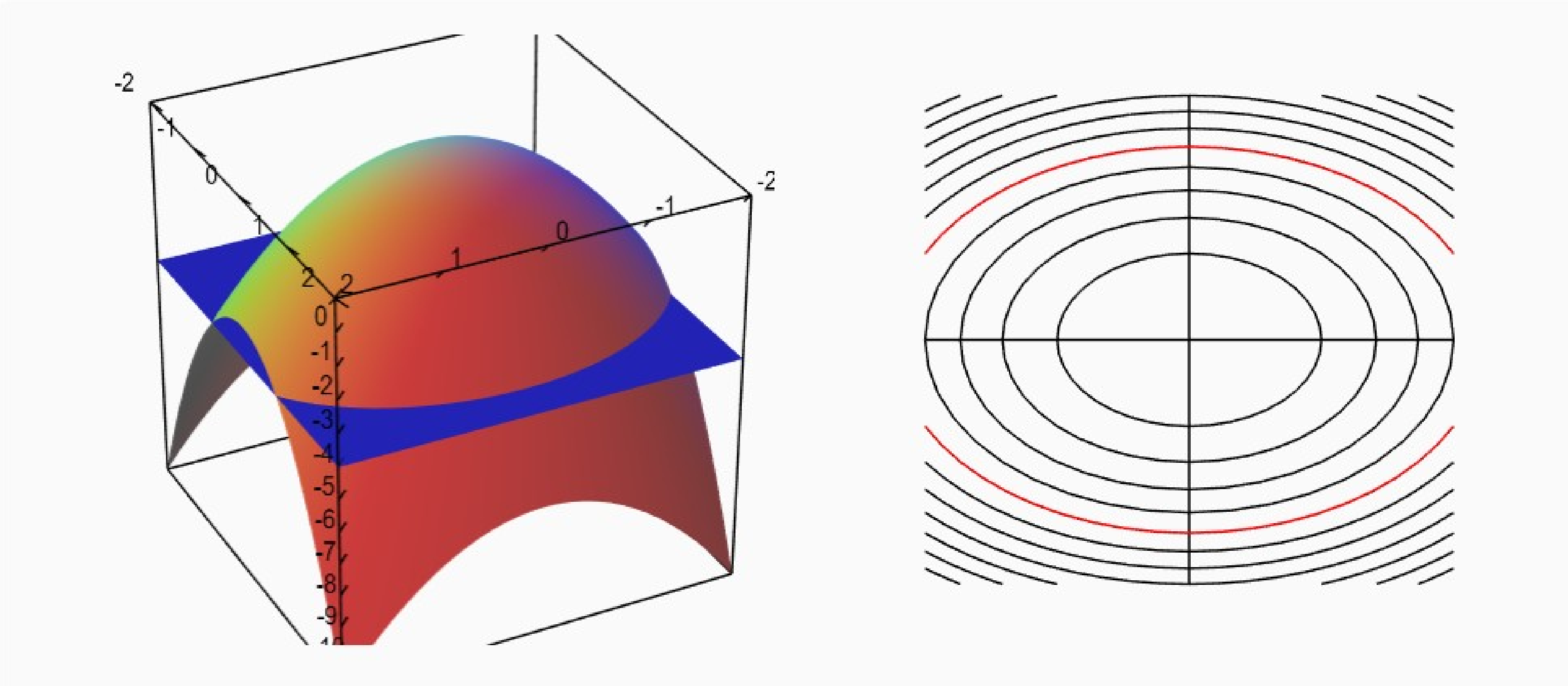
Computer Vision and Deep Learning

Lecture 6

Today’s agenda

- Optimization algorithms
- Convolutional neural networks:
  - History
  - Case studies
- How to read a research paper?

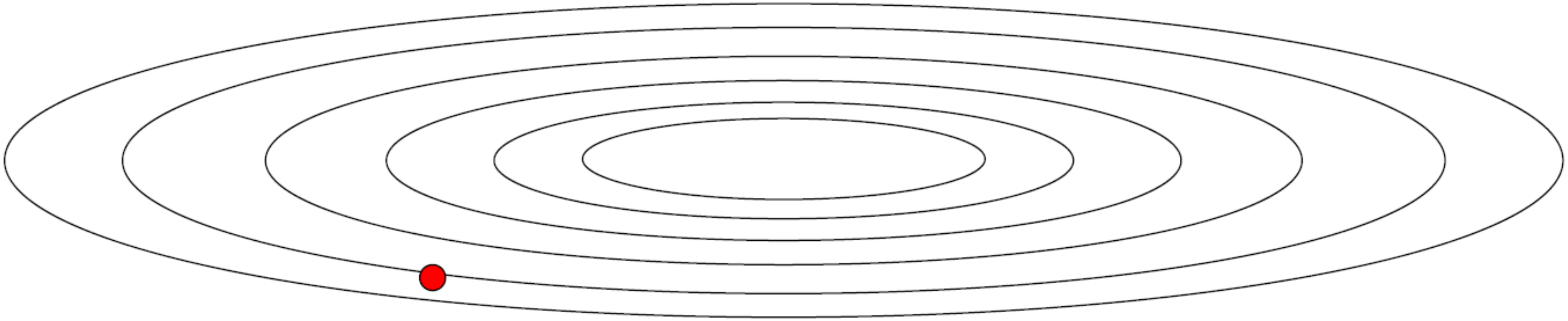
Level sets of a surface



https://mathinsight.org/level_sets

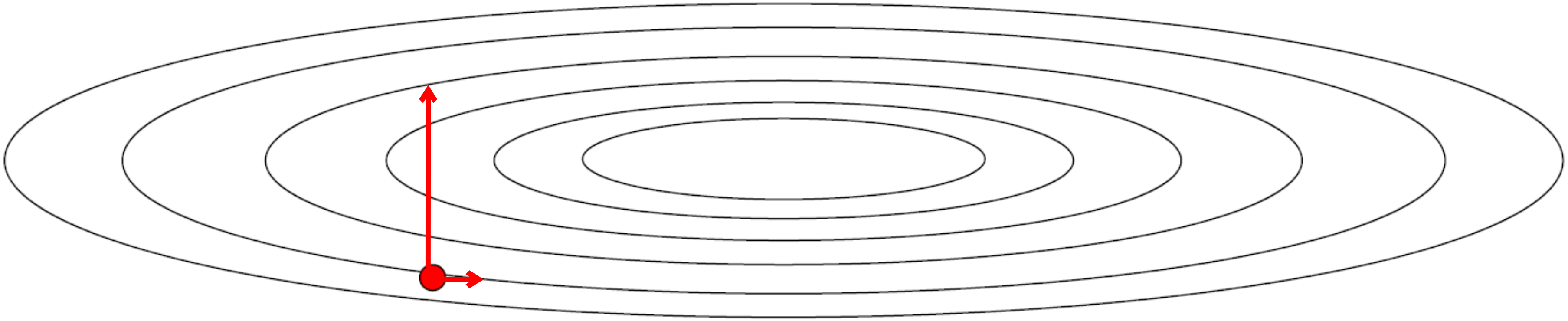
Problems with gradient descent

What if loss changes abruptly on one direction and slower in the other direction?



Problems with gradient descent

What if loss changes abruptly on one direction and slower in the other direction?

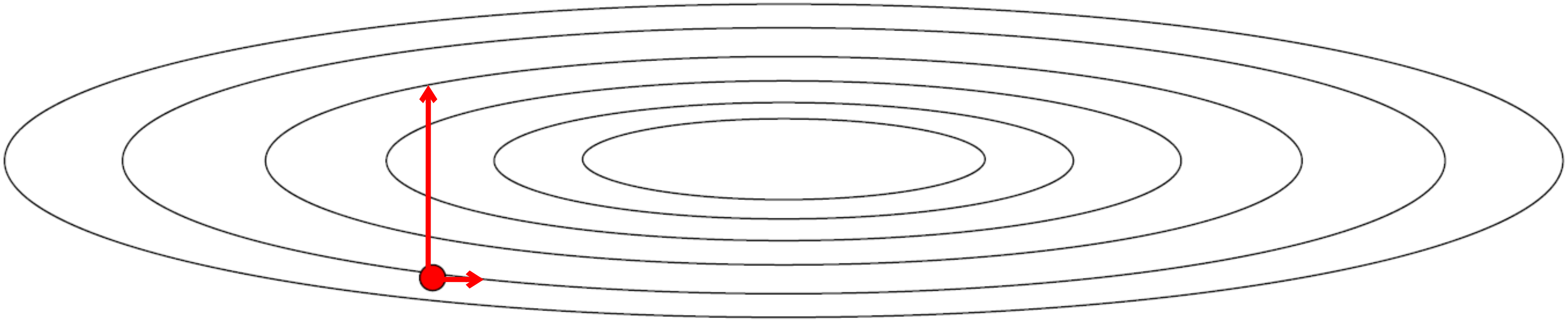


Problems with gradient descent

What if loss changes abruptly on one direction and slower in the other direction?

- Small gradient horizontally
- Large gradient vertically

Slow progress along the horizontal direction, jitter along the vertical direction (the steeper one)



Gradient descent with momentum

Compute an exponentially weighted average of the gradients and use this average to update the parameters of the network

Gradient descent with momentum

Update rule

Gradient descent update:

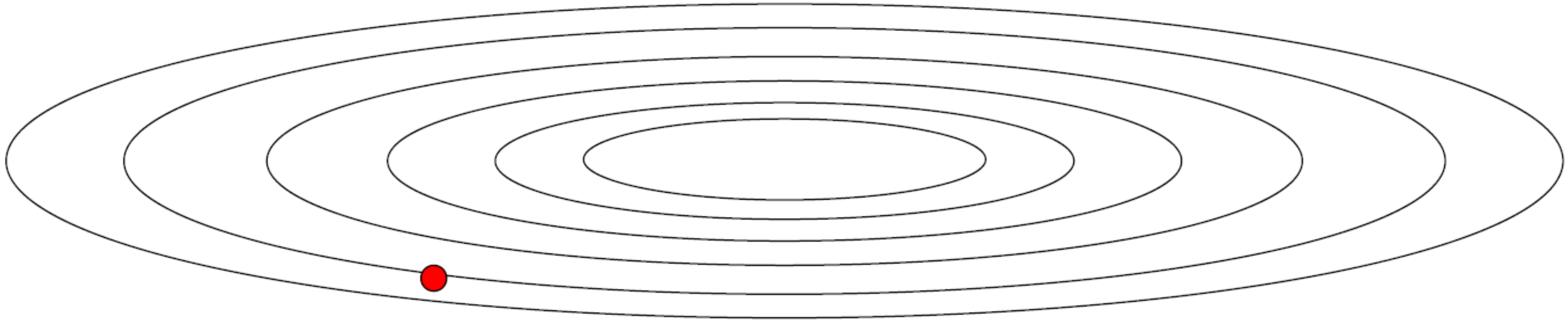
↓

Momentum update:

– hyper-parameter; common value 0.9

Gradient descent with momentum

Take more straightforward part (damp oscillations)



Slower learning vertically, faster learning horizontally

Gradient descent with momentum

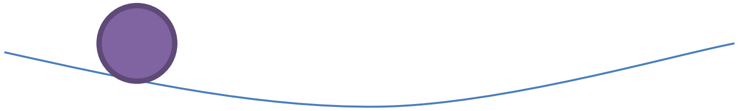
- Intuition
  - Ball rolling down the loss function with friction
  - Gradient: the force the ball is feeling (F = ma)

$$v_{dW} = \beta \cdot v_{dW} + (1-\beta) \cdot dW$$

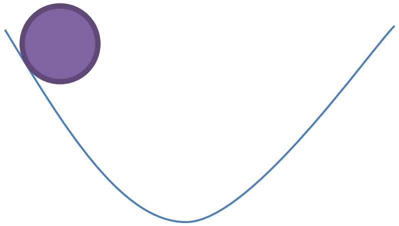
– friction
Hyper-parameter

velocity

acceleration



Shallow, consistent direction: build up the velocity across the dimension



Steep direction: attenuate velocity (quickly changing sign); oscillate to the “middle”

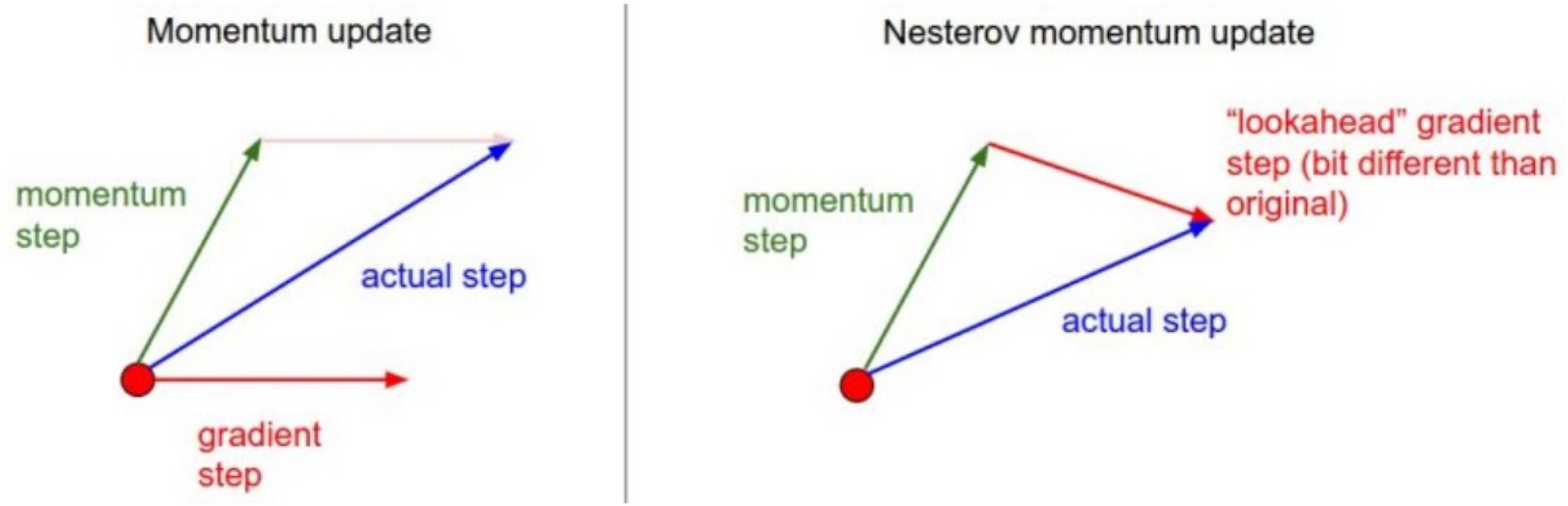
Gradient descent with momentum

Update rule - variant

– hyper-parameter; common value 0.9

Nesterov momentum

$$v_{dW} = \beta \cdot v_{dW} + dW$$



$$v_{t+1} = \mu v_t - \eta \nabla l(\theta)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

$$v_{t+1} = \mu v_t - \eta \nabla l(\theta + \mu v_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

Image source: https://cs231n.github.io/neural-networks-3/#sgd

https://dominikschmidt.xyz/nesterov-momentum/

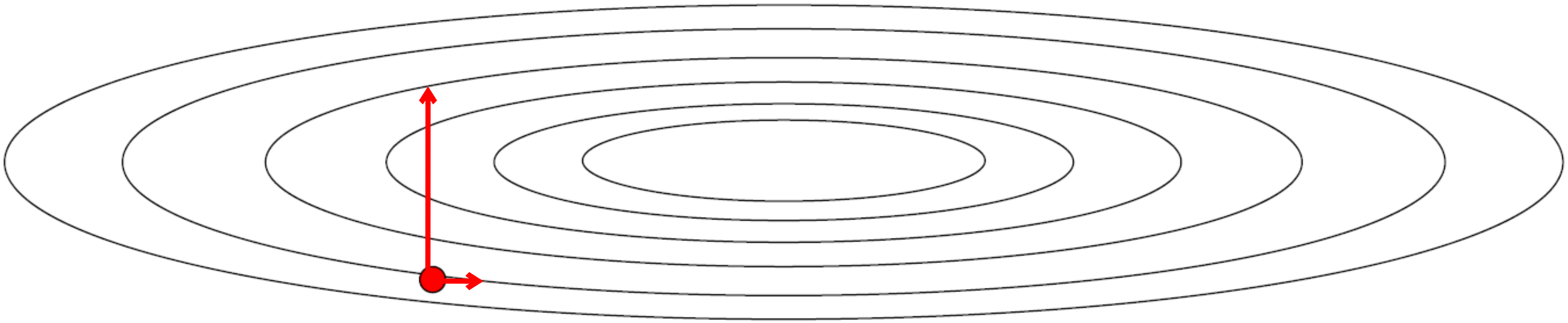
AdaGrad

cache – same size as W

Element wise scaling of the gradient based on the "historical" sum of squares in each dimension

Per parameter adaptive learning rate

AdaGrad



Equalizing effect

- Larger learning rate on “shallow” directions than on steeper directions

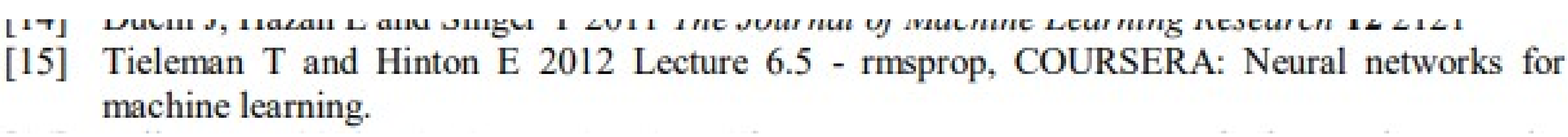
RMSProp

Adagrad

↓

RMSProp

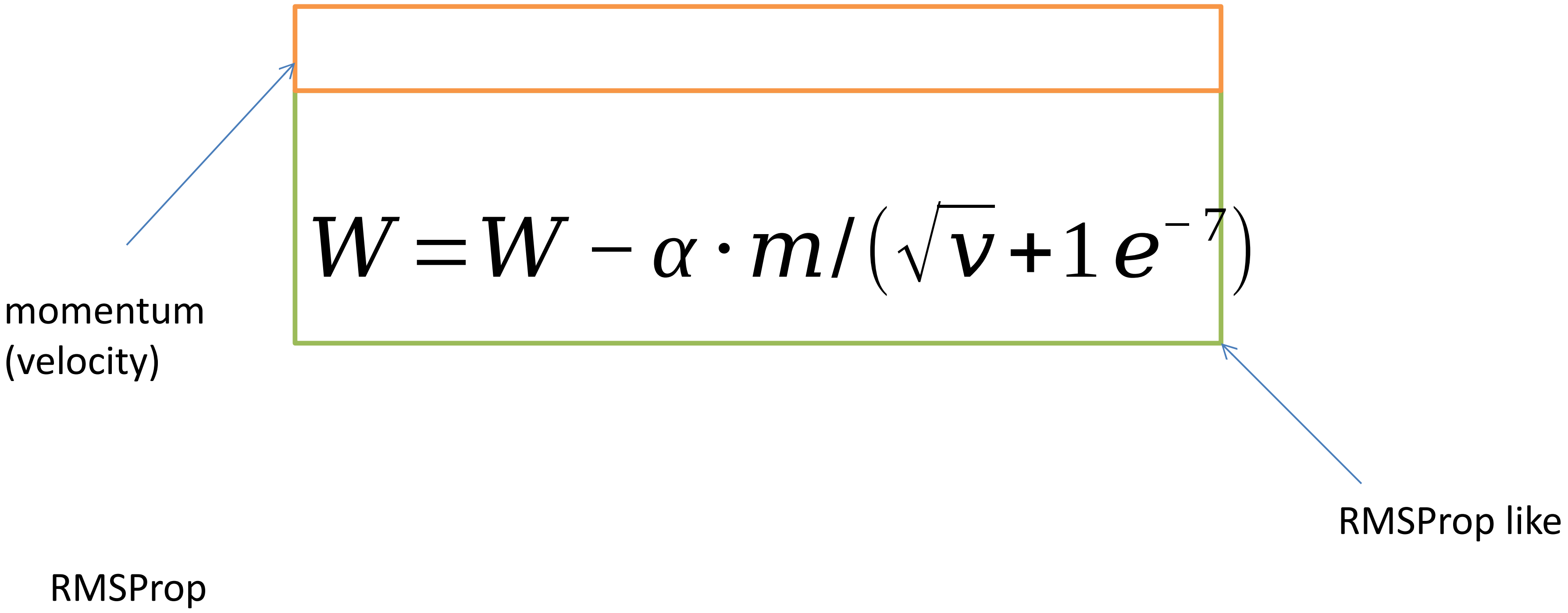
Introduced by Geoffrey Hinton on a lecture on Coursera:



[14] Duchi J, Hazan E and Singer Y 2011 *The Journal of Machine Learning Research* 12 2121
[15] Tieleman T and Hinton E 2012 Lecture 6.5 - rmsprop, COURSERA: Neural networks for machine learning.

Adam

- Combine RMSProp with momentum



$$W = W - \alpha \cdot m / (\sqrt{v} + 1e^{-7})$$

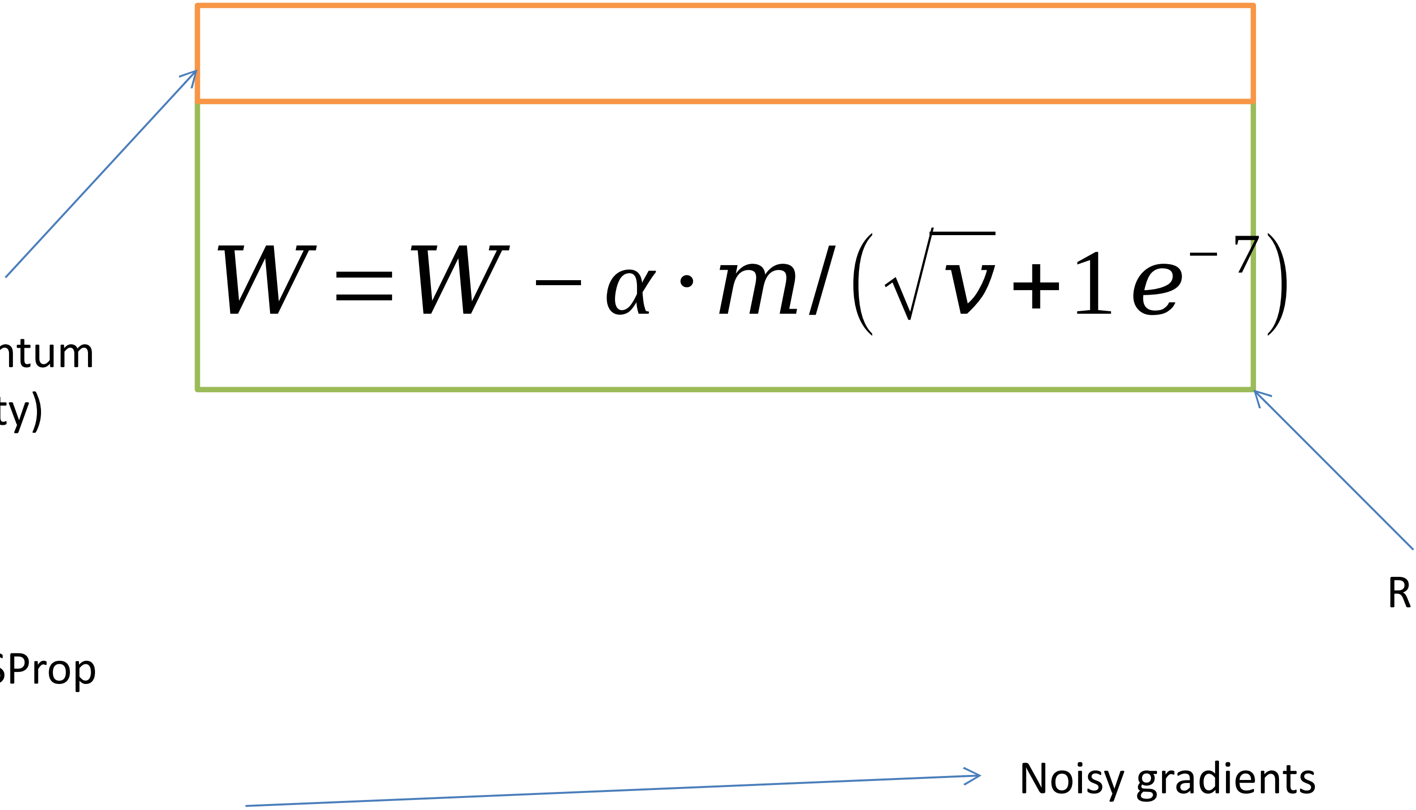
momentum
(velocity)

RMSProp like

RMSProp

Adam

- Combine RMSProp with momentum



$$W = W - \alpha \cdot m / \left(\sqrt{v} + 1e^{-7}\right)$$

momentum
(velocity)

RMSProp like

RMSProp

Noisy gradients

Adam

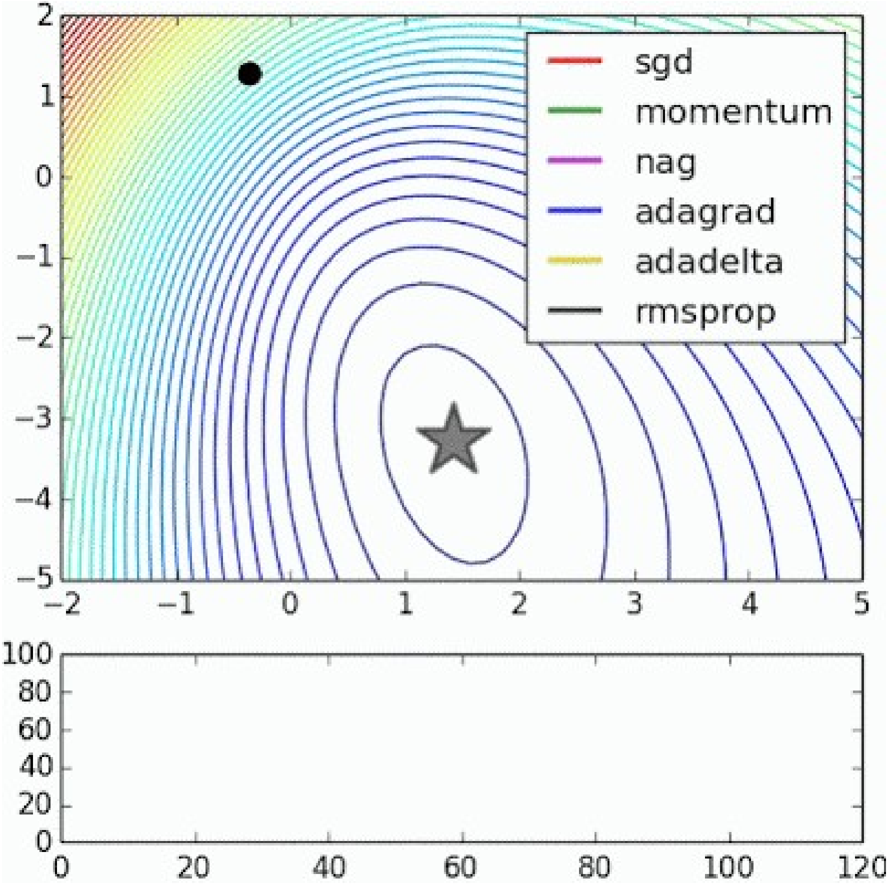
- Combine RMSProp with momentum

for t in range(0, iterations):

$$\left\{\begin{array}{l} \\ \nu = \nu/\left(1-\beta_2^t\right) \end{array}\right.$$

