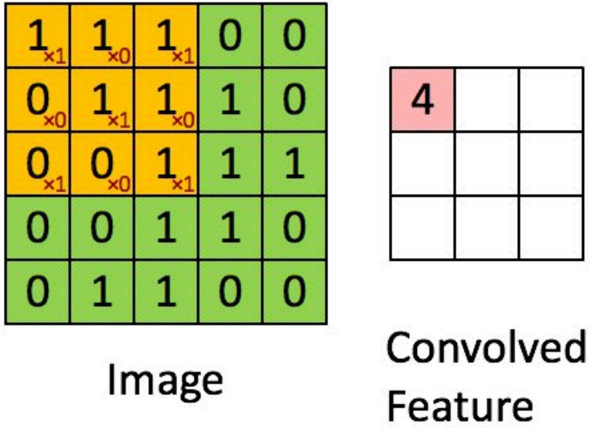
Convolutional Neural Networks

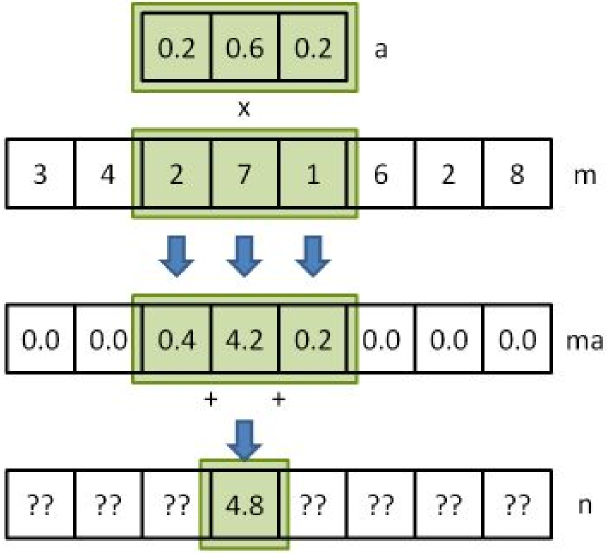
Convolution, LeNet, AlexNet, VGGNet, GoogleNet, Resnet, Deconvolution

Feb 1, 2017

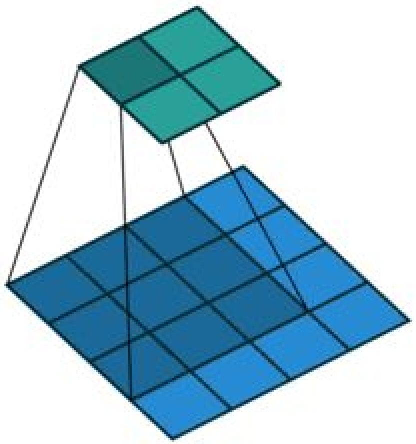
Aaditya Prakash

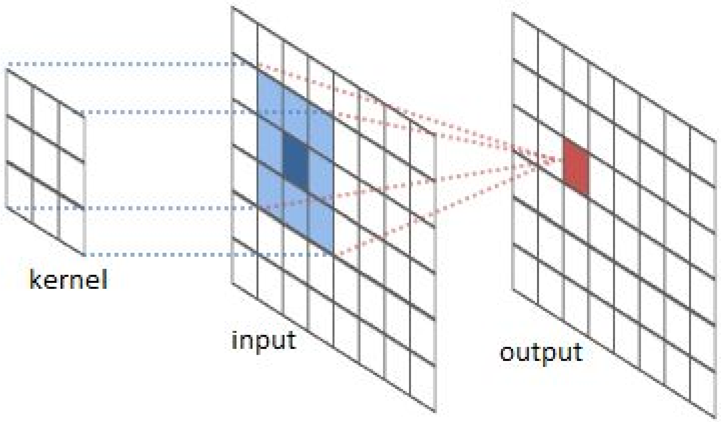
Convolution





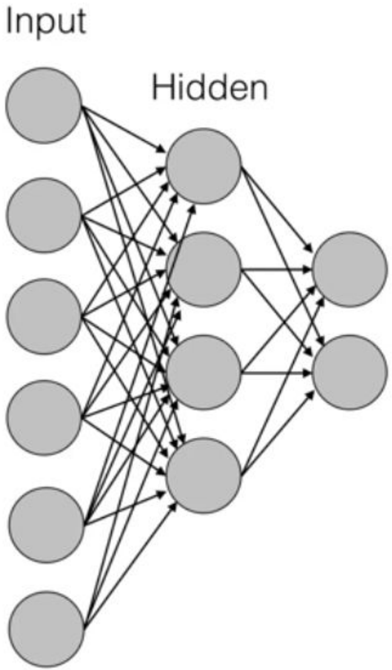
Convolution



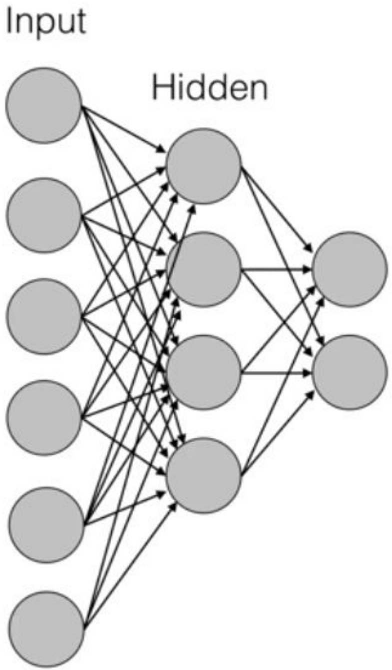


Demo Convolution

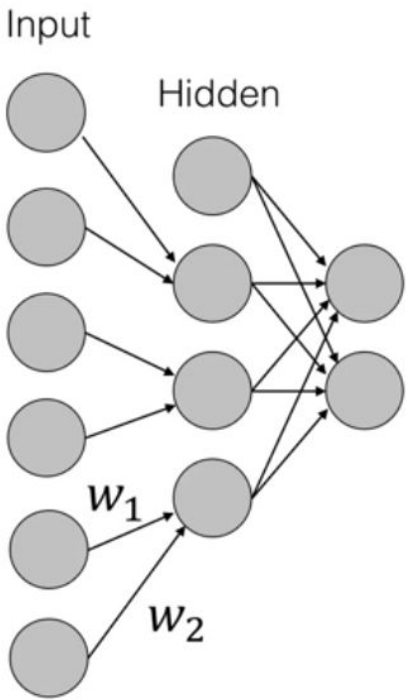
Convolution in Neural Networks



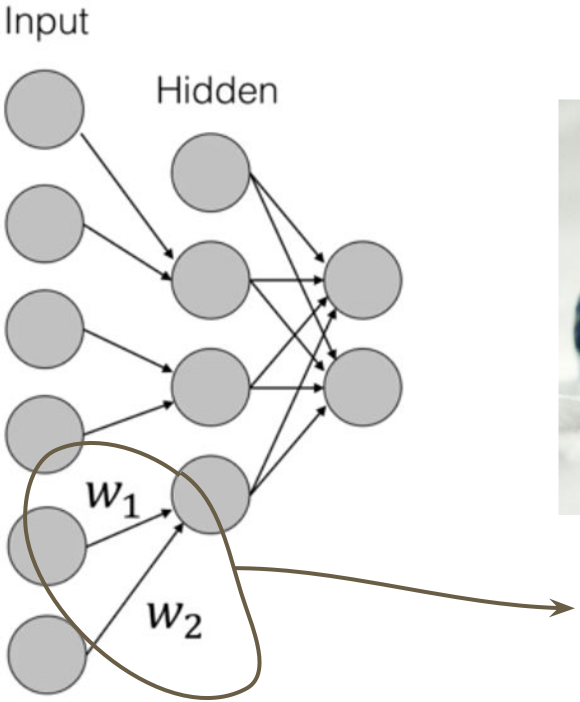
Convolution in Neural Networks



Convolution in Neural Networks

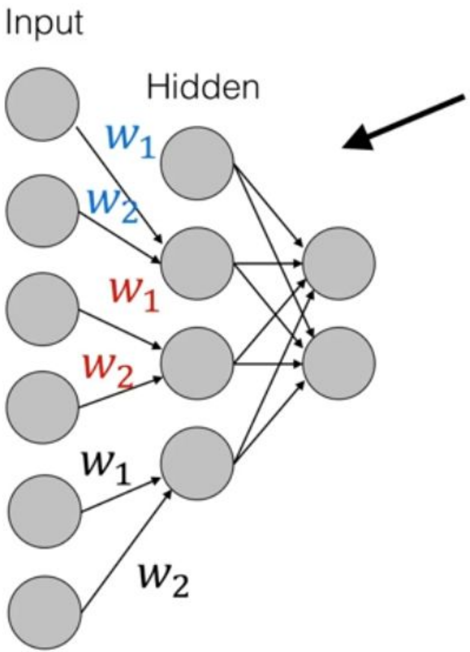


Convolution in Neural Networks



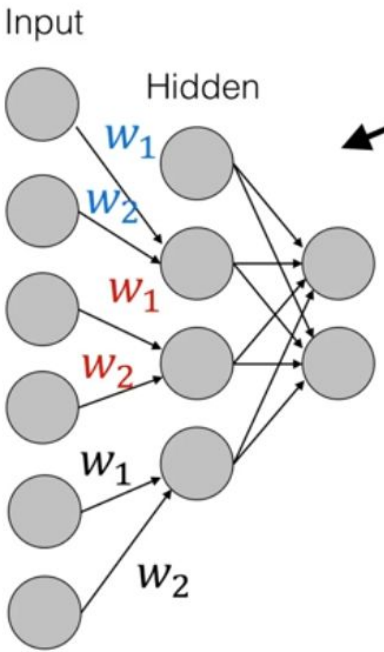
$$y = w_1 x_1 + w_2 x_2$$

Stride 1-D



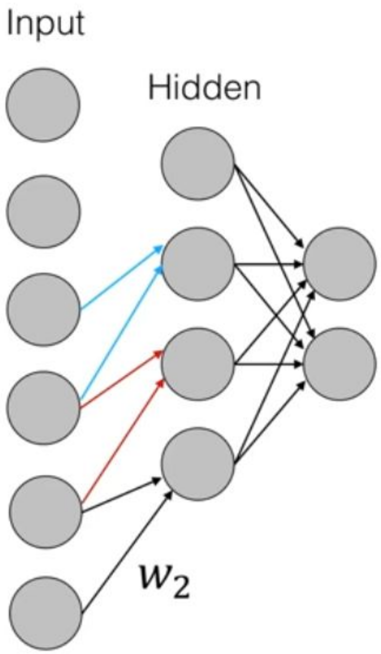
1-d convolution with
- filters: 1
- filter size: 2
- stride: 2

