

Q: какой размер на выходе? подсказка: (227-11)/4+1 = 55

$$W' = (W - F + 2P) / S + 1$$

Кейс: AlexNet

[Krizhevsky et al. 2012]



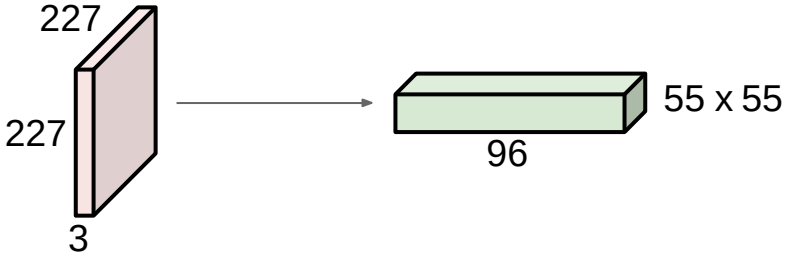
Вход: 227x227x3

Первый слой (CONV1): 96 11x11 фильтры с шагом 4

=>

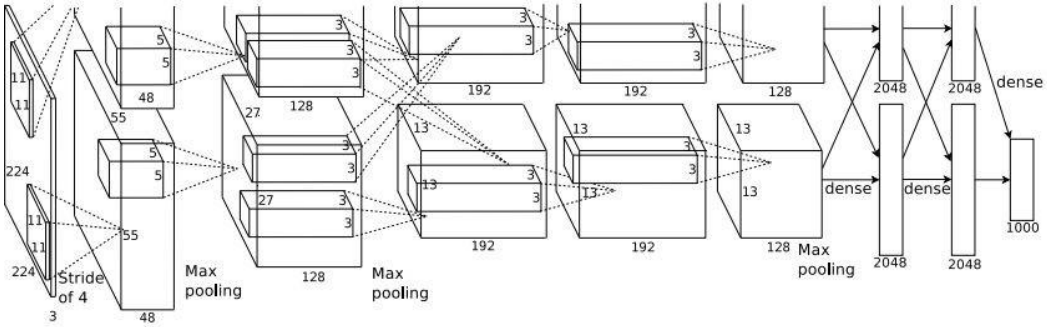
Выход **[55x55x96]**

$$W' = (W - F + 2P) / S + 1$$



Кейс: AlexNet

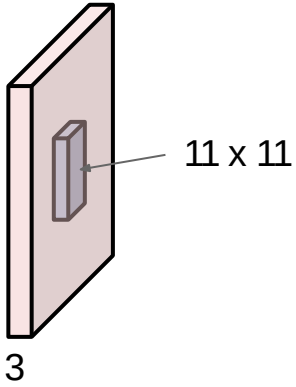
[Krizhevsky et al. 2012]



Вход: 227x227x3

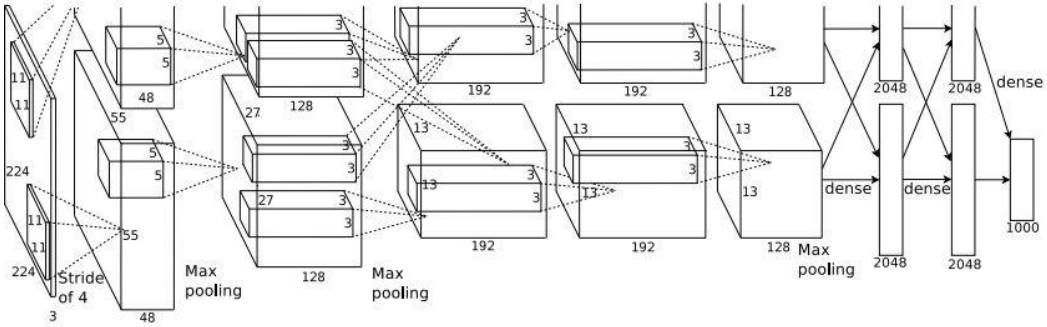
Первый слой (CONV1): 96 11x11 фильтров с шагом 4
=>
Выход **[55x55x96]**

Q: сколько параметров в этом слое?



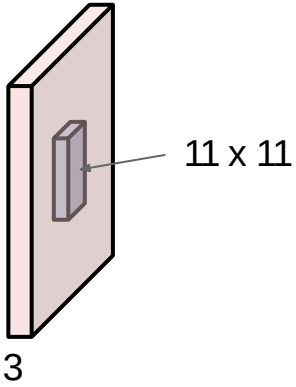
Кейс: AlexNet

[Krizhevsky et al. 2012]



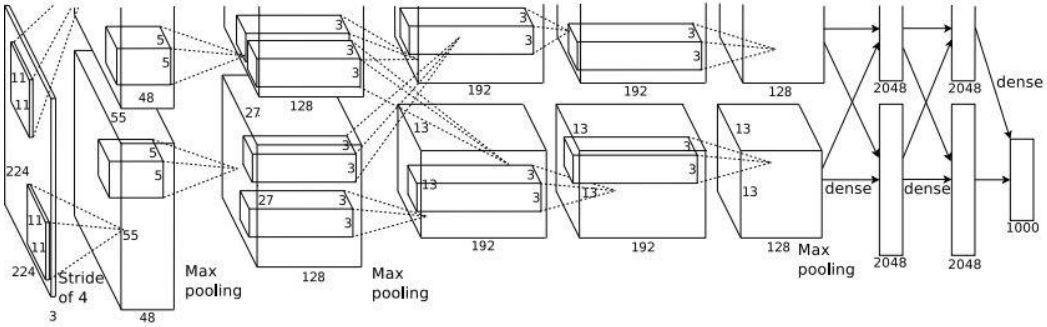
Вход: 227x227x3

Первый слой (CONV1): 96 11x11 фильтров с шагом 4
=>
Выход **[55x55x96]**
Параметры: (11*11*3 + 1)*96 = **35K**



Кейс: AlexNet

[Krizhevsky et al. 2012]



Вход: 227x227x3
После CONV1: 55x55x96

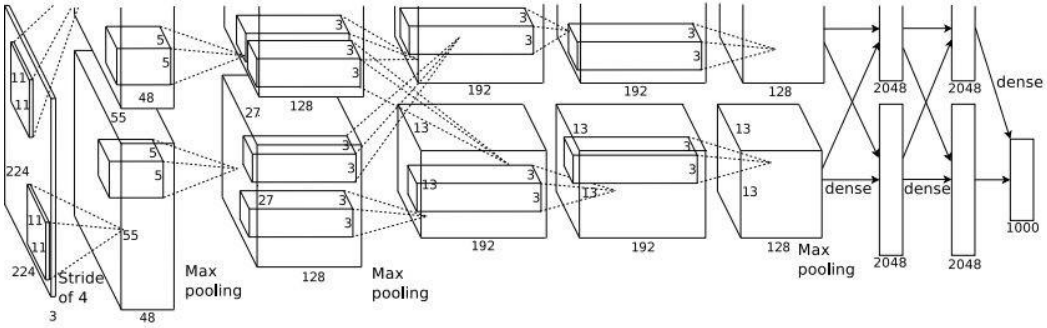
$W' = (W - F + 2P) / S + 1$

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

Q: какой размер выхода? Подсказка: (55-3)/2+1 = 27

Кейс: AlexNet

[Krizhevsky et al. 2012]



Вход: 227x227x3
После CONV1: 55x55x96

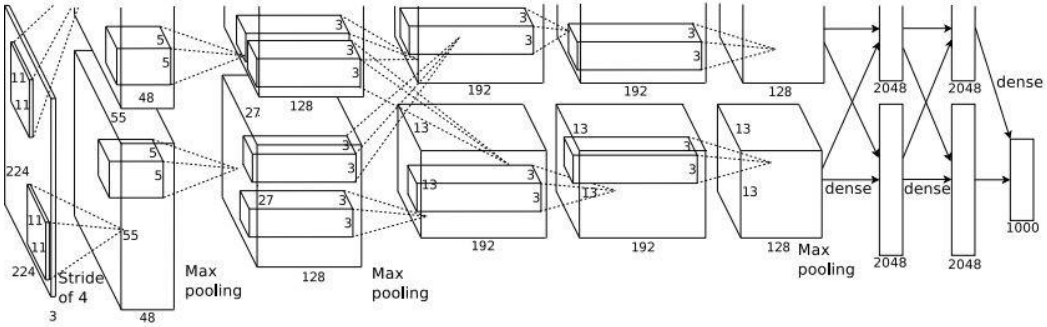
$$W' = (W - F + 2P) / S + 1$$

Второй слой (POOL1): 3x3 фильтры с шагом 2
Выход: 27x27x96

Q: сколько параметров во втором слое?

Кейс: AlexNet

[Krizhevsky et al. 2012]



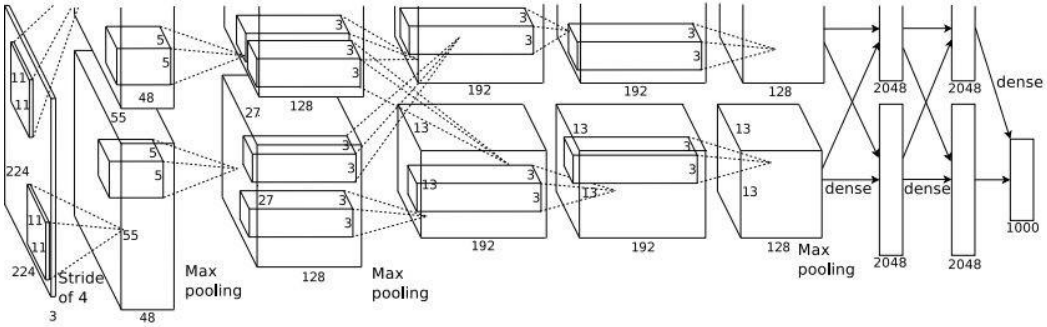
Вход: 227x227x3
После CONV1: 55x55x96

$$W' = (W - F + 2P) / S + 1$$

Второй слой (POOL1): 3x3 фильтры с шагом 2
Выход: 27x27x96
Параметров: 0 – pooling не обучаемый!

Кейс: AlexNet

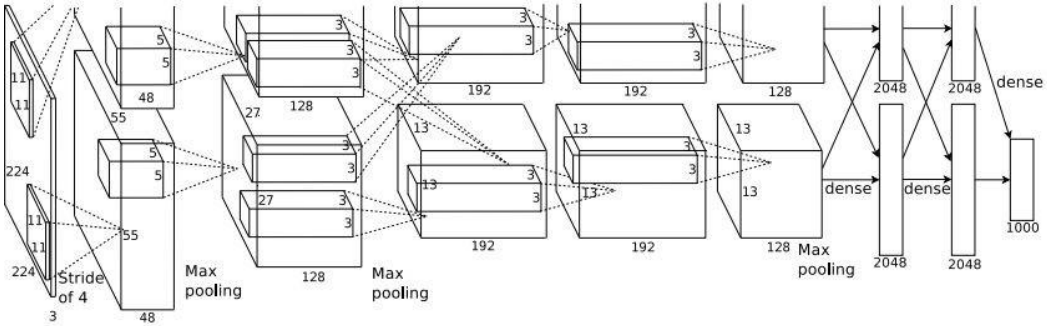
[Krizhevsky et al. 2012]



Вход: 227x227x3
После CONV1: 55x55x96
После POOL1: 27x27x96
...

Кейс: AlexNet

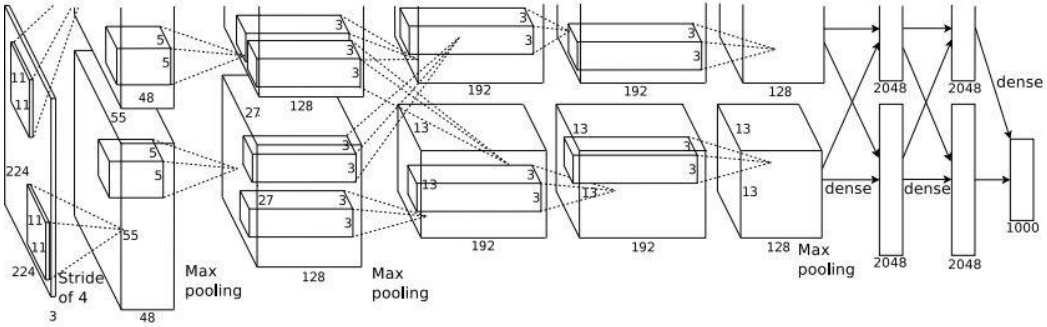
[Krizhevsky et al. 2012]



Архитектура AlexNet (упрощенная):
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 фильтров с шагом 4, pad 0
[27x27x96] MAX POOL1: 3x3 фильтры с шагом 2
[27x27x96] NORM1: слой нормализации (LRN)
[27x27x256] CONV2: 256 5x5 фильтров с шагом 1, pad 2
[13x13x256] MAX POOL2: 3x3 фильтры с шагом 2
[13x13x256] NORM2: слой нормализации (LRN)
[13x13x384] CONV3: 384 3x3 фильтра с шагом 1, pad 1
[13x13x384] CONV4: 384 3x3 фильтра с шагом 1, pad 1
[13x13x256] CONV5: 256 3x3 фильтров с шагом 1, pad 1
[6x6x256] MAX POOL3: 3x3 фильтры с шагом 2
[4096] FC6: 4096 нейронов
[4096] FC7: 4096 нейронов
[1000] FC8: 1000 нейронов, оценки для классов (class scores)

Кейс: AlexNet

[Krizhevsky et al. 2012]

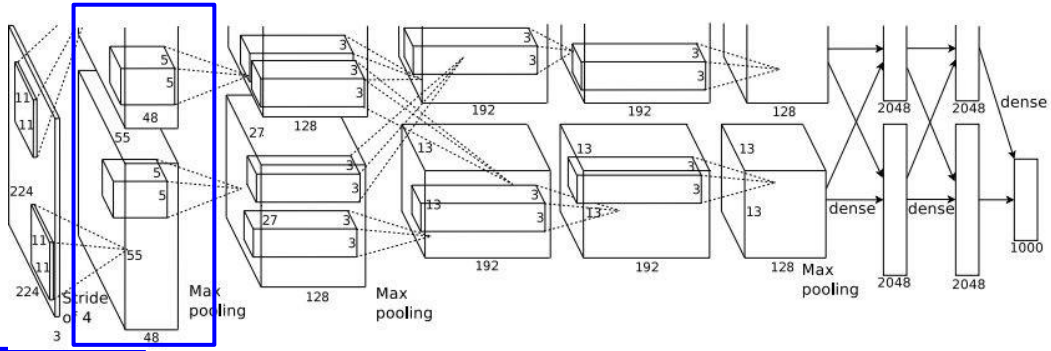


Архитектура AlexNet (упрощенная):
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 фильтров с шагом 4, pad 0
[27x27x96] MAX POOL1: 3x3 фильтры с шагом 2
[27x27x96] NORM1: слой нормализации (LRN)
[27x27x256] CONV2: 256 5x5 фильтров с шагом 1, pad 2
[13x13x256] MAX POOL2: 3x3 фильтры с шагом 2
[13x13x256] NORM2: слой нормализации (LRN)
[13x13x384] CONV3: 384 3x3 фильтра с шагом 1, pad 1
[13x13x384] CONV4: 384 3x3 фильтра с шагом 1, pad 1
[13x13x256] CONV5: 256 3x3 фильтров с шагом 1, pad 1
[6x6x256] MAX POOL3: 3x3 фильтры с шагом 2
[4096] FC6: 4096 нейронов
[4096] FC7: 4096 нейронов
[1000] FC8: 1000 нейронов, оценки для классов (class scores)

Идеи работы:
- Первое использование ReLU
- LRN Local Response Normalisation слои (больше нигде)
- Существенная аугментация данных
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
-Learning rate 1e-2, уменьшается в 10 раз вручную на плато валидационной точности
- L2 weight decay 5e-4
- 7 CHC в ансамбле: 18.2% -> 15.4%

Кейс: AlexNet

[Krizhevsky et al. 2012]

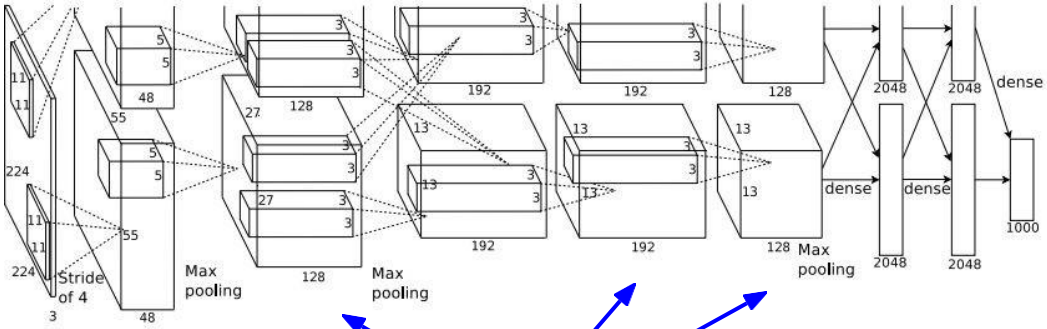


Архитектура AlexNet (упрощенная):
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 фильтров с шагом 4, pad 0 [55x55x48] x 2
[27x27x96] MAX POOL1: 3x3 фильтры с шагом 2
[27x27x96] NORM1: слой нормализации (LRN)
[27x27x256] CONV2: 256 5x5 фильтров с шагом 1, pad 2
[13x13x256] MAX POOL2: 3x3 фильтры с шагом 2
[13x13x256] NORM2: слой нормализации (LRN)
[13x13x384] CONV3: 384 3x3 фильтра с шагом 1, pad 1
[13x13x384] CONV4: 384 3x3 фильтра с шагом 1, pad 1
[13x13x256] CONV5: 256 3x3 фильтров с шагом 1, pad 1
[6x6x256] MAX POOL3: 3x3 фильтры с шагом 2
[4096] FC6: 4096 нейронов
[4096] FC7: 4096 нейронов
[1000] FC8: 1000 нейронов, оценки для классов (class scores)

Обучение на GPU:
На двух GTX 580 GPU с 3 ГБ памяти.
СНС распределена между 2 GPUs, по половине нейронов (карт активации) на каждом GPU.