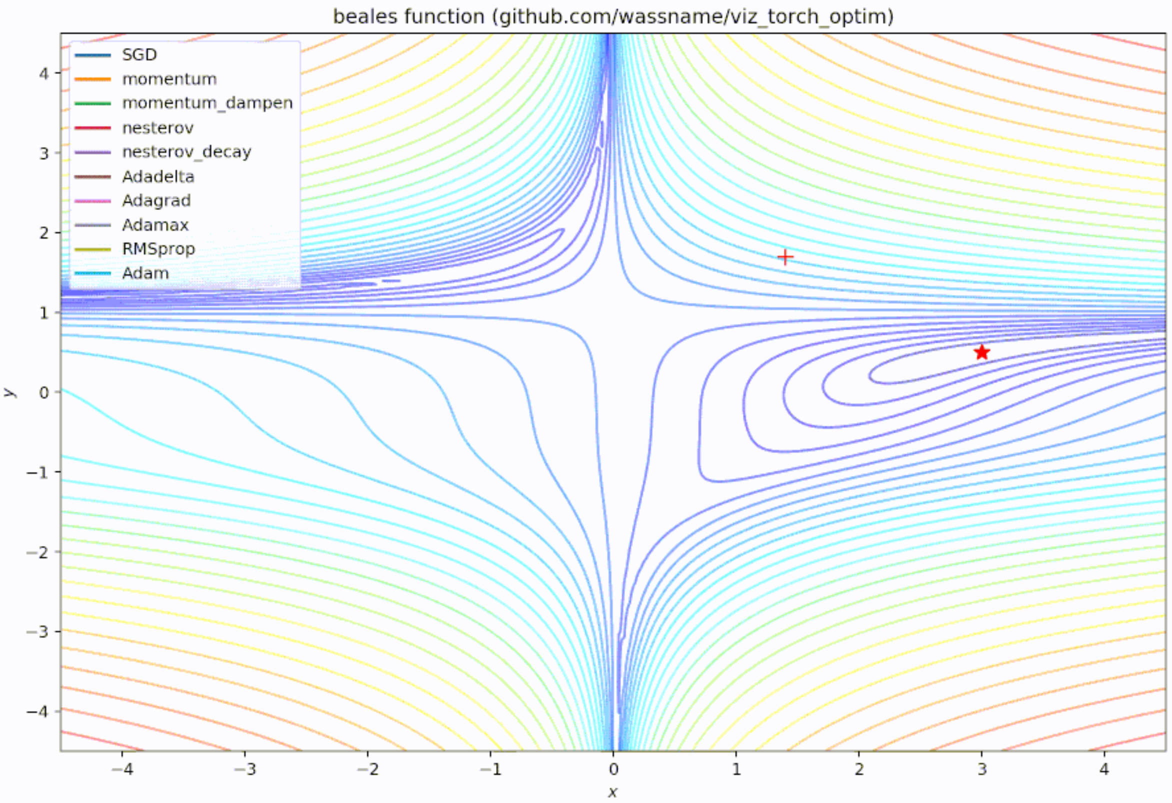
Bias correction

$$W = -\alpha\cdot m/\left(\sqrt{\nu}+1e^{-7}\right)$$



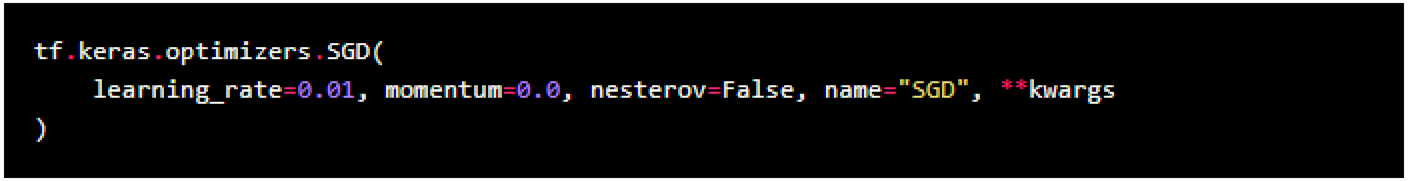
http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Optimization algorithms

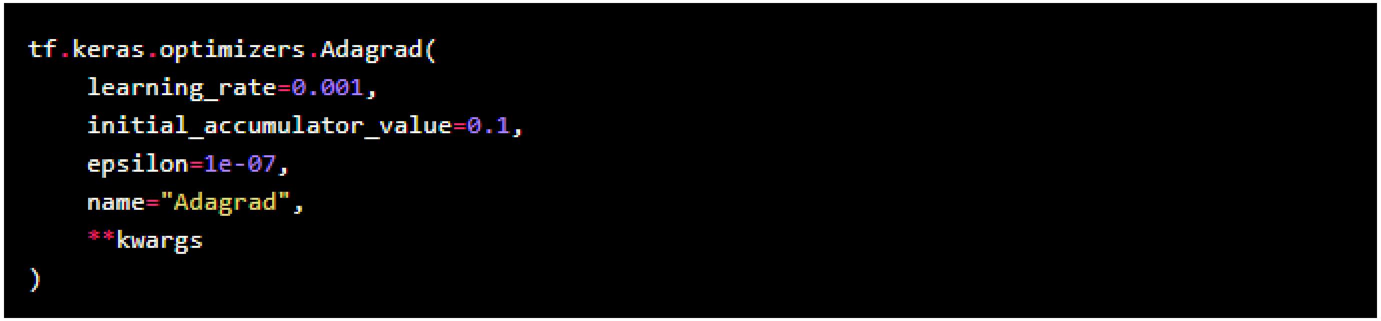


https://awesomeopensource.com/project/3springs/viz_torch_optim

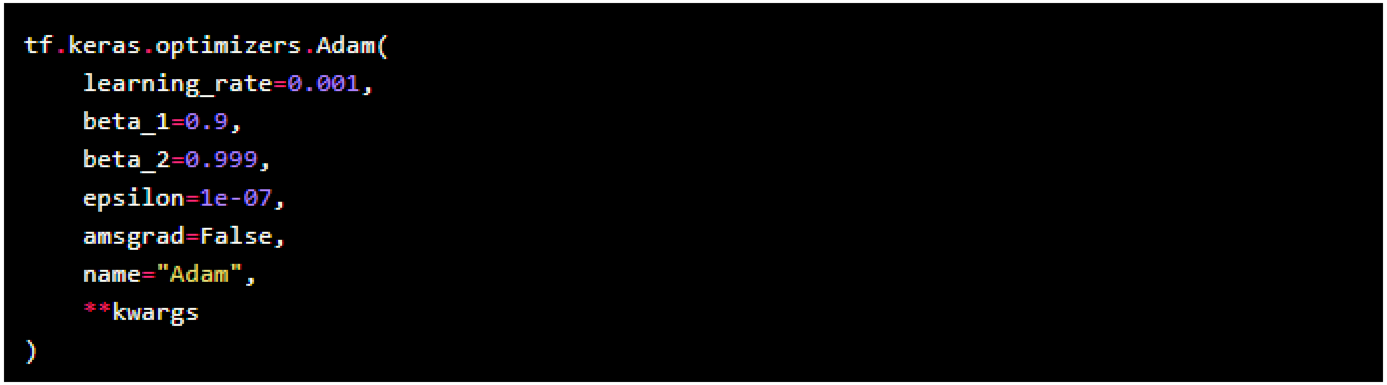
Optimizers in keras



```
tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.0, nesterov=False, name="SGD", **kwargs
)
```



```
tf.keras.optimizers.Adagrad(
    learning_rate=0.001,
    initial_accumulator_value=0.1,
    epsilon=1e-07,
    name="Adagrad",
    **kwargs
)
```



```
tf.keras.optimizers.Adam(
    learning_rate=0.001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    amsgrad=False,
    name="Adam",
    **kwargs
)
```

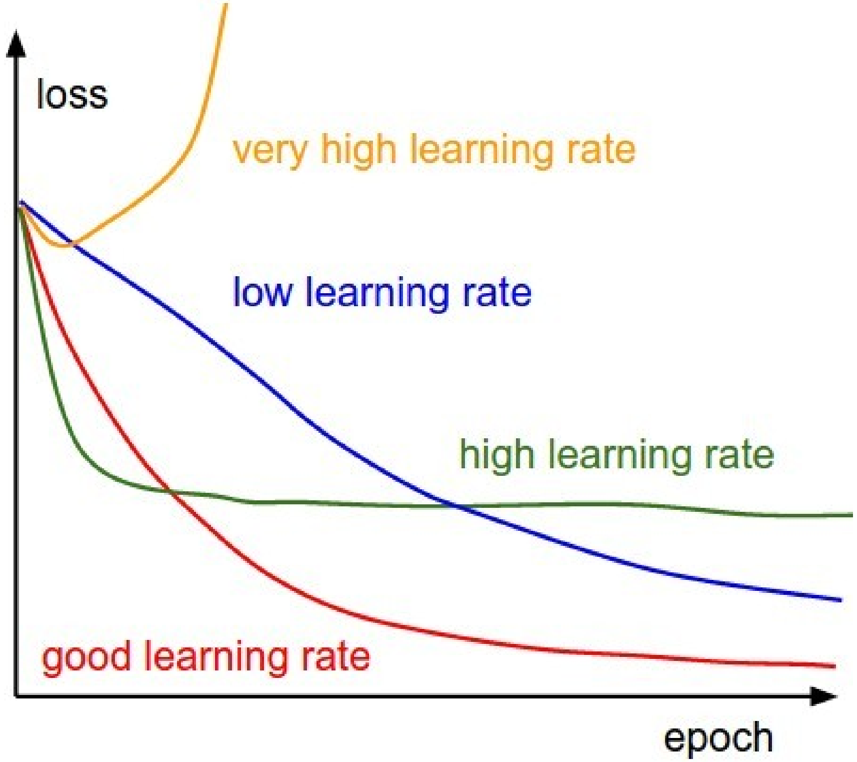
Optional additional reading

- https://ruder.io/optimizing-gradient-descent/index.html#otherrecentoptimizers
- https://www.youtube.com/watch?v=k8fTYJPd3_I
- https://www.youtube.com/watch?v=_e-LFe_igno
- https://www.coursera.org/lecture/deep-neural-network/the-problem-of-local-optima-RFANA

Training a neural network

Learning rate scheduling

Learning rate



Learning rate decay

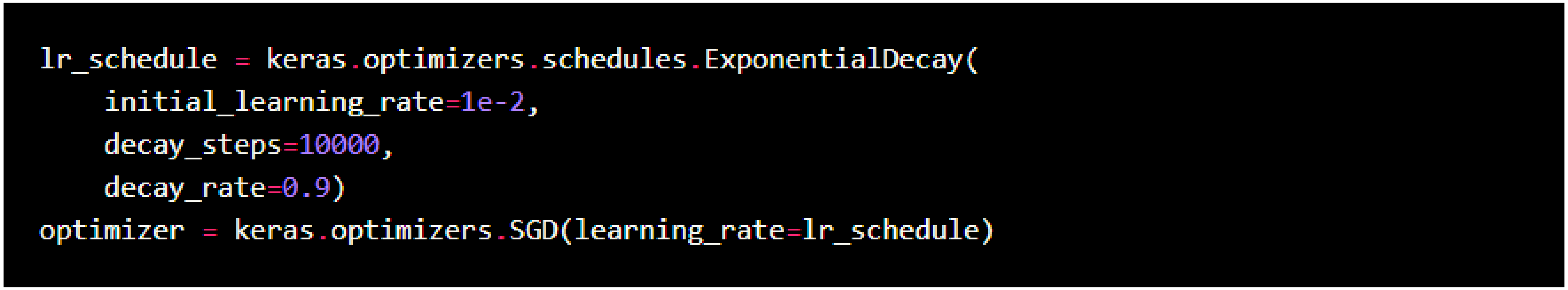
- Learning rate decay over time!
  - Use a larger learning rate at the beginning and progressively reduce the learning rate
  - *t* epoch, *k* – decay rate
- Step decay:
  - reduce learning rate by half after some epochs
- Exponential decay

$$\alpha = \alpha_0 \cdot e^{-kt}$$

- 1/t decay

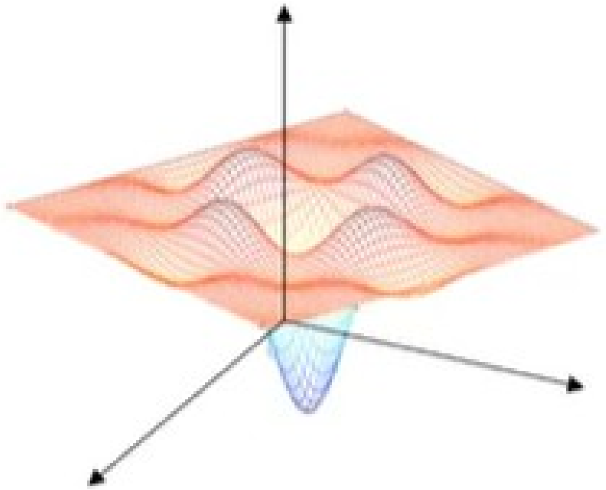
$$\alpha = \alpha_0 / (1 + kt)$$

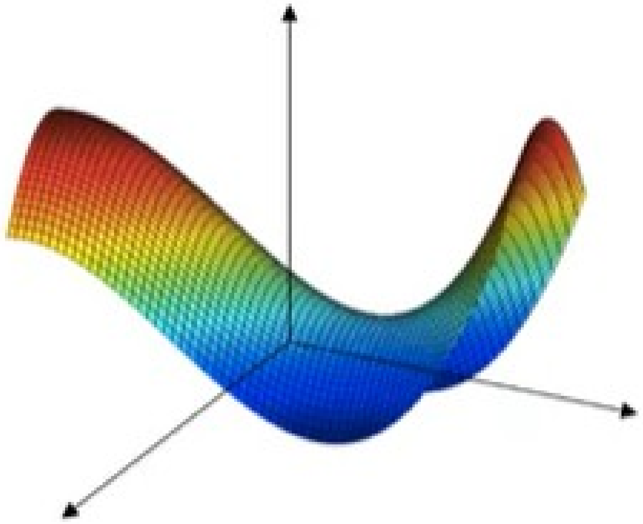
keras optimizers schedules



```
lr_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=1e-2,
    decay_steps=10000,
    decay_rate=0.9)
optimizer = keras.optimizers.SGD(learning_rate=lr_schedule)
```

Problem of local optima



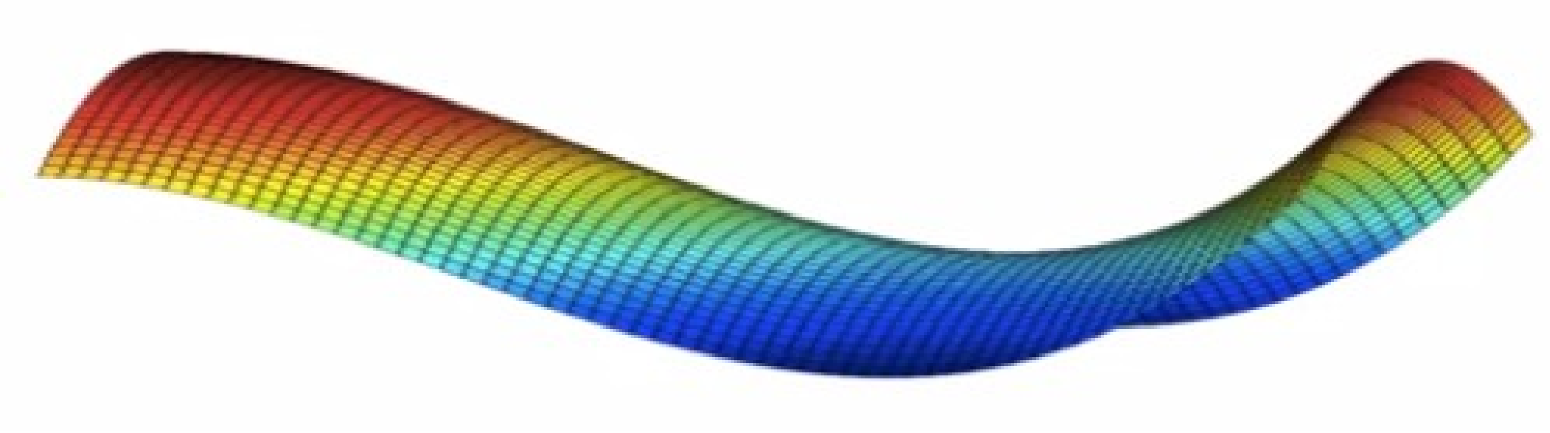


Gradient is zero -> in each direction it can either be a convex like a concave like function.
Saddle points.

Unlikely to get stuck in local optima

https://
www.coursera.org/lecture/deep-neural-network/the-problem-of-l

Problem of plateaus

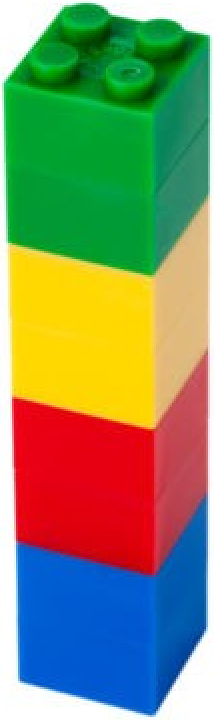


Plateaus can make the learning process too slow

Previously

Convolutional neural networks

- Convolutional neural networks
  - Convolutional layers
  - Pooling layers
  - Fully connected layers



Convolution layers

Shared parameters

Sparsity of connections

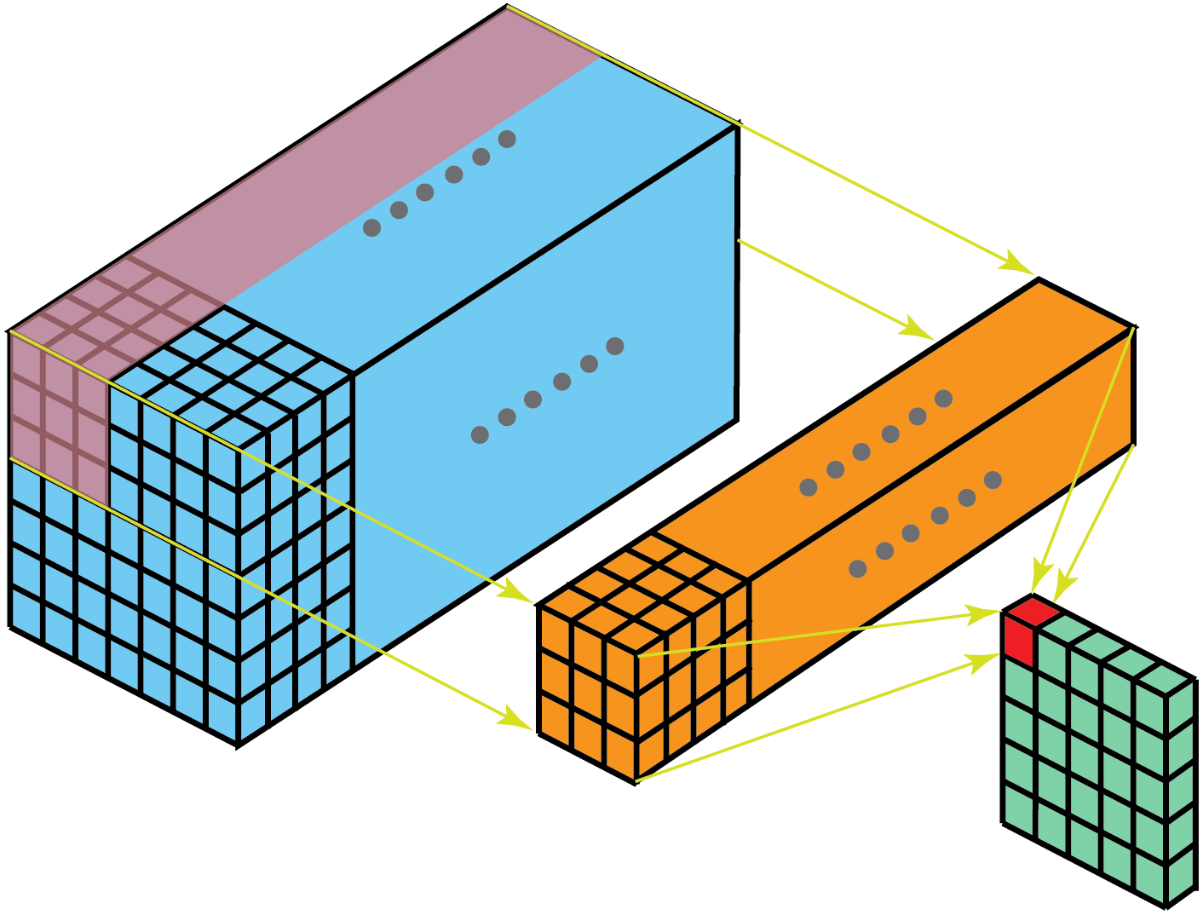
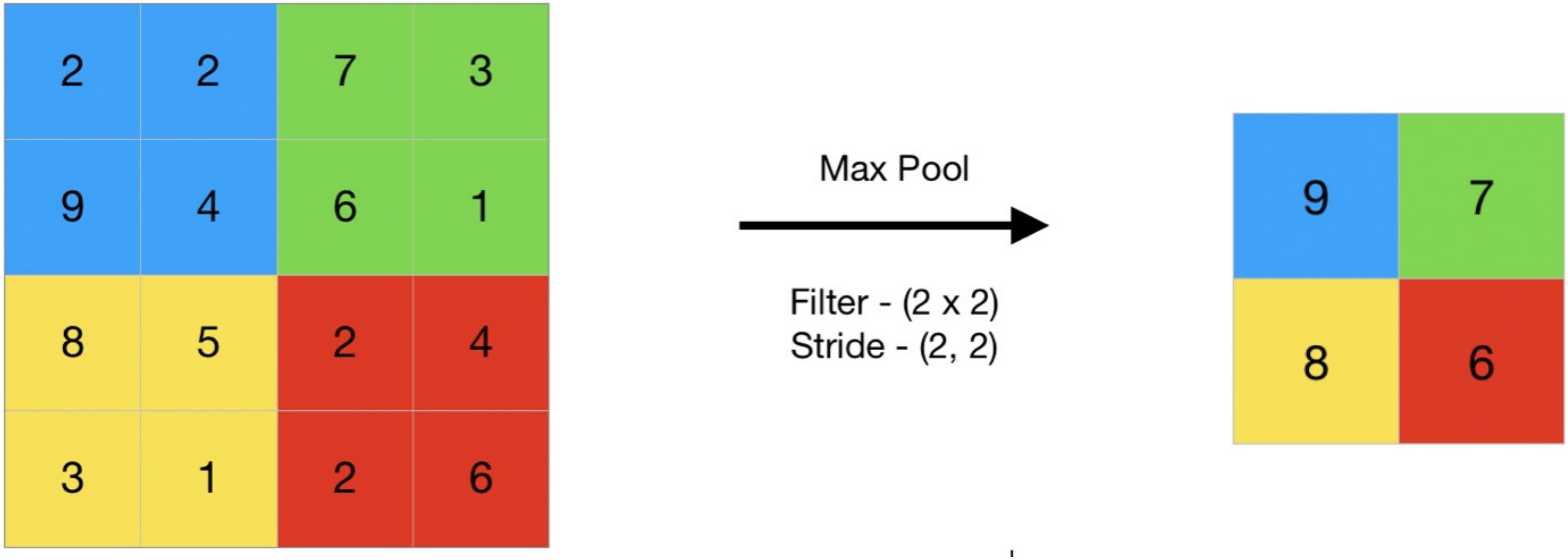


Image source: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Pooling layers



2	2	7	3
9	4	6	1
8	5	2	4
3	1	2	6

Max Pool

Filter - (2 x 2)
Stride - (2, 2)

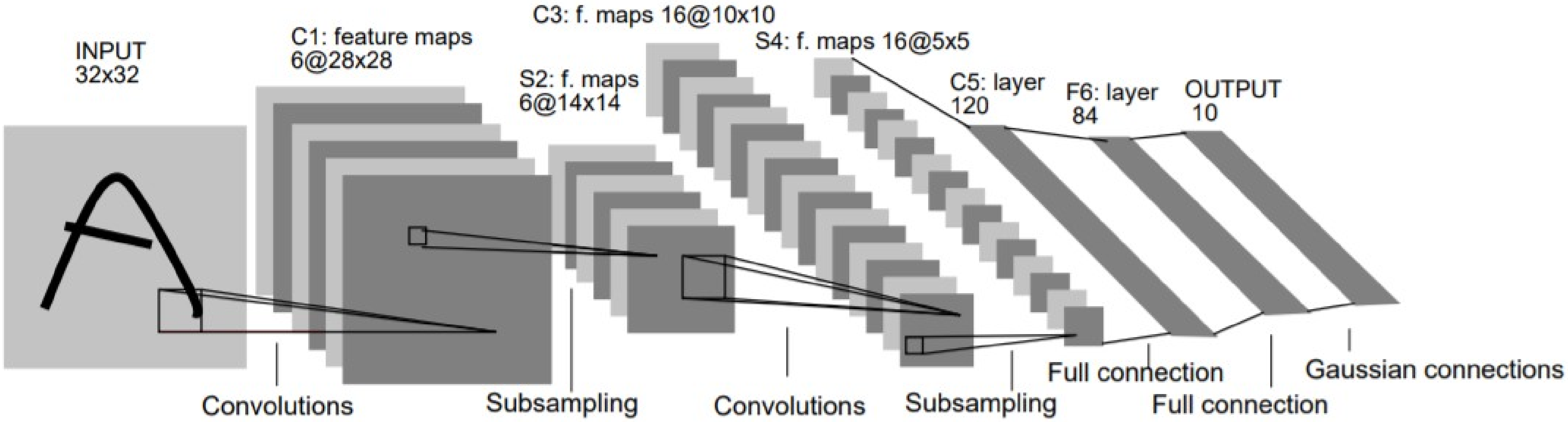
9	7
8	6

Previously

Convolutional neural networks

- Training a neural network
  - Good initialization
  - Batch normalization
  - Regularization
  - Batch training
    - Stochastic gradient descent
    - Mini-batch gradient descent
    - Batched gradient descent
  - Optimizers

LeNet 5

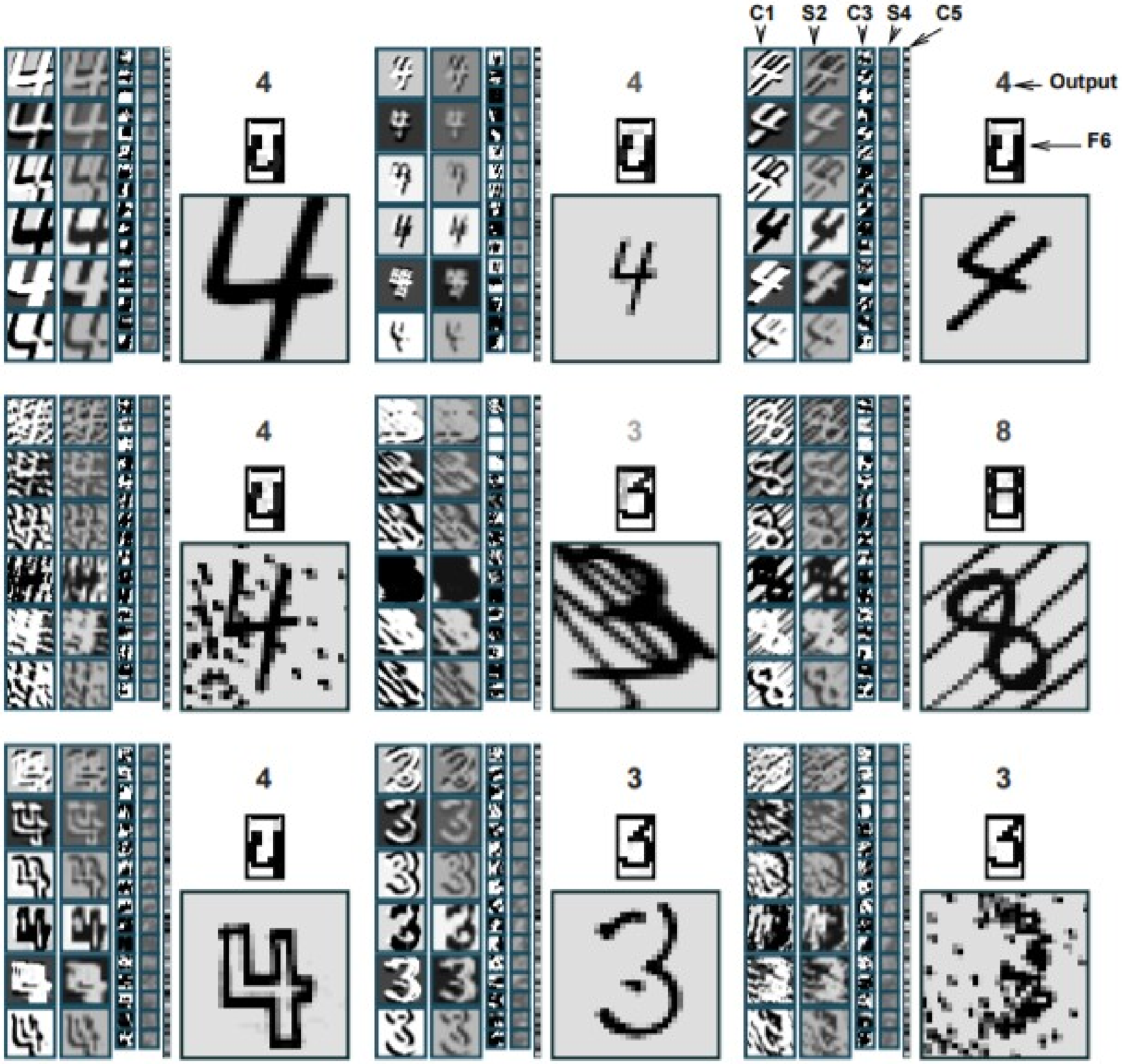


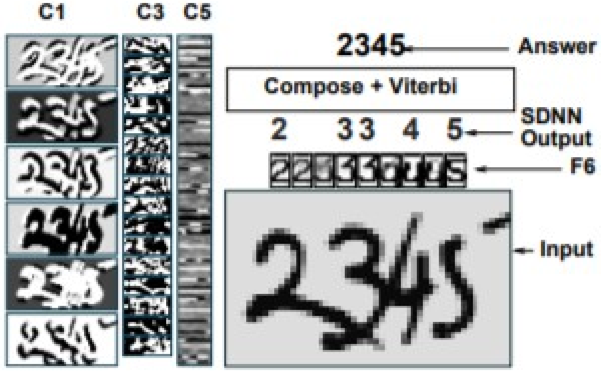
Conv filters have size 5x5 and are applied at stride 1

