

CNN architectures

Stanford University

- <https://www.youtube.com/watch?v=DAOcjcFr1Y&t=0s&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&index=10>
- <http://cs231n.github.io/convolutional-networks/#architectures>

CNN arhitektura

Today: CNN Architectures

Case Studies

- AlexNet
- VGG
- GoogLeNet
- ResNet

Also....

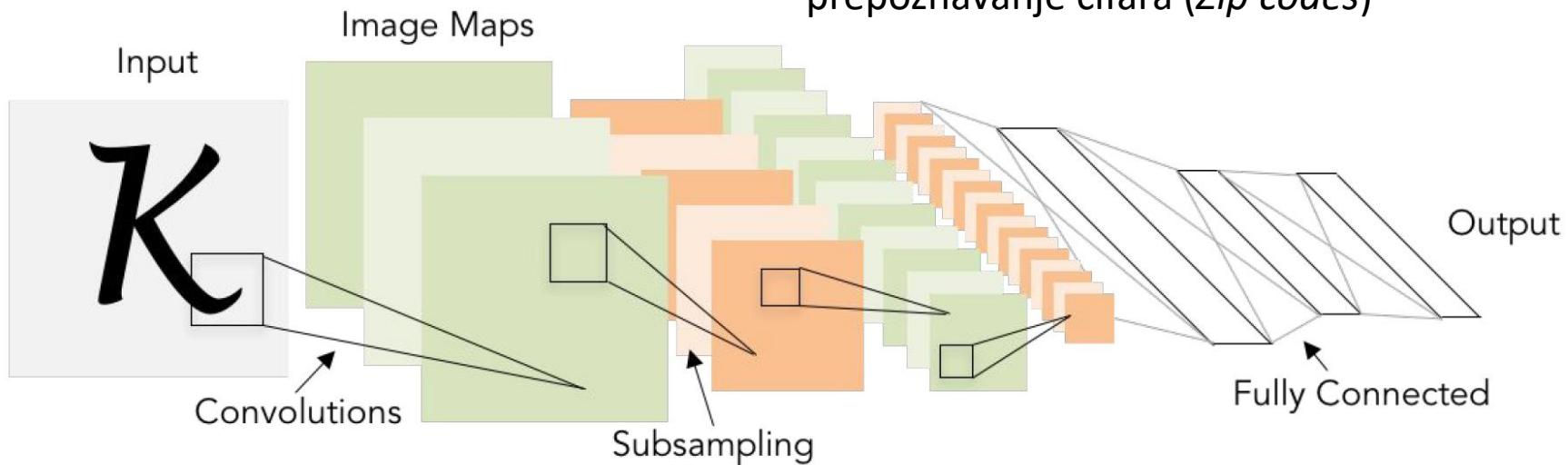
- NiN (Network in Network)
- Wide ResNet
- ResNeXT
- Stochastic Depth
- DenseNet
- FractalNet
- SqueezeNet

LeNet – jedna od prvih uspešnih CNN

Review: LeNet-5

[LeCun et al., 1998]

- Jedna od prvih instanci CNN koja je uspešno upotrebljena u praksi
- Veoma uspešno primenjena za prepoznavanje cifara (*Zip codes*)



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]

AlexNet – prvi pobednik na ImageNet

Case Study: AlexNet

[Krizhevsky et al. 2012]

Architecture:

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

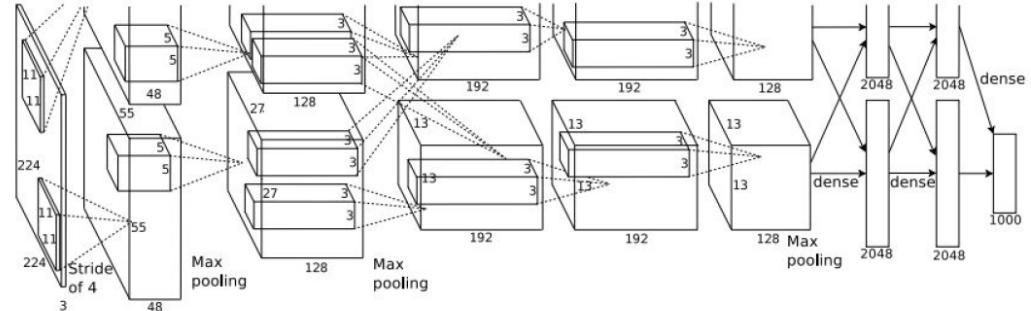
CONV5

Max POOL3

FC6

FC7

FC8



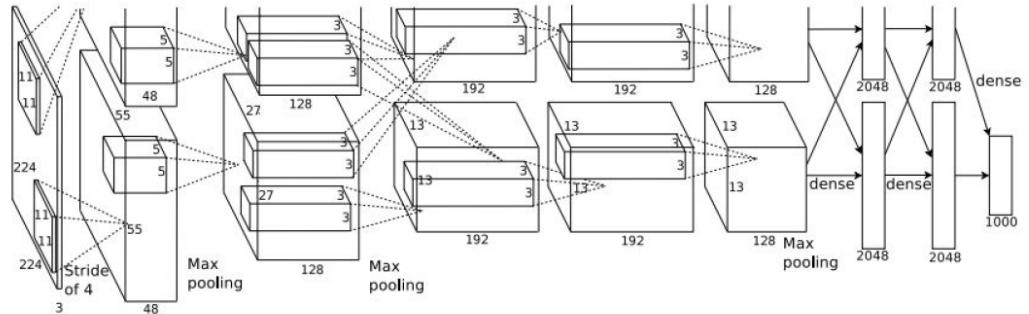
- Prva zaista velika CNN koja je mogla dobro da radi na problemu klasifikacije slika. Popularizovala CNN pobedom na *ImageNet* takmičenju 2012.
- Veoma slična arhitekturi LeNet mreže, samo više slojeva
- Za razliku od LeNet, takođe je stekovala više CONV slojeva jedan na drugi (ranije je bilo uobičajeno imati samo jedan CONV sloj odmah praćen POOL slojem)

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

AlexNet – prvi pobednik na ImageNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

=>

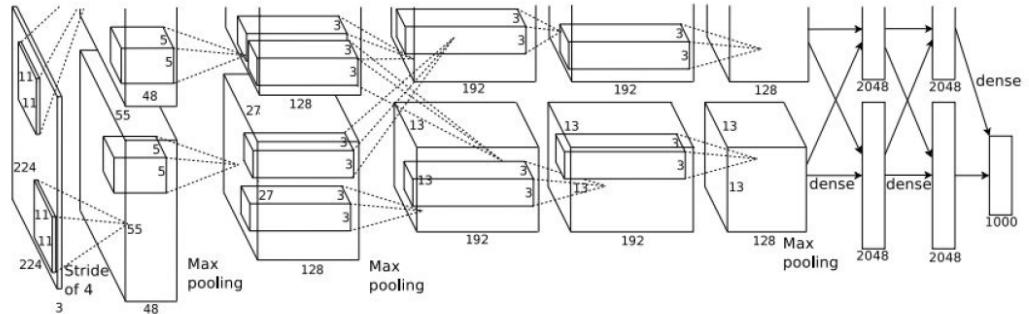
Q: what is the output volume size? Hint: $(227-11)/4+1 = 55$

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

AlexNet – prvi pobednik na ImageNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Output volume **[55x55x96]**

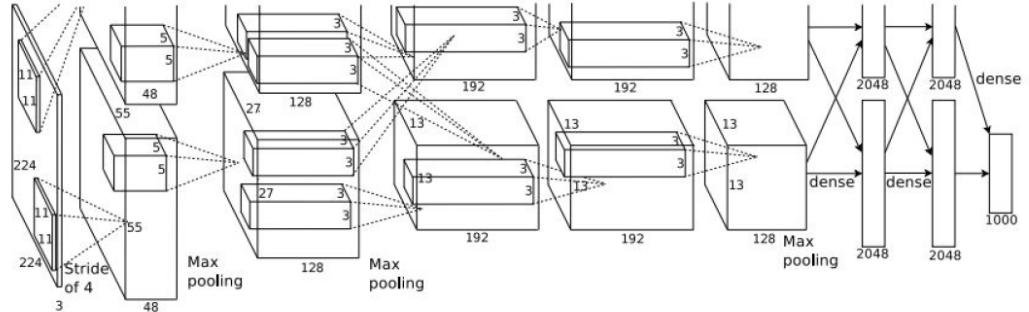
Q: What is the total number of parameters in this layer?

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

AlexNet – prvi pobednik na ImageNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Output volume **[55x55x96]**

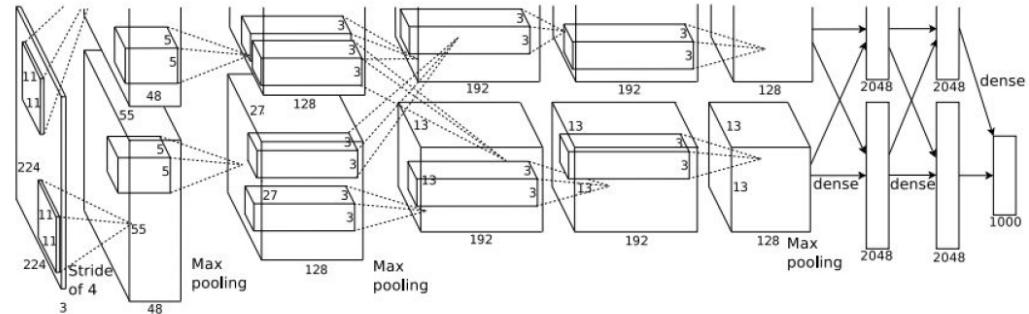
Parameters: $(11 \times 11 \times 3) \times 96 = 35K$

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

AlexNet – prvi pobednik na ImageNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

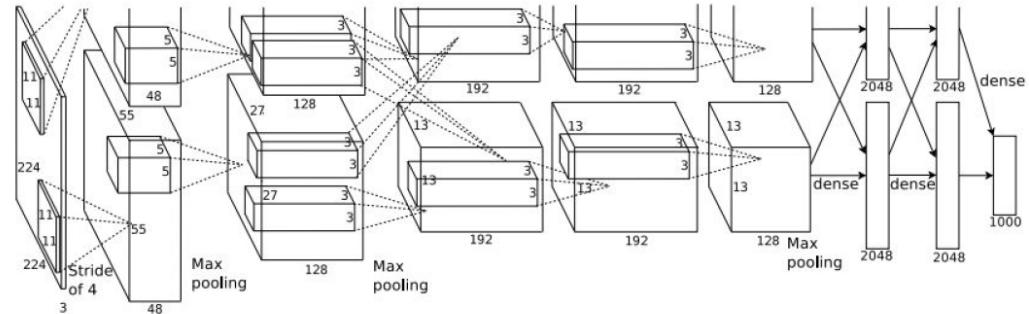
Q: what is the output volume size? Hint: $(55-3)/2+1 = 27$

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

AlexNet – prvi pobednik na ImageNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Output volume: 27x27x96

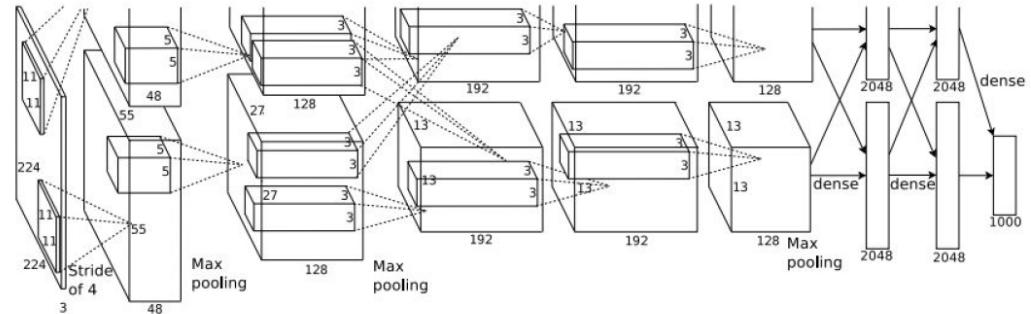
Q: what is the number of parameters in this layer?

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

AlexNet – prvi pobednik na ImageNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Output volume: 27x27x96

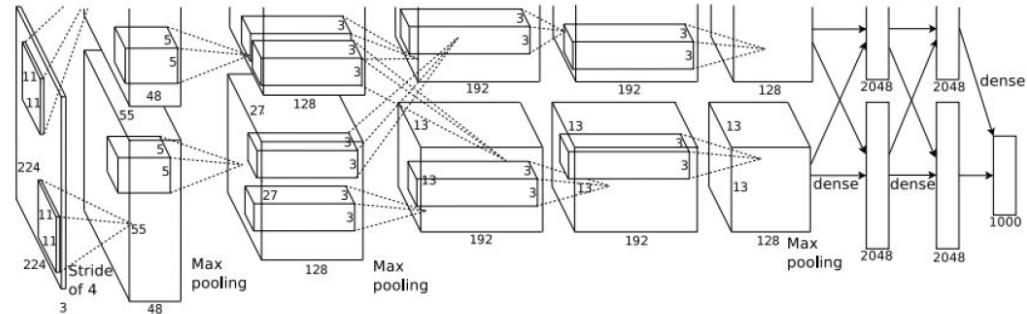
Parameters: 0!

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

AlexNet – prvi pobednik na ImageNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

After POOL1: 27x27x96

...

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

AlexNet – prvi pobednik na ImageNet

Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

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

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

[27x27x96] NORM1: Normalization layer

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

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

[13x13x256] NORM2: Normalization layer

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

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

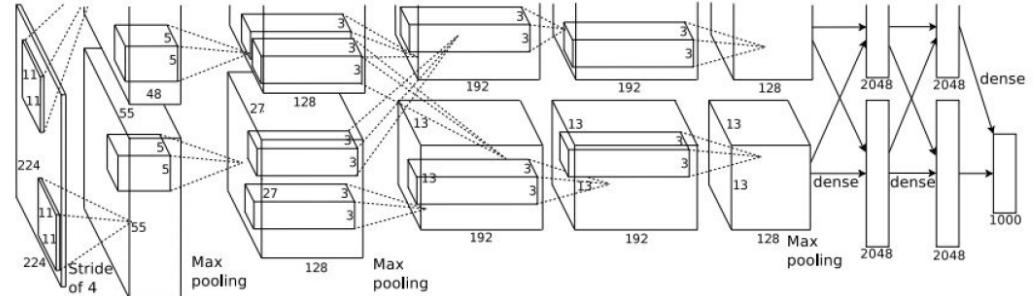
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

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

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)



Details/Retrospectives:

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

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

AlexNet – prvi pobednik na ImageNet

Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

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

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

[27x27x96] NORM1: Normalization layer

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

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

[13x13x256] NORM2: Normalization layer

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

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

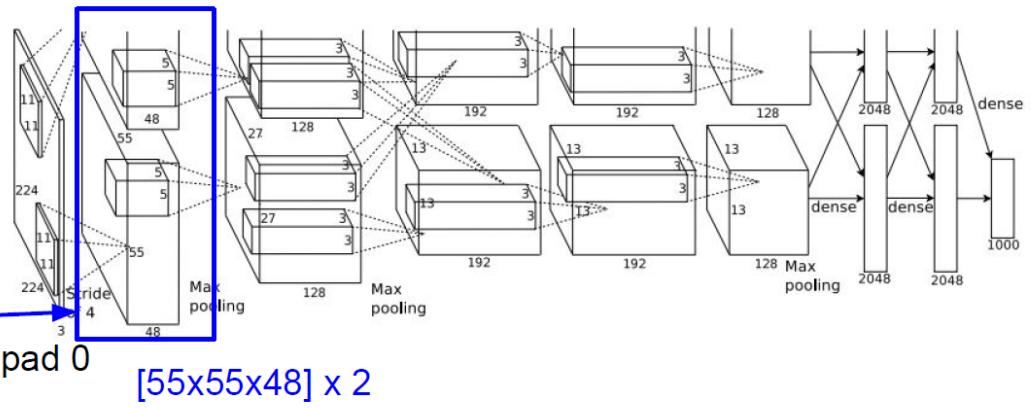
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

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

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)



Historical note: Trained on GTX 580 GPU with only 3 GB of memory.
Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

AlexNet – prvi pobednik na ImageNet

Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

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

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

[27x27x96] NORM1: Normalization layer

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

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

[13x13x256] NORM2: Normalization layer

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

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

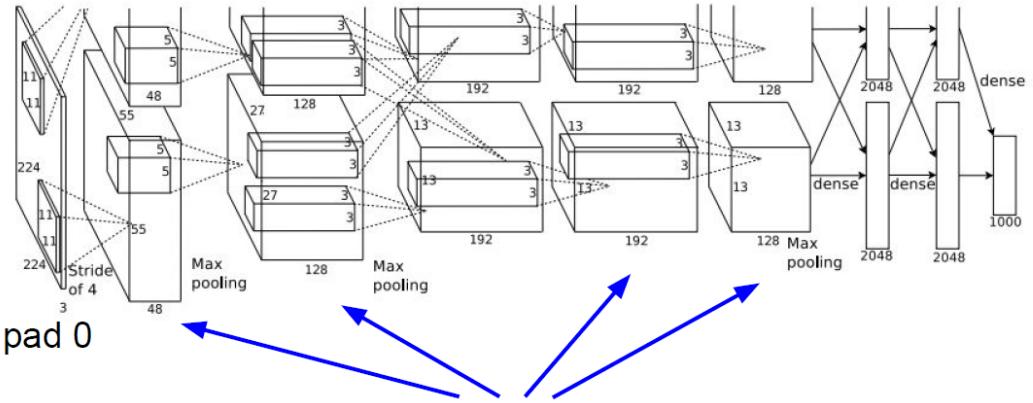
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[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

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

AlexNet – prvi pobednik na ImageNet

Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

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

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

[27x27x96] NORM1: Normalization layer

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

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

[13x13x256] NORM2: Normalization layer

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

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

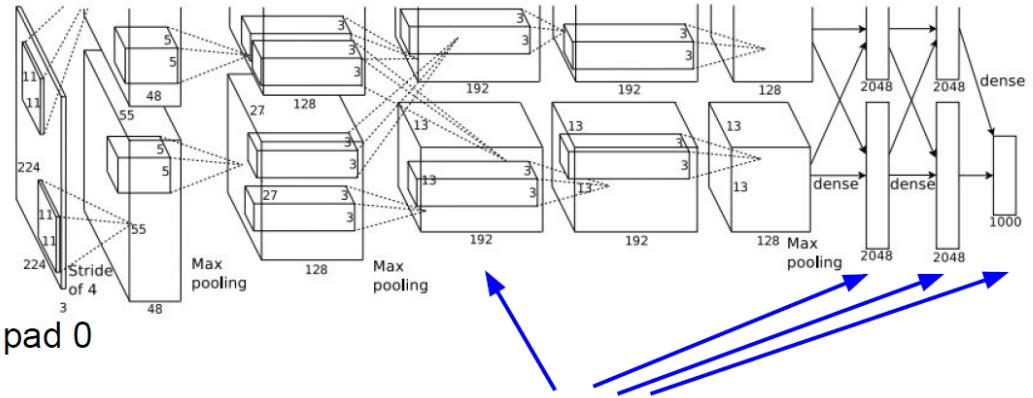
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[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

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

AlexNet – prvi pobednik na ImageNet

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

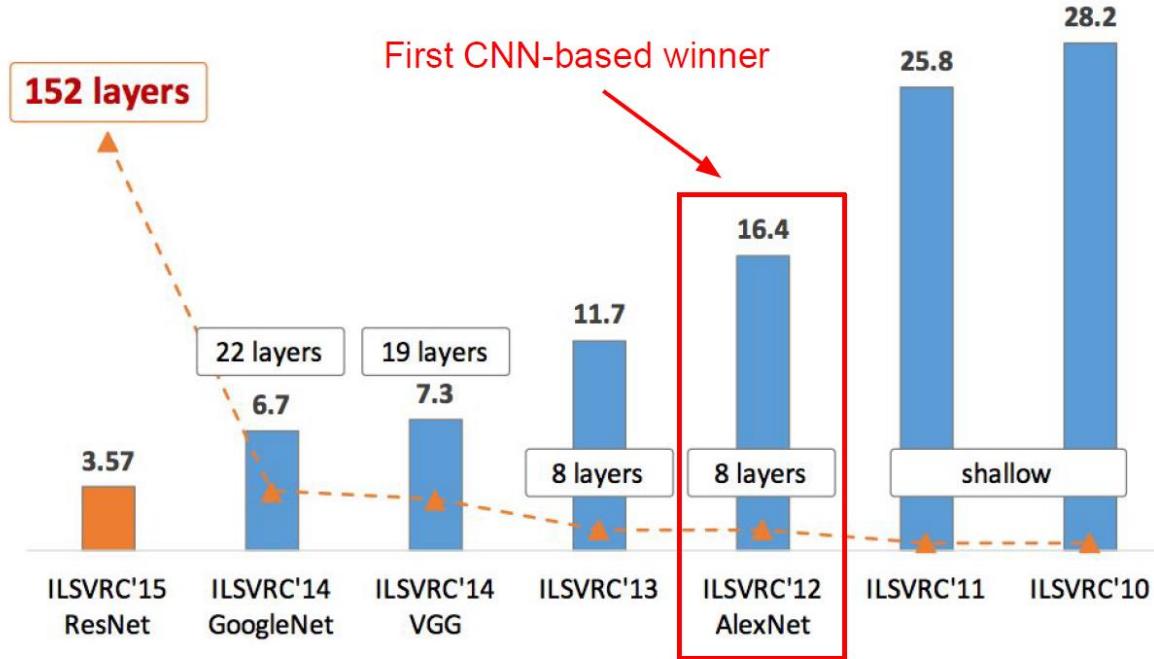


Figure copyright Kaiming He, 2016. Reproduced with permission.