Кейс: AlexNet

[Krizhevsky et al. 2012]

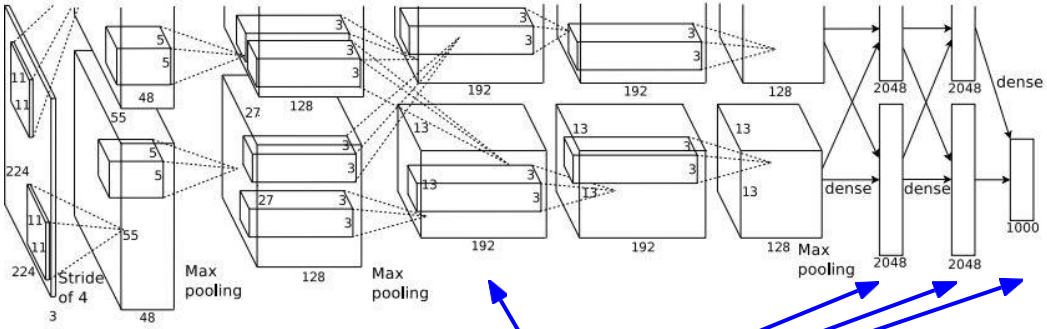


Архитектура AlexNet (упрощенная):
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 фильтров с шагом 4, pad 0
[27x27x96] MAX POOL1: 3x3 фильтры с шагом 2
[27x27x96] NORM1: слой нормализации (LRN)
[27x27x256] CONV2: 256 5x5 фильтров с шагом 1, pad 2
[13x13x256] MAX POOL2: 3x3 фильтры с шагом 2
[13x13x256] NORM2: слой нормализации (LRN)
[13x13x384] CONV3: 384 3x3 фильтра с шагом 1, pad 1
[13x13x384] CONV4: 384 3x3 фильтра с шагом 1, pad 1
[13x13x256] CONV5: 256 3x3 фильтров с шагом 1, pad 1
[6x6x256] MAX POOL3: 3x3 фильтры с шагом 2
[4096] FC6: 4096 нейронов
[4096] FC7: 4096 нейронов
[1000] FC8: 1000 нейронов, оценки для классов (class scores)

CONV1, CONV2, CONV4, CONV5:
Связаны только с картами активации **своего GPU**

Кейс: AlexNet

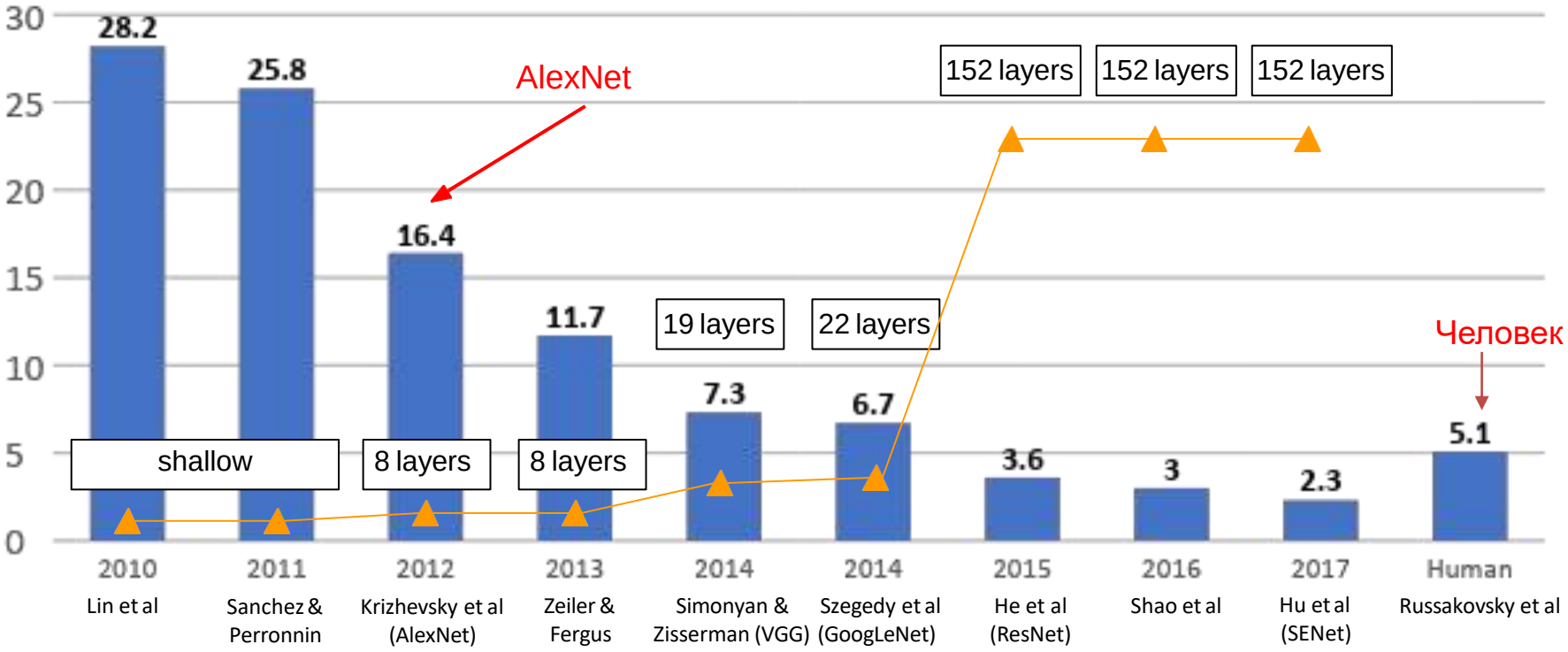
[Krizhevsky et al. 2012]



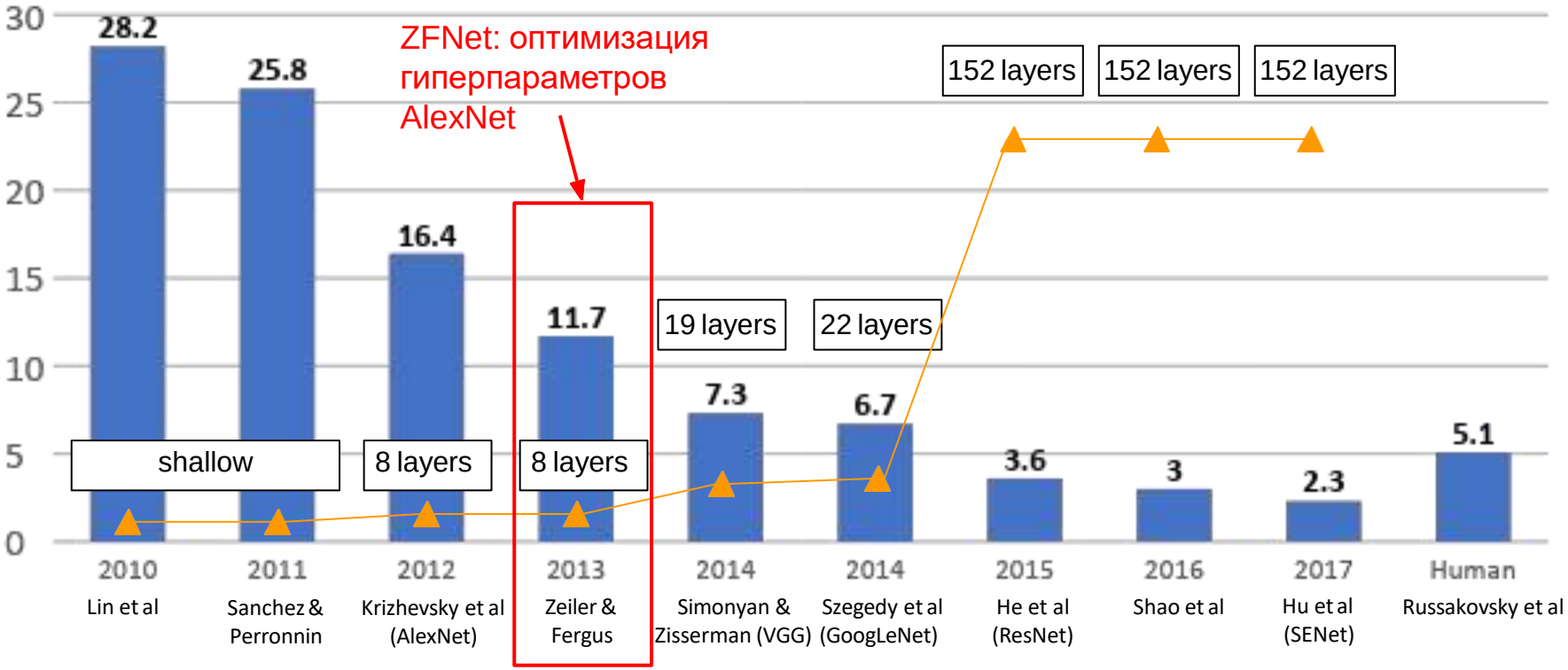
Архитектура AlexNet (упрощенная):
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 фильтров с шагом 4, pad 0
[27x27x96] MAX POOL1: 3x3 фильтры с шагом 2
[27x27x96] NORM1: слой нормализации (LRN)
[27x27x256] CONV2: 256 5x5 фильтров с шагом 1, pad 2
[13x13x256] MAX POOL2: 3x3 фильтры с шагом 2
[13x13x256] NORM2: слой нормализации (LRN)
[13x13x384] CONV3: 384 3x3 фильтра с шагом 1, pad 1
[13x13x384] CONV4: 384 3x3 фильтра с шагом 1, pad 1
[13x13x256] CONV5: 256 3x3 фильтров с шагом 1, pad 1
[6x6x256] MAX POOL3: 3x3 фильтры с шагом 2
[4096] FC6: 4096 нейронов
[4096] FC7: 4096 нейронов
[1000] FC8: 1000 нейронов, оценки для классов (class scores)

CONV3, FC6, FC7, FC8:
Связаны со всеми активациями предыдущего слоя, за счет коммуникации между GPUs

Победители ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

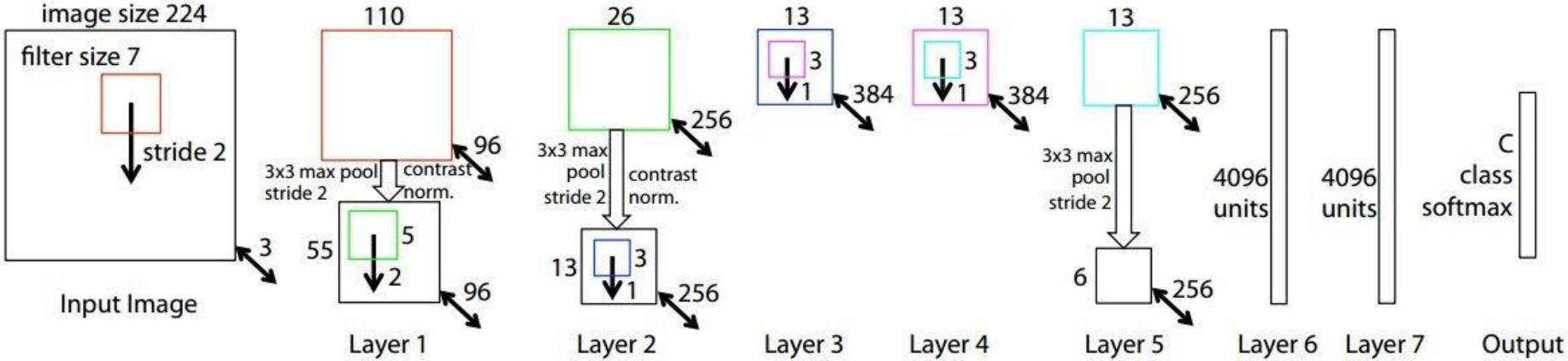


Победители ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



ZFNet

[Zeiler and Fergus, 2013]



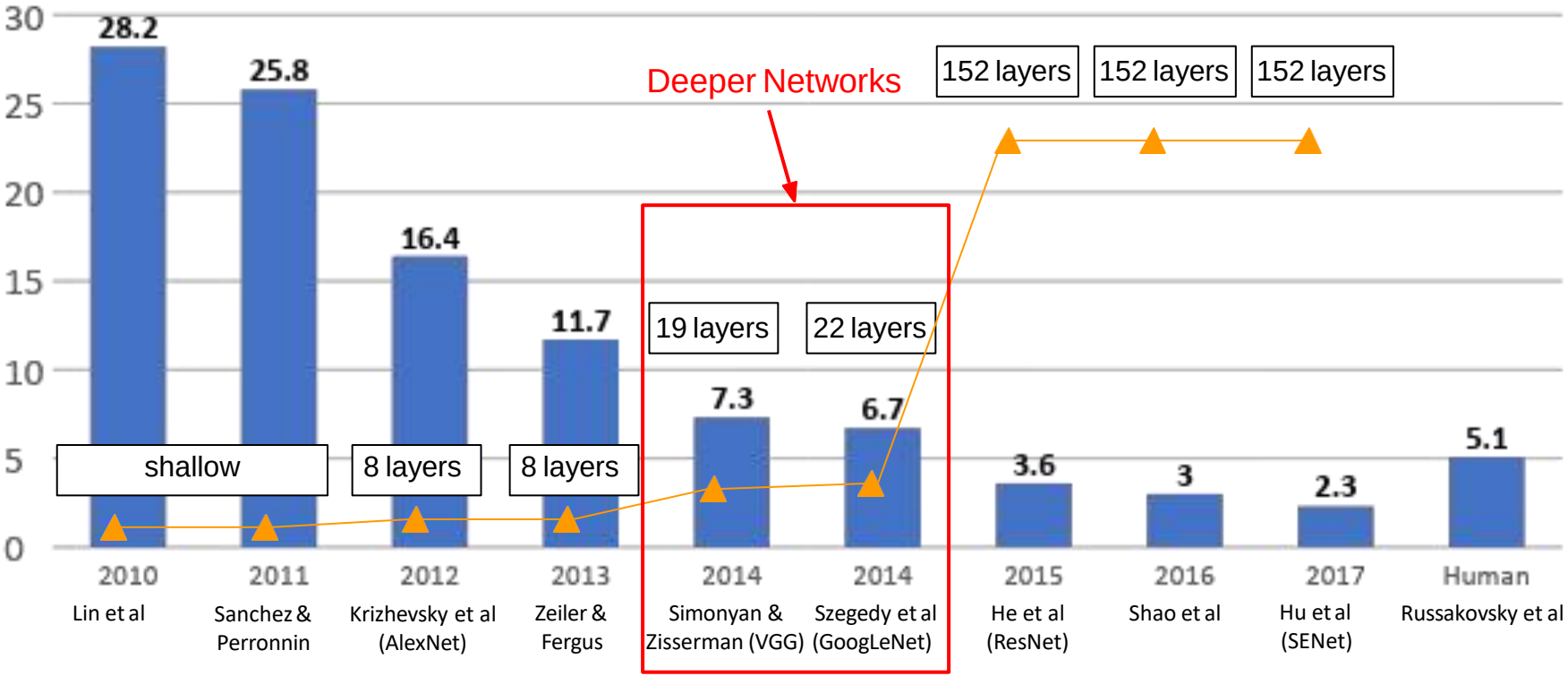
Отличия от AlexNet:

CONV1: изменили (11x11 stride 4) на (7x7 stride 2)

CONV3,4,5: вместо 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 16.4% -> 11.7%

Победители ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



Кейс: VGGNet

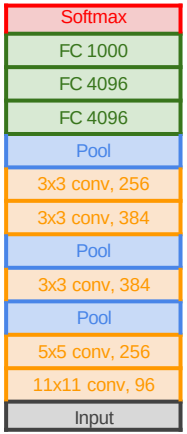
[Simonyan and Zisserman, 2014]

Маленькие фильтры, большая глубина сети

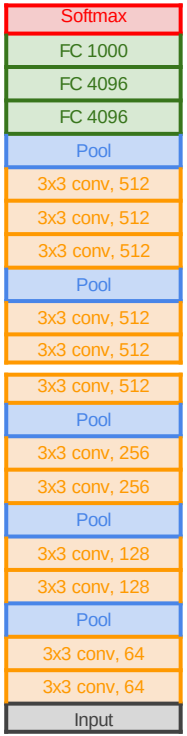
8 слоев (AlexNet)
-> 16 - 19 слоев (VGG16Net)

Только 3x3 CONV tride 1, pad 1
и 2x2 MAX POOL stride 2

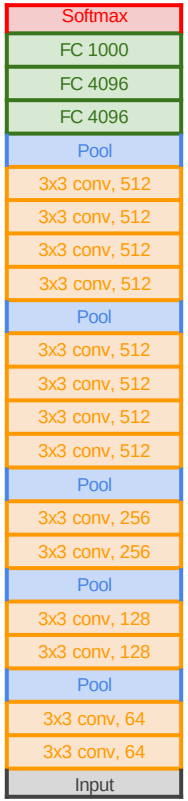
11.7% top 5 error in ILSVRC'13 (ZFNet)
-> 7.3% top 5 error in ILSVRC'14



AlexNet



VGG16

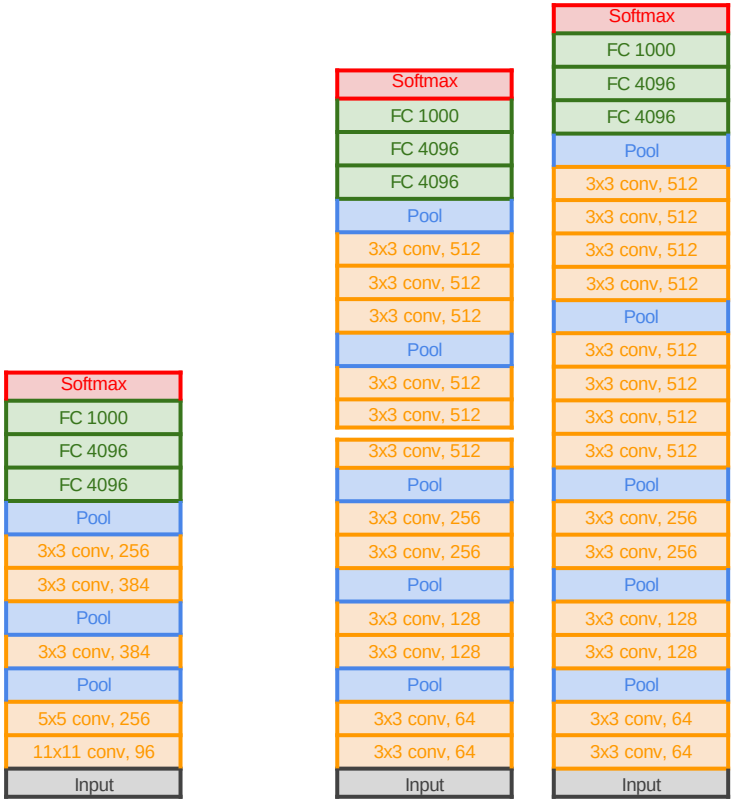


VGG19

Кейс: VGGNet

[Simonyan and Zisserman, 2014]

Q: почему маленькие фильтры? (3x3 conv)



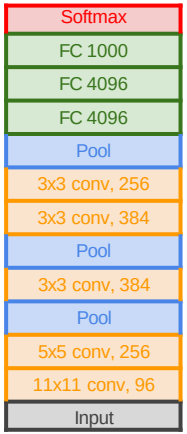
Кейс: VGGNet

[Simonyan and Zisserman, 2014]

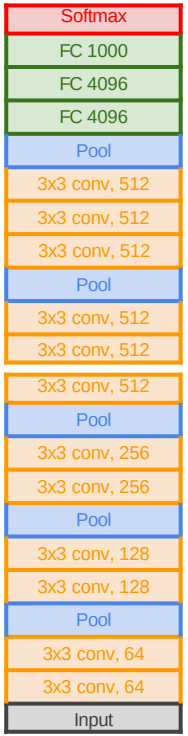
Q: почему маленькие фильтры? (3x3 conv)

Стек из трех 3x3 conv (stride 1) слоев имеет то же поле зрения (**effective receptive field)** как один слой 7x7

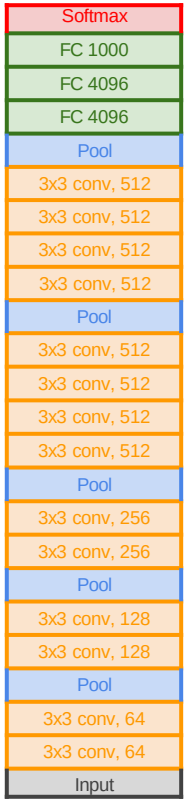
Q: Как посчитать receptive field трех 3x3 conv (stride 1) слоев?



AlexNet



VGG16

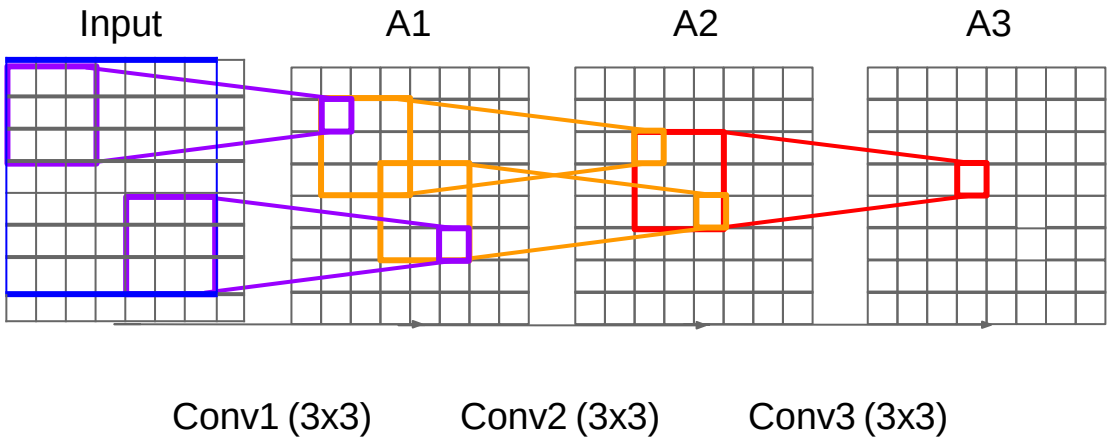


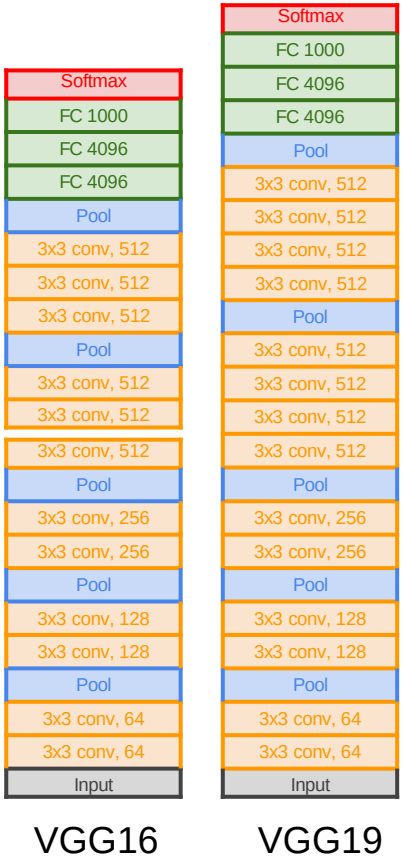
VGG19

Кейс: VGGNet

[Simonyan and Zisserman, 2014]

Q: Как посчитать receptive field трех 3x3 conv (stride 1) слоев?