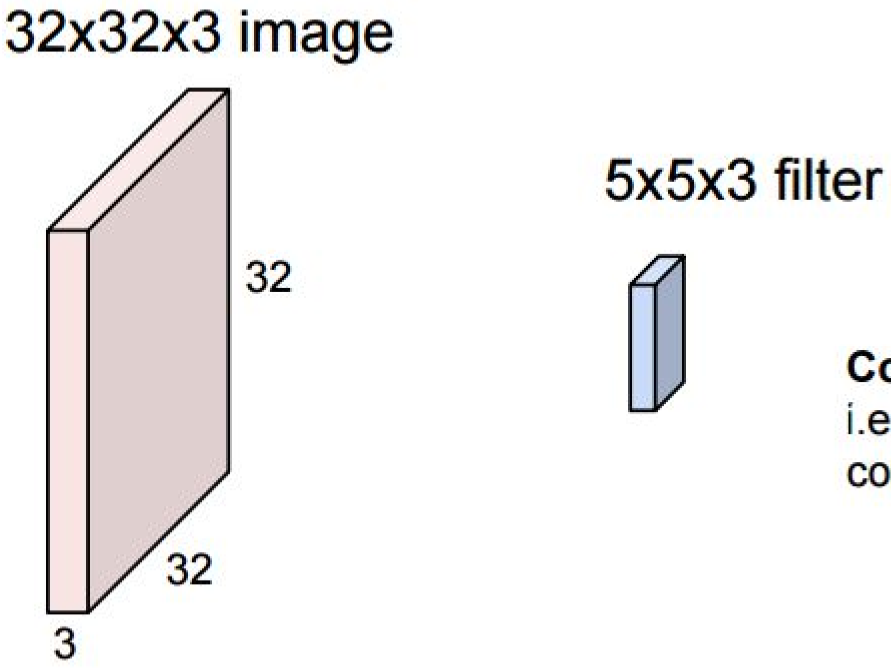
Stride 1-D



1-d convolution with
- filters: 1
- filter size: 2
- stride: 2

1-d convolution with
- filters: 1
- filter size: 2
- stride: **1**

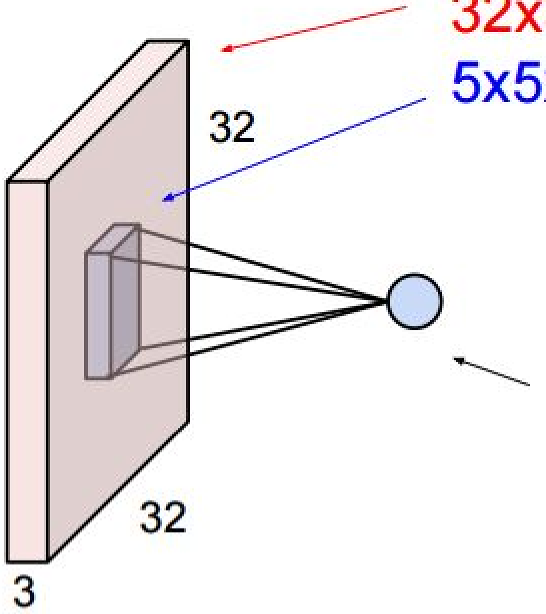




Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

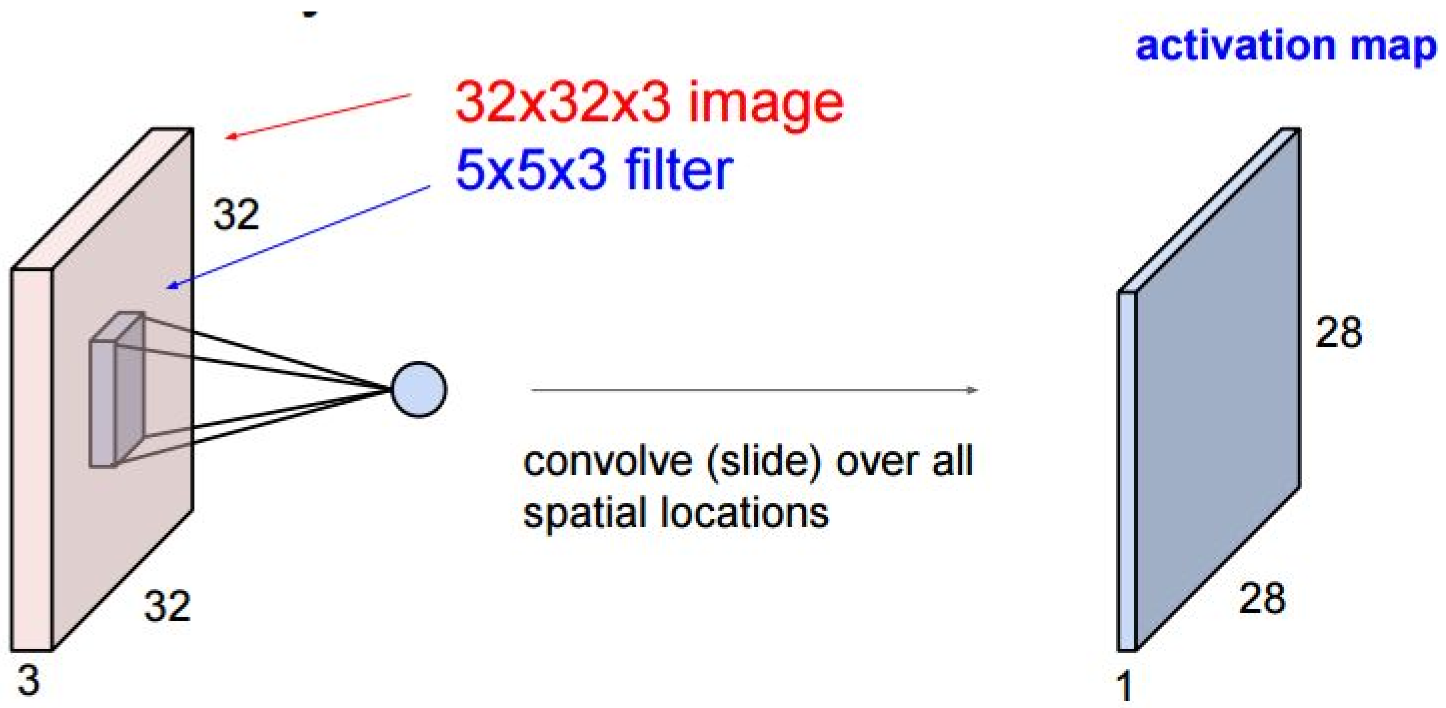
32x32x3 image

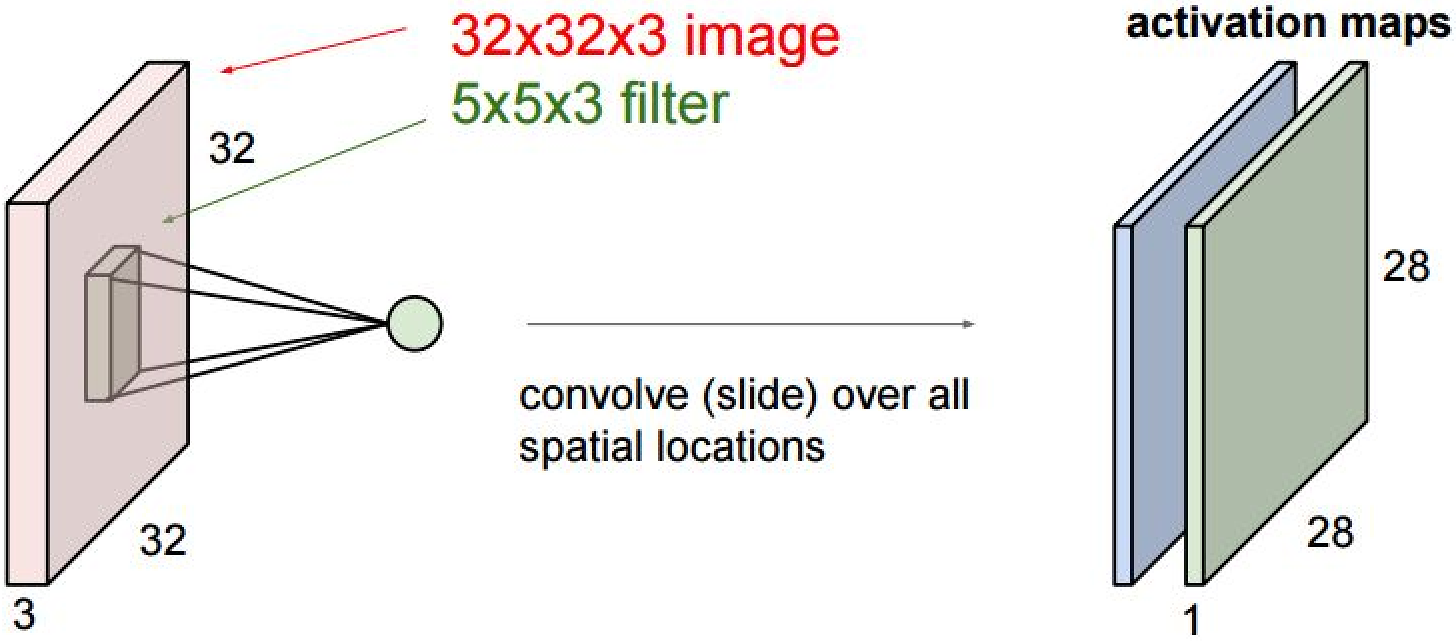
5x5x3 filter w



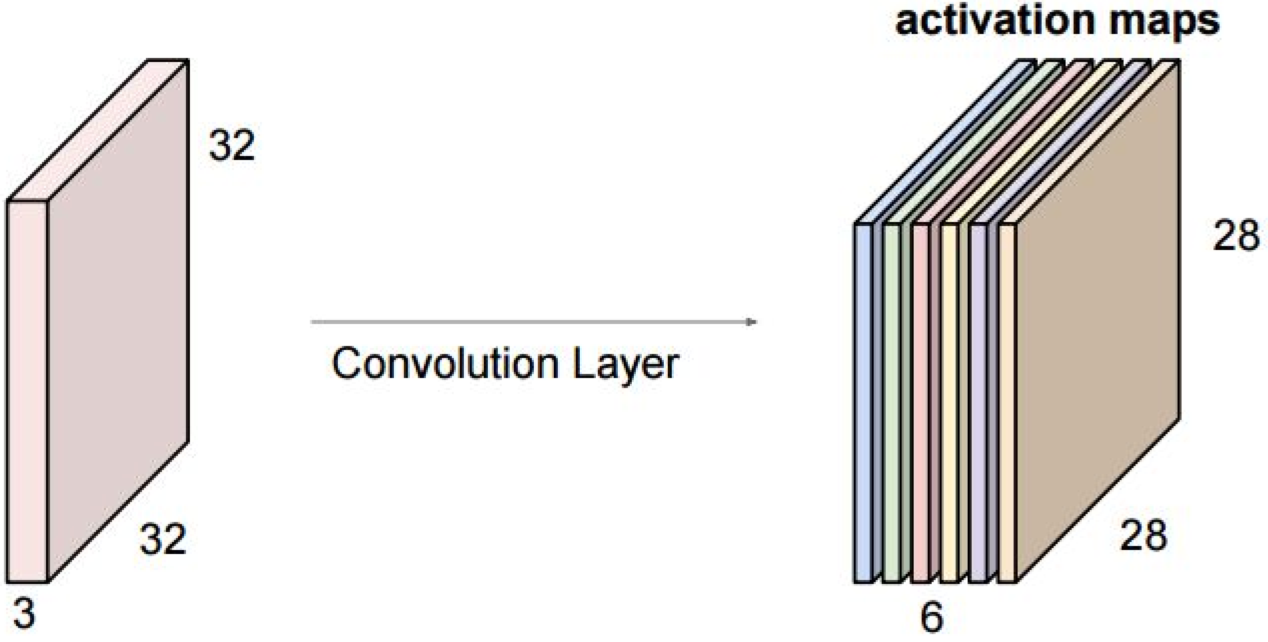
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

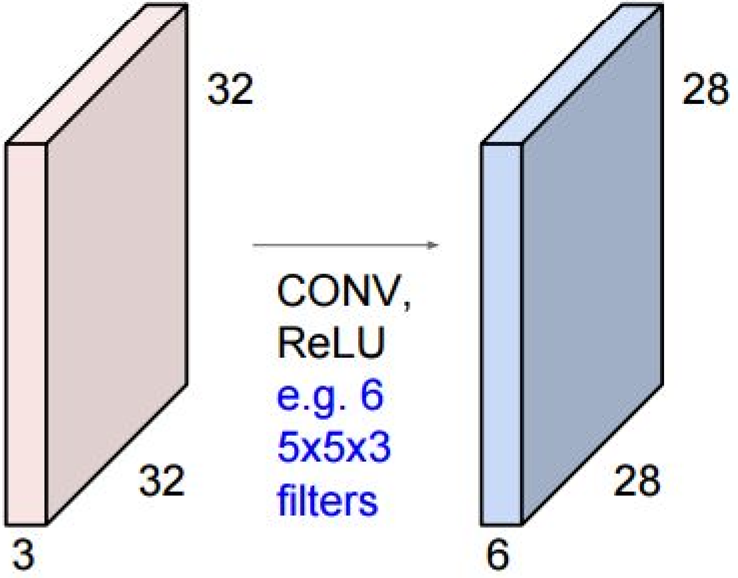

activation map
32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all spatial locations
28
28
1


32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all spatial locations
activation maps
28
28
1