AlexNet – prvi pobjednik na ImageNet

- Široko se koristila kao osnova mnogih narednih arhitektura, bila je sveprisutna do pre par godina
 - Ali se i danas u velikoj meri koristi, npr. za *transfer learning*
- Danas ima skorašnjijih arhitektura sa boljim performansama (naredni slajdovi)
 - Danas se češće koriste u praksi

ZFNet – pobojšanje parametara AlexNet

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

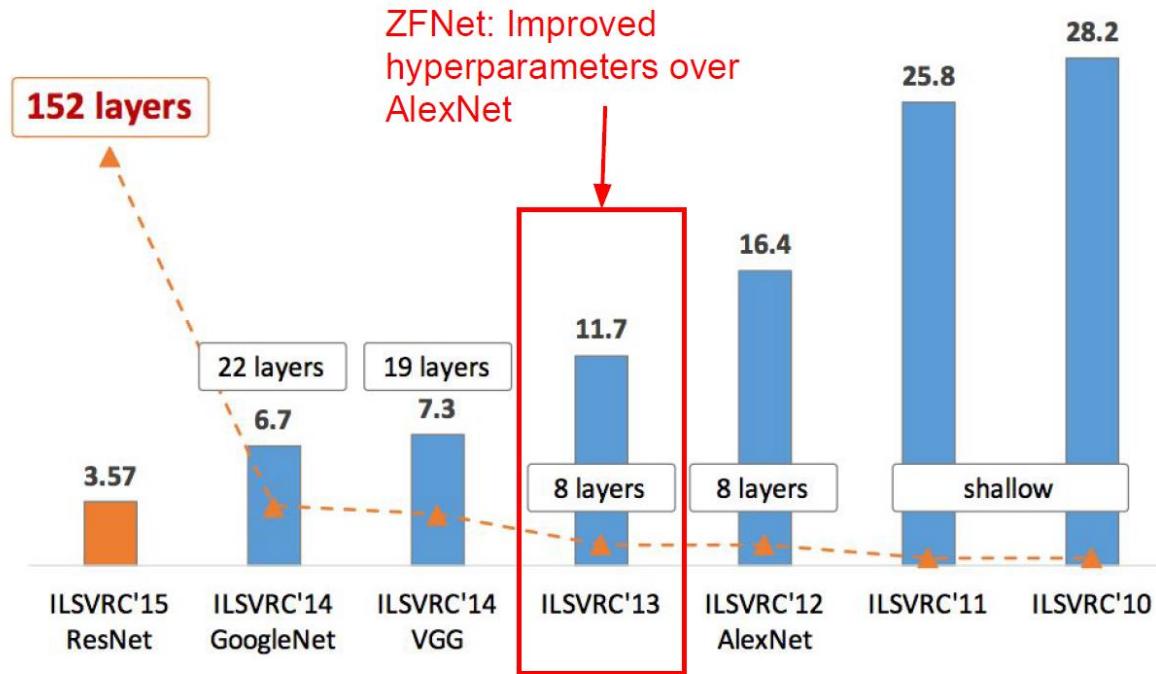
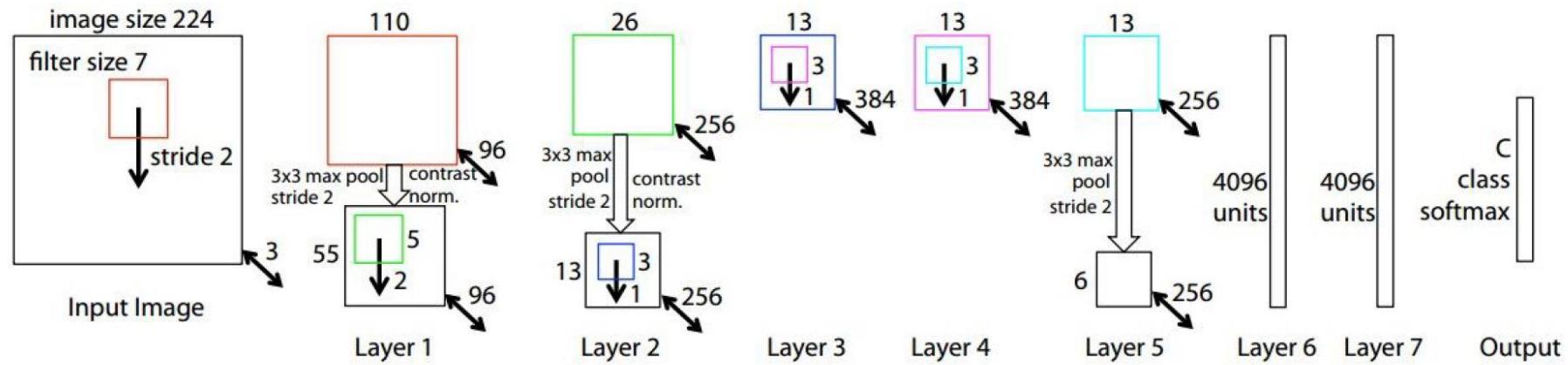


Figure copyright Kaiming He, 2016. Reproduced with permission.

ZFNet – pobojšanje parametara AlexNet

ZFNet

[Zeiler and Fergus, 2013]



AlexNet but:

CONV1: change from (11x11 stride 4) to (7x7 stride 2)

CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

TODO: remake figure

ImageNet top 5 error: 16.4% -> 11.7%

ZFNet – pobojšanje parametara AlexNet

- Uglavnom pobojšanje hiper-parametara AlexNet mreže
 - Isti broj slojeva i ista struktura
 - Promenili su stride, broj filtera
 - Igrajući se sa hiper-parametrima su uspeli da poprave *error rate*, ali je u pitanju ista ideja

VGG i GoogLeNet

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

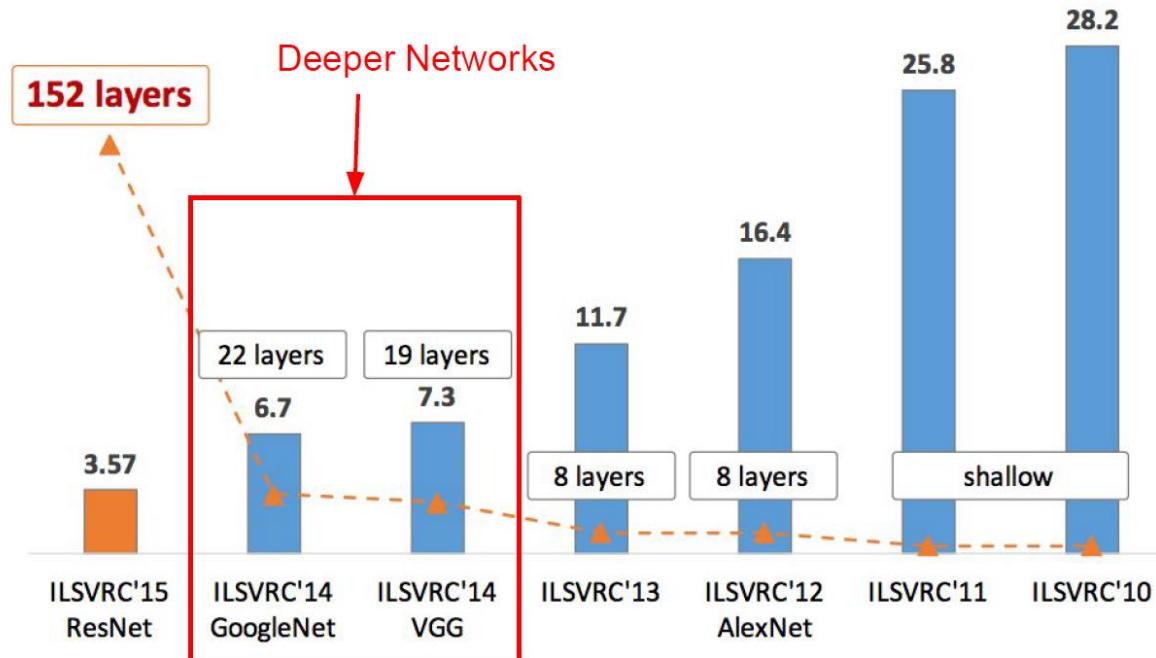


Figure copyright Kaiming He, 2016. Reproduced with permission.

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)

-> 16 - 19 layers (VGG16Net)

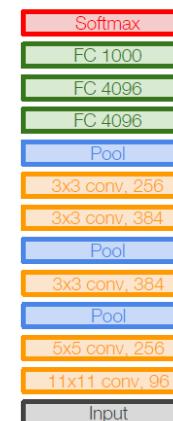
Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13

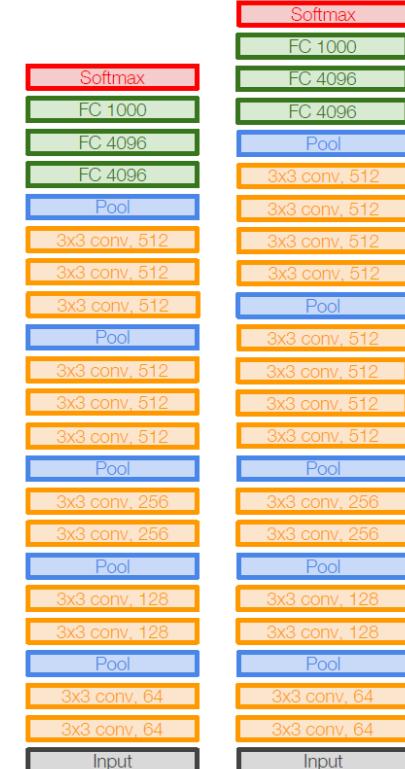
(ZFNet)

-> 7.3% top 5 error in ILSVRC'14

- Glavni doprinos: dubina mreže je kritična za performanse



AlexNet



VGG16

VGG19

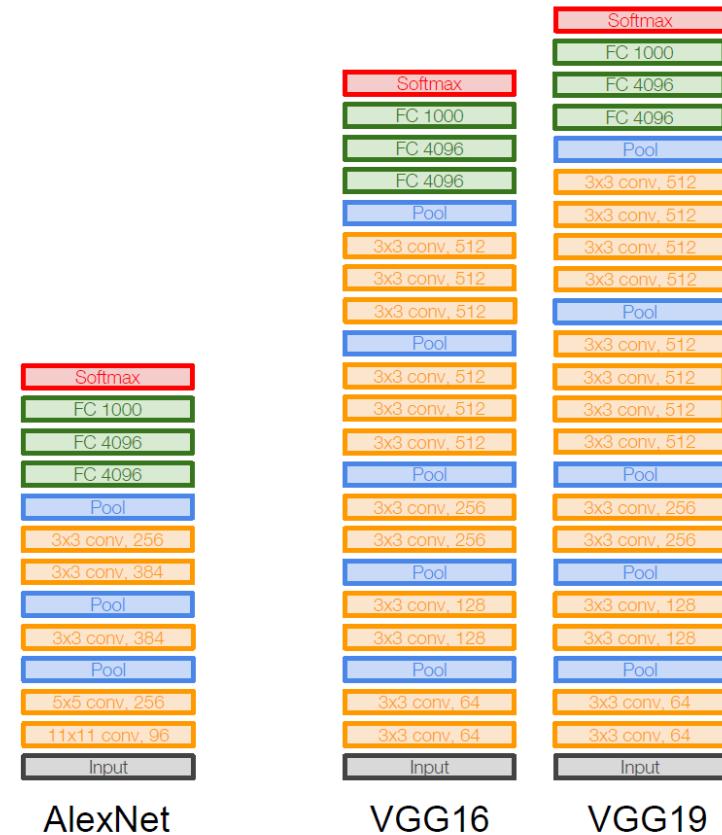
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

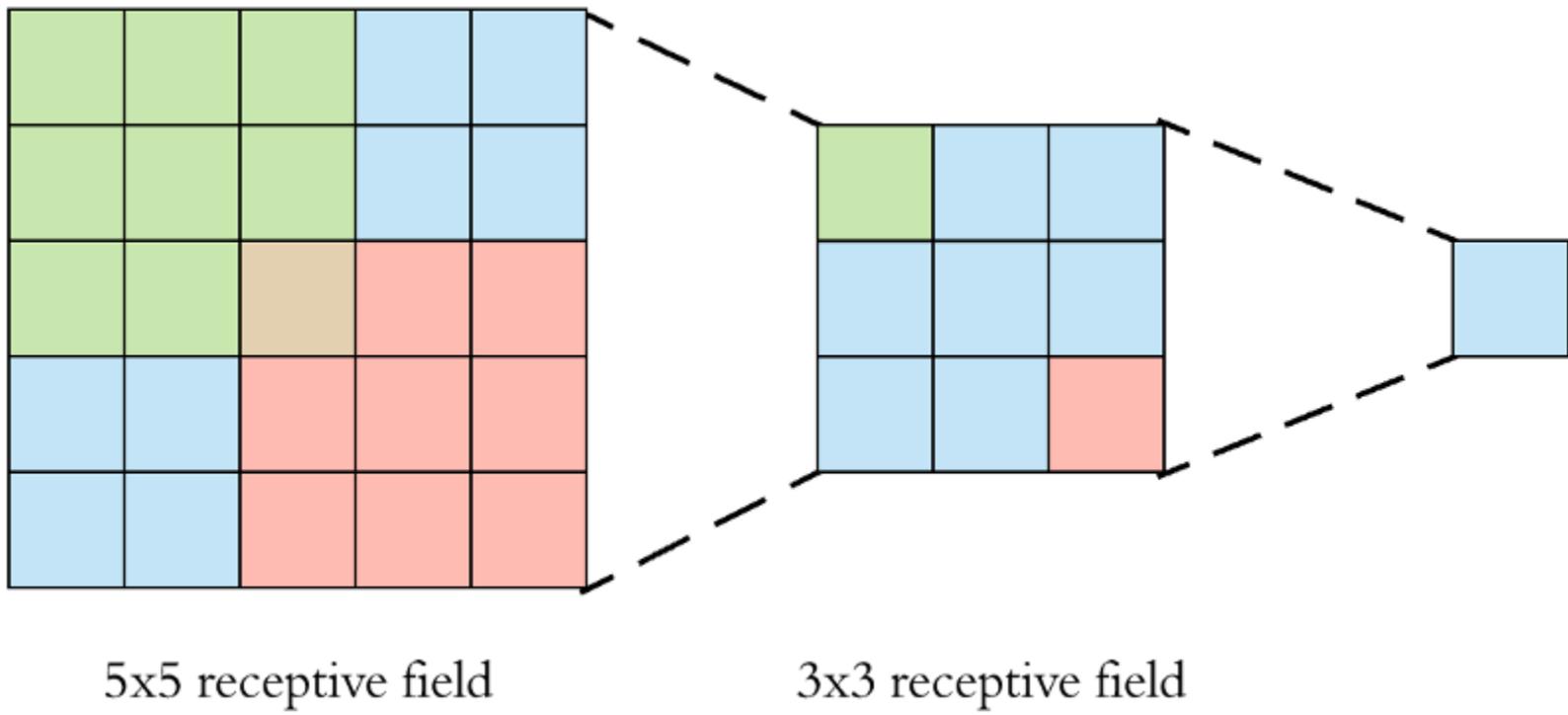
Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers
has same **effective receptive field** as
one 7x7 conv layer

Q: What is the effective receptive field of
three 3x3 conv (stride 1) layers?



