Neural Network and Deep Learning

Lecture5:

Convolutional Neural Networks (CNNs)

lecture inspiration:

Fei-Fei Li, Ehsan Adeli, Zane Durante 2024

Fei-Fei Li & Justin Johnson & Serena Yeung 2017

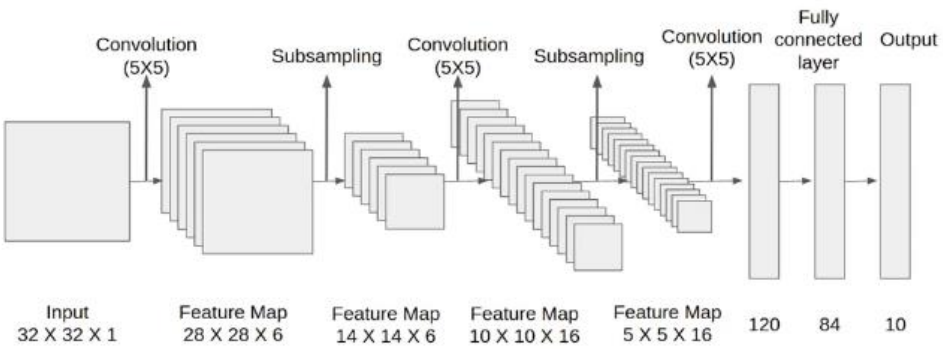
Some papers

OUTLINE

- CNN Architectures
- Weight Initialization
- Towards Principled Design of Deep Convolutional Networks
- Transfer Learning with CNNs

CNN ARCHITECTURES: LENET-5

- LeNet-5 (1998)
  - 7 layers
  - Input: 32×32 pixel image Gray level (one channel)
  - 60,000 trainable free parameters (weight sharing)
  - Mnist dataset→ 60,000 training examples , 10,000 test examples
  - Training time: 2 to 3 days with CPU



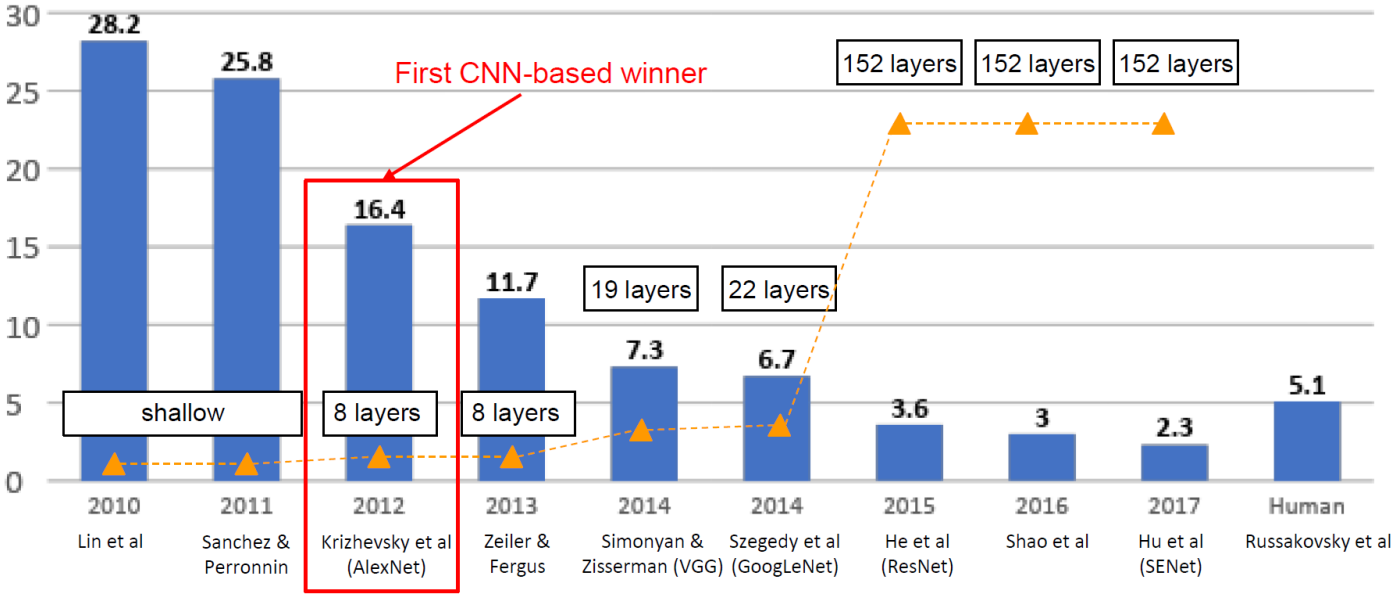
Layer	# filters / neurons	Filter size	Stride	Size of feature map	Activation function
Input	-	-	-	32 X 32 X 1	
Conv 1	6	5 * 5	1	28 X 28 X 6	tanh
Avg. pooling 1		2 * 2	2	14 X 14 X 6	
Conv 2	16	5 * 5	1	10 X 10 X 16	tanh
Avg. pooling 2		2 * 2	2	5 X 5 X 16	
Conv 3	120	5 * 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully Connected 2	-	-	-	10	Softmax

[LeCun et al., 1998]

CNN ARCHITECTURES

- ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

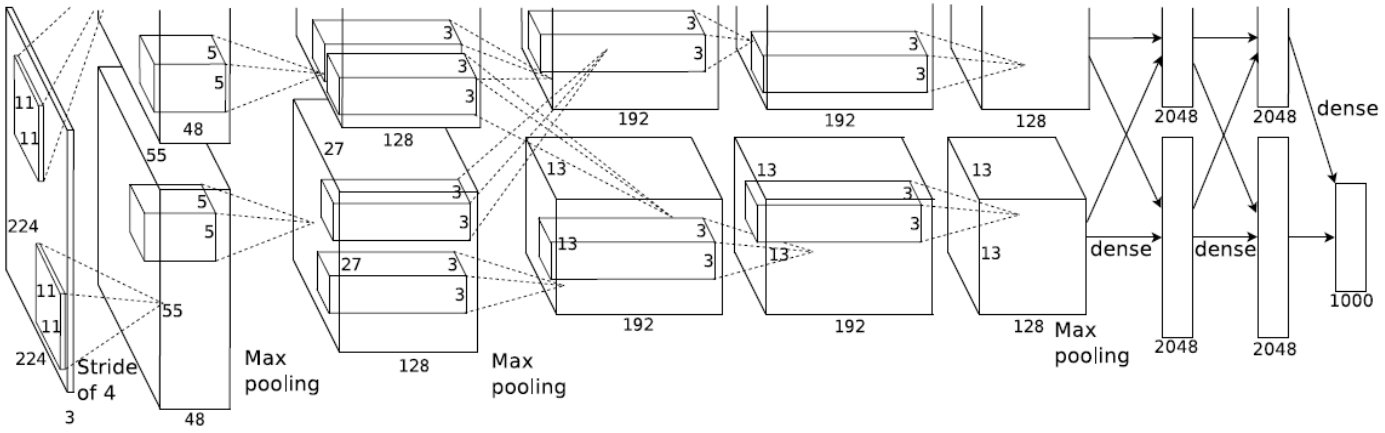
https://www.image-net.org/challenges/LSVRC/
https://www.kaggle.com/discussions/getting-started/149448



CNN ARCHITECTURES: ALEXNET

- **Case Study: AlexNet** *[Krizhevsky et al. 2012]*

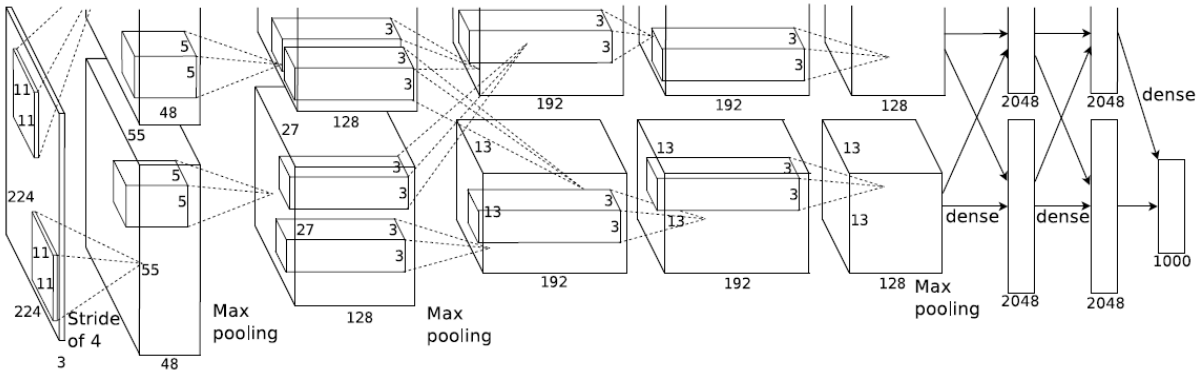
Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8



CNN ARCHITECTURES: ALEXNET

- **Case Study: AlexNet**

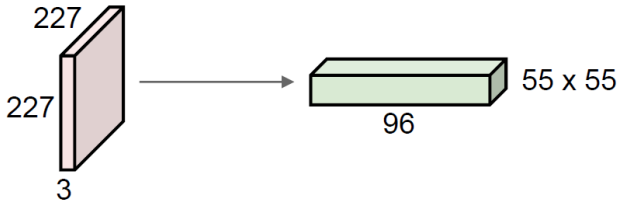
[Krizhevsky et al. 2012]



Input: 227x227x3 images
First layer (CONV1): 96 11x11 filters applied at stride 4 => W' = (W -F + 2P) / S + 1
Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Output volume **[55x55x96]**

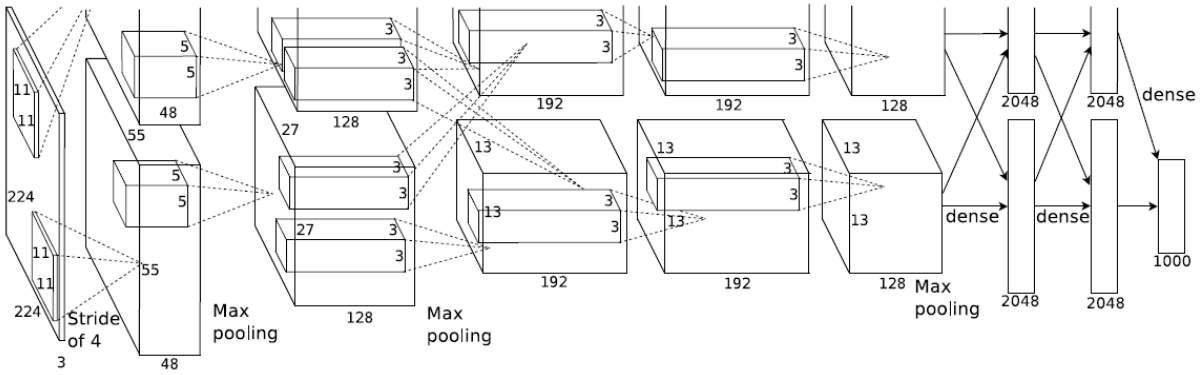
Parameters: (11*11*3 + 1)*96 = 35K



CNN ARCHITECTURES: ALEXNET

- **Case Study: AlexNet**

[Krizhevsky et al. 2012]



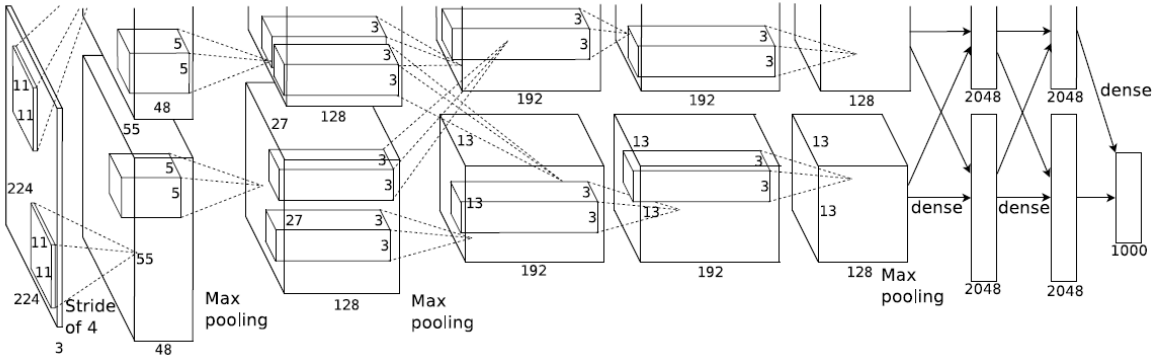
Input: 227x227x3 images
After CONV1: 55x55x96

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

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

CNN ARCHITECTURES: ALEXNET

- **Case Study: AlexNet**



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)

CNN ARCHITECTURES: ALEXNET

Case Study: AlexNet

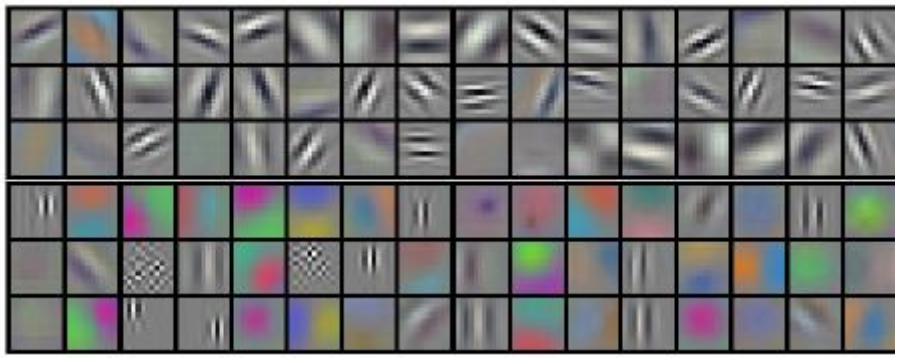
Details/Retrospectives:

- first use of ReLU
- GTX 580 GPU has only 3GB of memory
- 1.2 million training examples, 60 million parameters
- spreading the net across two GPUs
- half of the kernels (or neurons) on each GPU
- heavy data augmentation:
  1. Generating image translations and horizontal reflections.
  2. Altering the intensities of the RGB channels in training images.
- 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

- In terms of training time with gradient descent, these saturating nonlinearities are much slower than the non-saturating nonlinearity
- Deep convolutional neural networks with **ReLUs** train several times faster than their equivalents with tanh units

CNN ARCHITECTURES: ALEXNET

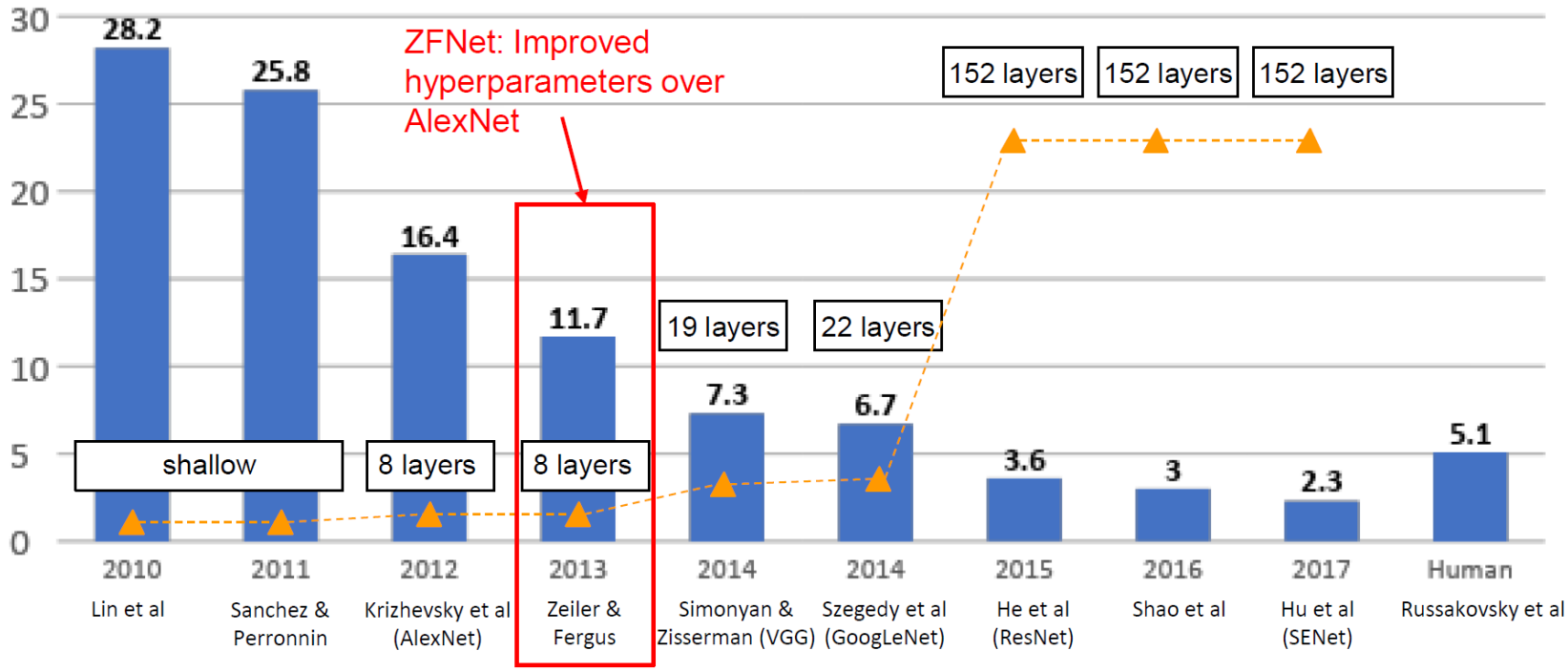
- **Case Study: AlexNet**



96 convolutional kernels of size 11*11*3 learned by the first convolutional layer on the 224*224*3 input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2.

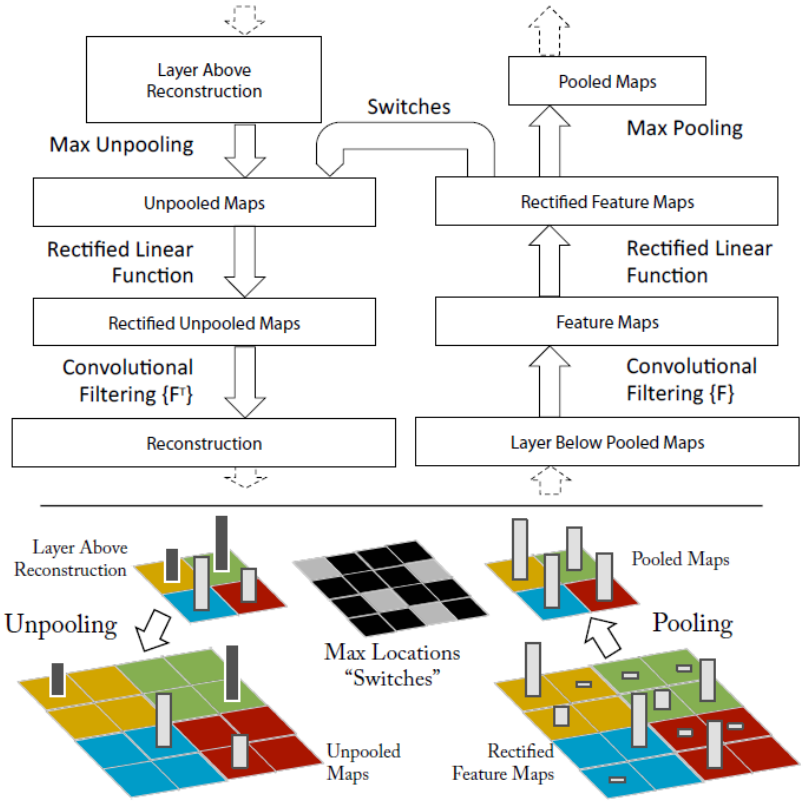
CNN ARCHITECTURES

- **ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners**



CNN ARCHITECTURES: ZFNET

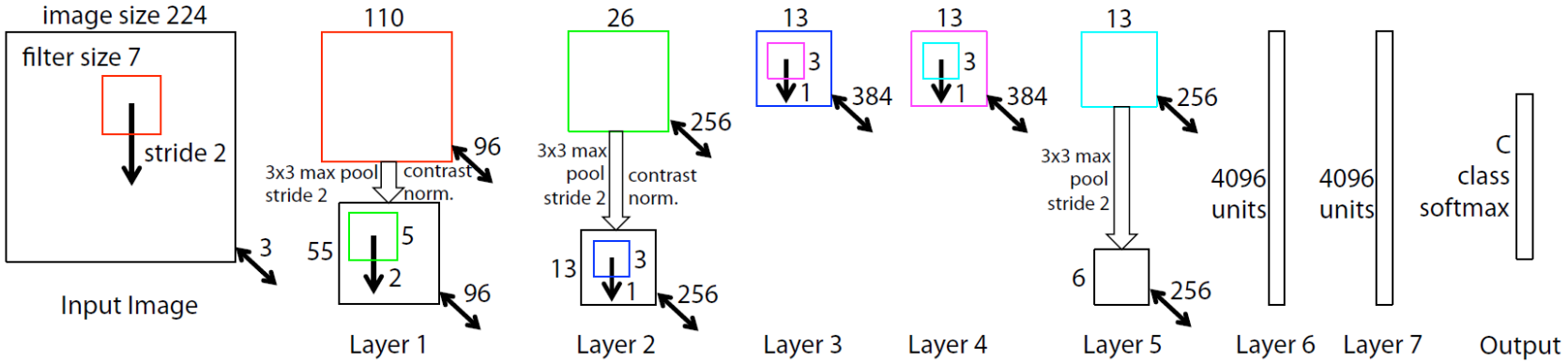
- **Visualizing and Understanding Convolutional Networks**



Zeiler and Fergus, 2013

CNN ARCHITECTURES:
ZFNET

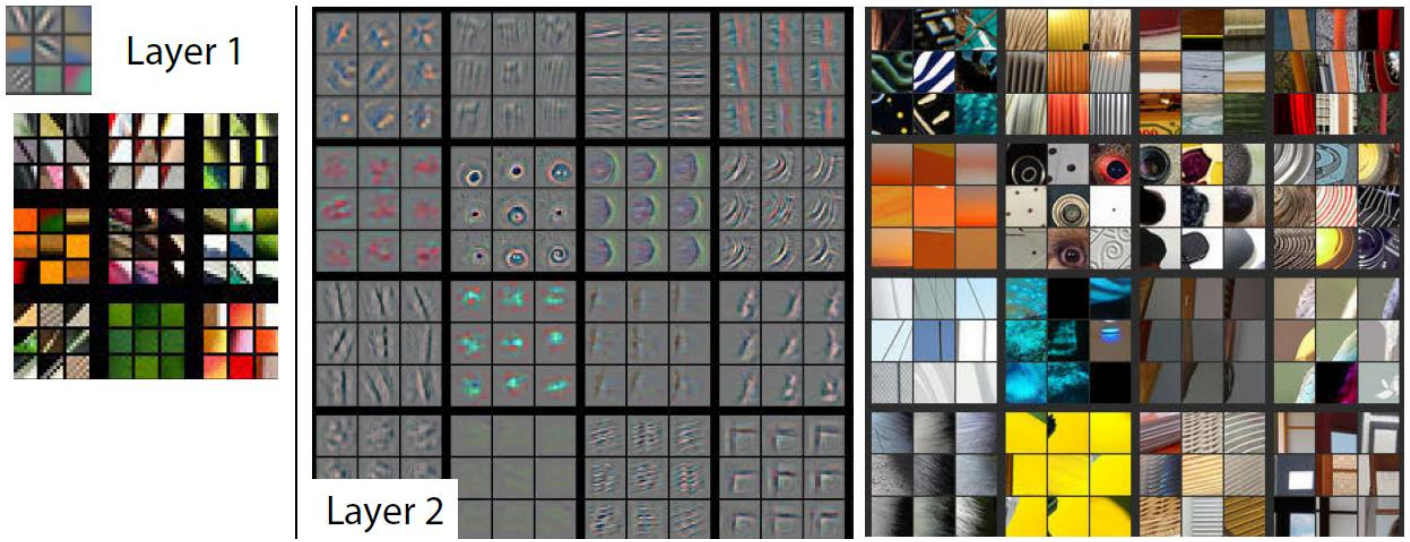
- **Visualizing and Understanding Convolutional Networks**