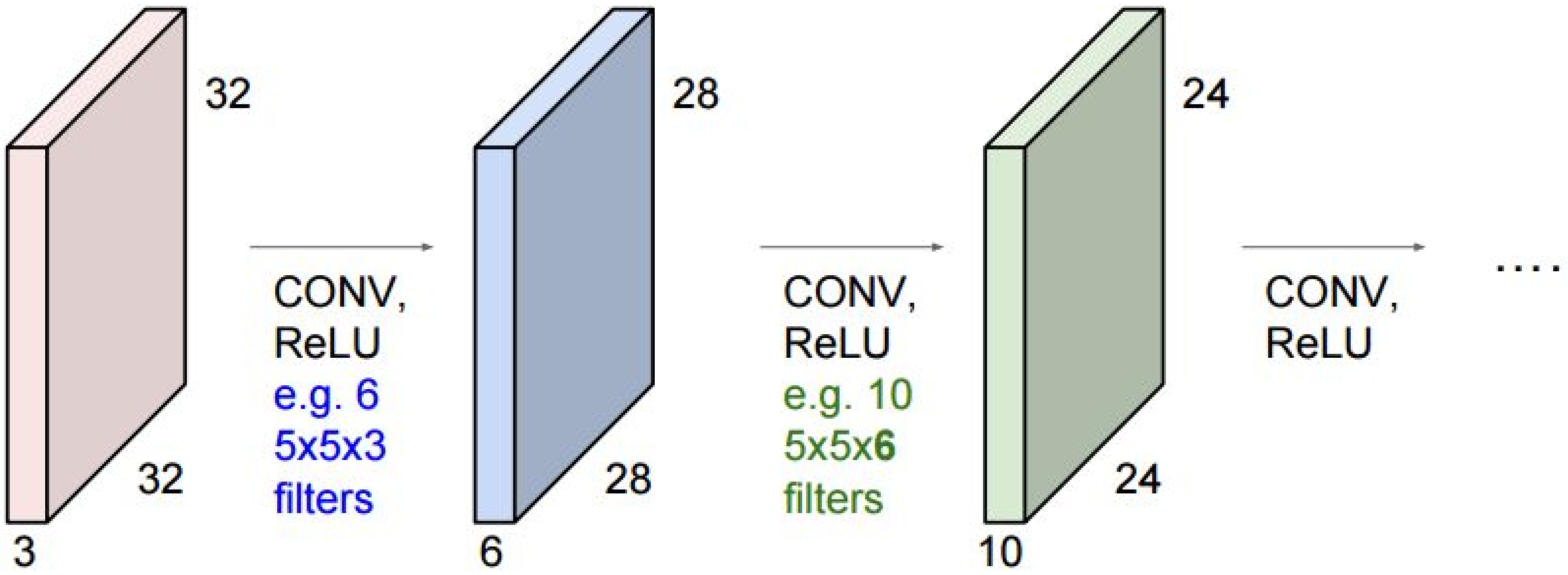
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



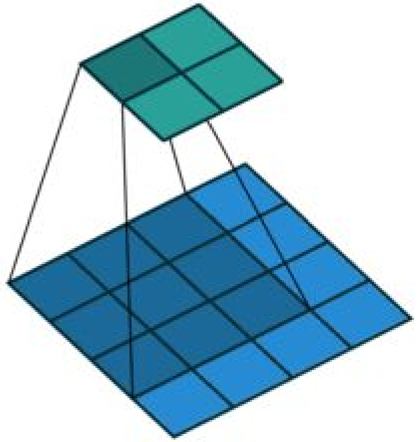
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