Subsampling (Pooling) layers have size 2x2 and are applied at stride 2

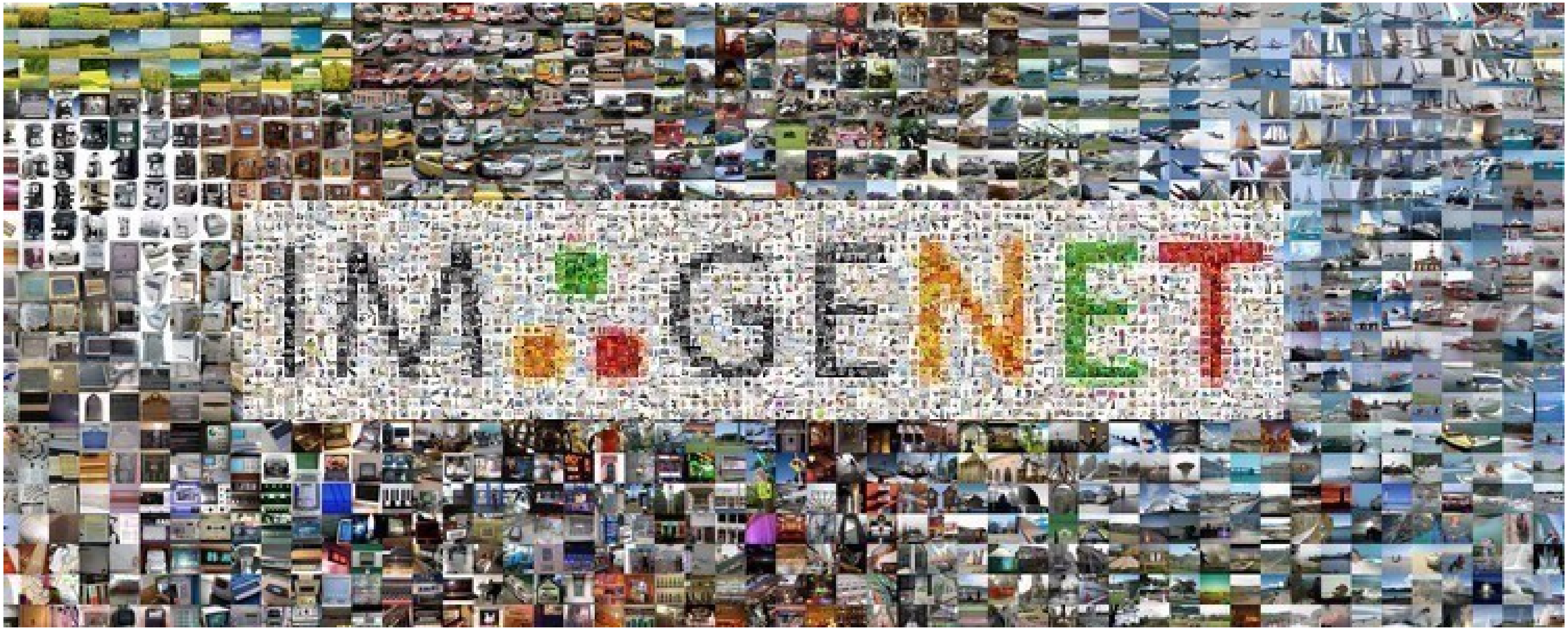
[CONV-POOL-CONV-POOL-FC-FC]

LeNet 5

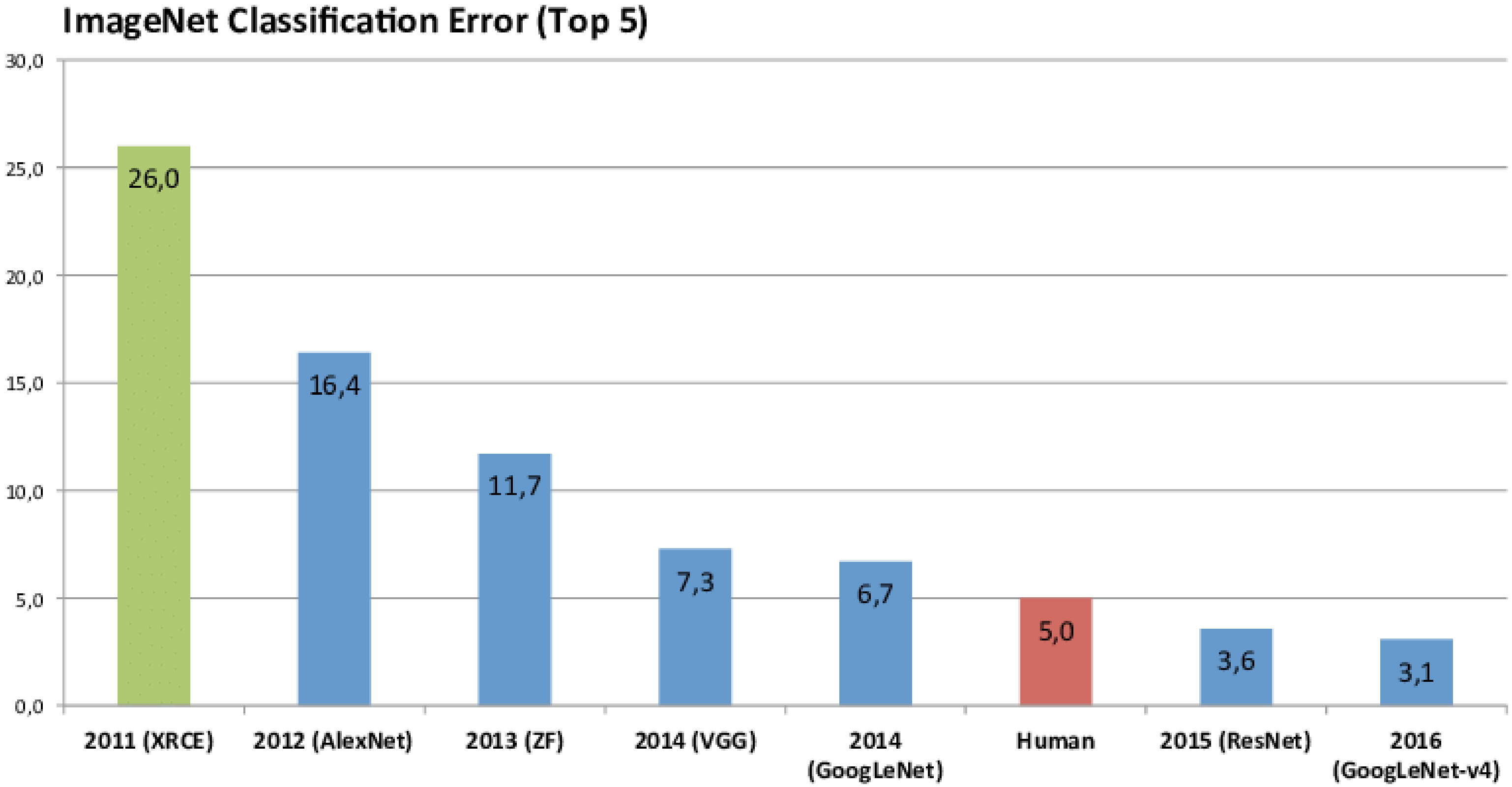

C1
S2
C3
S4
C5
4
4
4
Output
F6
4
3
8
4
3
3


C1
C3
C5
2345
Answer
Compose + Viterbi
SDNN Output
2 33 4 5
F6
Input

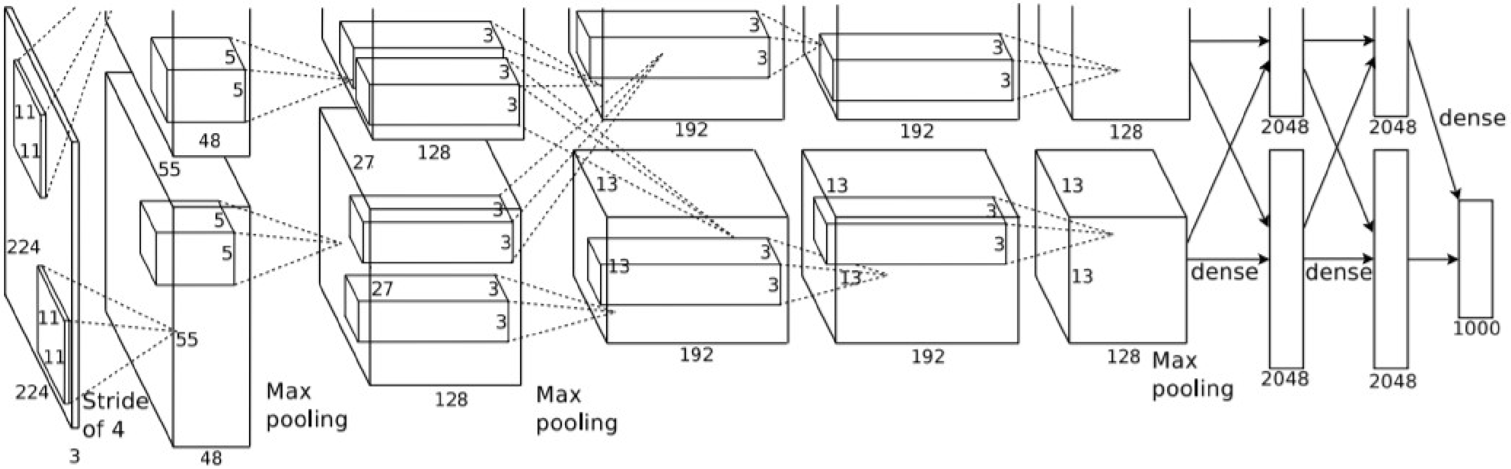
2012



14,197,122 images
1000 categories

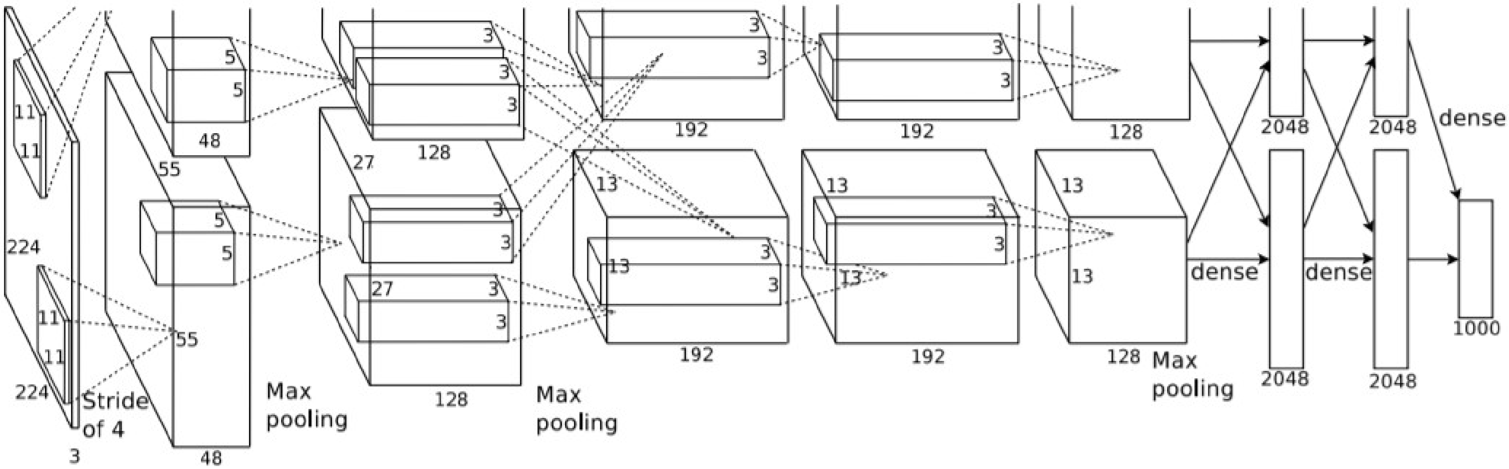

ImageNet Classification Error (Top 5)
30,0
25,0
20,0
15,0
10,0
5,0
0,0
26,0
16,4
11,7
7,3
6,7
5,0
3,6
3,1
2011 (XRCE)
2012 (AlexNet)
2013 (ZF)
2014 (VGG)
2014 (GoogLeNet)
Human
2015 (ResNet)
2016 (GoogLeNet-v4)

Alexnet, 2012



A single GTX 580 GPU has only 3GB of memory, which limits the maximum size of the networks that can be trained on it.

Alexnet, 2012

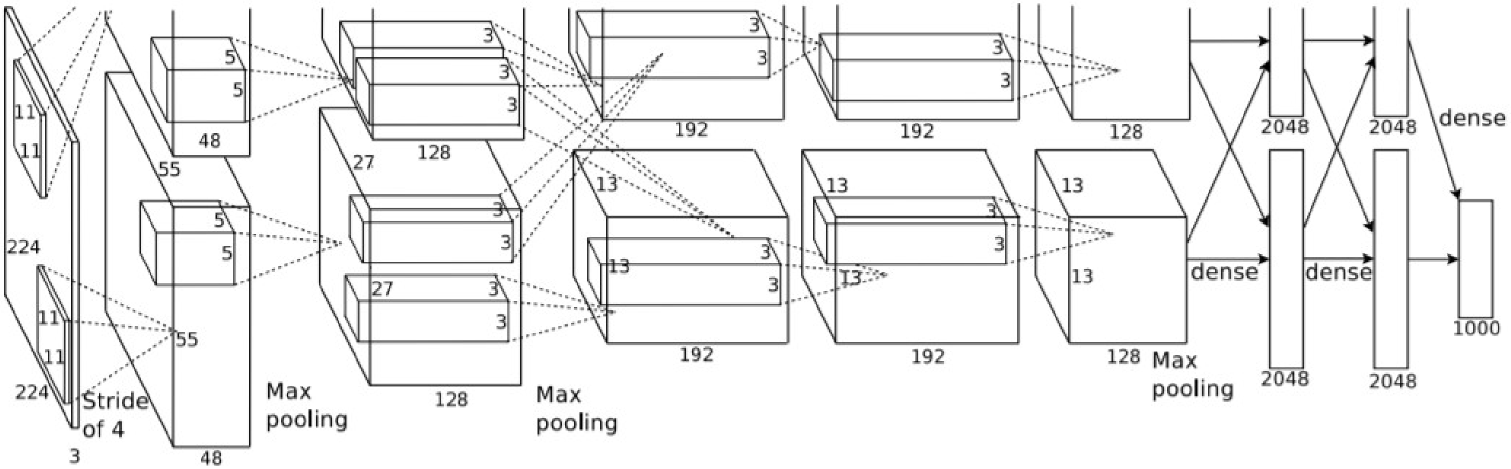


$$W_o = \frac{W_I - F + 2P}{S} + 1$$

Image size: (227, 227, 3)
First layer: 96 filter of size 11x11

What is the output volume size and the number of parameter in this layer?

Alexnet, 2012

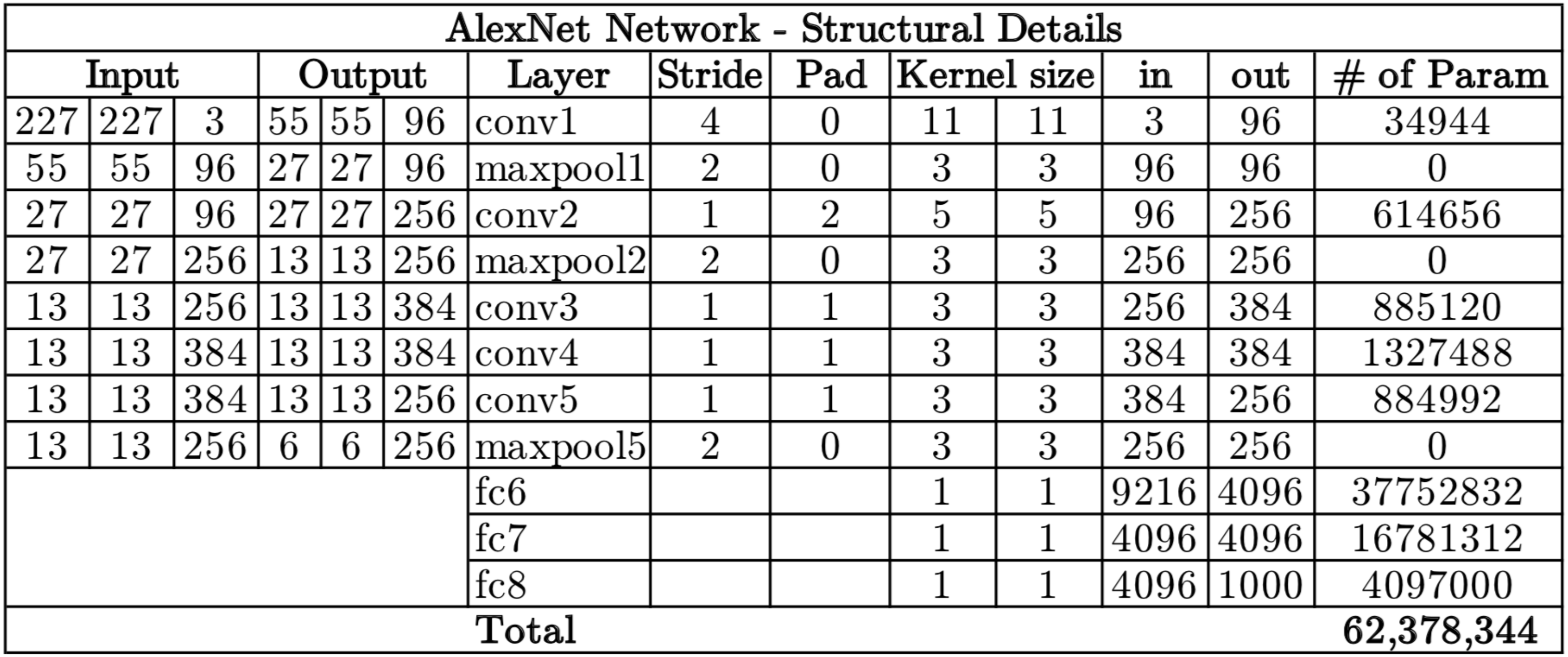


$$W_o = \frac{W_I - F + 2P}{S} + 1$$

Image size: (227, 227, 3)
First layer: 96 filter of size 11x11

Output volume: 55x55x96
Parameters: (11*11*3)*96 ~ 35K

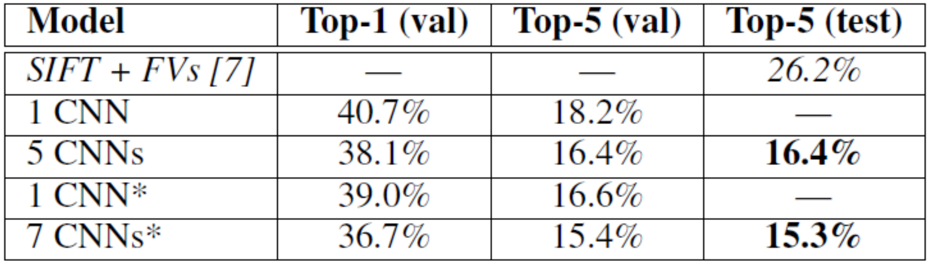
Alexnet



AlexNet Network - Structural Details

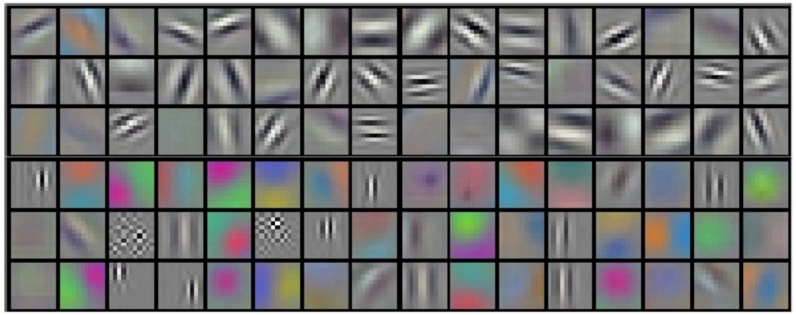
Input			Output			Layer	Stride	Pad	Kernel size		in	out	# of Param
227	227	3	55	55	96	conv1	4	0	11	11	3	96	34944
55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	0
27	27	96	27	27	256	conv2	1	2	5	5	96	256	614656
27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	0
13	13	256	13	13	384	conv3	1	1	3	3	256	384	885120
13	13	384	13	13	384	conv4	1	1	3	3	384	384	1327488
13	13	384	13	13	256	conv5	1	1	3	3	384	256	884992
13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	0
						fc6			1	1	9216	4096	37752832
						fc7			1	1	4096	4096	16781312
						fc8			1	1	4096	1000	4097000
						Total							**62,378,344**

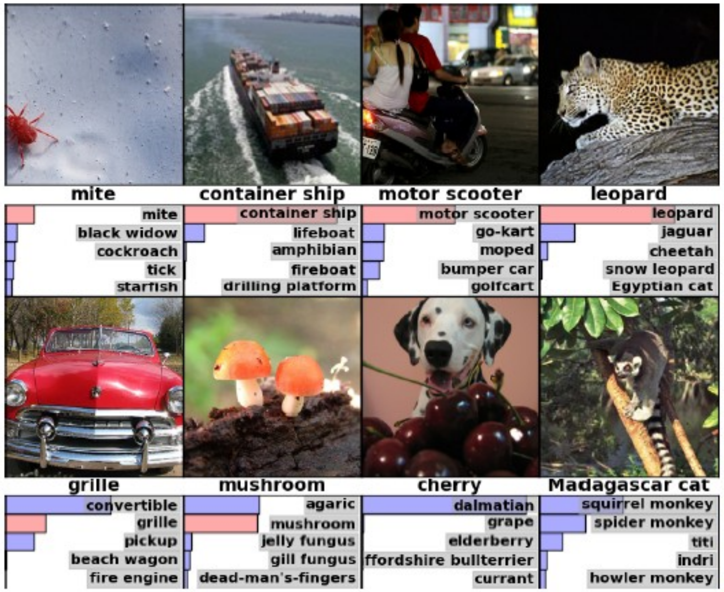
Alexnet

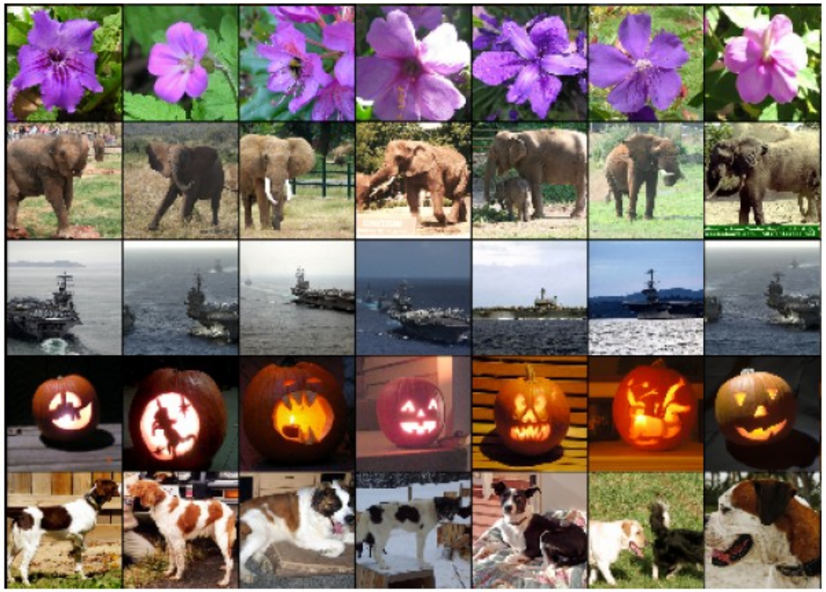


Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	*—*	*—*	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

Table 2: Comparison of error rates on ILSVRC-2012 validation and test sets. In *italics* are best results achieved by others. Models with an asterisk* were "pre-trained" to classify the entire ImageNet 2011 Fall release. See Section 6 for details.