Кейс: VGGNet

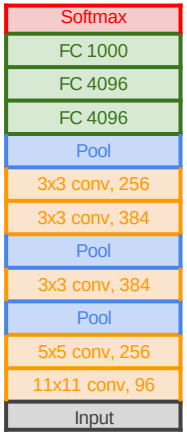
[Simonyan and Zisserman, 2014]

Q: Почему только маленькие фильтры? (3x3 conv)

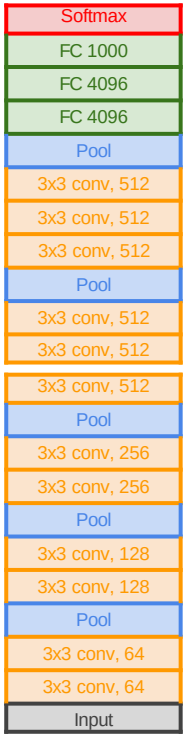
Стек из трех 3x3 conv (stride 1) слоев имеет то же поле зрения (**effective receptive field**) что и 7x7 conv слой

Но более глубокий, больше нелинейностей

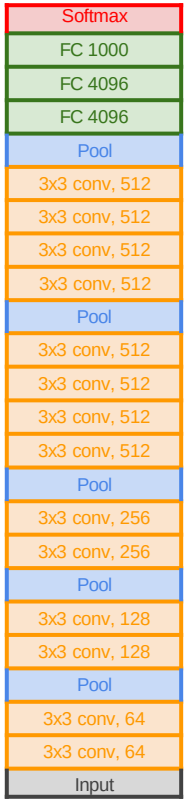
И меньше параметров: $3 * (3^2C^2)$
vs. 7^2C^2 для C каналов
=> более устойчиво



AlexNet

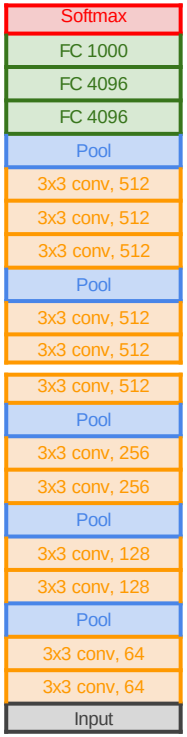


VGG16



VGG19

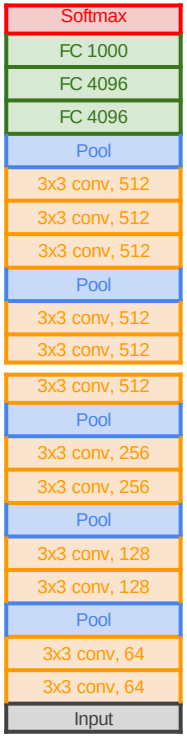
INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128]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



VGG16

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

TOTAL memory: 24M * 4 bytes ~= 96MB / image (for a forward pass)
TOTAL params: 138M parameters



VGG16

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

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

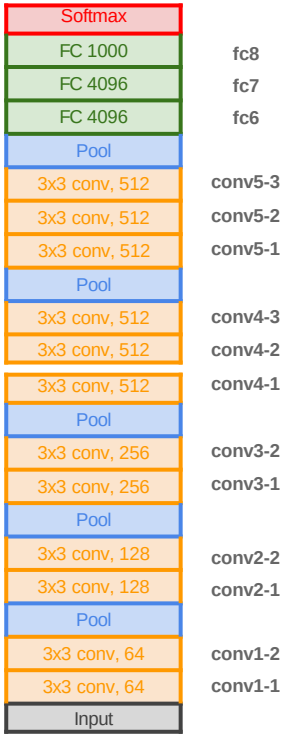
Отметим:

Основная память в ранних CONV слоях

Основные параметры в поздних полносвязных слоях FC

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

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters



VGG16

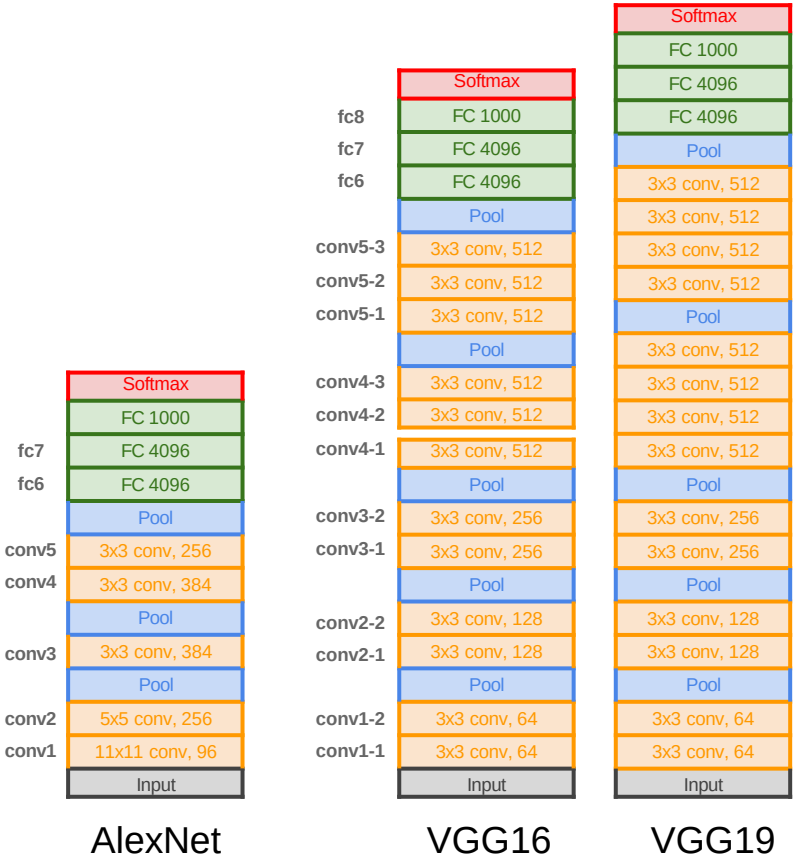
Общие названия

Кейс: VGGNet

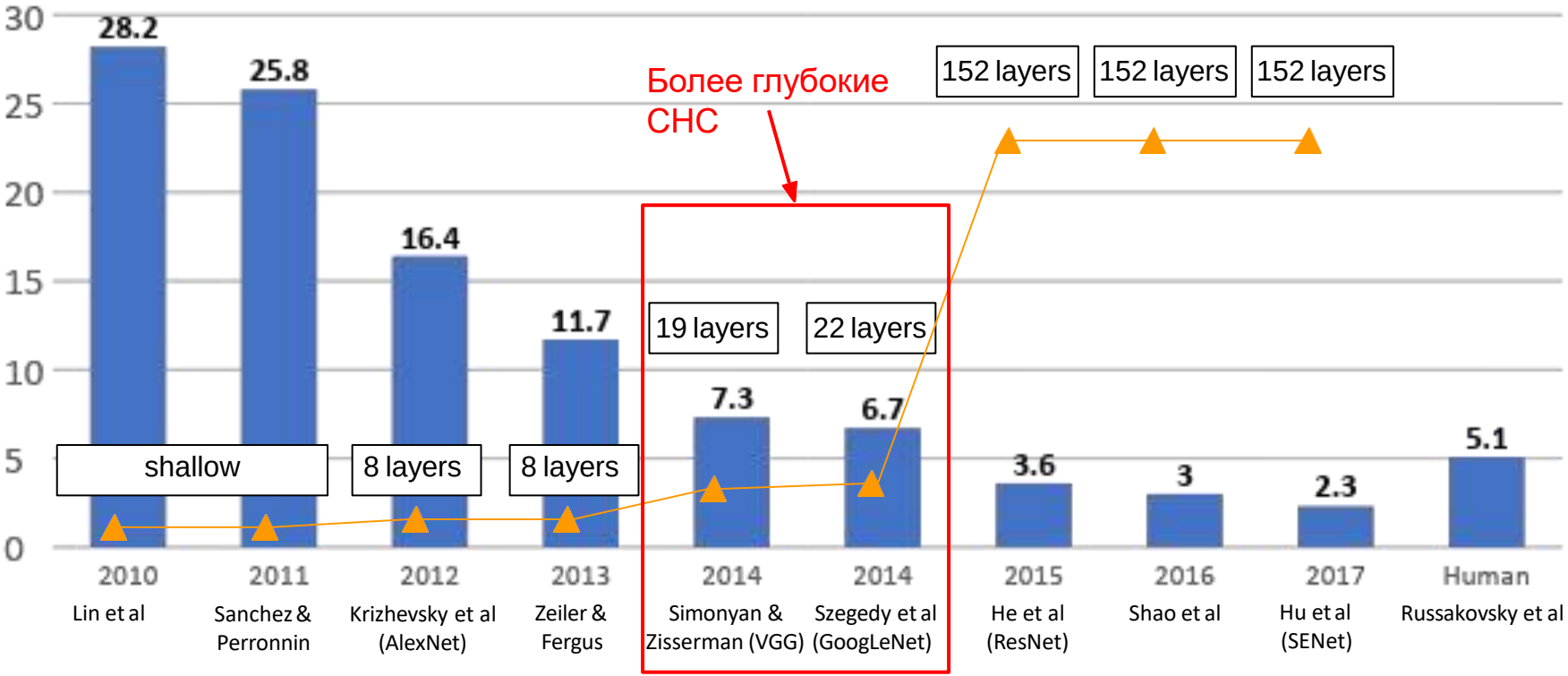
[Simonyan and Zisserman, 2014]

Идеи/достижения:

- ILSVRC'14 второй в классификации, уступил GoogleNet, первый в локализации
- Обучение подобно AlexNet
- Нет нормализациии Local Response Normalisation (LRN)
- Используем VGG16 or VGG19 (VGG19 немного лучше, но сильно больше)
- Ансамбли для уточнения
- FC7 признаки хороши для многих задач, в том числе для perceptual loss



Победители ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

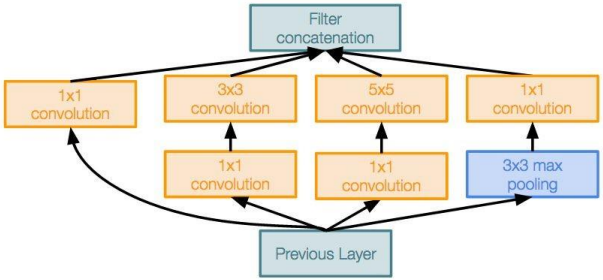


Кейс: GoogLeNet

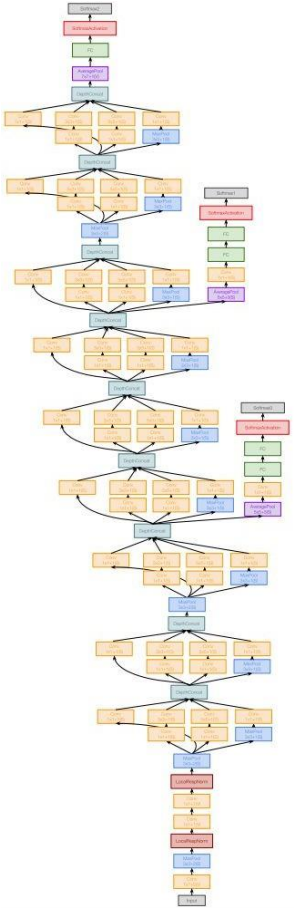
[Szegedy et al., 2014]

Глубокие, вычислительно эффективные CHC

- ILSVRC'14 победитель (6.7% top 5 error)
- 22 слоев
- Всего 5 миллионов весов!
  в 12 раз меньше AlexNet
  в 27 раз меньше VGG-16
- Эффективный "Inception" module
- Нет FC слоев!



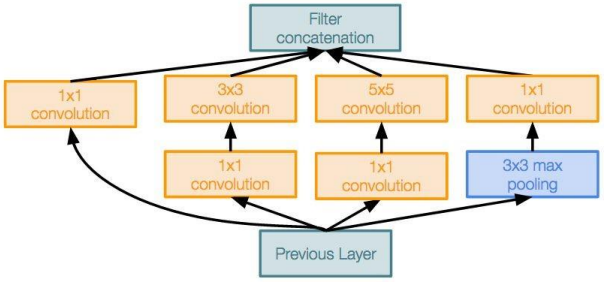
Inception module

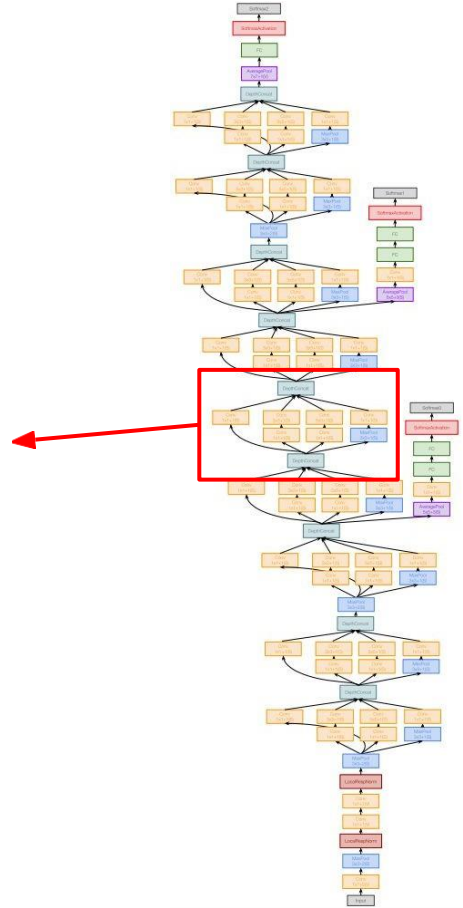


Кейс: GoogLeNet

[Szegedy et al., 2014]

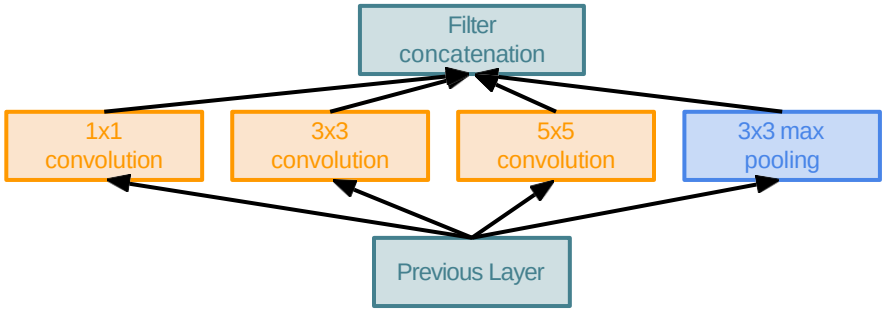
“Inception module”: удачная топология, сеть внутри сети (network within a network) и соберем их в стек





Кейс: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

Применим параллельные фильтры:

- Несколько receptive field sizes для свертки (1x1, 3x3, 5x5)
- Pooling (3x3)

Соберем все это поканально

Q: В чем может быть проблема?
В вычислительной сложности

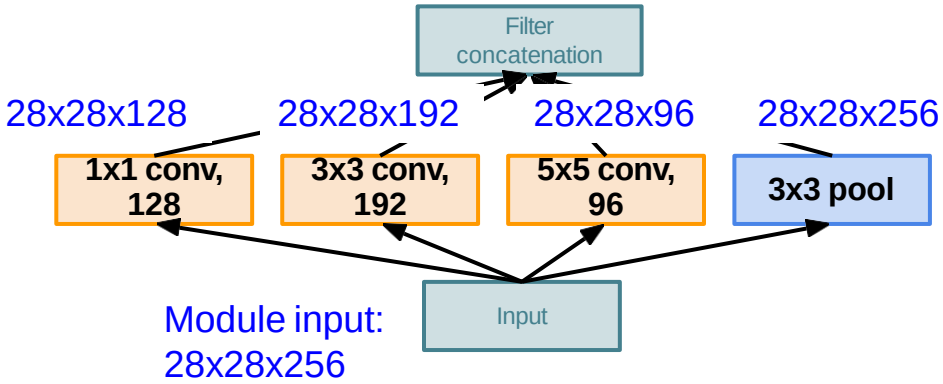
Кейс: GoogLeNet

[Szegedy et al., 2014]

Проблема в вычислительной сложности

Пример:

Q1: Какие будут размерности по выходу у всех фильтров?



Naive Inception module

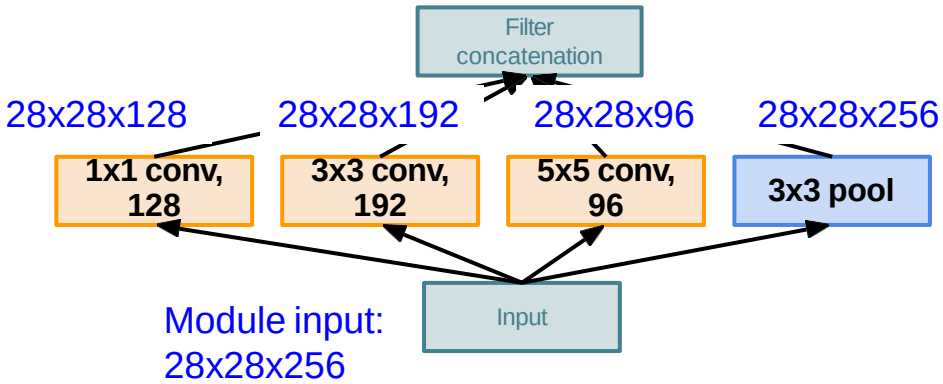
Кейс: GoogLeNet

[Szegedy et al., 2014]

Проблема в вычислительной сложности

Пример:

Q2: Какая будет размерность после конкатенации фильтров…?



Naive Inception module

Кейс: GoogLeNet

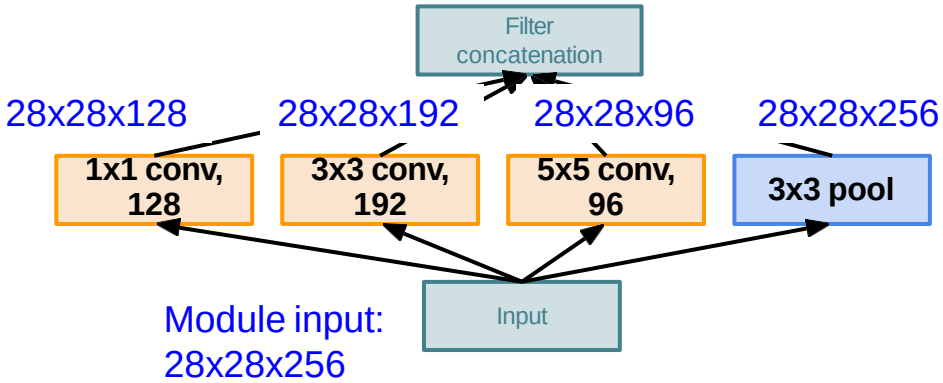
[Szegedy et al., 2014]

Проблема в вычислительной сложности

Пример:

Q2: Какая будет размерность после конкатенации фильтров…?

28x28x(128+192+96+256) = 28x28x672



Naive Inception module

Кейс: GoogLeNet

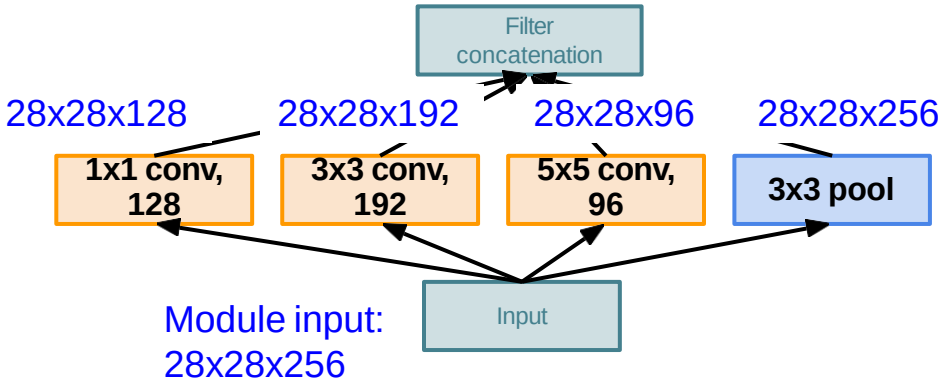
[Szegedy et al., 2014]

Проблема в вычислительной сложности

Пример:

Q2: Какая будет размерность после конкатенации фильтров…?

28x28x(128+192+96+256) = 28x28x672



Naive Inception module

Сколько это операций:

[1x1 conv, 128] 28x28x128x1x1x256

[3x3 conv, 192] 28x28x**192x3x3x256**

[5x5 conv, 96] 28x28x**96x5x5x256**

Всего: 854M операций

Кейс: GoogLeNet

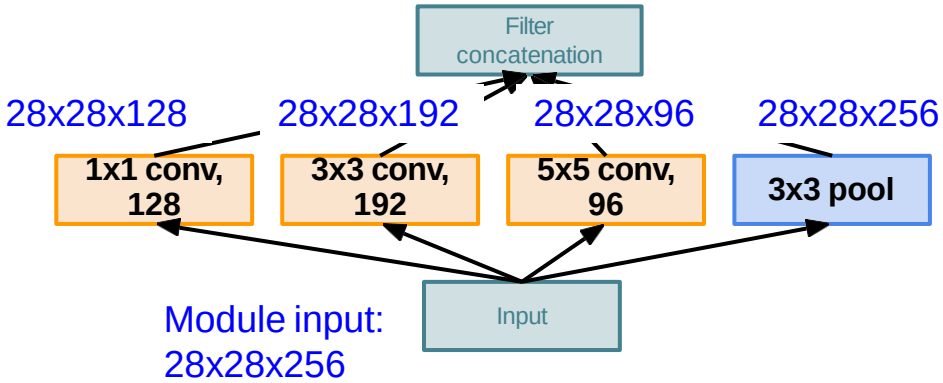
[Szegedy et al., 2014]

Проблема в вычислительной сложности

Пример:

Q2: Какая будет размерность после конкатенации фильтров…?