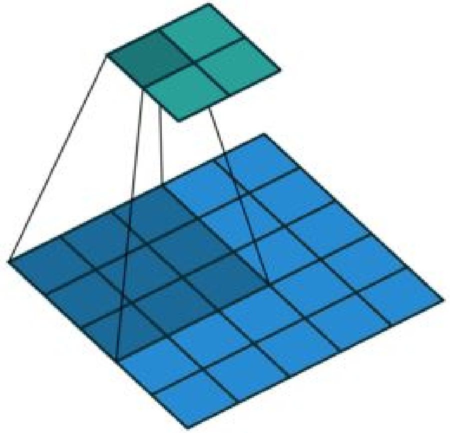


Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions

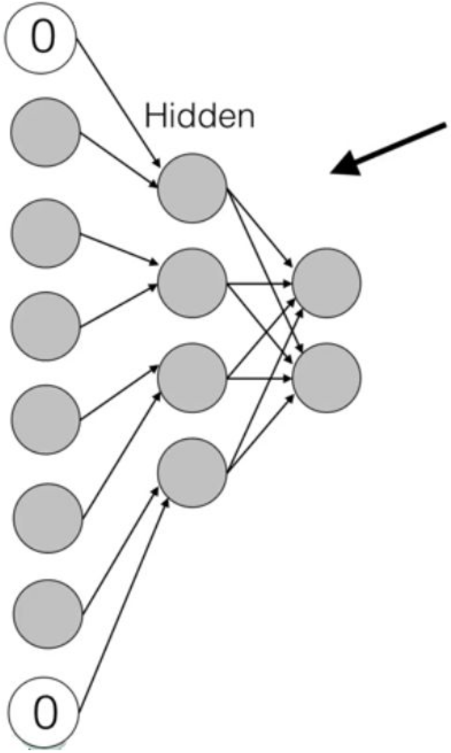


Stride 2-D

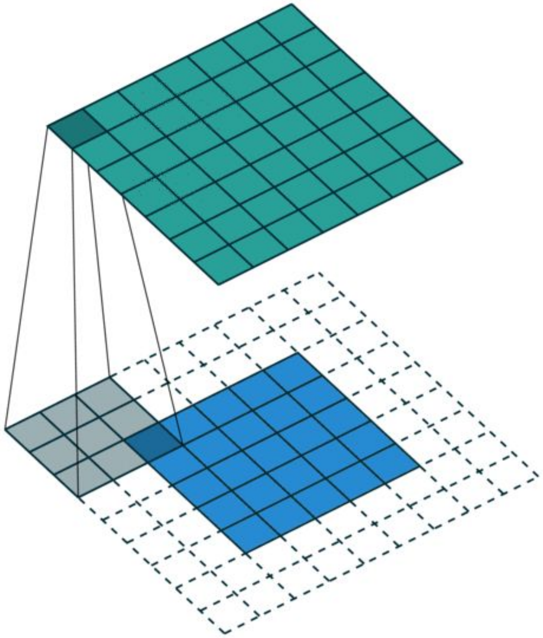




Padding

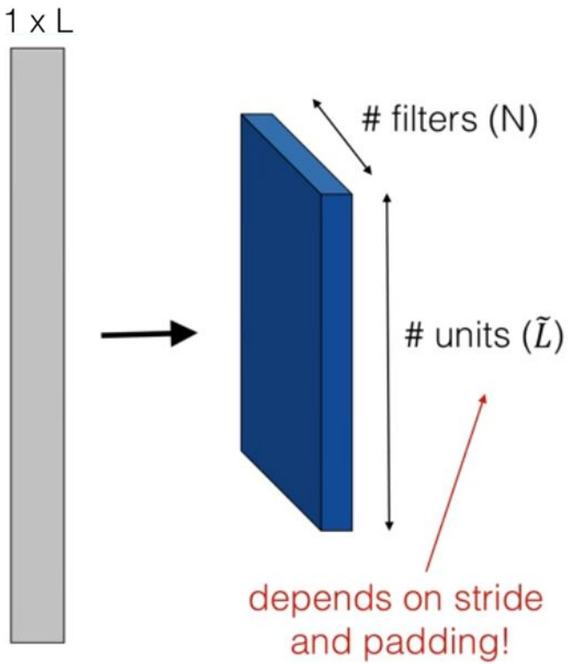


1-d convolution with
- filters: 1
- filter size: 2
- stride: 2
- padding: 1

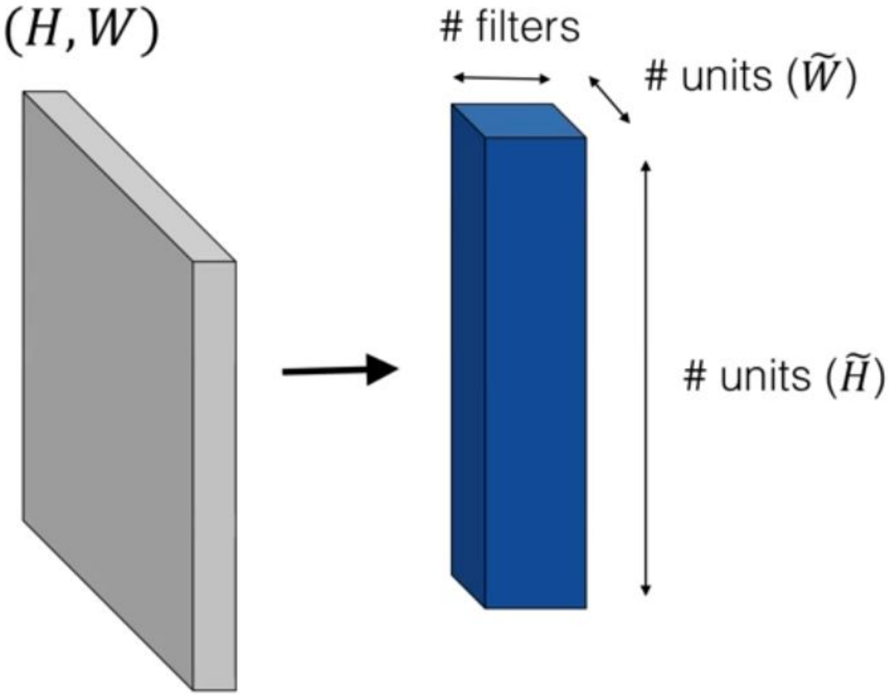


Features / Filters

1-D

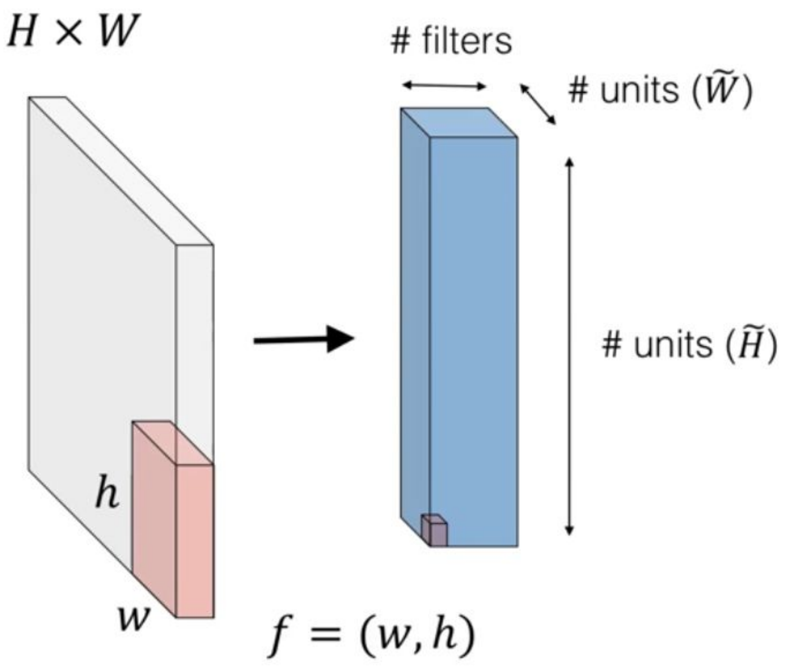


2-D



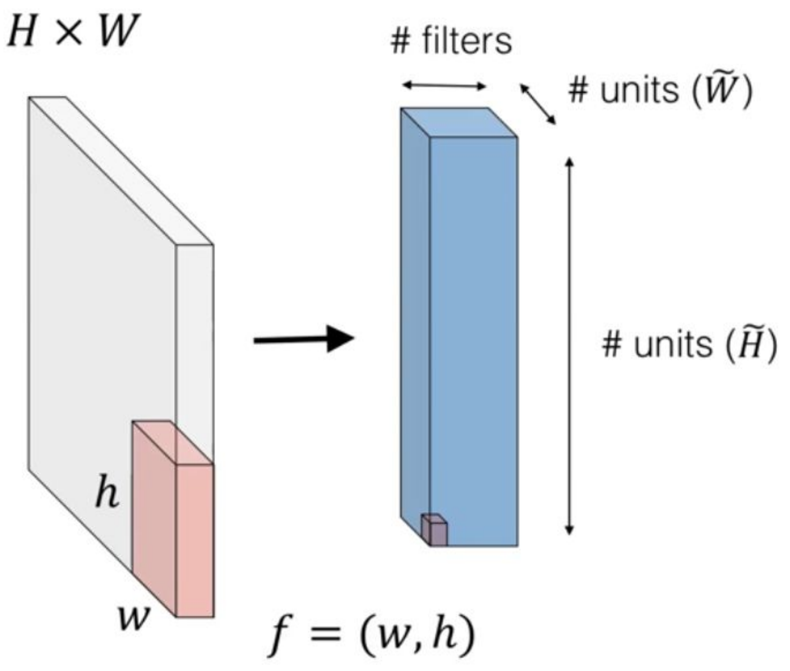
Features / Filters

2-D

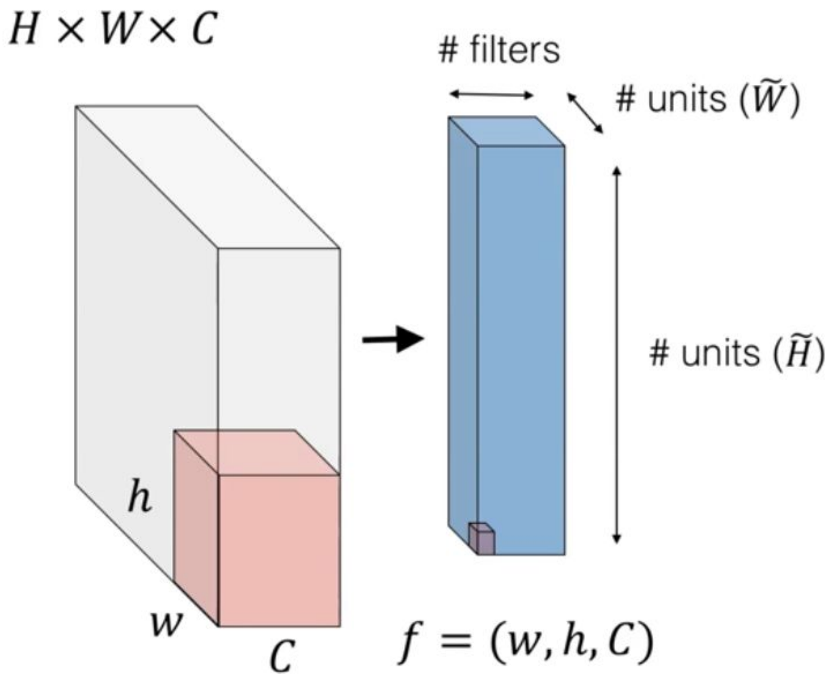


Features / Filters

2-D

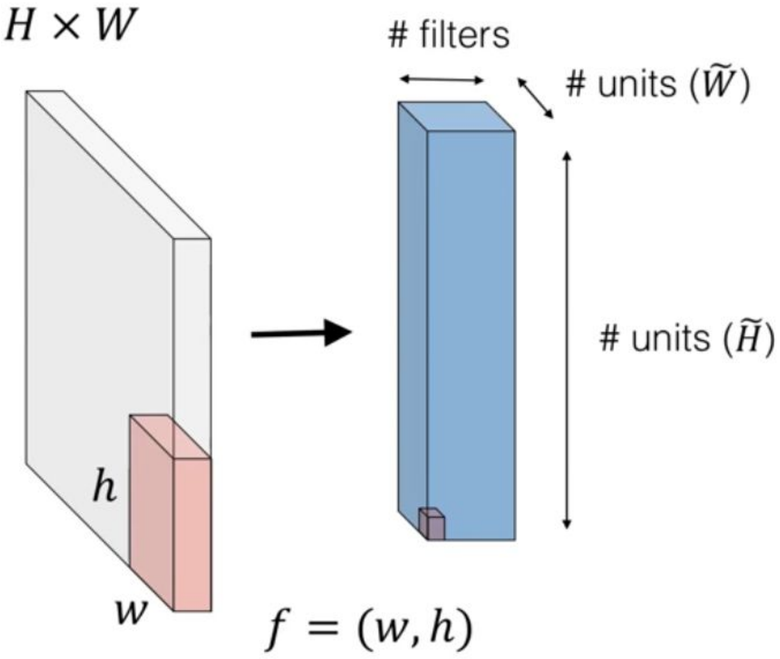


3-D

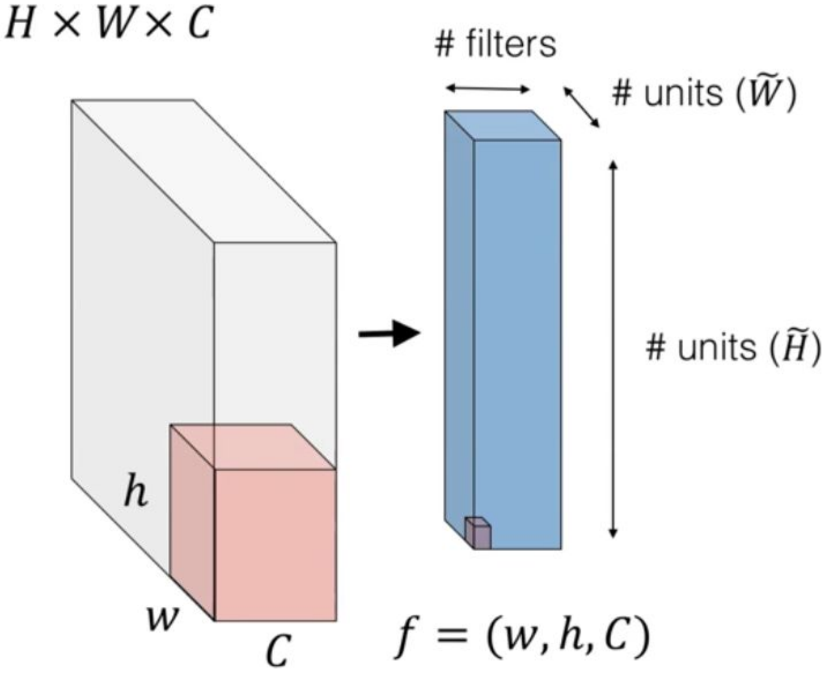


Features / Filters

2-D

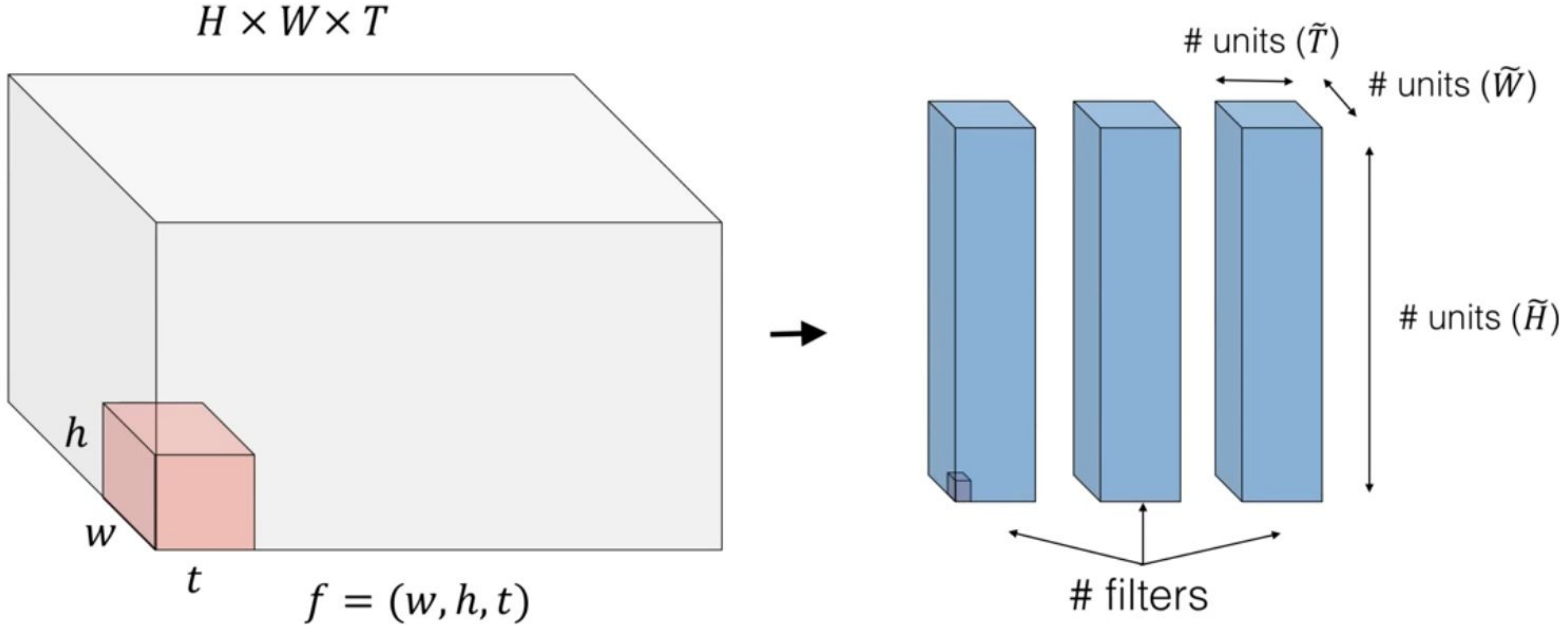


~~3-D~~ 2-D Multichannel