VGG – efektivno receptivno polje



- Dva stekovana 3×3 filtera
- Sa desna na levo:
 - Filter u drugom sloju gleda 3×3 receptivno polje prvog sloja
 - Ovo receptivno polje prvog sloja gleda 5×5 receptivno polje ulaznog sloja

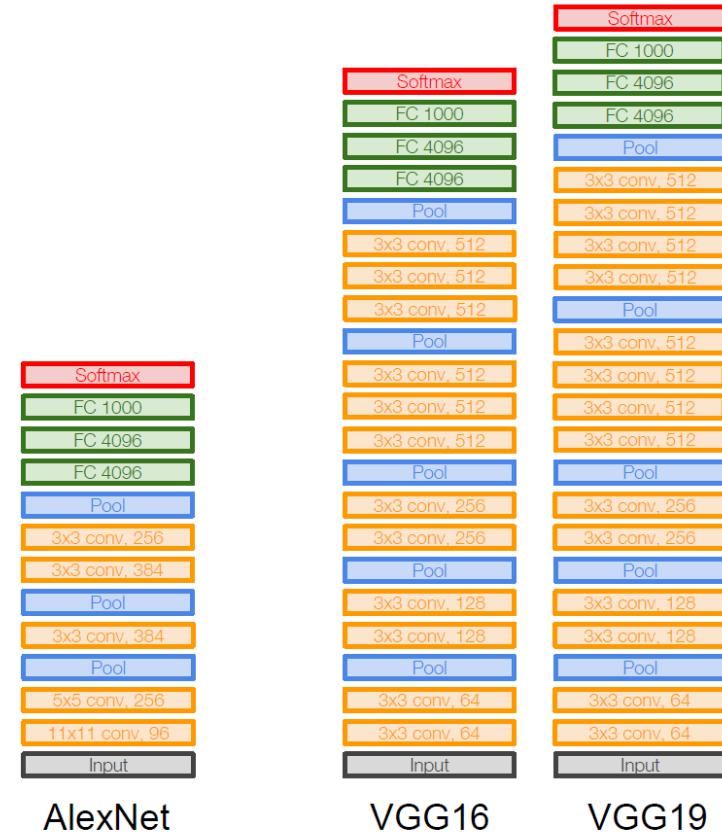
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers
has same **effective receptive field** as
one 7x7 conv layer

[7x7]



Case Study: VGGNet

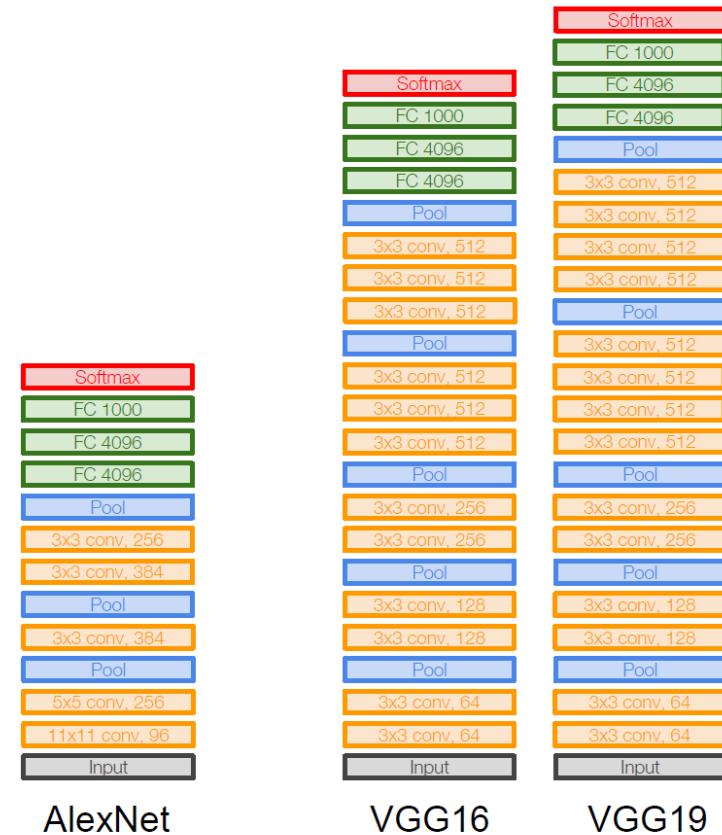
[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers
has same **effective receptive field** as
one 7x7 conv layer

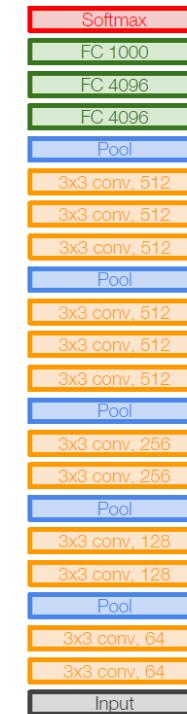
But deeper, more non-linearities

And fewer parameters: $3 * (3^2 C^2)$ vs.
 $7^2 C^2$ for C channels per layer



VGG

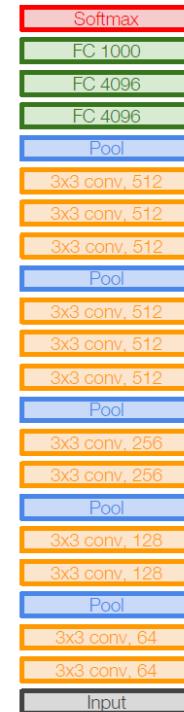
INPUT: [224x224x3] memory: $224 \times 224 \times 3 = 150\text{K}$ params: 0 (not counting biases)
 CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2\text{M}$ params: $(3 \times 3 \times 3) \times 64 = 1,728$
 CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2\text{M}$ params: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



VGG16

VGG

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



VGG16

TOTAL memory: $24M * 4 \text{ bytes} \approx 96\text{MB} / \text{image}$ (only forward! ~ 2 for bwd)

TOTAL params: 138M parameters

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

TOTAL memory: $24\text{M} * 4 \text{ bytes} \approx 96\text{MB} / \text{image}$ (only forward! ~ 2 for bwd)

TOTAL params: 138M parameters

Note:

Most memory is in early CONV

(veće prostorne dimenzije)

Most params are in late FC

Kasnije arhitekture su pokazale da se ovi slojevi mogu ukloniti, bez gubitka performansi

VGG

Conv3-64: 64 filtera veličine 3×3

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

TOTAL memory: $24\text{M} * 4 \text{ bytes} \approx 96\text{MB} / \text{image}$ (only forward! ~ 2 for bwd)

TOTAL params: 138M parameters



VGG16

Common names

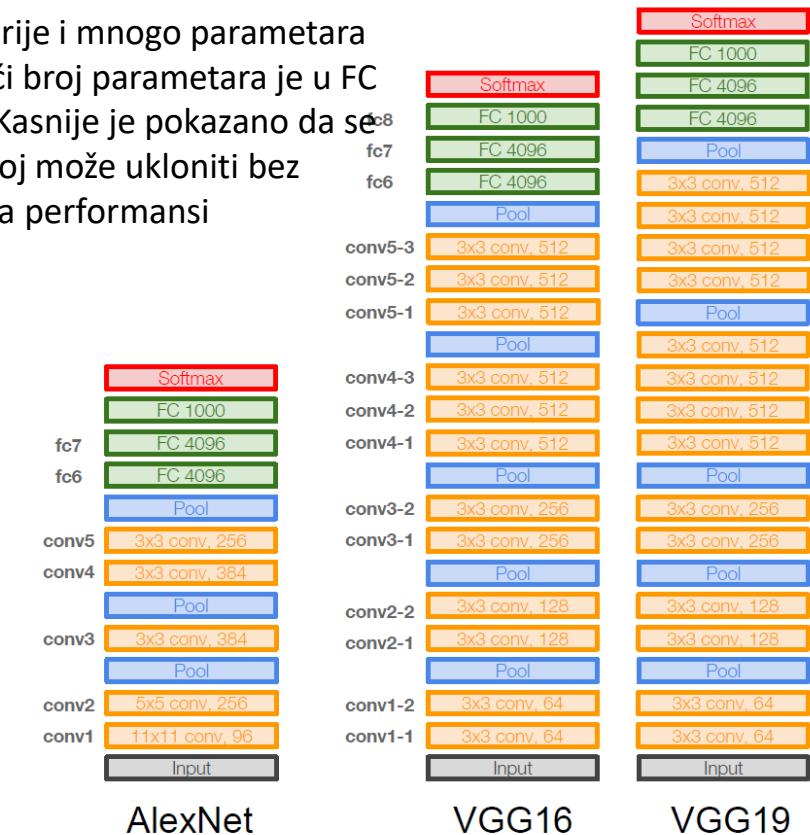
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Details:

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks

- Nedostatak: veliko zauzeće memorije i mnogo parametara
- Najveći broj parametara je u FC sloju. Kasnije je pokazano da se ovaj sloj može ukloniti bez gubitka performansi



GoogLeNet

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

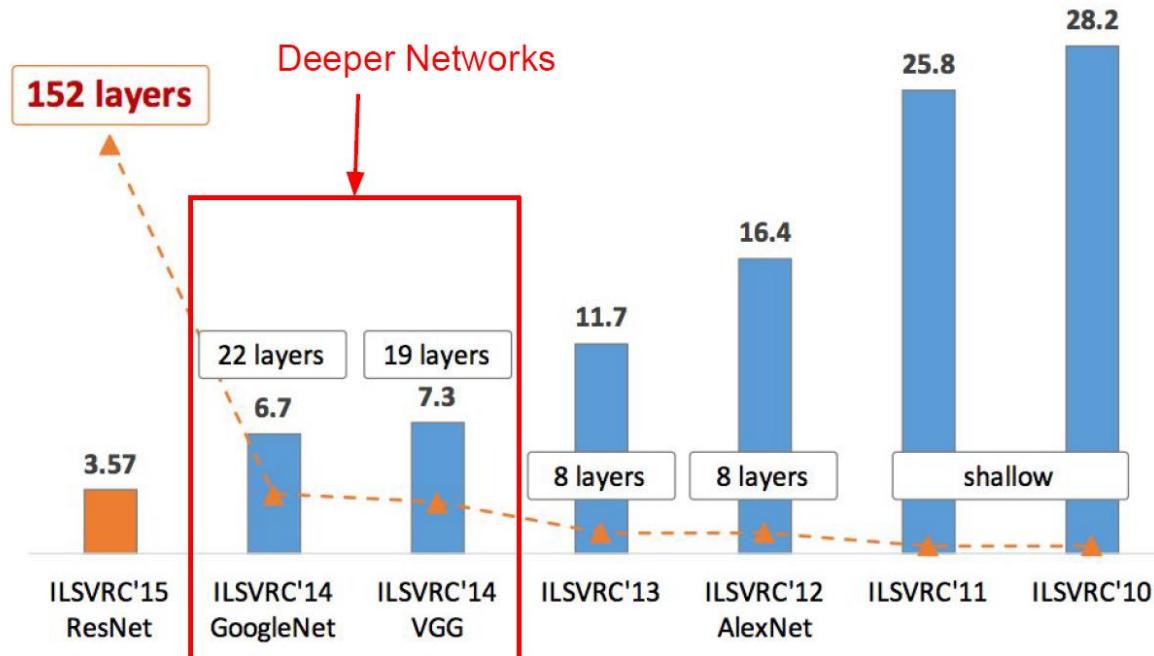


Figure copyright Kaiming He, 2016. Reproduced with permission.

GoogLeNet

Case Study: GoogLeNet

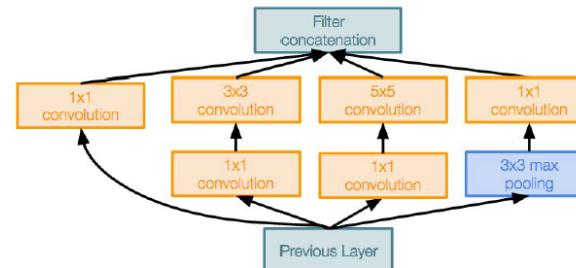
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

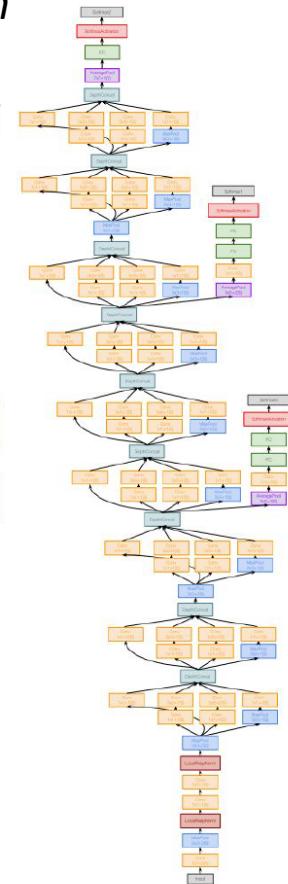
- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!
12x less than AlexNet
- ILSVRC’14 classification winner
(6.7% top 5 error)

Glavni doprinos je *Inception module*

- dramatično smanjio broj parametara
- Efikasno izračunavanje



Inception module

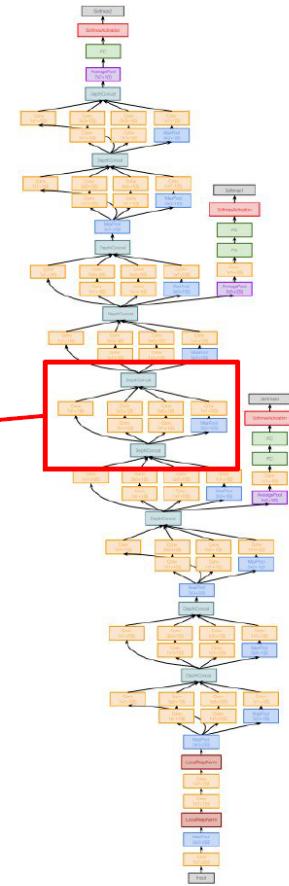
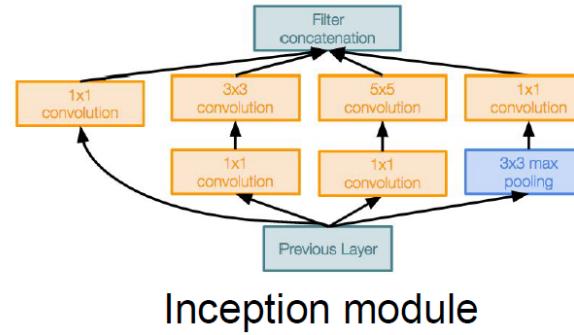


GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

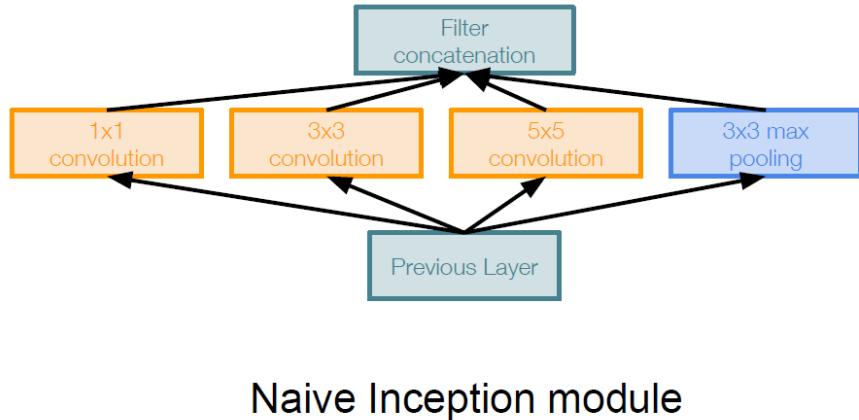
“Inception module”: design a good local network topology (network within a network) and then stack these modules on top of each other



GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]



Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depth-wise

GoogLeNet

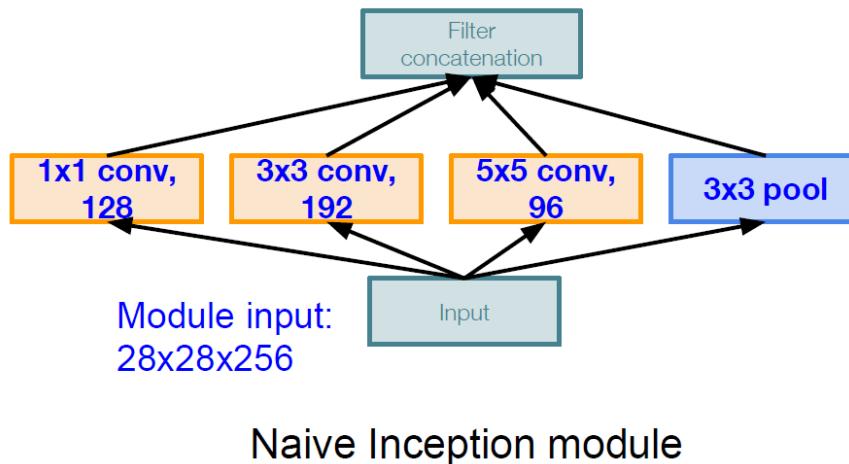
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q1: What is the output size of the
1x1 conv, with 128 filters?

Q: What is the problem with this?
[Hint: Computational complexity]



GoogLeNet

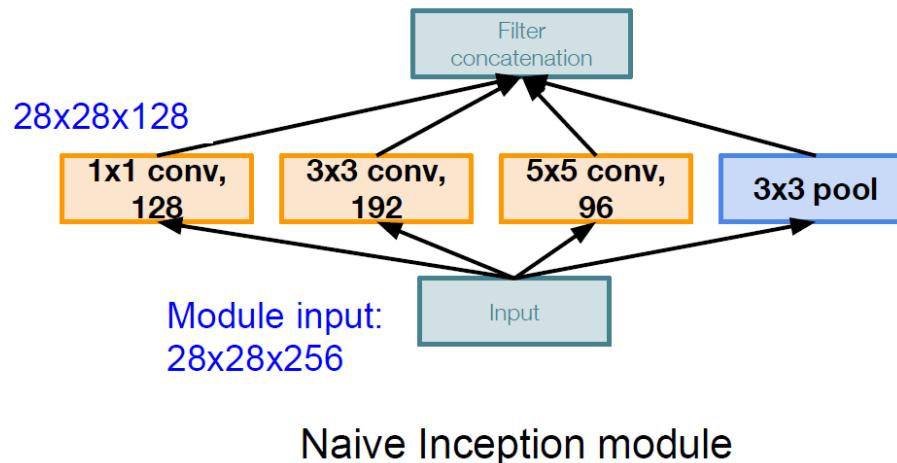
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q2: What are the output sizes of all different filter operations?

Q: What is the problem with this?
[Hint: Computational complexity]



GoogLeNet

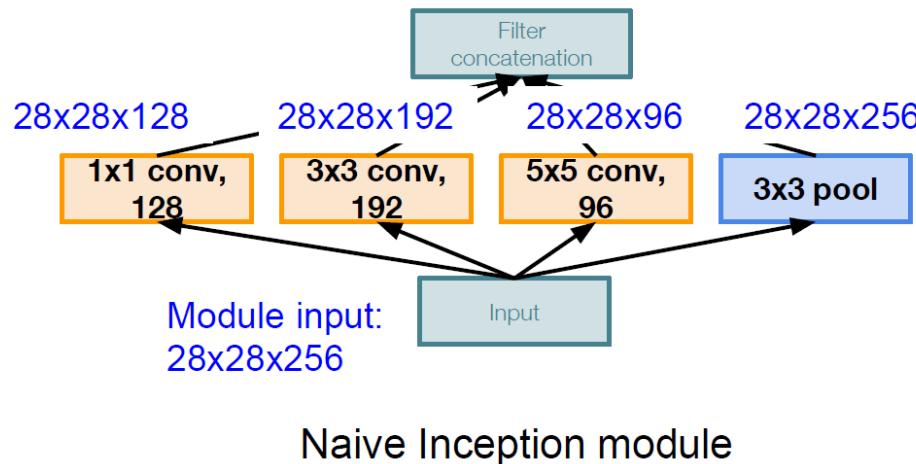
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q2: What are the output sizes of all different filter operations?