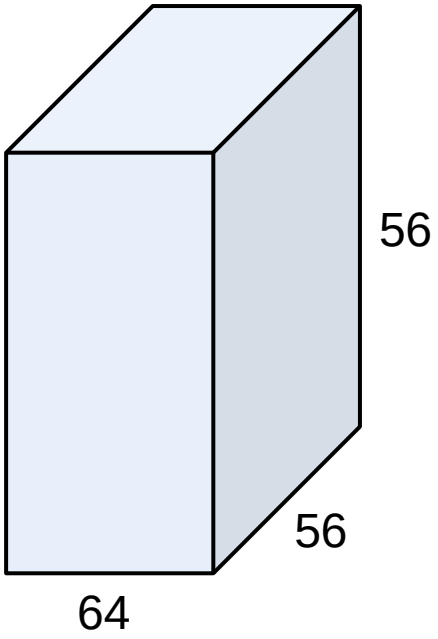
28x28x(128+192+96+256) = 28x28x672



Naive Inception module

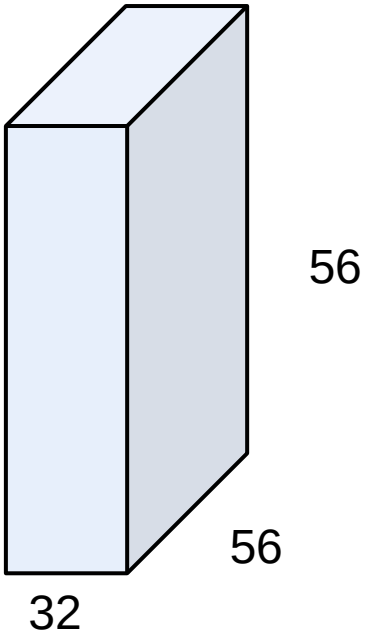
Решение: “bottleneck” слои из сверток 1x1 чтобы сократить глубину – количество каналов

Идея: 1x1 convolutions



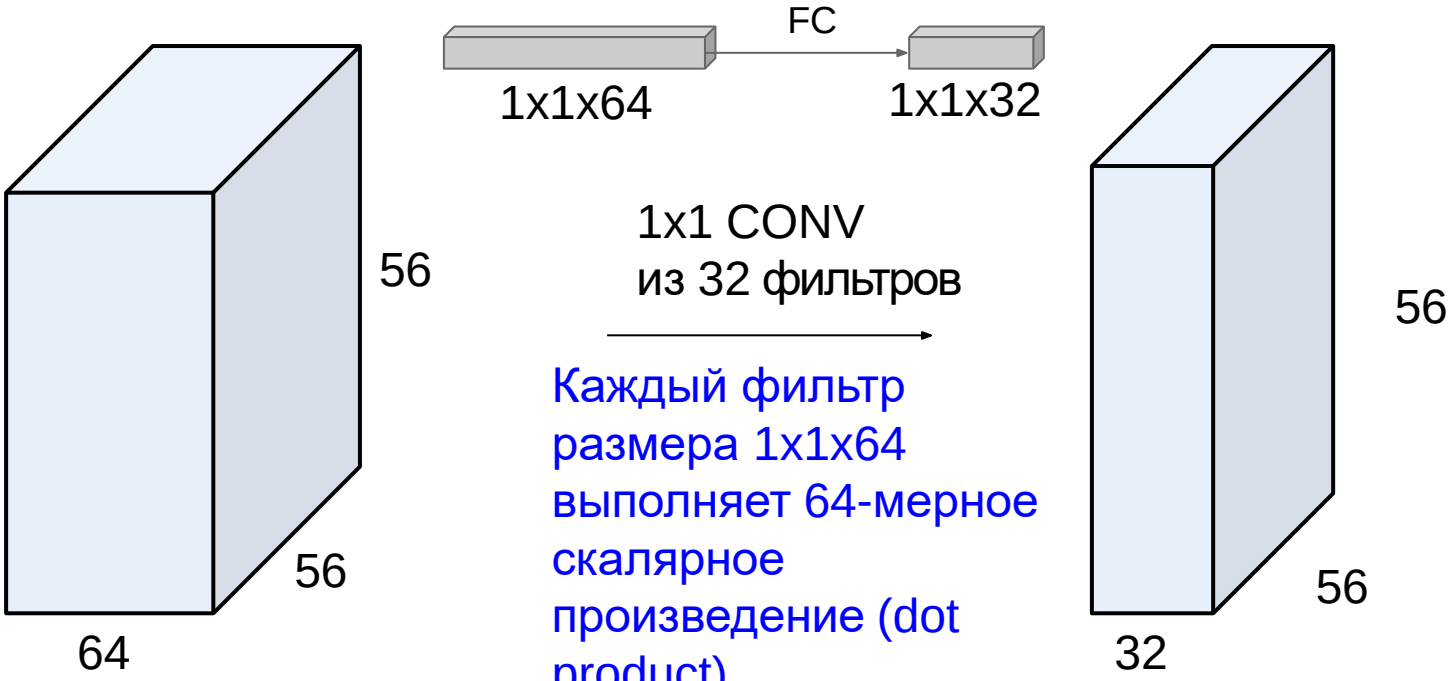
1x1 CONV
из 32 фильтров

Каждый фильтр размера 1x1x64 выполняет 64-мерное скалярное произведение (dot product)



Идея: 1x1 convolutions

Альтернативная интерпретация - это FC слой в каждом пикселе!



Каждый фильтр размера 1x1x64 выполняет 64-мерное скалярное произведение (dot product)

Идея: 1x1 convolutions

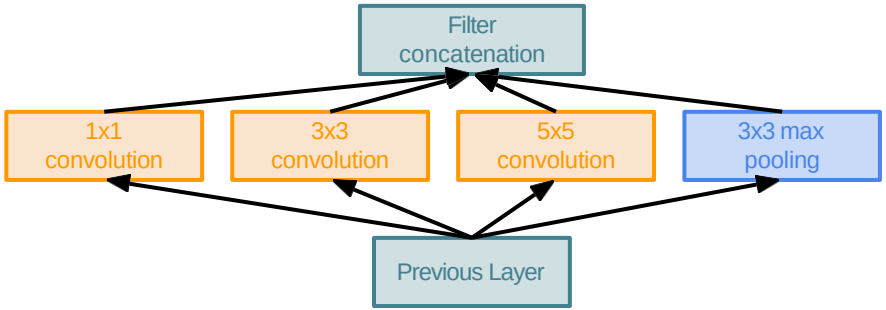
Альтернативная интерпретация - это FC слой в каждом пикселе!

Сохраняет пространственные размерности, сокращает глубину

Реализует проекцию в меньшую размерность - комбинирует слои активации

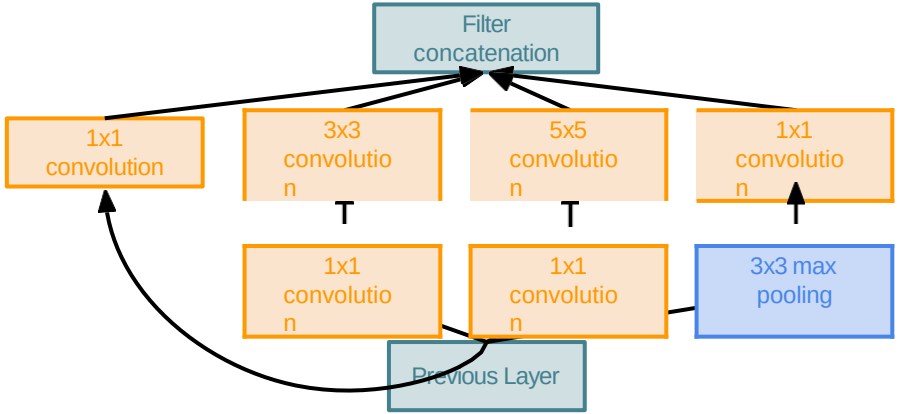
Кейс: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

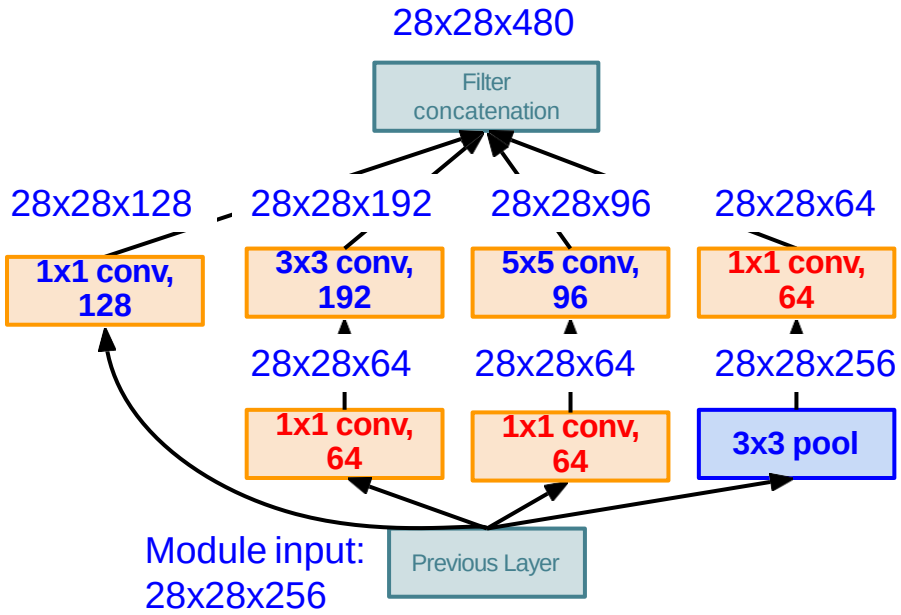
Добавили 1x1 conv “bottleneck” слои



Inception module с сокращением размерности

Кейс: GoogLeNet

[Szegedy et al., 2014]



Inception module с сокращением размерности

Используем параллельные фильтры и 1x1 conv, 64 фильтры:

Операции:

[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256

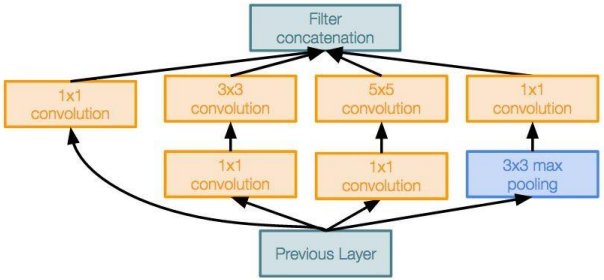
Всего: 358M операций

Лучше чем 854M операций можем сократить глубину еще и после pooling

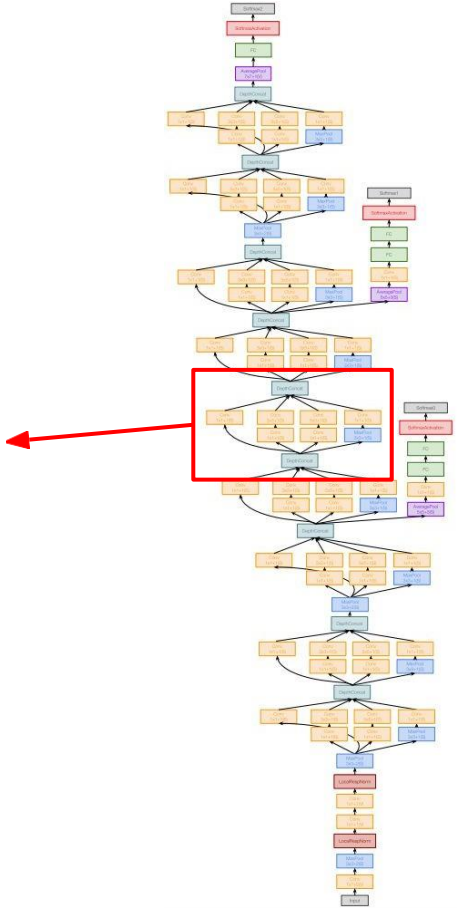
Кейс: GoogLeNet

[Szegedy et al., 2014]

Соберем несколько inception модулей в стек



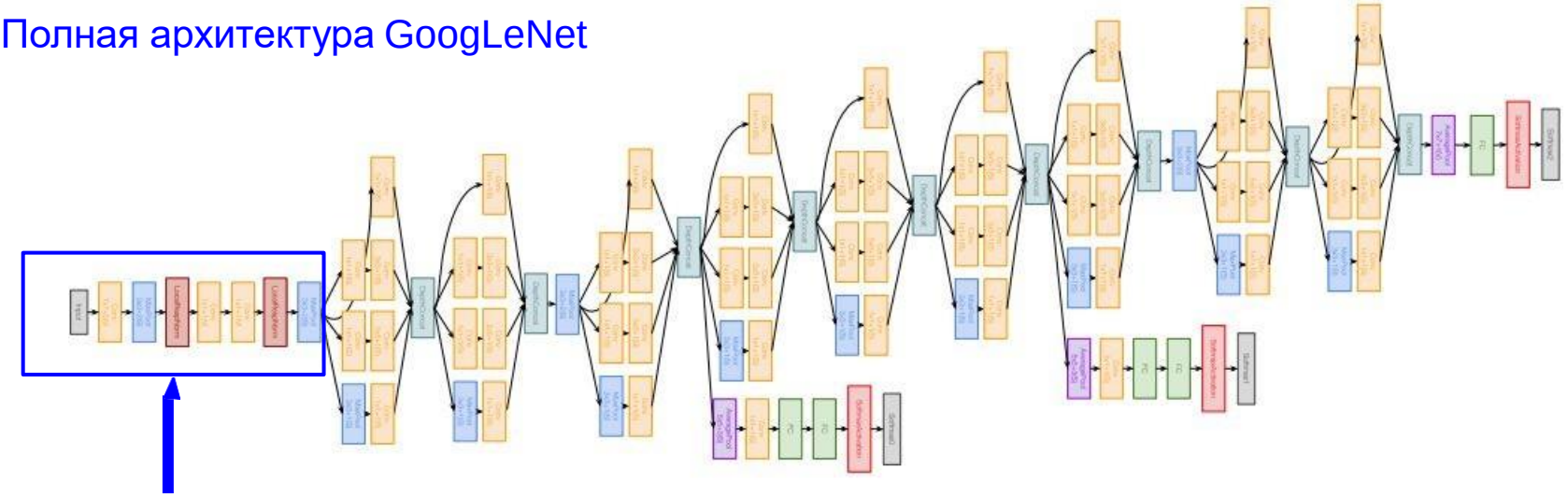
Inception module



Кейс: GoogLeNet

[Szegedy et al., 2014]

Полная архитектура GoogLeNet

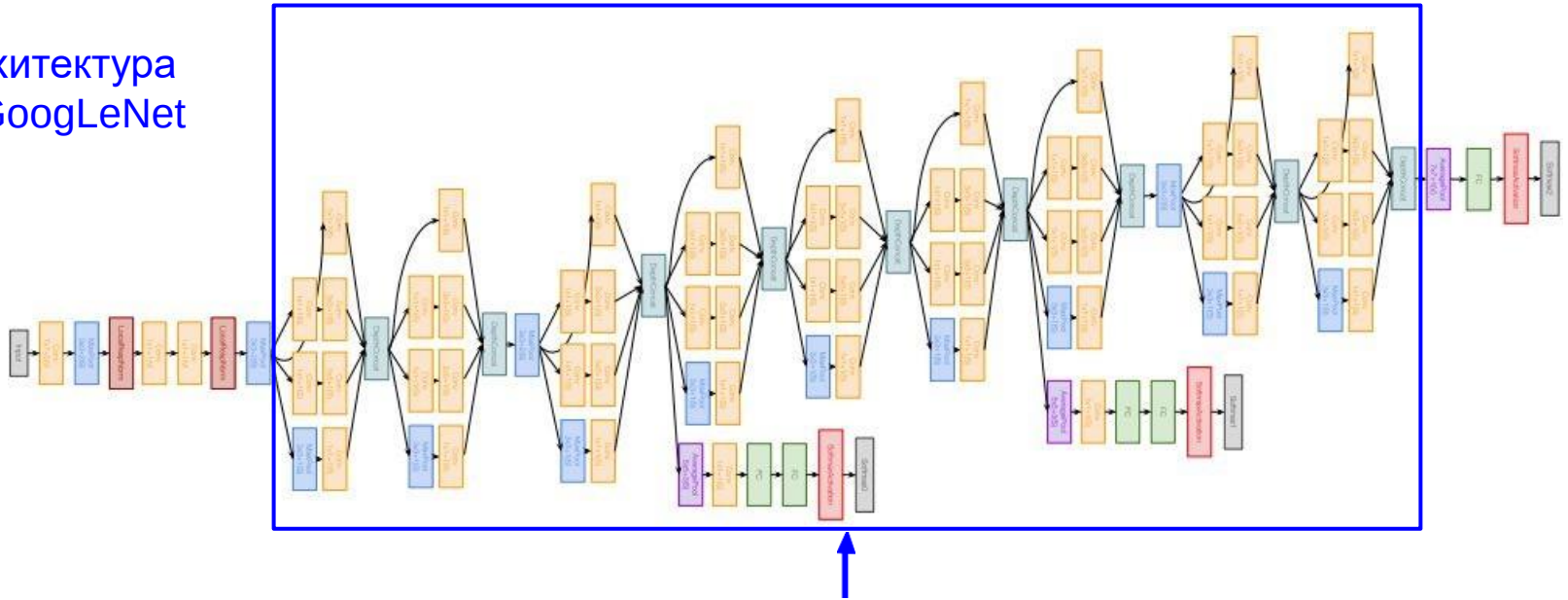


Stem Network:
Conv-Pool-
2x Conv-Pool

Кейс: GoogLeNet

[Szegedy et al., 2014]

Архитектура GoogLeNet

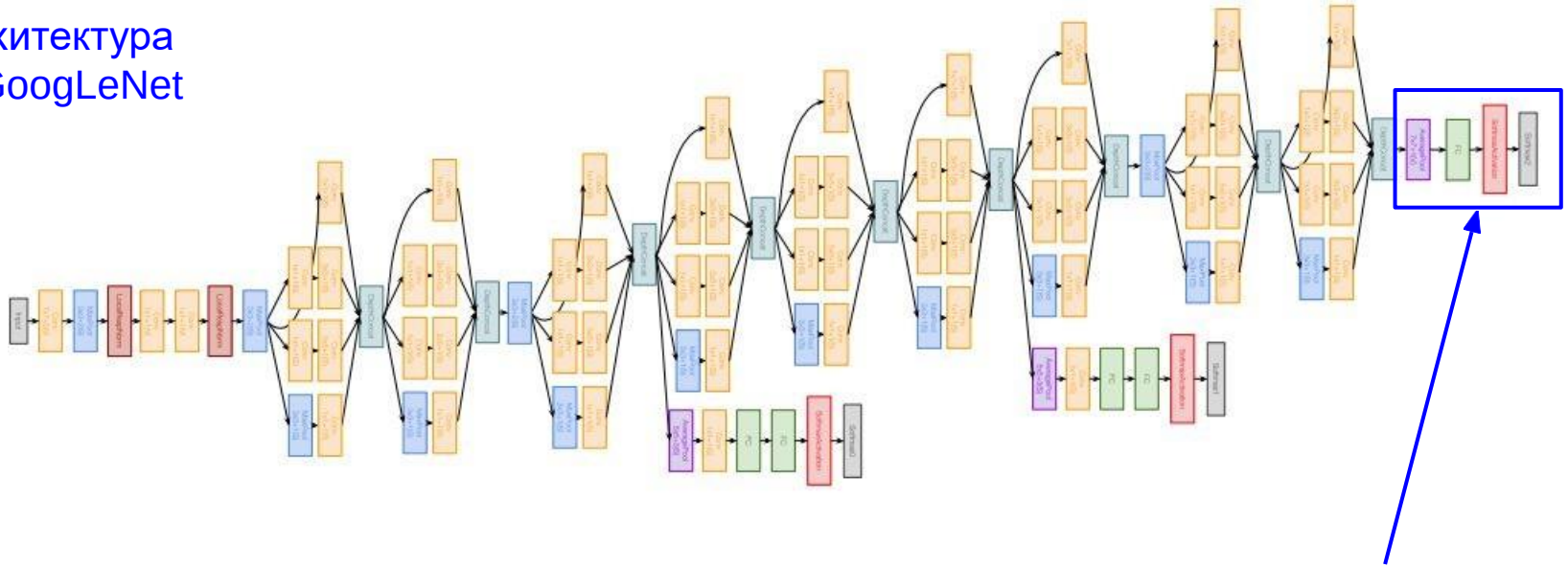


Стек из Inception модулей

Кейс: GoogLeNet

[Szegedy et al., 2014]