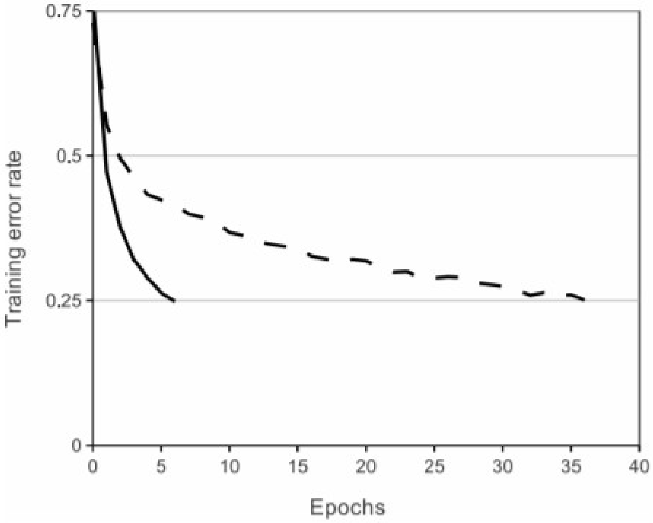






Alexnet – key features

- First use of ReLU
- Overlapped max pooling
- Used normalization layers
  - Not used anymore



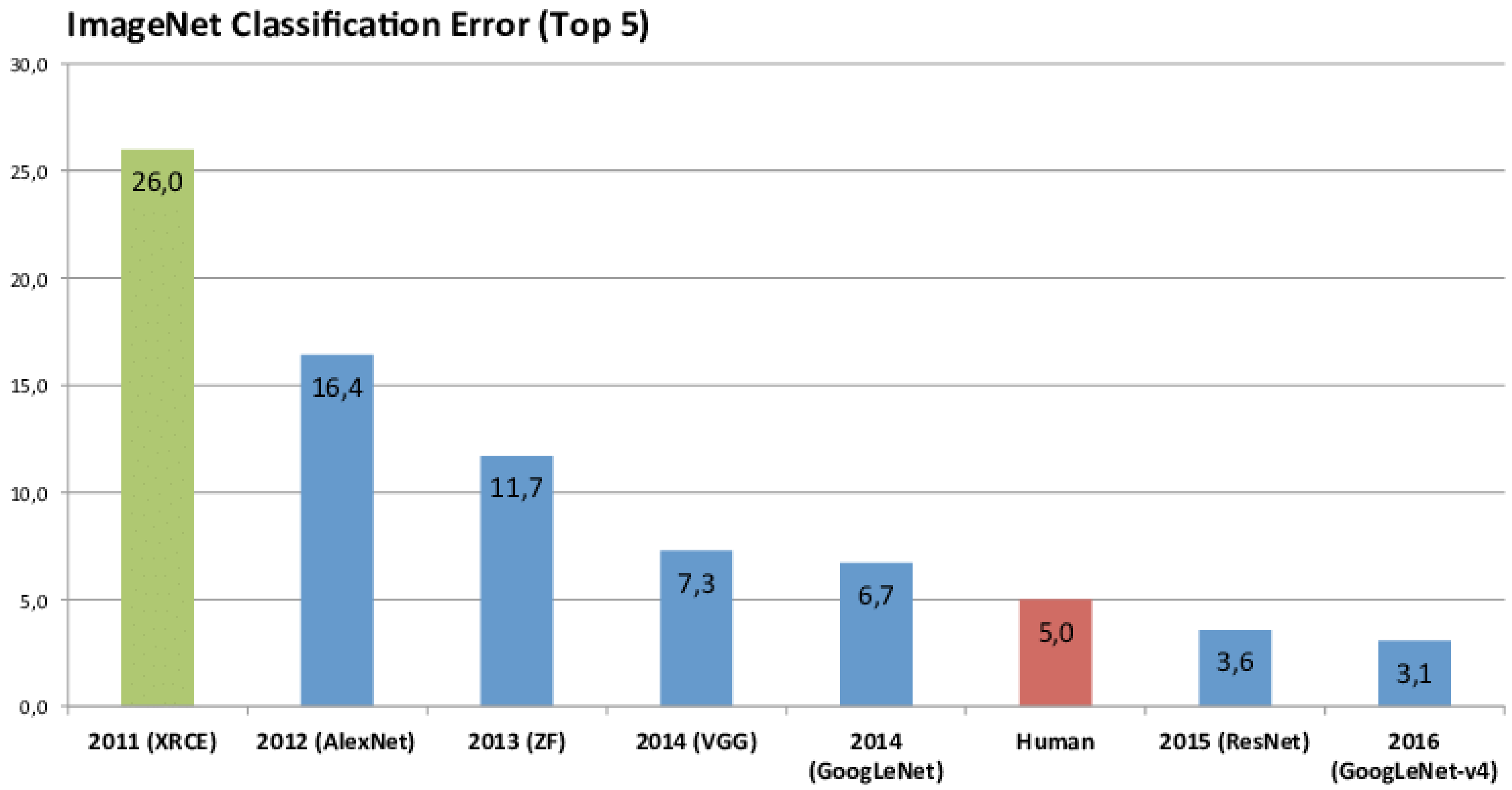
RELU (solid line) effect on training vs tanh (dashed line)

$$b^i_{x,y} = a^i_{x,y} \Bigg/ \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^\beta$$

Alexnet – key features

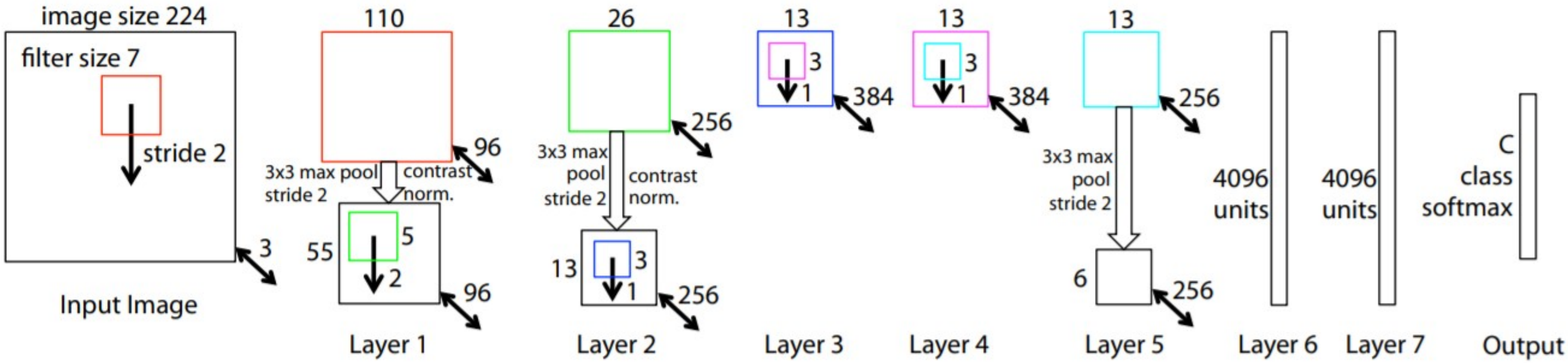
- Training setup
  - Dropout 0.5
  - Data augmentation
  - Batch size 128
  - Gradient descent with momentum (beta = 0.9)
  - Initial learning rate: 1e-2, reduced manually, when a plateau was reached
  - 7 Alexnet ensemble: 18.2% -> 15.4%

ZFNet, 2013



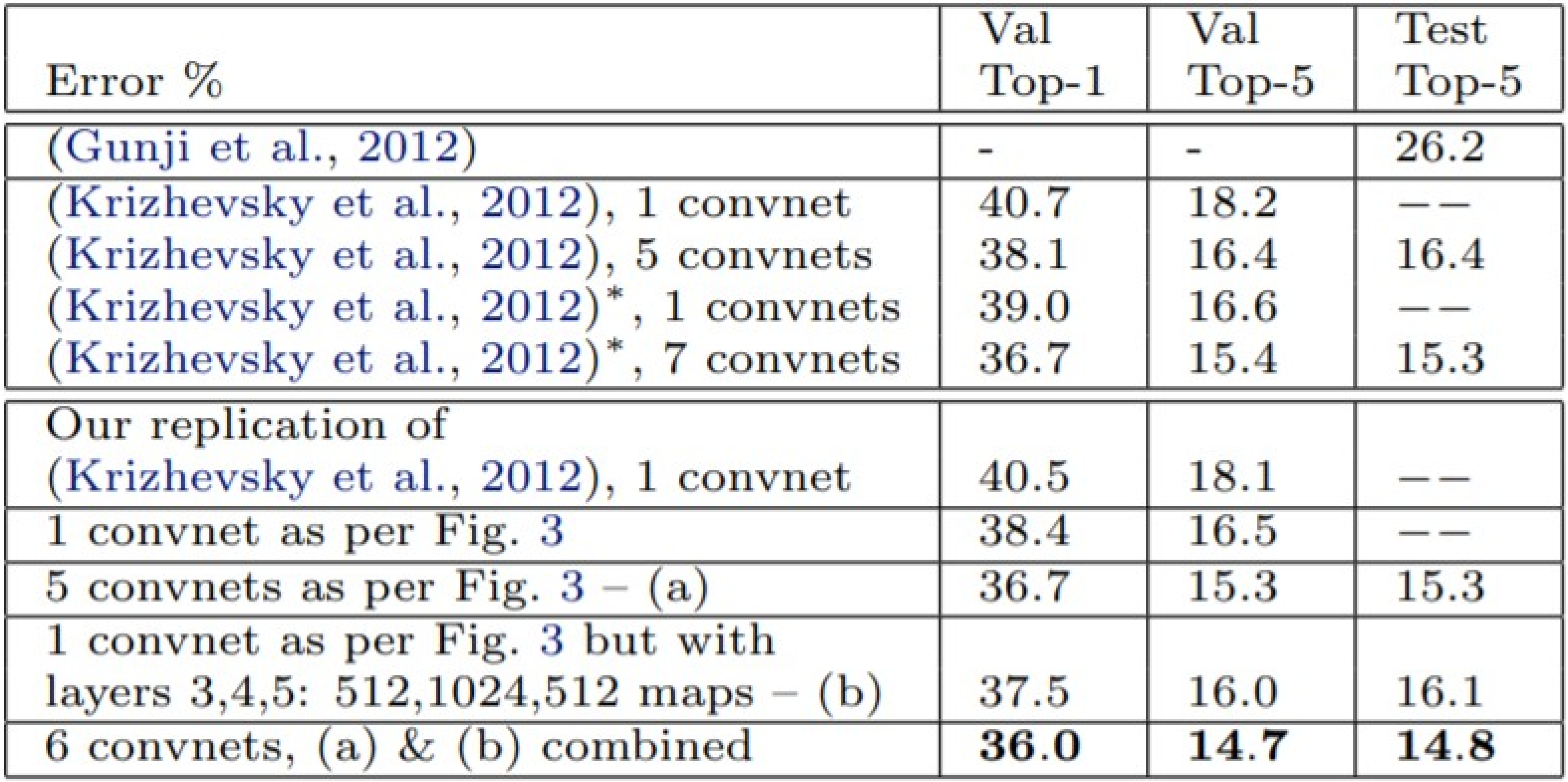
Built on top of Alexnet, improved hyperparameters

ZFNet



CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3, 4, 5: instead of 384, 384, 256 filters use 512, 1024, 512

ZFNet



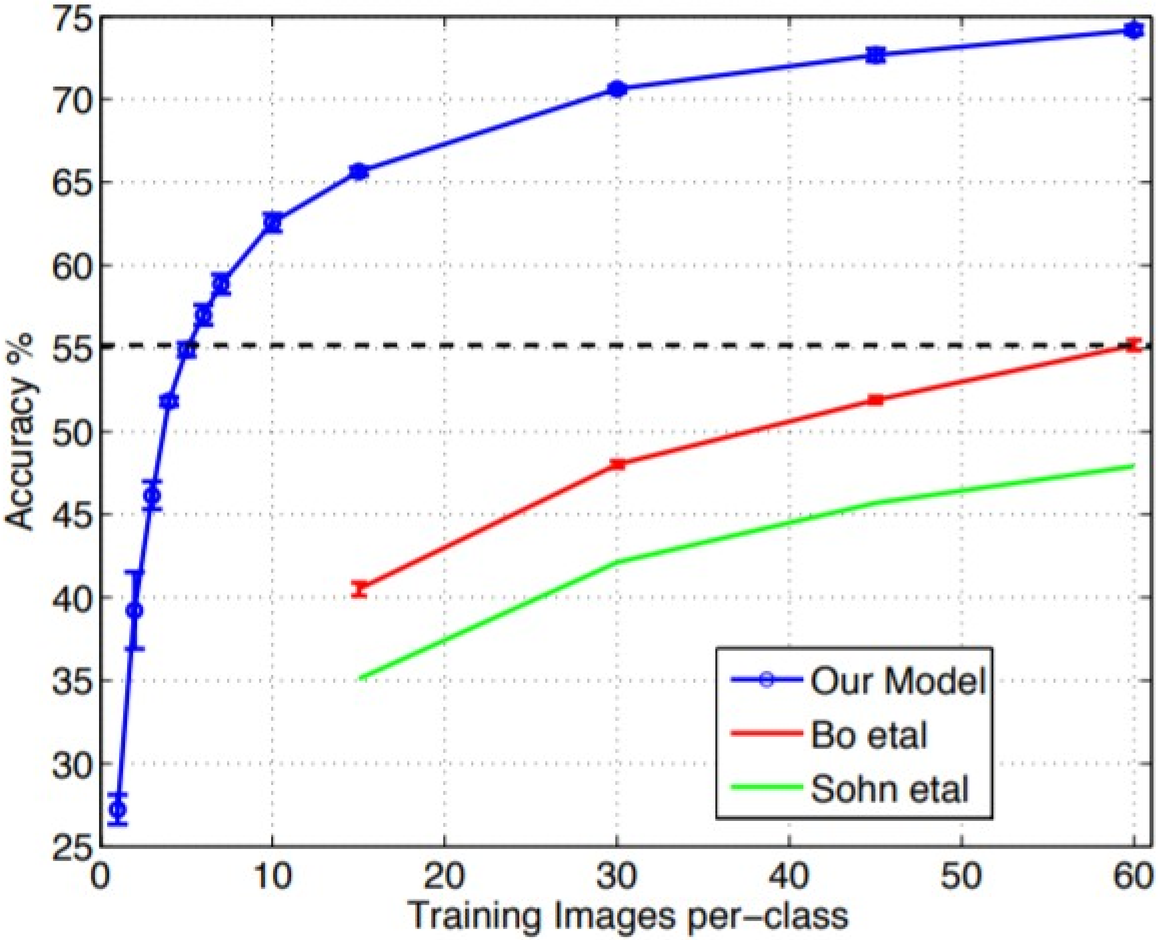
Error %	Val Top-1	Val Top-5	Test Top-5
(Gunji et al., 2012)	-	-	26.2
(Krizhevsky et al., 2012), 1 convnet	40.7	18.2	−−
(Krizhevsky et al., 2012), 5 convnets	38.1	16.4	16.4
(Krizhevsky et al., 2012)*, 1 convnets	39.0	16.6	−−
(Krizhevsky et al., 2012)*, 7 convnets	36.7	15.4	15.3
Our replication of (Krizhevsky et al., 2012), 1 convnet	40.5	18.1	−−
1 convnet as per Fig. 3	38.4	16.5	−−
5 convnets as per Fig. 3 – (a)	36.7	15.3	15.3
1 convnet as per Fig. 3 but with layers 3,4,5: 512,1024,512 maps – (b)	37.5	16.0	16.1
6 convnets, (a) & (b) combined	**36.0**	**14.7**	**14.8**

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: 15.4% -> 14.8%

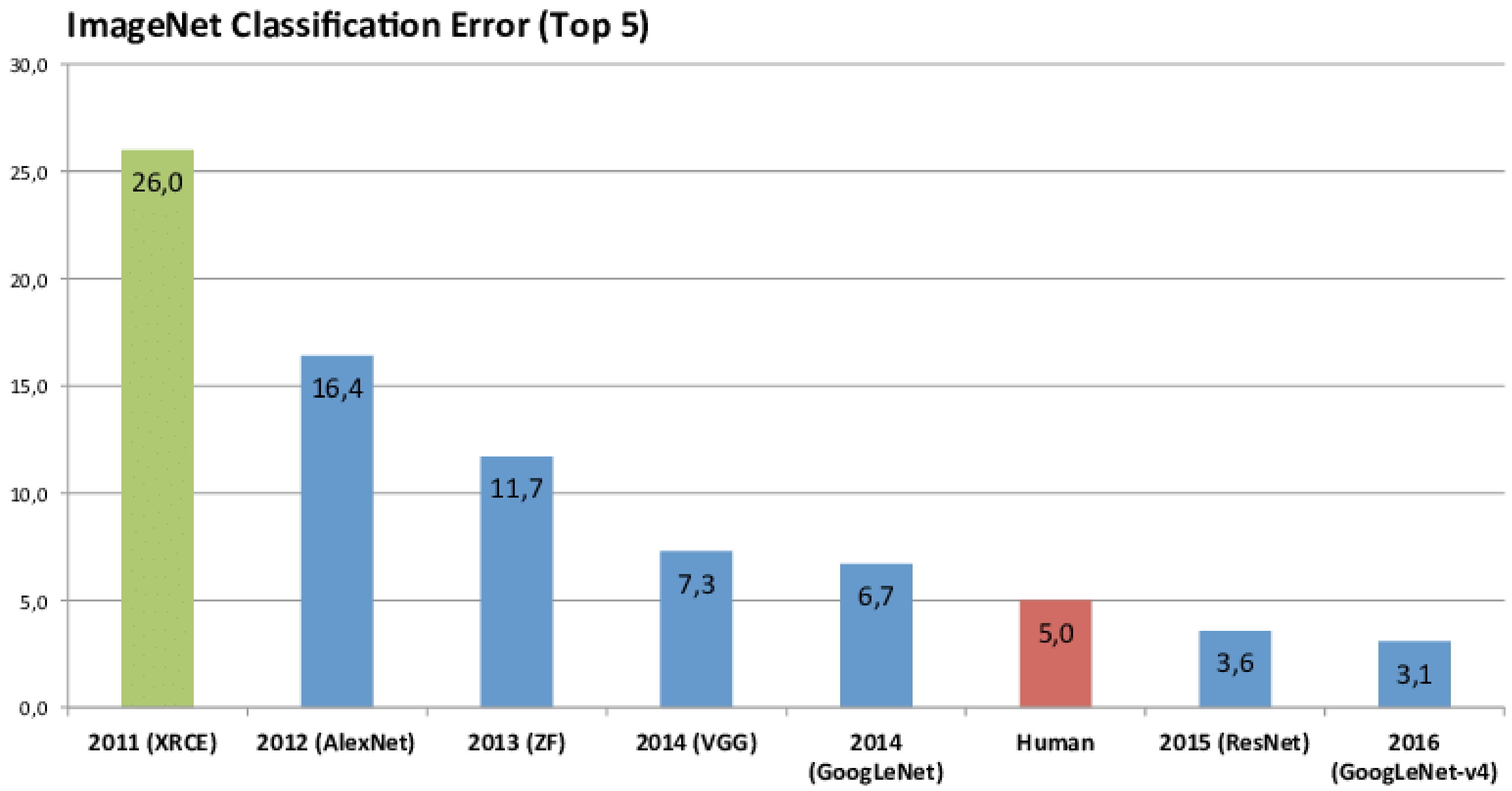
clarifai

ZFNet – feature generalization

- Caltech 256



VGG, 2014


ImageNet Classification Error (Top 5)
30,0
25,0
20,0
15,0
10,0
5,0
0,0
26,0
16,4
11,7
7,3
6,7
5,0
3,6
3,1
2011 (XRCE)
2012 (AlexNet)
2013 (ZF)
2014 (VGG)
2014 (GoogLeNet)
Human
2015 (ResNet)
2016 (GoogLeNet-v4)

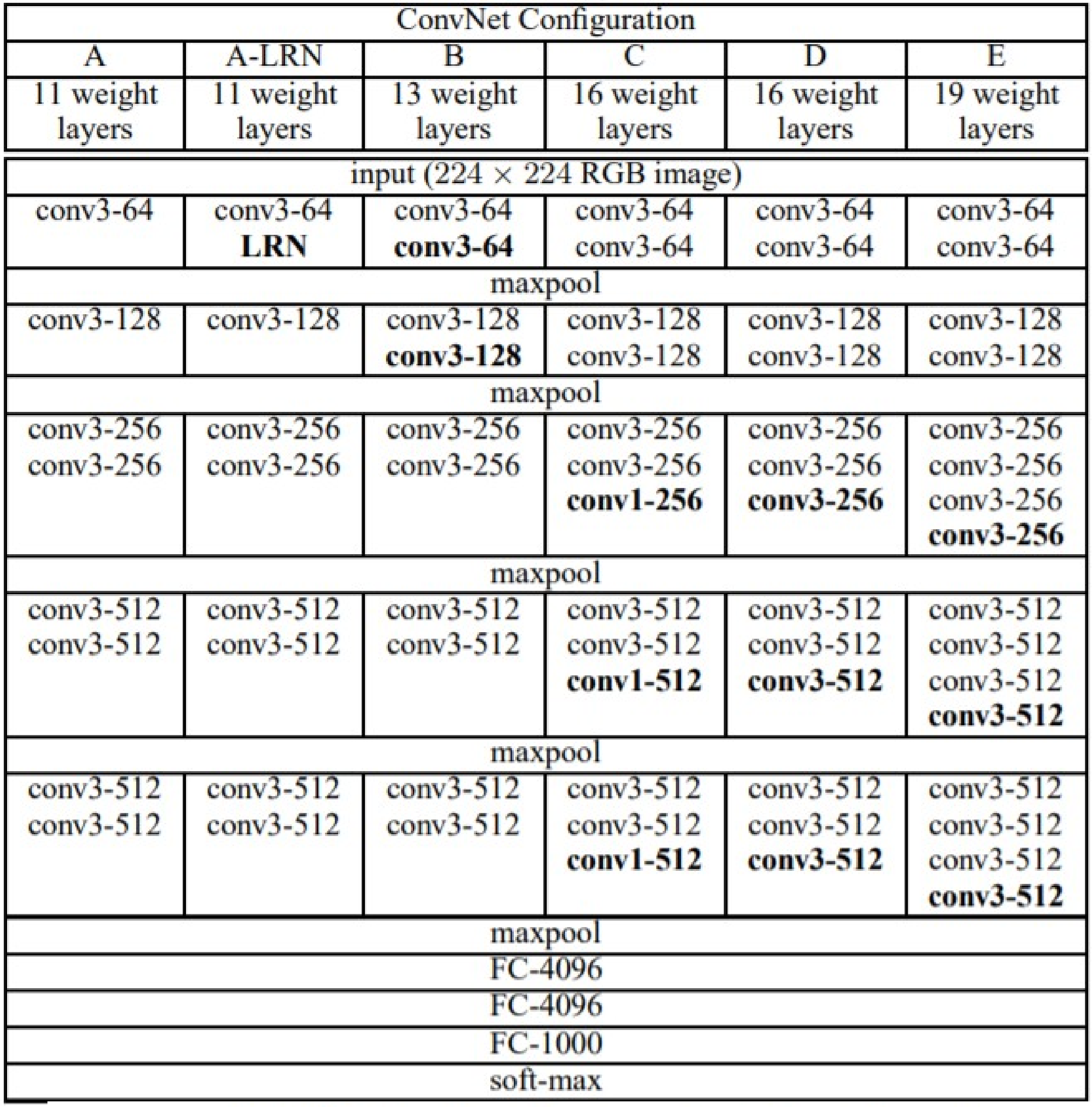
VGG

- Simplifies the network architectures
  - Always use 3x3 filters for convolutions
  - Always use max pooling of filter size 2x2 and a stride of 2

VGG

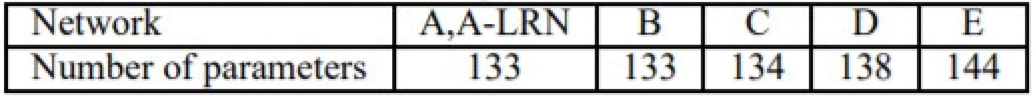
- Always use 3x3 filters for convolutions
- Always use max pooling of filter size 2x2 and a stride of 2

Top 5 accuracy: 11.7 -> 7.3



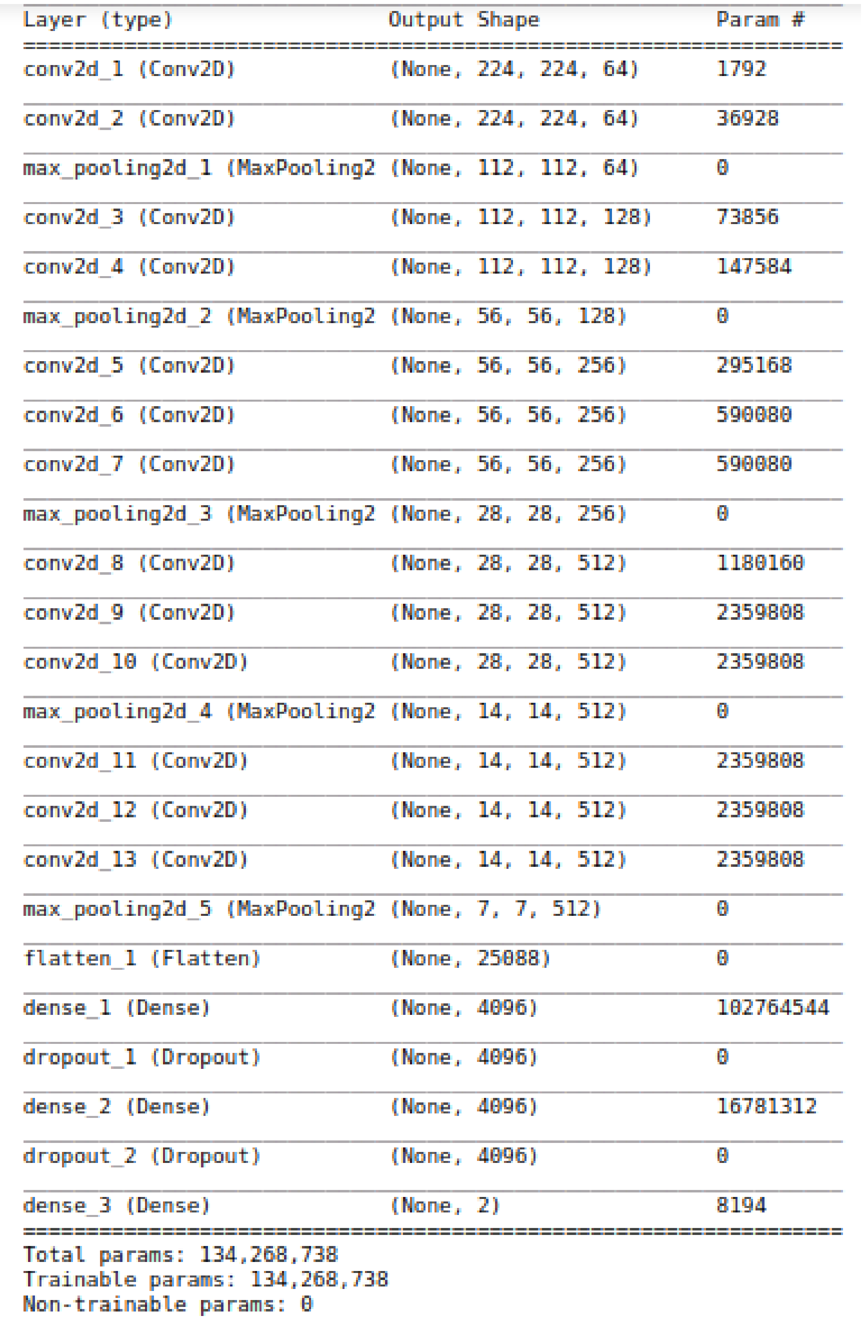
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	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					

Table 2: **Number of parameters** (in millions).