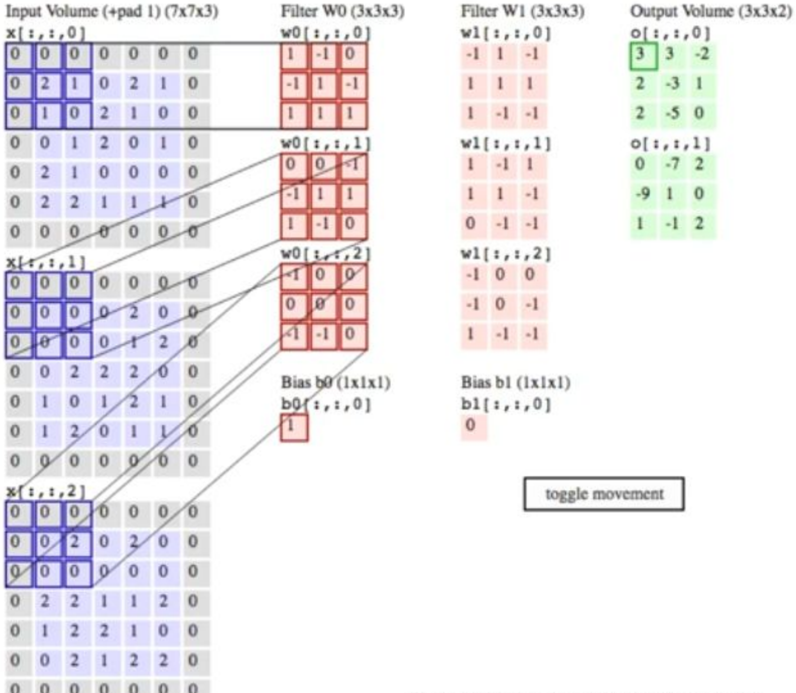


Features / Filters

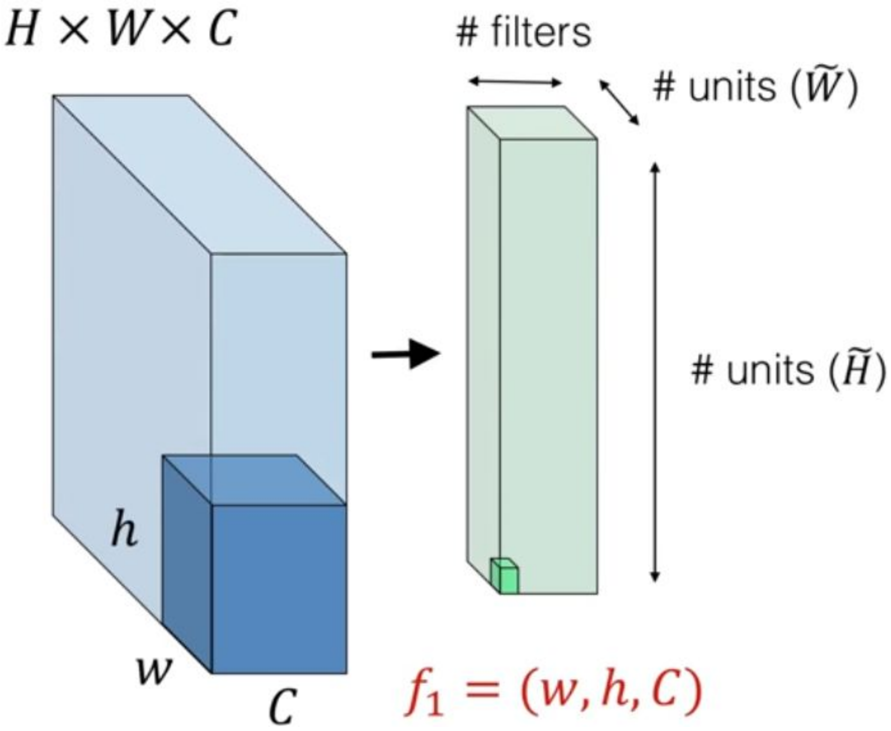
3-D



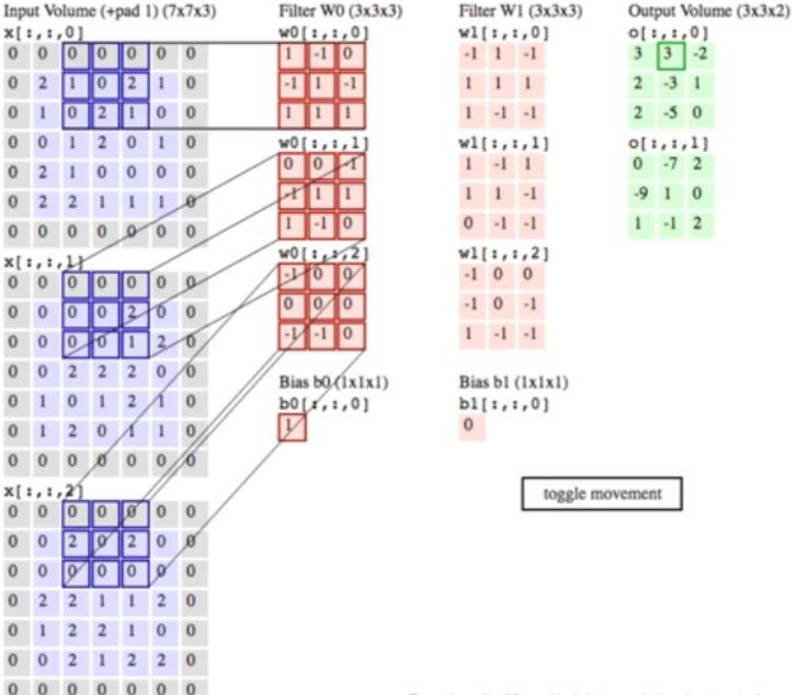
2-D Convolution



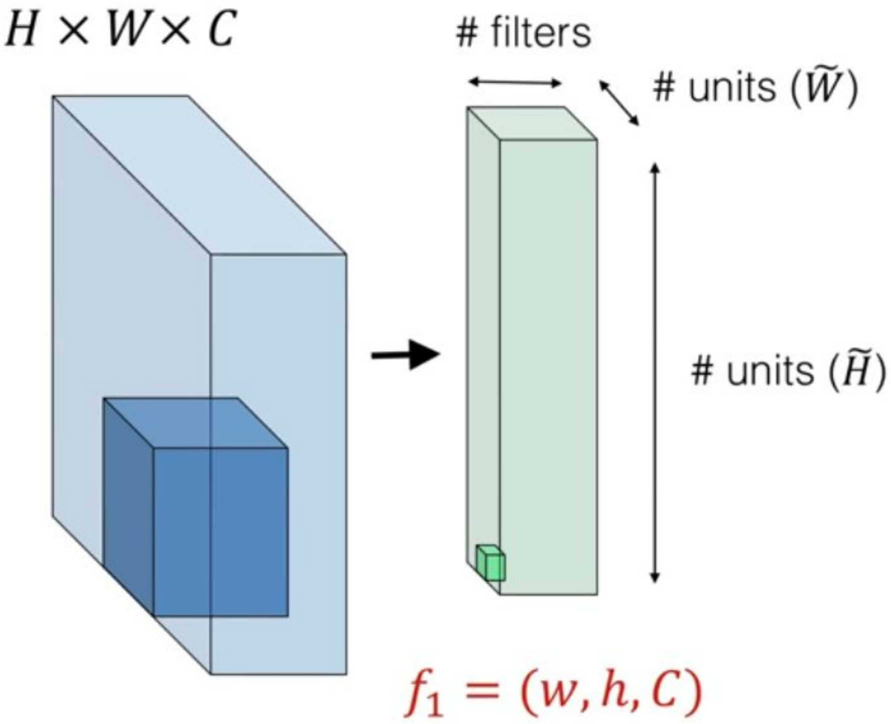
From: http://cs231n.github.io/convolutional-networks/



2-D Convolution

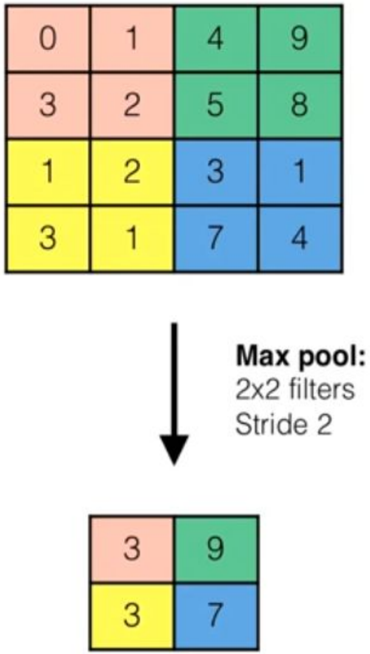


From: http://cs231n.github.io/convolutional-networks/

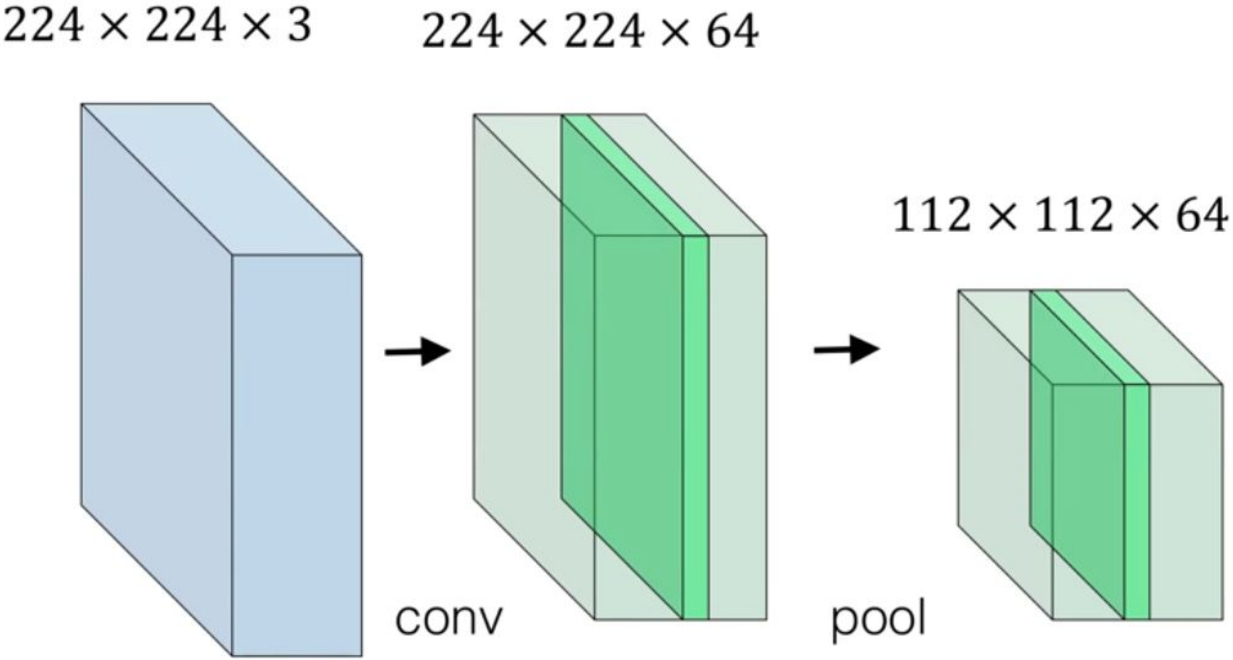


Pooling

1-D



2-D



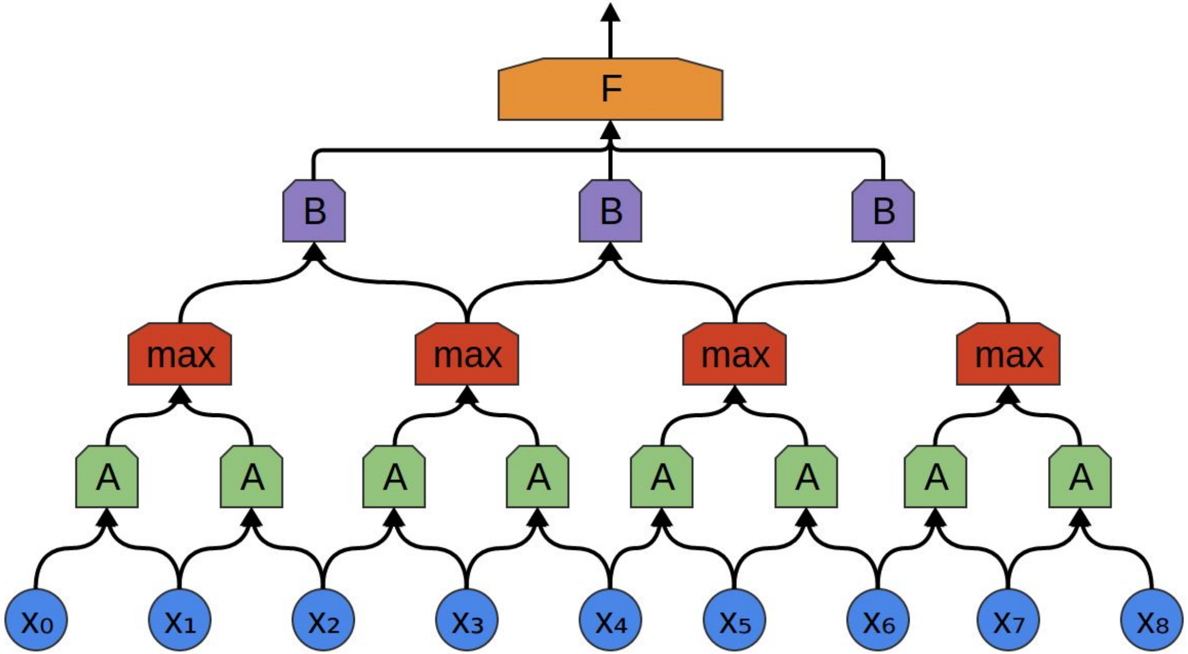
Pooling

Geoff Hinton on Pooling----

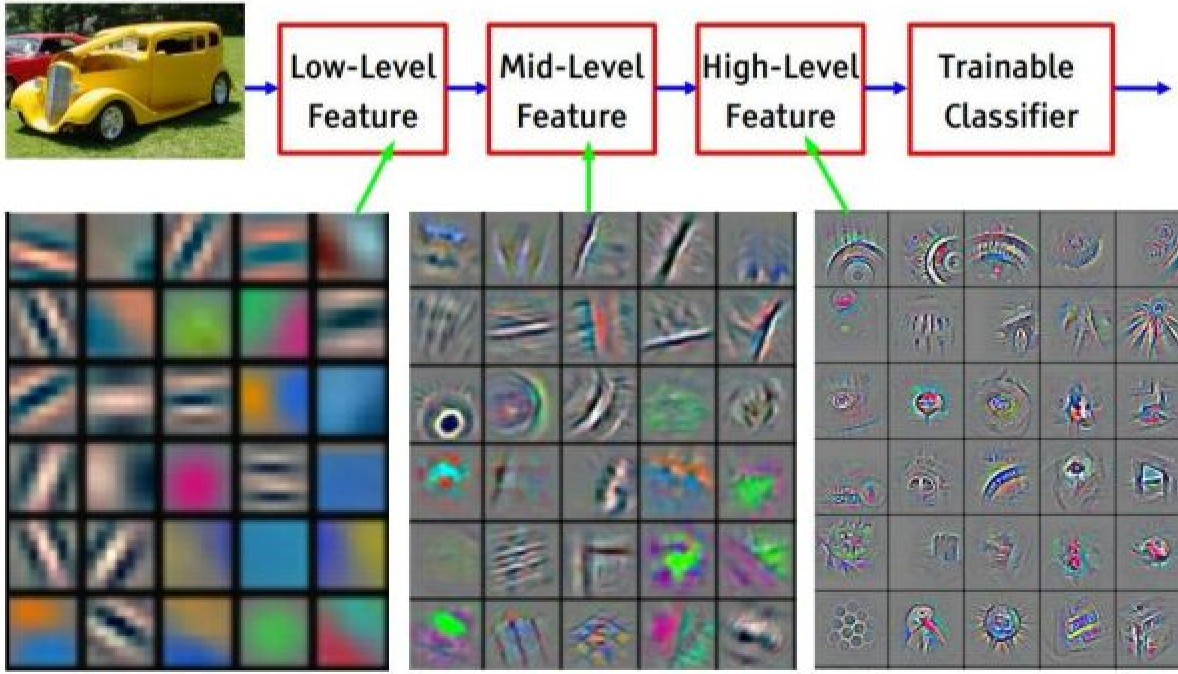
The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster.

If the pools do not overlap, pooling loses valuable information about where things are. We need this information to detect precise relationships between the parts of an object.