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
ImageNet top 5 error: 16.4% -> 11.7%

CNN ARCHITECTURES:
ZFNET

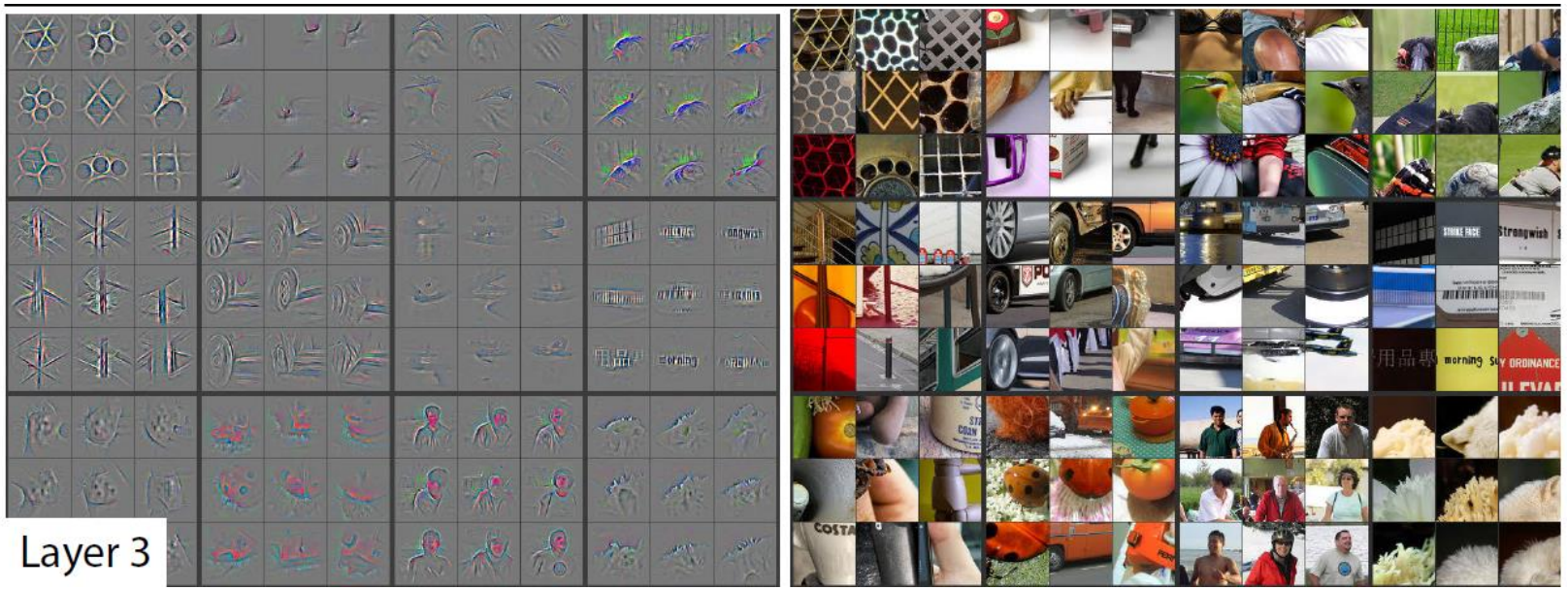
- **Visualizing and Understanding Convolutional Networks**



Top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using deconvolutional network approach.

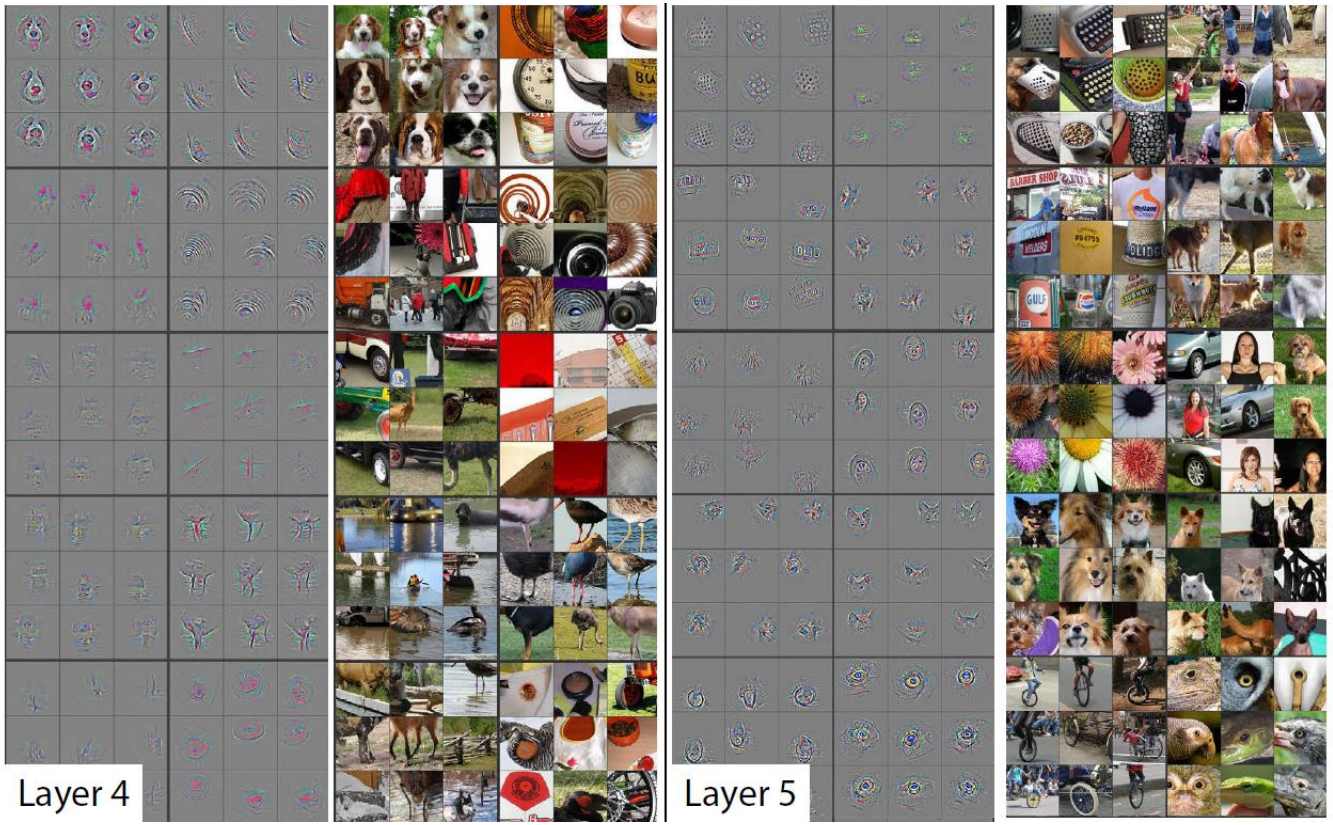
CNN ARCHITECTURES:
ZFNET

- **Visualizing and Understanding Convolutional Networks**

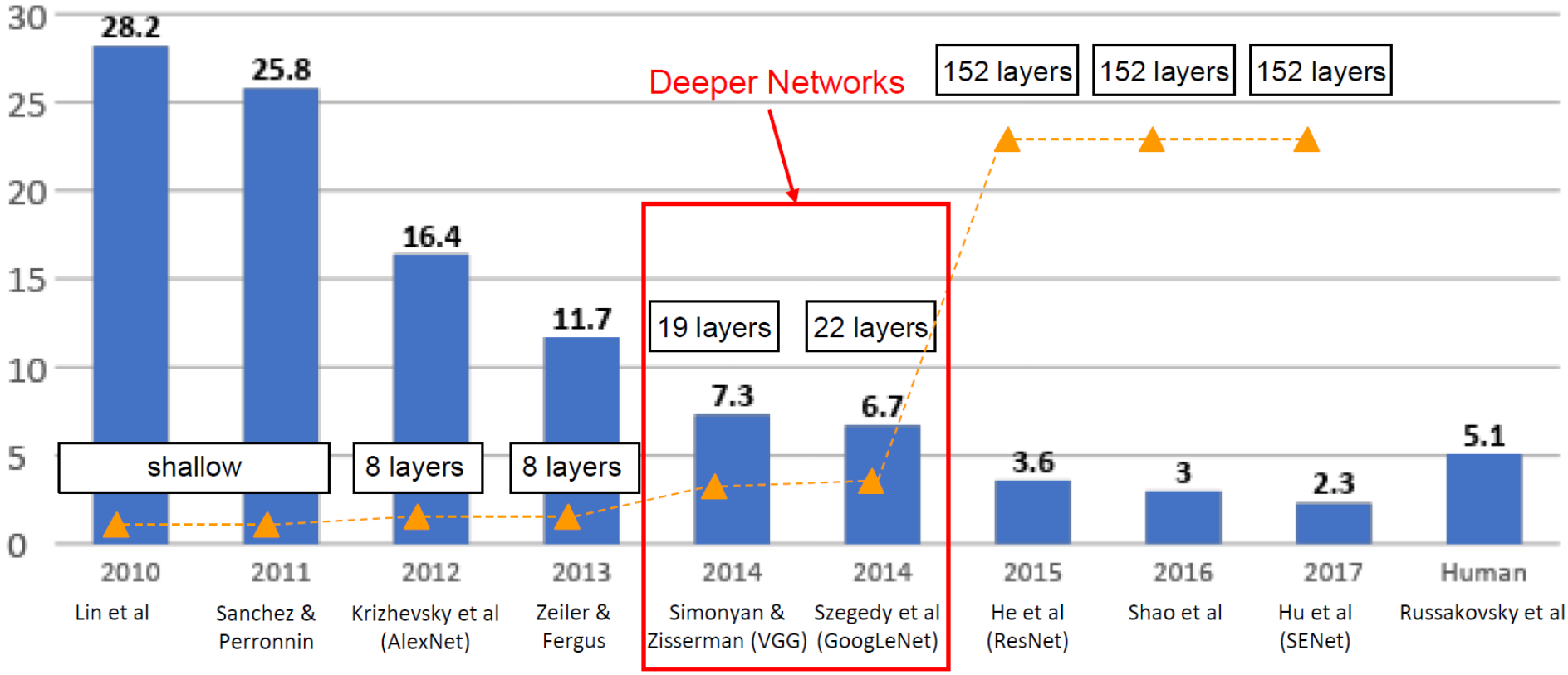


CNN ARCHITECTURES:
ZFNET

- **Visualizing and Understanding Convolutional Networks**



CNN ARCHITECTURES



CNN ARCHITECTURES: VGGNET

➢ **Case Study: VGGNet**

[Simonyan and Zisserman, 2014]

- Small filters (3*3), Deeper networks
- Input: 224 × 224 RGB image
- Preprocessing: subtracting the mean RGB value computed on the training set, from each pixel
- The convolution stride is fixed to 1 pixel
- The spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1 pixel for 3 × 3 conv. layers.
- Five max-pooling layers follow some of the conv. layers (not all the conv. layers are followed by max-pooling).
- Max-pooling is performed over a 2 × 2 pixel window, with stride 2.

CNN ARCHITECTURES: VGGNET

- **Case Study: VGGNet**

[Simonyan and Zisserman, 2014]

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

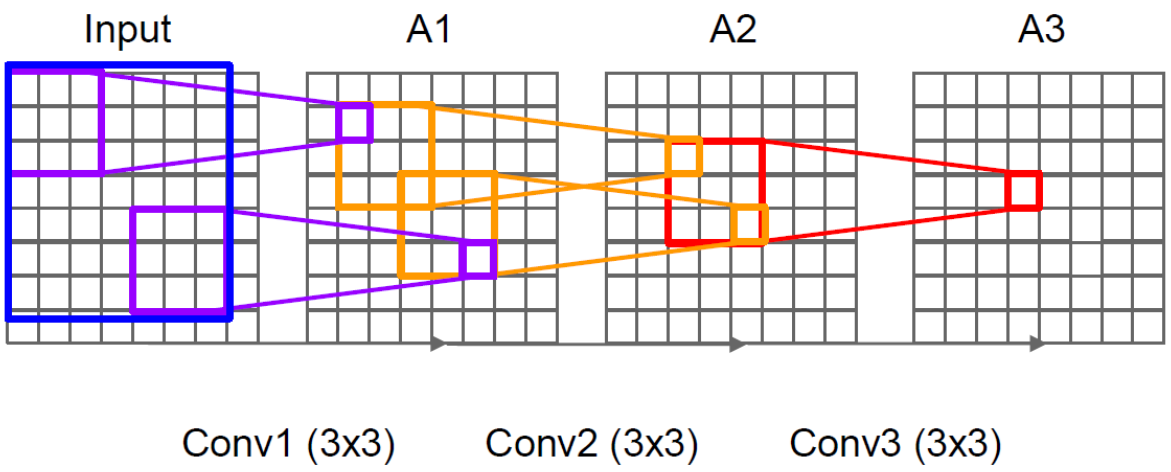
Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

CNN ARCHITECTURES: VGGNET

- **Case Study: VGGNet**

[Simonyan and Zisserman, 2014]

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



CNN ARCHITECTURES: VGGNET

➢ **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:

  $3 * (3^2C^2)$ vs 7^2C^2 for C channels per layer

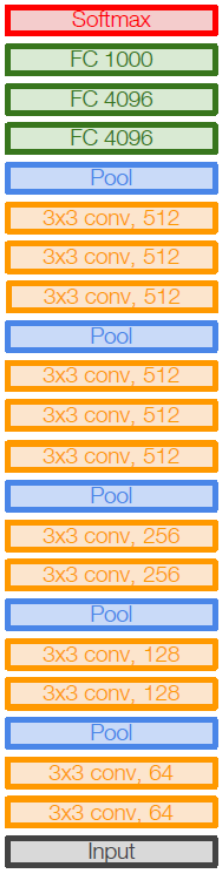
CNN ARCHITECTURES: VGGNET

Case Study: VGGNet

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [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

CNN ARCHITECTURES: VGGNET

Case Study: VGGNet

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [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

Note:

Most memory is in early CONV

Most params are in late FC

CNN ARCHITECTURES: VGGNET

- **Case Study: VGGNet**

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Number of parameters (in millions).

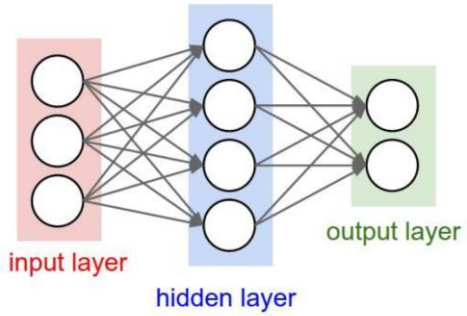
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

WEIGHT INITIALIZATION: ACTIVATION STATISTICS

- **Q: what happens when W=constant init is used?**
  - First idea: **Small random numbers**

(gaussian with zero mean and 1e-2 standard deviation)

```
Din = 3;
Dout = 5;
W = 0.01*np.random.randn(Din, Dout)
```



Works ~okay for small networks, but problems with deeper networks.

WEIGHT INITIALIZATION: ACTIVATION STATISTICS

- **What will happen to the activations for the last layer?**

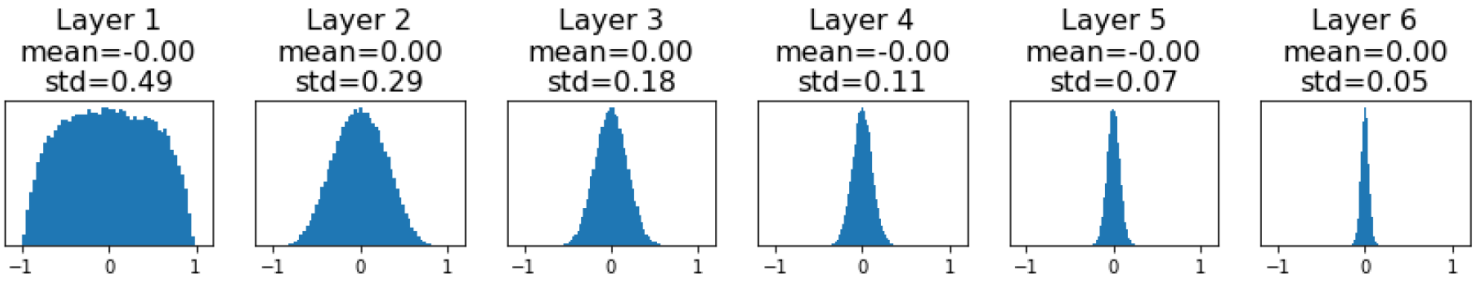
```
dims = [4096]*7
hs = []
x = 0.01*np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.01*np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

Forward pass for a 6-layer net with hidden size 4096

All activations tend to zero for deeper network layers

Q: What do the gradients dL/dW look like?

A: All zero, no learning



WEIGHT INITIALIZATION: ACTIVATION STATISTICS

- **What will happen to the activations for the last layer?**

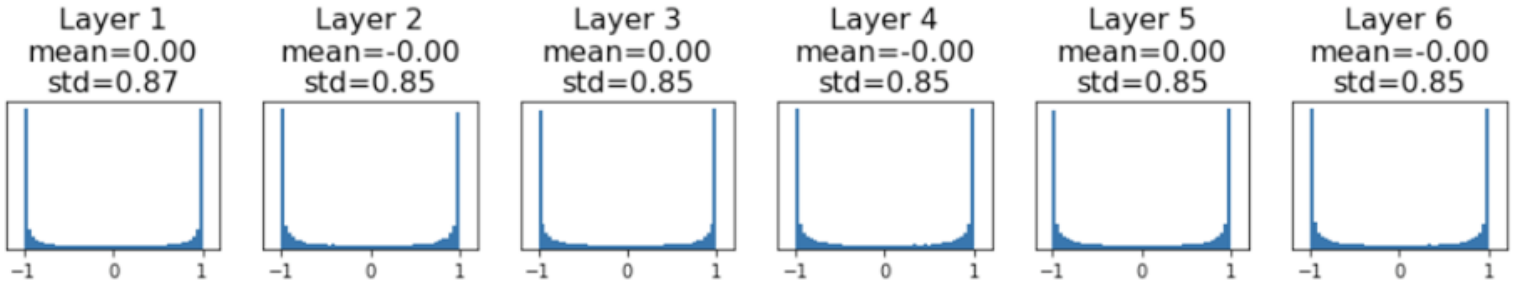
```
dims = [4096]*7
hs = []
x = 0.01*np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.05*np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

Increase std of initial weights from 0.01 to 0.05

What will happen to the activations for the last layer?

Q: What do the gradients look like?

A: Local gradients all zero, no learning



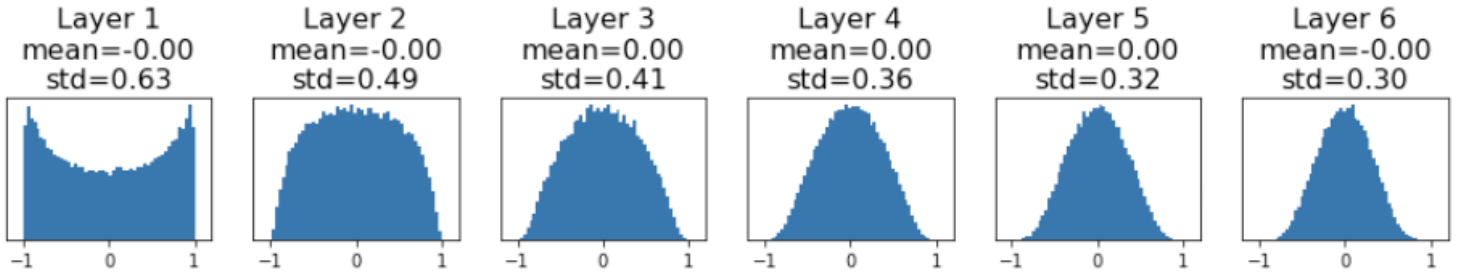
WEIGHT INITIALIZATION: "XAVIER" INITIALIZATION

- **What will happen to the activations for the last layer?**

```
dims = [4096]*7
hs = []
x = 0.01*np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout)/np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

"Just right": Activations are nicely scaled for all layers!

For conv layers, Din is filter_size^2* input_channels



Glorot and Bengio, "Understanding the difficulty of training deep feedforward neural networks", AISTAT 2010

WEIGHT INITIALIZATION: "XAVIER" INITIALIZATION

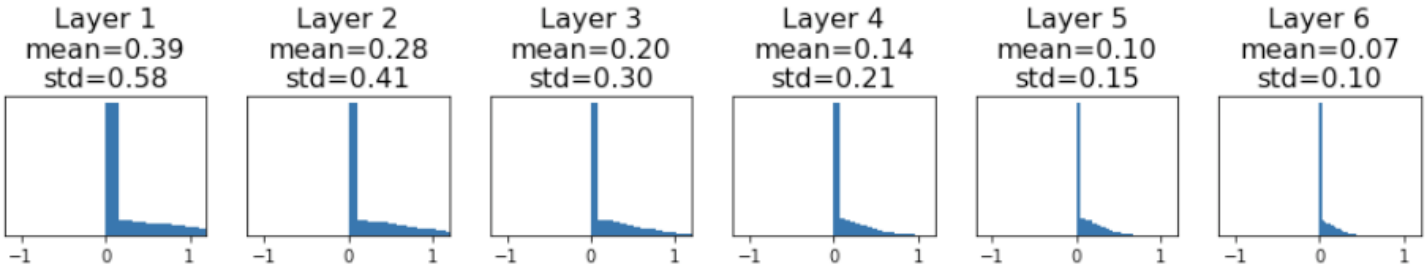
- **What will happen to the activations for the last layer?**

```
dims = [4096]*7
hs = []
x = 0.01*np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout)/np.sqrt(Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