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

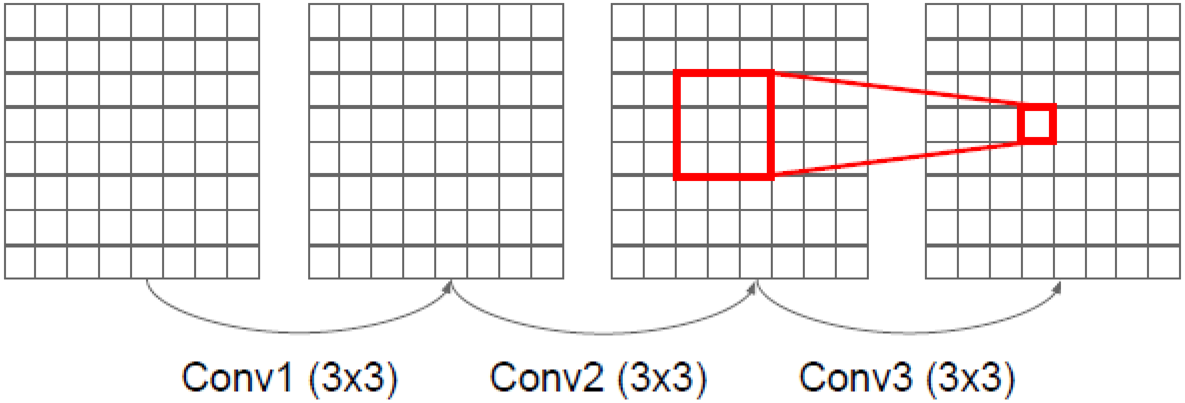
VGG



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 64)      1792      
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 224, 224, 64)      36928     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 64)      0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 112, 112, 128)     73856     
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 112, 112, 128)     147584    
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 128)       0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 56, 56, 256)       295168    
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 56, 56, 256)       590080    
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 56, 56, 256)       590080    
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 28, 28, 256)       0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 28, 28, 512)       1180160   
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 28, 28, 512)       2359808   
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 28, 28, 512)       2359808   
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 512)       0         
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 14, 14, 512)       2359808   
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 14, 14, 512)       2359808   
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 14, 14, 512)       2359808   
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 512)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0         
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              102764544 
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 4096)              16781312  
_________________________________________________________________
dropout_2 (Dropout)          (None, 4096)              0         
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 8194      
=================================================================
Total params: 134,268,738
Trainable params: 134,268,738
Non-trainable params: 0
```

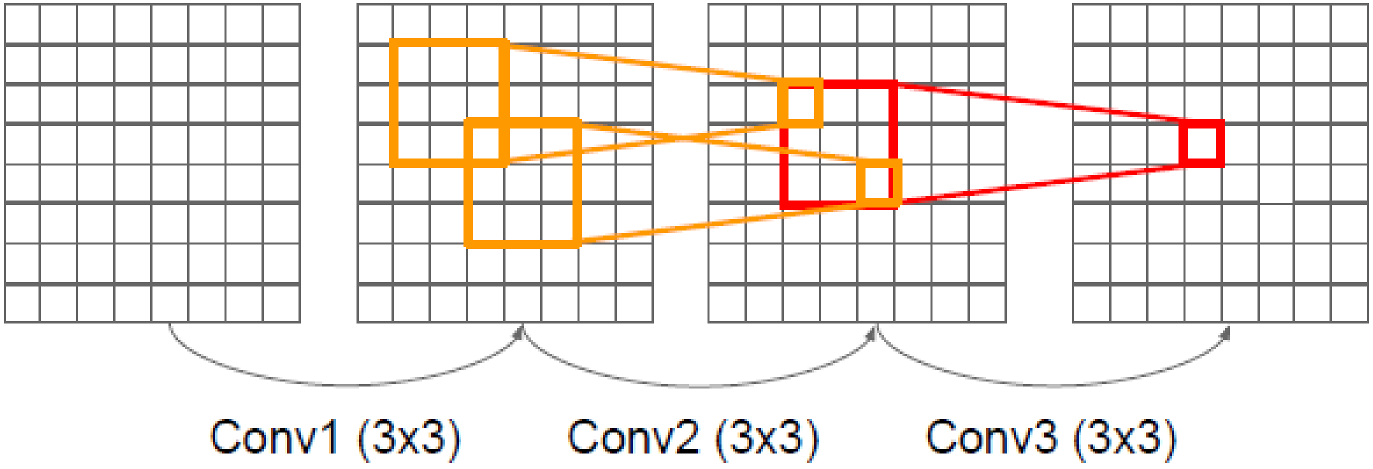
VGG

- *“It is easy to see that a stack of two 3×3 conv. layers has an effective receptive field of 5×5; three such layers have a 7 × 7 effective receptive field.”*



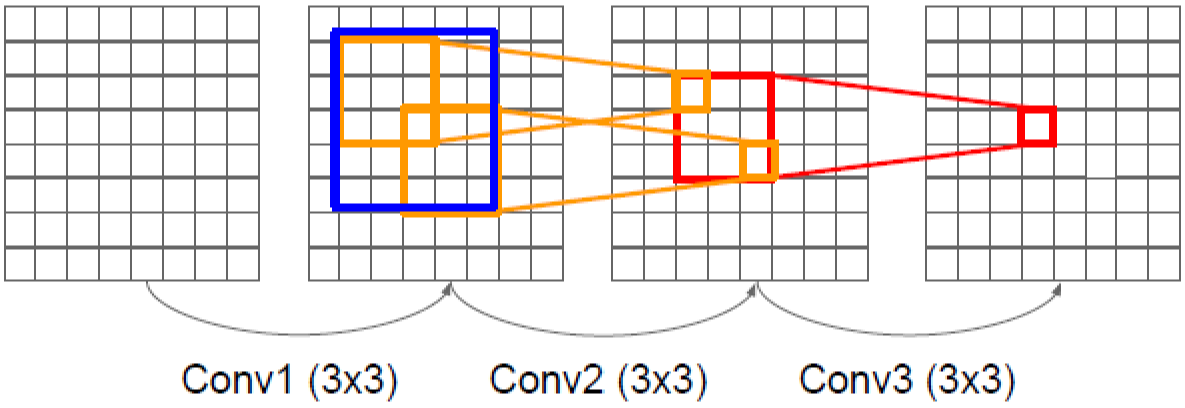
VGG

- *"It is easy to see that a stack of two 3×3 conv. layers has an effective receptive field of 5×5; three such layers have a 7 × 7 effective receptive field."*



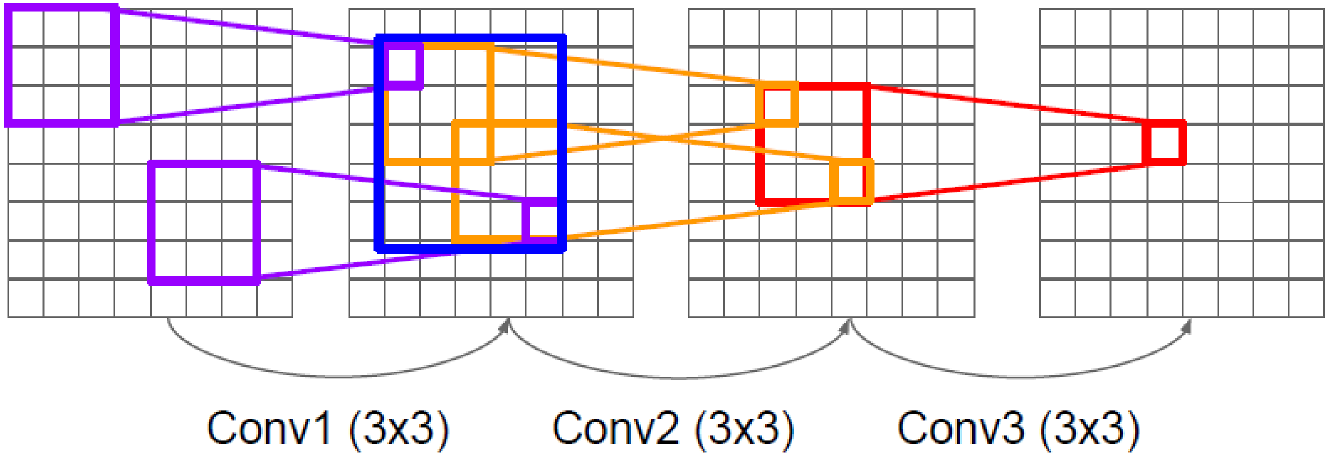
VGG

- *"It is easy to see that a stack of two 3×3 conv. layers has an effective receptive field of 5×5; three such layers have a 7 × 7 effective receptive field."*



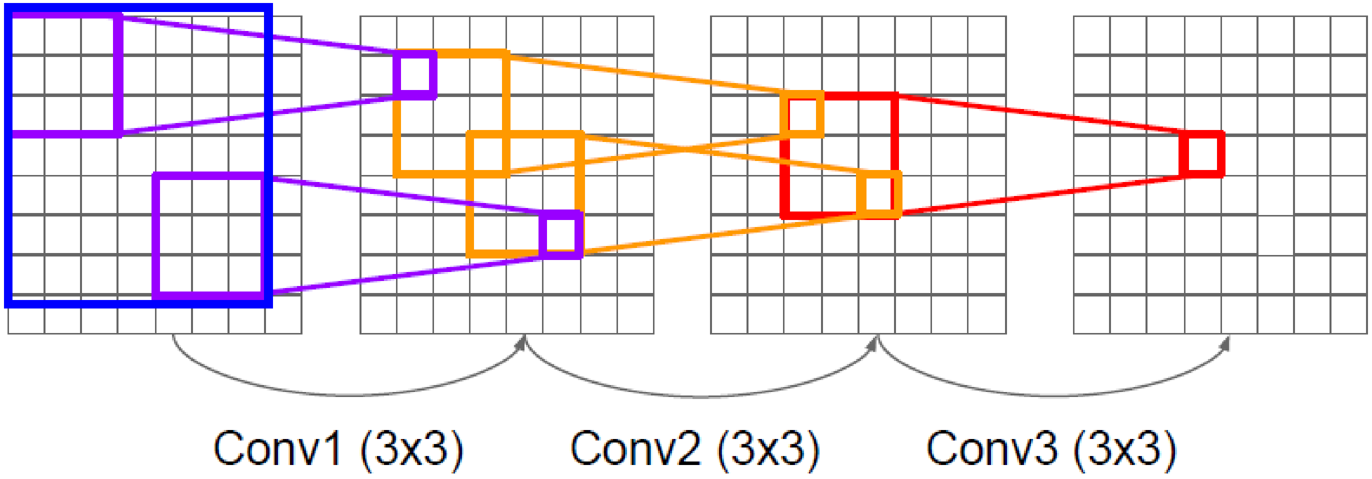
VGG

- *"It is easy to see that a stack of two 3×3 conv. layers has an effective receptive field of 5×5; three such layers have a 7 × 7 effective receptive field."*



VGG

- *"It is easy to see that a stack of two 3×3 conv. layers has an effective receptive field of 5×5; three such layers have a 7 × 7 effective receptive field."*



VGG

- *“It is easy to see that a stack of two 3×3 conv. layers has an effective receptive field of 5×5; three such layers have a 7 × 7 effective receptive field.”*
- What is the effect of this replacement in terms on the number of parameters?
- What about non-linearities?

VGG

- *“It is easy to see that a stack of two 3×3 conv. layers has an effective receptive field of 5×5; three such layers have a 7 × 7 effective receptive field.”*
- What is the effect of this replacement in terms on the number of parameters?

$3*(3^2C) = 27C$ vs $7^2C = 49C$

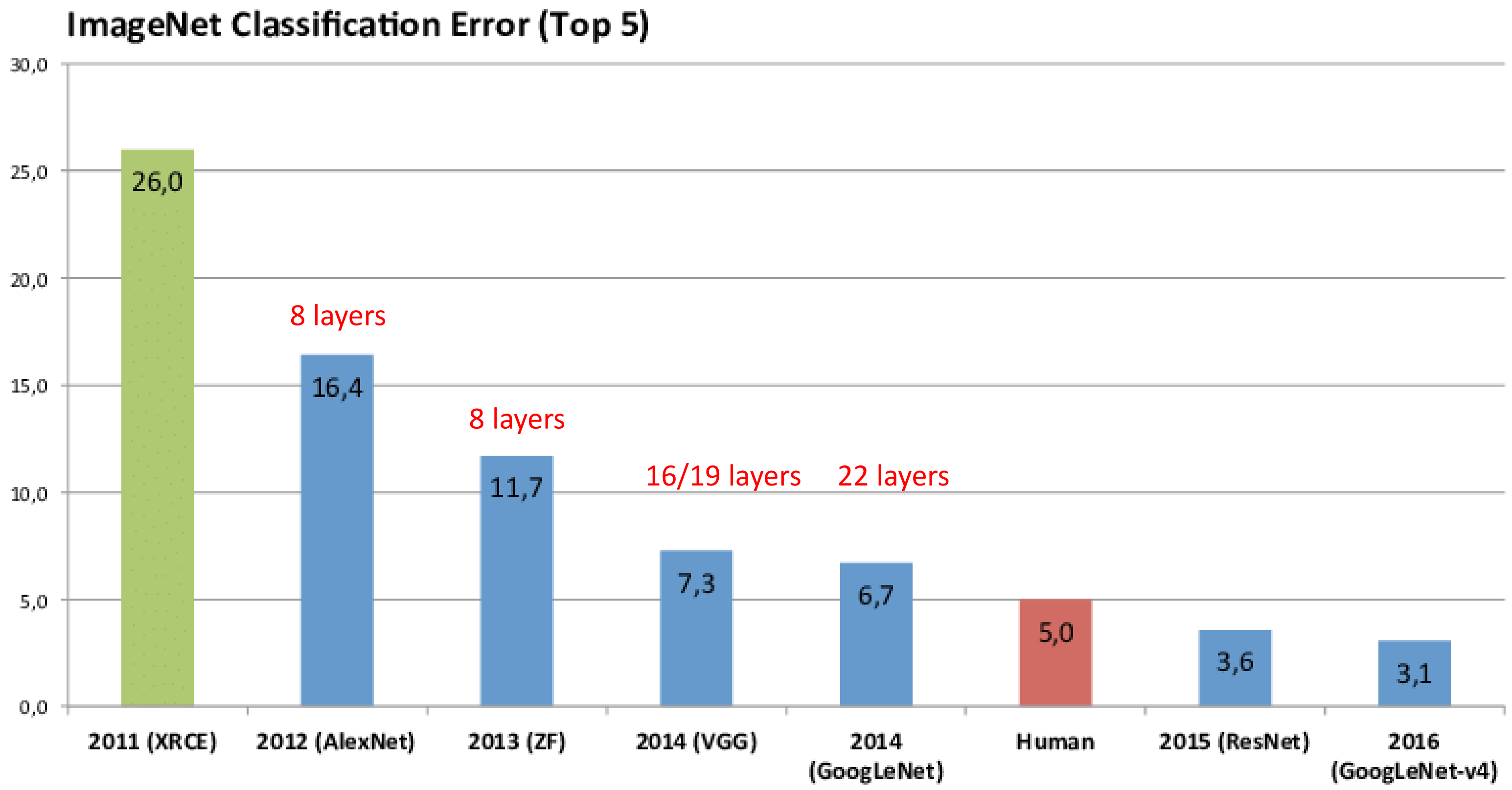