Convolutional Neural Network



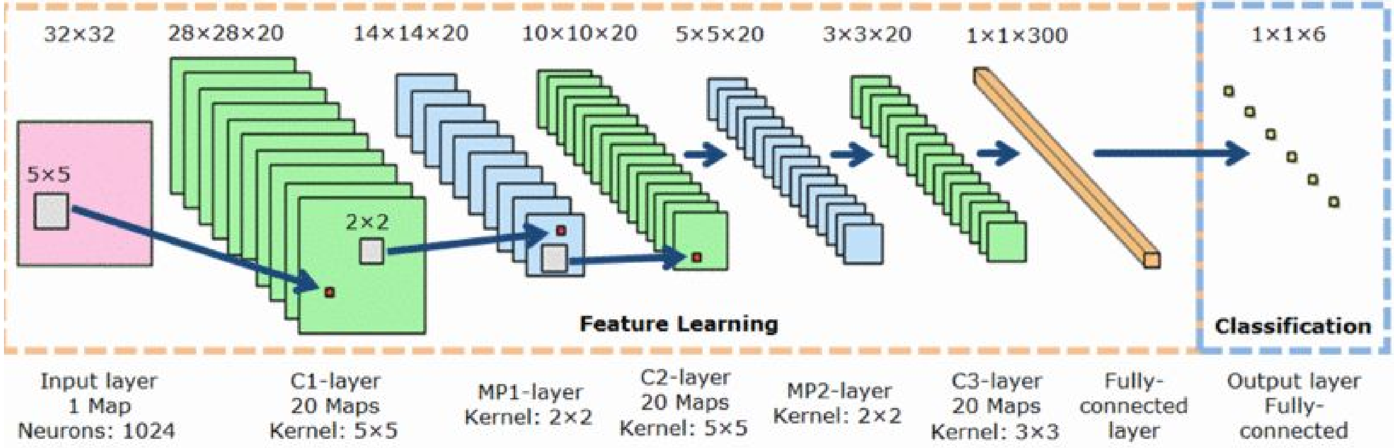
Convolutional Neural Network



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Demo Embedding

LeNet

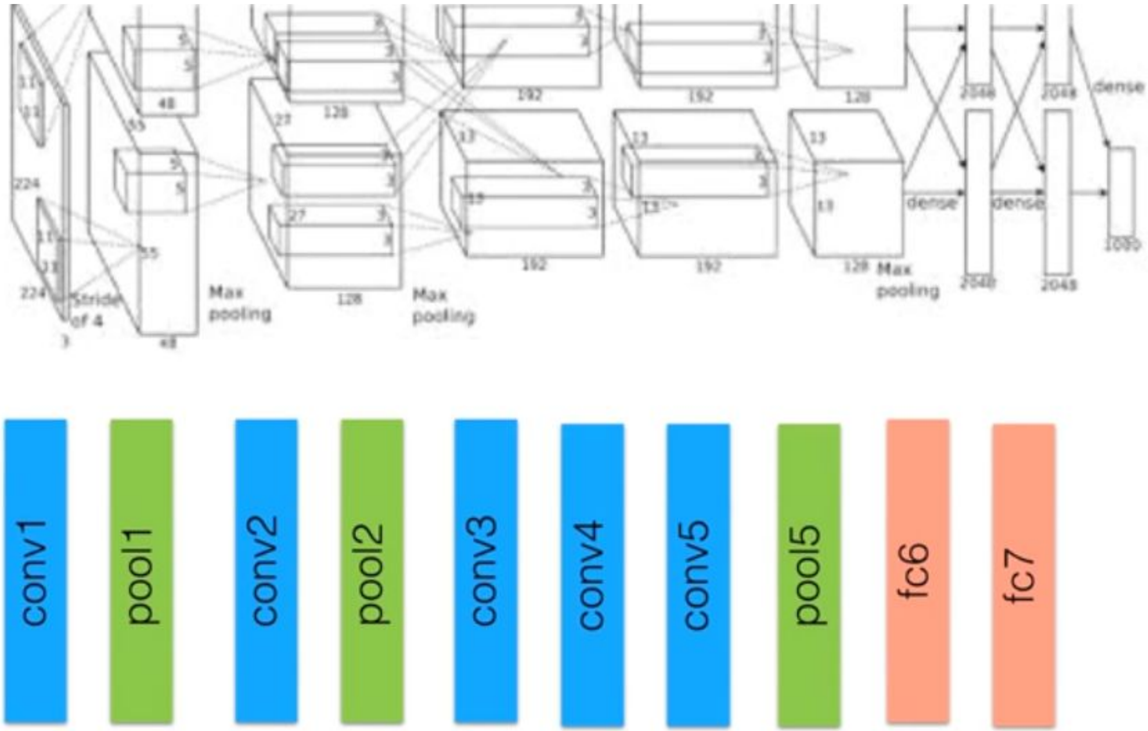


*Original LeNet-5 has two FCL at the end, and filter sizes are slightly different

AlexNet (2012 Winner)

AlexNet architecture:

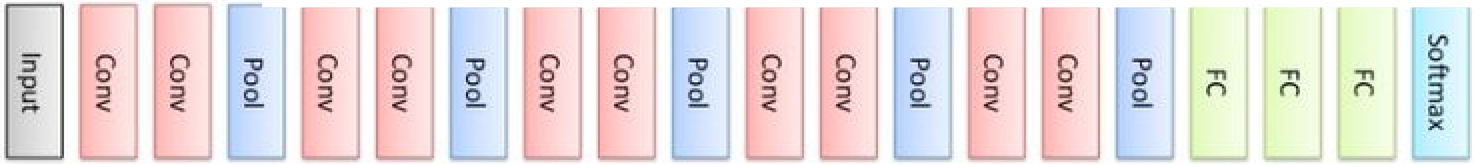
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad
[6x6x256] MAX POOL3: 3x3 filters at stride 2 [4096]
FC6: 4096 neurons [4096] FC7: 4096 neurons [1000]
FC8: 1000 neurons (class scores)



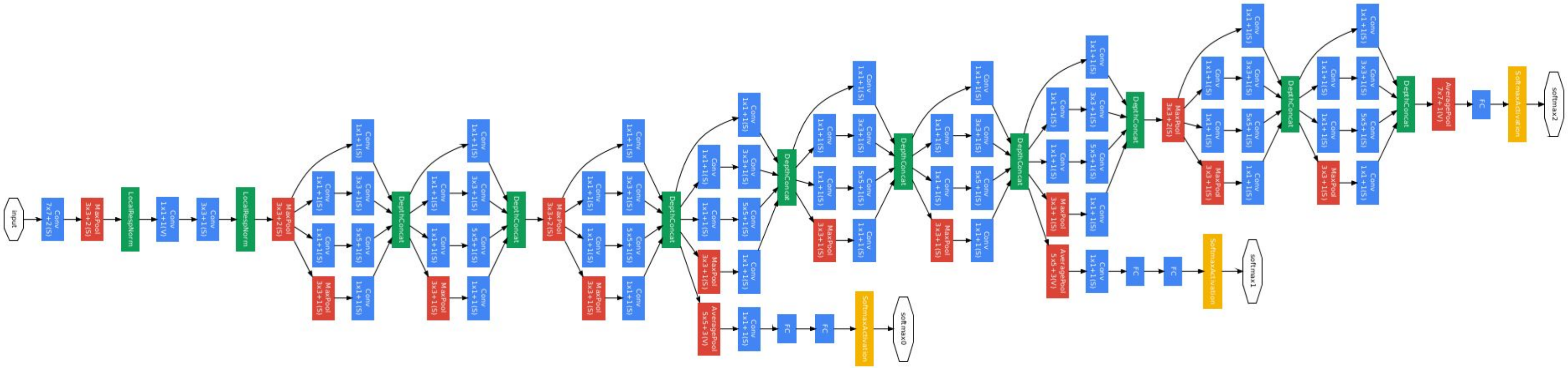
VGG Net

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

ConvNet Configuration		
B	C	D
13 weight layers	16 weight layers	16 weight layers
put (224 × 224 RGB image)		
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
maxpool		
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**
maxpool		
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**
maxpool		
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**
maxpool		
FC-4096		
FC-4096		
FC-1000		
soft-max		

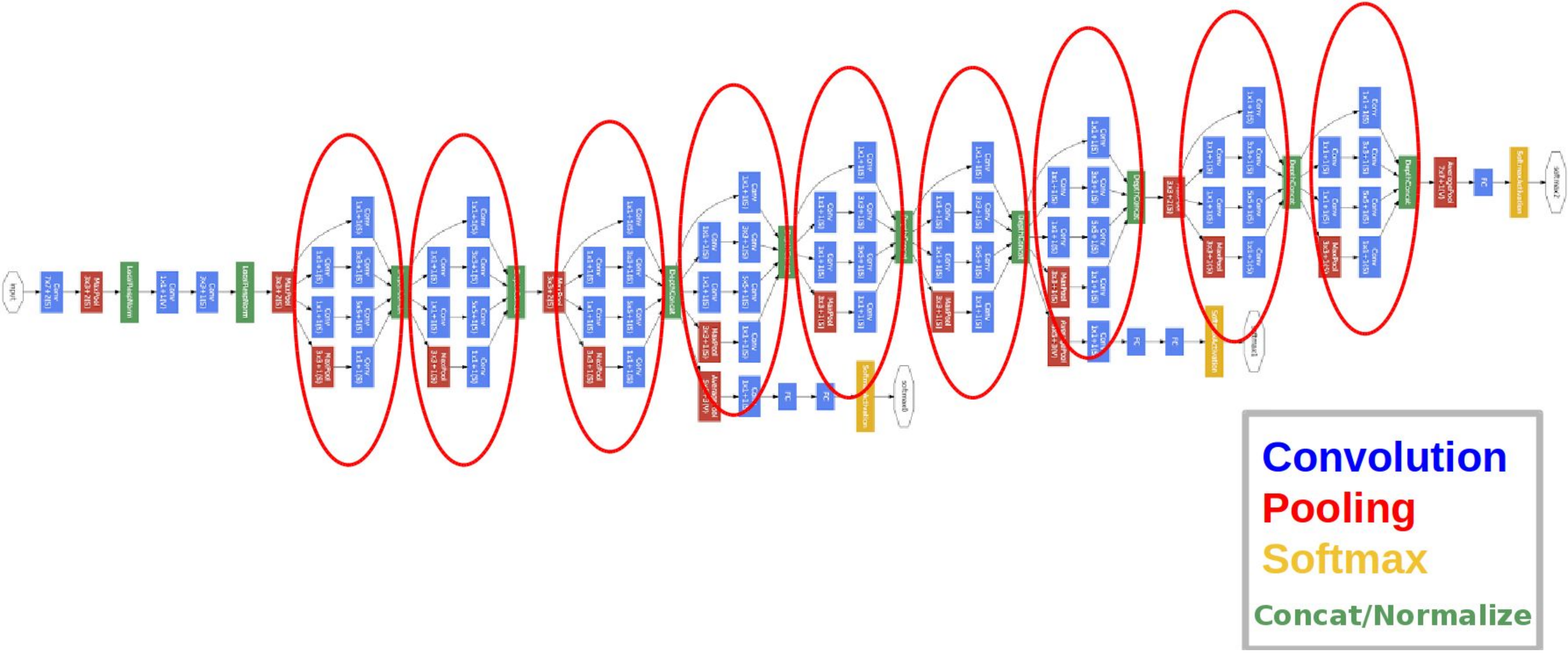


GoogLeNet (2014 winner)

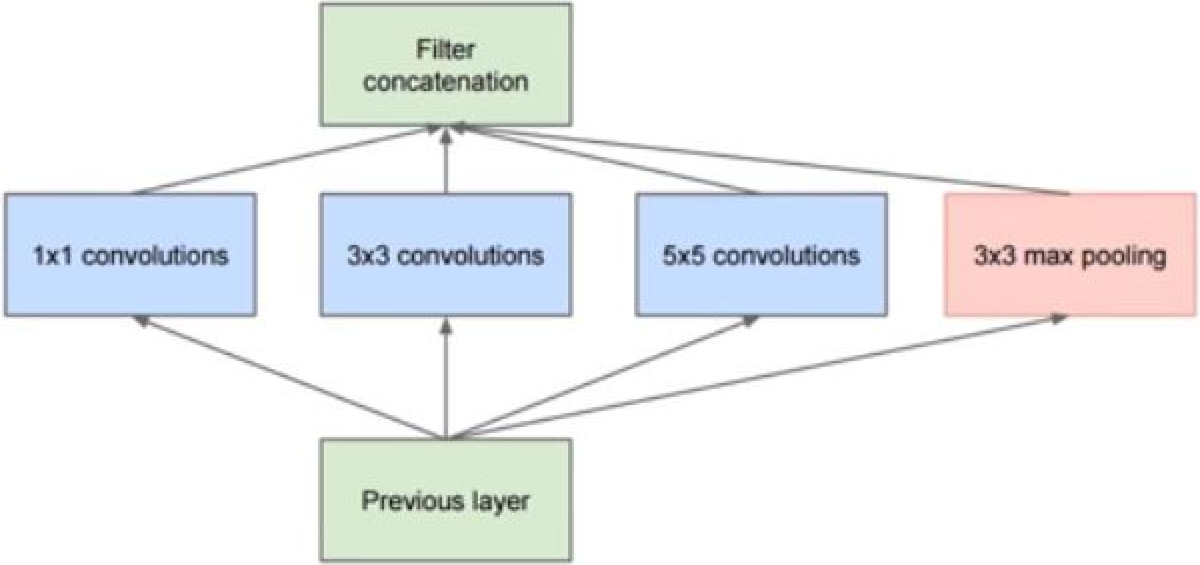


Peasant’s network vs Google’s

GoogLeNet (2014 winner)