Q: What is the problem with this?
[Hint: Computational complexity]



Pool – očuvava dubinu, stride je podešen tako da se očuva ulazna prostorna dimenzija

GoogLeNet

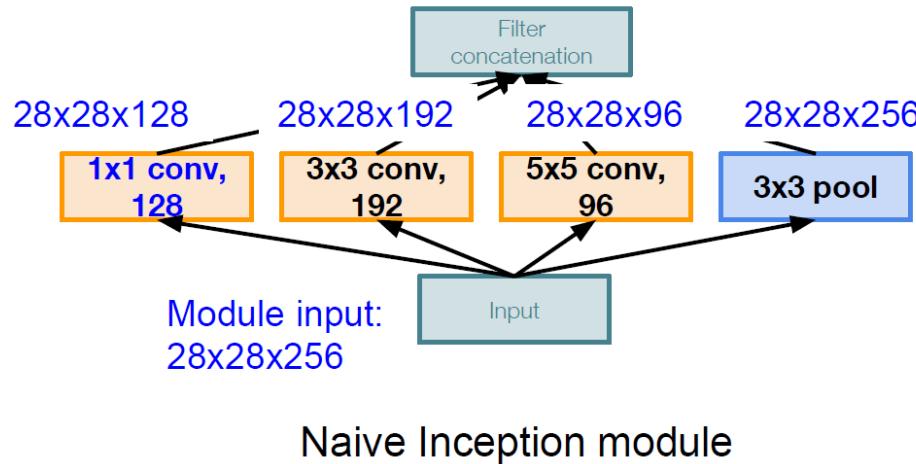
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3: What is output size after
filter concatenation?

$$28 \times 28 \times (128 + 192 + 96 + 256) = 28 \times 28 \times 672$$



GoogLeNet

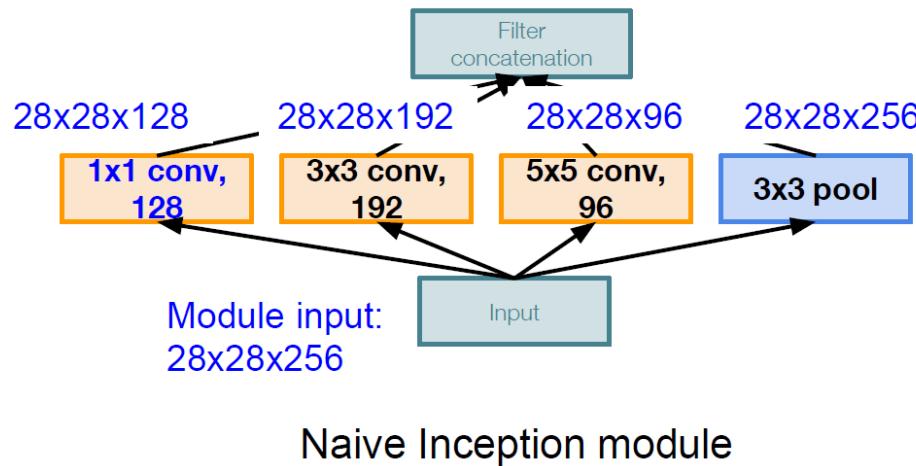
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3: What is output size after filter concatenation?

$$28 \times 28 \times (128 + 192 + 96 + 256) = 28 \times 28 \times 672$$



Q: What is the problem with this?
[Hint: Computational complexity]

Conv Ops:

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

Total: 854M ops

Very expensive compute

Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!

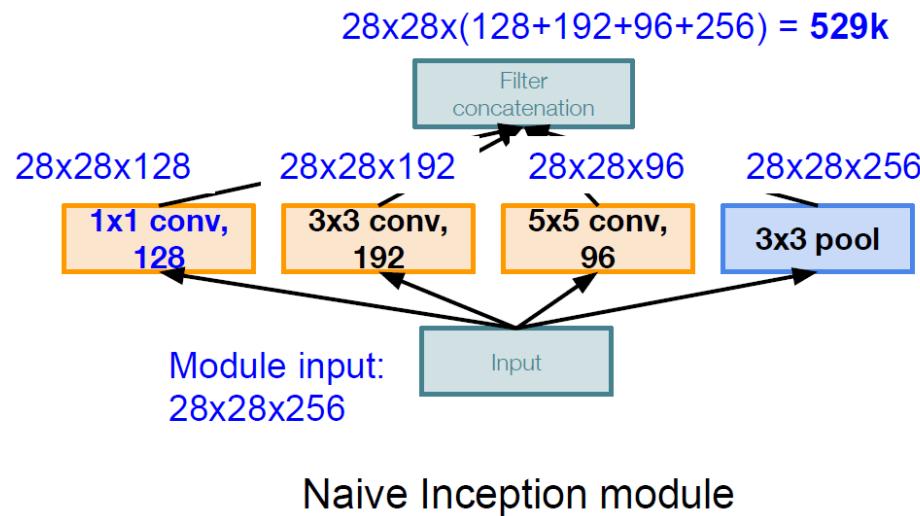
GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3: What is output size after filter concatenation?



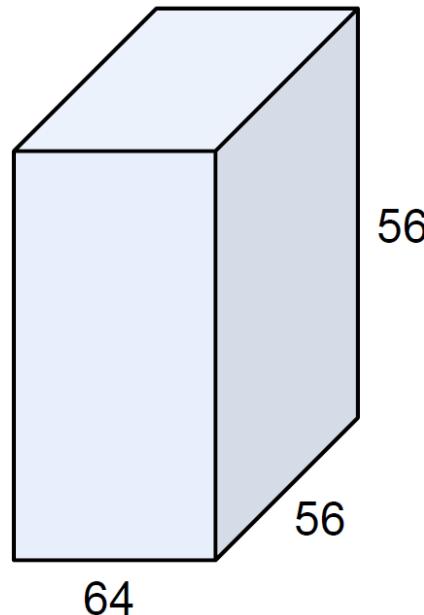
Q: What is the problem with this?
[Hint: Computational complexity]

Solution: “bottleneck” layers that use 1x1 convolutions to reduce feature depth

projektujemo mape obeležja na nižu dimenziju pre konvolucionih operatora („skupih“ slojeva)

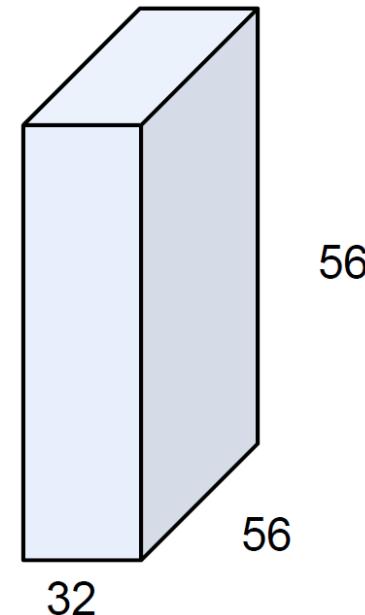
GoogLeNet

Reminder: 1x1 convolutions



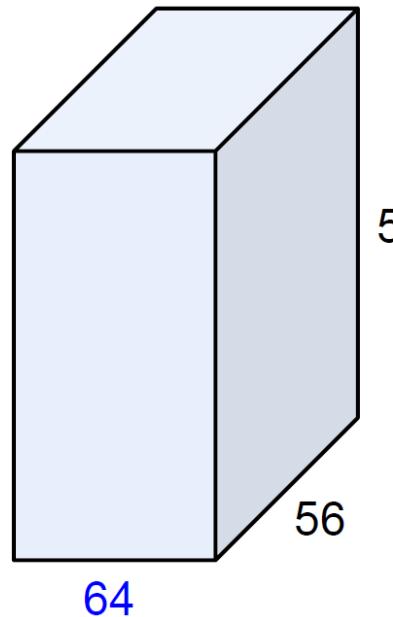
1x1 CONV
with 32 filters

*(each filter has size
1x1x64, and performs a
64-dimensional dot
product)*



GoogLeNet

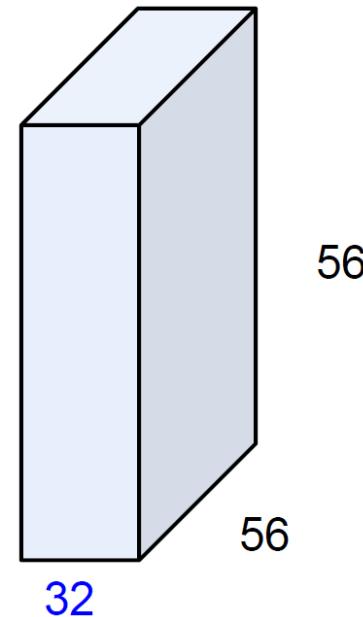
Reminder: 1x1 convolutions



1x1 CONV
with 32 filters

preserves spatial
dimensions, reduces depth!

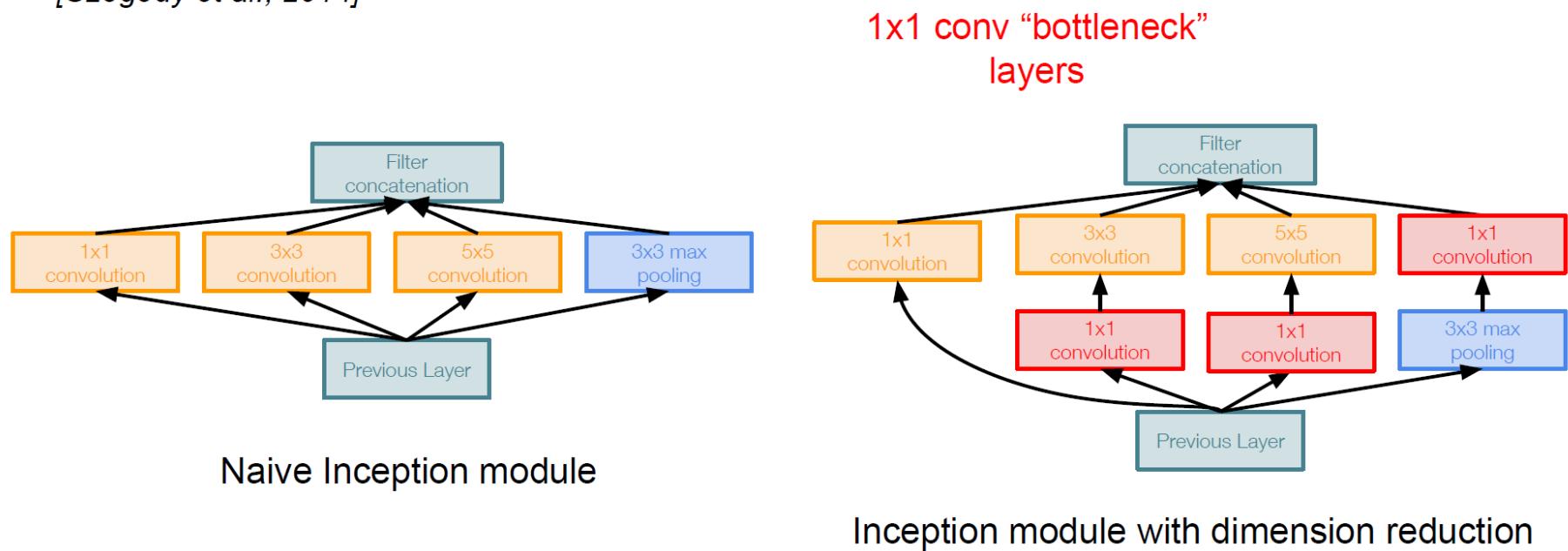
Projects depth to lower
dimension (combination of
feature maps)



GoogLeNet

Case Study: GoogLeNet

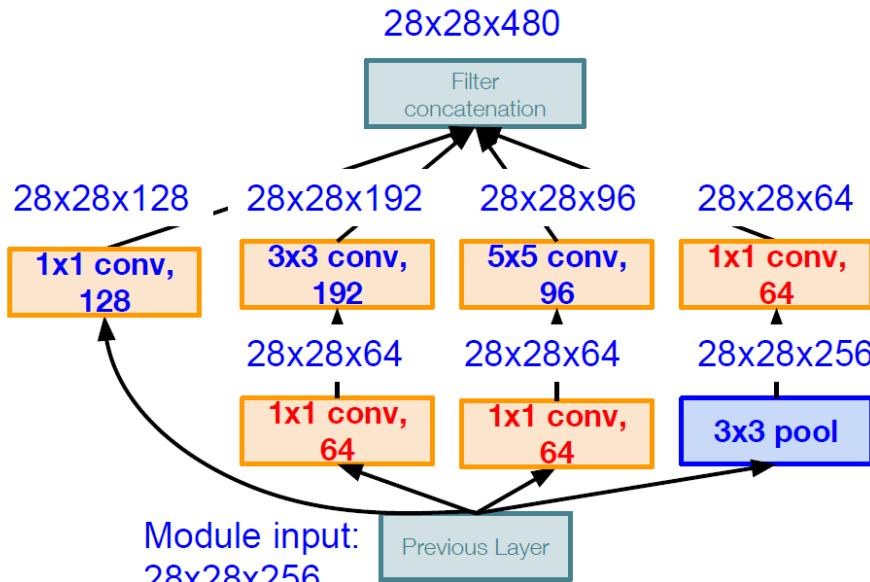
[Szegedy et al., 2014]



GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]



Inception module with dimension reduction

Using same parallel layers as naive example, and adding “1x1 conv, 64 filter” bottlenecks:

Conv Ops:

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

Total: 358M ops

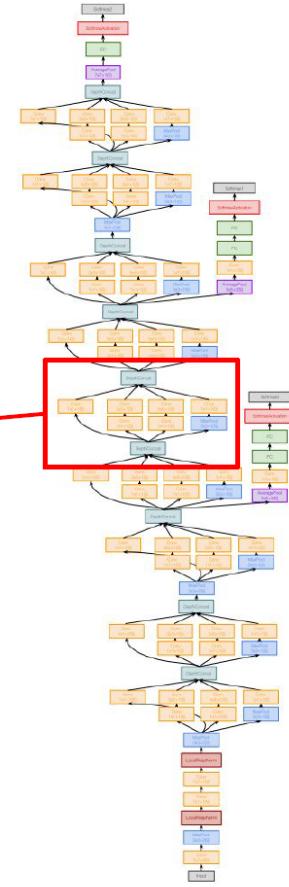
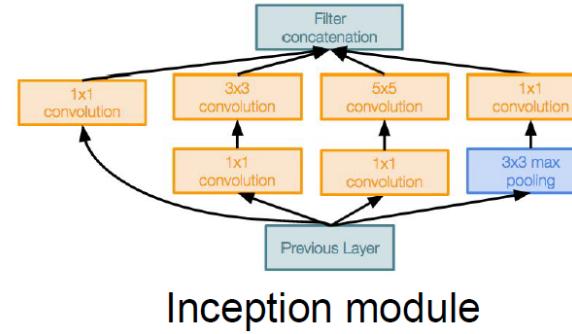
Compared to 854M ops for naive version
Bottleneck can also reduce depth after pooling layer

GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Stack Inception modules
with dimension reduction
on top of each other