- What about non-linearities?

VGG – key features

- Did not use local response normalization layers
- Use ensembles to boost performance
- Use VGG 16 or VGG 19 (VGG 19 brings only a small increase in performance, but it requires more memory)
- Similar training procedure as in AlexNet

WE NEED TO GO
DEEPER

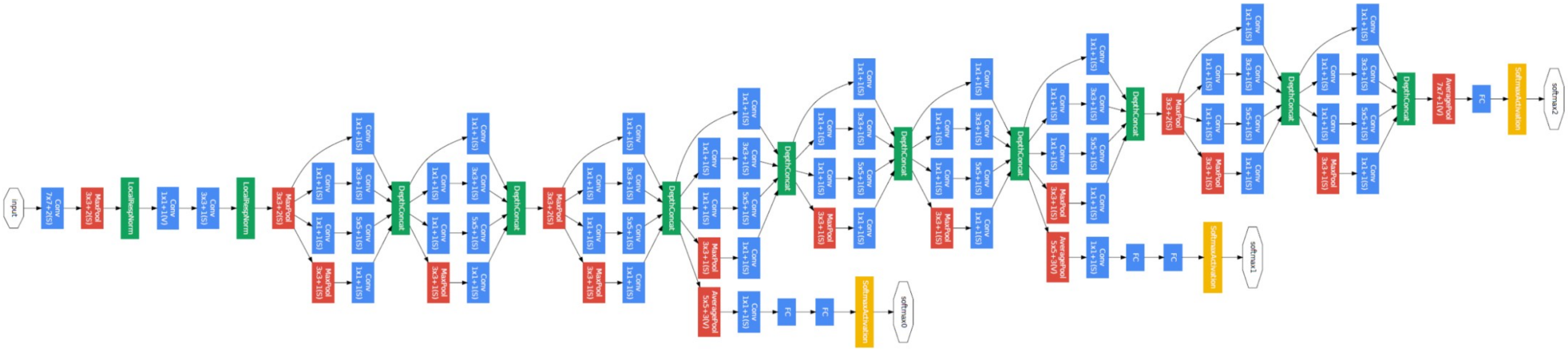
GoogLeNet, 2014



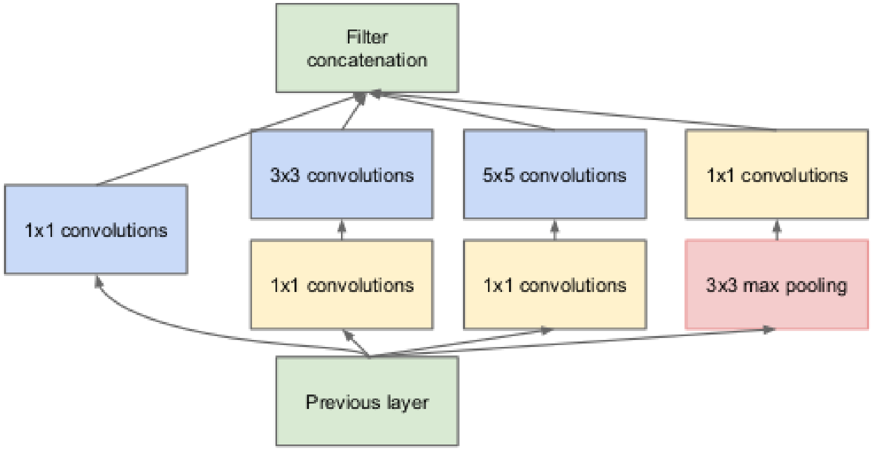
6.7 top-5 accuracy

GoogLeNet

This name is an homage to Yann LeCun's pioneering LeNet 5 network



Inception module



Filter concatenation

1x1 convolutions

3x3 convolutions

5x5 convolutions

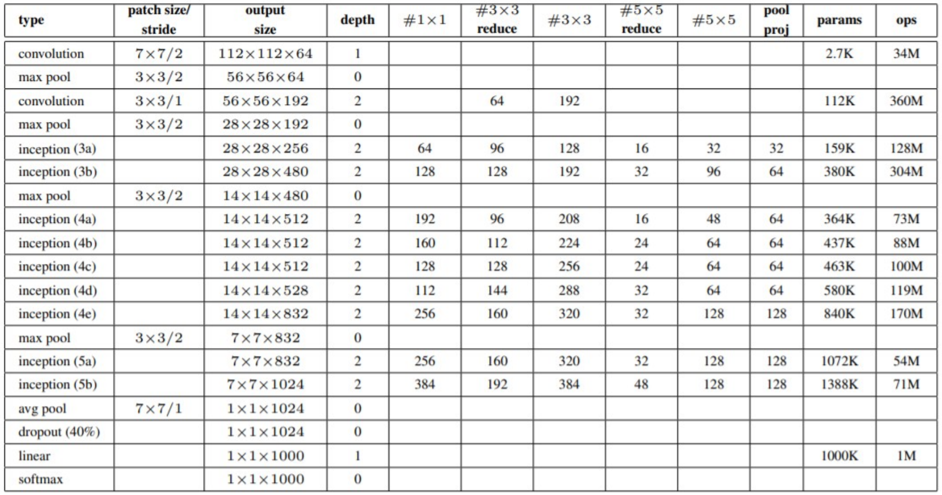
1x1 convolutions

1x1 convolutions

1x1 convolutions

3x3 max pooling

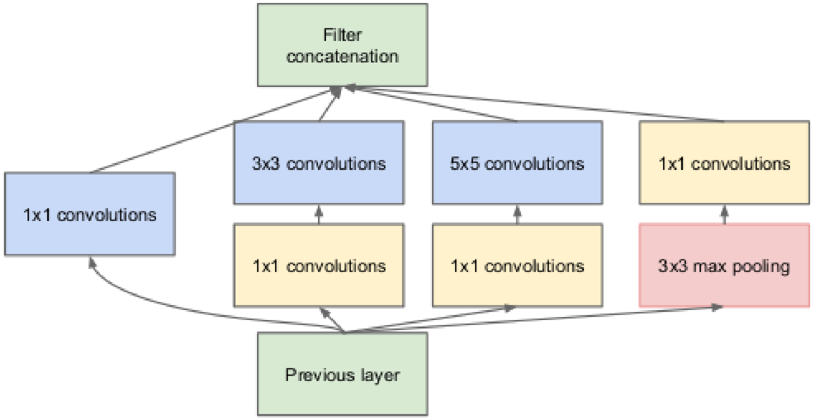
Previous layer



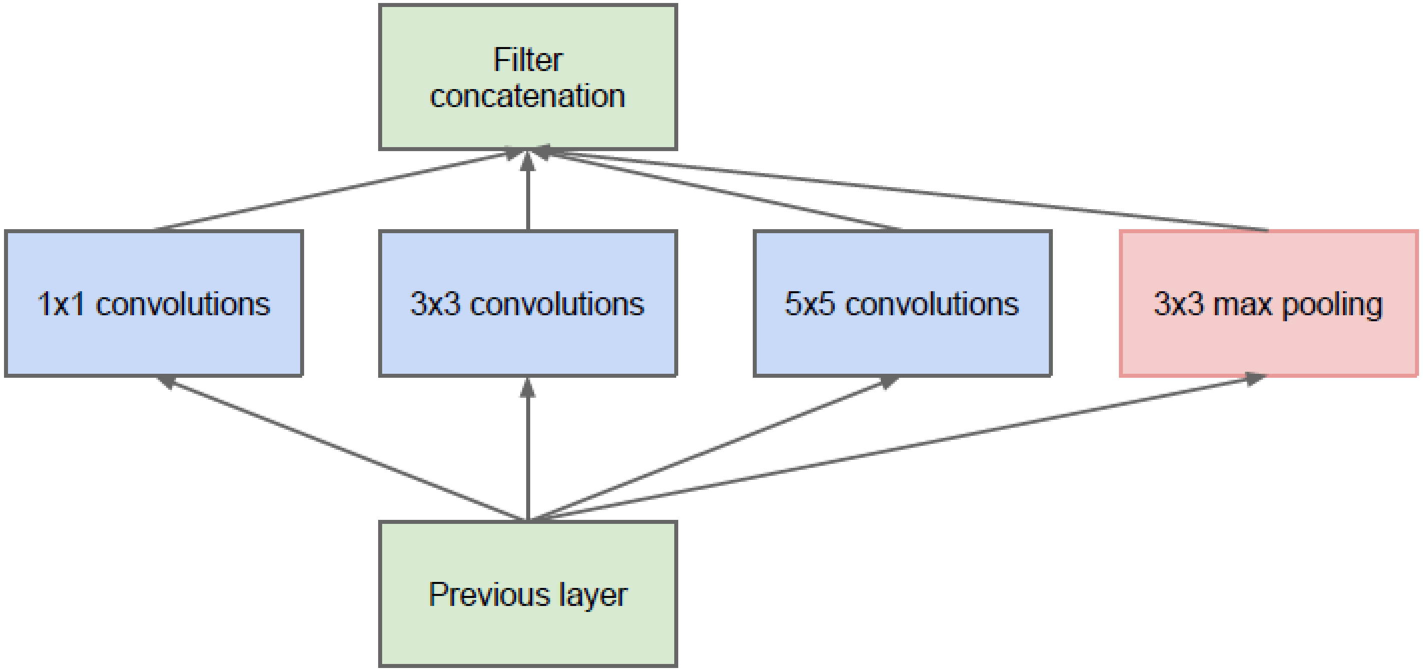
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96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

Inception module

- You don’t need to ”pick” the filter sizes, instead you let the network “choose” between several values
  - Filters of different sizes
  - Pooling layer
  - Concatenate activations depth-wise



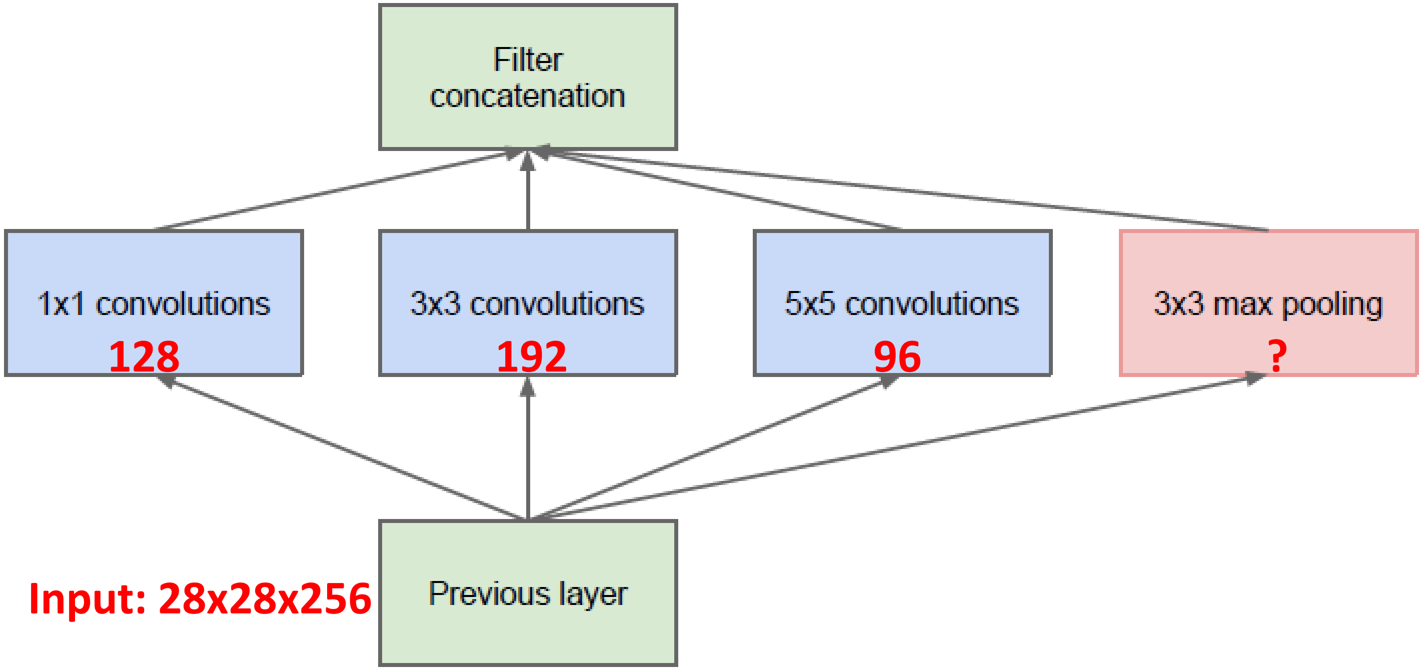
Inception module



(a) Inception module, naïve version

Inception module

What are the output sizes of all these operations?



Filter concatenation

1x1 convolutions 128

3x3 convolutions 192

5x5 convolutions 96

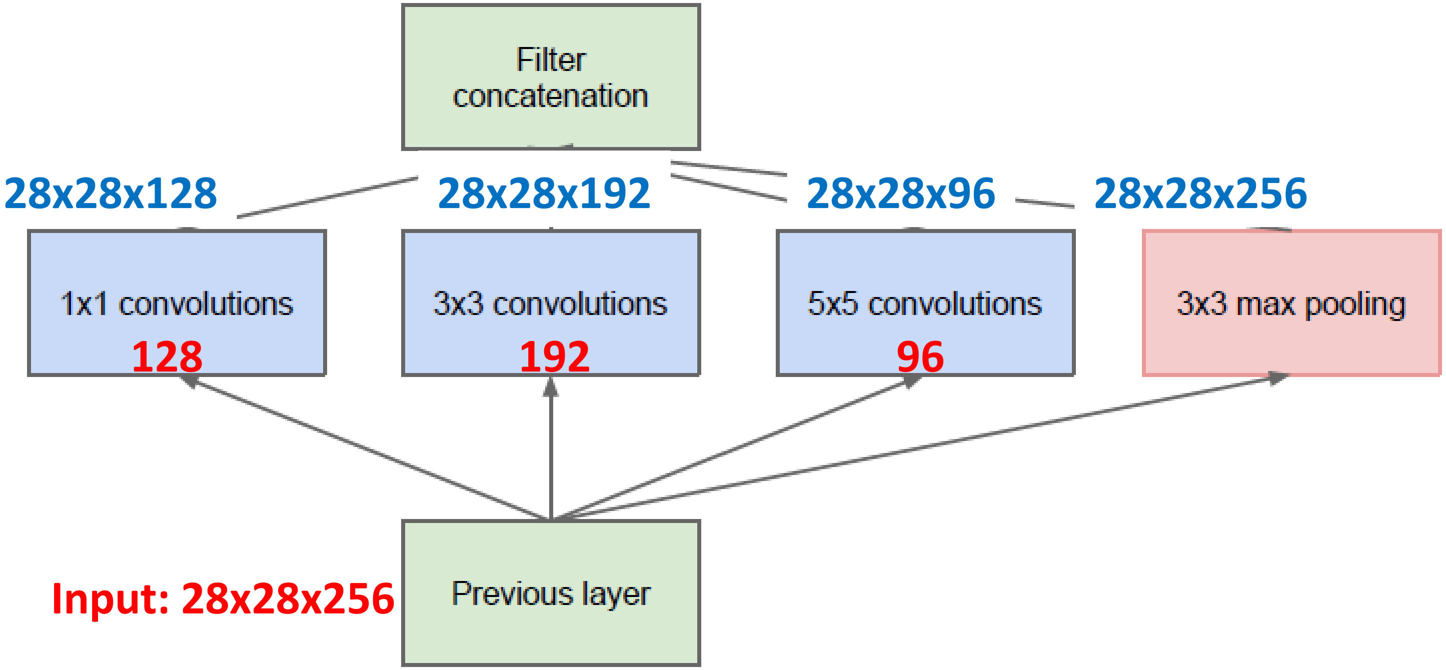
3x3 max pooling ?

Input: 28x28x256

Previous layer

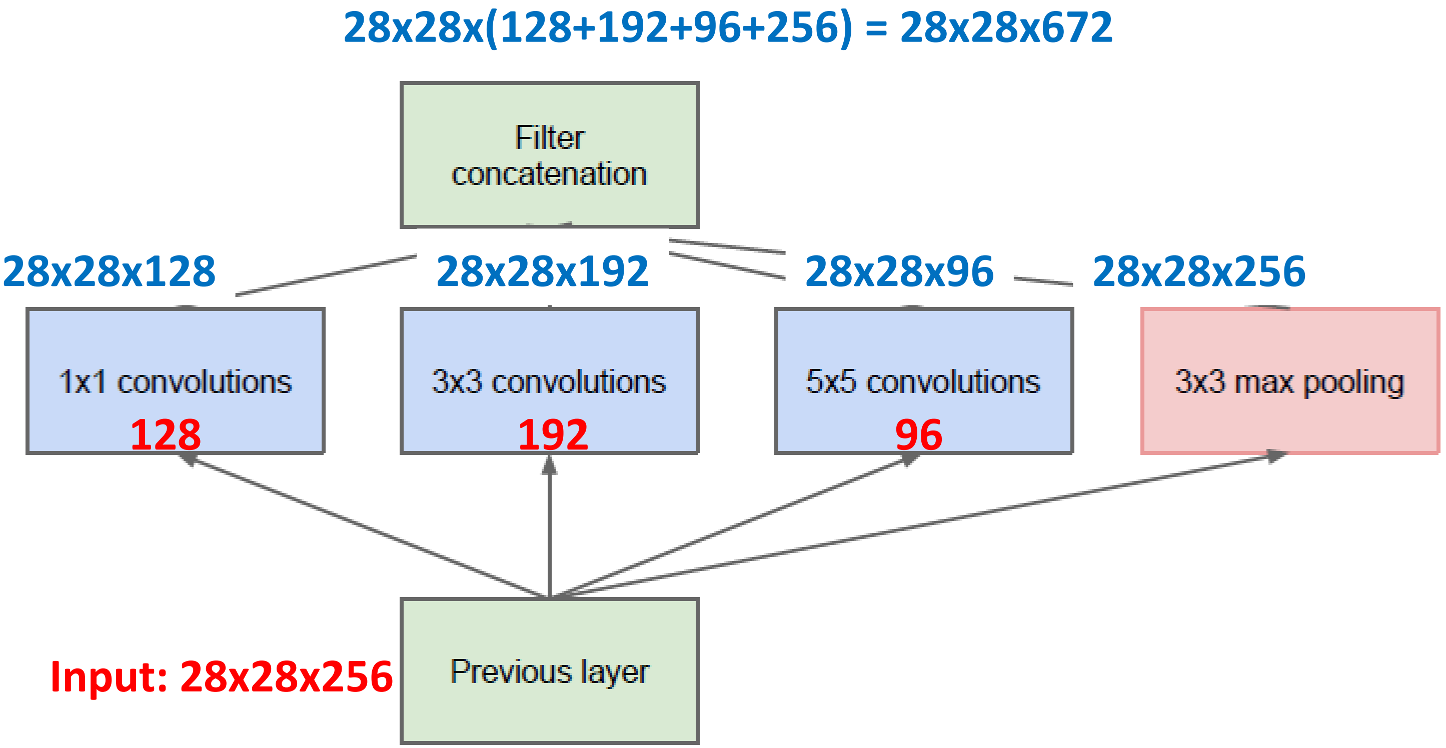
(a) Inception module, naïve version

Inception module



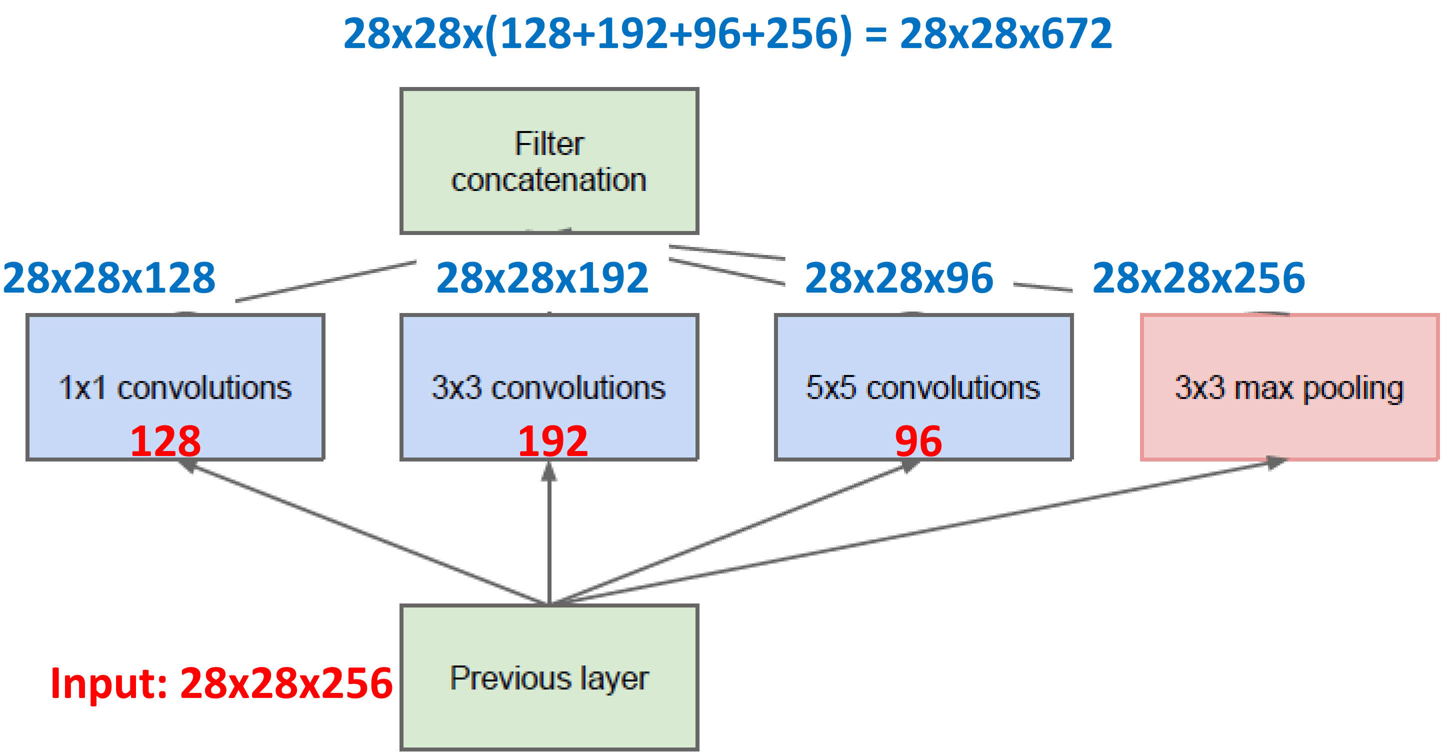
(a) Inception module, naïve version

Inception module



(a) Inception module, naïve version

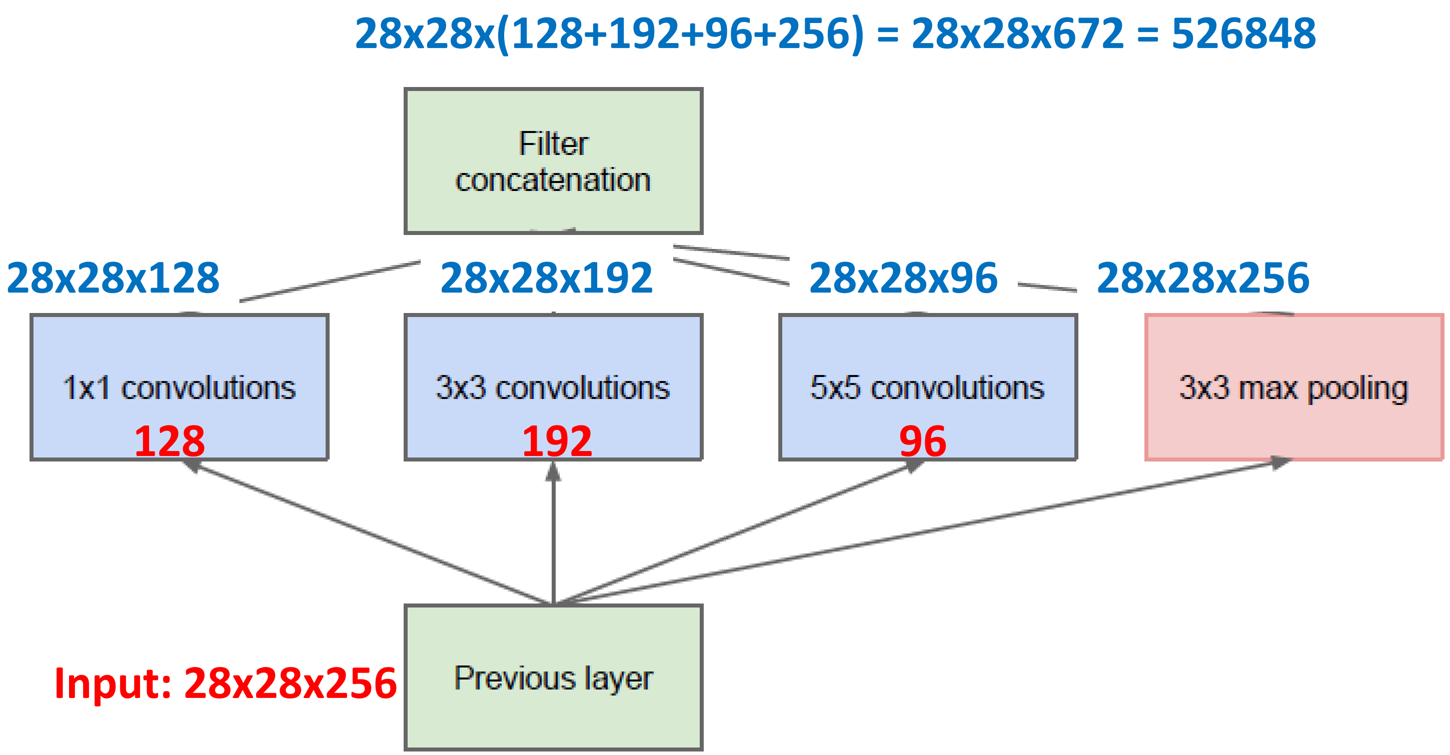
Inception module



(a) Inception module, naïve version

[1x1 conv, 128 filters]: ? convolution operations
[3x3 conv, 192 filters]: ? convolution operations
[5x5 conv, 96 filters]: ? convolution operations

Inception module



28x28x(128+192+96+256) = 28x28x672 = 526848

Filter concatenation

28x28x128

28x28x192

28x28x96

28x28x256

1x1 convolutions 128

3x3 convolutions 192

5x5 convolutions 96

3x3 max pooling

Input: 28x28x256

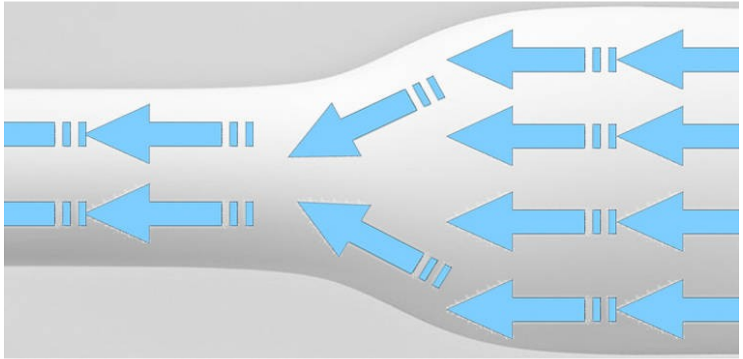
Previous layer

(a) Inception module, naïve version

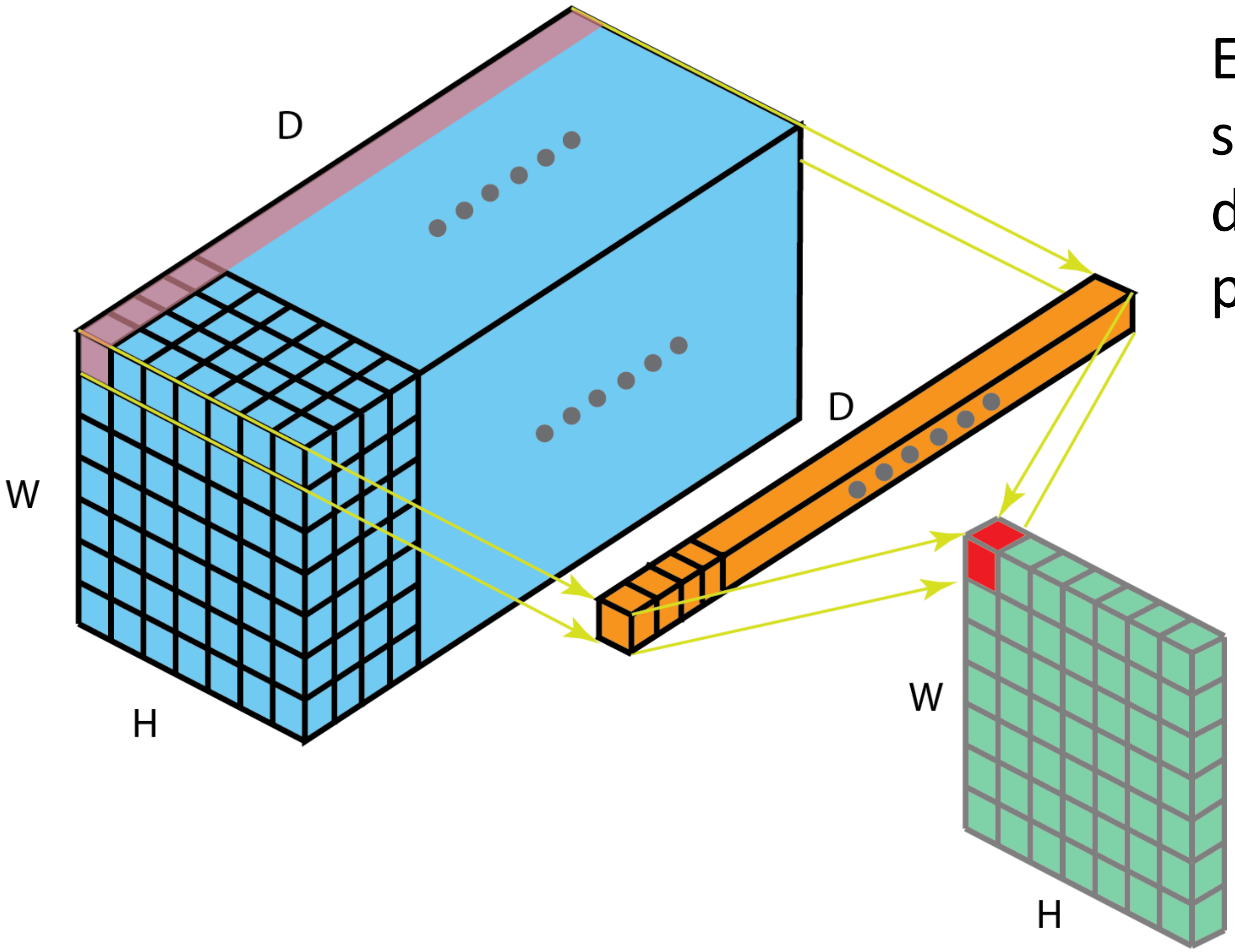
[1x1 conv, 128 filters]: 28x28x128x1x1x256
[3x3 conv, 192 filters]: 28x28x192x3x3x256 → **>850M**
[5x5 conv, 96 filters]: 28x28x96x5x5x256

Bottleneck

- Use *bottleneck* layers to reduce the depth dimensions of the activations
  - use 1x1 convolutions



1x1 convolutions



Each filter has a 1x1xD size and performs a D-dimensional dot product

1x1 convolutions

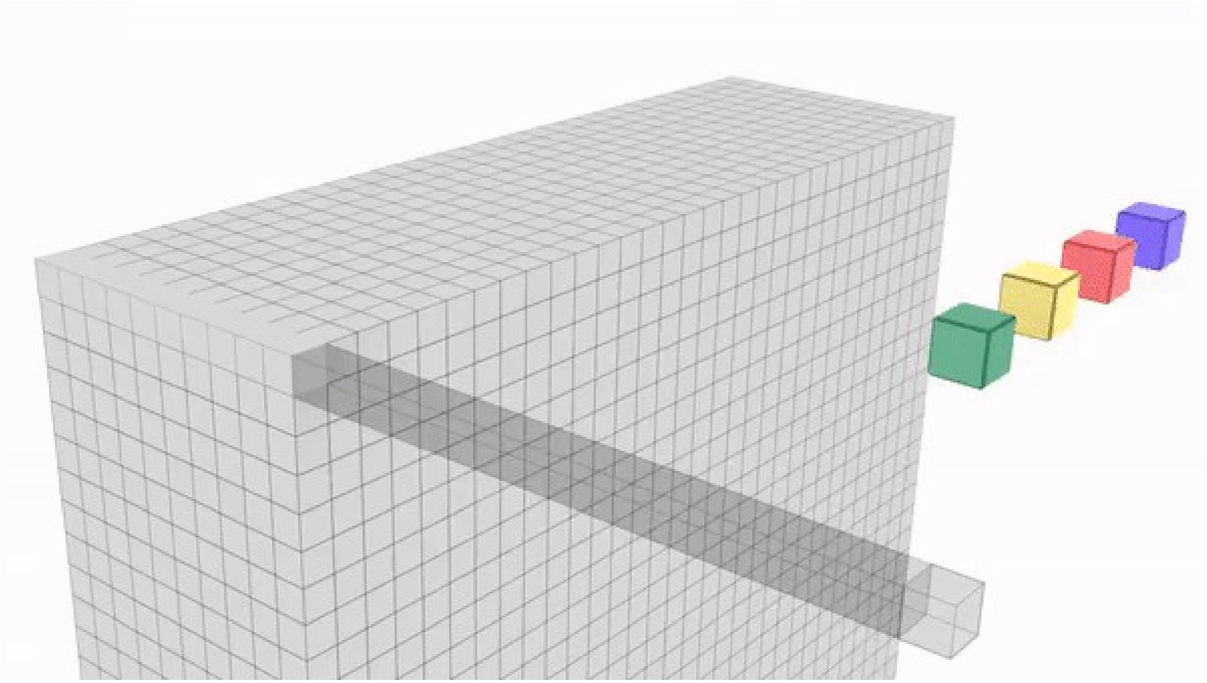
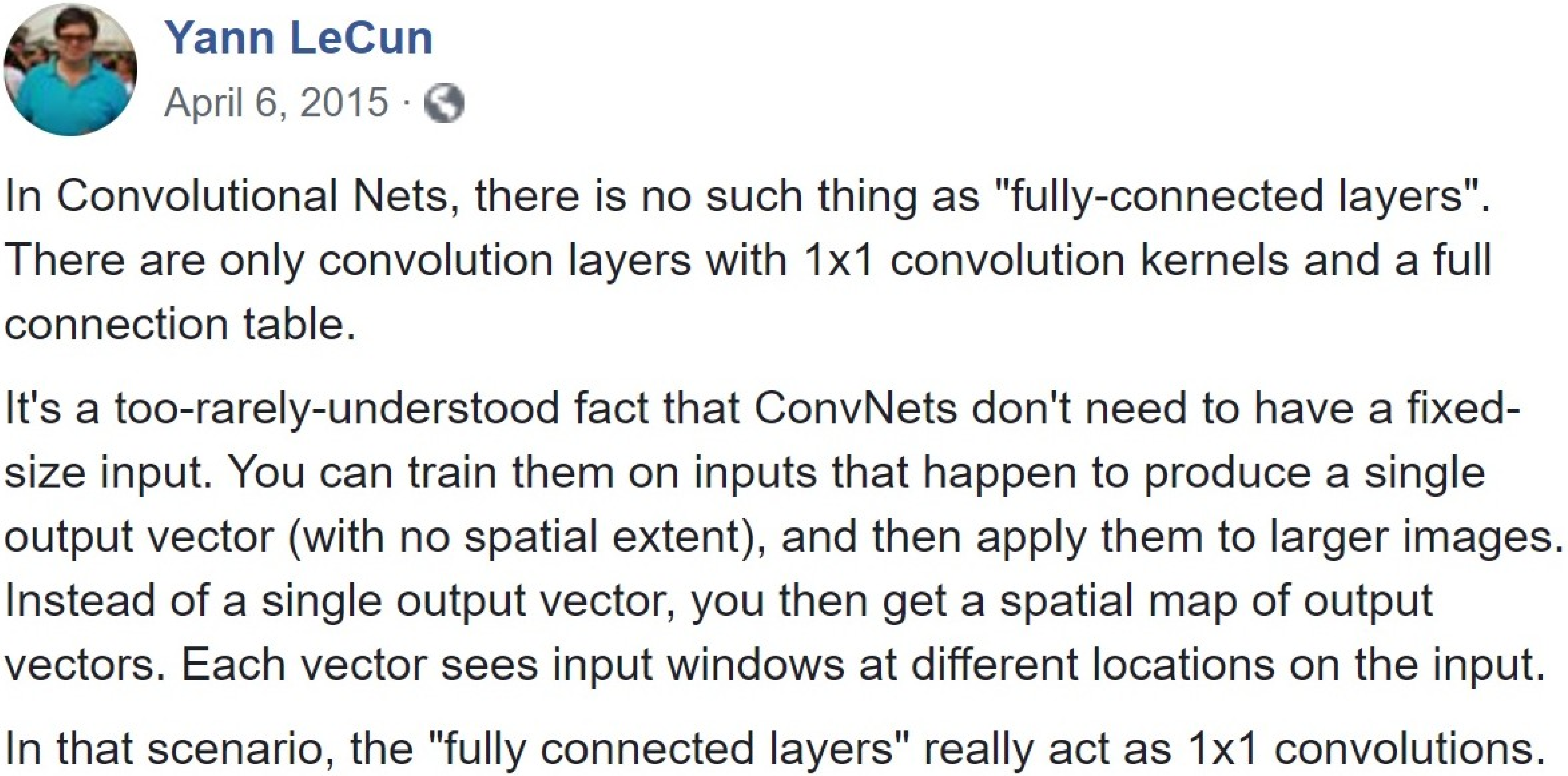


image source : https://medium.com/hitchhikers-guide-to-deep-learning/6-introduction-to-deep-learning-with-computer-vision-3x3-is-a-lie-1x1-convolutions-9edd2baf7fd5

1x1 convolution



Yann LeCun

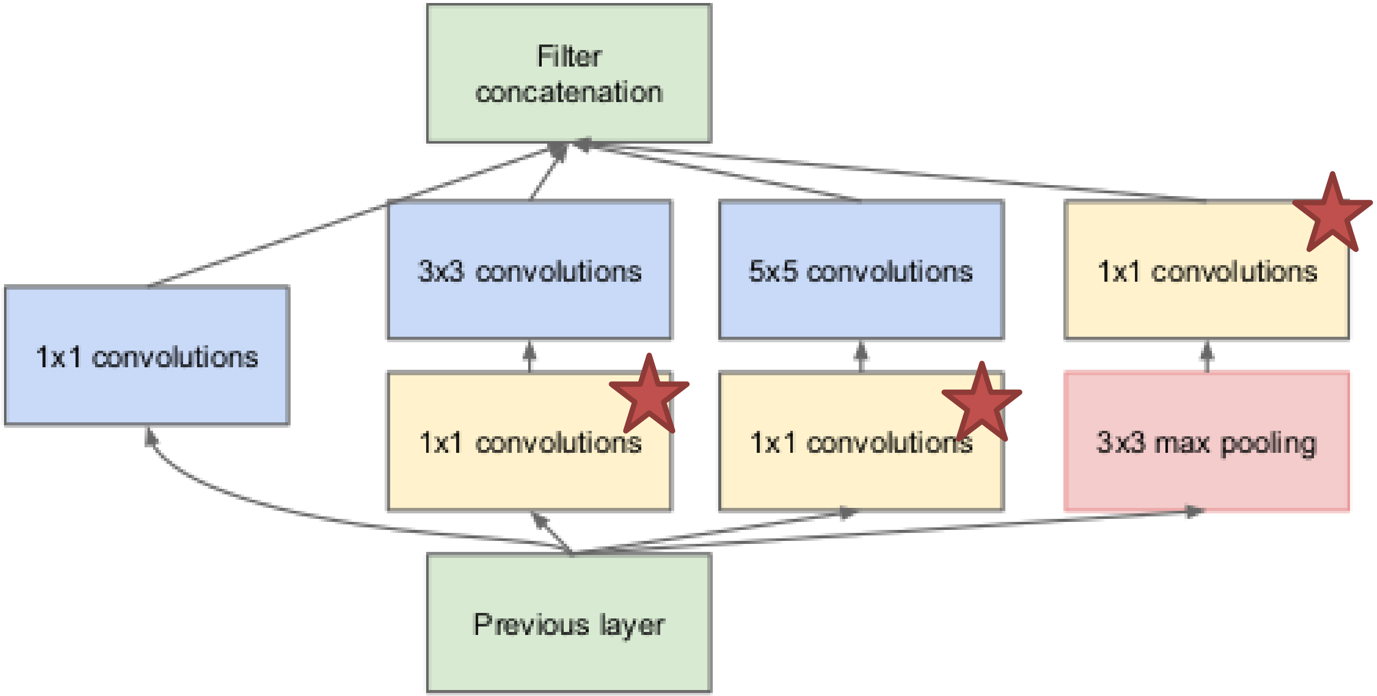
April 6, 2015 ·

In Convolutional Nets, there is no such thing as "fully-connected layers". There are only convolution layers with 1x1 convolution kernels and a full connection table.

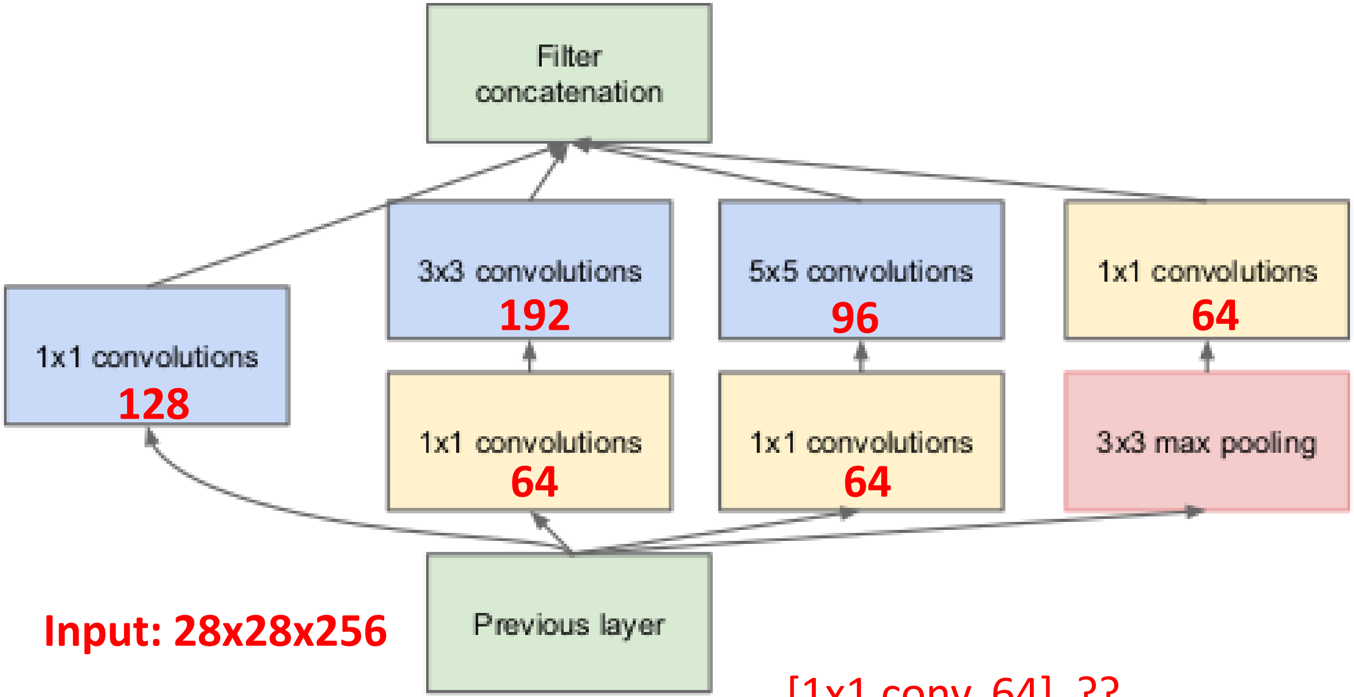
It's a too-rarely-understood fact that ConvNets don't need to have a fixed-size input. You can train them on inputs that happen to produce a single output vector (with no spatial extent), and then apply them to larger images. Instead of a single output vector, you then get a spatial map of output vectors. Each vector sees input windows at different locations on the input.

In that scenario, the "fully connected layers" really act as 1x1 convolutions.