Change from tanh to ReLU

Xavier assumes zero centered activation function

Activations collapse to zero again, no learning

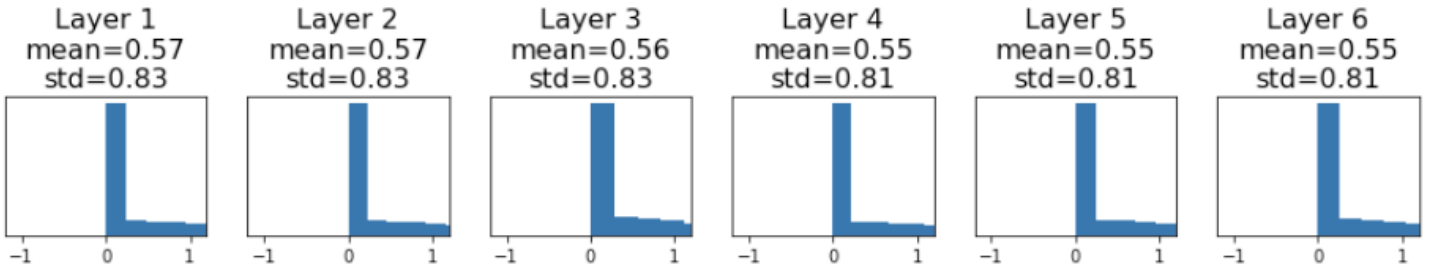


WEIGHT INITIALIZATION: "XAVIER" INITIALIZATION

- **What will happen to the activations for the last layer?**

```
dims = [4096]*7
hs = []
x = 0.01*np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout)/np.sqrt(2/Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

"Just right": Activations are nicely scaled for all layers!



He et al, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", ICCV 2015

PROPER INITIALIZATION IS AN ONGOING AREA OF RESEARCH…

Understanding the difficulty of training deep feedforward neural networks

by Glorot and Bengio, 2010

Exact solutions to the nonlinear dynamics of learning in deep linear neural networks by Saxe et al, 2013

Random walk initialization for training very deep feedforward networksby Sussillo and Abbott, 2014

Delving deep into rectifiers: Surpassing human-level performance on ImageNet classificationby He et al., 2015

Data-dependent Initializations of Convolutional Neural Networksby Krähenbühl et al., 2015

All you need is a good init, Mishkin and Matas, 2015

Fixup Initialization: Residual Learning Without Normalization, Zhang et al, 2019

The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks, Frankle and Carbin, 2019

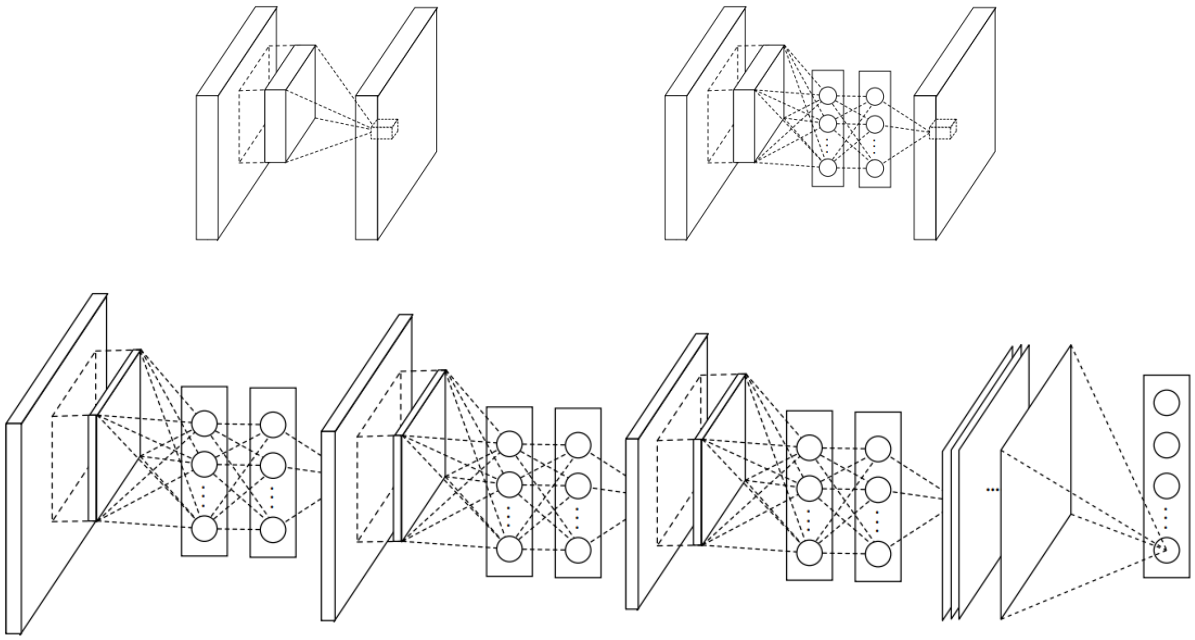
CNN ARCHITECTURES: NETWORK IN NETWORK

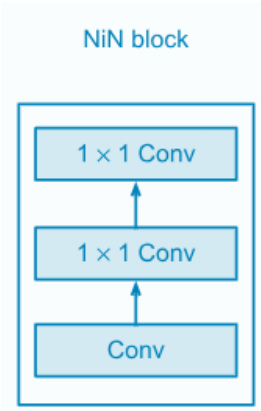
- **Case study: Network In Network**
  - The convolution filter in CNN is a generalized linear model (GLM)
  - The level of abstraction is low with GLM
  - In NIN, the GLM is replaced with a ”micro network” structure
  - ”micro network” structure is a general nonlinear function approximator
  - nonlinear function approximator can enhance the abstraction ability of the local model.
  - The micro neural network with a multilayer perceptron (mlpconv layer), which is a potent function approximator.
  - This micro neural network is trainable by back-propagation
  - Rectified linear unit is used as the activation function in the multilayer perceptron
  - The feature maps are obtained by sliding the micro networks over the input in a similar manner as CNN
  - directly output the spatial average of the feature maps from the last mlpconv layer
  - Applying global average pooling layer on the last mlpconv layer and then the resulting vector is fed into the softmax layer

Lin, M., 2013. Network in network

CNN ARCHITECTURES:
NETWORK IN NETWORK

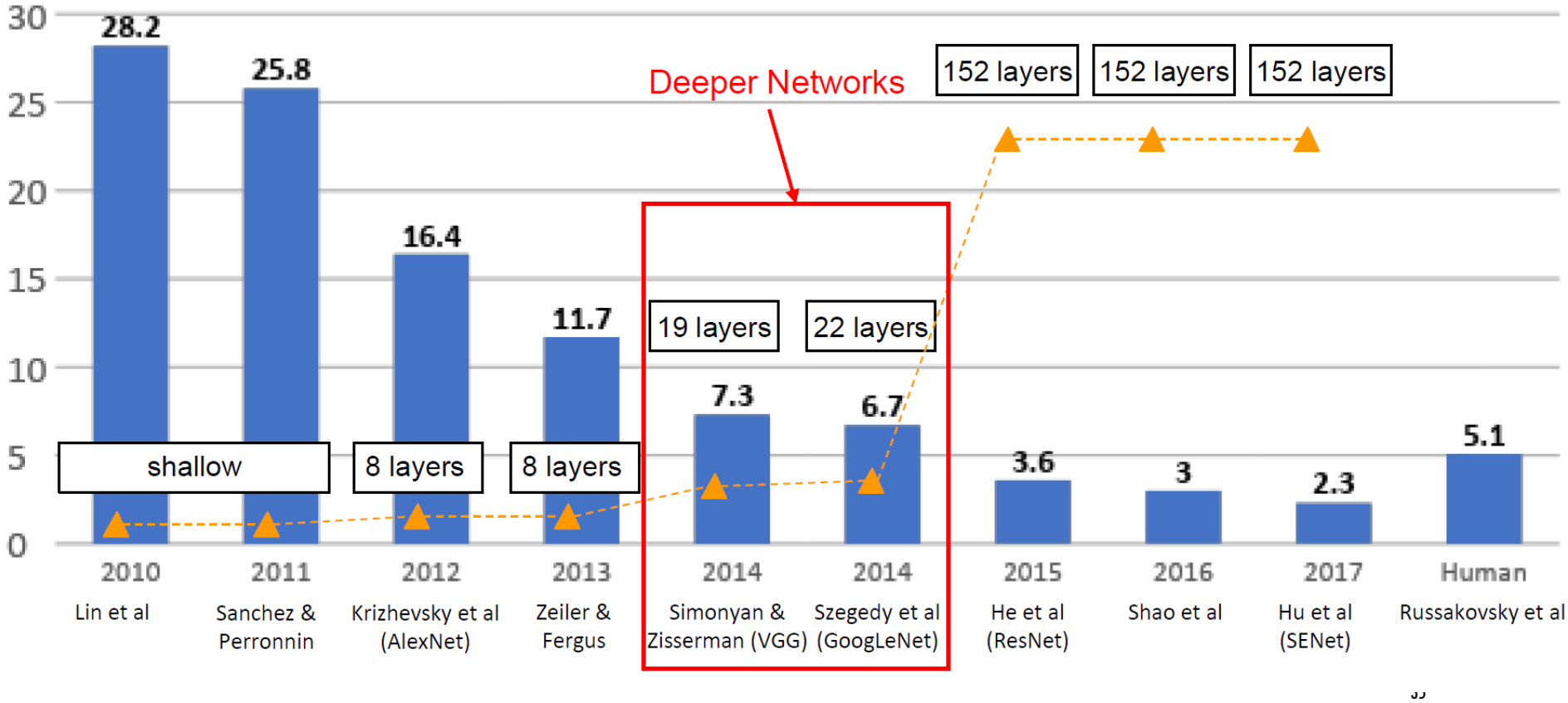
- **Case study: Network In Network**





The overall structure of Network In Network. In this paper the NINs include the stacking of three mlpconv layers and one global average pooling layer.

CNN ARCHITECTURES

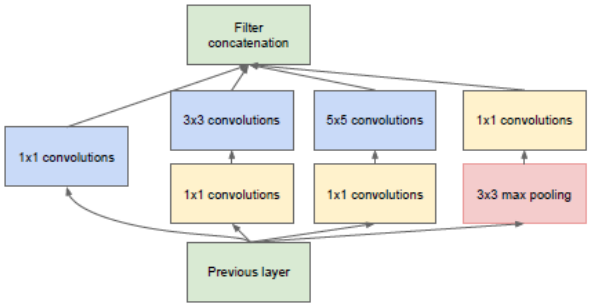


CNN ARCHITECTURES: GOOGLENET

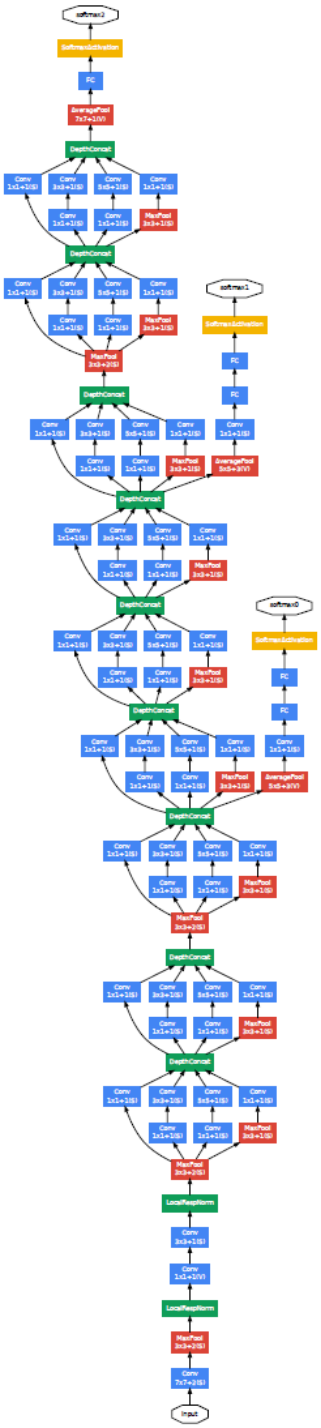
- **Case Study: GoogLeNet**

[Szegedy et al., 2014]

- Deeper networks, with computational efficiency
- ILSVRC'14 classification winner (6.7% top 5 error)
- 22 layers
- Only 5 million parameters!
- Avoids expensive FC layers
- 12x less than AlexNet
- 27x less than VGG-16
- Efficient "Inception" module



Inception module with dimension reductions



CNN ARCHITECTURES: GOOGLENET

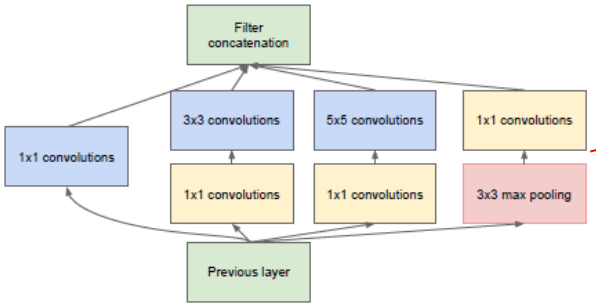
- **Motivation and High Level Considerations**
  - The most straightforward way of improving the performance of deep neural networks $\Longrightarrow$ increasing their size (This includes both increasing the depth – the number of levels – of the network and its width: the number of units at each level.)
  - However this simple solution comes with two major drawbacks:
    - Bigger size typically means a larger number of parameters, which makes the enlarged network more prone to overfitting
    - Dramatically increased use of computational resources
  - The fundamental way of solving both issues would be by ultimately moving from fully connected to sparsely connected architectures, even inside the convolutions.
  - The main idea of the Inception architecture is based on finding out how an optimal local sparse structure in a convolutional vision network can be approximated and covered by readily available dense components.
  - All we need is to find the optimal local construction and to repeat it spatially.
  - Well known **Hebbian principle** – **neurons that fire together, wire together-**.

CNN ARCHITECTURES: GOOGLENET

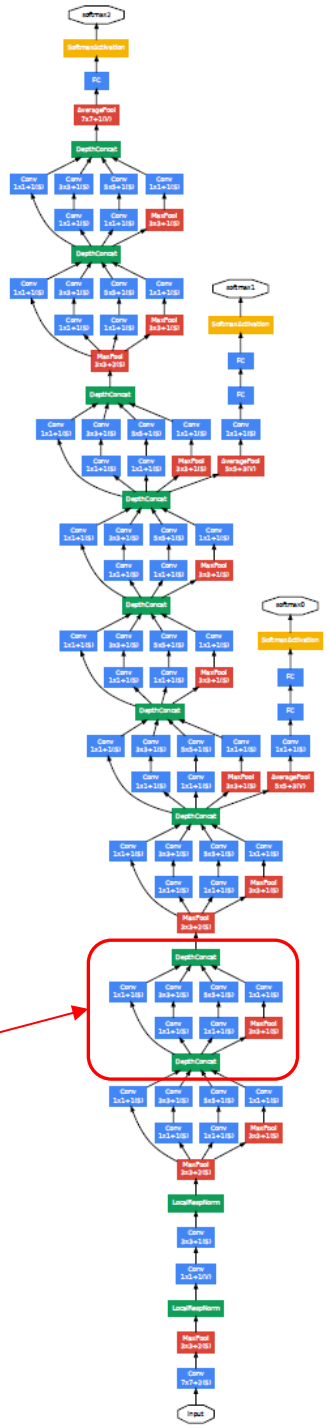
- **Case Study: GoogLeNet**

[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



Inception module with dimension reductions



CNN ARCHITECTURES: GOOGLENET

➢ **Case Study: GoogLeNet**

[Szegedy et al., 2014]

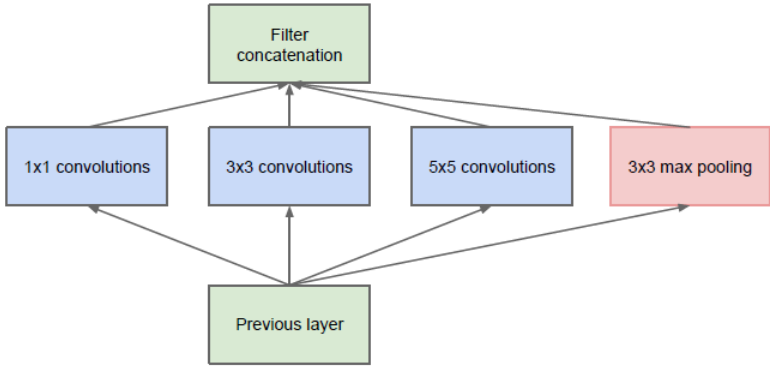
Apply parallel filter operations on the input from previous layer:

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

Concatenate all filter outputs together channel-wise

Q: What is the problem with this?

Computational complexity



Naive Inception module

CNN ARCHITECTURES: GOOGLENET

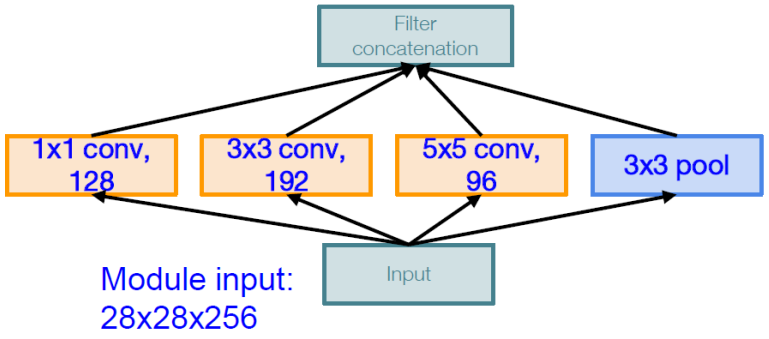
- **Case Study: GoogLeNet**

[Szegedy et al., 2014]

Q: What is the problem with this?

- Computational complexity

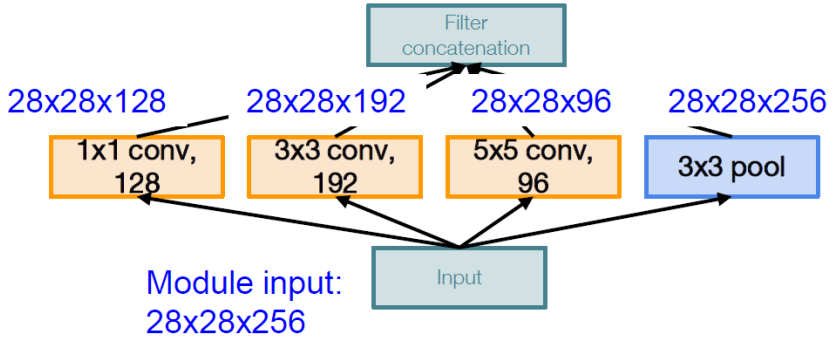
Example :



Naive Inception module

Q1: What are the output sizes of all different filter operations?

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



Naive Inception module

CNN ARCHITECTURES: GOOGLENET

- **Case Study: GoogLeNet**

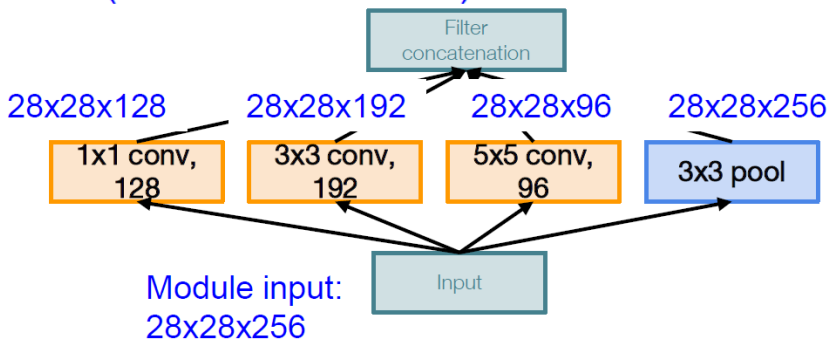
[Szegedy et al., 2014]

Q: What is the problem with this?

- Computational complexity