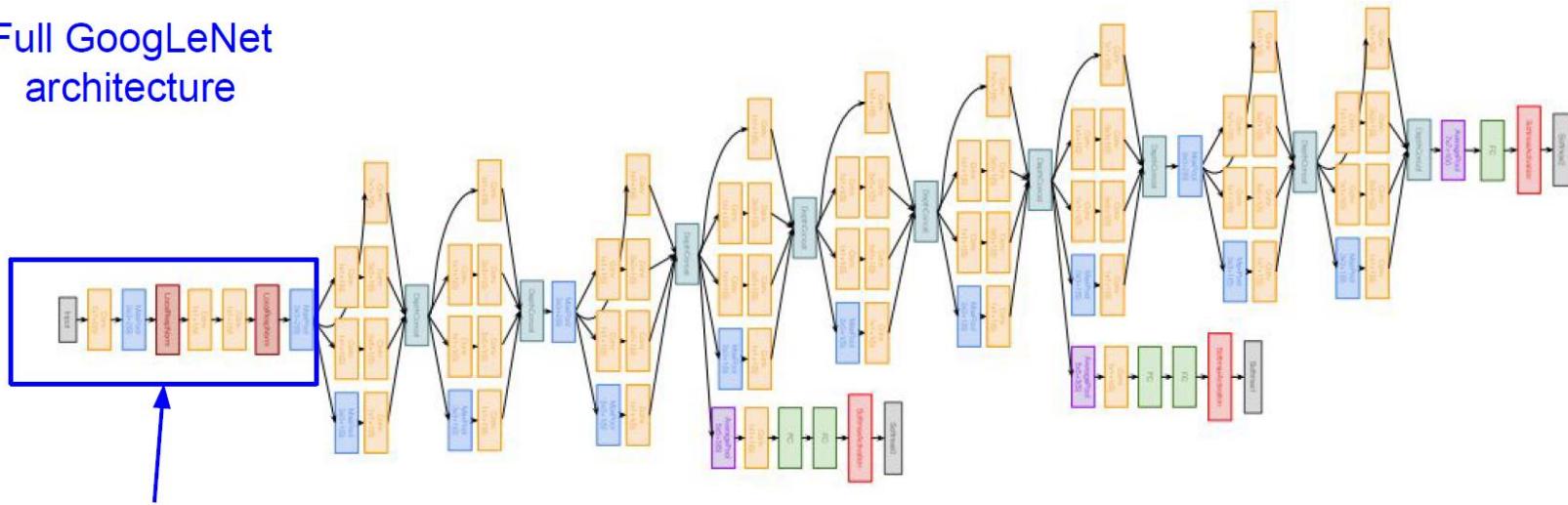


GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture



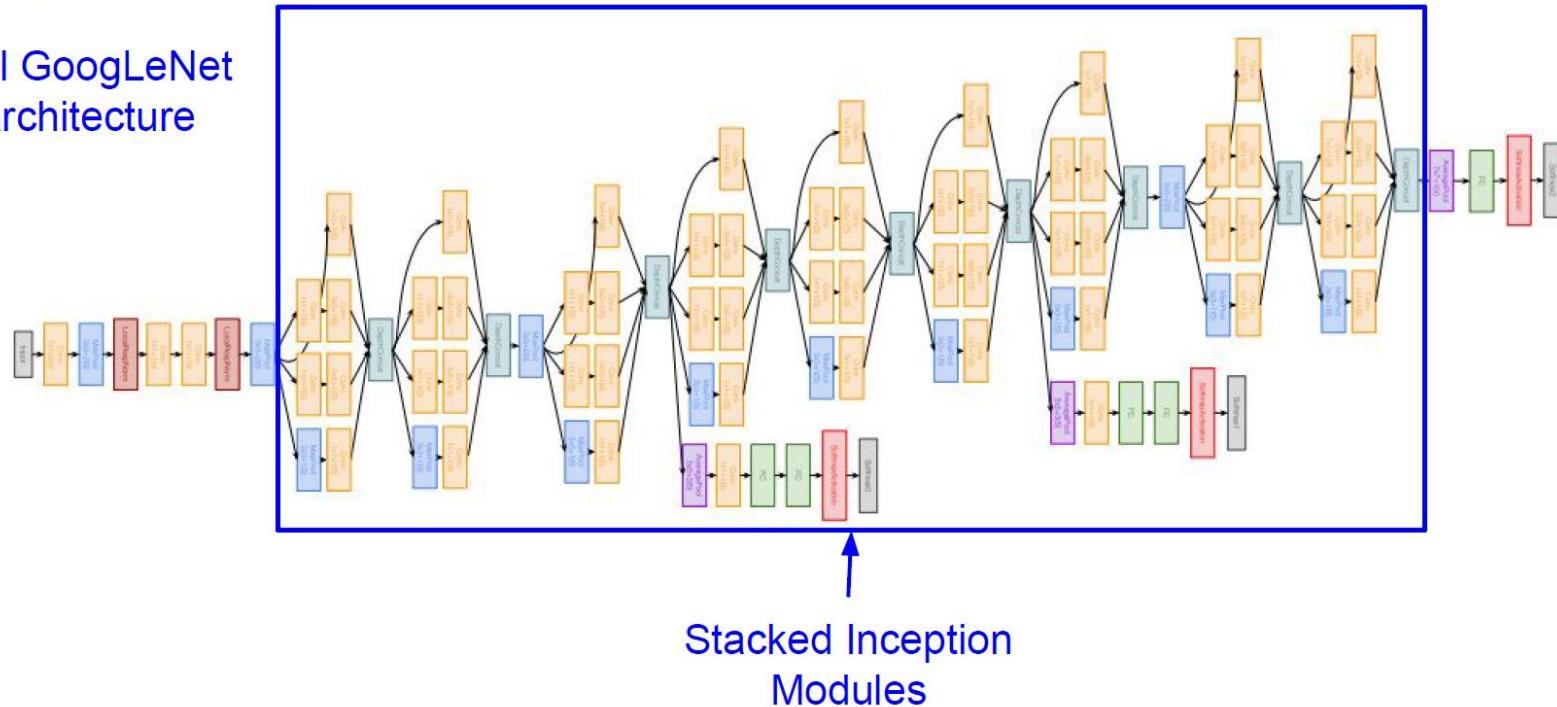
Stem Network:
Conv-Pool-
2x Conv-Pool

GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture

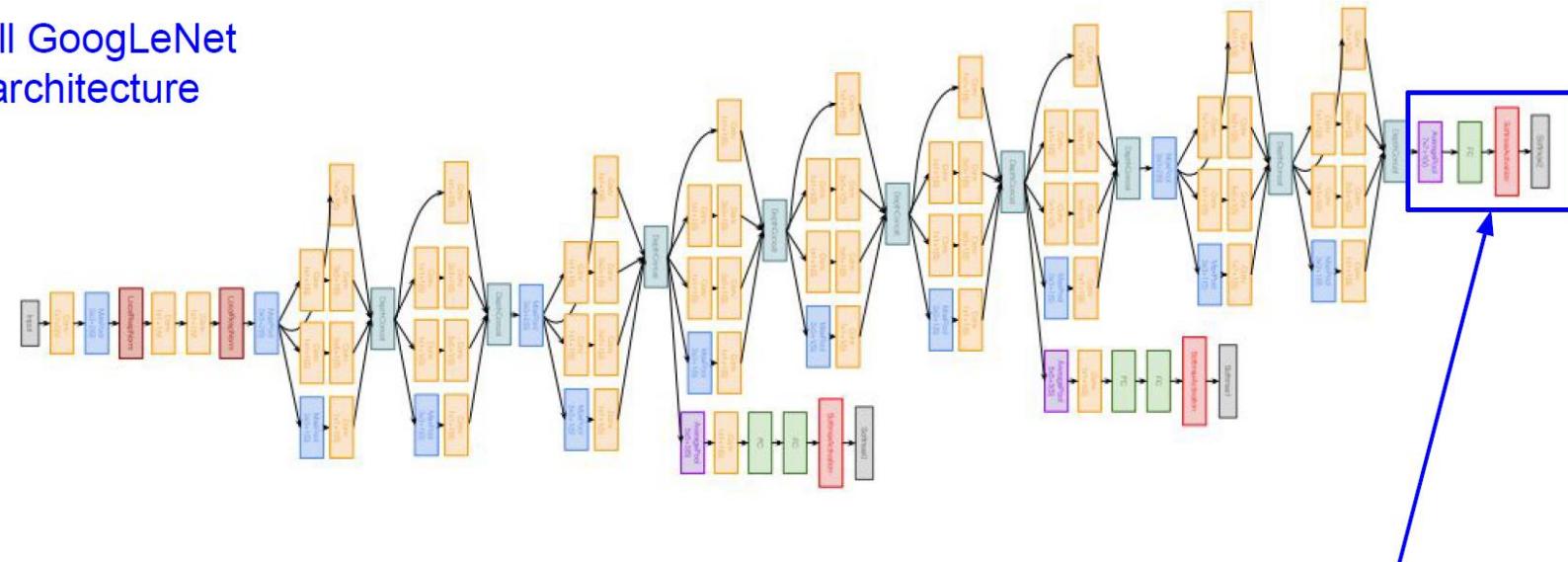


GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture



Average Pooling
umesto FC – eliminiše
veliki broj parametara

Classifier output
(removed expensive FC layers!)

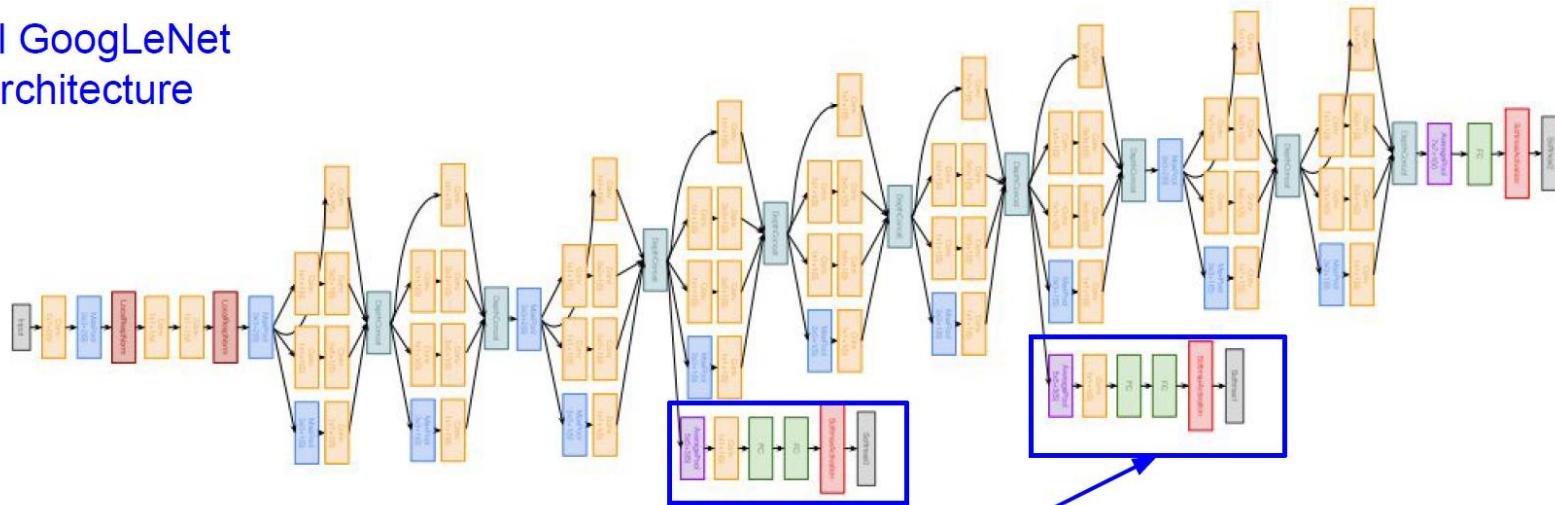
GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture

- Dodatni stemovi koji su pomoći klasifikacioni izlazi (softmax izlaz, 1000 ImageNet klasa)
- Pomoćne mreže koje obezbeđuju bolji tok gradijenta u rane slojeve



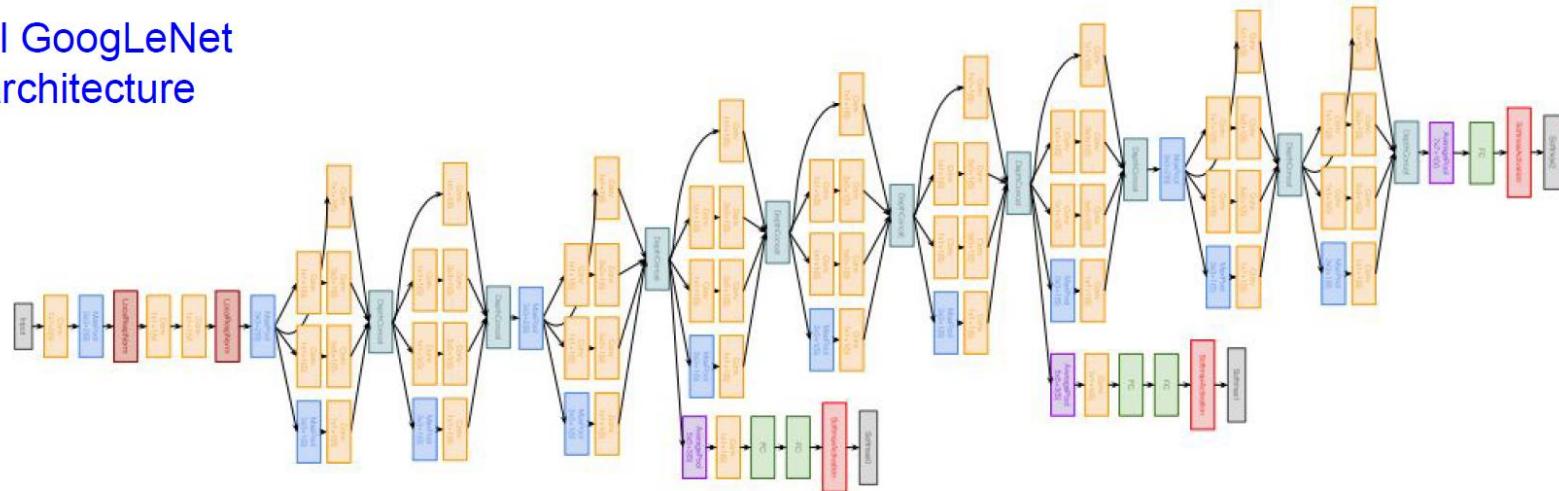
Auxiliary classification outputs to inject additional gradient at lower layers
(AvgPool-1x1Conv-FC-FC-Softmax)

GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet
architecture



22 total layers with weights (including each parallel layer in an Inception module)

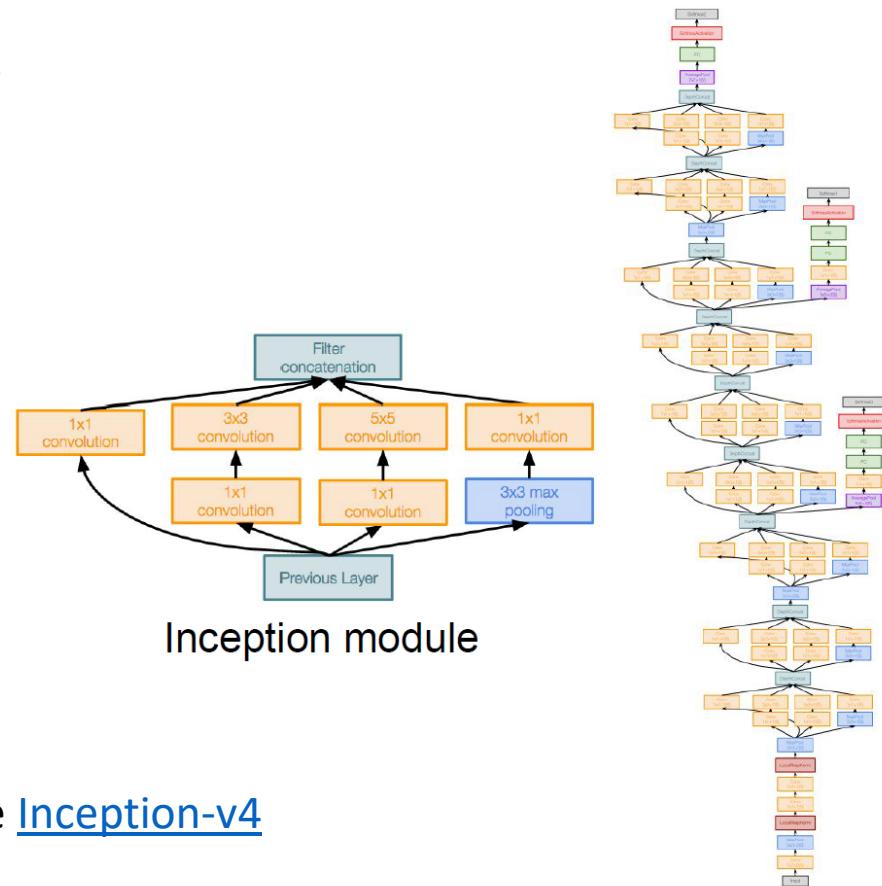
GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
 - Efficient “Inception” module
 - No FC layers
 - 12x less params than AlexNet
 - ILSVRC’14 classification winner (6.7% top 5 error)
-
- Ima nekoliko varijanti, najskoršnjija je [Inception-v4](#)



ResNet

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

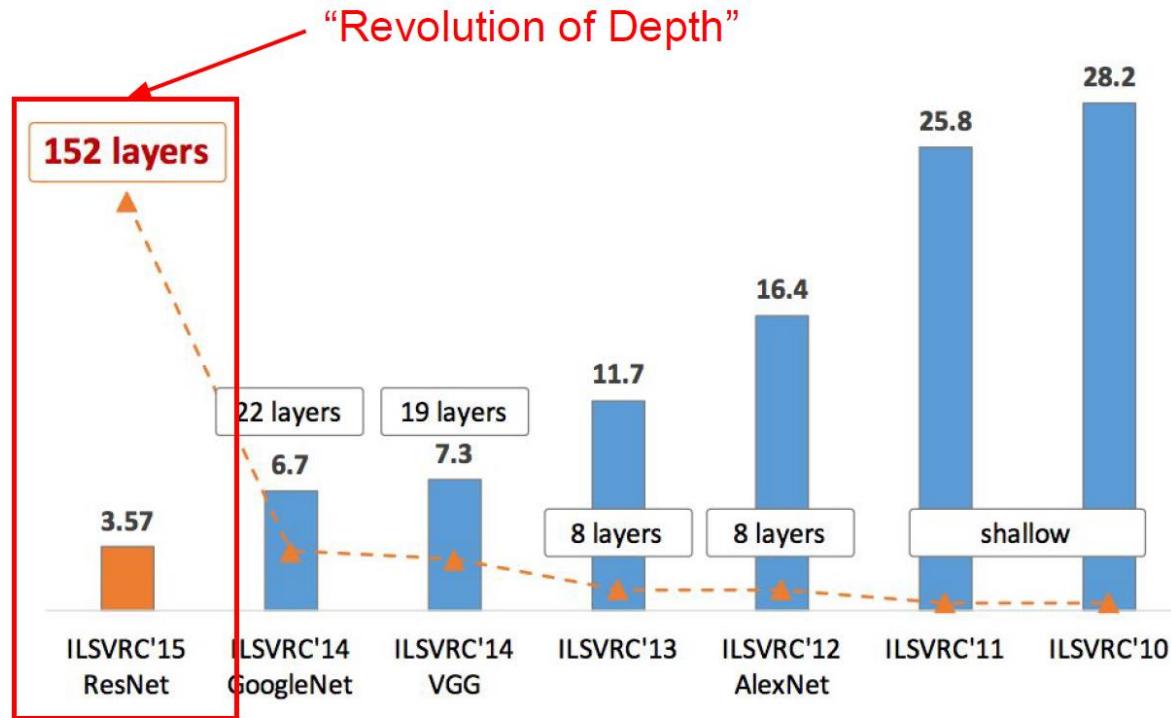


Figure copyright Kaiming He, 2016. Reproduced with permission.

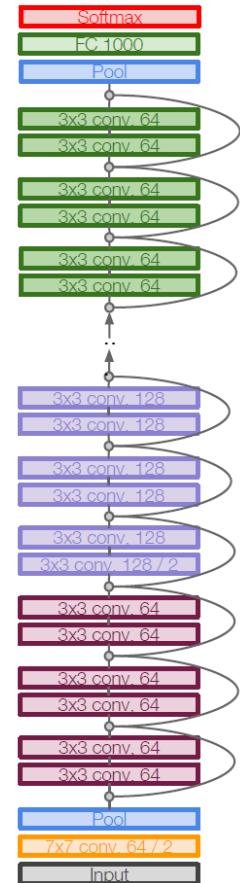
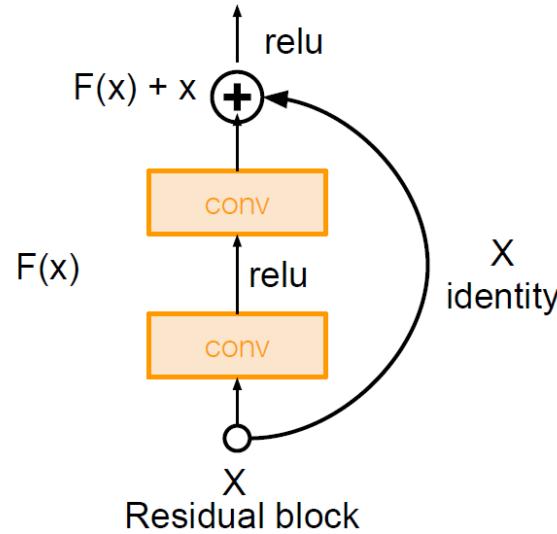
ResNet

Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

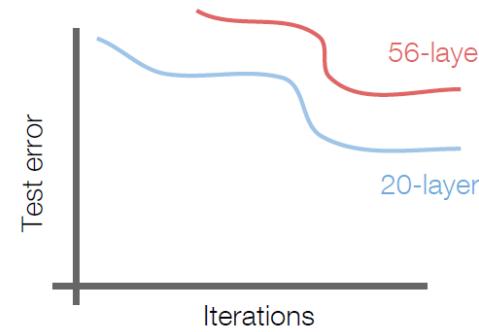
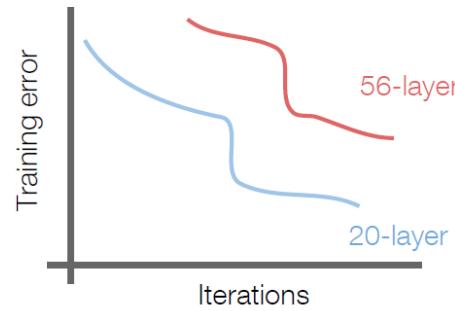
- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



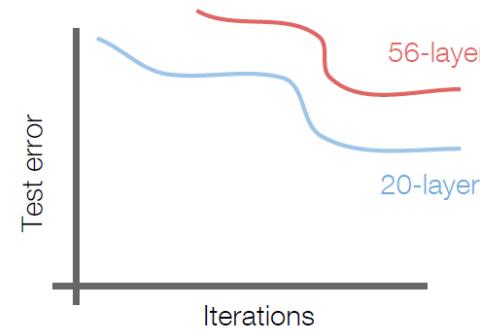
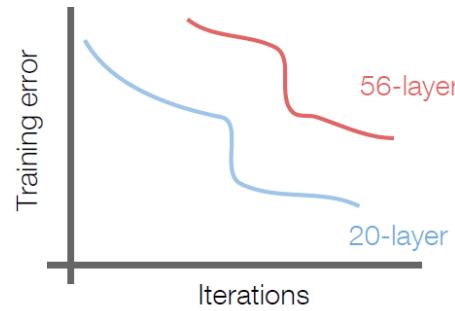
Q: What's strange about these training and test curves?