The real Inception module



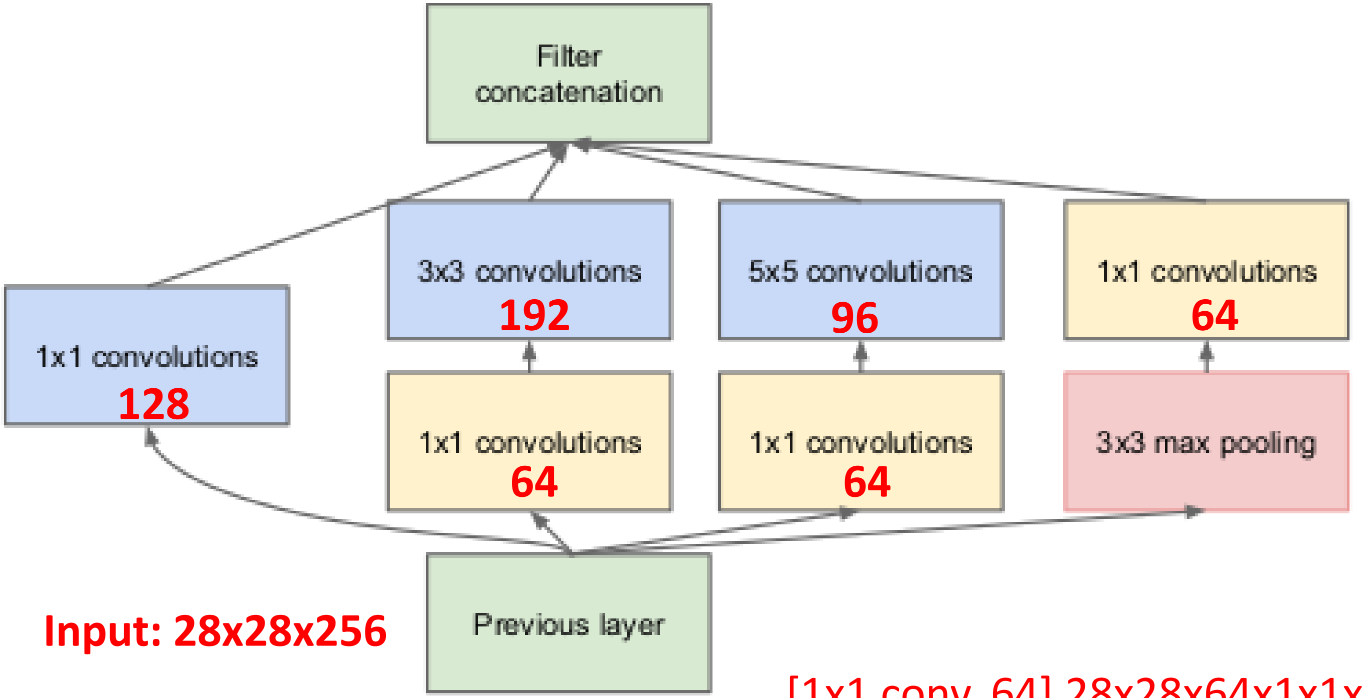
The real Inception module



Input: 28x28x256

[1x1 conv, 64] ??
[1x1 conv, 64] ??
[1x1 conv, 128] ??
[3x3 conv, 192] ??
[5x5 conv, 96]   ??
[1x1 conv, 64]   ??

The real Inception module



Input: 28x28x256

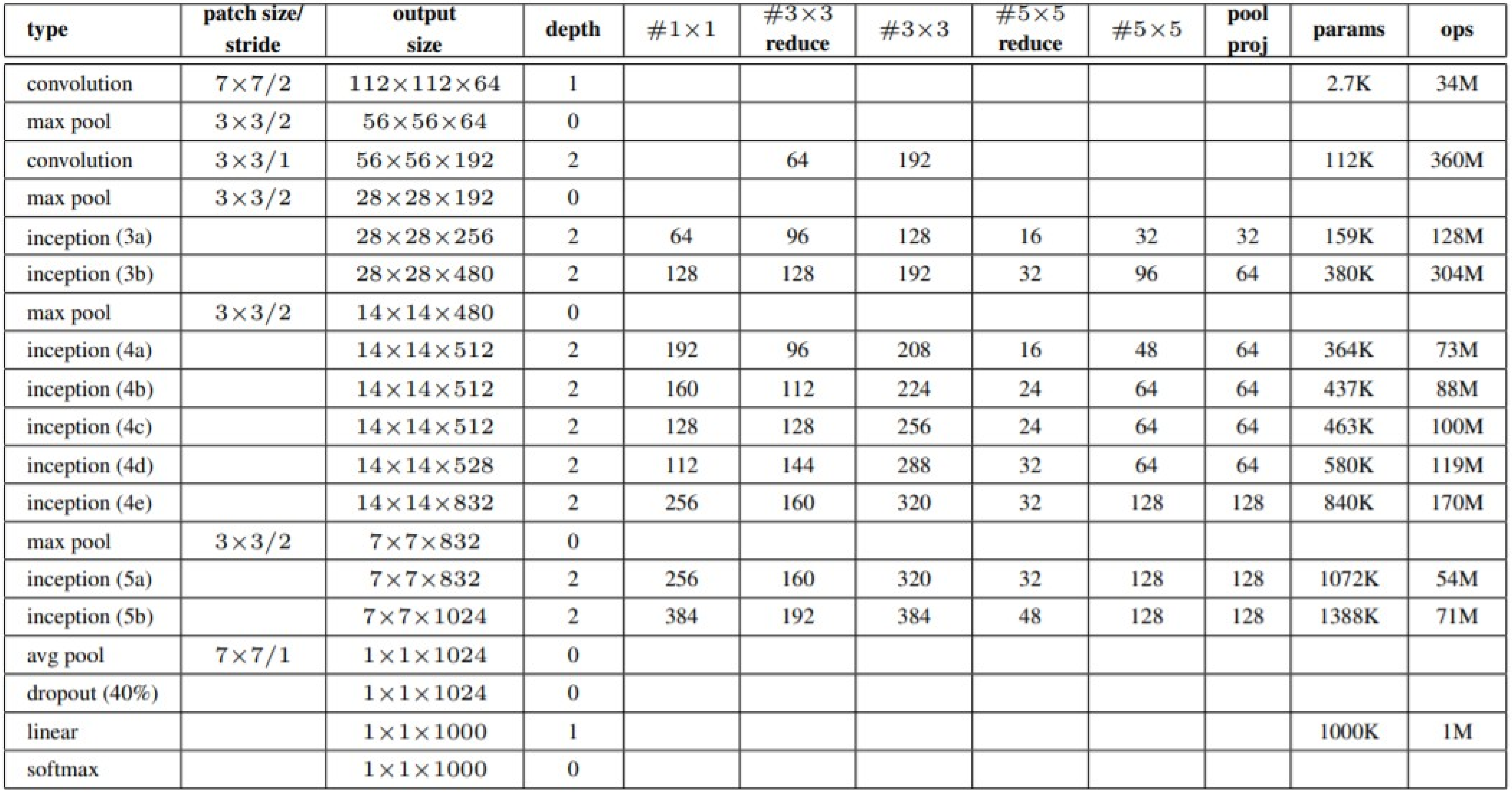
~~>850M~~

358M ←

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

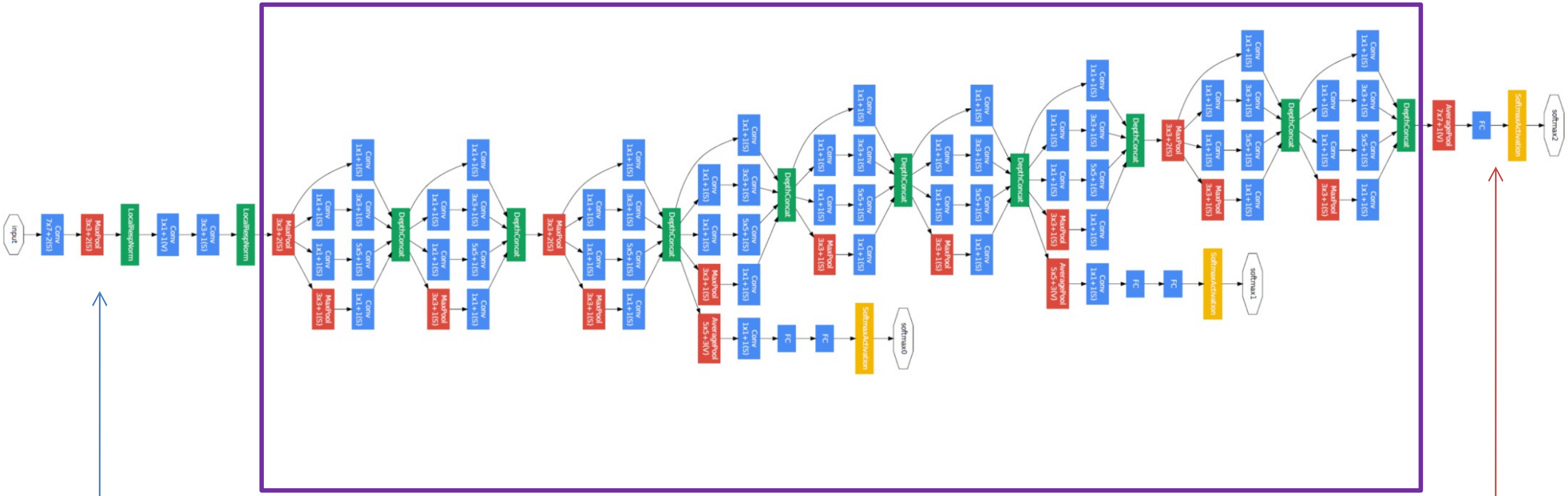
GoogLe Net

Stack Inception modules on top of each other



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

GoogLe Net

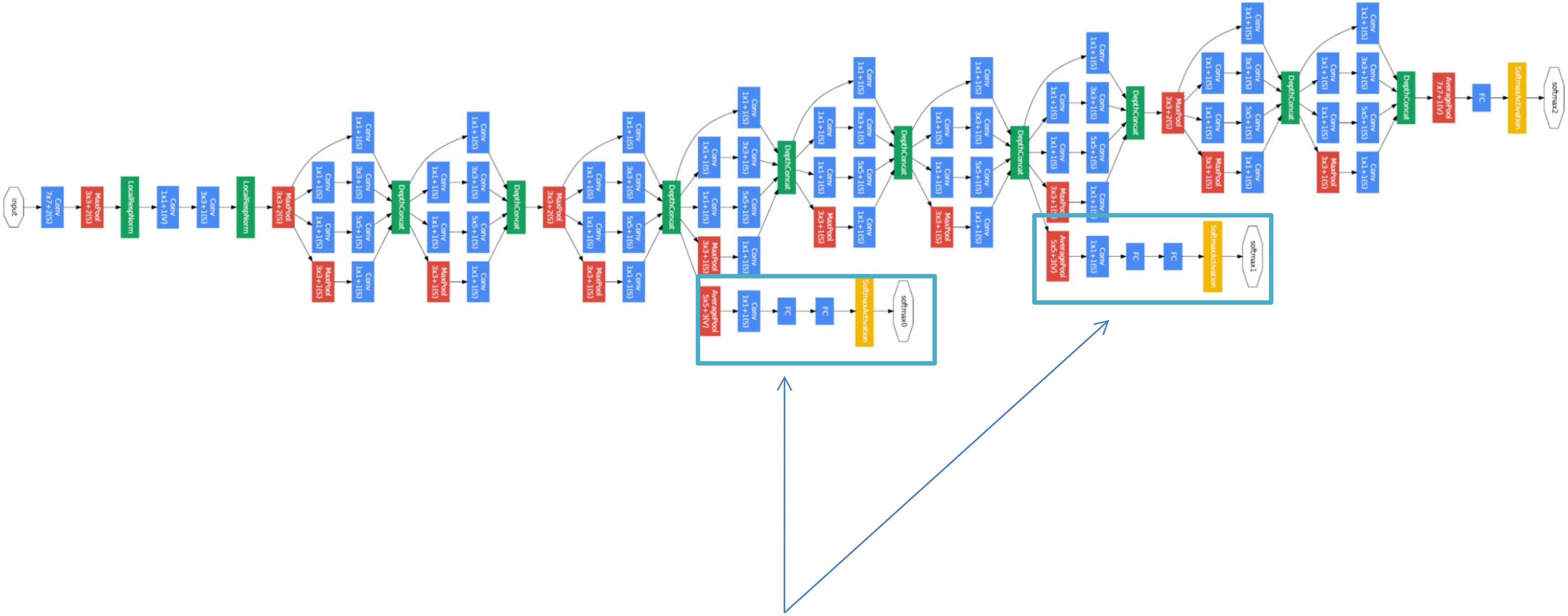


Inception modules

Stem network:
Conv – Pool – 2x Conv-Pool

Classifier output removes FC layers

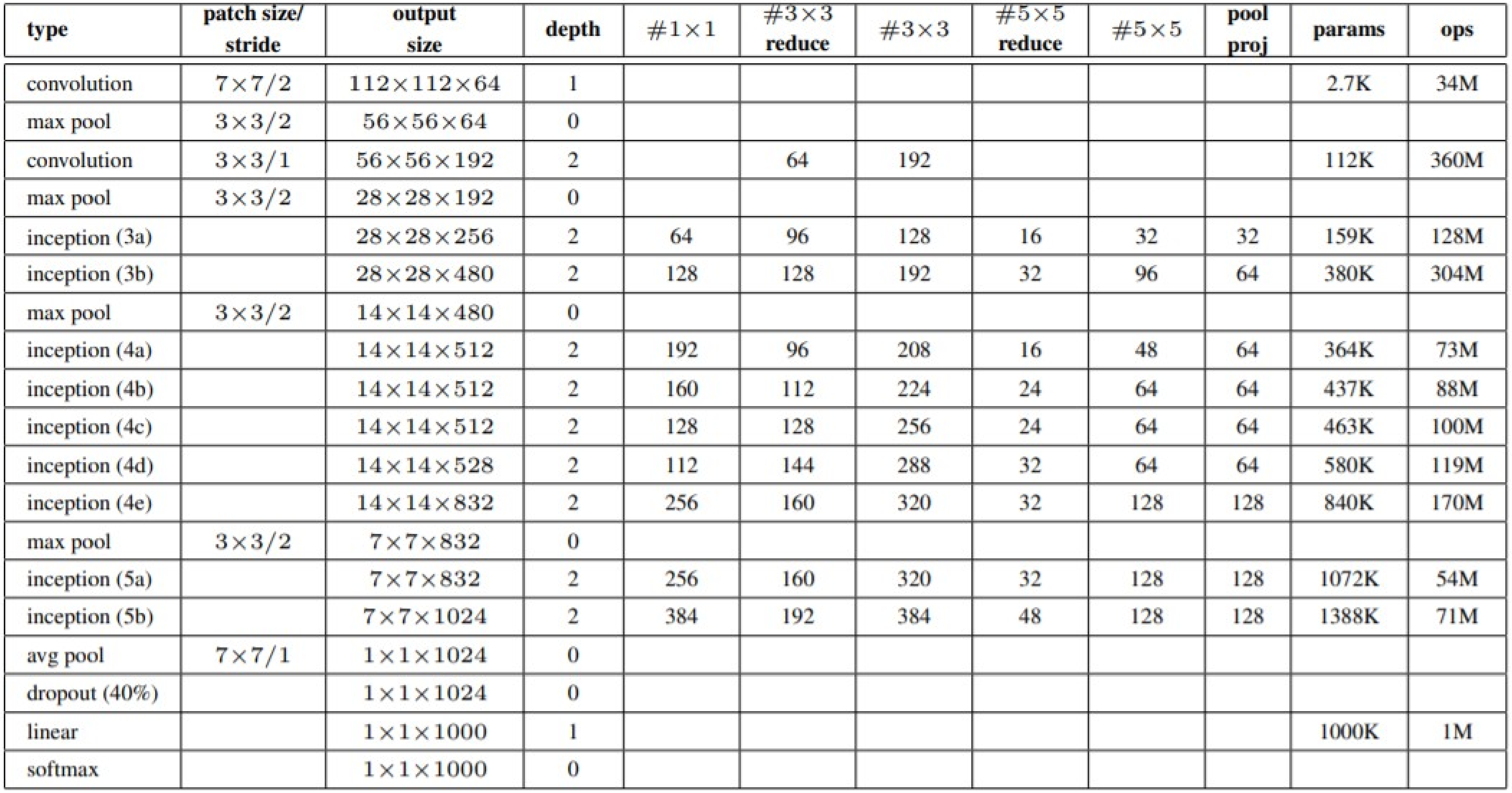
GoogLe Net



Output layers to inject additional gradient at lower layers
AvgPool – 1x1 Conv – FC – FC - Softmax

GoogLe Net

Stack Inception modules on top of each other



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

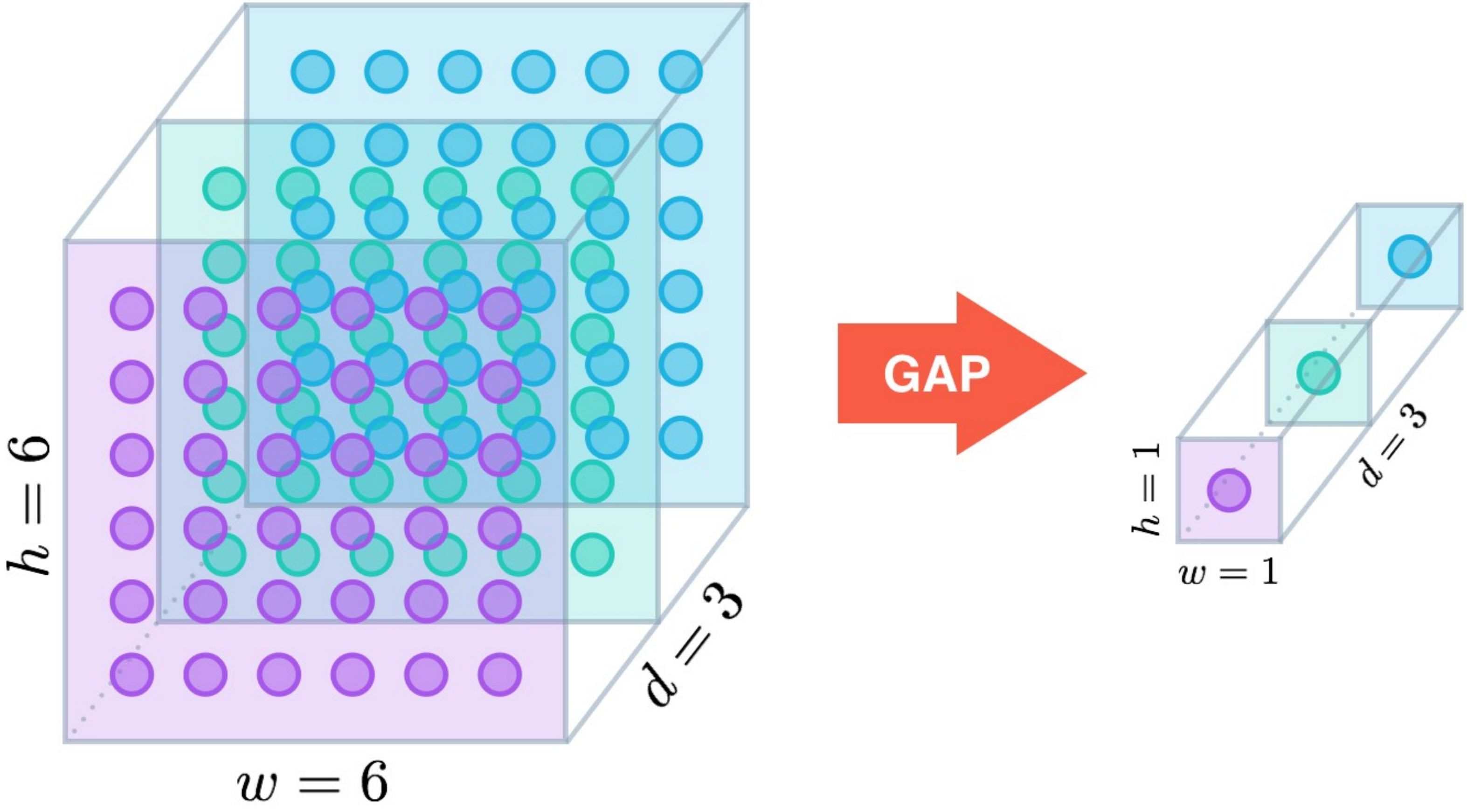
GoogLe Net

- Removes fully connected layers
- *It was found that a move from fully connected layers to average pooling improved the top-1 accuracy by about 0.6%.*

Global average pooling
Network In Network, 2014

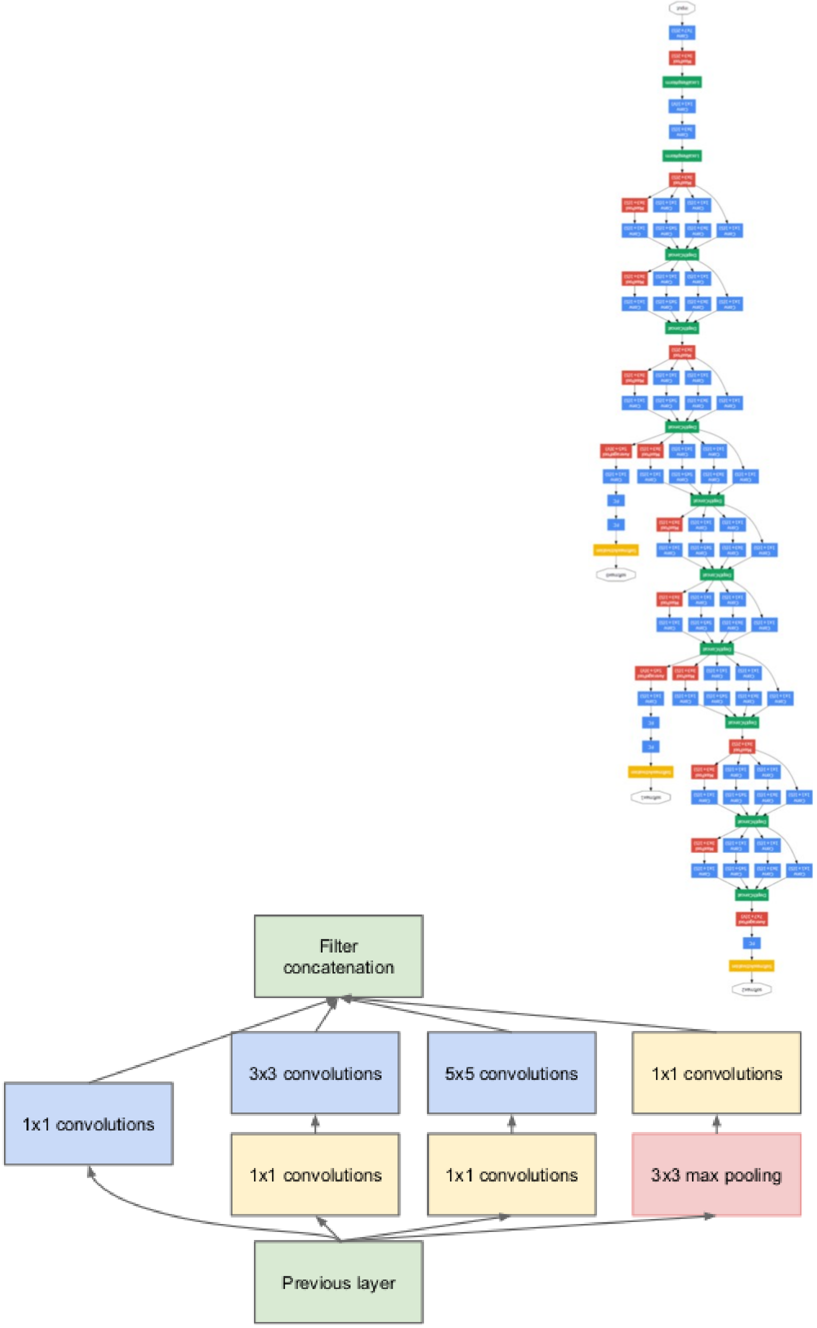
- Instead of adding fully connected layers on top of the feature maps, take **the average** of each feature map, and the resulting vector is fed directly into the softmax layer.
  - is more native to the convolution structure by enforcing correspondences between feature maps and categories (feature maps can be easily interpreted as categories confidence maps)
  - no parameter to optimize in the global average pooling thus overfitting is avoided at this layer
  - sums up spatial information -> more robust to spatial translations of the input

Global average pooling (GAP)

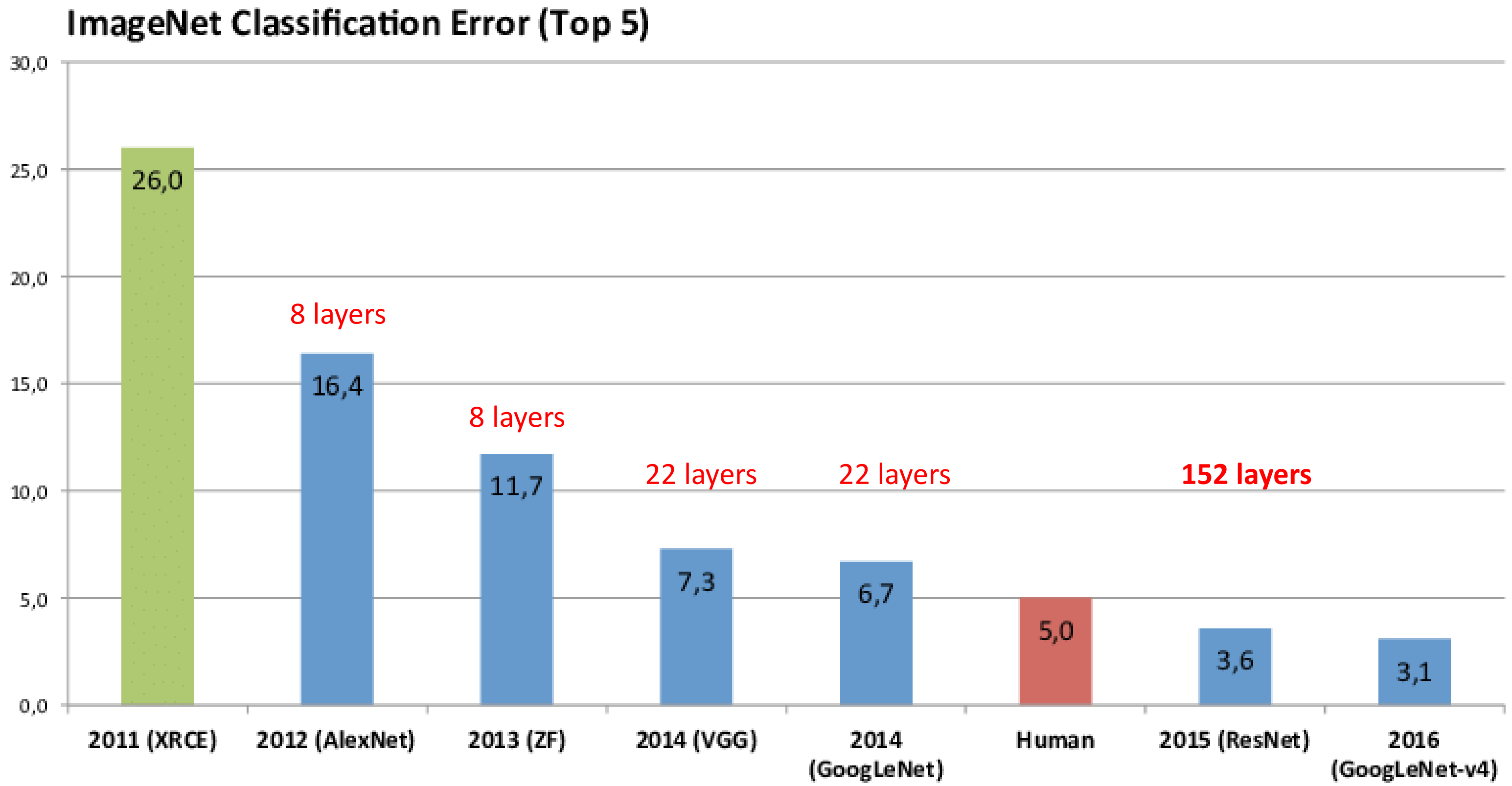


GoogLe Net

- 22 layers
- Inception module
- 12x less parameters than AlexNet
- **Top 5 accuracy: 6.7%**
- Removes fully connected layers as in NIN
  - – implementation differs in that we use an extra linear layer



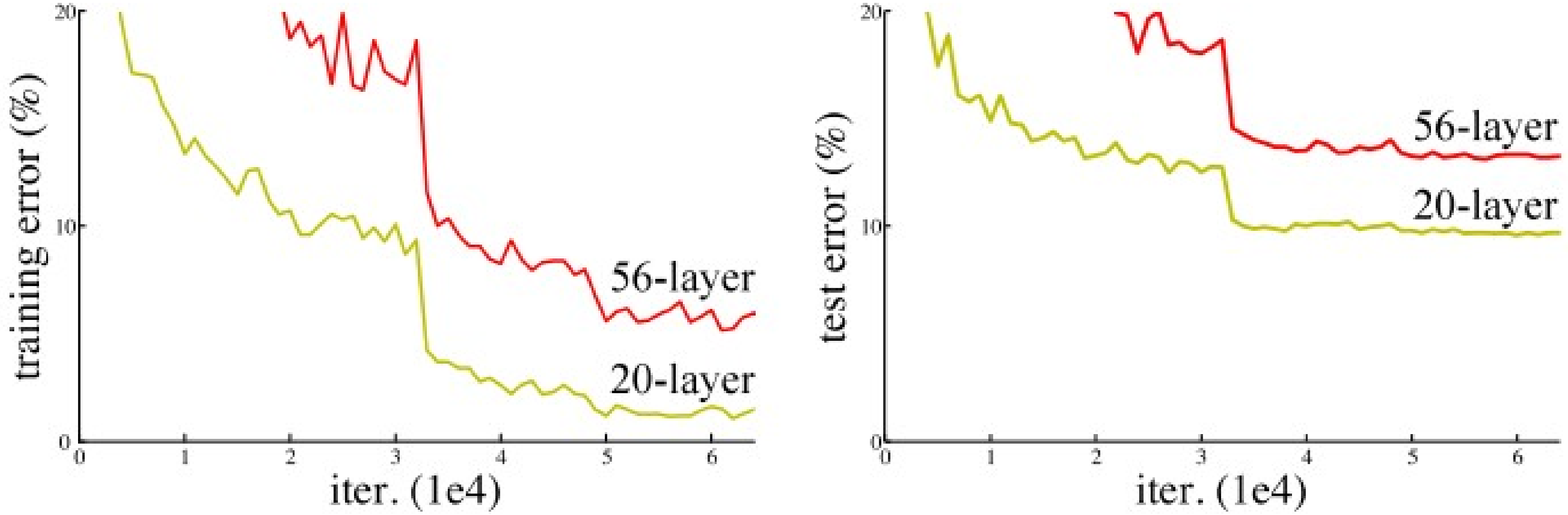
ResNet, 2015



6.7 top-5 accuracy

Res Net

Increasing a network depth

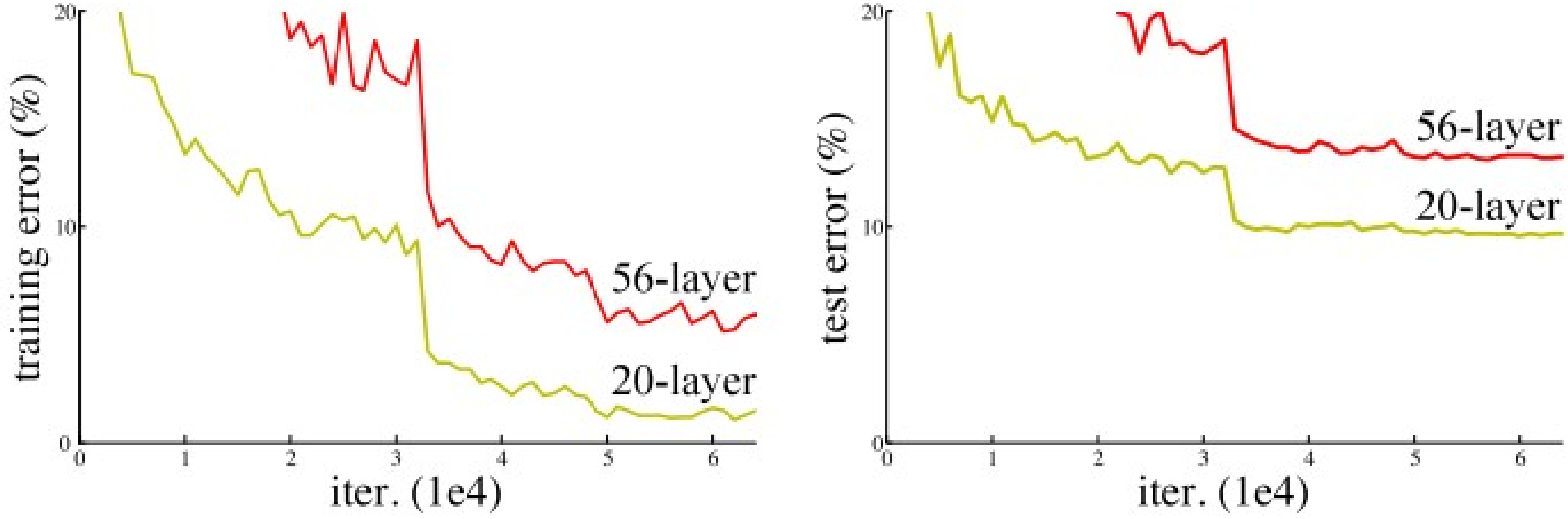


Overfitting?

Image source: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

Res Nets

Increasing a network depth



Overfitting? NO!

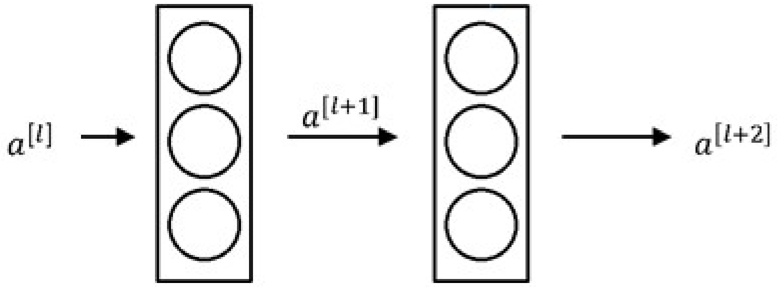
Hypothesis: Deeper models are harder to optimize

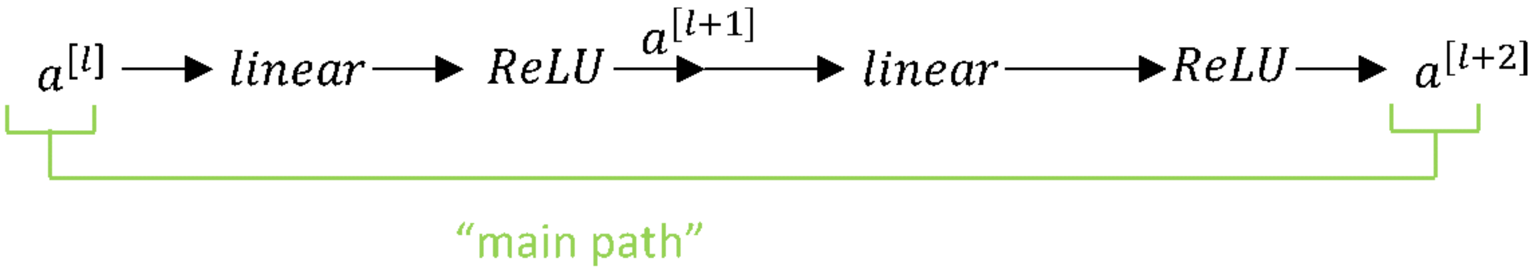
Image source: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

Res Nets

- Deeper models shouldn’t hurt performance, they should be at least as good as shallower ones
- Copy the learned layers from the shallower layer and set additional layers to identity mapping

Classical neural network block





$$a^{[l]} \longrightarrow linear \longrightarrow ReLU \xrightarrow{a^{[l+1]}} linear \longrightarrow ReLU \longrightarrow a^{[l+2]}$$

"main path"

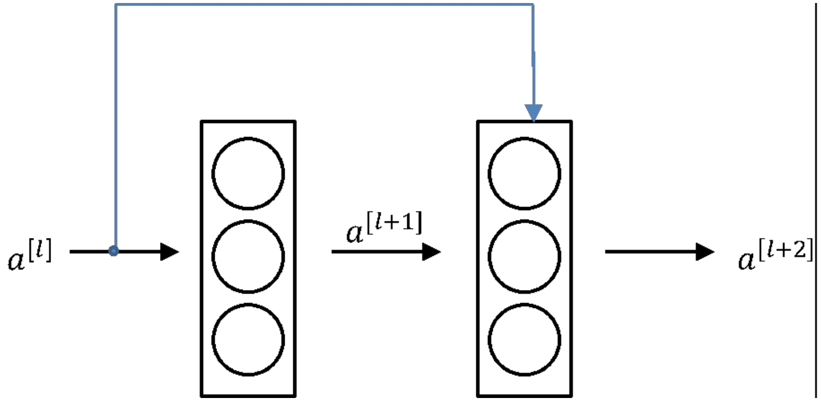
$$a^{[l]} \rightarrow linear \rightarrow ReLU \rightarrow a^{[l+1]} \rightarrow linear \rightarrow ReLU \rightarrow a^{[l+2]}$$

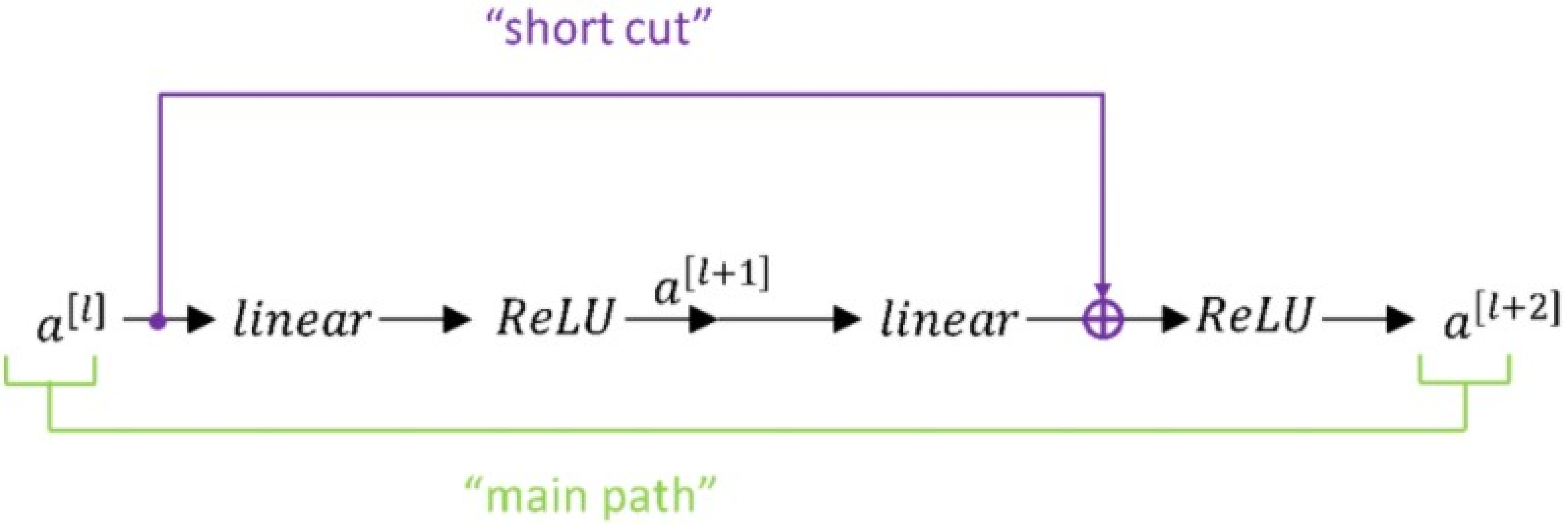
$$z^{[l+1]} = W^{[l+1]}a^{[l]} + b^{[l+1]} \qquad z^{[l+2]} = W^{[l+2]}a^{[l+1]} + b^{[l+2]}$$

$$a^{[l+1]} = g\left(z^{[l+1]}\right) \qquad a^{[l+2]} = g(z^{[l+2]})$$

Image source: http://datahacker.rs/deep-learning-residual-networks/
Example credit: Andrew Ng

Residual block





$$z^{[l+1]} = W^{[l+1]}a^{[l]} + b^{[l+1]}$$

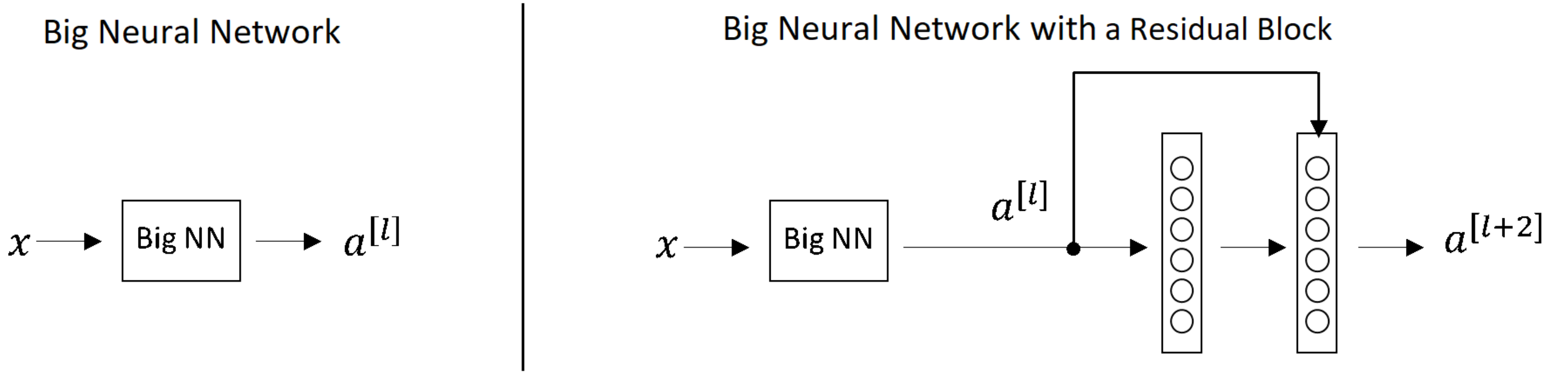
$$a^{[l+1]} = g\left(z^{[l+1]}\right)$$

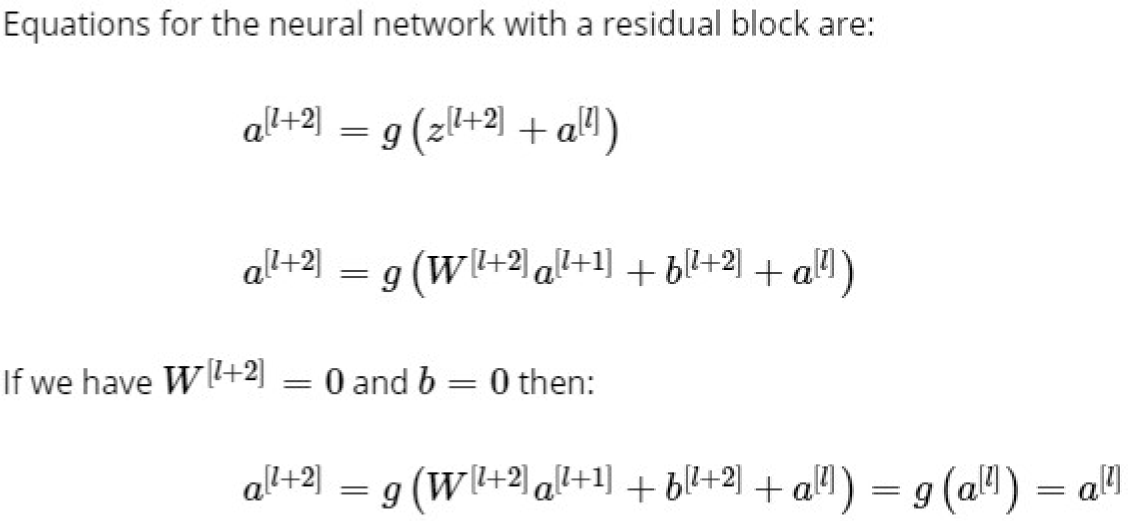