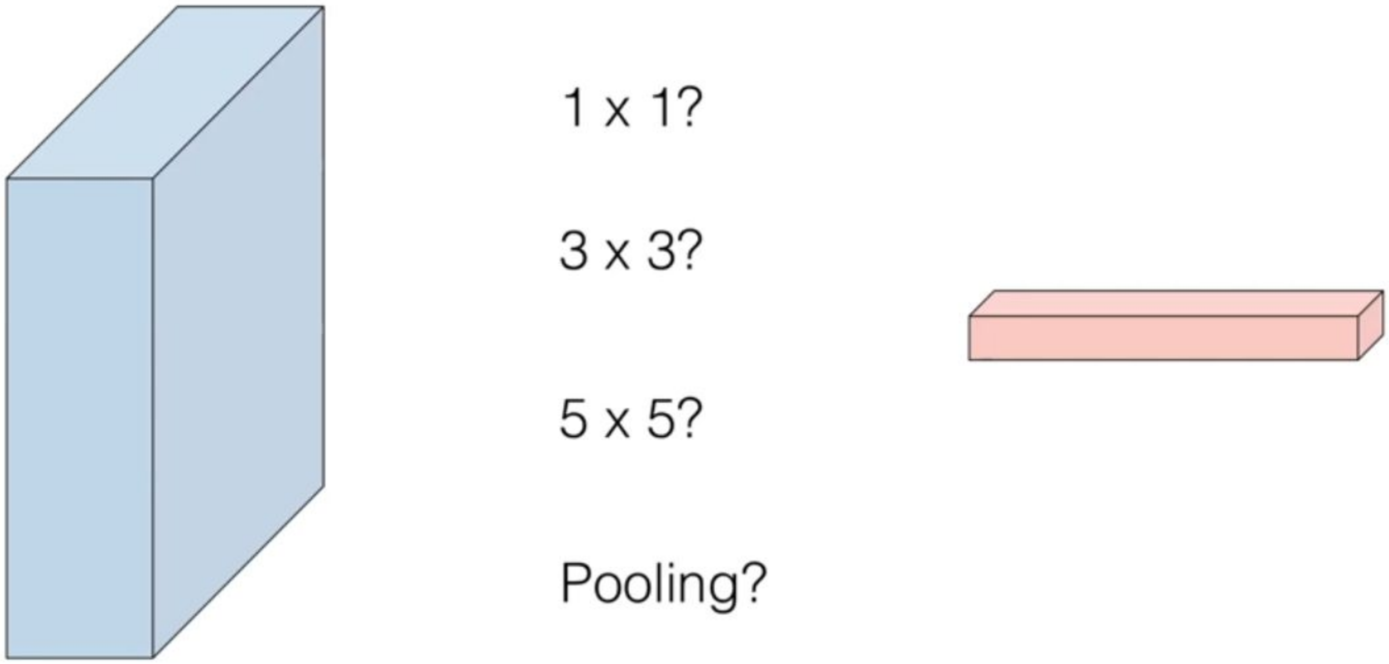


Inception

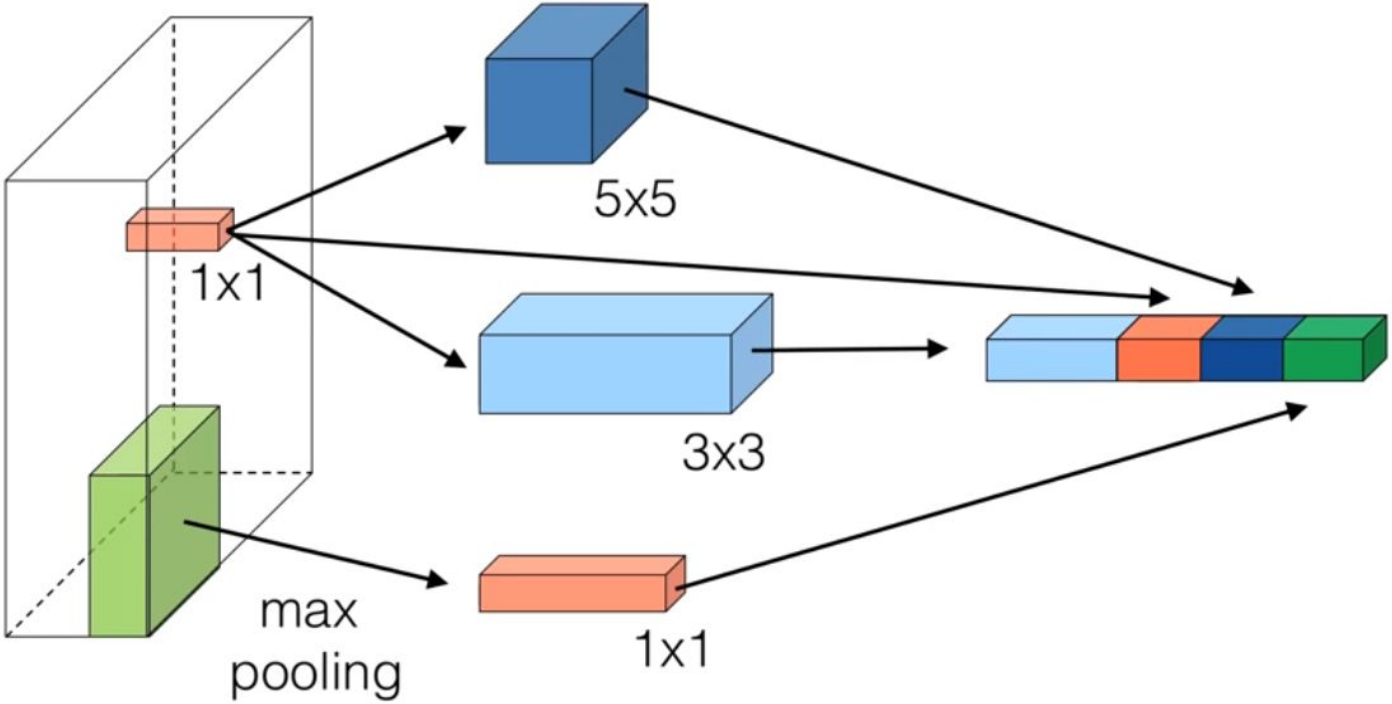


Filter concatenation
1x1 convolutions
3x3 convolutions
5x5 convolutions
3x3 max pooling
Previous layer

Inception



Inception



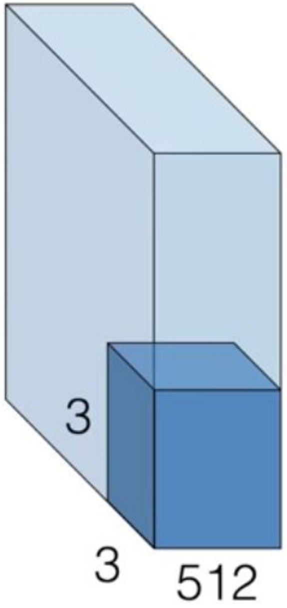
Inception

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M

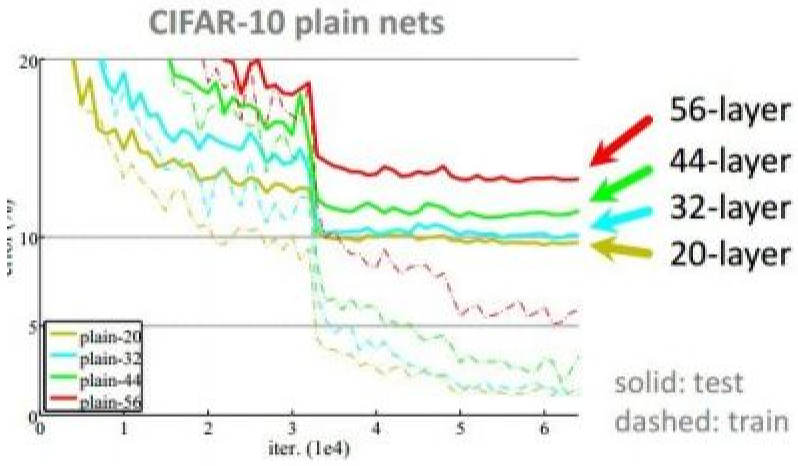
If we used (3 x 3, 512) convolution:

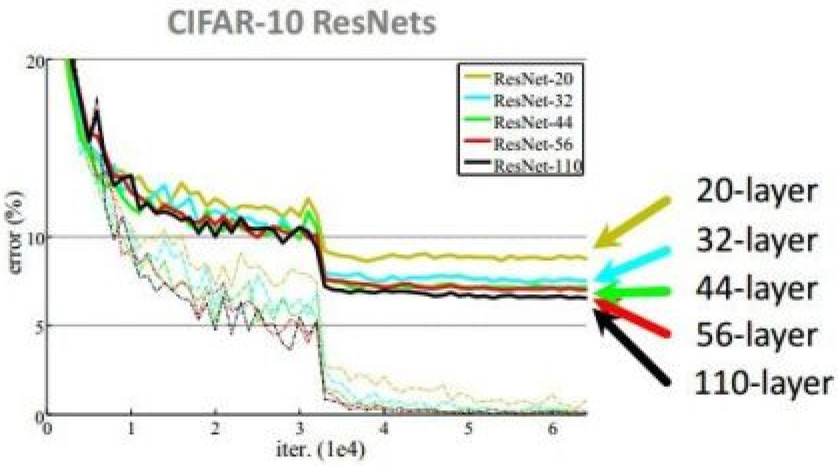
(3 x 3 x 512 x 512) parameters = 2.359 million parameters

Inception module: 437K parameters

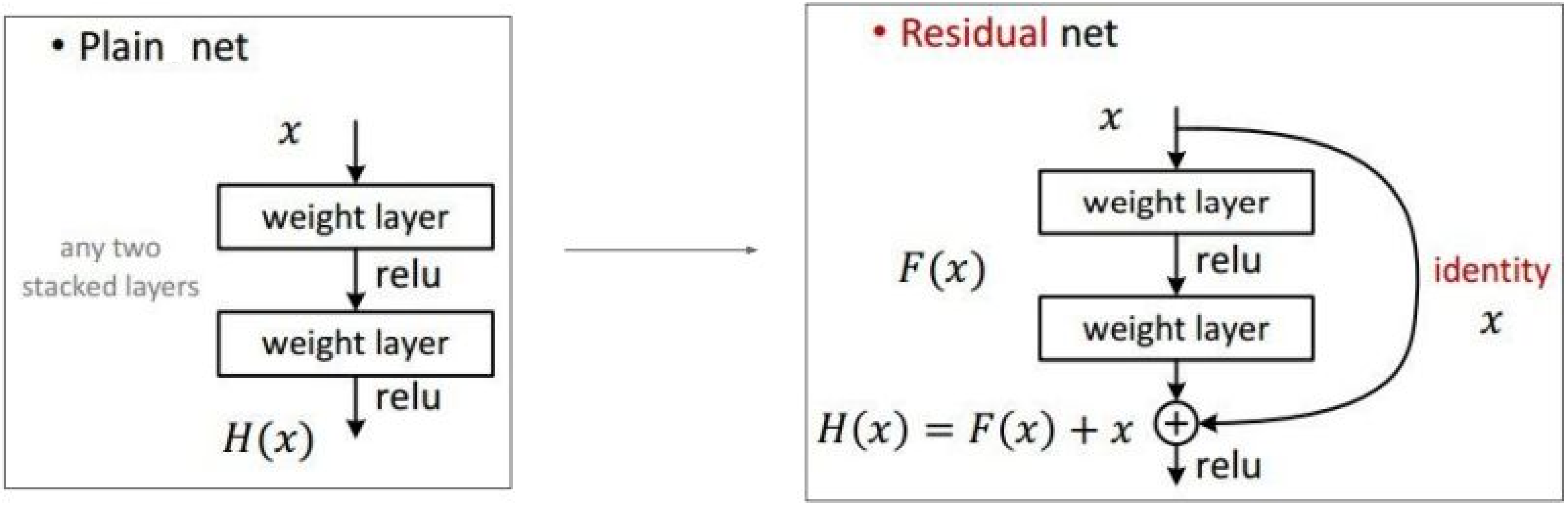


Going Deeper





Resnet (2015 winner)