[Hint: look at the order of the curves]

Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it's not caused by overfitting!

Case Study: ResNet

[He et al., 2015]

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

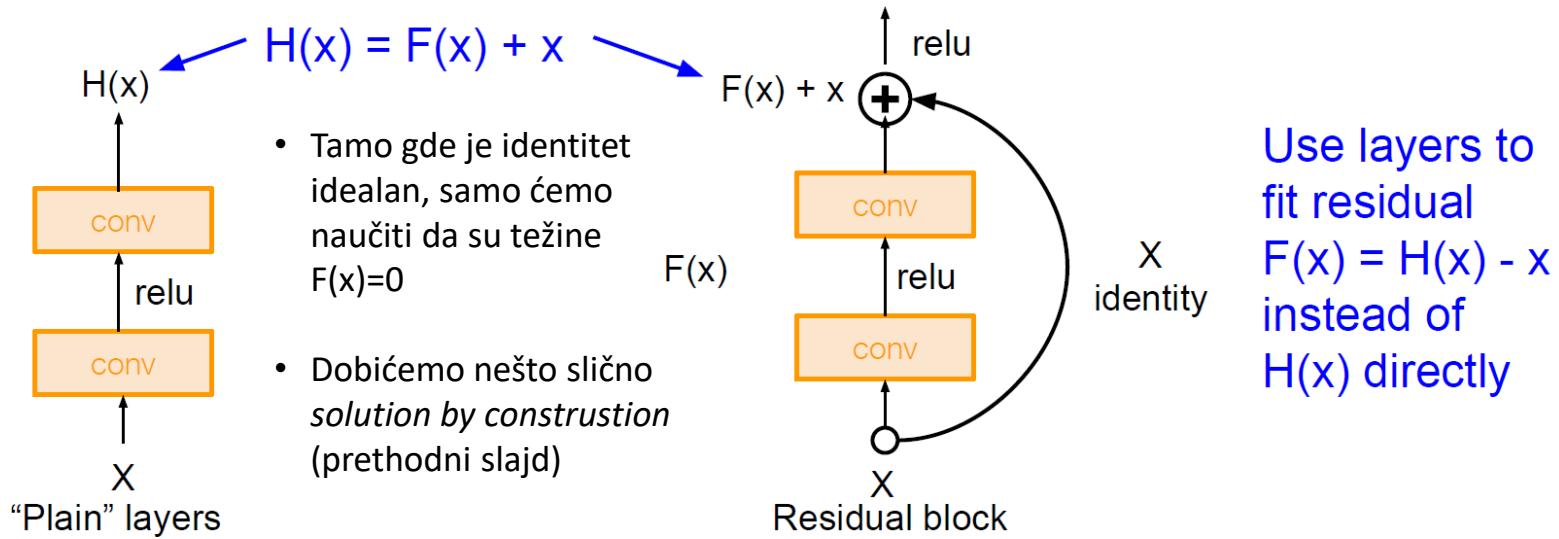
The deeper model should be able to perform at least as well as the shallower model.

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



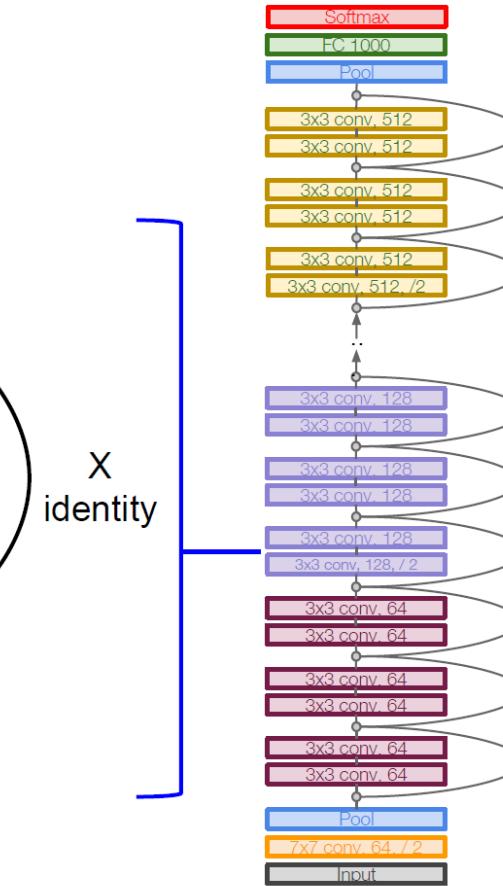
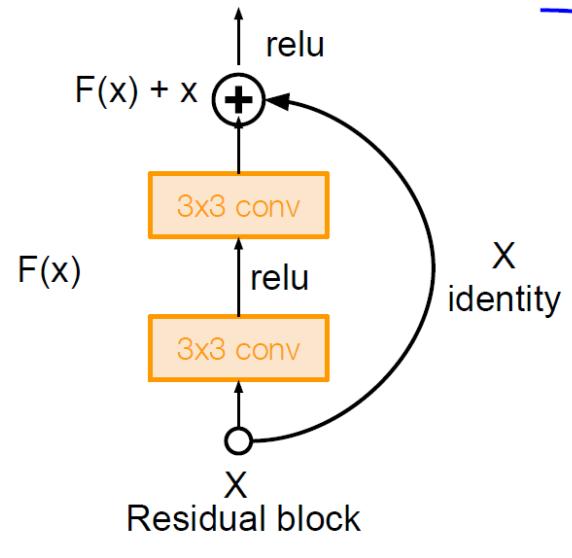
ResNet

Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers



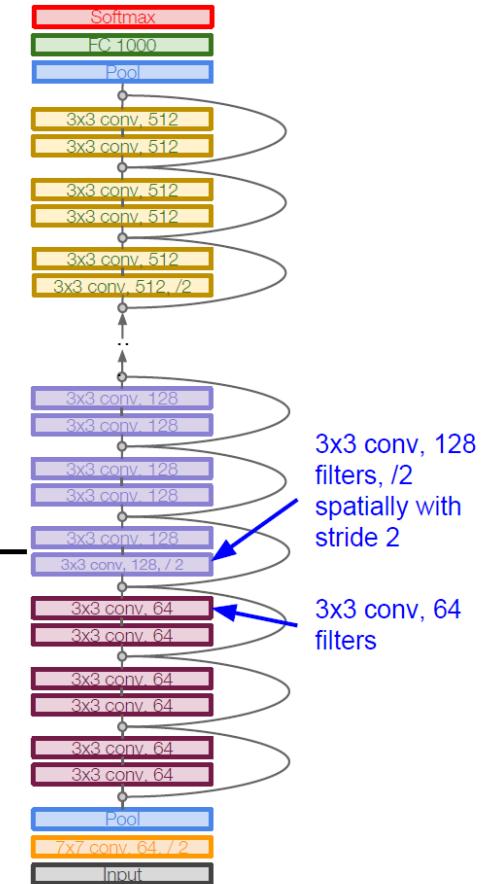
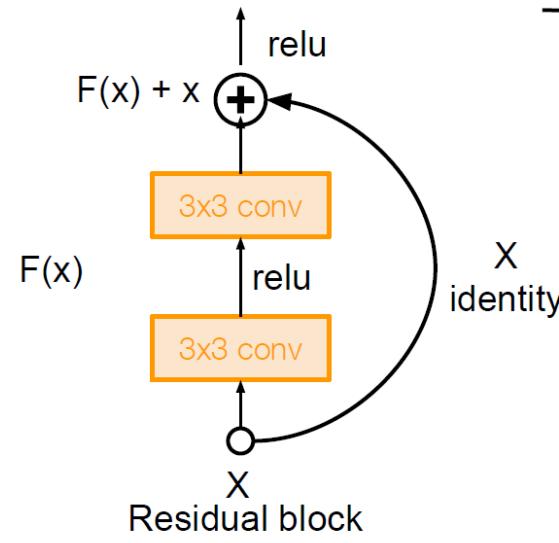
ResNet

Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
 - Every residual block has two 3x3 conv layers
 - Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)



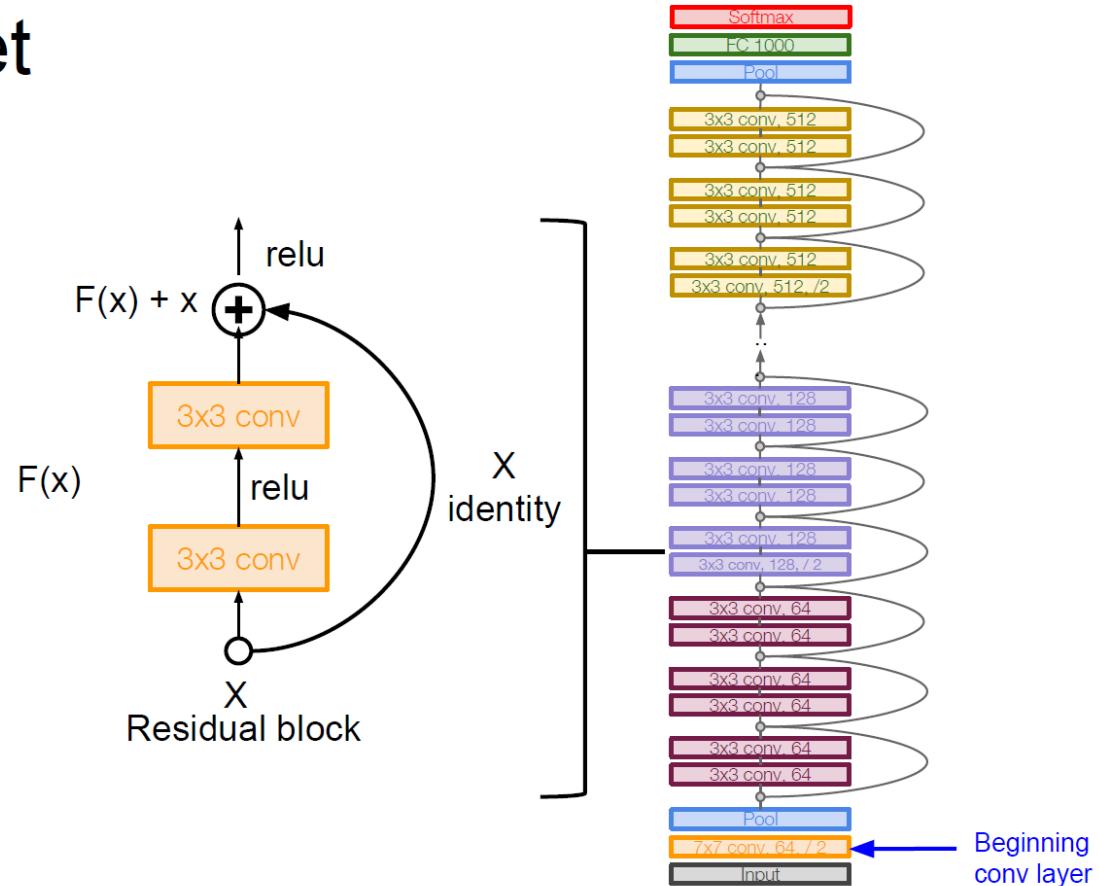
ResNet

Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning



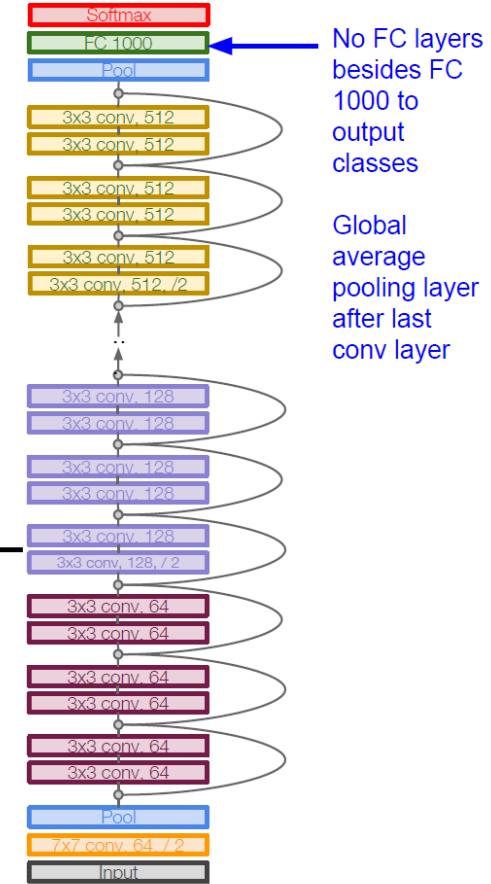
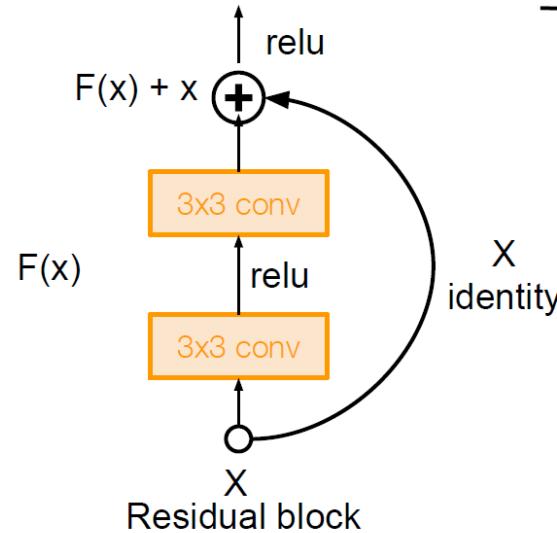
ResNet

Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)



ResNet

Case Study: ResNet

[He et al., 2015]

Total depths of 34, 50, 101, or
152 layers for ImageNet

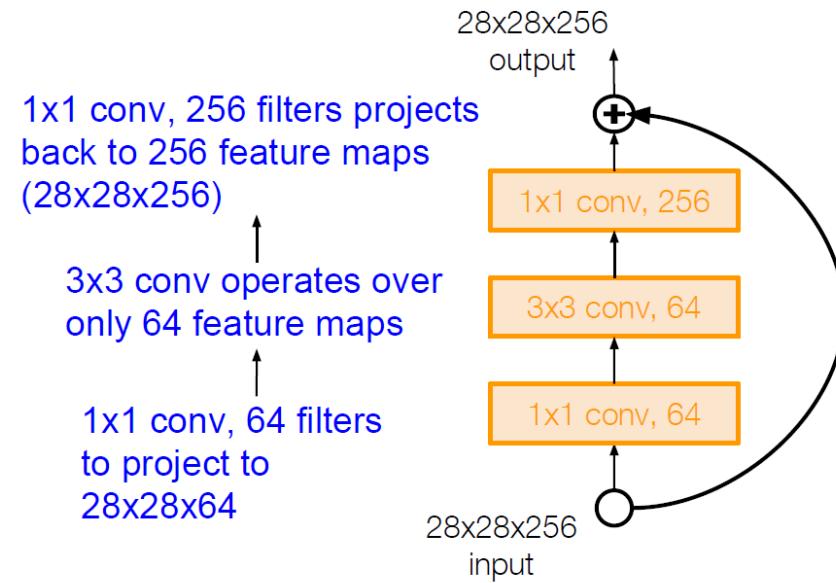


ResNet

Case Study: ResNet

[He et al., 2015]

For deeper networks
(ResNet-50+), use “bottleneck”
layer to improve efficiency
(similar to GoogLeNet)



Case Study: ResNet

[He et al., 2015]

Training ResNet in practice:

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

Case Study: ResNet

[He et al., 2015]

Experimental Results

- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lower training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

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

ILSVRC 2015 classification winner (3.6% top 5 error) -- better than “human performance”! (Russakovsky 2014)

ResNet

- Dodatni materijali:
 - <https://www.youtube.com/watch?v=1PGLj-uKT1w> i Slajdovi
 - <https://github.com/gcr/torch-residual-networks>
- State-of-the-art CNN model koji se koristi u praksi

ResNet

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

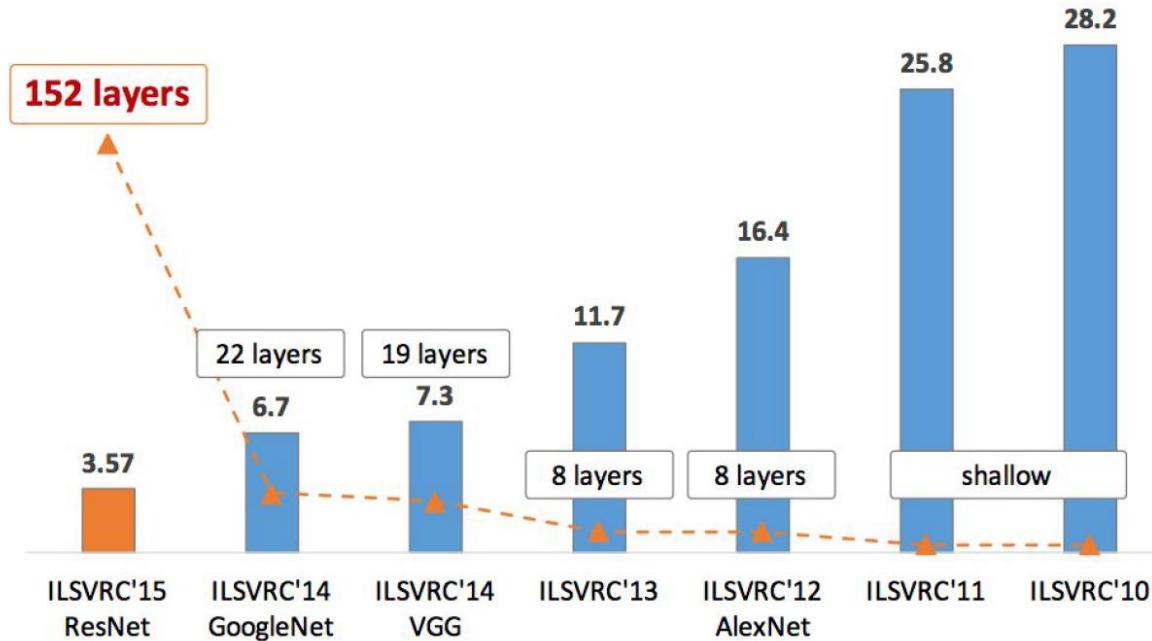
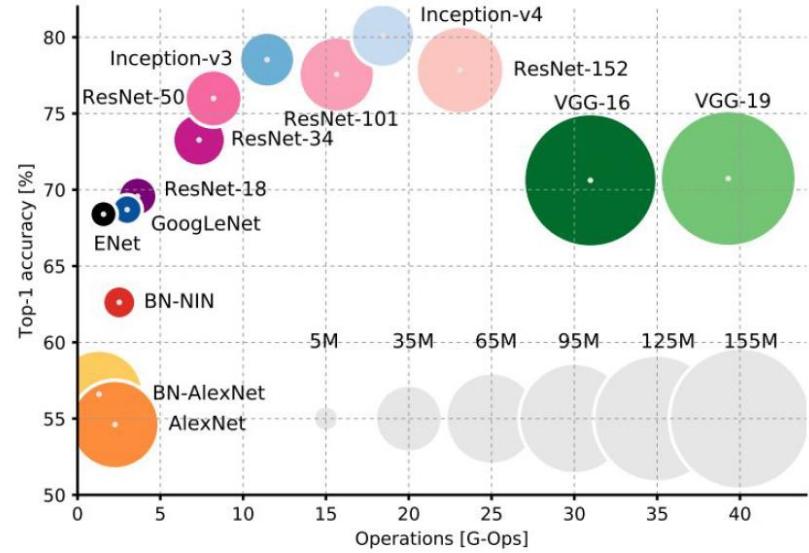
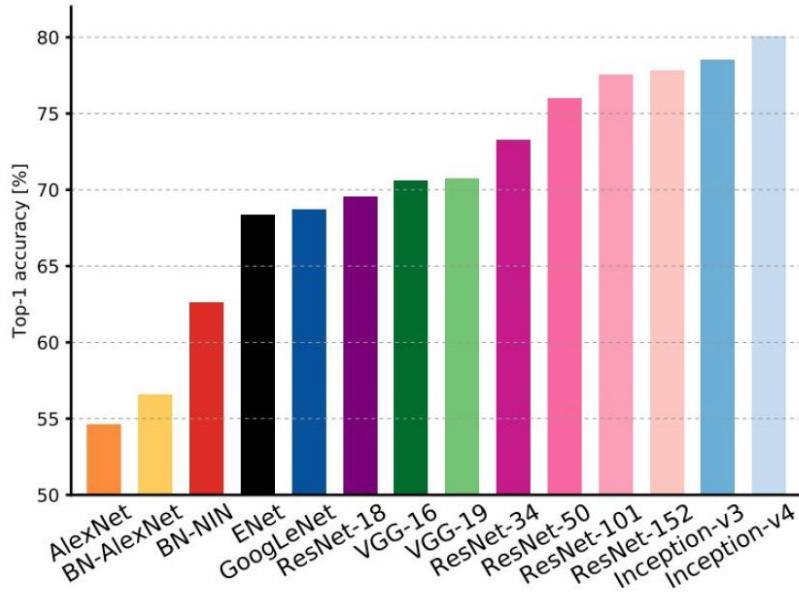


Figure copyright Kaiming He, 2016. Reproduced with permission.