Example :

28x28x(128+192+96+256) = 28x28x672 = **529k**



Naive Inception module

Conv Ops:

[1x1 conv, 128] 28x28x128x1x1x256

[3x3 conv, 192] 28x28x192x3x3x256

[5x5 conv, 96] 28x28x96x5x5x256

Total: 854M ops

Very expensive compute

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

CNN ARCHITECTURES: GOOGLENET

- **Case Study: GoogLeNet**

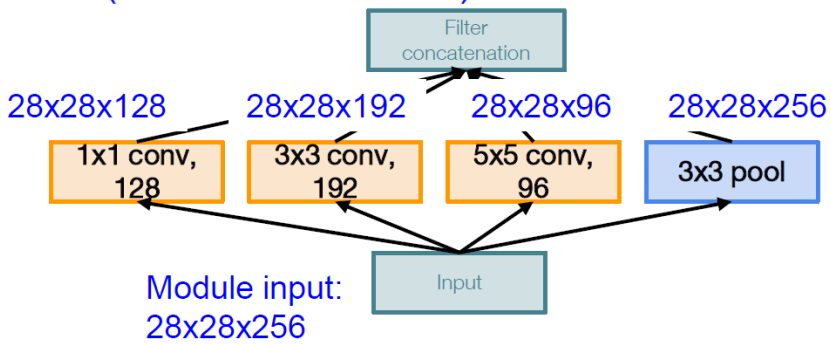
[Szegedy et al., 2014]

Q: What is the problem with this?

- Computational complexity

Example :

28x28x(128+192+96+256) = 28x28x672 = **529k**



Naive Inception module

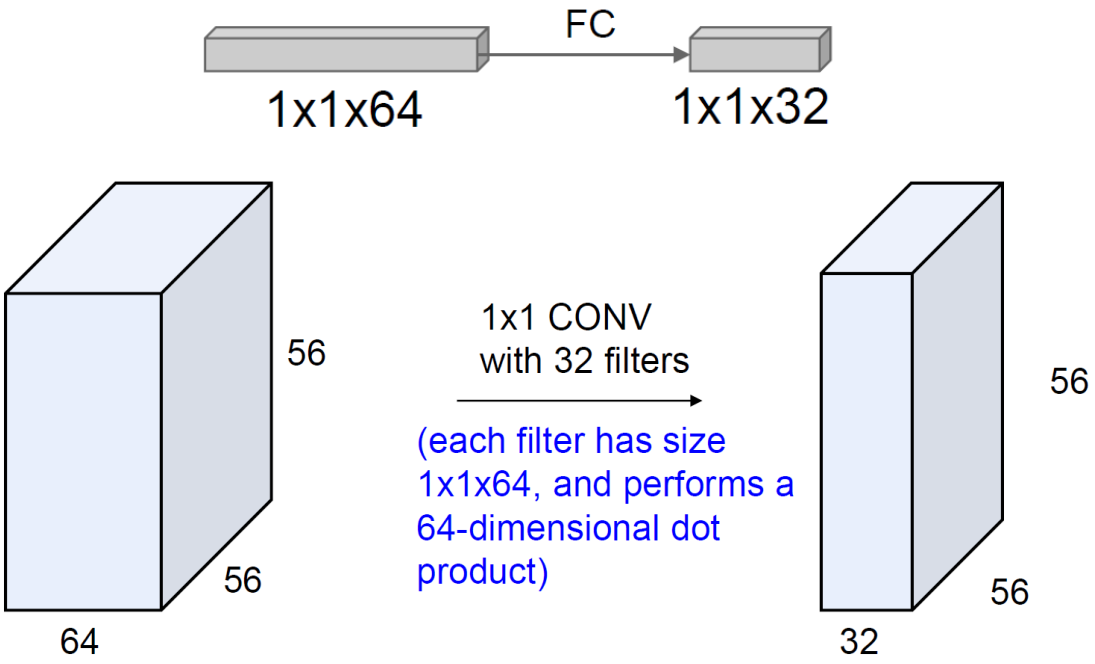
Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops
Very expensive compute

Solution: "bottleneck" layers that use 1x1 convolutions to reduce feature channel size

CNN ARCHITECTURES: GOOGLENET

- **Review: 1x1 convolutions**

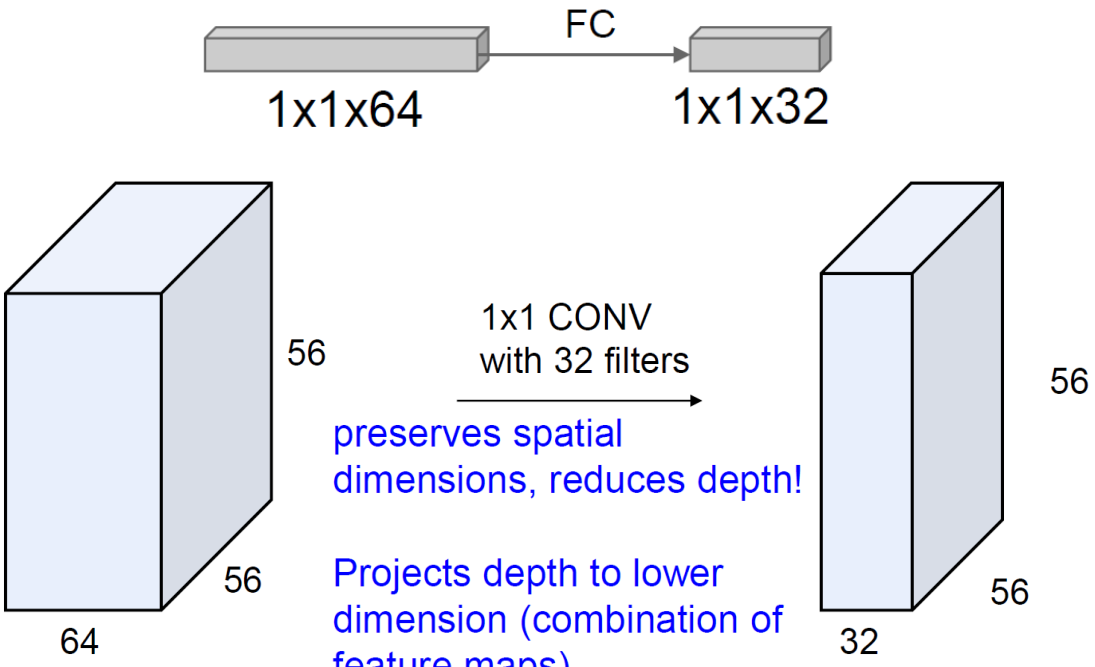
Alternatively, interpret it as applying the same FC layer on each input pixel



CNN ARCHITECTURES: GOOGLENET

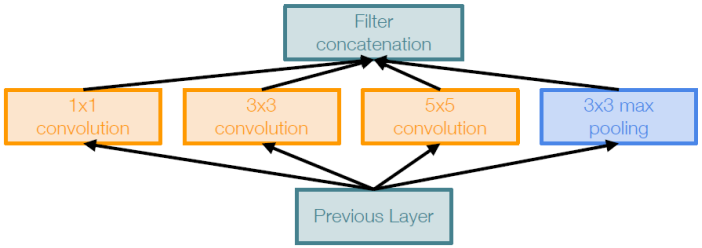
- **Review: 1x1 convolutions**

Alternatively, interpret it as applying the same FC layer on each input pixel



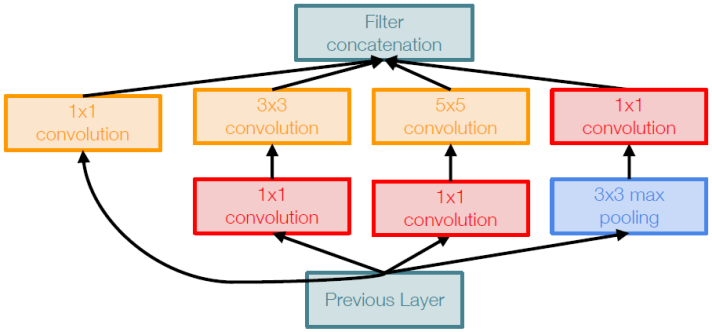
CNN ARCHITECTURES: GOOGLENET

Case Study: GoogLeNet



Naive Inception module

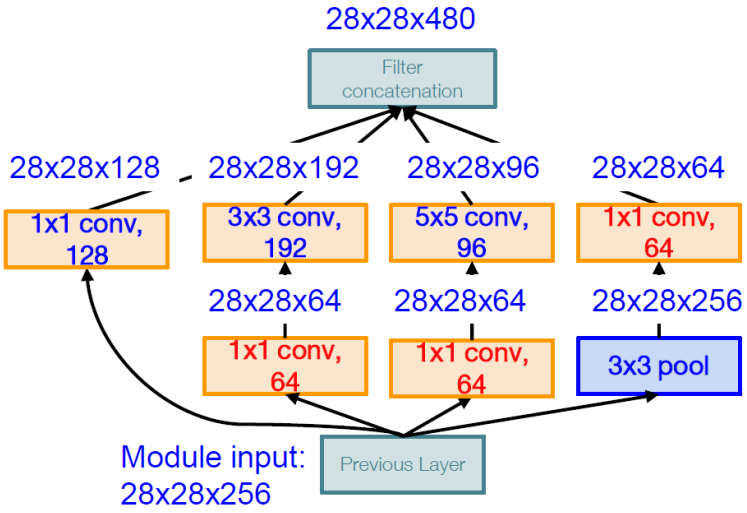
1x1 conv “bottleneck” layers



Inception module with dimension reduction

CNN ARCHITECTURES: GOOGLENET

➢ **Case Study: GoogLeNet**



Inception module with dimension reduction

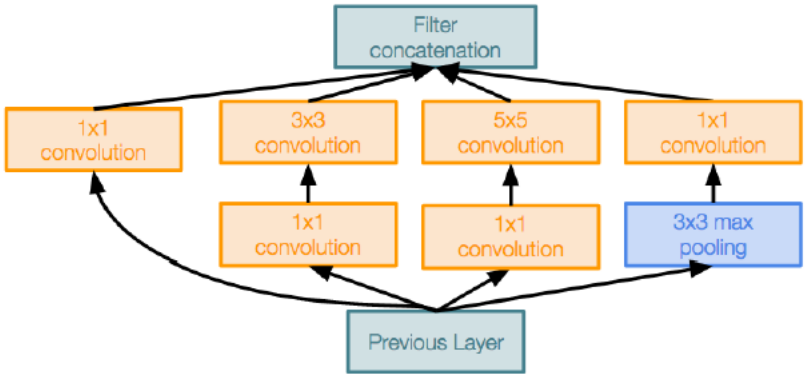
Using same parallel layers as naive example, and adding "1x1 conv, 64 filter" bottlenecks:

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

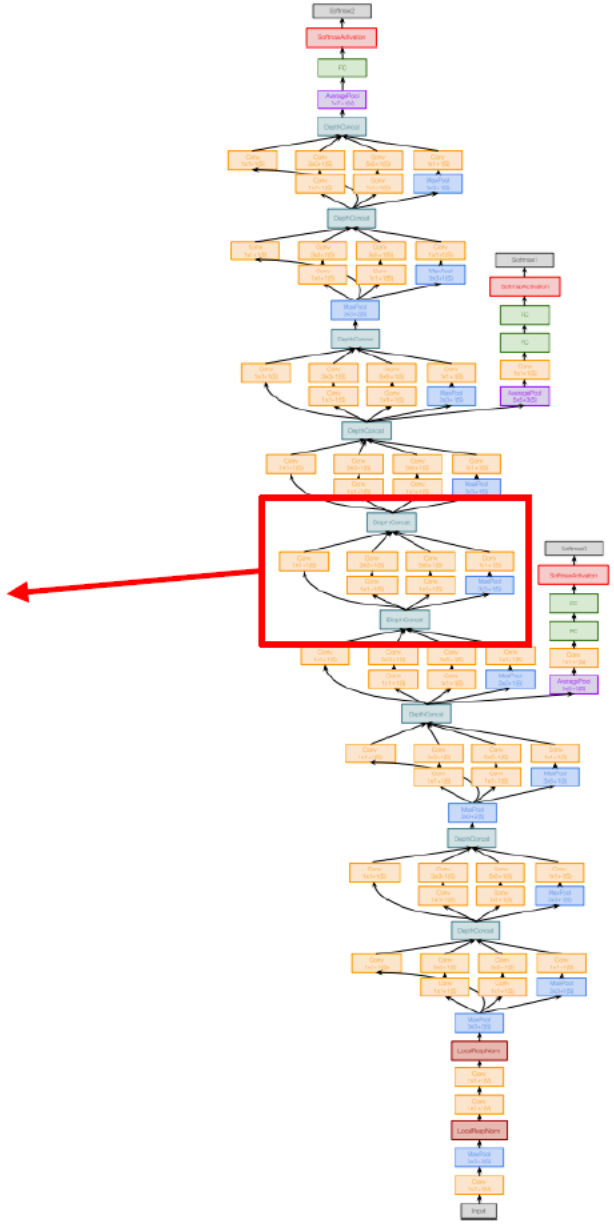
Compared to 854M ops for naïve version
Bottleneck can also reduce depth after pooling layer

CNN ARCHITECTURES: GOOGLENET

- **Case Study: GoogLeNet**



Inception module

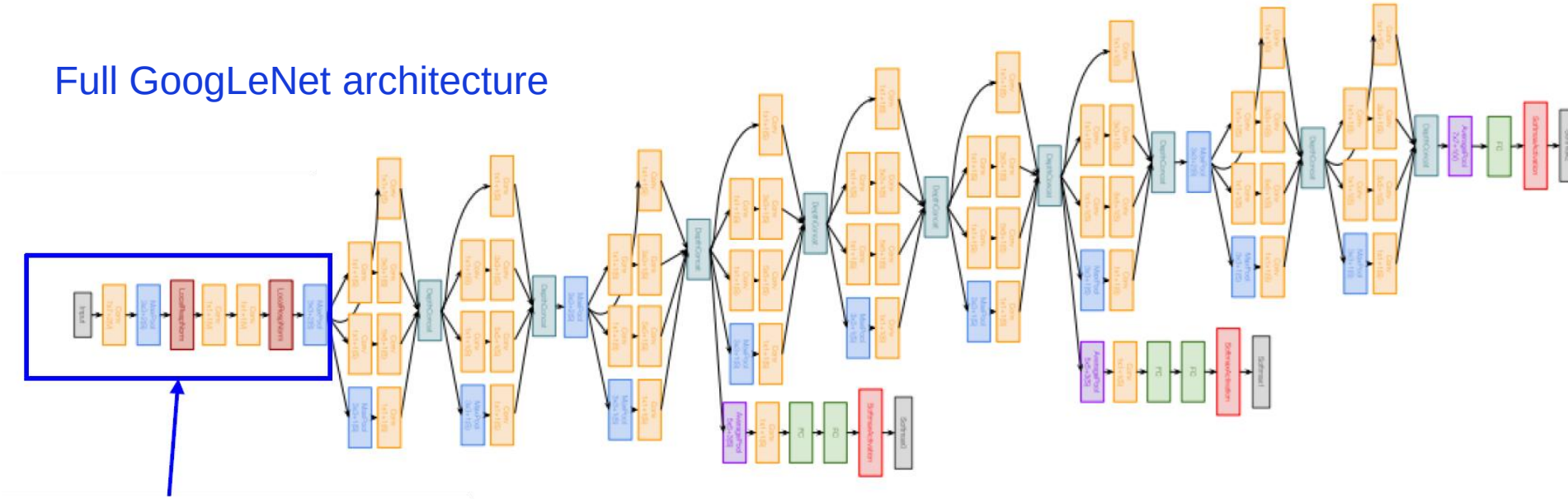


Stack Inception modules with dimension reduction on top of each other

CNN ARCHITECTURES: GOOGLENET

- **Case Study: GoogLeNet**

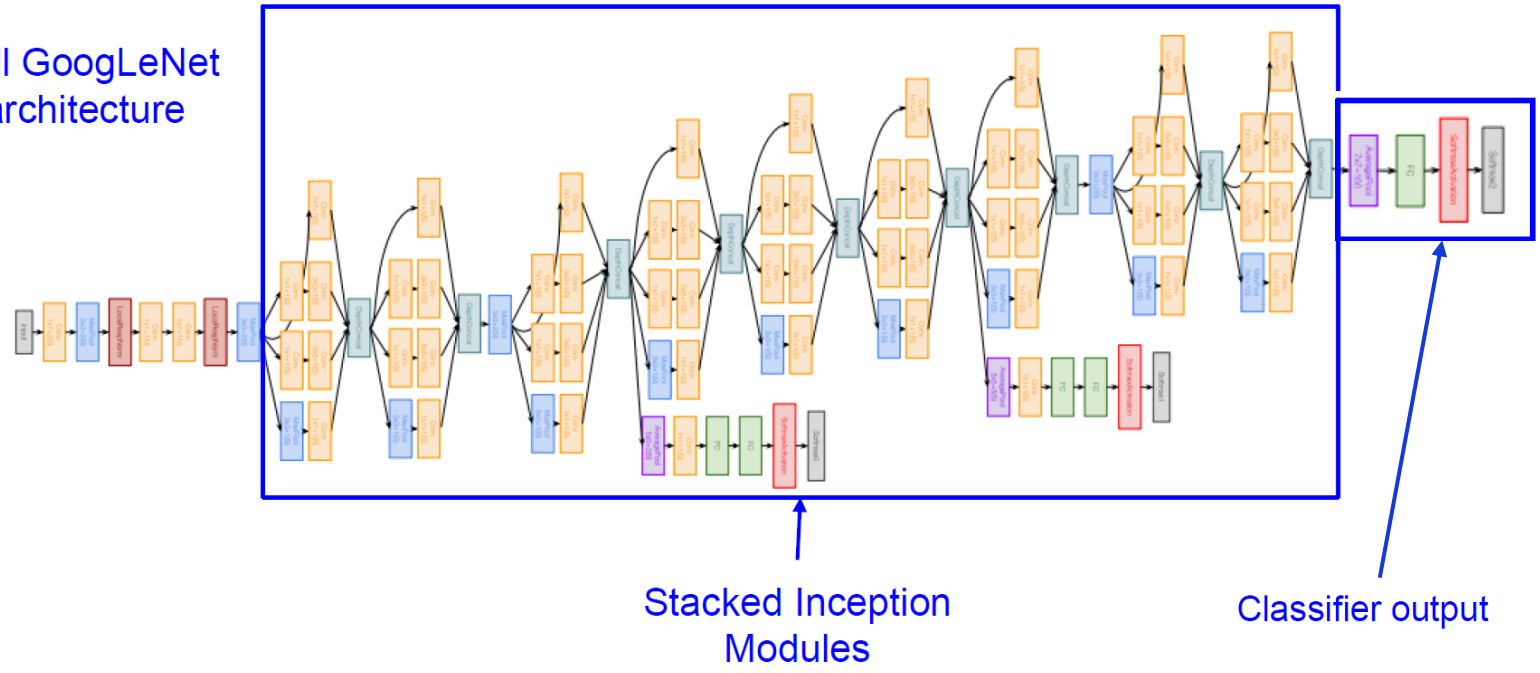
Full GoogLeNet architecture



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

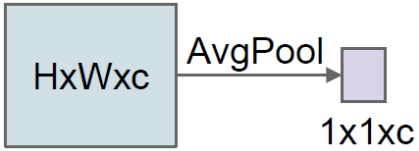
CNN ARCHITECTURES: GOOGLENET

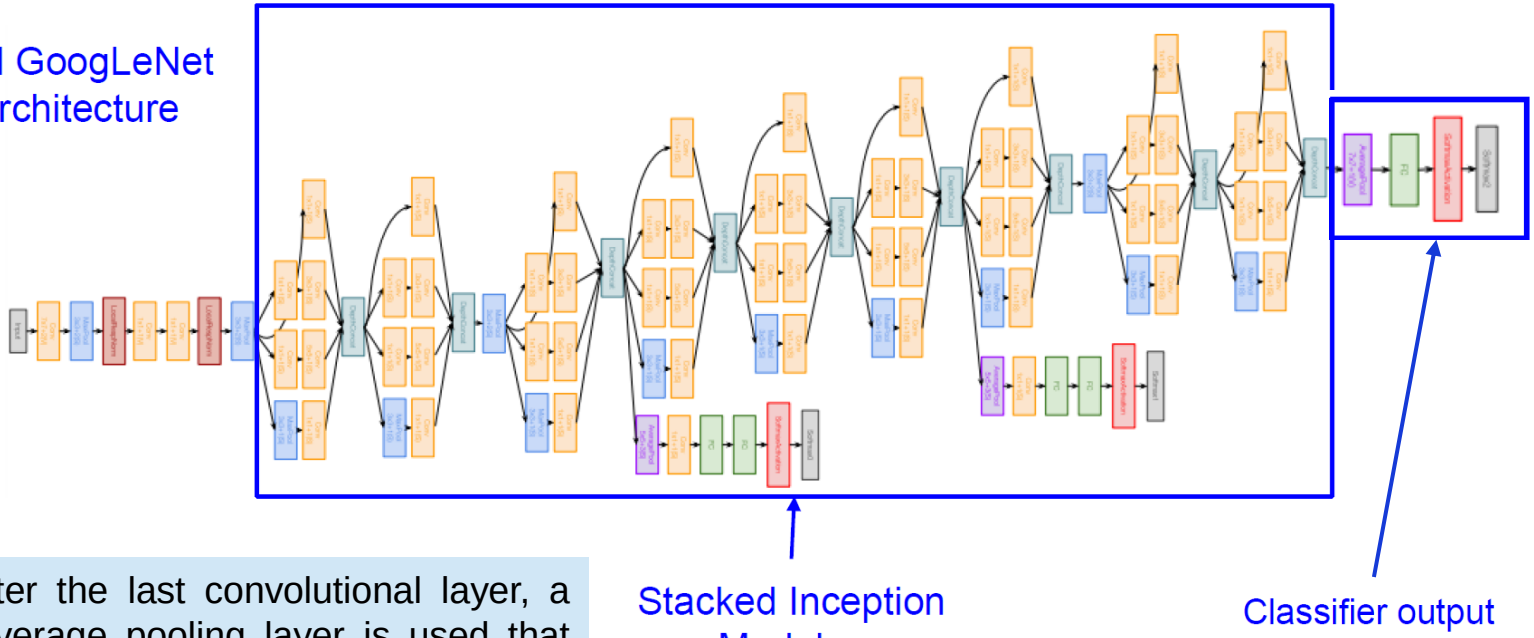
- **Case Study: GoogLeNet**



CNN ARCHITECTURES: GOOGLENET

- **Case Study: GoogLeNet**



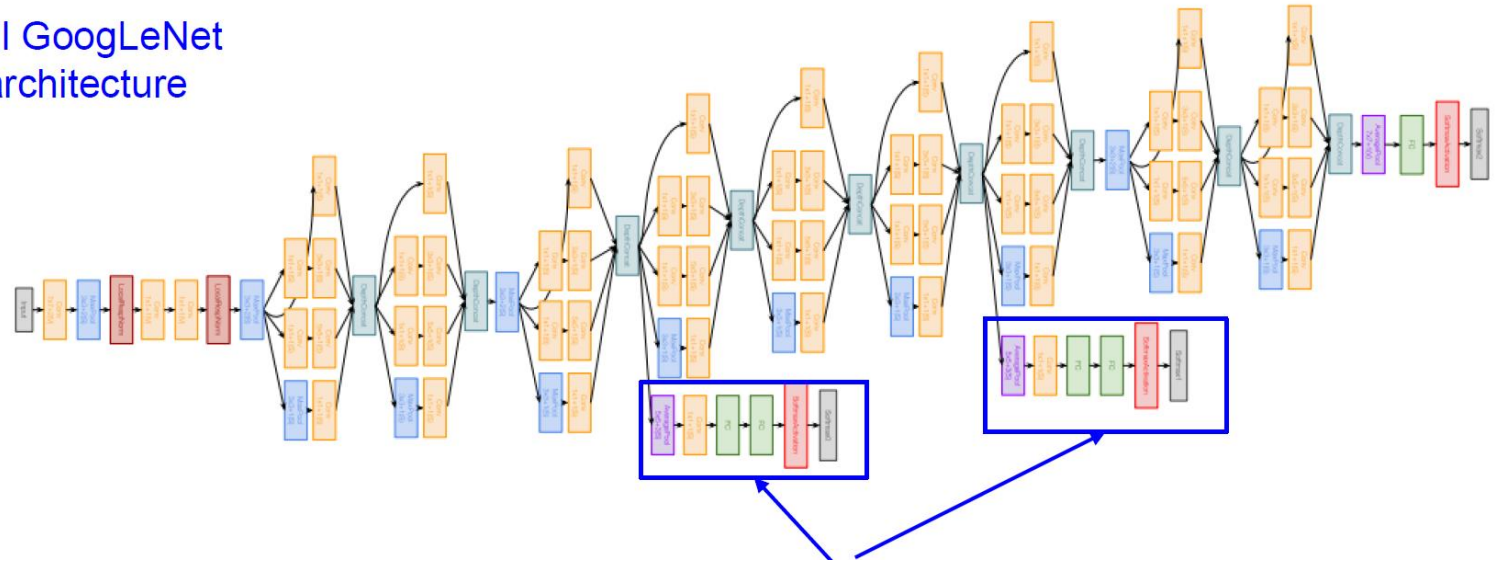


Note: after the last convolutional layer, a global average pooling layer is used that spatially averages across each feature map, before final FC layer. No longer multiple expensive FC layers!

CNN ARCHITECTURES: GOOGLENET

- **Case Study: GoogLeNet**

Full GoogLeNet architecture

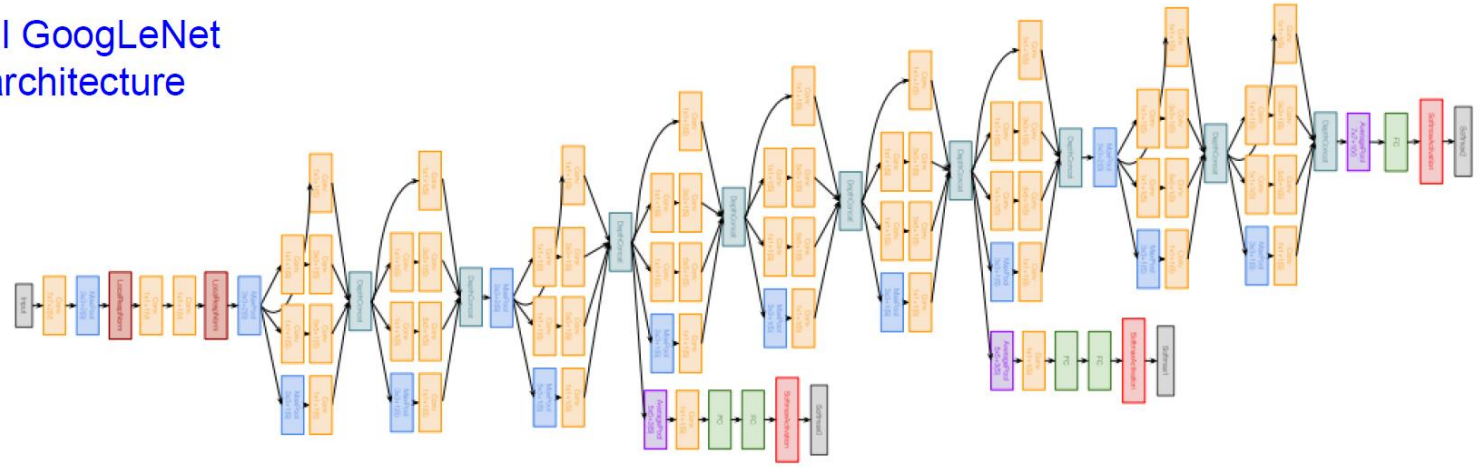


Auxiliary classification outputs to inject additional gradient at lower layers (AvgPool-1x1Conv-FC-FC-Softmax)

CNN ARCHITECTURES: GOOGLENET

- **Case Study: GoogLeNet**

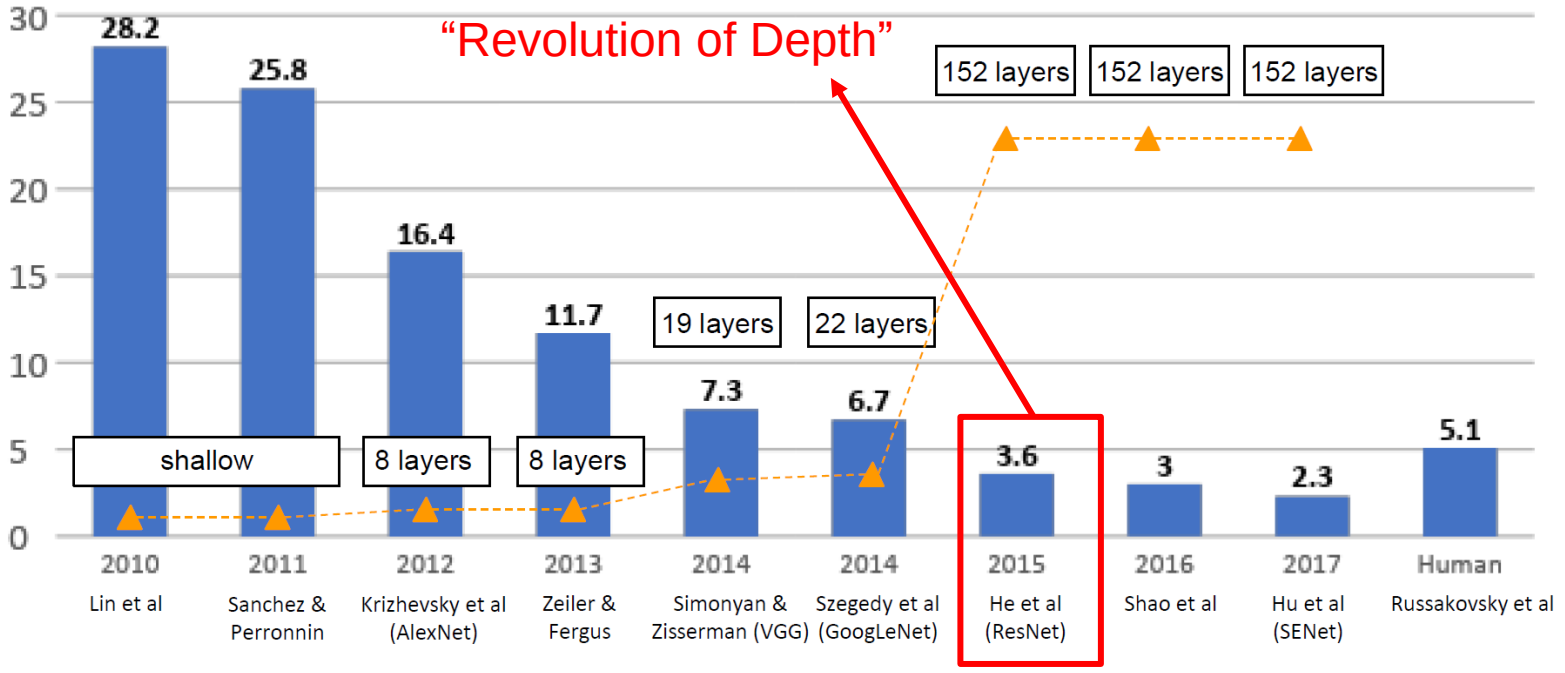
Full GoogLeNet architecture



22 total layers with weights
(parallel layers count as 1 layer => 2 layers per Inception module. Don't count auxiliary output layers)

CNN ARCHITECTURES:

- ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

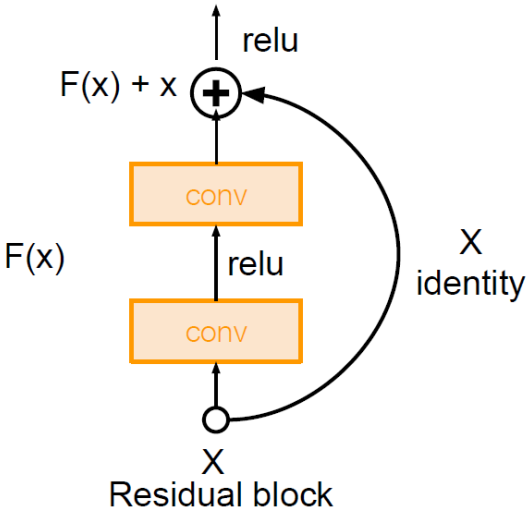


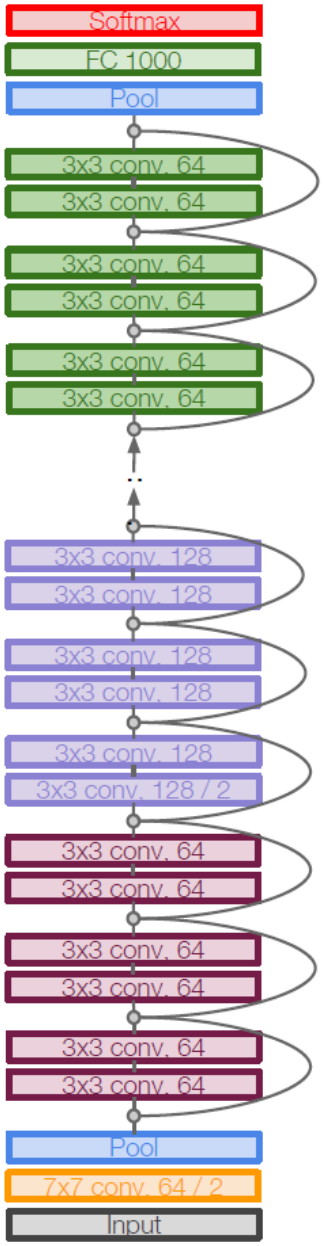
CNN ARCHITECTURES: RESNET

- **Case Study: ResNet**

[He et al., 2015]

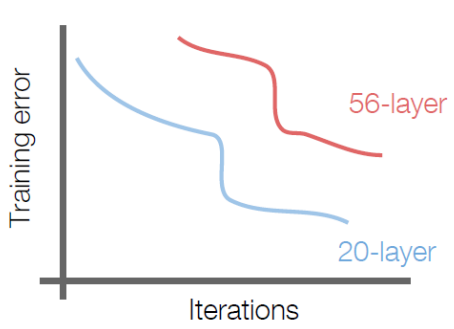
Very deep networks using residual connections
- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)

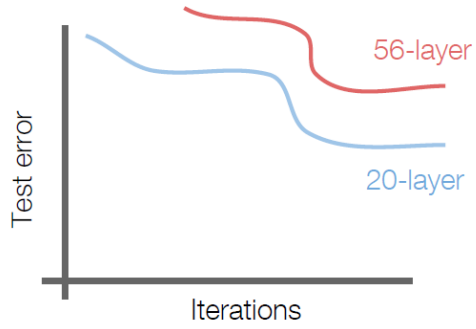




CNN ARCHITECTURES: RESNET

- **Case Study: ResNet**
  - Deeper networks facing with vanishing gradient
  - This problem, however, has been addressed by normalized initialization and intermediate normalization layers.
  - 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!

CNN ARCHITECTURES:
RESNET

- **Is learning better networks as easy as stacking more layers?**
  - **Problem**: When deeper networks are able to start converging, a **degradation** problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error.
  - **Finding solution**: Let us consider a shallower architecture and its deeper counterpart that adds more layers onto it. There exists a solution by construction to the deeper model:
    - The added layers are identity mapping, and the other layers are copied from the learned shallower model.
    - The existence of this constructed solution indicates that a deeper model should produce no higher training error than its shallower counterpart. But experiments show that our current solvers on hand are unable to find solutions that are comparably good or better than the constructed solution

CNN ARCHITECTURES: RESNET

- **Case Study: ResNet**

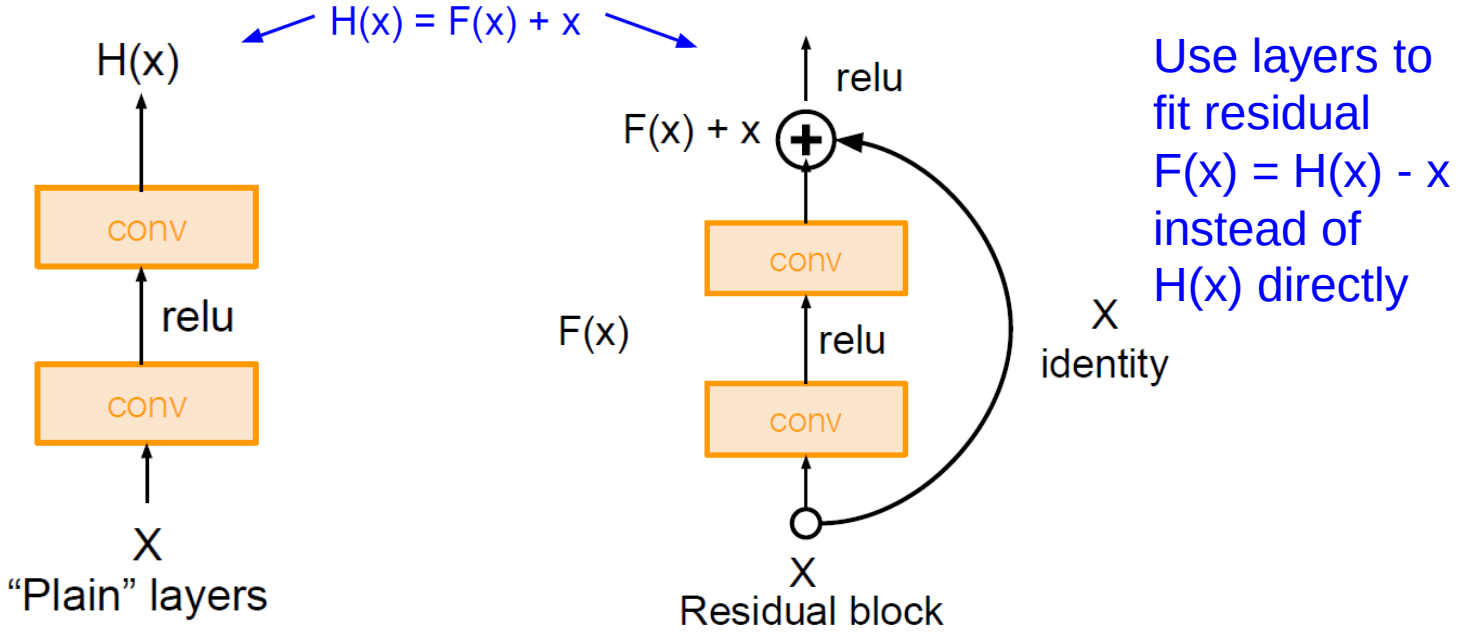
Hypothesis: the problem is an ***optimization*** problem, deeper models are harder to optimize

- 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.

CNN ARCHITECTURES: RESNET

- **Case Study: ResNet**

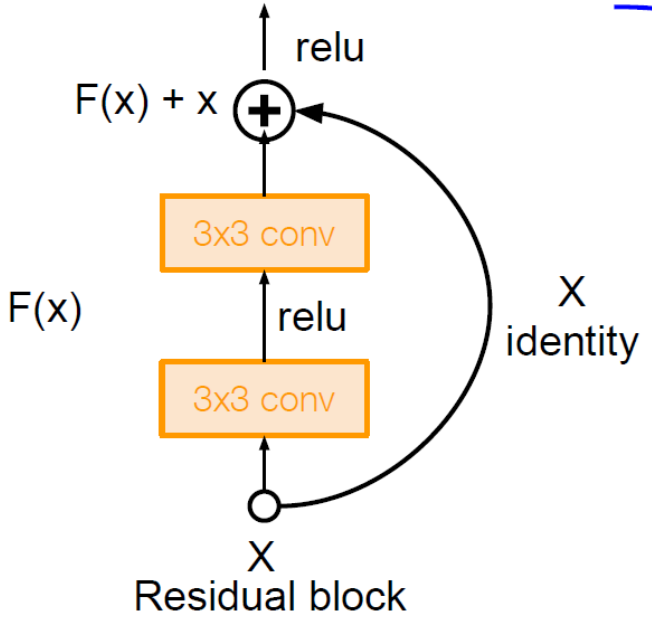
Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

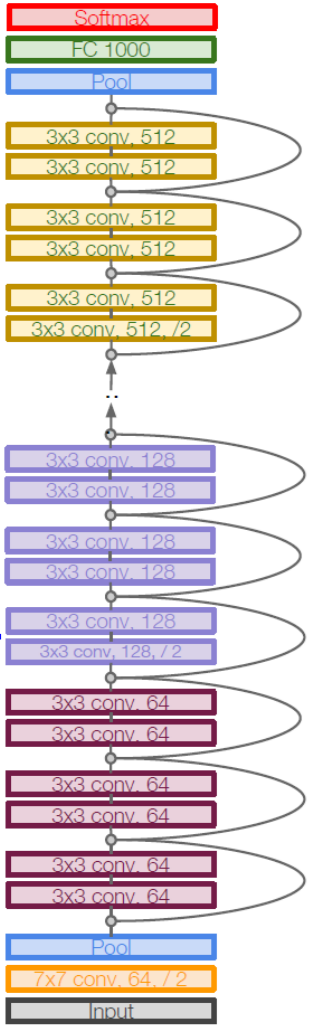


CNN ARCHITECTURES: RESNET

- **Case Study: ResNet**

Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers



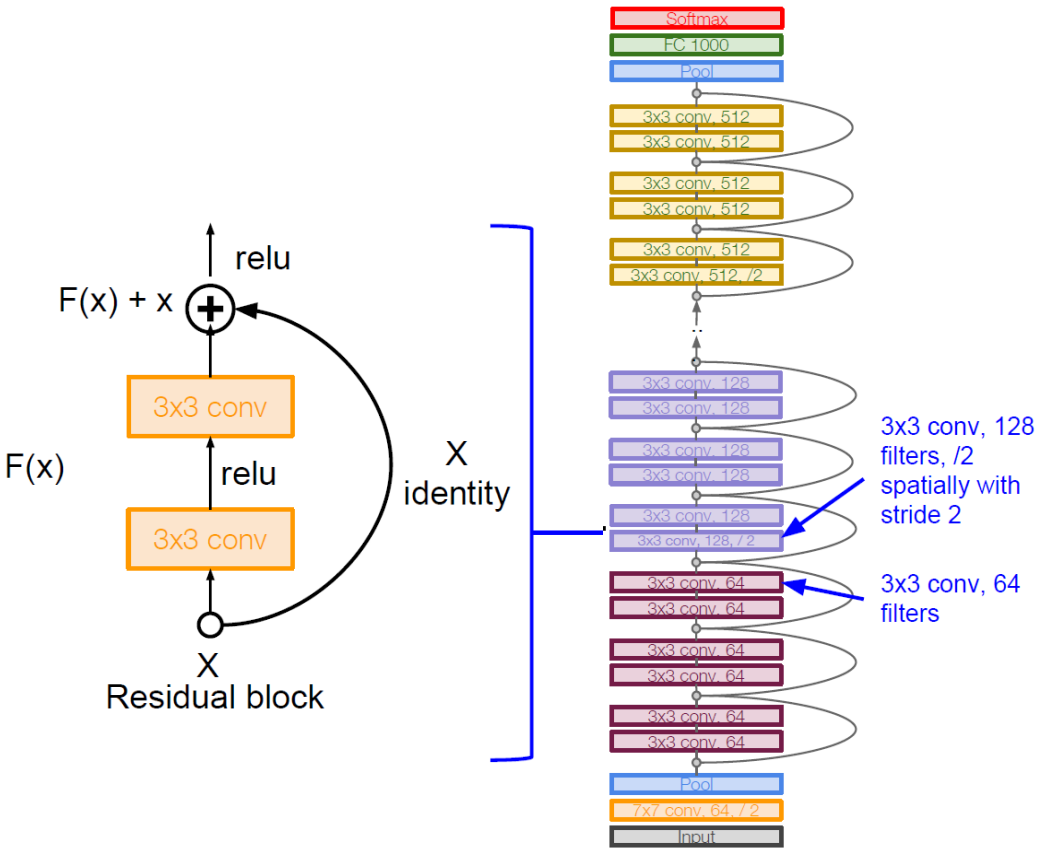


CNN ARCHITECTURES: RESNET

- **Case Study: ResNet**

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)

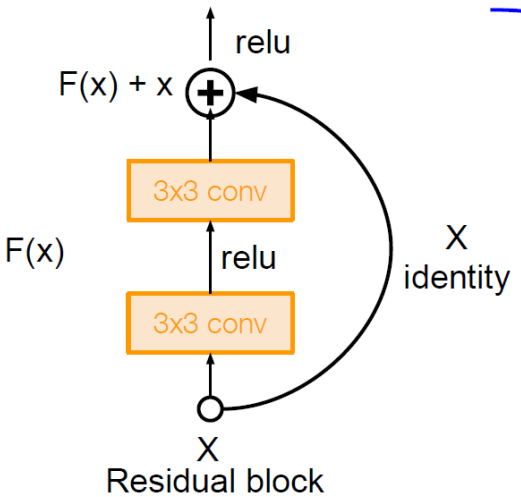


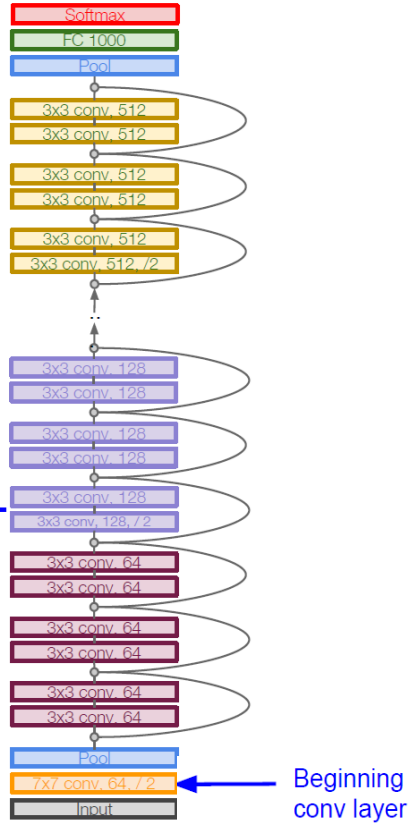
CNN ARCHITECTURES: RESNET

- **Case Study: ResNet**

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



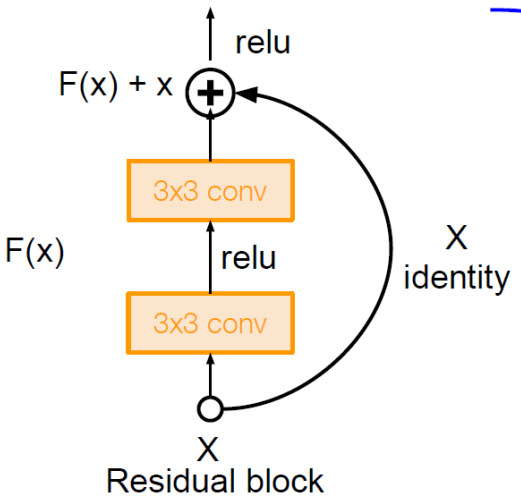