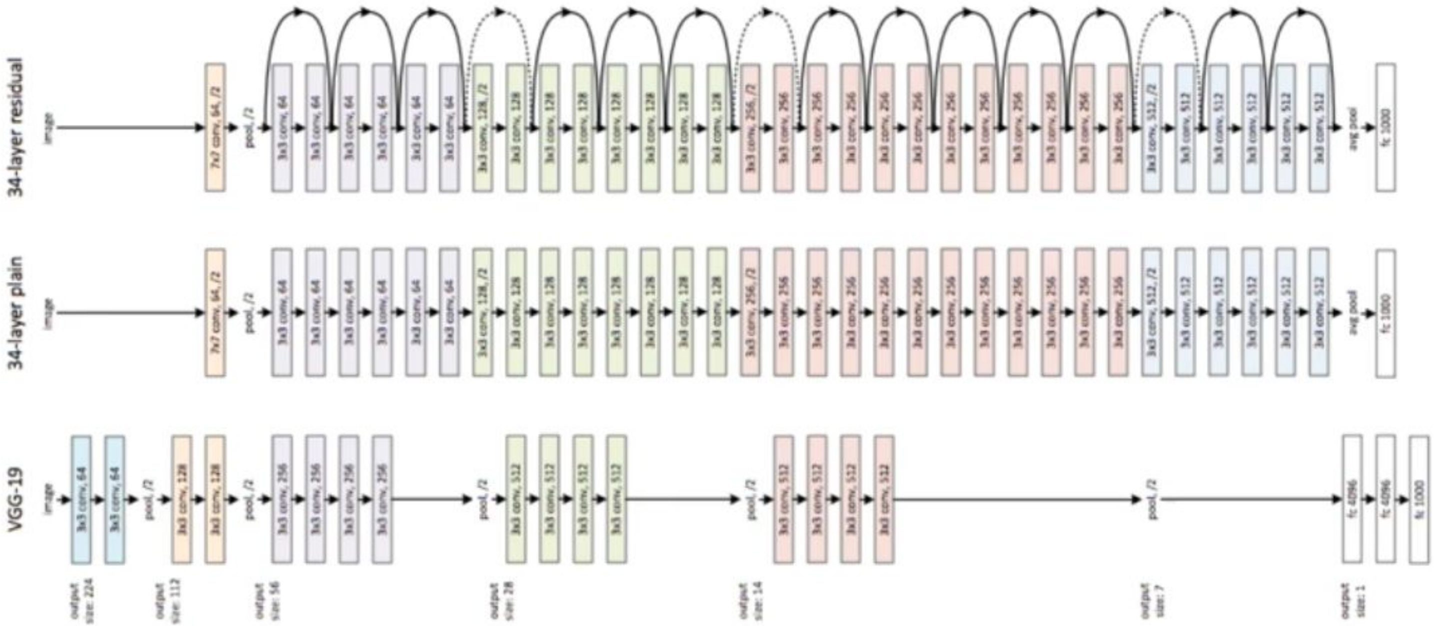


Resnet - comparison with other nets

34-residual

34-plain

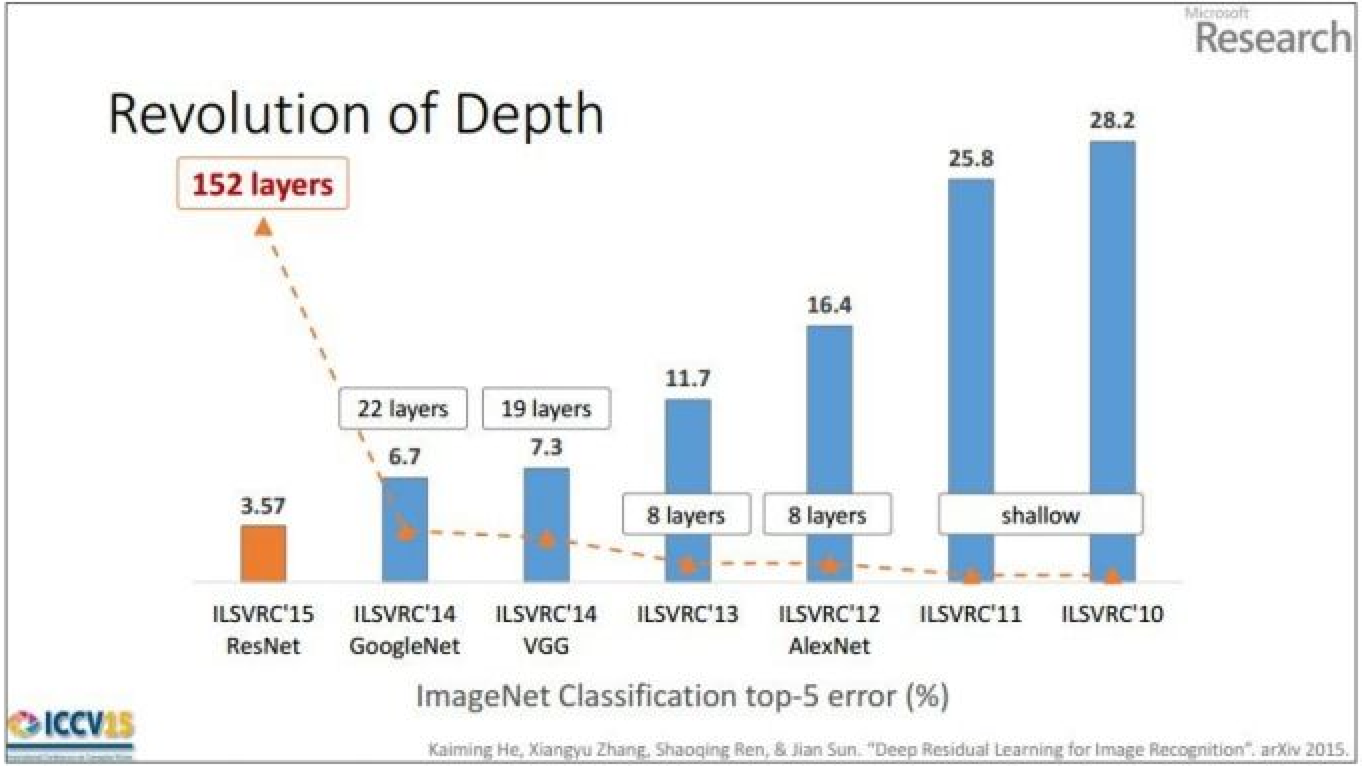
VGG



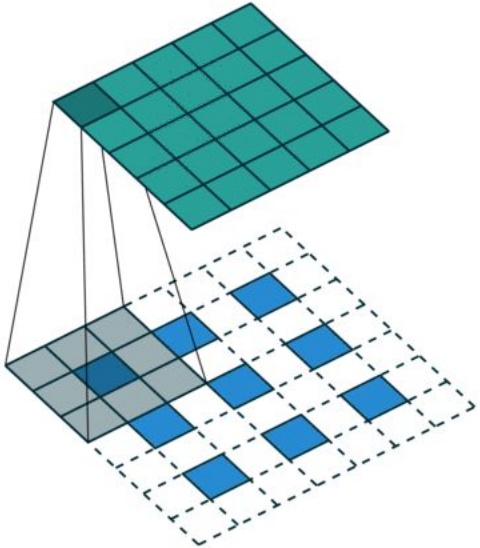
Resnet - Number of layer comparison

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

Depth - The Stigma



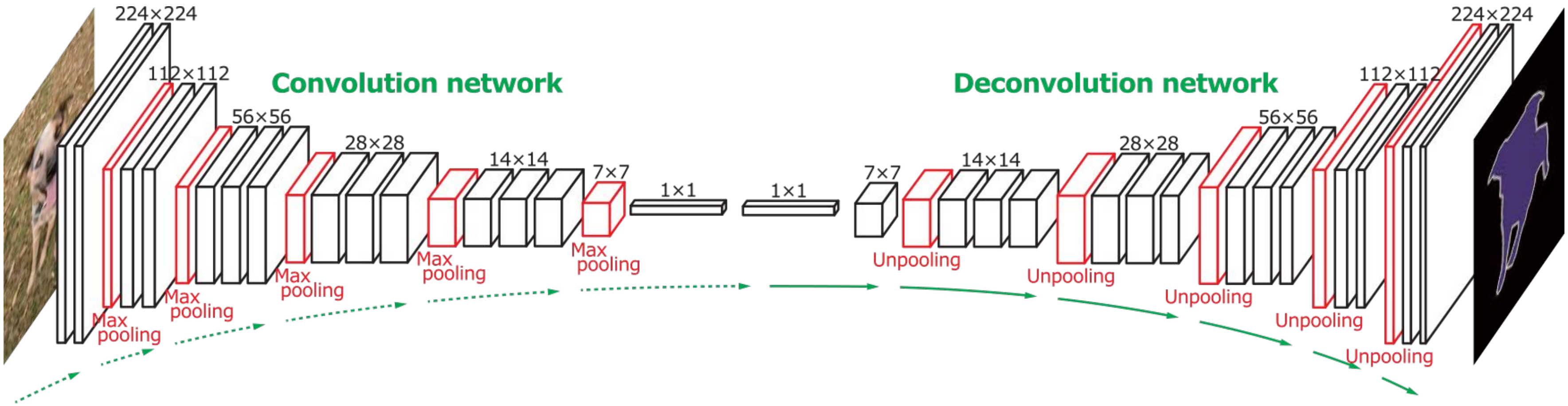
Deconvolution / Transpose Convolution / Fractional Convolution



Demo Deconv


conv1 p1 n1 conv2 p2 n2 conv3 conv4 conv5 p5 fc6 fc7 fc8 prob
fwd conv5_151 | Back: deconv (from conv5_151, disp raw) | Boost: 0/1

Semantic Segmentation



Artistic Style Transfer

- Feed the artistic image through the VGG net and compute and save the Gram matrix G.

- Feed the photograph through the VGG net and save the feature maps F.

- Generate a white noise image. Through backpropagation, iteratively update this image until it has a feature map and a Gram matrix that are close to F and G, respectively.

Fooling CNNs

Adversarial Examples and Rubbish Classes

Answers -

1. Due to high dimensional dot products
2. Occurs in both linear (ReLu) & Non-linear models
3. Direction of perturbation matters not specific point
4. Also occurs in Shallow networks not just DNN
5. Regularisation doesn't prevent fooling examples
6. Adversarial training is good regularization
7. Extremely low probability (not observed in test)

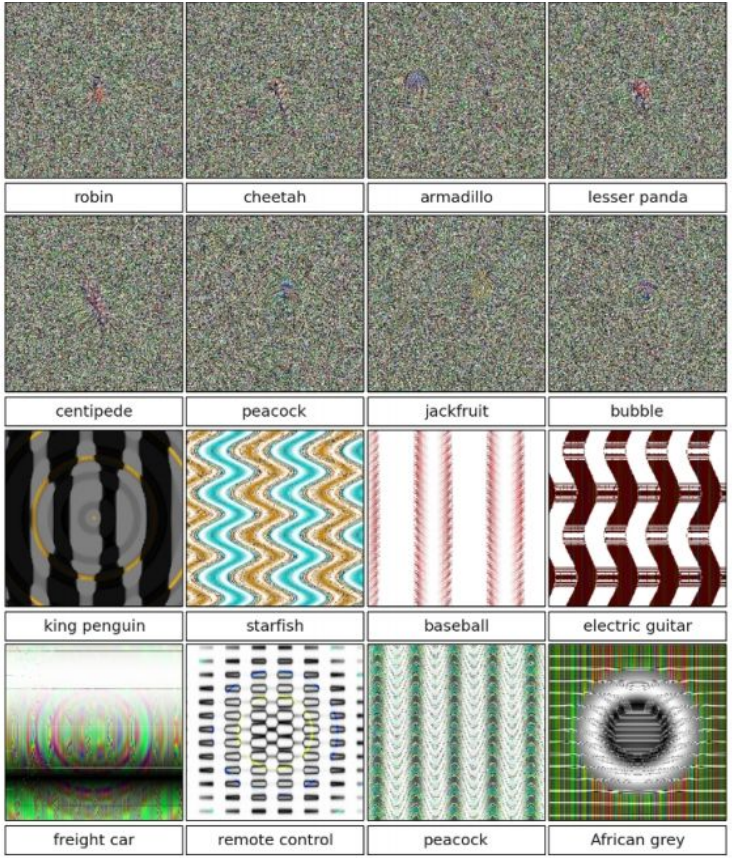


Figure 1. Evolved images that are unrecognizable to humans, but that state-of-the-art DNNs trained on ImageNet believe with $\geq$ 99.6% certainty to be a familiar object. This result highlights differences between how DNNs and humans recognize objects. Images are either directly (*top*) or indirectly (*bottom*) encoded.

Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like

  [(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX

  where N is usually up to ~5, M is large, 0 <= K <= 2.
- but recent advances such as ResNet/GoogLeNet challenge this paradigm

Credits

1. CS231n Convnets
2. Chris Colah's awesome blog
3. Chris Burger - Style transfer images
4. Convolutional Neural Networks - Nervana Systems
5. DeepVis - Jason Yosiniki
6. Fooling CNNs - Anh Nguyen
7. More demos - Yann LeCun