$$z^{[l+2]} = W^{[l+2]}a^{[l+1]} + b^{[l+2]}$$

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

Image source: http://datahacker.rs/deep-learning-residual-networks/

Residual block





Equations for the neural network with a residual block are:

$$a^{[l+2]} = g\left(z^{[l+2]} + a^{[l]}\right)$$

$$a^{[l+2]} = g\left(W^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]}\right)$$

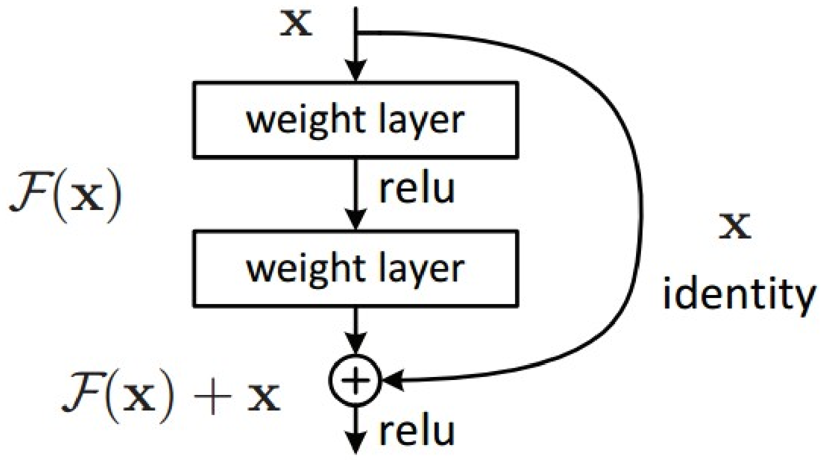
If we have $W^{[l+2]} = 0$ and $b = 0$ then:

$$a^{[l+2]} = g\left(W^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]}\right) = g\left(a^{[l]}\right) = a^{[l]}$$

Image source: http://datahacker.rs/deep-learning-residual-networks/

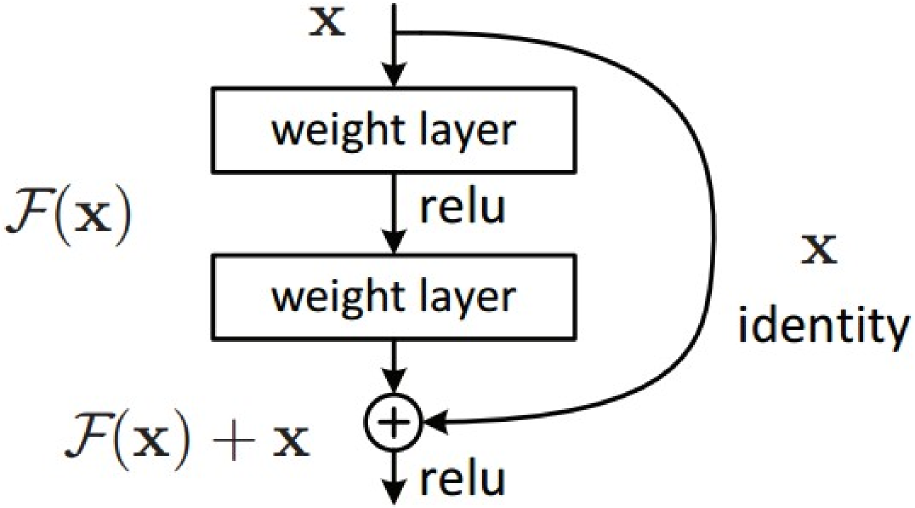
Res Net

- Try to fit a residual mapping instead of directly trying to fit a desired underlying mapping



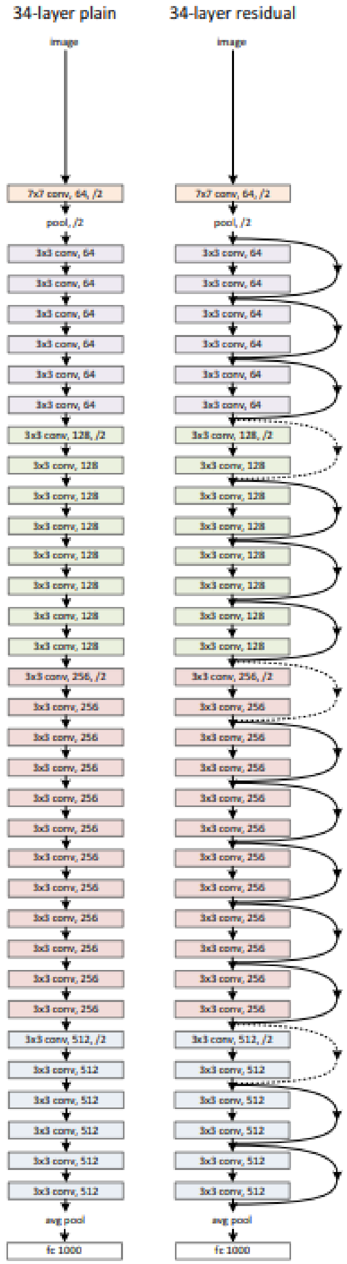
Residual blocks

- Residual blocks
  - Main path
  - Shortcut



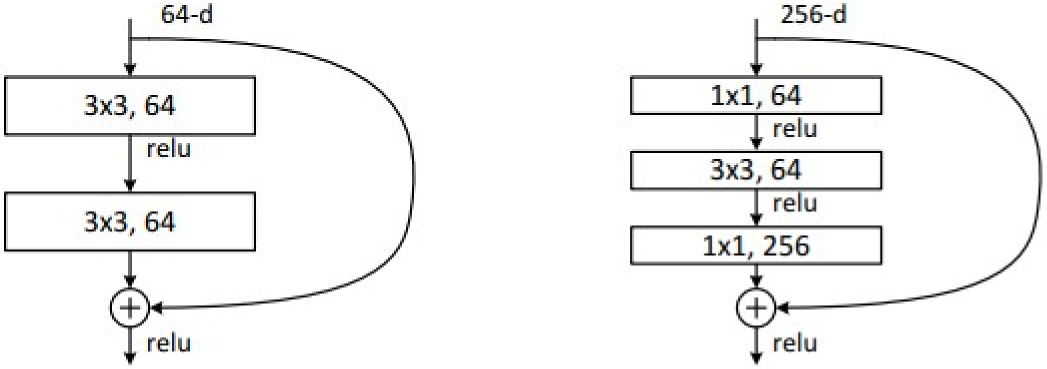
Res Net

- Start with a CONV layer
- Stack multiple residual blocks on top of each other
  - Every residual block has two 3x3 conv layers
  - Periodically, double number of filters and downsample spatially using stride 2
- No fully connected layers



Res Net > 50 layers

- Use "bottlenecks" just like in Inception networks



A deeper residual function F for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet34. Right: a "bottleneck" building block for ResNet-50/101/152.

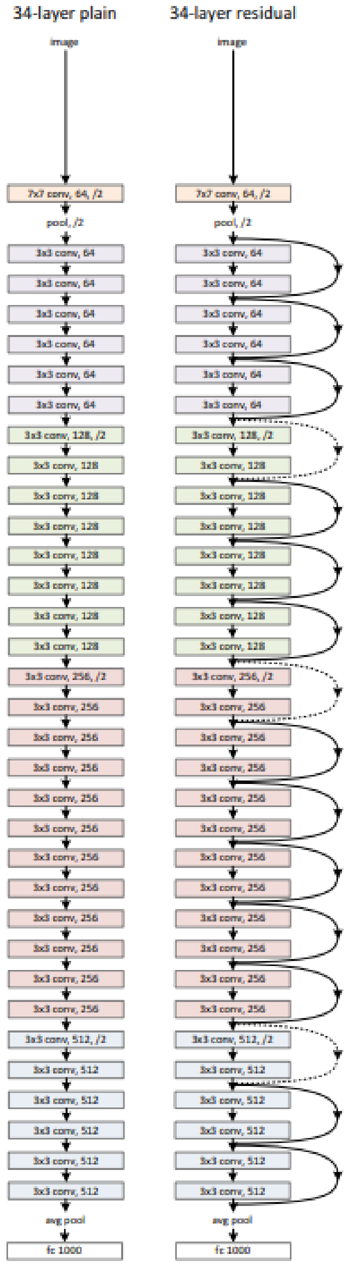
Res Net

- Allows us to train really deep networks
  - “Identity Mappings in Deep Residual Networks”, 2016 – trained a **1001-layer deep** ResNet to outperform its shallower counterparts
- Helps us with vanishing and exploding gradients
- It is easy for ResNet to learn the identity function

Res Net

1st places on:

- ImageNet detection
- ImageNet localization
- COCO detection
- COCO segmentation



Res Net

- Batch Normalization after every CONV layer
- He initialization
- Momentum gradient descent (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

Conv nets – the big picture

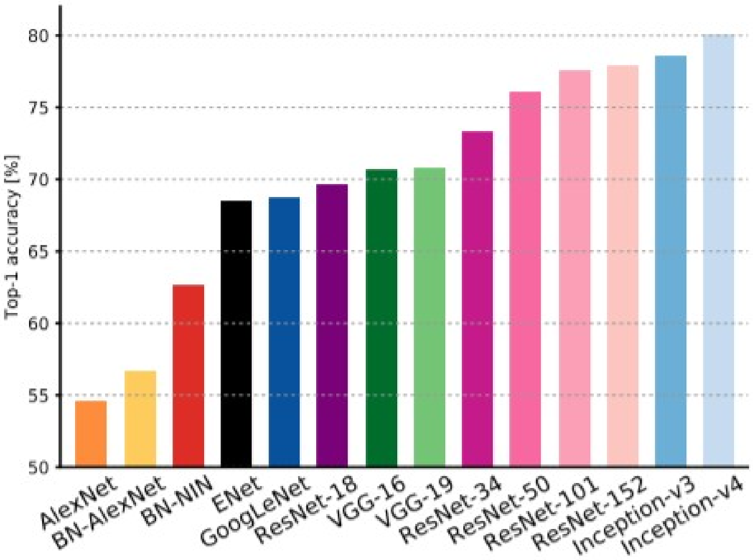


Figure 1: **Top1 *vs.* network.** Single-crop top-1 validation accuracies for top scoring single-model architectures. We introduce with this chart our choice of colour scheme, which will be used throughout this publication to distinguish effectively different architectures and their correspondent authors. Notice that networks of the same group share the same hue, for example ResNet are all variations of pink.

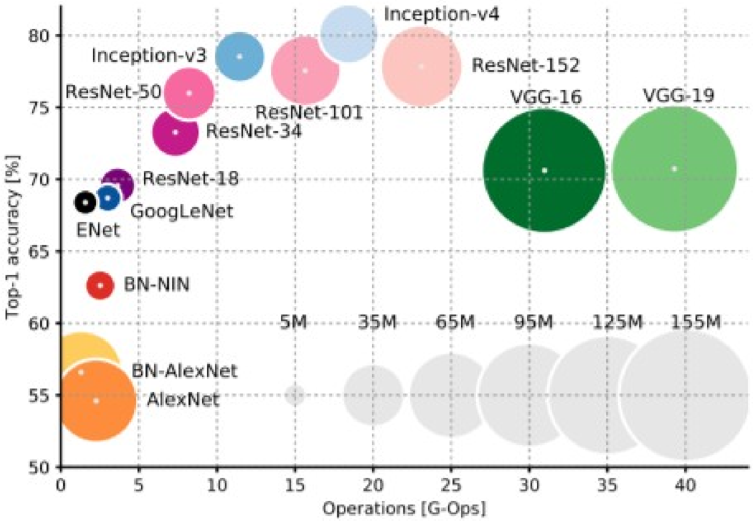
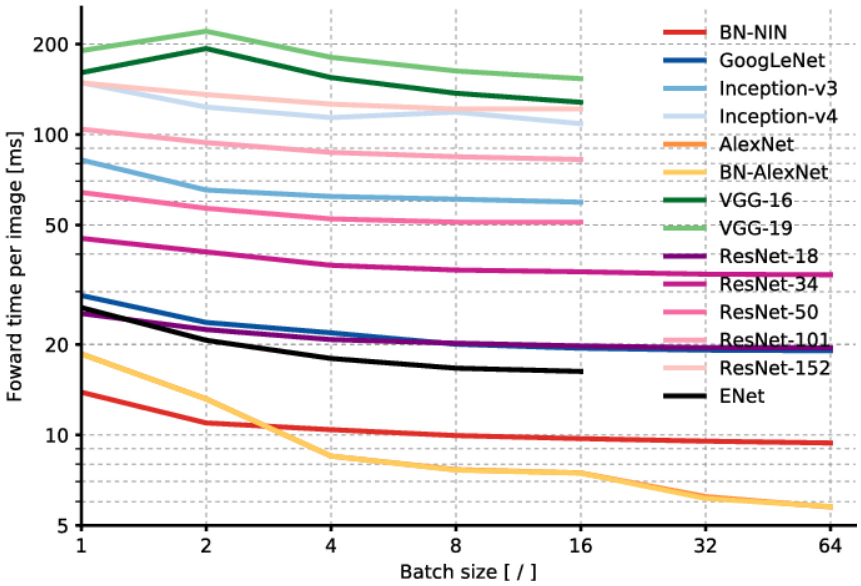
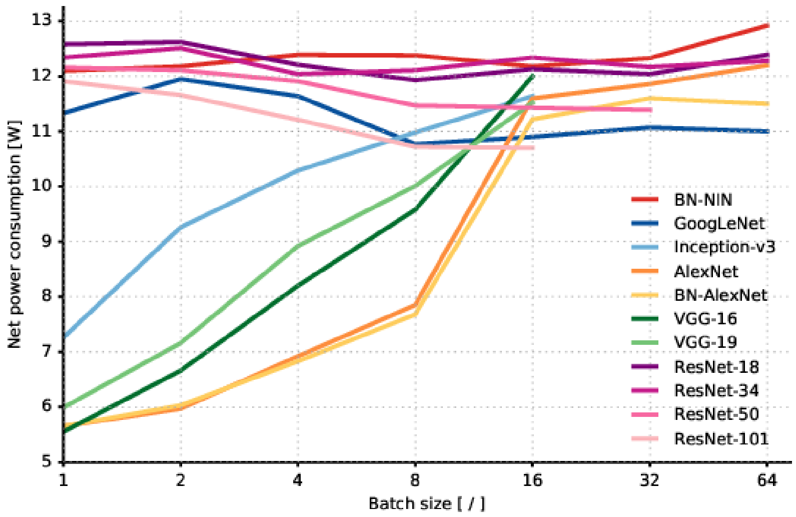


Figure 2: **Top1 *vs.* operations, size $\propto$ parameters.** Top-1 one-crop accuracy versus amount of operations required for a single forward pass. The size of the blobs is proportional to the number of network parameters; a legend is reported in the bottom right corner, spanning from 5×10^6 to 155×10^6 params. Both these figures share the same *y*-axis, and the grey dots highlight the centre of the blobs.

An Analysis of Deep Neural Network Models for Practical Applications

Conv nets – the big picture





How to read a research paper?

1. **Read the Title, the abstract and the figures**
   - get a general sense of the concepts in the paper
2. **Read the introduction + conclusions + figures + skim the rest**
   - author(s) try to summarize their work carefully to clarify for the reviewer why their paper should be accepted for publication
3. **Read the paper but skip the math**
4. **Read the whole thing but skip the parts that don't make sense**
   - some of it doesn't make sense (it's not unusual), it's okay to skim it initially.