Poređenje

Comparing complexity...



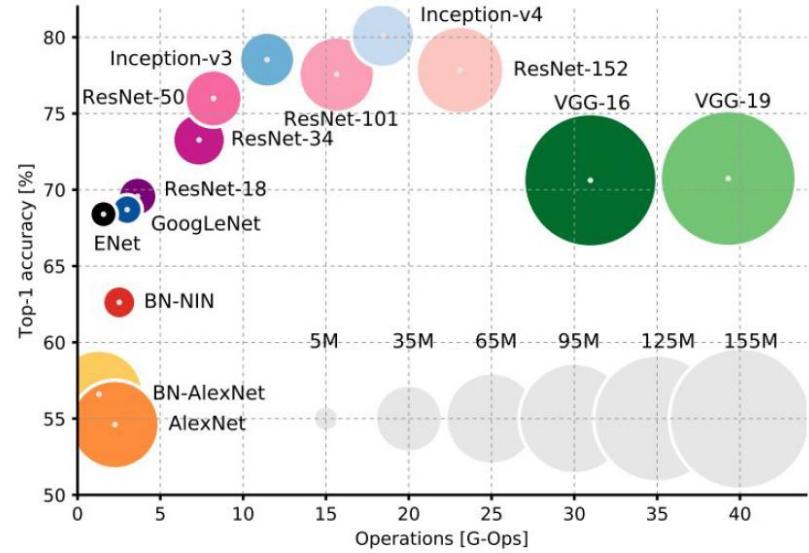
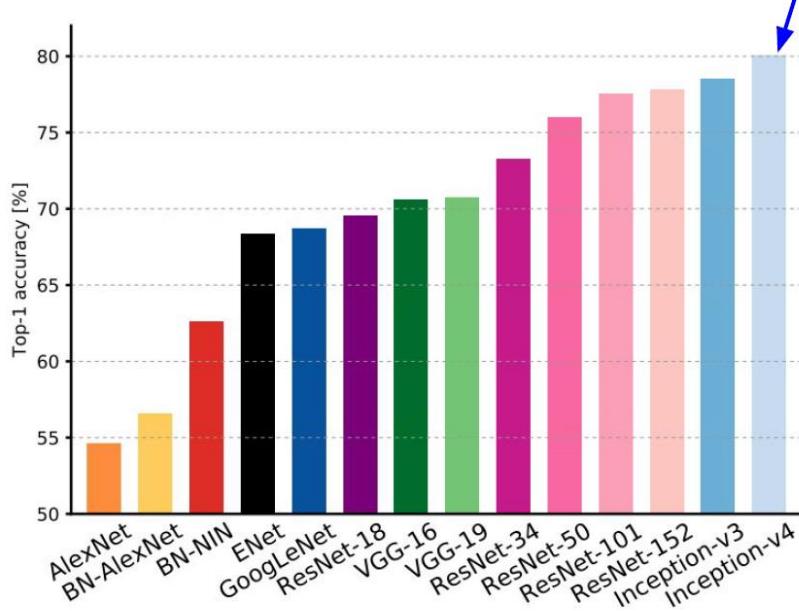
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Poređenje

Comparing complexity...

Inception-v4: Resnet + Inception!

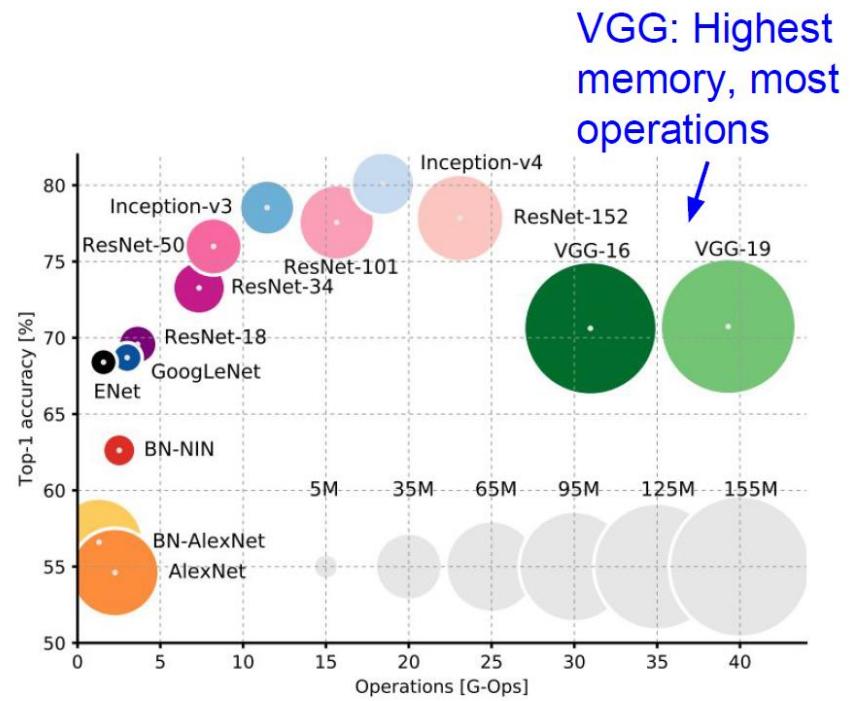
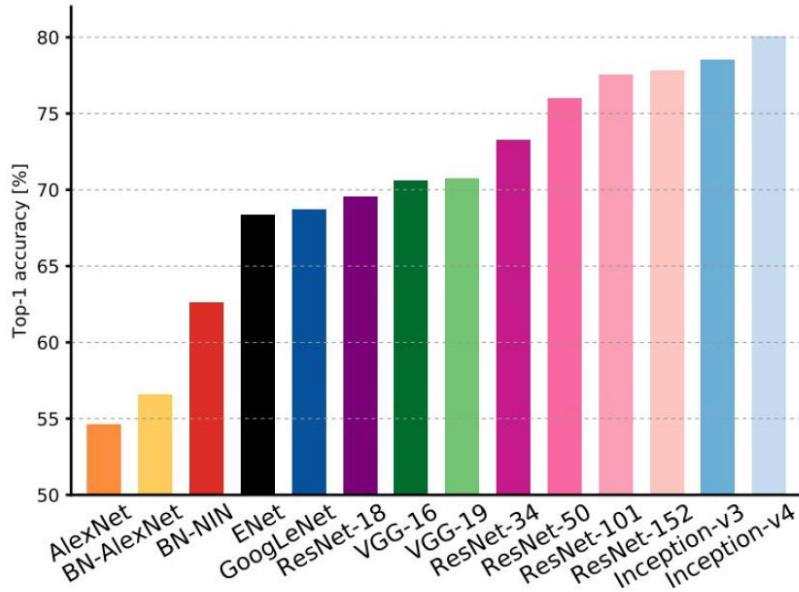


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Poređenje

Comparing complexity...

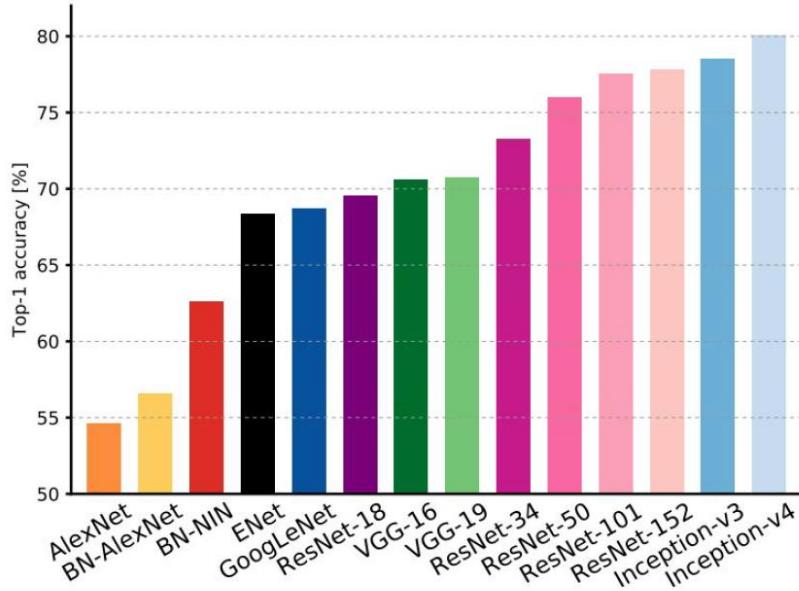


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

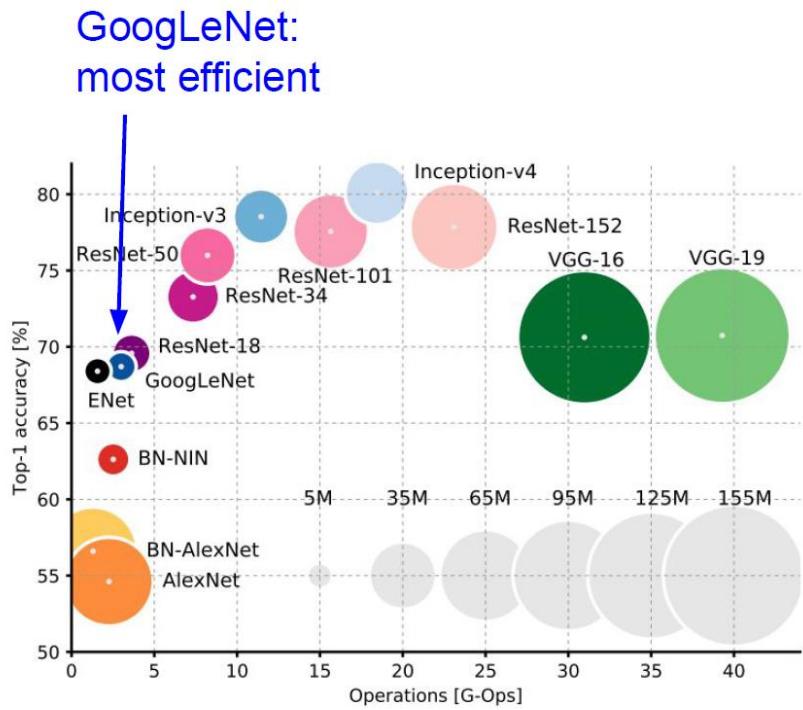
Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Poređenje

Comparing complexity...



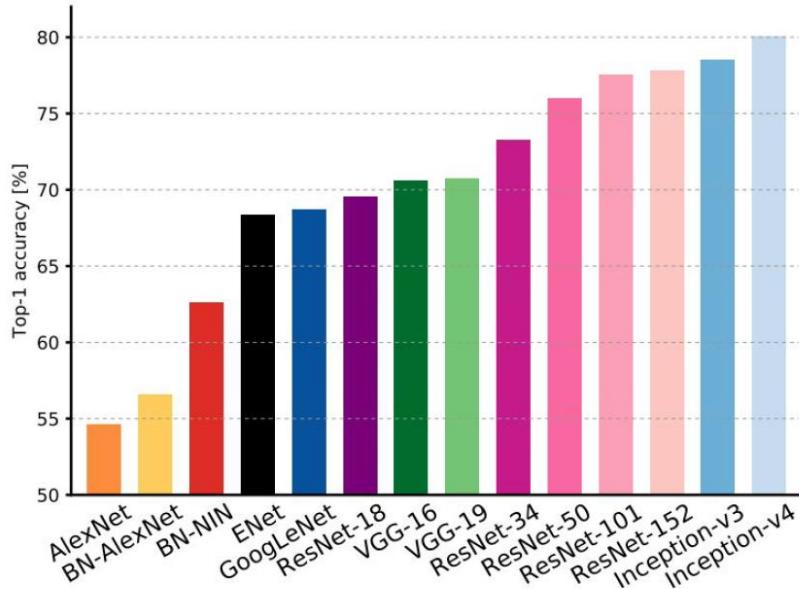
An Analysis of Deep Neural Network Models for Practical Applications, 2017.



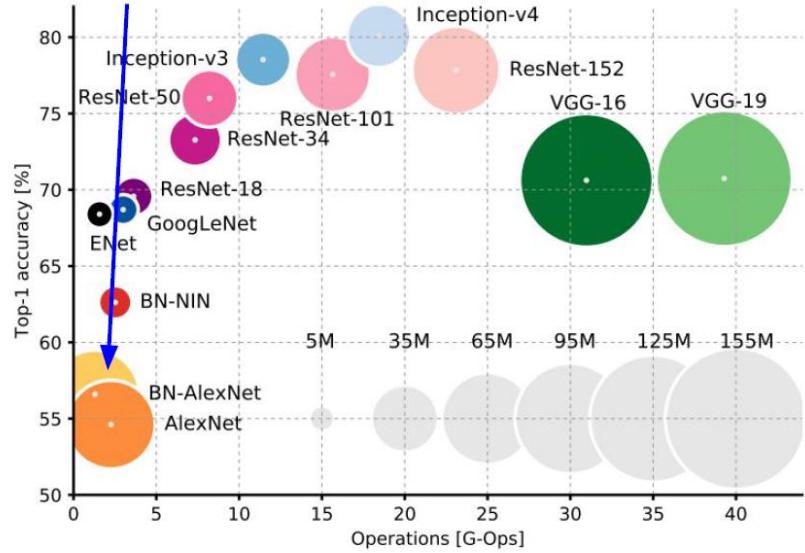
Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Poređenje

Comparing complexity...



AlexNet:
Smaller compute, still memory
heavy, lower accuracy

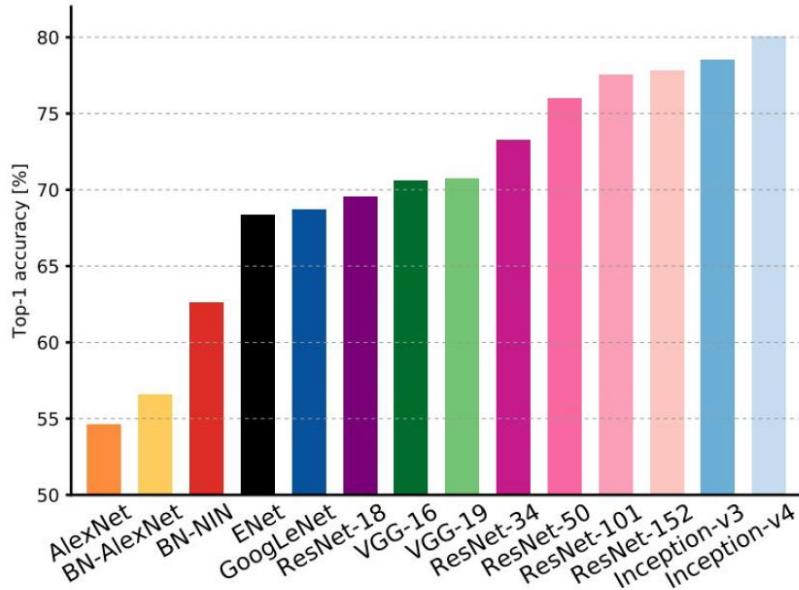


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

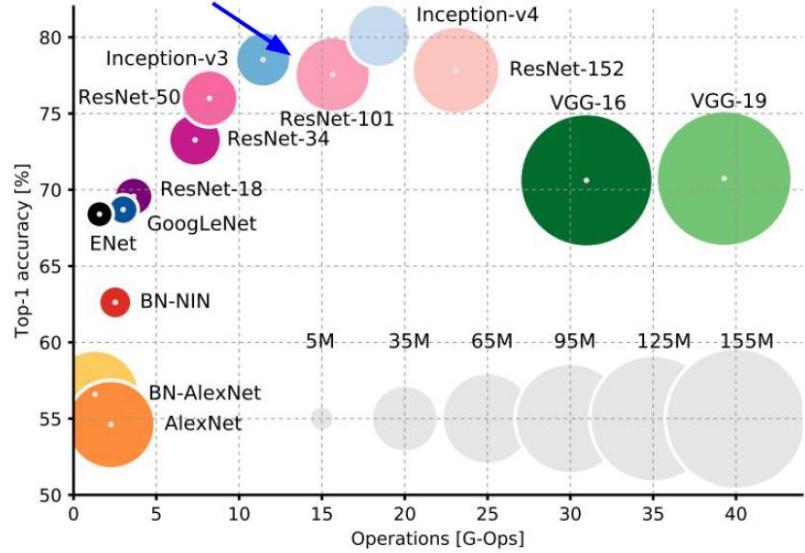
Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Poređenje

Comparing complexity...