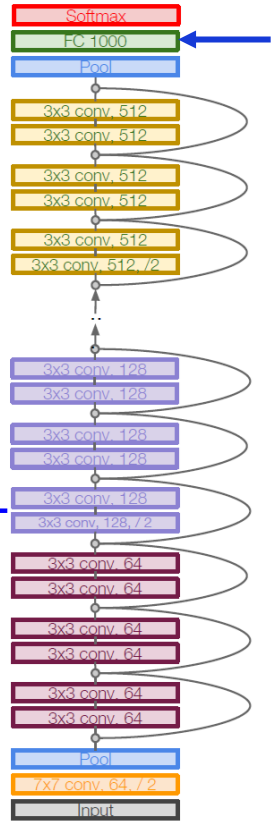
CNN ARCHITECTURES: RESNET

- **Case Study: ResNet**

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)





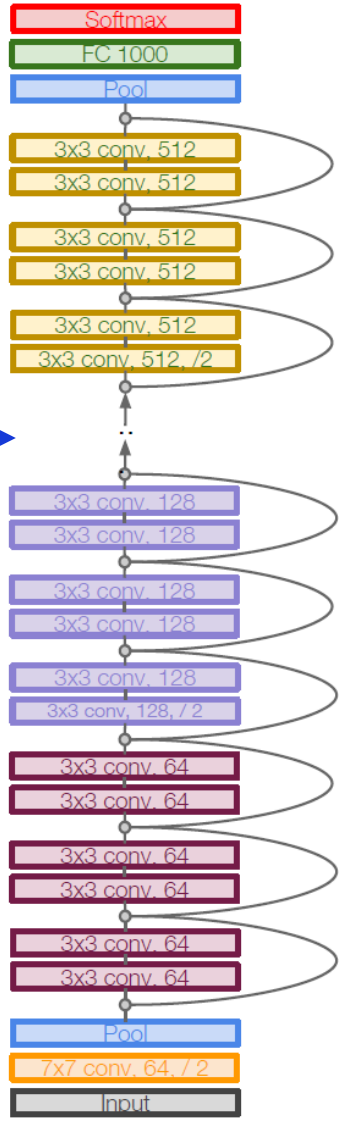
No FC layers besides FC 1000 to output classes

Global average pooling layer after last conv layer

CNN ARCHITECTURES: RESNET

- **Case Study: ResNet**

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



CNN ARCHITECTURES:
RESNET

- **Case Study: ResNet**

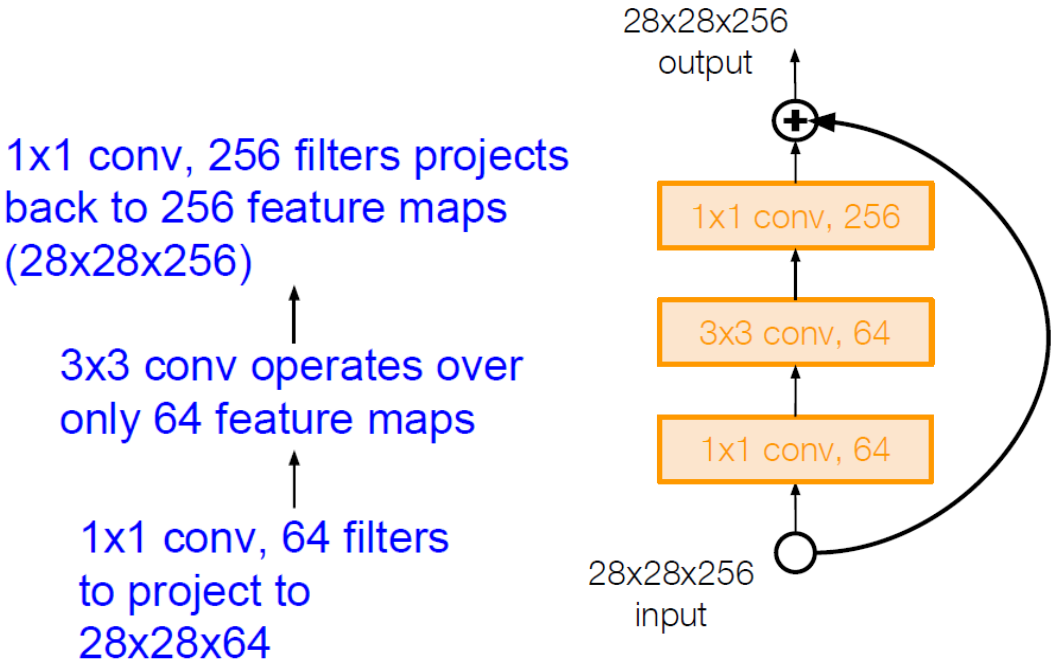
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

CNN ARCHITECTURES: RESNET

- **Case Study: ResNet**

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

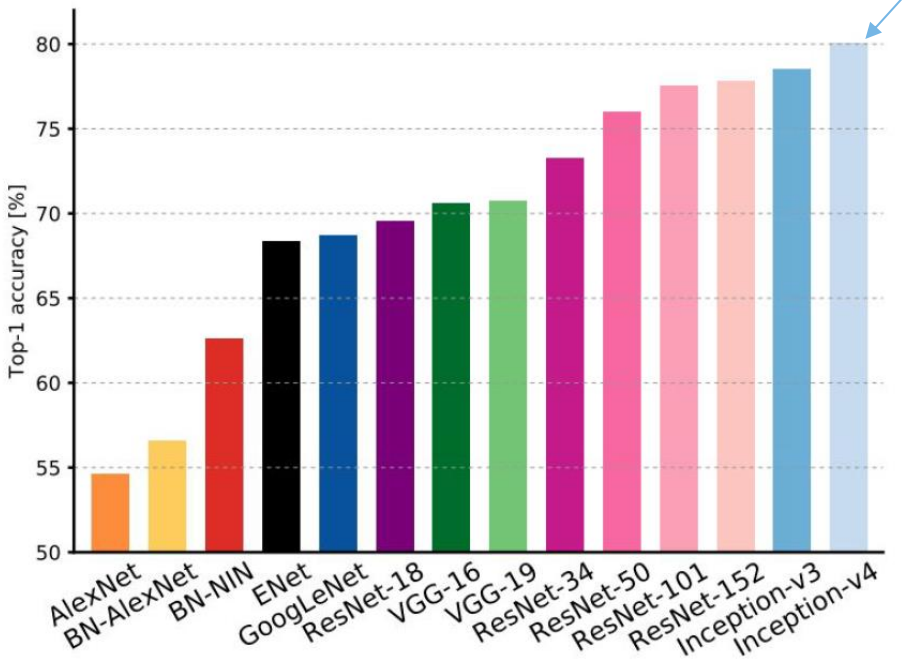


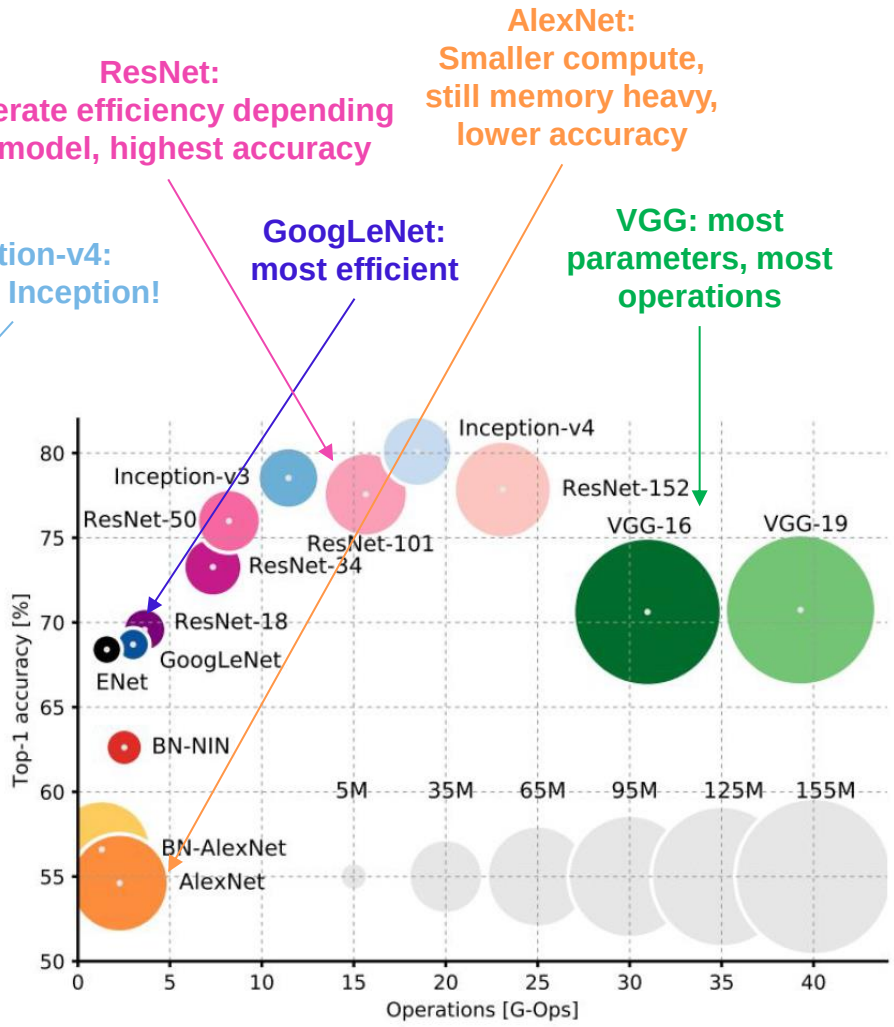
CNN ARCHITECTURES: RESNET

- **Case Study: ResNet**
  - 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

CNN ARCHITECTURES

- **Comparing complexity...**





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

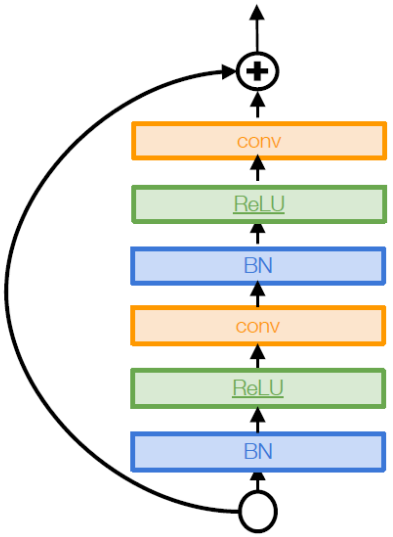
CNN ARCHITECTURES

➢ **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

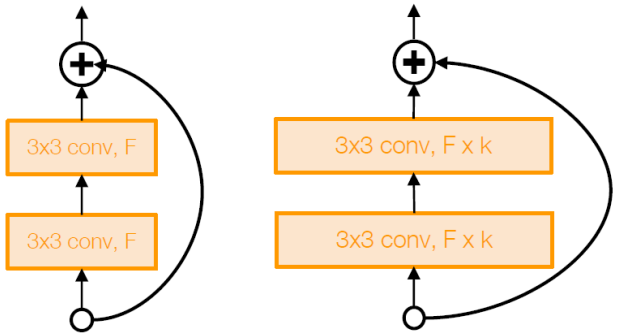


CNN ARCHITECTURES: WIDE RESNET

➢ **Improving ResNets...**
Wide Residual Networks

[Zagoruyko et al. 2016]

- Argues that residuals are the important factor, not depth
- User wider residual blocks (F x 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)
- having 50 times less layers and being more than 2 times faster.
- For instance, wide 16-layer deep network has the same accuracy as a 1000-layer thin deep network and a comparable number of parameters, although being several times faster to train.



Basic residual block Wide residual block

Model	top-1 err, %	top-5 err, %	#params
ResNet-50	24.01	7.02	25.6M
ResNet-101	22.44	6.21	44.5M
ResNet-152	22.16	6.16	60.2M
WRN-50-2-bottleneck	21.9	6.03	68.9M

ImageNet 1k

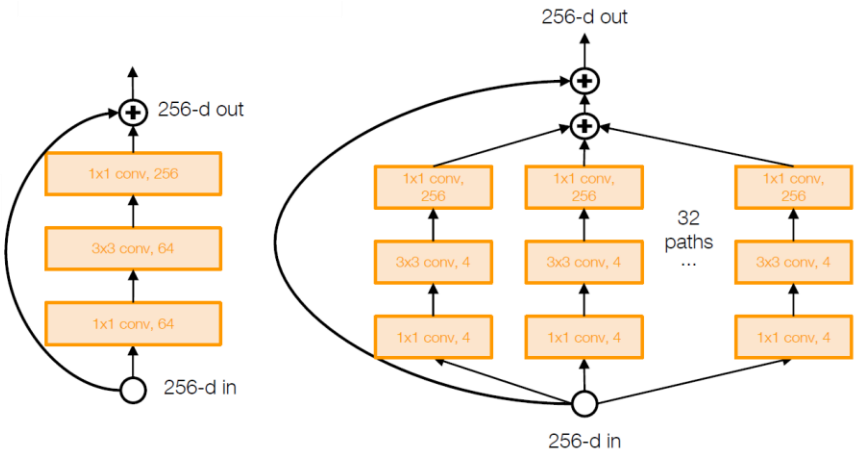
CNN ARCHITECTURES: RESNEXT

- **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



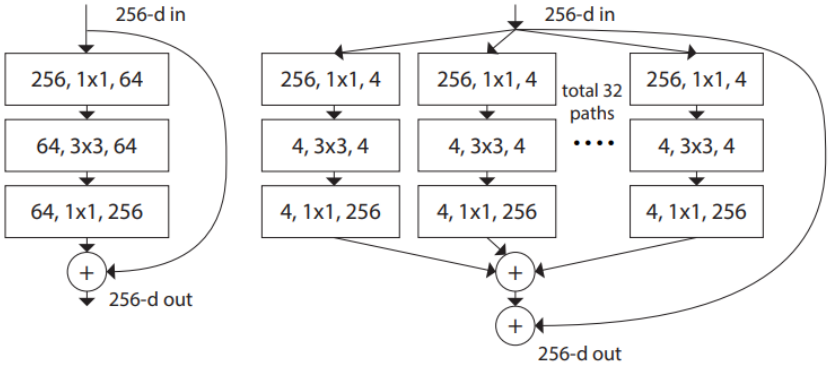
	setting	top-1 err (%)	top-5 err (%)
1× complexity references:			
ResNet-101	1 × 64d	22.0	6.0
ResNeXt-101	32 × 4d	21.2	5.6

ImageNet 1k

CNN ARCHITECTURES:
RESNEXT

- **Improving ResNets...**

  Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)



$$256 \cdot 64 + 3 \cdot 3 \cdot 64 \cdot 64 + 64 \cdot 256 \approx 70\text{k}$$

$$C \cdot (256 \cdot d + 3 \cdot 3 \cdot d \cdot d + d \cdot 256)$$

When C = 32 and d = 4, $\Longrightarrow \approx 70$k.

stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C{=}32 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C{=}32 \\ 1\times1, 512 \end{bmatrix} \times 4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C{=}32 \\ 1\times1, 1024 \end{bmatrix} \times 6$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C{=}32 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		**25.5**×10^6	**25.0**×10^6
FLOPs		**4.1**×10^9	**4.2**×10^9

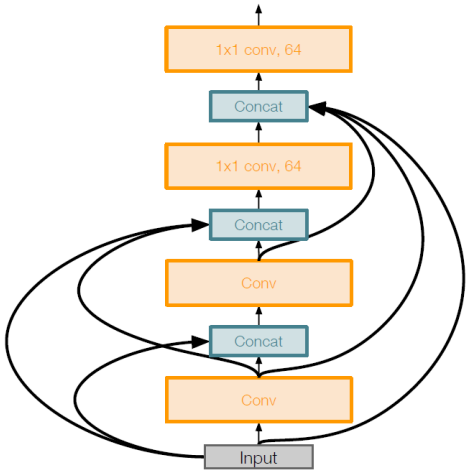
CNN ARCHITECTURES: DENSENET

- **Beyond ResNets...**

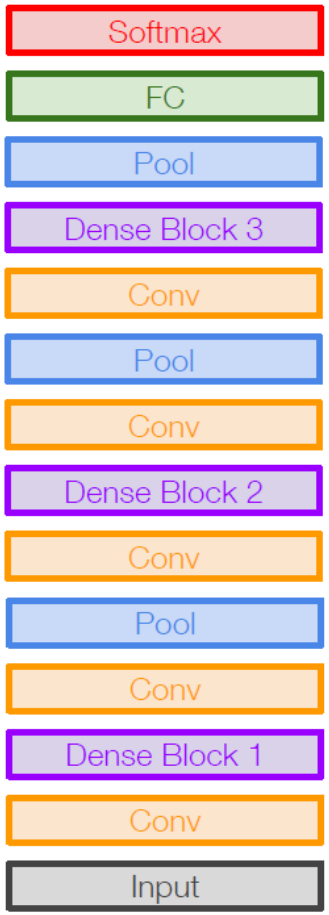
Densely Connected Convolutional Networks (DenseNet)

[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
- Showed that shallow 50-layer network can outperform deeper 152 layer ResNet



Dense Block

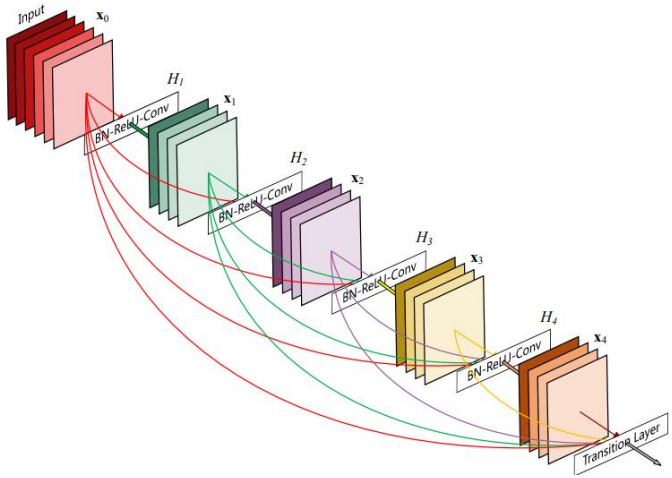


CNN ARCHITECTURES: DENSENET

➢Beyond ResNets...

Densely Connected Convolutional Networks

- Traditional convolutional networks with L layers have L connections one between each layer and its subsequent layer
- DensNet has L(L+1)/2 direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers.

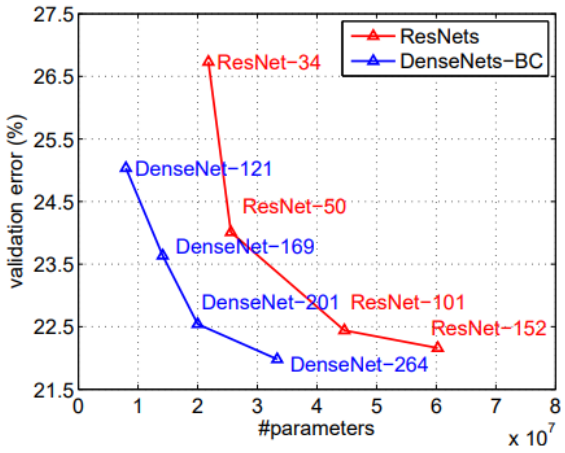


Self Study

CNN ARCHITECTURES:
DENSENET

- **Beyond ResNets...**

Densely Connected Convolutional Networks



Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

DenseNet architectures for ImageNet.

CNN ARCHITECTURES: SQUEEZENET

- **Efficient networks...**

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

  - With equivalent accuracy, smaller CNN architectures offer at least three advantages:
    - less communication
    - less bandwidth
    - more feasible to deploy on hardware with limited memory
  - ARCHITECTURAL DESIGN STRATEGIES
    - Replace 3x3 filters with 1x1 filters
    - Decrease the number of input channels to 3x3 filters
    - Downsample late in the network so that convolution layers have large activation maps

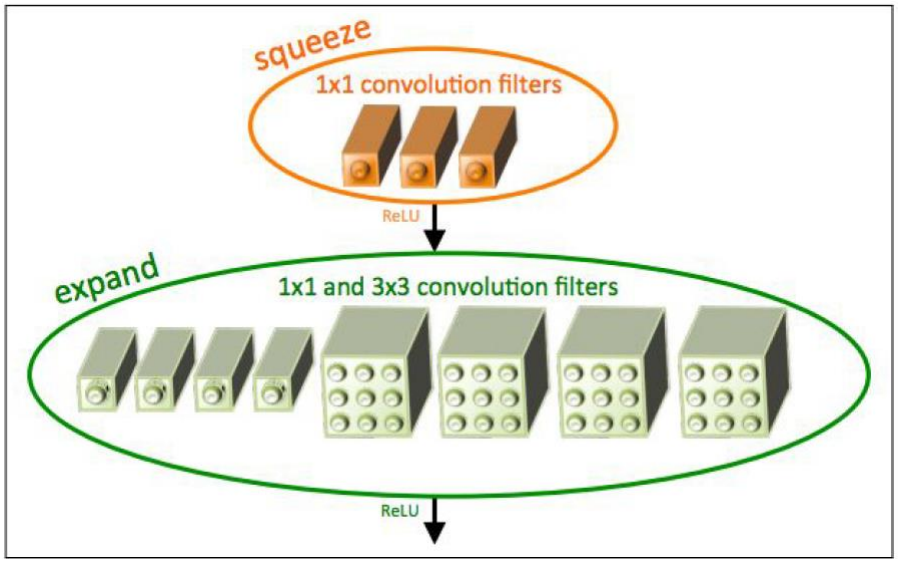
CNN ARCHITECTURES: SQUEEZENET

➢ **Efficient networks...**

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

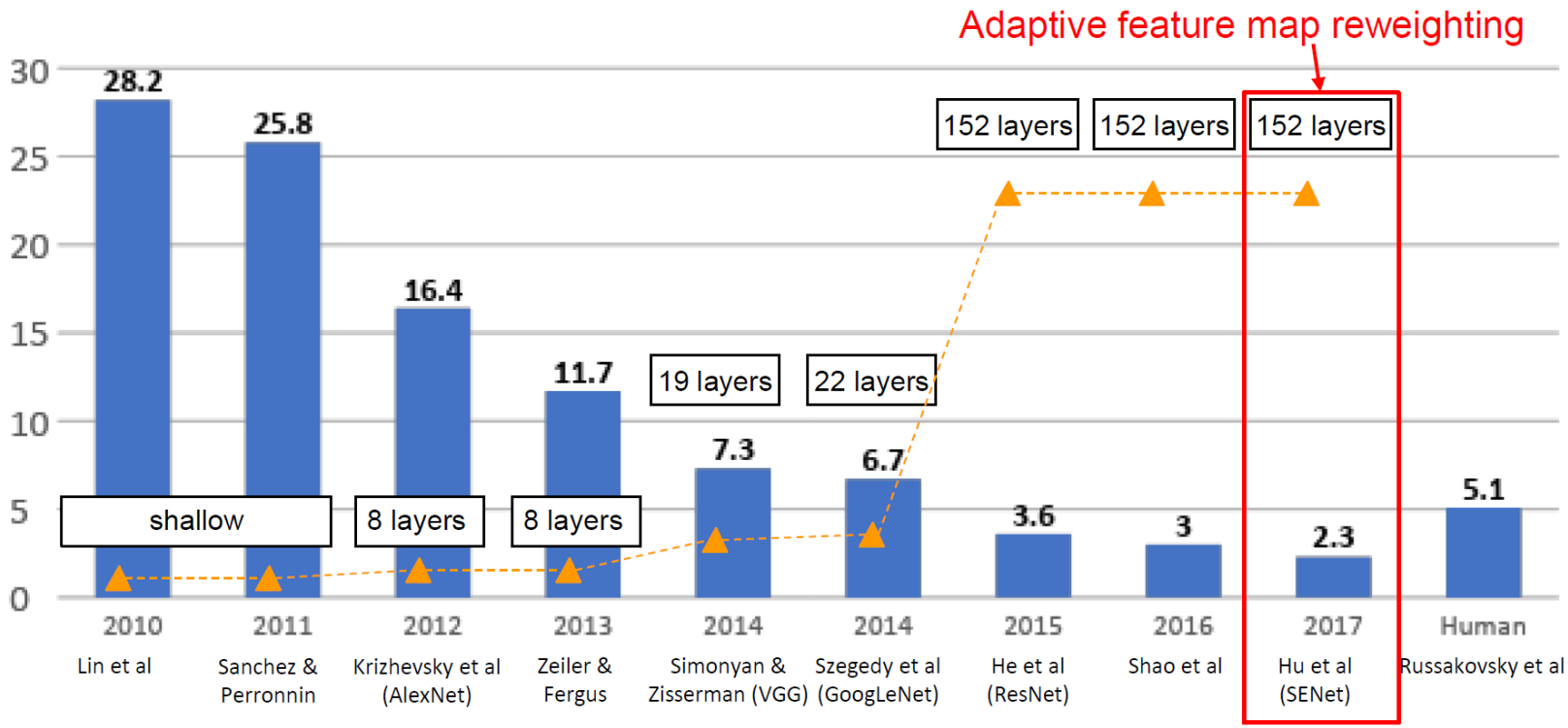
[Iandola et al. 2017]

- 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)



Self Study

CNN ARCHITECTURES: SENET



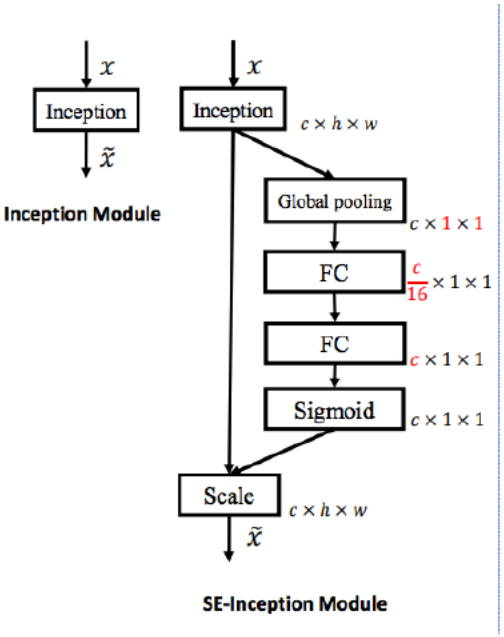
Self Study

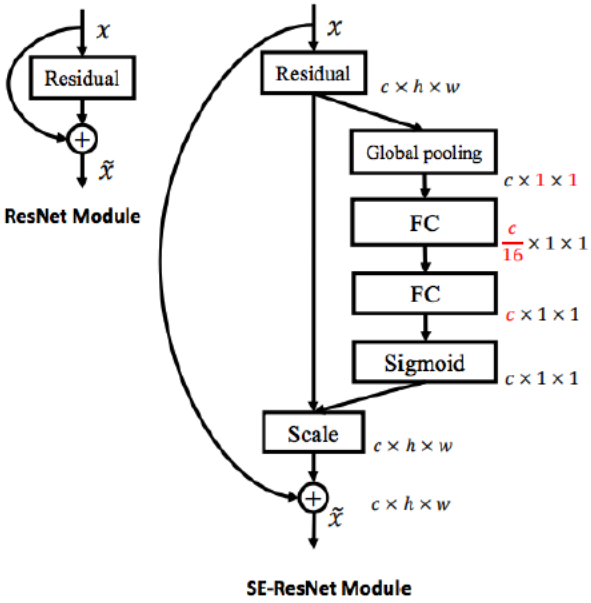
CNN ARCHITECTURES: SENET

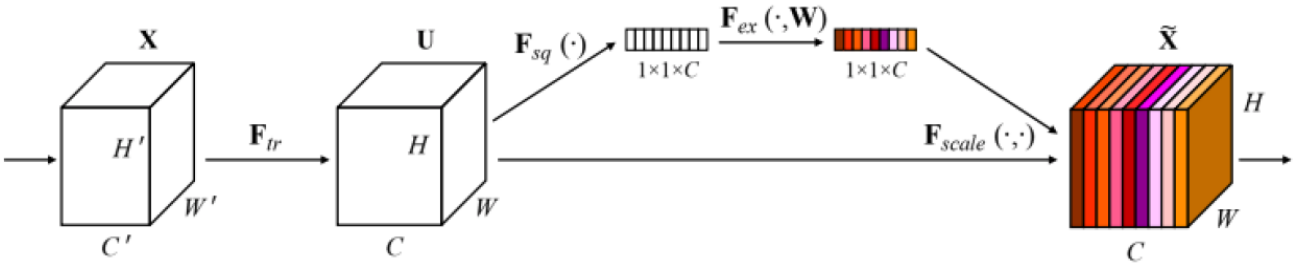
- **Squeeze-and-Excitation Networks (SENet)**

[Hu et al. 2017]

- Add a "feature recalibration" module that learns to adaptively reweight feature maps
- Global information (global avg. pooling layer) + 2 FC layers used to determine feature map weights
- ILSVRC'17 classification winner (using ResNeXt-152 as a base architecture)







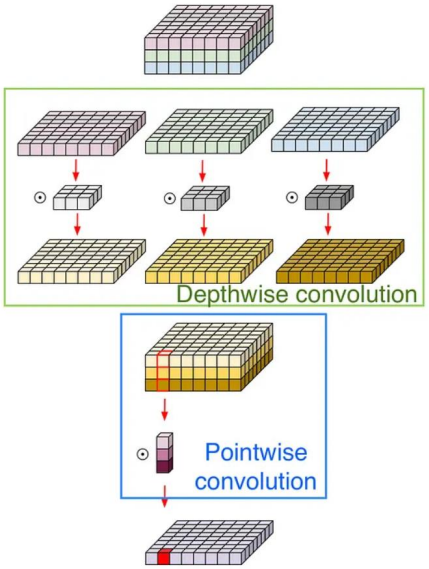
CNN ARCHITECTURES: MOBILENETS

➢ **Efficient networks...**

MobileNets: Efficient Convolutional Neural Networks for Mobile Applications

[Howard et al. 2017]

- MobileNets for mobile and embedded vision applications
- Depthwise separable convolutions replace standard convolutions by factorizing them into a depthwise convolution and a 1x1 convolution
- Much more efficient, with little loss in accuracy