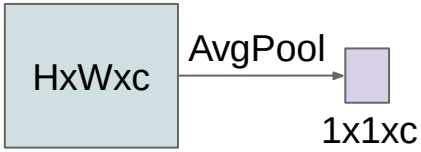
Архитектура
GoogLeNet



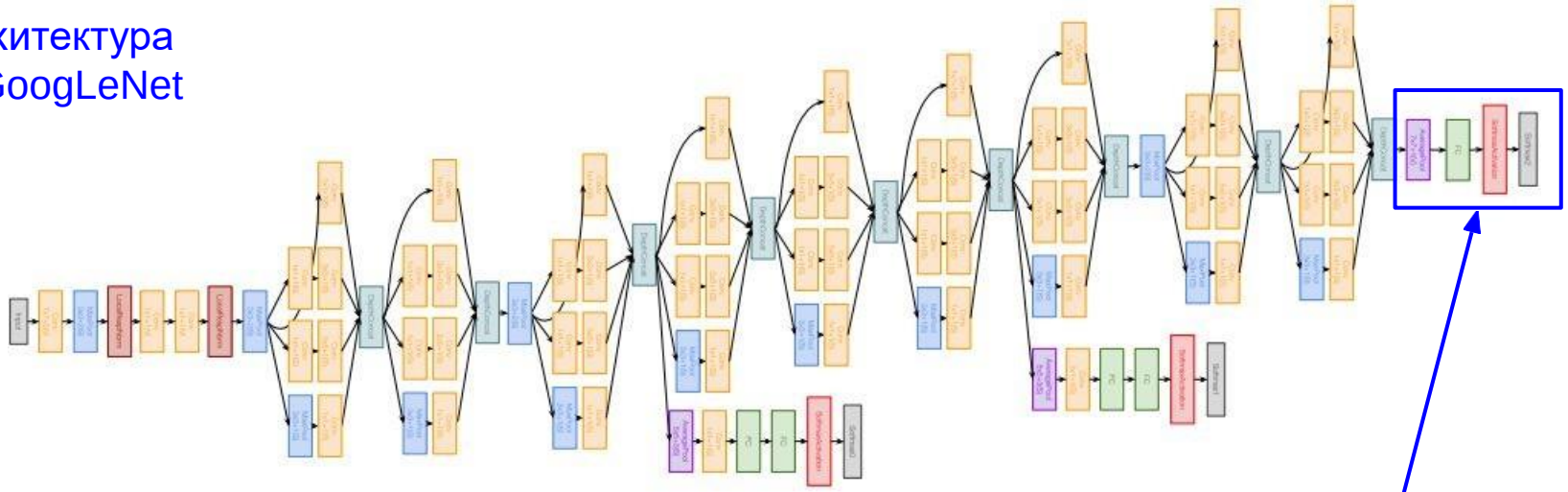
Выход
классификатора

Кейс: GoogLeNet

[Szegedy et al., 2014]



Архитектура GoogLeNet



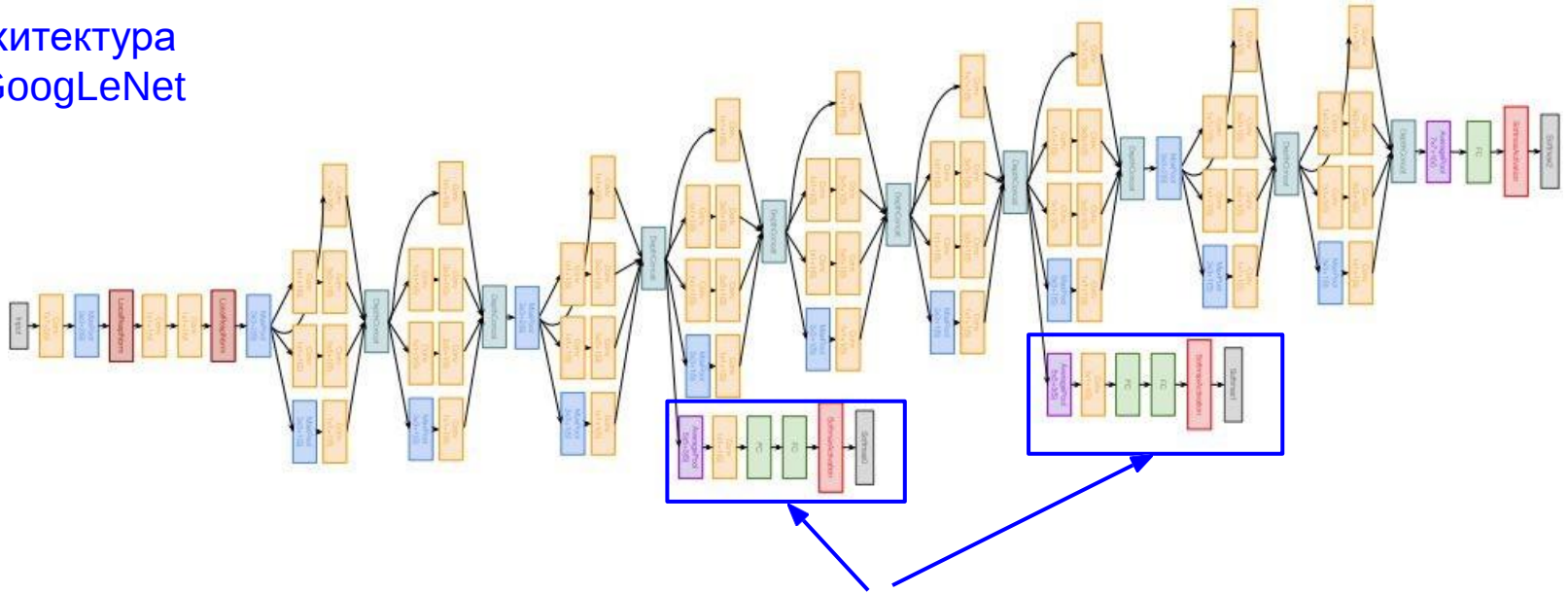
После последнего сверточного слоя, применяем глообальный усредняющий poolng (global average pooling) усредняя по пространству каждый слой активации, перед финальным FC слоем. Никаких стеков из FC слоев!

Выход классификатора

Кейс: GoogLeNet

[Szegedy et al., 2014]

Архитектура
GoogLeNet

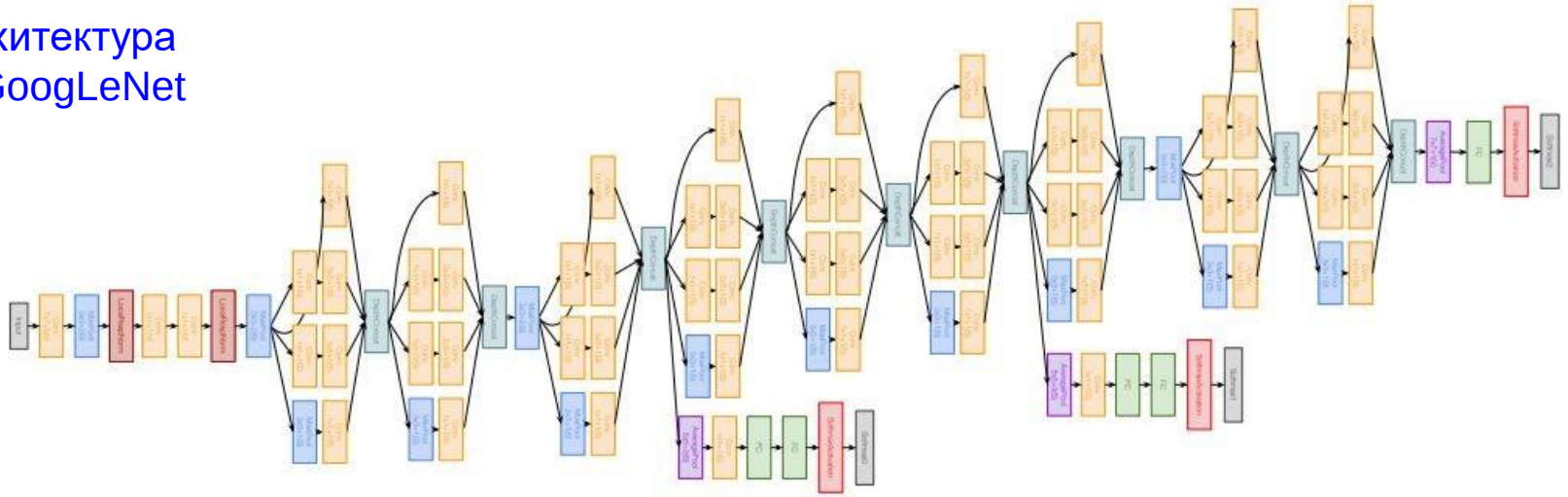


Дополнительные выходы классификаторов чтобы получить дополнительные градиенты на слоях меньшей глубины
AvgPool-1x1Conv-FC-FC-Softmax

Кейс: GoogLeNet

[Szegedy et al., 2014]

Архитектура GoogLeNet



22 слоя с весами

Параллельные фильтры считаем за 1 слой => 2 слоя на Inception модуль.

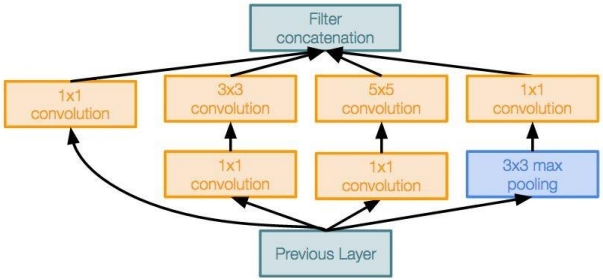
Дополнительные выходы не считаем

Кейс: GoogLeNet

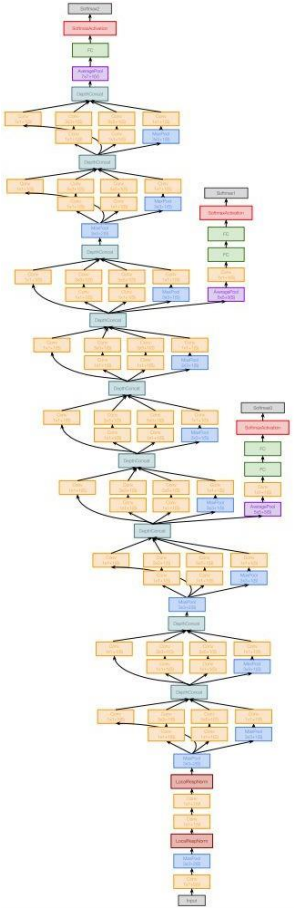
[Szegedy et al., 2014]

Глубокая, вычислительно эффективная CHC

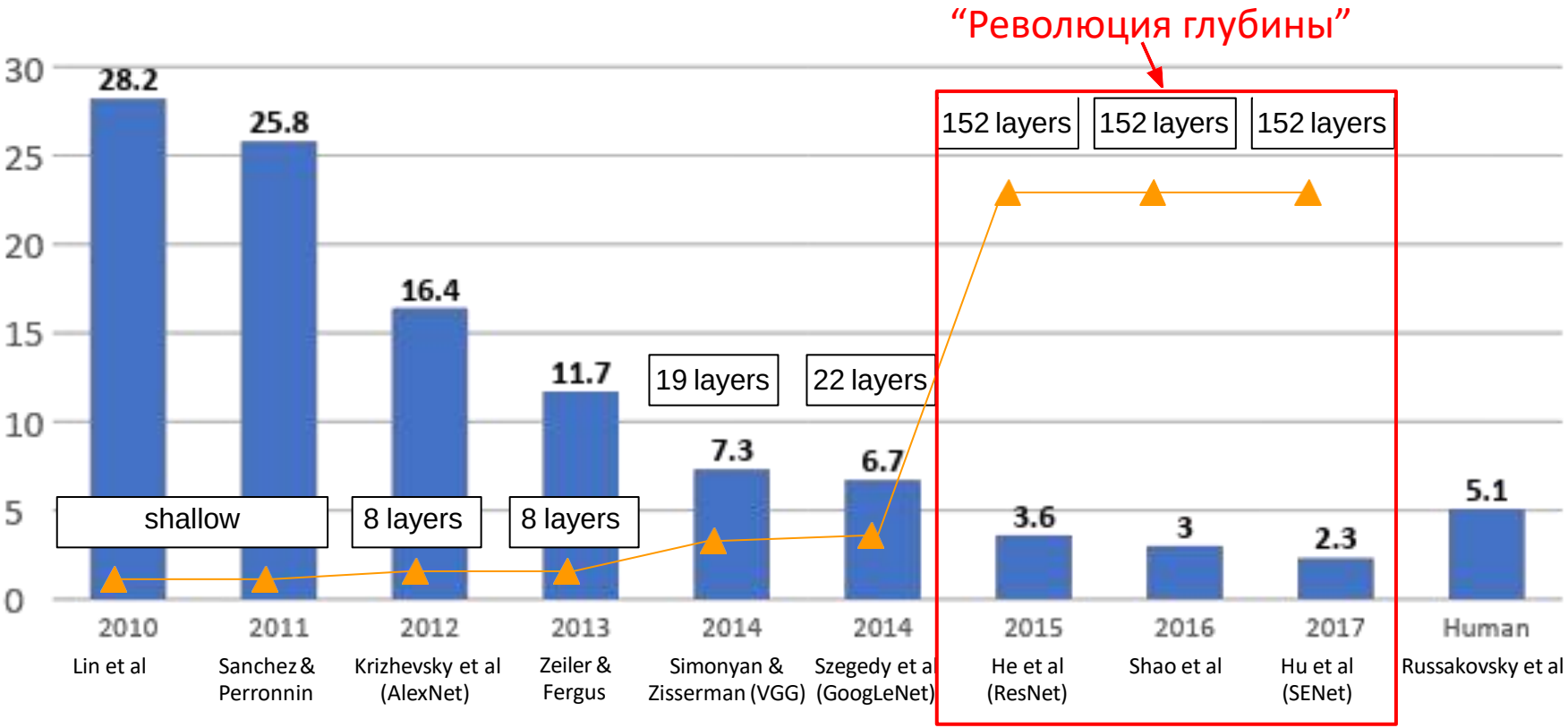
- 22 слоя
- Эффективный “Inception” модуль
- Убираем FC слои
- В 12раз меньше AlexNet
- В 27 раз меньше VGG-16
- ILSVRC’14 победитель (6.7% top 5 error)



Inception module



Победители ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

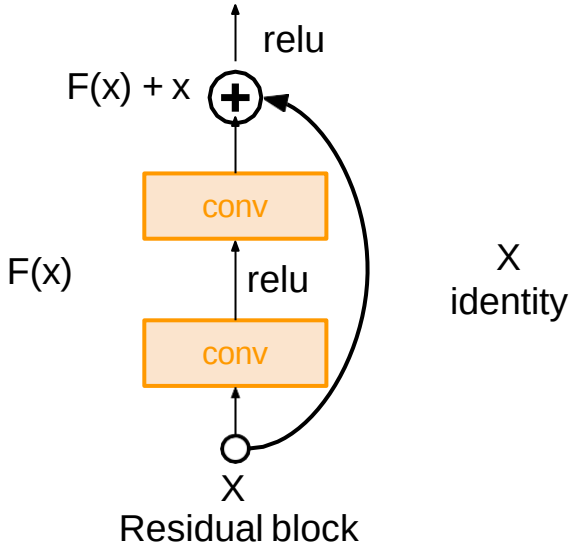


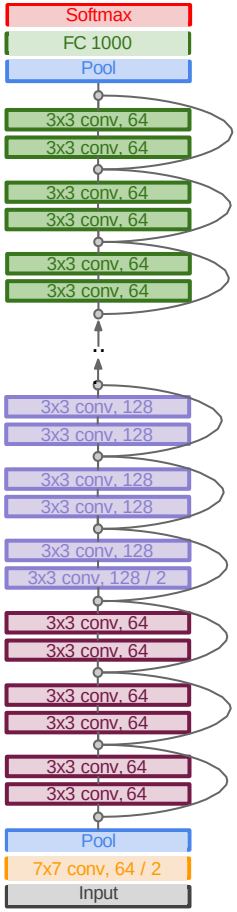
Кейс: ResNet

[He et al., 2015]

Очень глубокая сеть с остаточными связями (residual connections)

- 152-слоев для ImageNet
- ILSVRC'15 победитель (3.57% top 5 error)
- Победил в детекции и классификации в ILSVRC'15 and COCO'15!

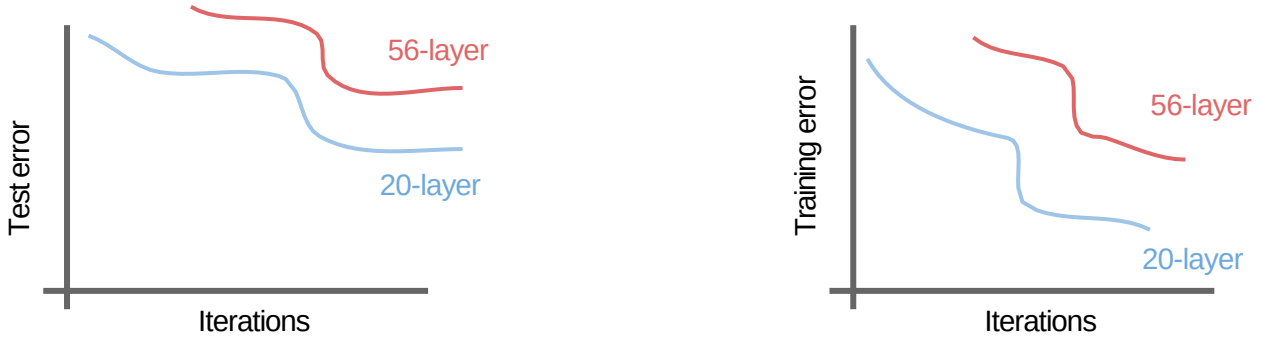




Кейс: ResNet

[He et al., 2015]

Что будет если продолжать увеличивать глубину “плоской” сверточной сети?



56-слое хуже по ошибке обучения и точности
-> глубокая СНС хуже, но у нее нет переобучения!

Case Study: ResNet

[He et al., 2015]

Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an *optimization* problem, **deeper models are harder to optimize**

Кейс: ResNet

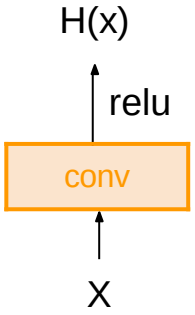
[He et al., 2015]

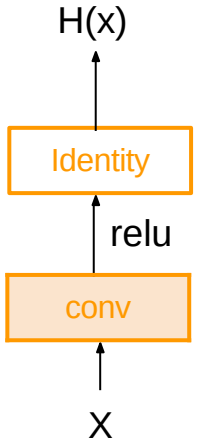
Факт: Глубокая сеть имеет больше способности к аппроксимации, т.к. больше весов.

Гипотеза: проблема в *оптимизации*, глубокие модели хуже сходятся

Чему должна научится более глубокая модель, чтобы как минимум быть не хуже более мелкой?

Решение в копировании обученных слоев и настройка дополнительных слоев для идентичного преобразования!