ResNet:
Moderate efficiency depending on
model, highest accuracy

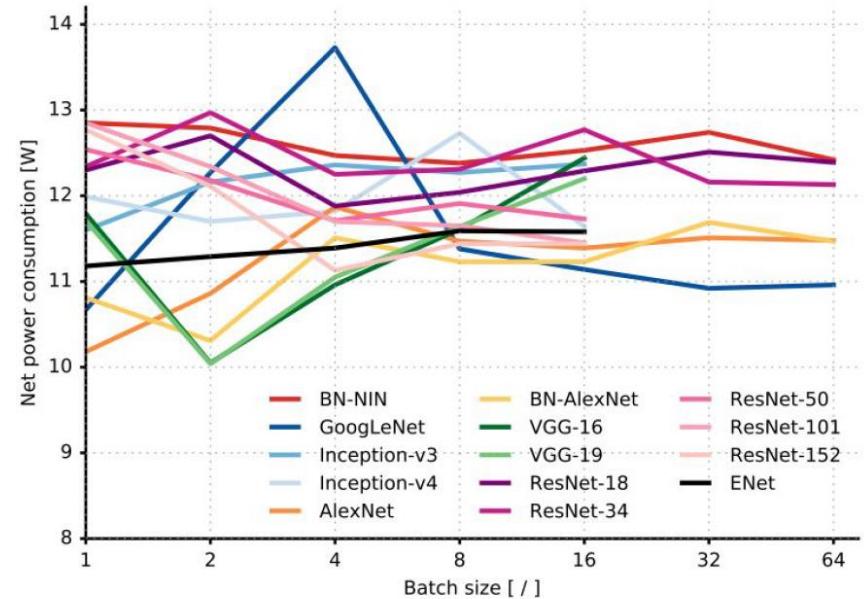
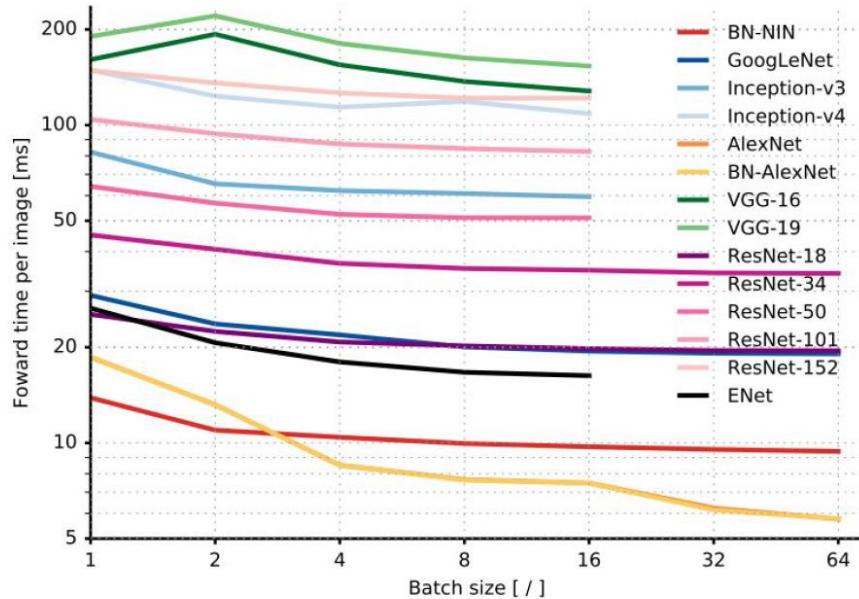


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Poređenje

Forward pass time and power consumption



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

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

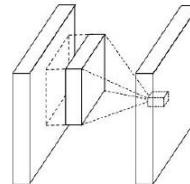
Ostale arhitekture

Network in Network (NiN)

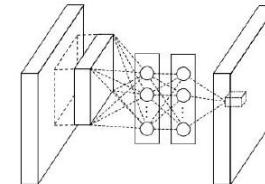
[Lin et al. 2014]

- Mlpconv layer with “micronetwork” within each conv layer to compute more abstract features for local patches
- Micronetwork uses multilayer perceptron (FC, i.e. 1x1 conv layers)
- Precursor to GoogLeNet and ResNet “bottleneck” layers
- Philosophical inspiration for GoogLeNet

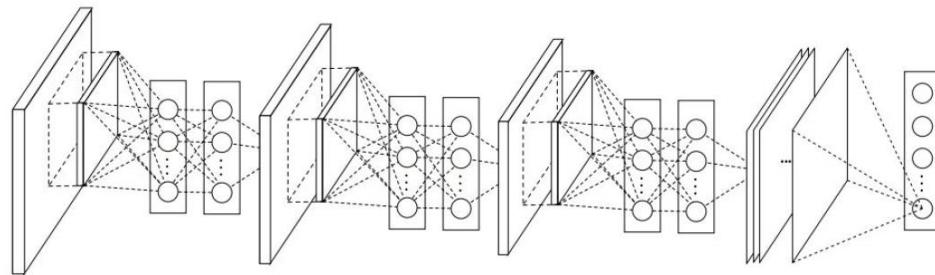
- NiN predlaže ideju MLP conv slojeva: unutar svakog CONV sloja stekujemo MLP sa nekoliko potpuno povezanih slojeva na vrh standardne konvolucije i tako da izračunamo apstraktnija obeležja za lokalne isečke
- Dakle, umesto da prevlačimo konvolucione filtere, prevlačimo ove kompleksnije hijerarhijske filtere kako bismo dobili aktivacione mape



(a) Linear convolution layer



(b) Mlpconv layer



Figures copyright Lin et al., 2014. Reproduced with permission.

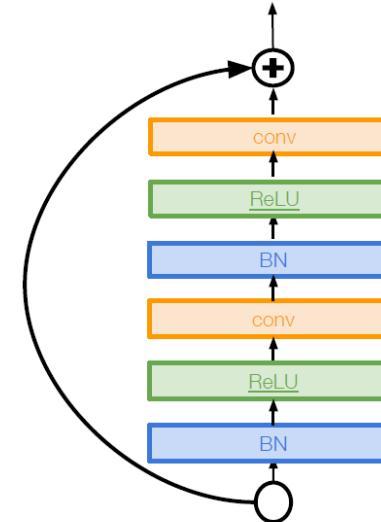
Ostale arhitekture

Improving ResNets...

Identity Mappings in Deep Residual Networks

[He et al. 2016]

- Improved ResNet block design from creators of ResNet
- Creates a more direct path for propagating information throughout network (moves activation to residual mapping pathway)
- Gives better performance



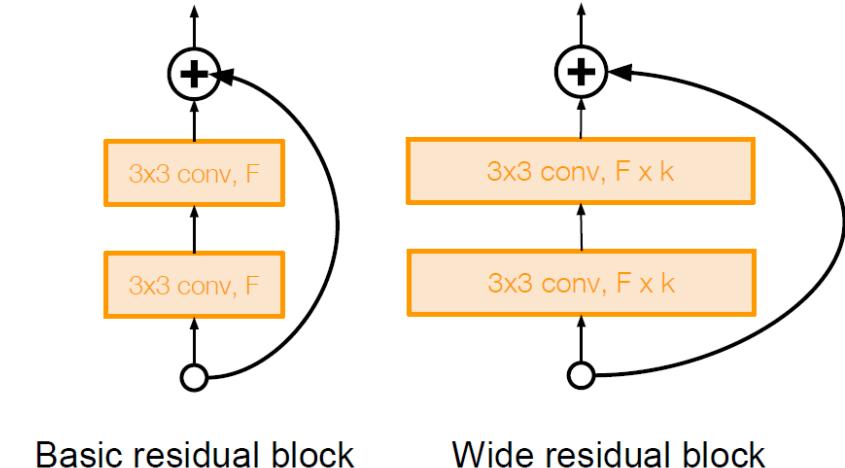
Ostale arhitekture

Improving ResNets...

Wide Residual Networks

[Zagoruyko et al. 2016]

- Argues that residuals are the important factor, not depth
- Use wider residual blocks ($F \times k$ filters instead of F filters in each layer)
- 50-layer wide ResNet outperforms 152-layer original ResNet
- Increasing width instead of depth more computationally efficient (parallelizable)



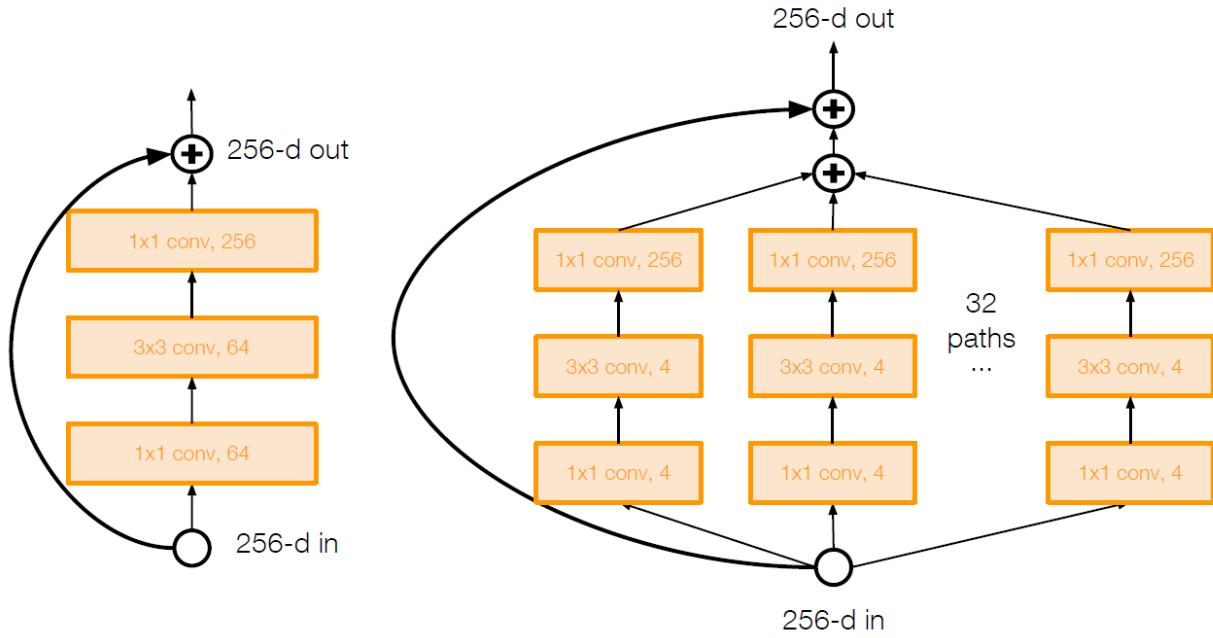
Ostale arhitekture

Improving ResNets...

Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

[Xie et al. 2016]

- Also from creators of ResNet
- Increases width of residual block through multiple parallel pathways (“cardinality”)
- Parallel pathways similar in spirit to Inception module



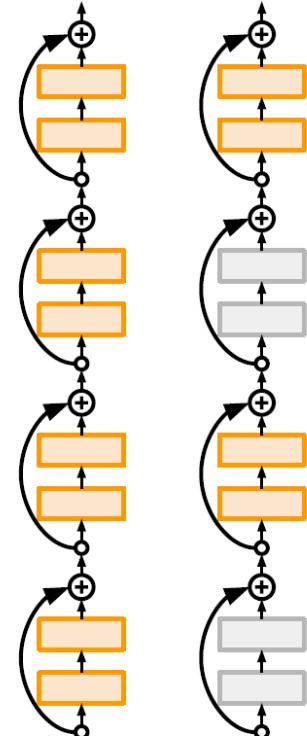
Ostale arhitekture

Improving ResNets...

Deep Networks with Stochastic Depth

[Huang et al. 2016]

- Motivation: reduce vanishing gradients and training time through short networks during training
 - Randomly drop a subset of layers during each training pass
 - Bypass with identity function
 - Use full deep network at test time
-
- Ideja: imamo kratke mreže tokom treniranja: *drop*-ujemo neki poskup slojeva tokom treniranja (*drop*-uju se težine i zamene *identity* vezom)
 - Kratke mreže: možemo bolje da prosledimo gradijent, efikasnije
 - Liči na dropout



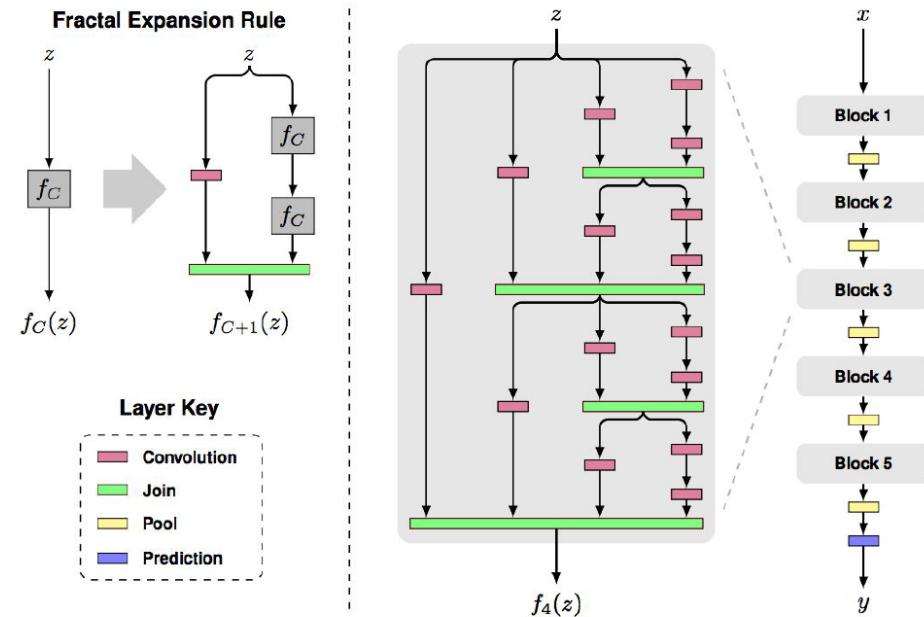
Ostale arhitekture

Beyond ResNets...

FractalNet: Ultra-Deep Neural Networks without Residuals

[Larsson et al. 2017]

- Argues that key is transitioning effectively from shallow to deep and residual representations are not necessary
- Fractal architecture with both shallow and deep paths to output
- Trained with dropping out sub-paths
- Full network at test time



Figures copyright Larsson et al., 2017. Reproduced with permission.

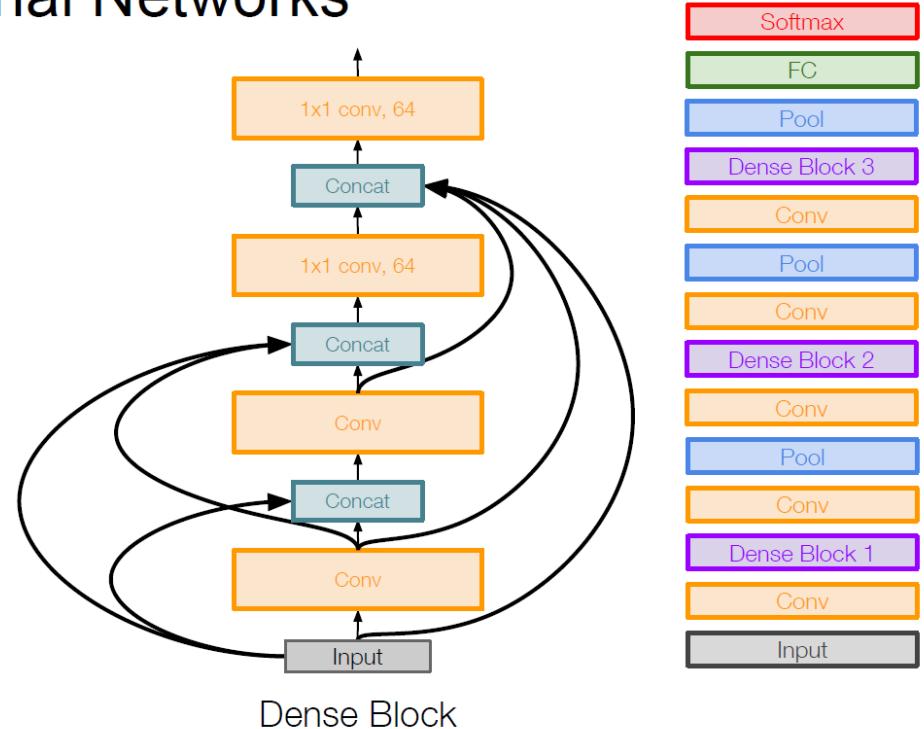
Ostale arhitekture

Beyond ResNets...

Densely Connected Convolutional Networks

[Huang et al. 2017]

- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse



Ostale arhitekture

Efficient networks...

SqueezeNet: AlexNet-level Accuracy With 50x Fewer Parameters and <0.5Mb Model Size

[Iandola et al. 2017]

- Fire modules consisting of a ‘squeeze’ layer with 1x1 filters feeding an ‘expand’ layer with 1x1 and 3x3 filters
- AlexNet level accuracy on ImageNet with 50x fewer parameters
- Can compress to 510x smaller than AlexNet (0.5Mb)

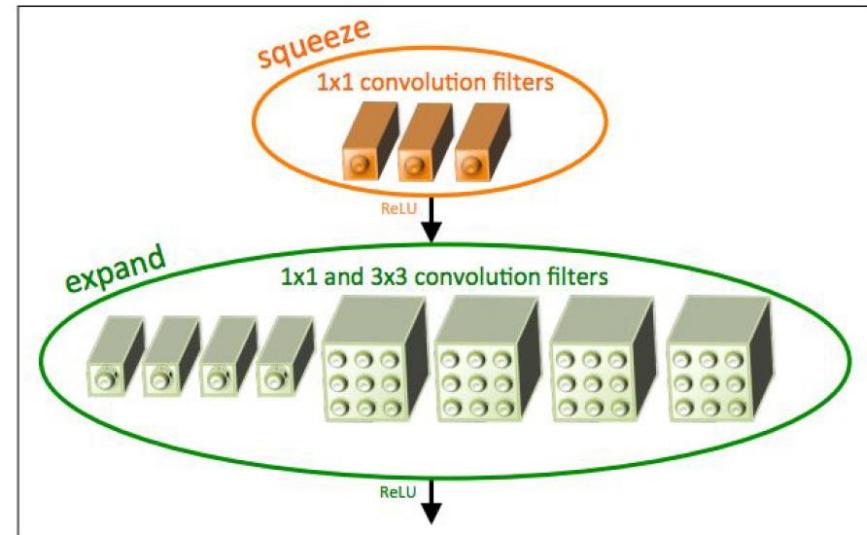


Figure copyright Iandola, Han, Moskewicz, Ashraf, Dally, Keutzer, 2017. Reproduced with permission.

Ostale arhitekture

- Ovo je bilo nekoliko ideja šta su alternative za ResNet, a da imaju uporedive ili bolje performanse
- Skorašnji rad ide dosta u smeru istraživanja kako su različiti slojevi povezani jedni sa drugim i kako se nosimo sa dubinom u ovim mrežama

Sumarizacija

Summary: CNN Architectures

Case Studies

- AlexNet
- VGG
- GoogLeNet
- ResNet

Also....

- NiN (Network in Network)
- Wide ResNet
- ResNeXT
- Stochastic Depth
- DenseNet
- FractalNet
- SqueezeNet

Summary: CNN Architectures

- VGG, GoogLeNet, ResNet all in wide use, available in model zoos
- ResNet current best default
- Trend towards extremely deep networks
- Significant research centers around design of layer / skip connections and improving gradient flow
- Even more recent trend towards examining necessity of depth vs. width and residual connections
- Next time: Recurrent neural networks

U praksi

- Umesto da isprobavate različite arhitekture za problem, preuzmite onu koja najbolje radi na *ImageNet*
 - Skinite pretrenirani model
 - *Fine-tune*-ujte ga za vaše podatke
- U retkim situacijama ćete morati da trenirate (ili dizajnirate) konvolucionu mrežu od početka