- The depthwise separable convolution splits this into two layers, a separate layer for filtering and a separate layer for combining.
- This factorization has the effect of drastically reducing computation and model size.
- Follow-up MobileNetV2 work in 2018 (Sandler et al.)



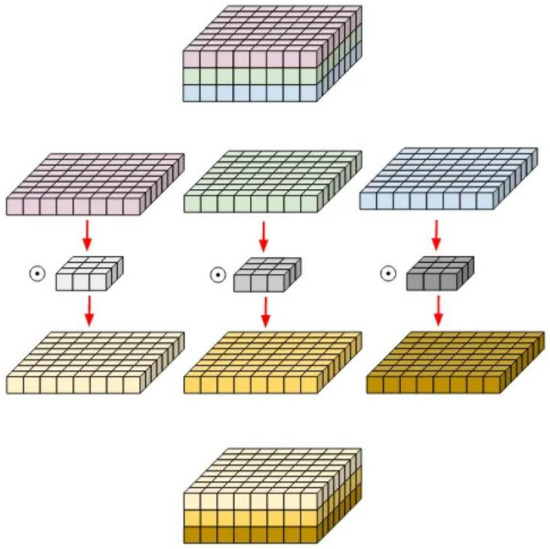
Source: Eli Bendersky

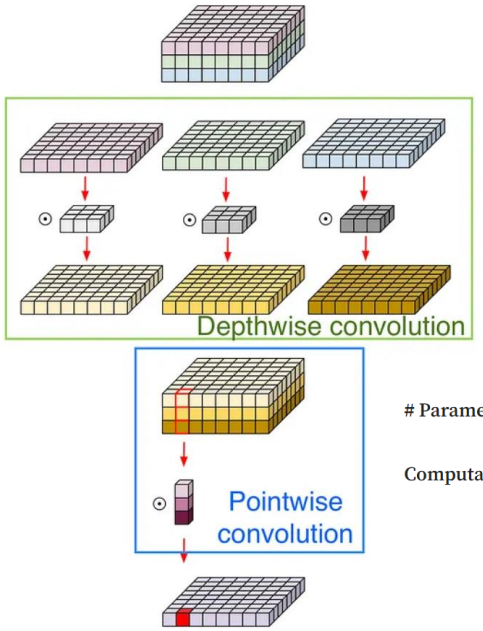
CNN ARCHITECTURES: MOBILENETS

- **Efficient networks... MobileNets:**

Parameters: K×K×C_in

Computation: H×W×C_in×K×K





Parameters: (K×K+C_out)×C_in

Computation: H×W×C_in×(K×K+C_out)

Depth-wise convolution. Filters and image have been broken into three different channels and then convolved separately and stacked thereafter

Source: Eli Bendersky

CNN ARCHITECTURES: MOBILENETS

- **Efficient networks...**
  MobileNets:

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

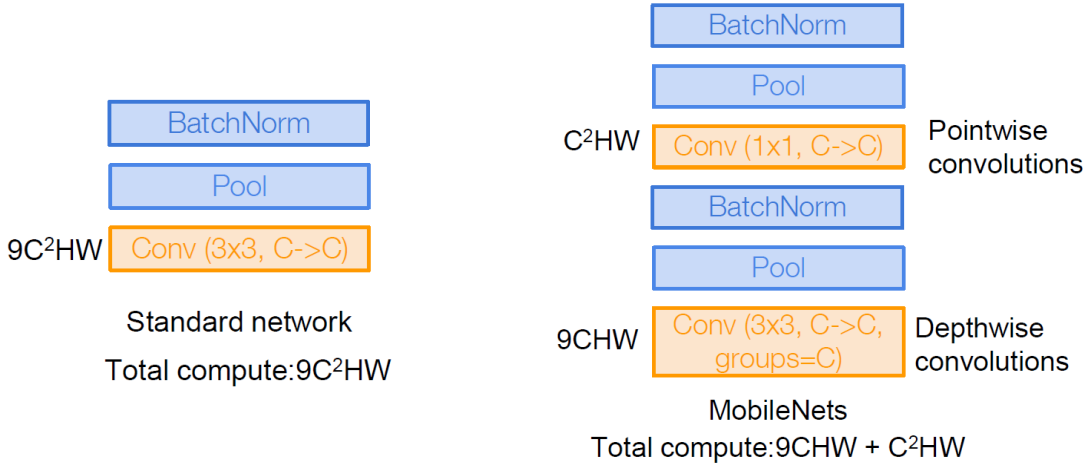
Self Study

CNN ARCHITECTURES: MOBILENETS

- **Efficient networks...**

  MobileNets: Efficient Convolutional Neural Networks for Mobile Applications

[Howard et al. 2017]



CNN ARCHITECTURES: NAS

- **Learning to search for network architectures...**

  Neural Architecture Search with Reinforcement Learning (NAS)

[Zoph et al. 2016]

- A recurrent network to generate the model descriptions of neural networks and train this RNN with reinforcement learning to maximize the expected accuracy of the generated architectures on a validation set
- Our CIFAR-10 model achieves a test error rate of 3.65, which is 0.09 percent better and 1.05x faster than the previous state-of-the-art model that used a similar architectural scheme.

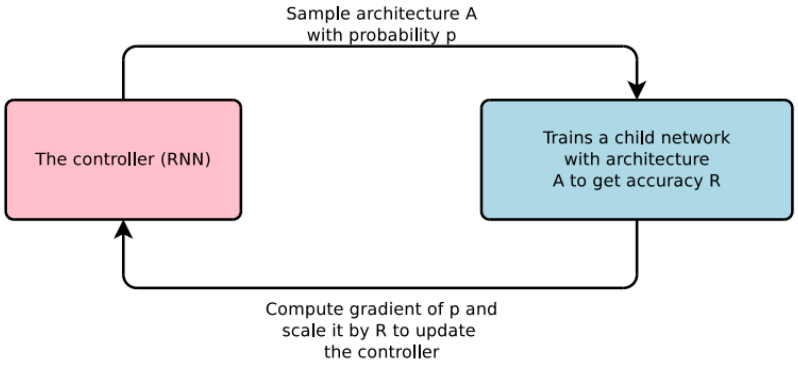
CNN ARCHITECTURES: NAS

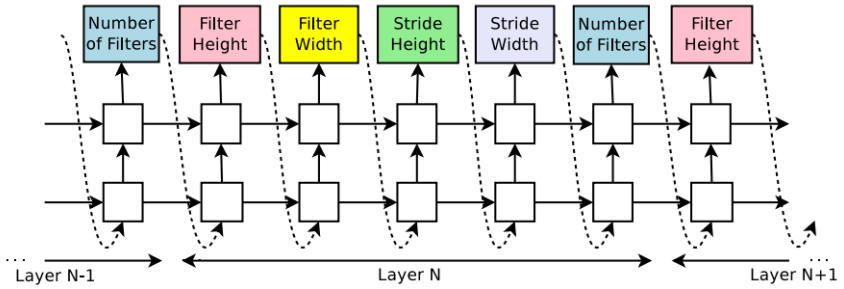
- **Learning to search for network architectures...**

  Neural Architecture Search with Reinforcement Learning (NAS)

[Zoph et al. 2016]

- The controller to generate their hyperparameters as a sequence of tokens
- CIFAR-10 model achieves a test error rate of 3.65, which is 0.09 percent better and 1.05x faster than the previous state-of-the-art model that used a similar architectural scheme





Self Study

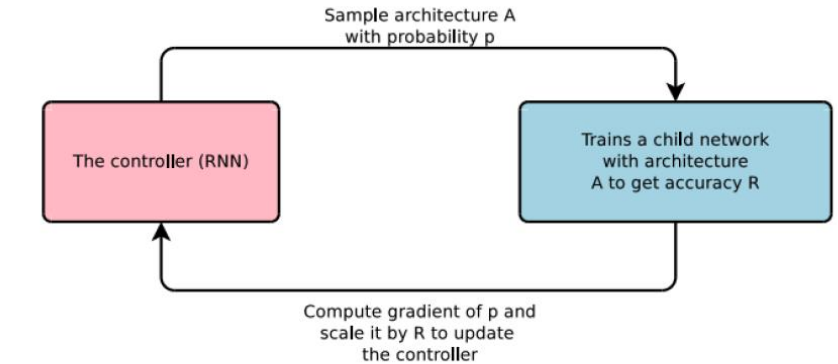
CNN ARCHITECTURES: NAS

➢ **Learning to search for network architectures...**

Neural Architecture Search with Reinforcement Learning (NAS)

[Zoph et al. 2016]

- "Controller" network that learns to design a good network architecture (output a string corresponding to network design)
- Iterate:
  1) Sample an architecture from search space
  2) Train the architecture to get a "reward" R corresponding to accuracy
  3) Compute gradient of sample probability, and scale by R to perform controller parameter update



Self Study

CNN ARCHITECTURES: NAS

- **Learning to search for network architectures...**

  Neural Architecture Search with Reinforcement Learning (NAS)

Performance of Neural Architecture Search and other state-of-the-art models on CIFAR-10.

Model	Depth	Parameters	Error rate (%)
Network in Network (Lin et al., 2013)	-	-	8.81
All-CNN (Springenberg et al., 2014)	-	-	7.25
Deeply Supervised Net (Lee et al., 2015)	-	-	7.97
Highway Network (Srivastava et al., 2015)	-	-	7.72
Scalable Bayesian Optimization (Snoek et al., 2015)	-	-	6.37
FractalNet (Larsson et al., 2016)	21	38.6M	5.22
with Dropout/Drop-path	21	38.6M	4.60
ResNet (He et al., 2016a)	110	1.7M	6.61
ResNet (reported by Huang et al. (2016c))	110	1.7M	6.41
ResNet with Stochastic Depth (Huang et al., 2016c)	110	1.7M	5.23
	1202	10.2M	4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16	11.0M	4.81
	28	36.5M	4.17
ResNet (pre-activation) (He et al., 2016b)	164	1.7M	5.46
	1001	10.2M	4.62
DenseNet ($L = 40, k = 12$) Huang et al. (2016a)	40	1.0M	5.24
DenseNet($L = 100, k = 12$) Huang et al. (2016a)	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) Huang et al. (2016a)	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) Huang et al. (2016b)	190	25.6M	3.46
Neural Architecture Search v1 no stride or pooling	15	4.2M	5.50
Neural Architecture Search v2 predicting strides	20	2.5M	6.01
Neural Architecture Search v3 max pooling	39	7.1M	4.47
Neural Architecture Search v3 max pooling + more filters	39	37.4M	3.65

Self Study

CNN ARCHITECTURES: NASNET

- **Learning to search for network architectures...**

  Learning Transferable Architectures for Scalable Image Recognition

[Zoph et al. 2017]

- The main search method has been used in this work is the Neural Architecture Search (NAS) framework
- The main contribution of this work is the design of a novel search space, such that the best architecture found on the CIFAR-10 dataset would scale to larger, higher resolution image datasets across a range of computational settings.
- We name this search space the NASNet search space as it gives rise to NASNet, the best architecture found in our experiments.
- Our model is 1.2% better in top-1 accuracy than the best human-invented architectures while having 9 billion fewer FLOPS – a reduction of 28% in computational demand from the previous state-of-the-art model.

Self Study

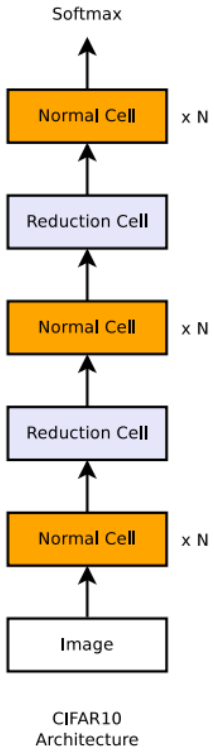
CNN ARCHITECTURES: NASNET

- **Learning to search for network architectures...**

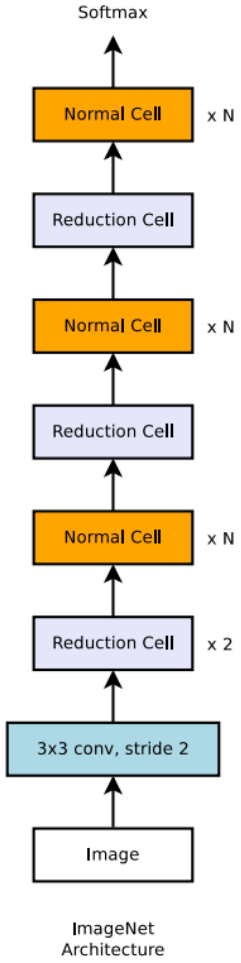
  Learning Transferable Architectures for Scalable Image Recognition

  [Zoph et al. 2017]

  - Applying neural architecture search (NAS) to a large dataset like ImageNet is expensive
  - Design a search space of building blocks (“cells”) that can be flexibly stacked



CIFAR10 Architecture



ImageNet Architecture

CNN ARCHITECTURES: EFFICIENTNET

EfficientNet: Smart Compound Scaling