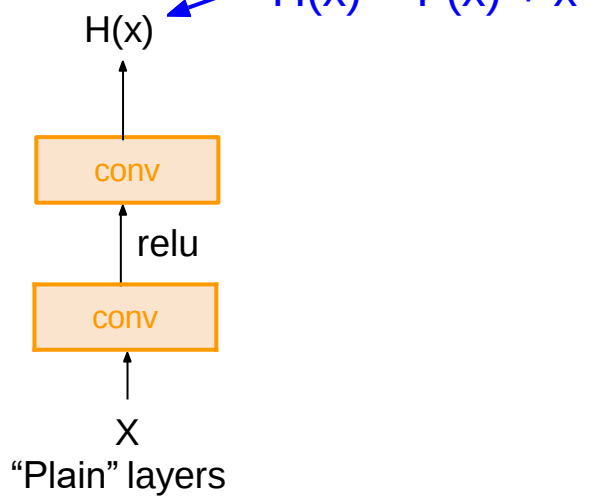


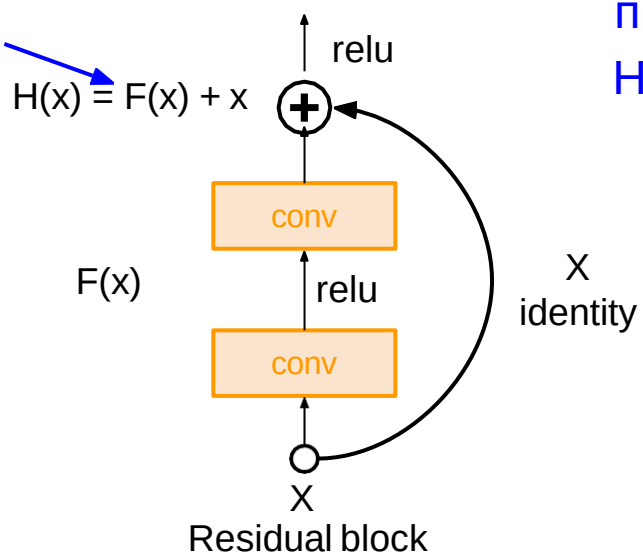
Кейс: ResNet

[He et al., 2015]

Решение: используем слои сети чтобы аппроксимировать остаток/ разницу, вместо того, чтобы аппроксимировать исходную функцию

H(x) = F(x) + x





Идентичное преобразование:

H(x) = x if F(x) = 0

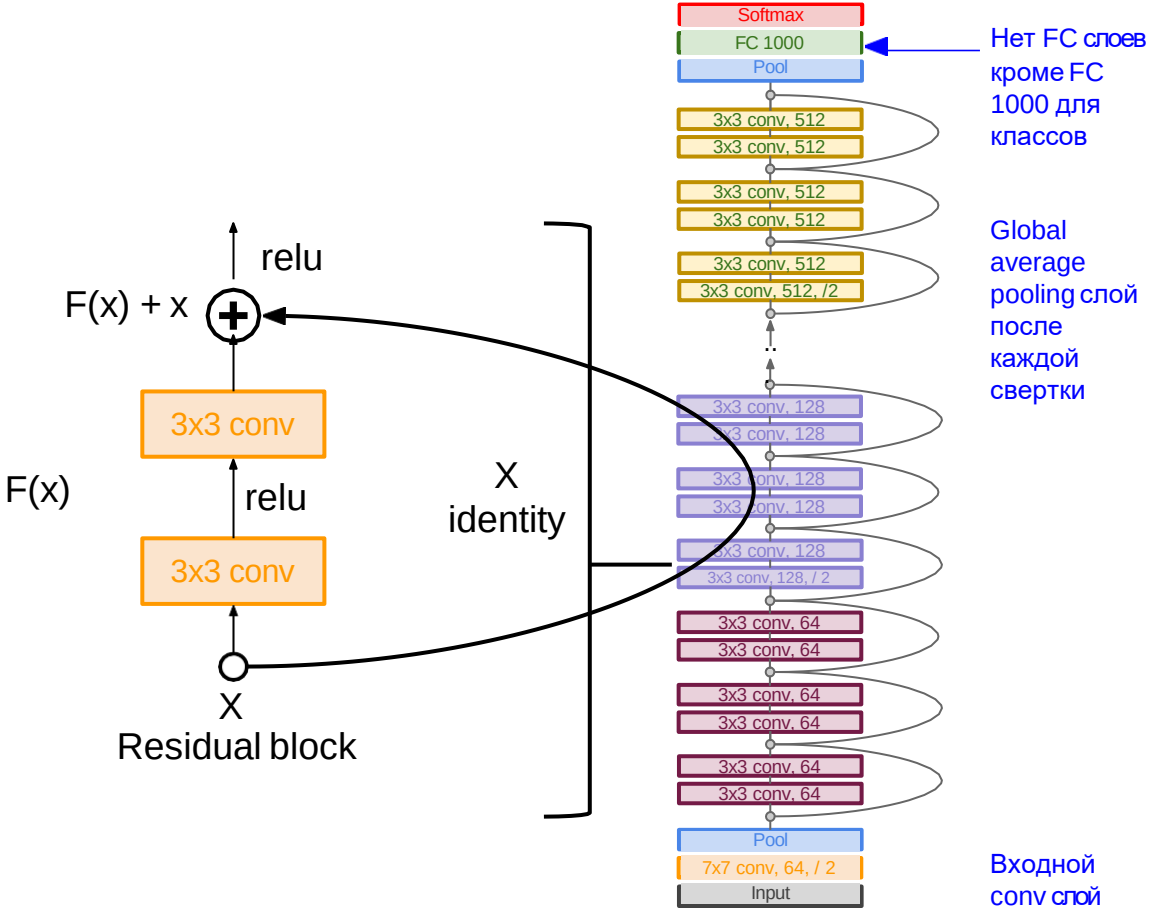
Аппроксимируем F(x) = H(x) - x вместо H(x)

Кейс: ResNet

[He et al., 2015]

Архитектура ResNet:

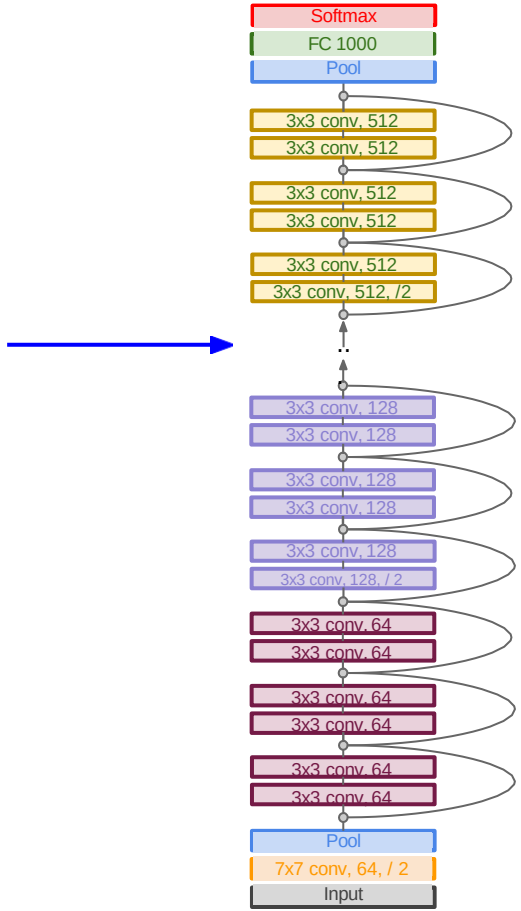
- Стек residual блоков
- Каждый residual блок состоит из 3x3 conv слоев
- Периодически дублируем число фильтров и делаем даунсемплинг с шагом 2 (/2 по каждой размерности)
- Дополнительный conv слой в начале (stem)
- Без FC слоев в конце (только FC 1000 для классов)
- (Теоретически, ResNet работает для любой размерности входной картинки)



Кейс: ResNet

[He et al., 2015]

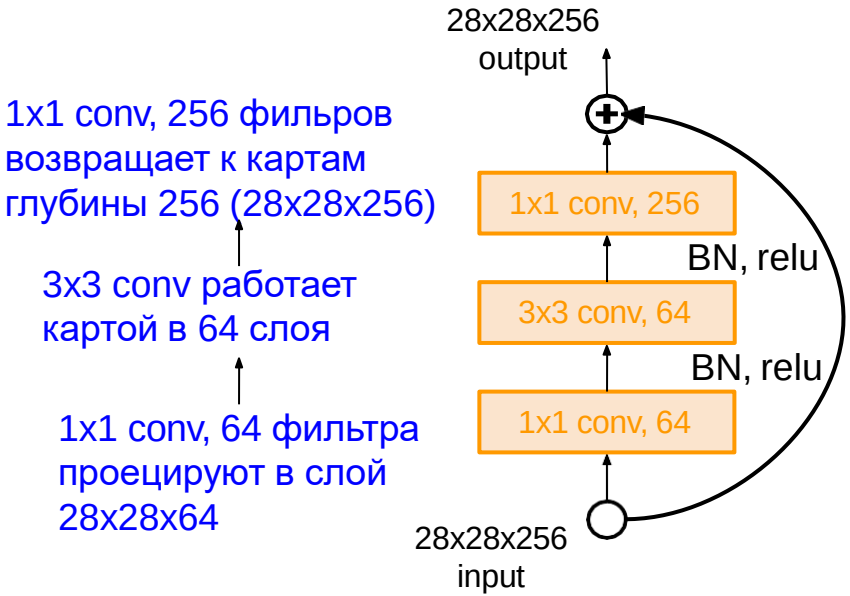
Общая глубина 18, 34, 50, 101, or 152 слоя



Кейс: ResNet

[He et al., 2015]

Для глубоких случаев (ResNet-50+), используем “bottleneck” слой для устойчивости (как в GoogLeNet)



Кейс: ResNet

[He et al., 2015]

Обучение ResNet на практике:

- Batch Normalization после каждого CONV слоя
- Xavier инициализация из He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, делим на 10 на каждом плато ошибки
- Mini-batch size 256
- Weight decay 1e-5
- Нет dropout!

Кейс: ResNet

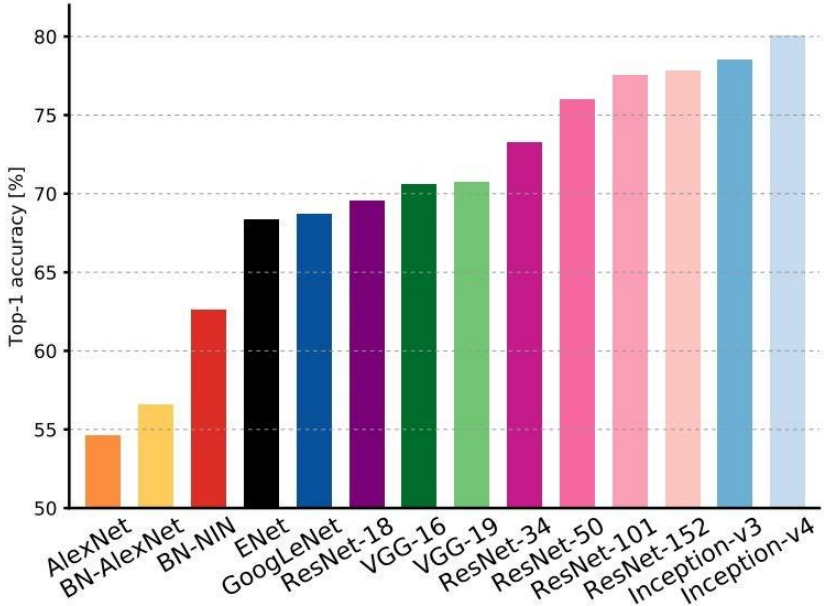
[He et al., 2015]

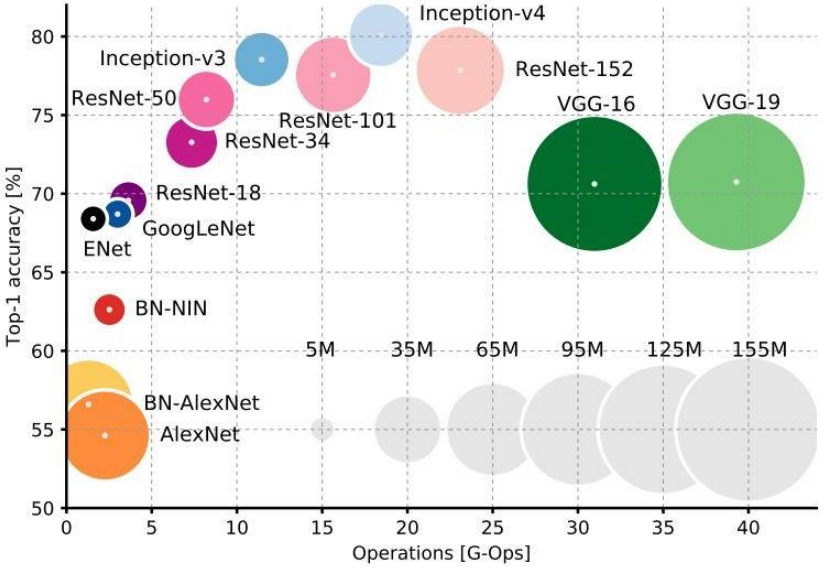
Результаты экспериментов

- Обучаем **очень глубокие** сети без деградации (152 слоев для ImageNet, 1202 для Cifar)
- Теперь глубокие сети могут получать очень маленькую ошибку
- Первое место во всех соревнованиях ILSVRC и COCO 2015

ILSVRC 2015 победитель (3.6% top 5 error) -- **" лучше человека"**! (Russakovsky 2014)

Сравним сложность...

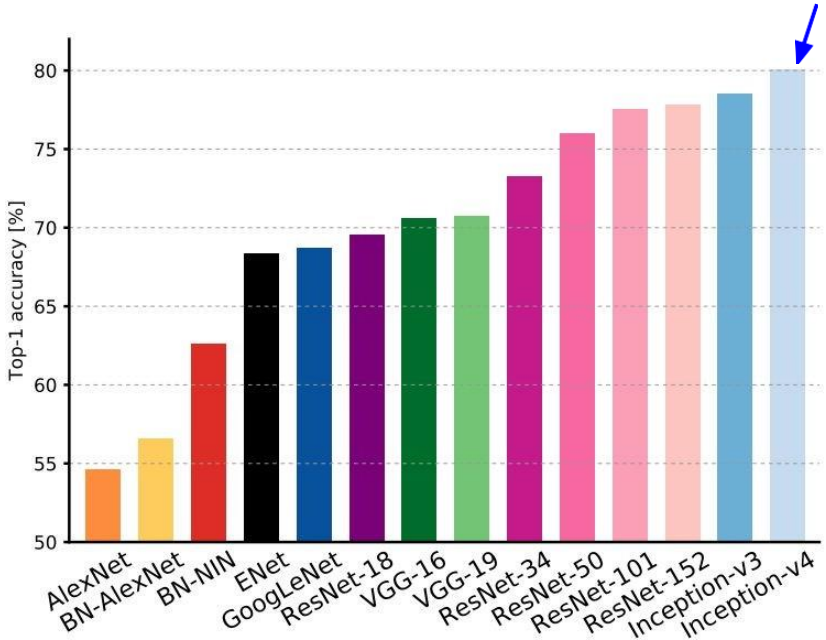


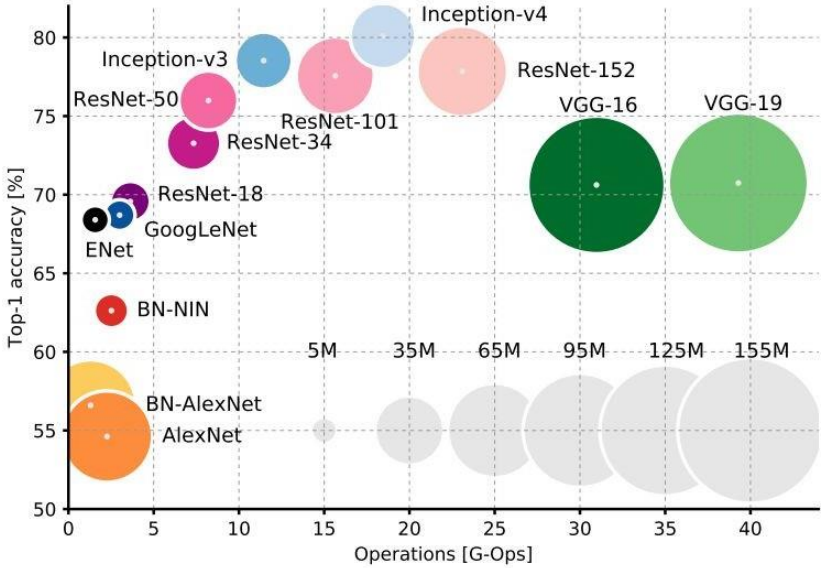


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Сравним сложность...

Inception-v4: Resnet + Inception!

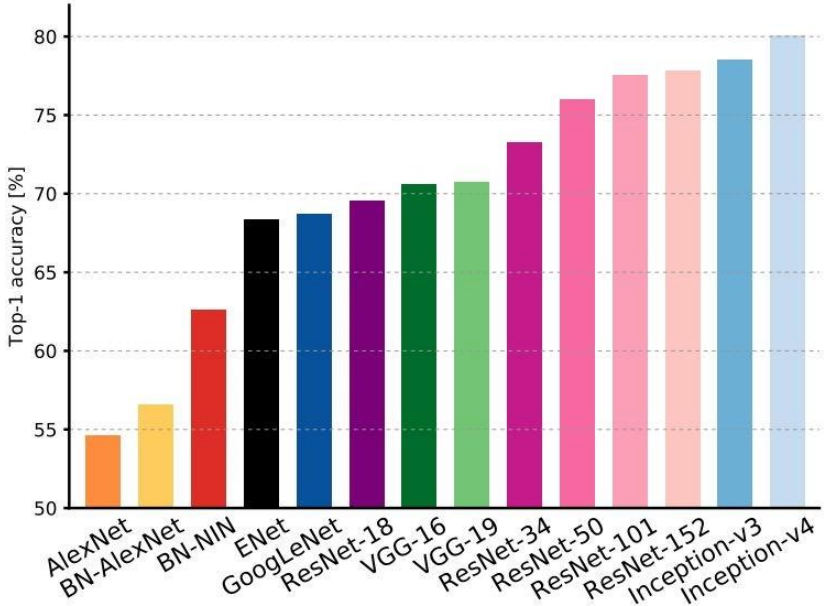


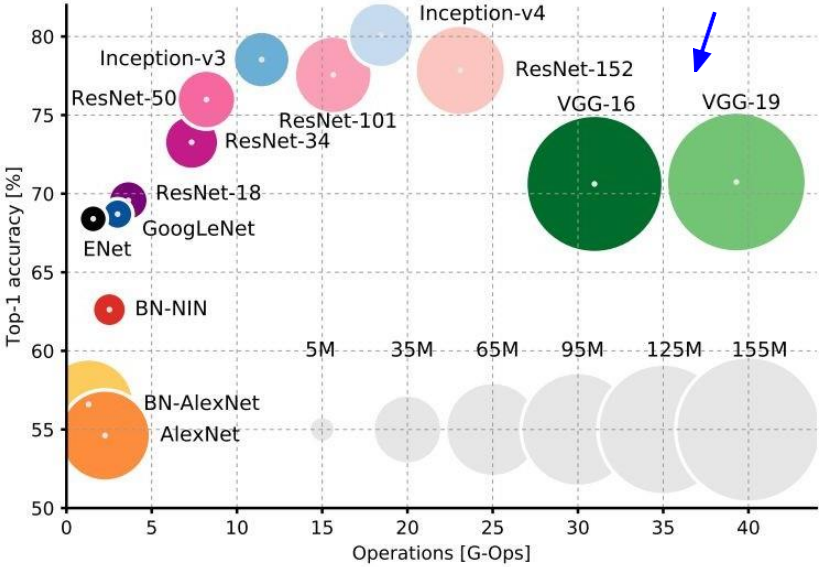


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Сравним сложность...

VGG: самая сложная и большая

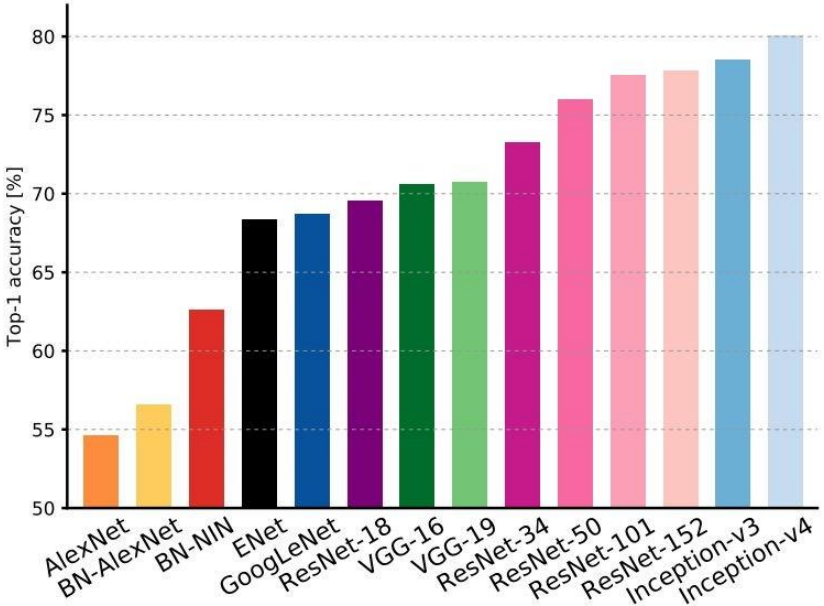


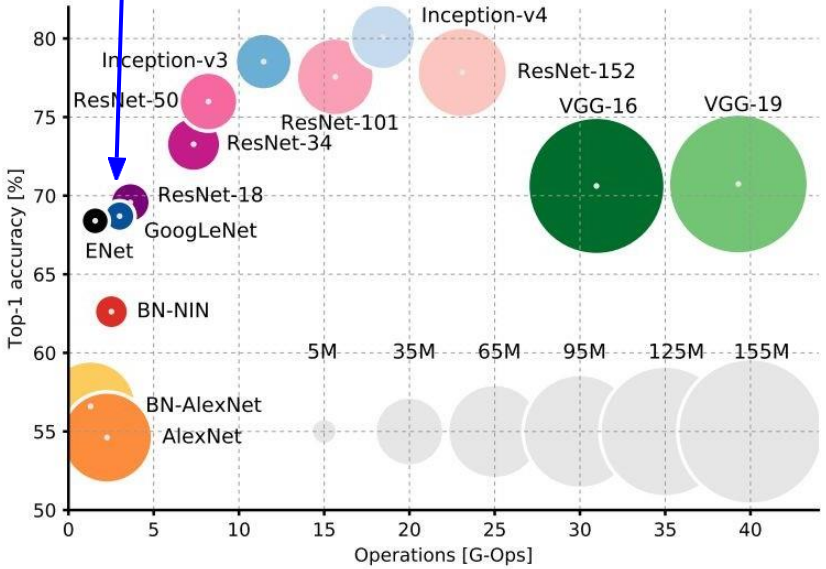


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Сравним сложность...