[Tan and Le. 2019]

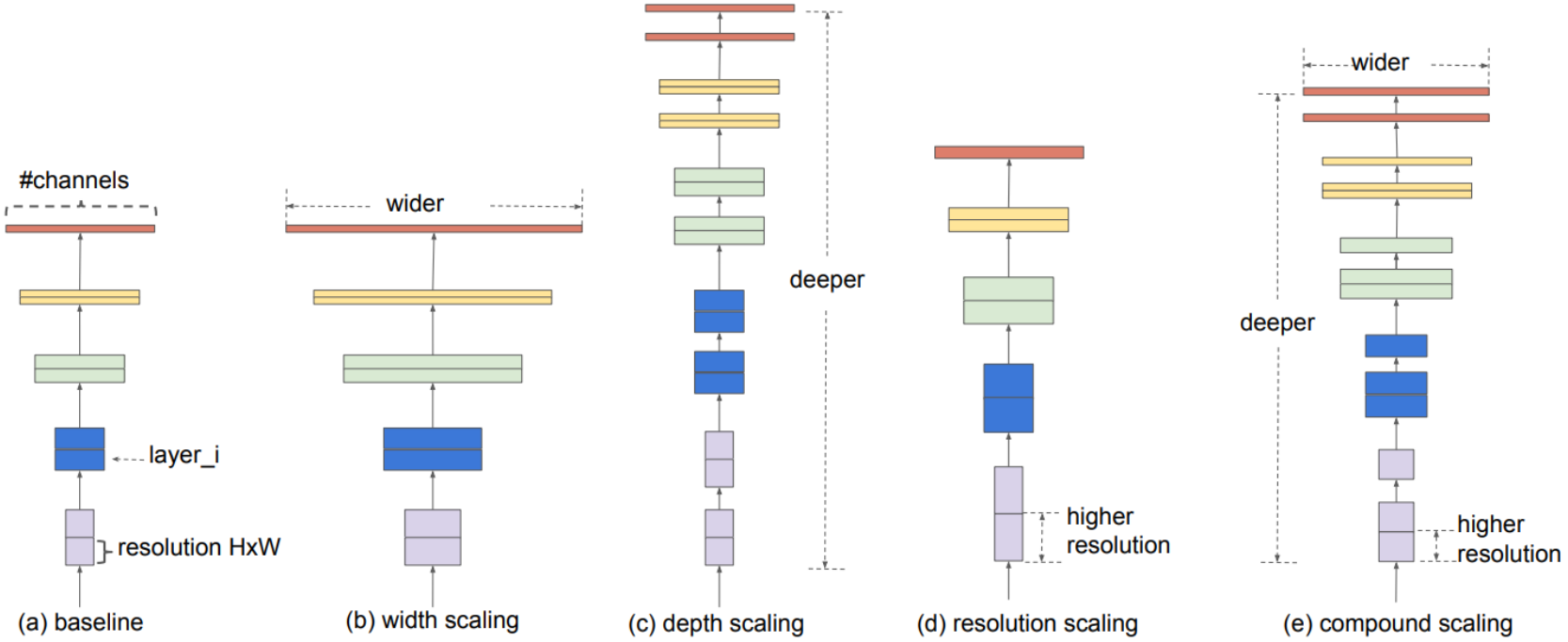
- **Question**: is there a principled method to scale up ConvNets that can achieve better accuracy and efficiency?
  - It is critical to balance all dimensions of network width/depth/resolution, and surprisingly such balance can be achieved by simply scaling each of them with constant ratio. Based on this observation, we propose a simple yet effective compound scaling method.
  - Unlike conventional practice that arbitrary scales these factors, our method uniformly scales network width, depth, and resolution with a set of fixed scaling coefficients.

    For example, if we want to use 2^N times more computational resources, then we can simply increase the network depth by α^N, width by β^N, and image size by γ^N, where α, β, γ are constant coefficients determined by a small grid search on the original small model. Figure in the next slide illustrates the difference between our scaling method and conventional methods.

CNN ARCHITECTURES: EFFICIENTNET

- **EfficientNet: Smart Compound Scaling**

[Tan and Le. 2019]



CNN ARCHITECTURES: EFFICIENTNET

➢EfficientNet: Smart Compound Scaling

[Tan and Le. 2019]

- A new scaling method that uniformly scales all dimensions of depth/width/resolution using a simple yet highly effective compound coefficient.
- We demonstrate the effectiveness of this method on scaling up **MobileNets** and **ResNet**.
- To go even further, we use **neural architecture search** to design a new baseline network and scale it up to obtain a family of models, called EfficientNets, which achieve much better accuracy and efficiency than previous ConvNets.
- EfficientNet-B7 achieves state-of-the-art 84.3% top-1 accuracy on ImageNet, while being **8.4x smaller and 6.1x faster** on inference than the **best existing ConvNet**.
- EfficientNets also transfer well and achieve state-of-the-art accuracy on CIFAR-100 dataset (91.7%)

CNN ARCHITECTURES: EFFICIENTNET

➢EfficientNet: Smart Compound Scaling

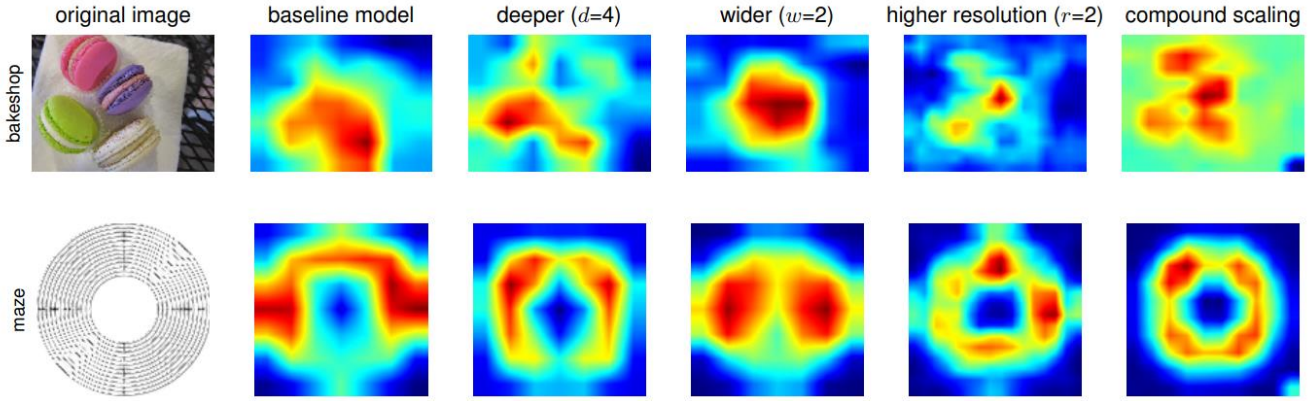


Figure: Class Activation Map (CAM) (Zhou et al., 2016) for Models with different scaling methods- Our compound scaling method allows the scaled model (last column) to focus on more relevant regions with more object details.

Table: Scaled Models Used in Figure

Model	FLOPS	Top-1 Acc.
Baseline model (EfficientNet-B0)	0.4B	77.3%
Scale model by depth (d=4)	1.8B	79.0%
Scale model by width (w=2)	1.8B	78.9%
Scale model by resolution (r=2)	1.9B	79.1%
Compound Scale (d=1.4, w=1.2, r=1.3)	**1.8B**	**81.1%**

Self Study

CNN ARCHITECTURES: EFFICIENTNET

➢ **EfficientNet: Smart Compound Scaling**

[Tan and Le. 2019]

- Increase network capacity by scaling width, depth, and resolution, while balancing accuracy and efficiency.
- Search for optimal set of compound scaling factors given a compute budget (target memory & flops).
- Scale up using smart heuristic rules

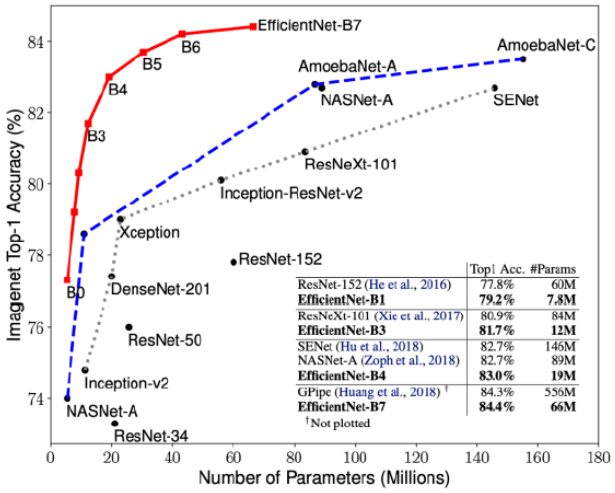
$$\text{depth: } d = \alpha^\phi$$

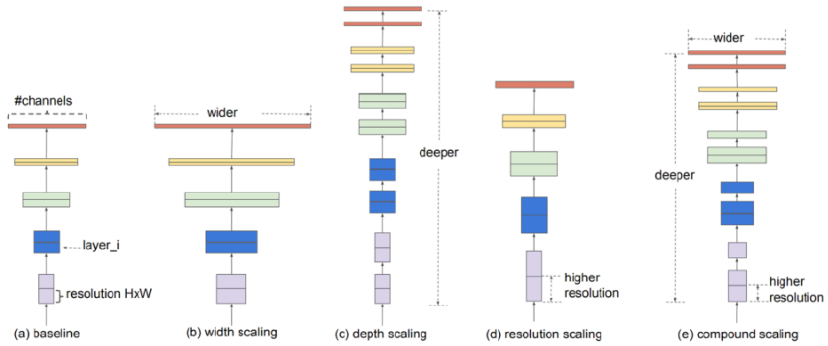
$$\text{width: } w = \beta^\phi$$

$$\text{resolution: } r = \gamma^\phi$$

$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$





TOWARDS PRINCIPLED DESIGN OF DEEP CONVOLUTIONAL NETWORKS

- **Gradual Expansion with Minimum Allocation**
  - Processing level saturation haunts shallower archictures
  - Processing level deprivation haunts deep architectures

TABLE II: Gradual expansion of the number of layers.

Network Properties	Parameters	Accuracy (%)
Arch1, 8 Layers	300K	90.21
Arch1, 9 Layers	300K	90.55
Arch1, 10 Layers	300K	90.61
Arch1, 13 Layers	300K	89.78

TABLE III: Shallow *vs*. Deep (related to *Minimum Allocations*), showing how a gradual increase can yield better performance with fewer number of parameters.

Network Properties	Parameters	Accuracy (%)
Arch1, 6 Layers	1.1M	92.18
Arch1, 10 Layers	570K	92.23

Hasanpour, S.H., Rouhani, M., Fayyaz, M., Sabokrou, M. and Adeli, E., 2018. Towards principled design of deep convolutional networks: Introducing simpnet.

TOWARDS PRINCIPLED DESIGN OF DEEP CONVOLUTIONAL NETWORKS

- **Homogenous groups of layers**
  - symmetric and homogeneous design, allows to easily manage the number of parameters
  - providing better information pools for each semantic level
  - Helps to achieve granular fine-tuning
  - Better management of processing unit distribution
  - Former architectures use it implicitly but have not taken the full advantage of such thing.

TOWARDS PRINCIPLED DESIGN OF DEEP CONVOLUTIONAL NETWORKS

- **Local Correlation Preservation**
  - avoiding 1 × 1 kernels in early layers
  - Correlation the corner stone of CNN
  - View point of SqueezeNet?
    - Server botteleneck which destroy a lot of useful information
    - It also can not scale property

Correlation Preservation: SqueezeNet vs. SimpNet on CIFAR10.

Network	Params	Accuracy (%)
SqueezeNet1.1_default	768K	88.60
SqueezeNet1.1_optimized	768K	92.20
SimpNet_Slim	300K	93.25
SimpNet_Slim	600K	94.03

TOWARDS PRINCIPLED DESIGN OF DEEP CONVOLUTIONAL NETWORKS

- **Maximum information utilization**
  - Sources for more data and information:
    - Larger datasets
    - Data-augmentation
    - Better information pools
  - Information pools
    - Resblock
    - Densblock
    - Pooling fusion
  - Simple form of information pool
    - Larger input/feature-maps
  - What if we don't have access to more data?
    - Utilize what you have in the most efficient way
  - how?

    Avoid rapid down-sampling/doing pooling
    - Larger feature-maps provide more information
    - Instead of adding more complexity and increasing depth or width, increase input dimension first

TOWARDS PRINCIPLED DESIGN OF DEEP CONVOLUTIONAL NETWORKS

- **Strided convolution vs Maxpooling**

TABLE VI: Effect of using strided convolution (†) *vs*. Max-pooling (∗). Max-pooling outperforms the strided convolution regardless of specific architecture. First three rows are tested on CIFAR100 and two last on CIFAR10.

Network Properties	Depth	Parameters	Accuracy (%)
SimpNet∗	13	360K	69.28
SimpNet∗	15	360K	68.89
SimpNet†	15	360K	68.10
ResNet∗	32	460K	93.75
ResNet†	32	460K	93.46

Self Study

TOWARDS PRINCIPLED DESIGN OF DEEP CONVOLUTIONAL NETWORKS

- **Maximum performance utilization**

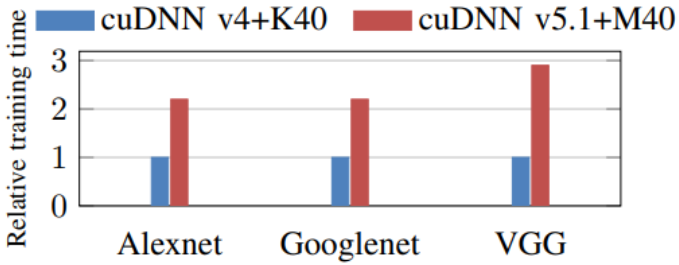


Fig. 1: Comparisons of the speedup of different architectures. This plot shows 2.7× faster training when using 3×3 kernels using cuDNN v5.x.

TABLE I: Improved performance by utilizing cuDNN v7.x.

	k80+cuDNN 6	P100+ cuDNN 6	v100+cuDNN 7
2.5x + CNN	100	200	600
3x + LSTM	1x	2x	6x

TOWARDS PRINCIPLED DESIGN OF DEEP CONVOLUTIONAL NETWORKS

- **Balanced Distribution Scheme**

TABLE IV: Balanced distribution scheme is demonstrated by using two variants of SimpNet architecture with 10 and 13 layers, each showing how the difference in allocation results in varying performance and ultimately improvements for the one with balanced distribution of units.

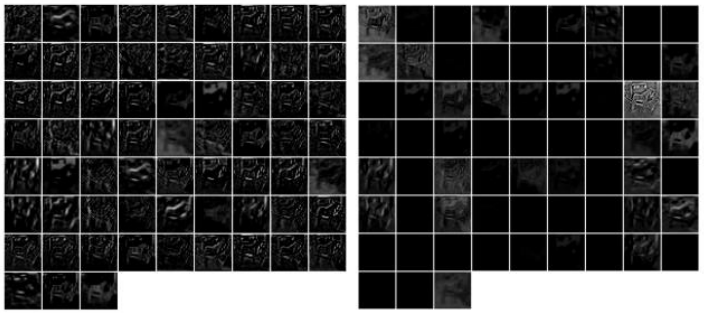
Network Properties	Parameters	Accuracy (%)
Arch2, 10 Layers (wide end)	8M	95.19
Arch2, 10 Layers (balanced width)	8M	95.51
Arch2, 13 Layers (wide end)	128K	87.20
Arch2, 13 Layers (balanced width)	128K	89.70

TOWARDS PRINCIPLED DESIGN OF DEEP CONVOLUTIONAL NETWORKS

- **Dropout Utilization**
  - Start with a small dropout probability e.g. 0.2 (especially if you are using BN)
  - What's the effect of dropout applied on all layers?
    - Results in better generalization by creating diverse feature compositions
    - Creates more robust filters
    - Becomes more robust against noise and different changes in input
    - Prevents dead ReLU issue
    - Provides more sparsity in the network which enhances generalization

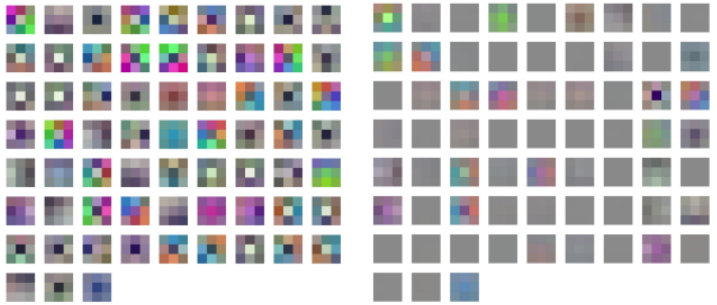
TOWARDS PRINCIPLED DESIGN OF DEEP CONVOLUTIONAL NETWORKS

- **Dropout Utilization**



(a) With dropout. (b) Without dropout.

Fig. 3: When dropout is used, all neurons are forced to take part and thus less dead neurons occur. It also results in the development of more robust and diverse filters.



(a) Filters learned in the first layer, when dropout is used. (b) Filters learned in the first layer, when dropout is not used.

Fig. 4: When dropout is not used, filters are not as vivid and more dead units are encountered, which is an indication of the presence of more noise.

Self Study

TOWARDS PRINCIPLED DESIGN OF DEEP CONVOLUTIONAL NETWORKS

- **Dropout Utilization**

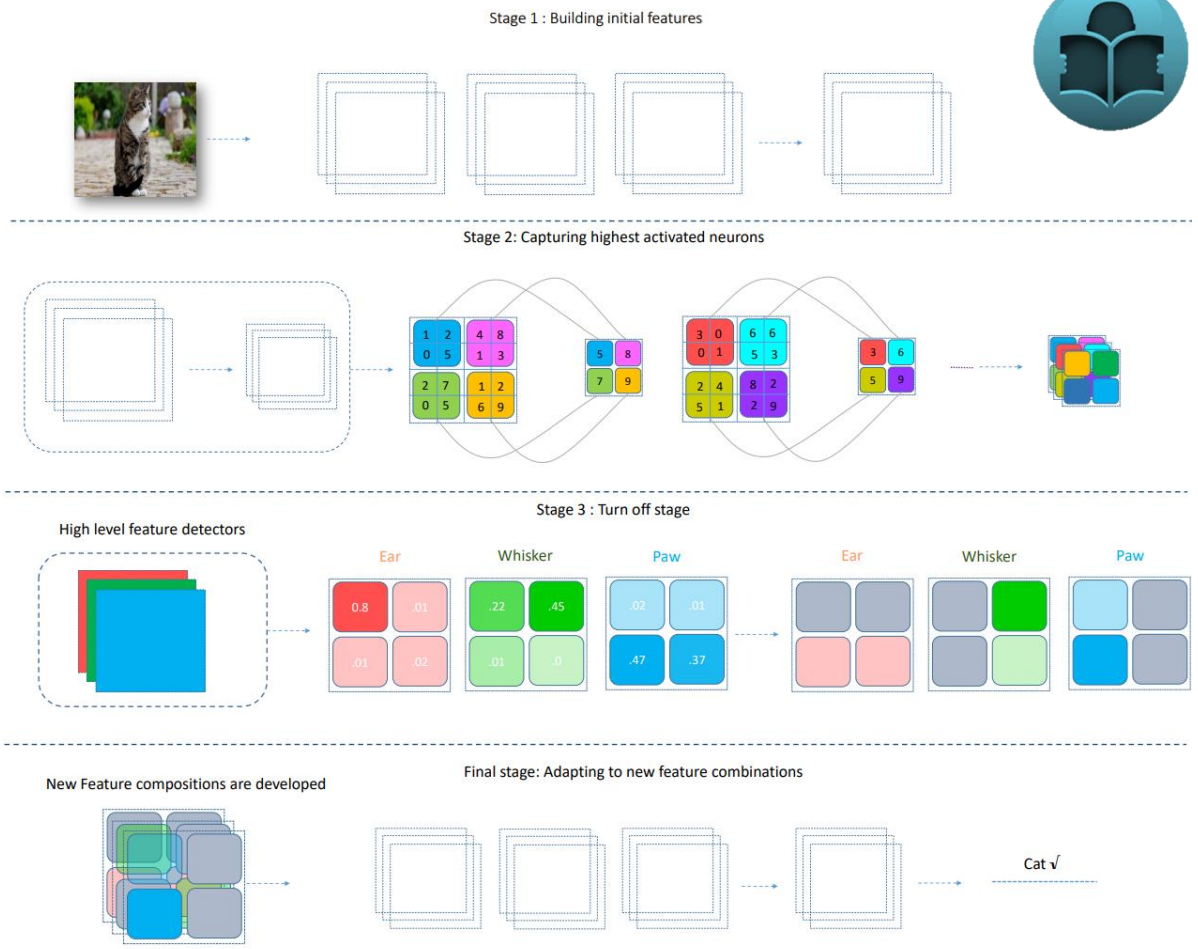


Fig. S1: SAF-Pooling: In this scheme, the network is forced to adapt itself to several different feature combinations through deterministically deactivating some feature responses, so that in cases some features are absent, due to occlusion, viewpoint changes, noise, *etc*., the network can still carry out its task using other available features. The process happens in several stages. After some mid/high level features are available (Stage 1), most prominent features (responses) are pooled (Stage 2). Then, some of them are deterministically turned off. The network now has to adapt to the absence of some prominent features and therefore develop new feature compositions (Stage 3). After several iterations, less relevant (or irrelevant) features will be gradually replaced by more informative ones as weaker features are more frequently given the chance to be further developed. As the cycle continues, more feature compositions are developed which ultimately results in the network maximizing the capacity utilization, by creating more diverse feature detectors, as more neurons will be sparsely activated and independently learn patterns.

TOWARDS PRINCIPLED DESIGN OF DEEP CONVOLUTIONAL NETWORKS

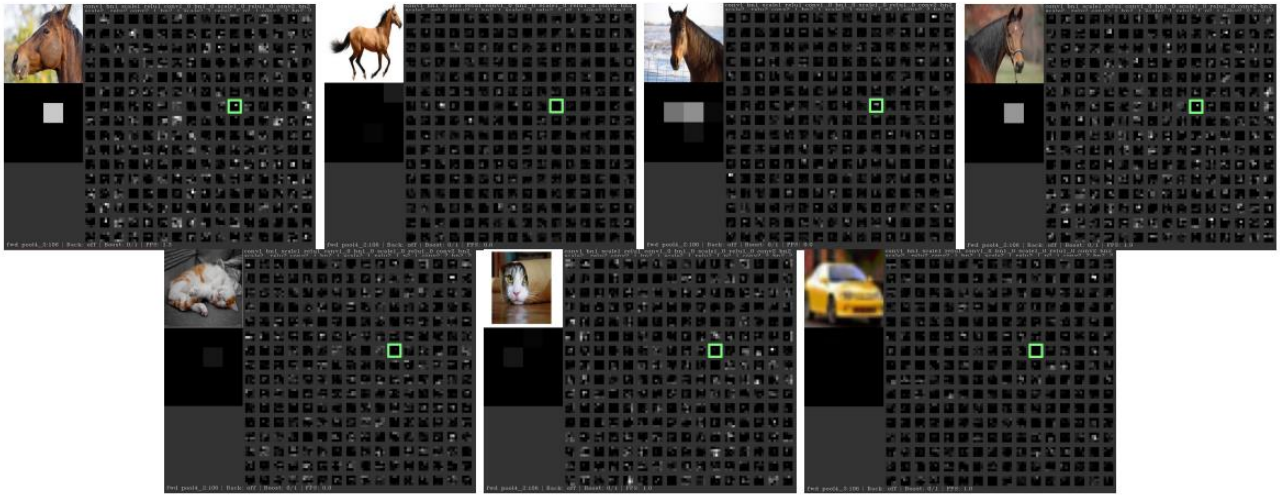


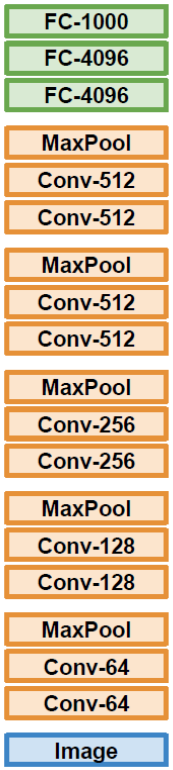
Fig. S2: Throughout the network, the notion of input changes into a feature-map, in which each response denotes the presence of a specific feature. Using Max-pooling, it is possible to filter out responses and capture the most prominent ones and use them to form interesting feature compositions later in the pipeline.

TRANSFER LEARNING WITH CNNS

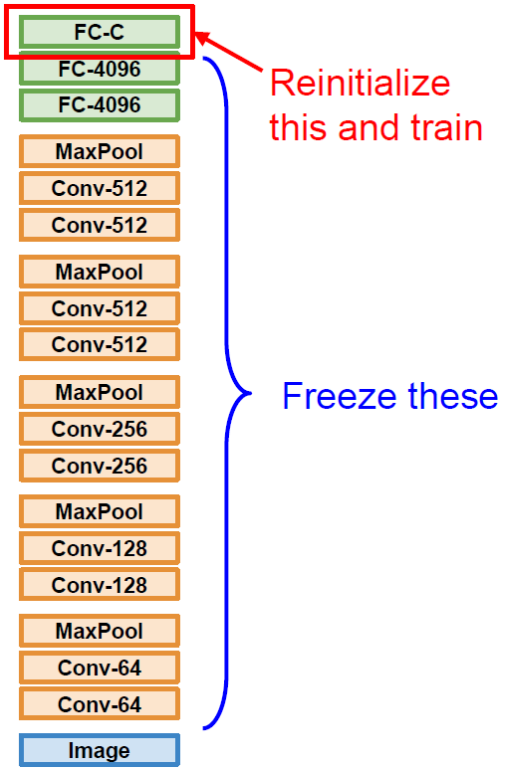
- Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014
- Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

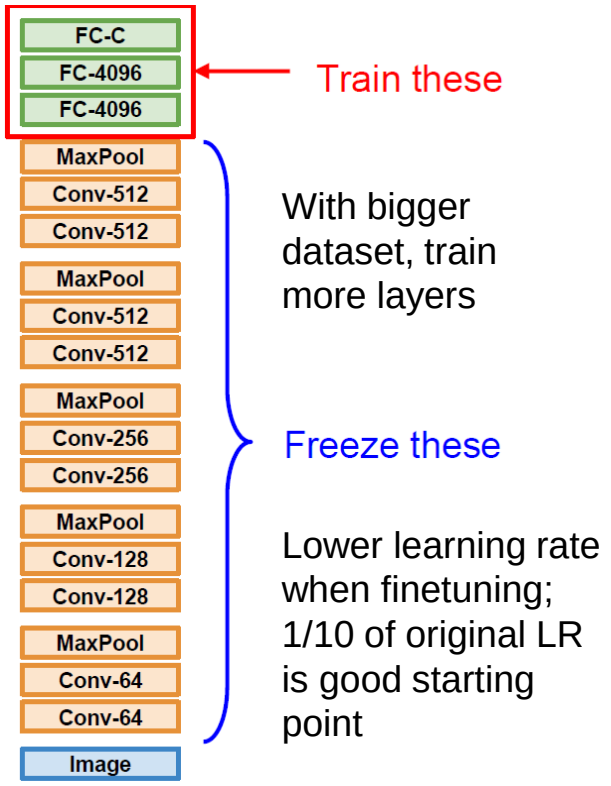
➢ **You need a lot of a data if you want to train/use CNNs?**

1. Train on Imagenet



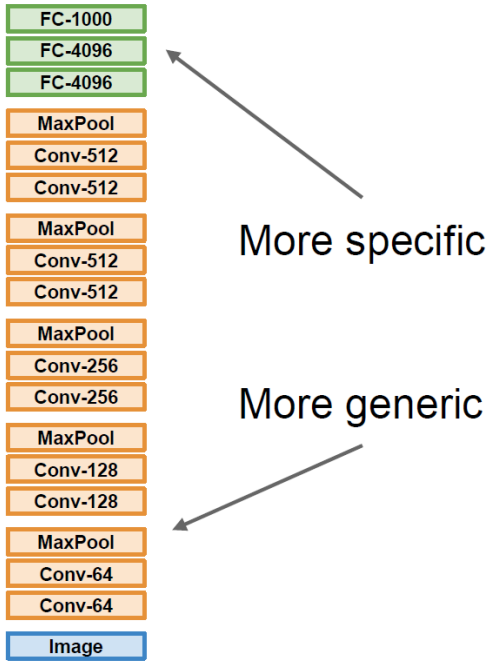
2. Small Dataset (C classes)





Self Study

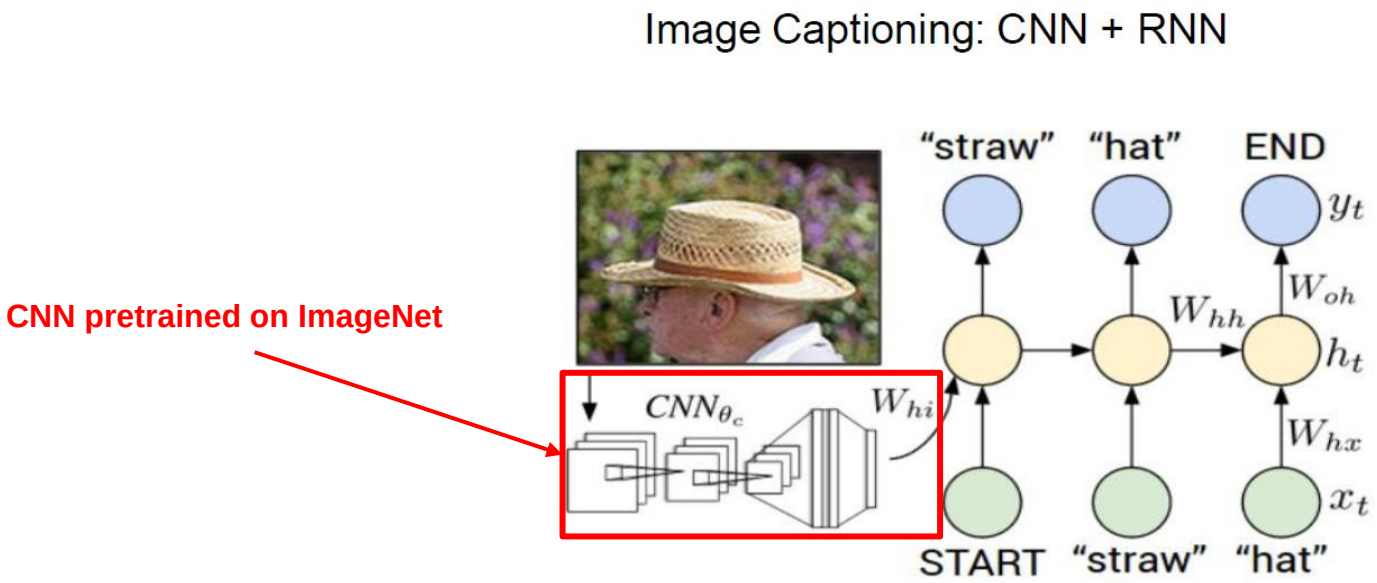
TRANSFER LEARNING WITH CNNS



	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers or start from scratch!

TRANSFER LEARNING WITH CNNS

- **Transfer learning with CNNs is pervasive…**



Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

TRANSFER LEARNING WITH CNNS

- **Takeaway for your projects and beyond:**
  - Have some dataset of interest but it has < ~1M images?
    1. Find a very large dataset that has similar data, train a big model there
    2. Transfer learn to your dataset
  - Deep learning frameworks provide a "Model Zoo" of pretrained models so you don't need to train your own
  - **TensorFlow**: https://github.com/tensorflow/models
  - **PyTorch**: https://github.com/pytorch/vision/tree/main/torchvision/models