GoogLeNet: самая эффективная

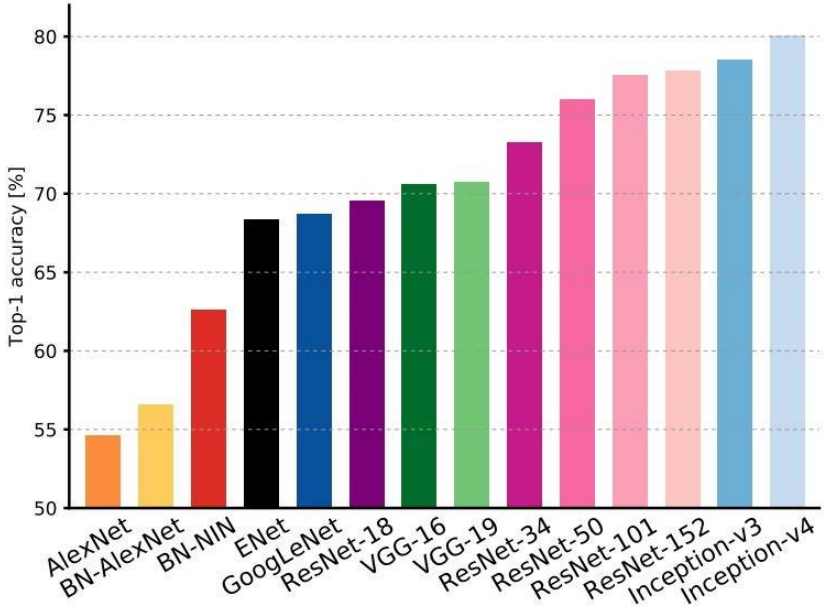


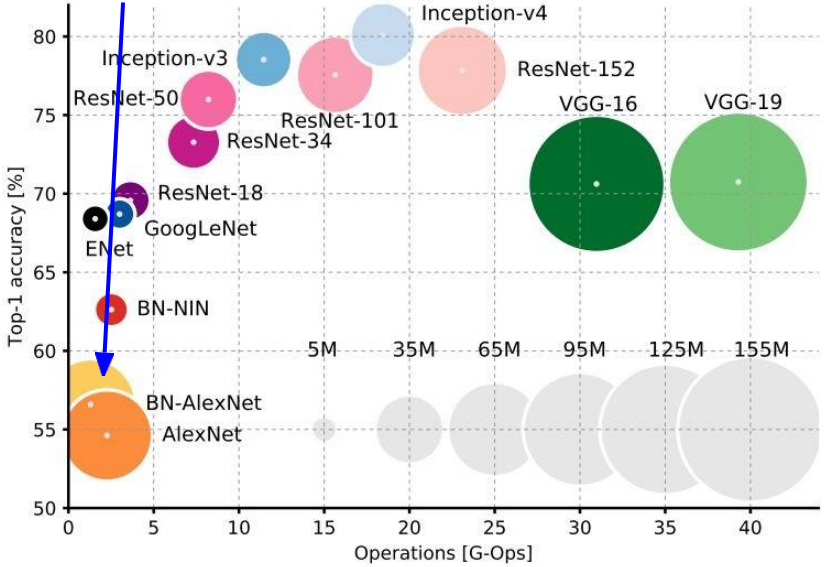


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Сравним сложность...

AlexNet:
Простая, большая, плохая точность

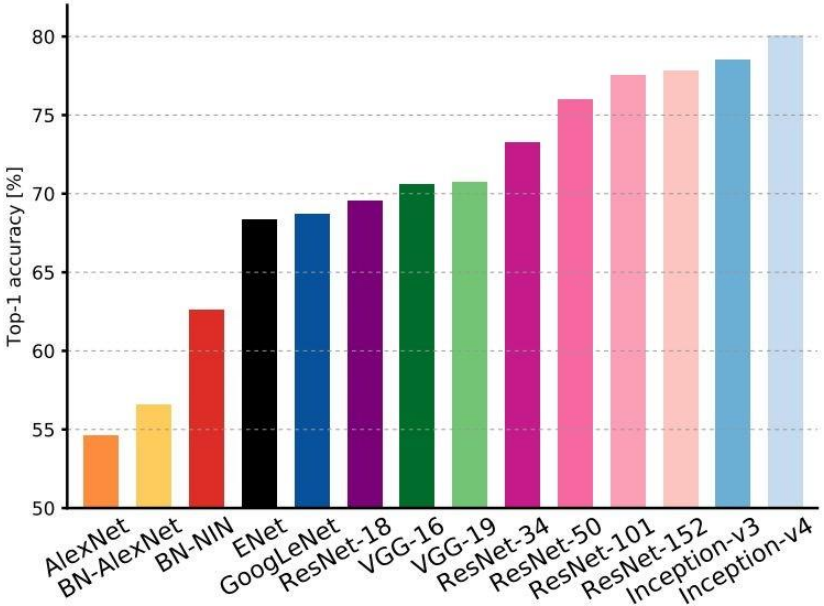


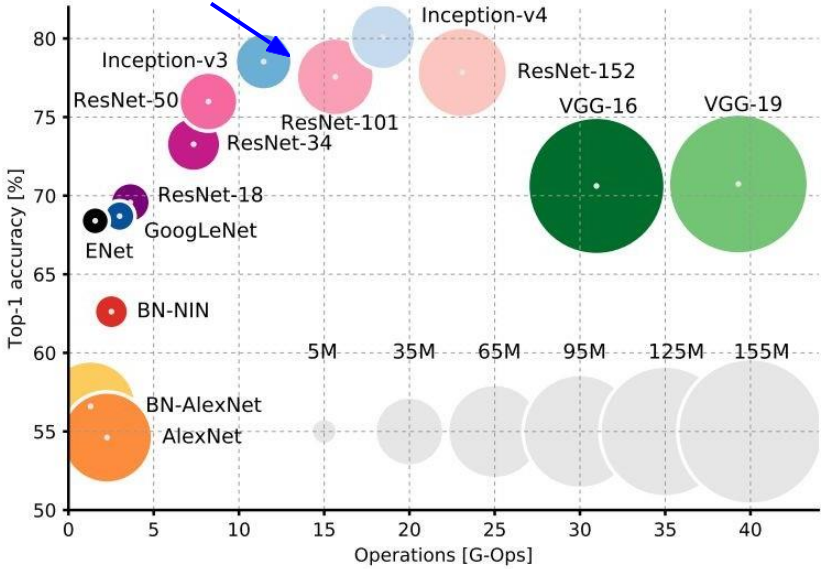


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Сравним сложность...

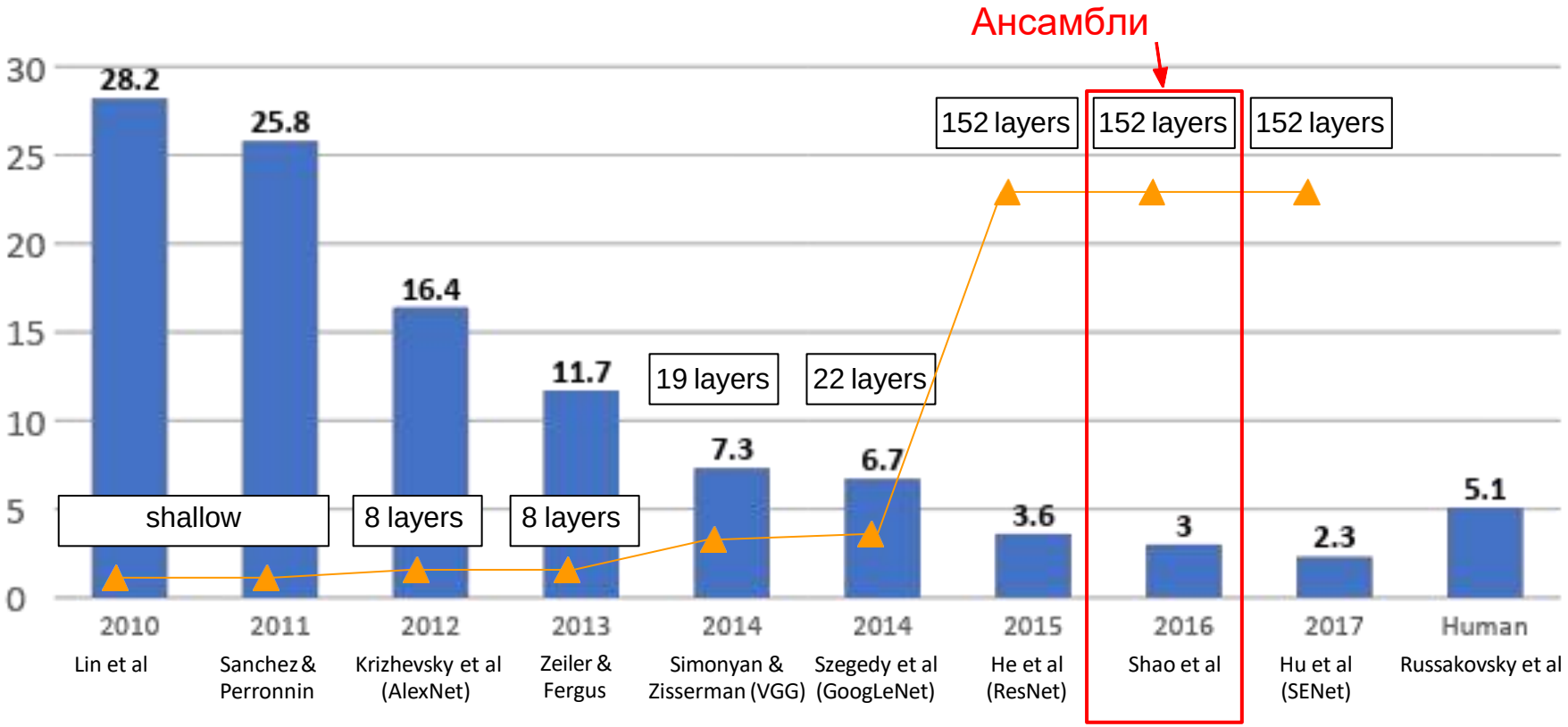
ResNet:
Средняя сложность,
максимальная точность





An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Победители ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



Улучшим ResNets...

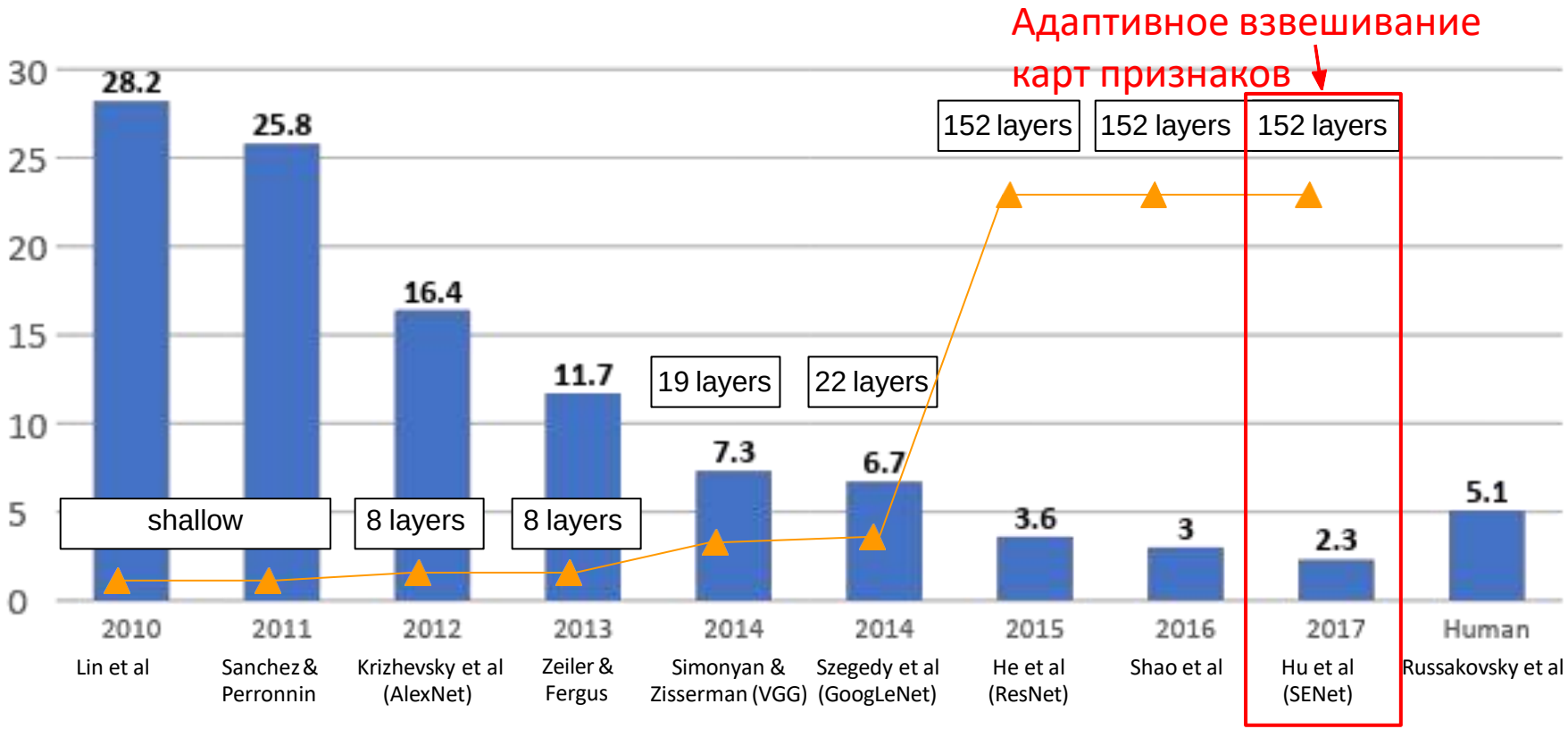
“Good Practices for Deep Feature Fusion”

[Shao et al. 2016]

- Мультимасштабный ансамбль Inception, Inception-Resnet, Resnet, Wide Resnet моделей
- ILSVRC’16 победитель

	Inception-v3	Inception-v4	Inception-Resnet-v2	Resnet-200	Wrn-68-3	Fusion（Val.）	Fusion（Test）
Err. (%)	4.20	4.01	3.52	4.26	4.65	2.92 (-0.6)	2.99

Победители ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

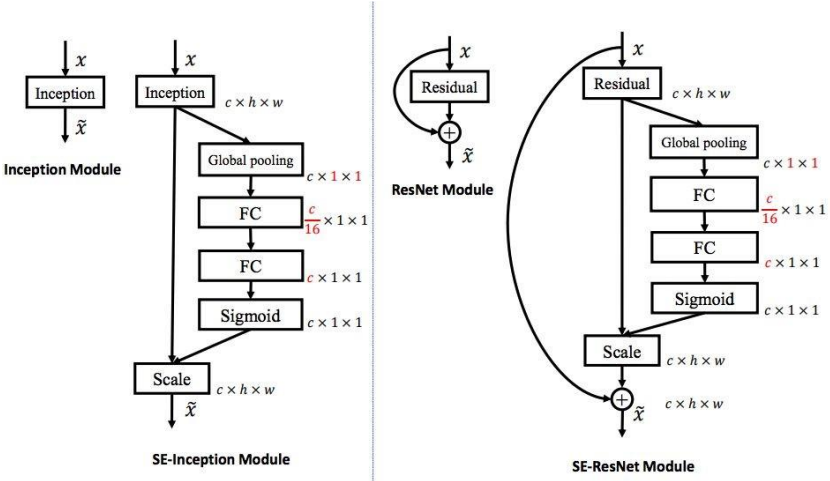


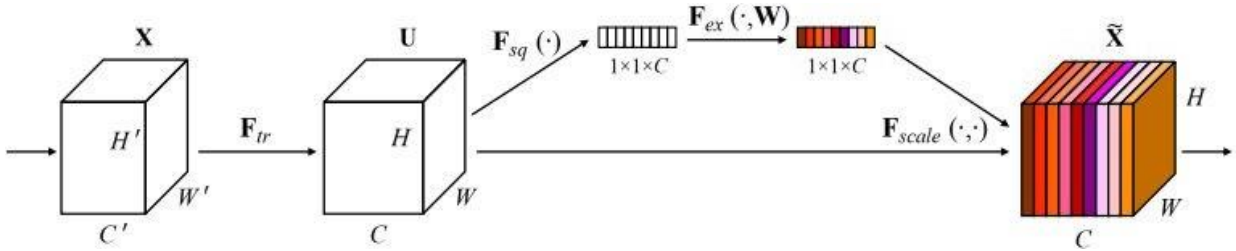
Улучшим ResNets...

Squeeze-and-Excitation Networks (SENet)

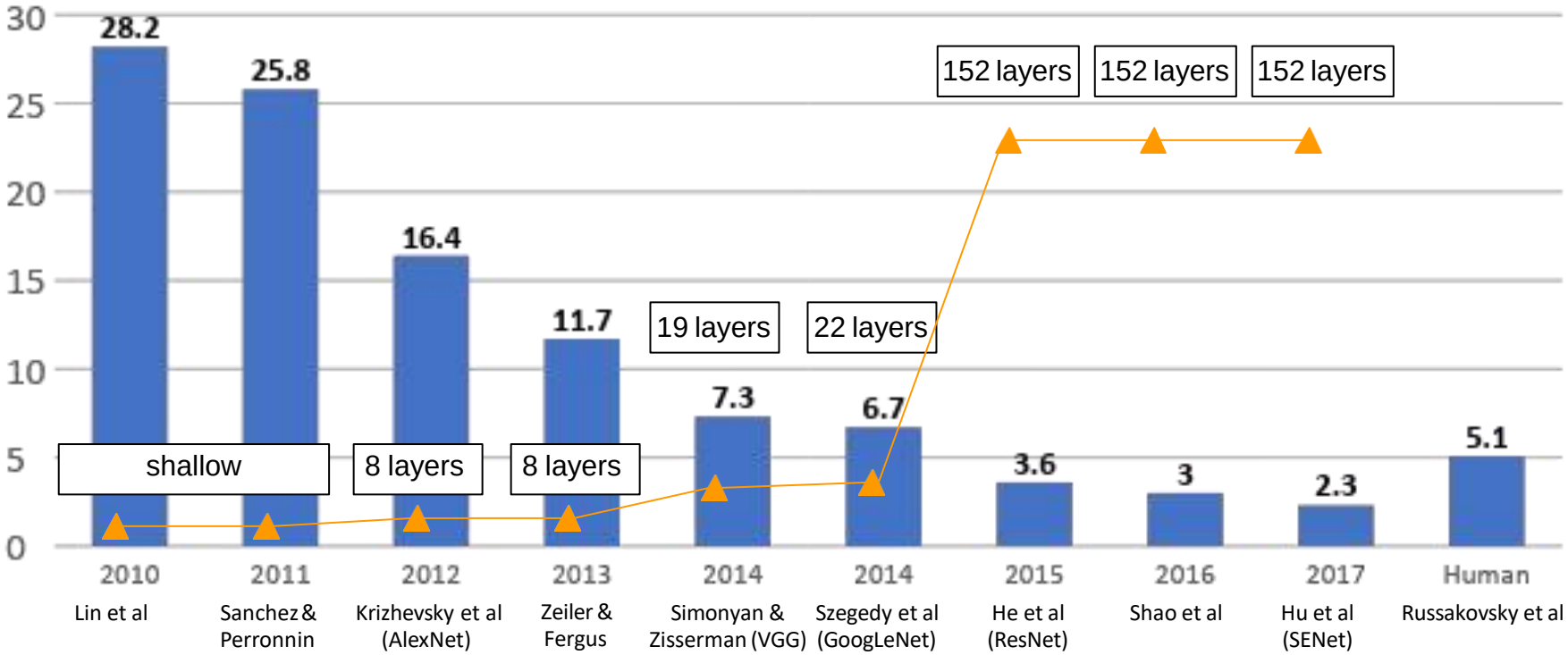
[Hu et al. 2017]

- Добавим модуль “калибровки признаков” который учится адаптивно взвешивать карты признаков
- Глобальная информация (global avg. pooling layer) + 2 FC слоя, чтобы определить веса признаков
- ILSVRC’17 победитель (на основе архитектуры ResNeXt-152)

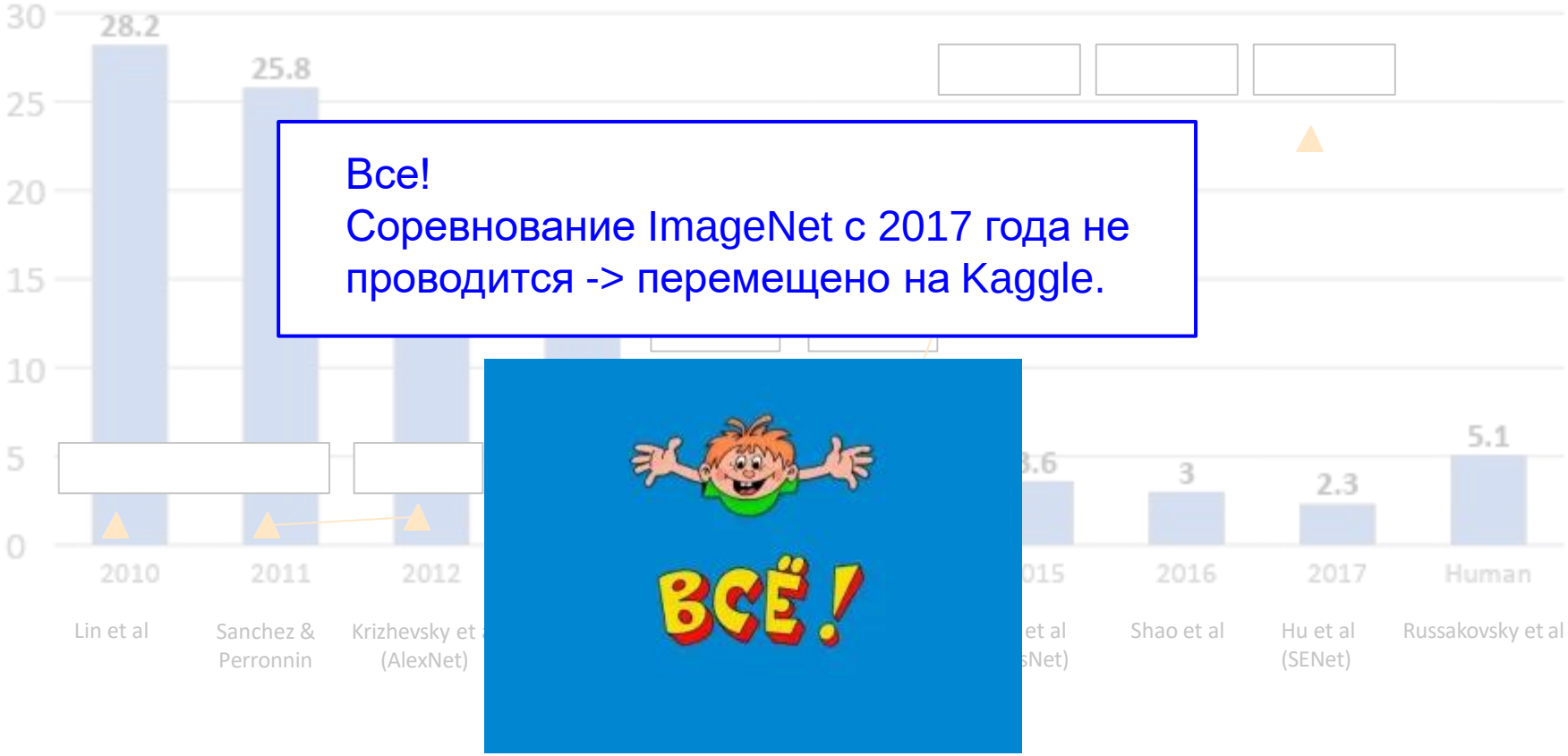




Победители ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



Победители ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



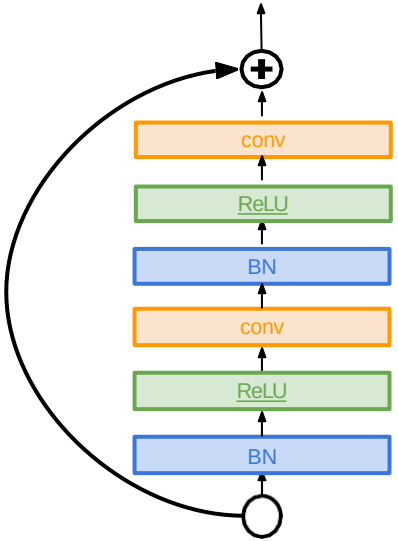
Дальнейшее развитие архитектур CHC

Улучшим ResNets...

Identity Mappings in Deep Residual Networks

[He et al. 2016]

- Улучшение ResNet блока от создателей ResNet
- Более прямой путь для распространения по сети
- Лучшая точность

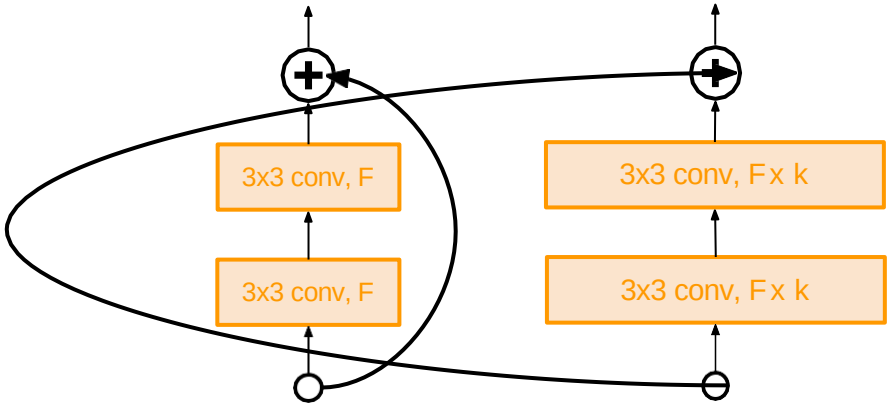


Улучшим ResNets...

Широкие Residual сети

[Zagoruyko et al. 2016]

- Остаток важнее глубины
- Широкие residual blocks (F x k фильтров вместо F)
- 50-слойная wide ResNet точнее ResNet-152
- Увеличение ширины выгоднее увеличения глубины, т.к. параллельно по данным



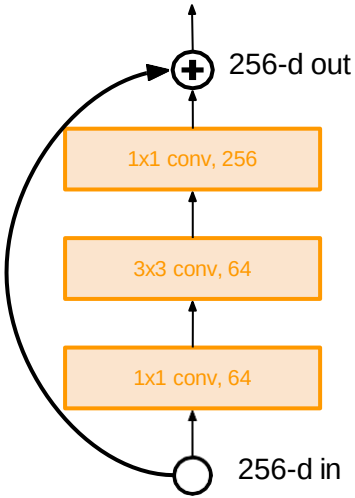
Basic residual block Wide residual block

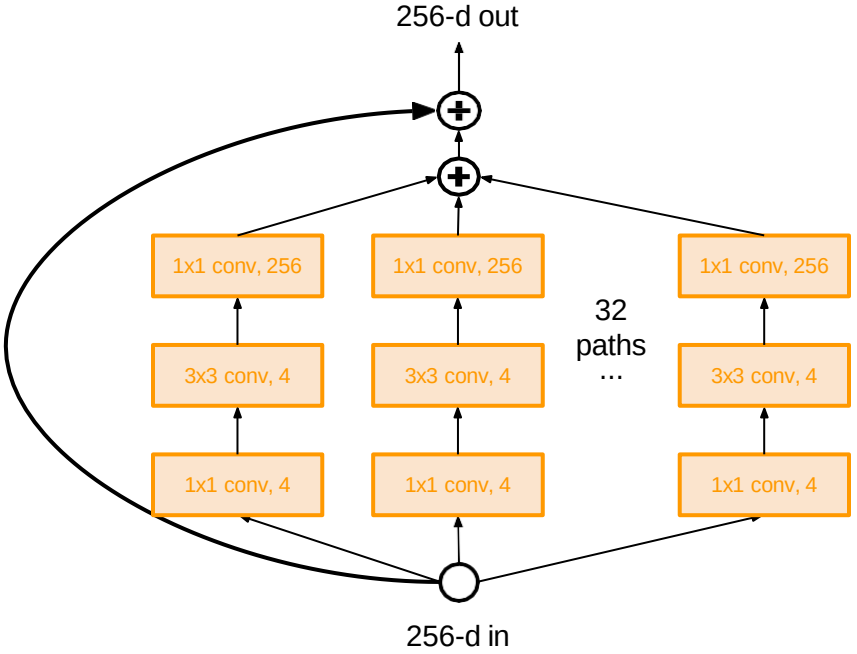
Улучшаем ResNets...

Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

[Xie et al. 2016]

- Также от создателей ResNet
- Увеличение ширины residual block за счет нескольких проходов (“кардинальность/ мощность”)
- Параллельность в стиле Inception



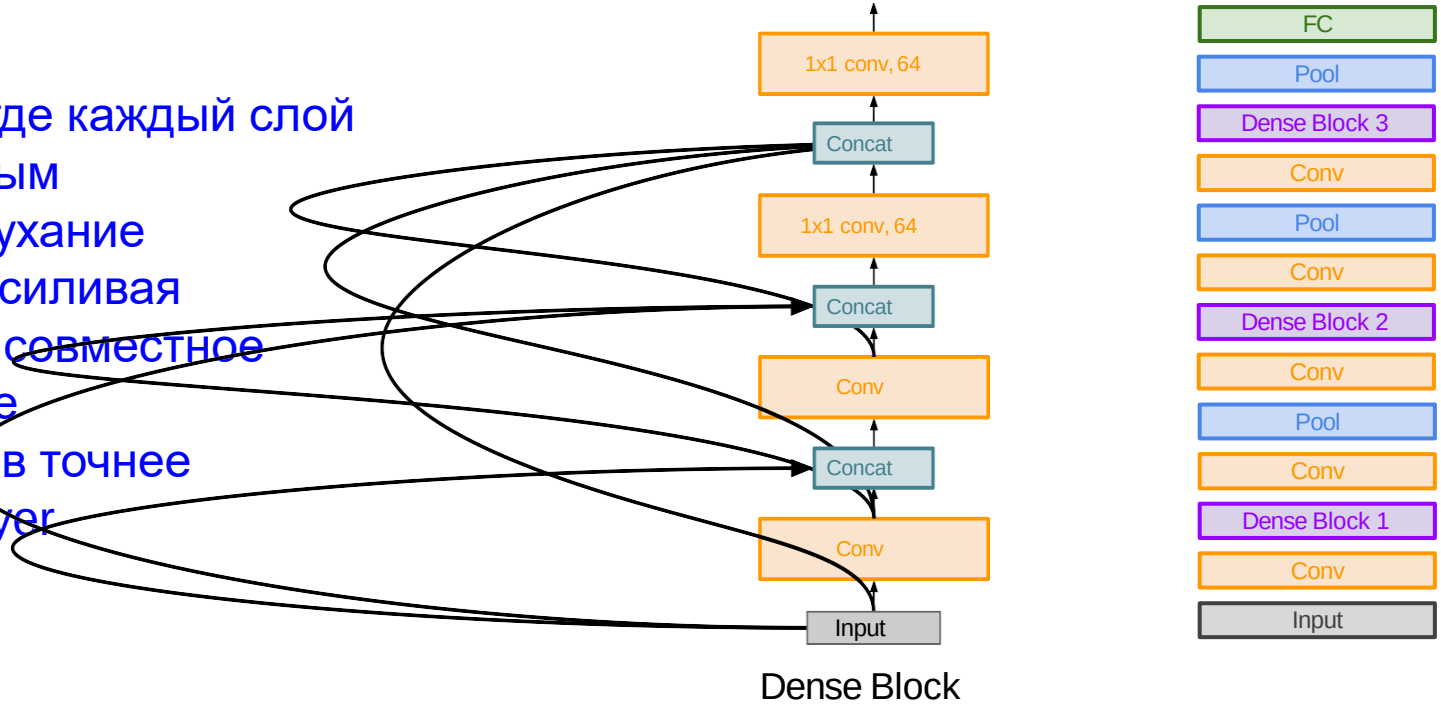


Еще идеи...

Densely Connected Convolutional Networks (DenseNet)

[Huang et al. 2017]

- Dense blocks где каждый слой связан с каждым
- Устраняет затухание градиентов, усиливая признаки и их совместное использование
- Сеть в 50слоев точнее ResNet-152 layer

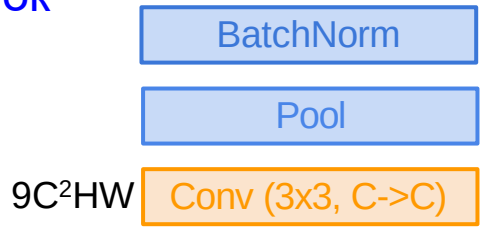


Эффективные сети...

MobileNets: Efficient Convolutional Neural Networks for Mobile Applications *[Howard et al. 2017]*

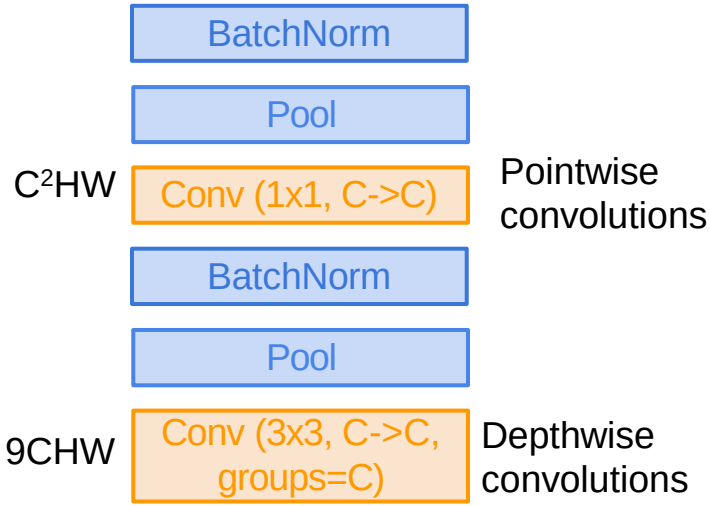
- Свертки в глубину вместо обычных на основе факторизации 1x1 сверток и «глубинных» сверток

- Более эффективные с меньшей точностью
- MobileNetV2 в 2018 (Sandler et al.)
- ShuffleNet: Zhang et al, CVPR 2018



Standard network

Total compute:$9C^2HW$



MobileNets

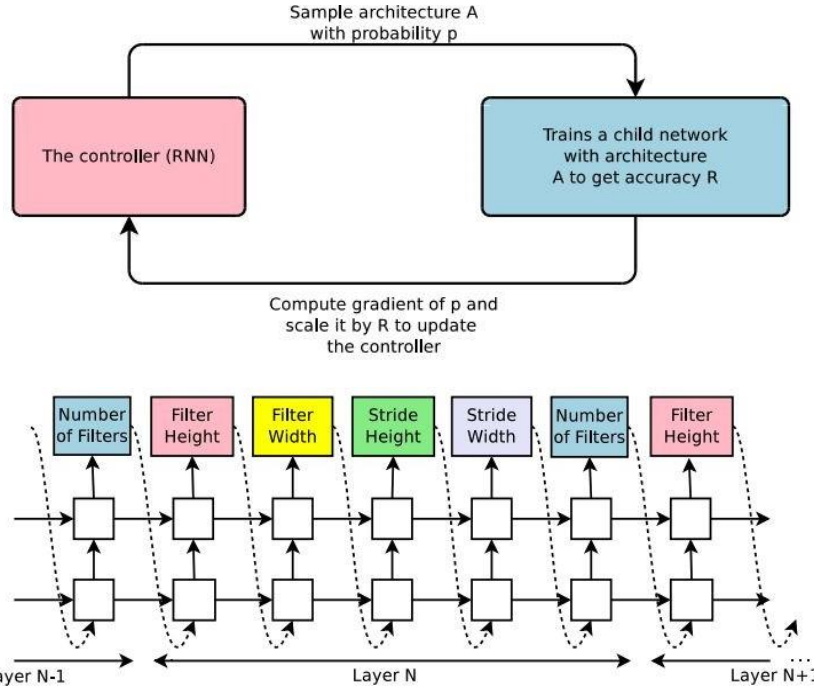
Total compute:$9CHW + C^2HW$

Обучаемый поиск архитектур...

Neural Architecture Search with Reinforcement Learning (NAS)

[Zoph et al. 2016]

- Сеть “контроллер” которая учится искать архитектуру (выход строка, кодирующая архитектуру)
- Итерируем:
  1) Выбираем архитектуру из выборочного распределения
  2) Учим сеть этой архитектуры чтобы получить “вознаграждение” R на основе точности
  3) Считаем градиент распределения, взвешиваем с R и обновляем параметры т.е., увеличиваем правдоподобие для хорошей архитектуры, уменьшаем для плохой)

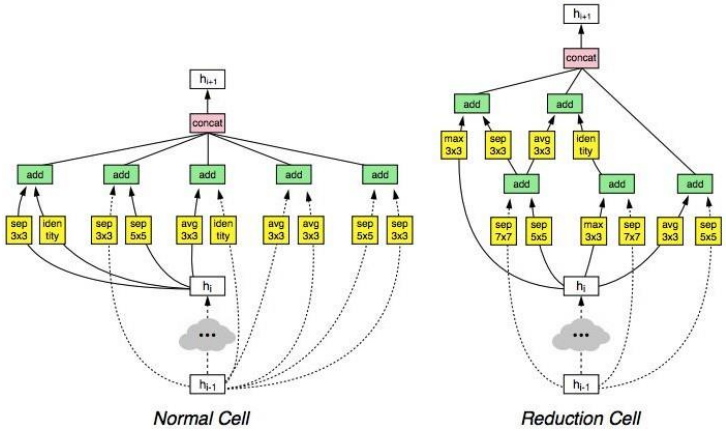


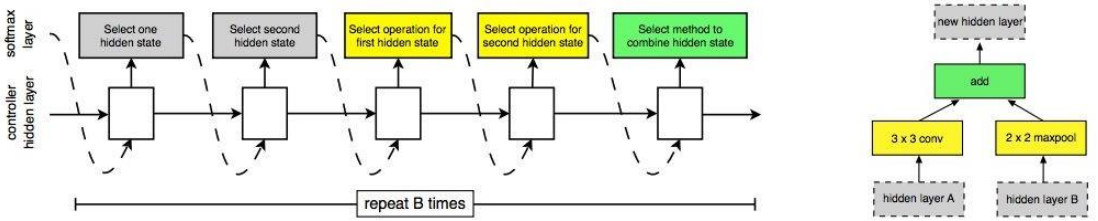
Обучаемый поиск архитектур…

Learning Transferable Architectures for Scalable Image Recognition

[Zoph et al. 2017]

- Применение обучаемого поиска (NAS) к ImageNet очень затратно
- Определяем пространство поиска из ячеек (“cells”) которые можно стековать
- NASNet: На основе NAS ищем лучшую структуру ячейки на CIFAR-10, переносим на ImageNet
- Много работ для этого подхода, AmoebaNet (Real et al. 2019) and ENAS (Pham, Guan et al. 2018)





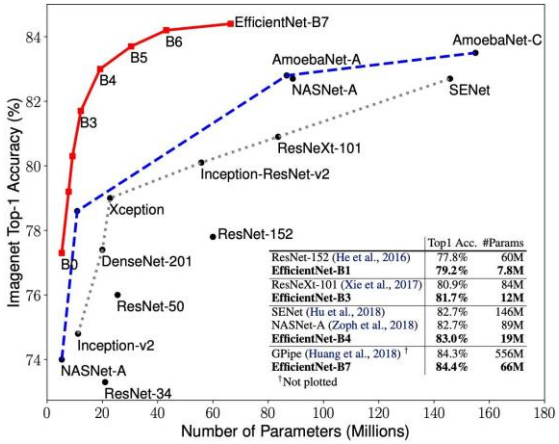
Но эвристика пока побеждает NAS ...

EfficientNet: Smart Compound Scaling

[Tan and Le. 2019]

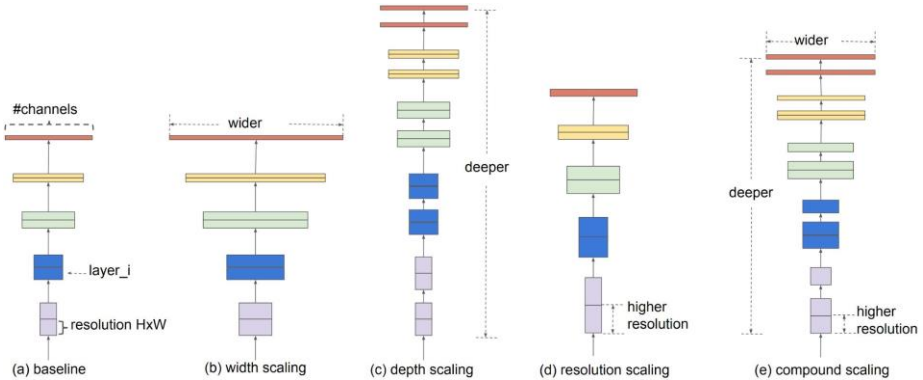
- Увеличим емкость сети масштабируя ширину, глубину, и разрешение, сохраняя баланс точности и эффективности.
- Ищем оптимальные масштабирующие факторы для заданного бюджета памяти и флопс.
- Масштабируем на основе хитрой эвристики

$$\text{depth: } d = \alpha^\phi$$
$$\text{width: } w = \beta^\phi$$
$$\text{resolution: } r = \gamma^\phi$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.2%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.7%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**83.0%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted



Efficient networks...

https://openai.com/blog/ai-and-efficiency/

Архитектуры CHC - списком

Кейсы

- AlexNet
- VGG
- GoogLeNet
- ResNet

А также....

- SENet
- Wide ResNet
- ResNeXT
- DenseNet
- MobileNets
- NASNet

Что надо запомнить по архитектурам

AlexNet утвердило победу CHC в компьютерном зрении.
ZFNet, **VGG** показали что большие модели лучше,
GoogLeNet впервые сфокусировались на эффективности, на основе 1x1 conv и global avg pooling вместо FCслоев
ResNet показал что можно учить очень глубокие сети

- Ограничение только в GPU & памяти!
- Показала что с увеличением сети прирост точности снижается

После ResNet: CHC уже стали лучше людей, и фокус сместился в сторону Эффективных сетей:

- Множество сетей для мобильных устройств: **MobileNet**, **ShuffleNet**
- **Neural Architecture Search** автоматизирует поиск
- Но EfficientNet все равно лучшая!

Архитектуры CHC: Итоги

- Основные популярные модели есть на github и в model zoos.
- ResNets/EfficientNet на сегодня выбор по умолчанию.
- Сети становятся все глубже и глубже.
- Остальные аспекты тоже улучшаются постоянно.