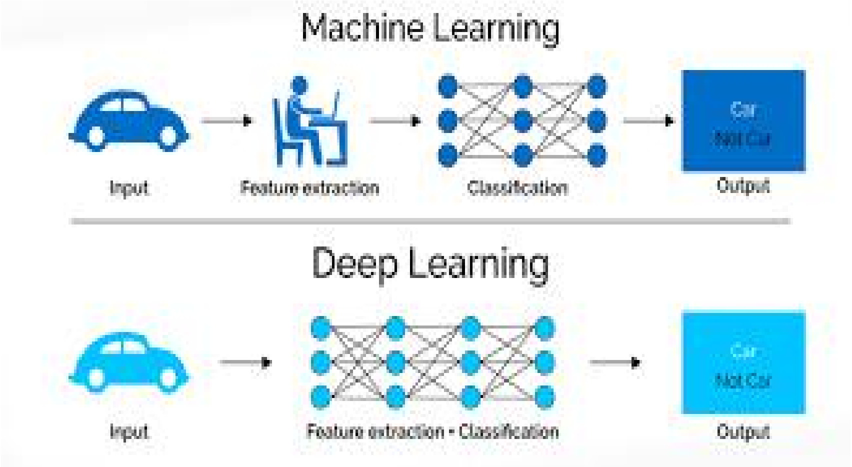
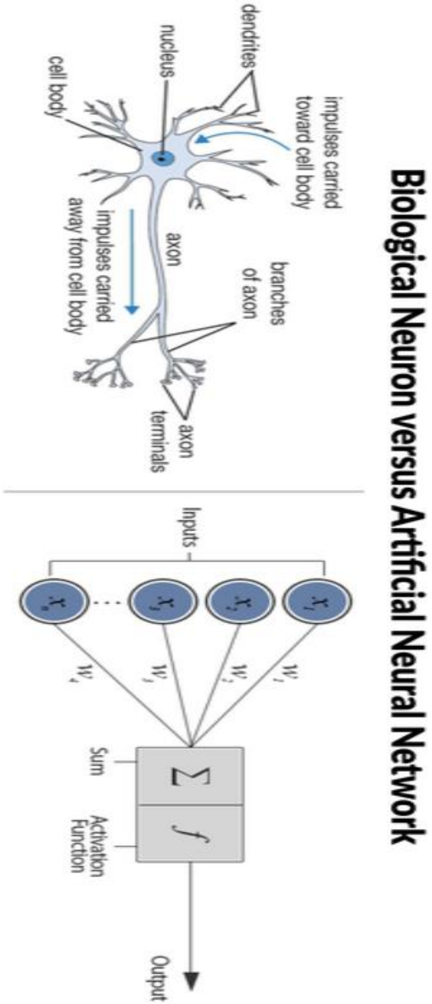
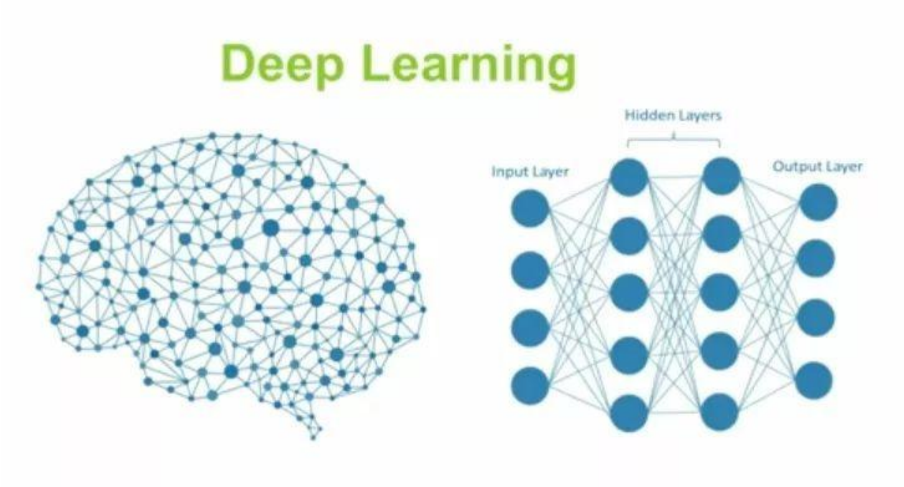
Chapter-3-I

Convolutional Neural Networks


julia
mxnet
K
DEEP
LEARNING
theano
Microsoft
CNTK
Caffe2
Chainer


Machine Learning
Input
Feature extraction
Classification
Output
Deep Learning
Input
Feature extraction + Classification
Output

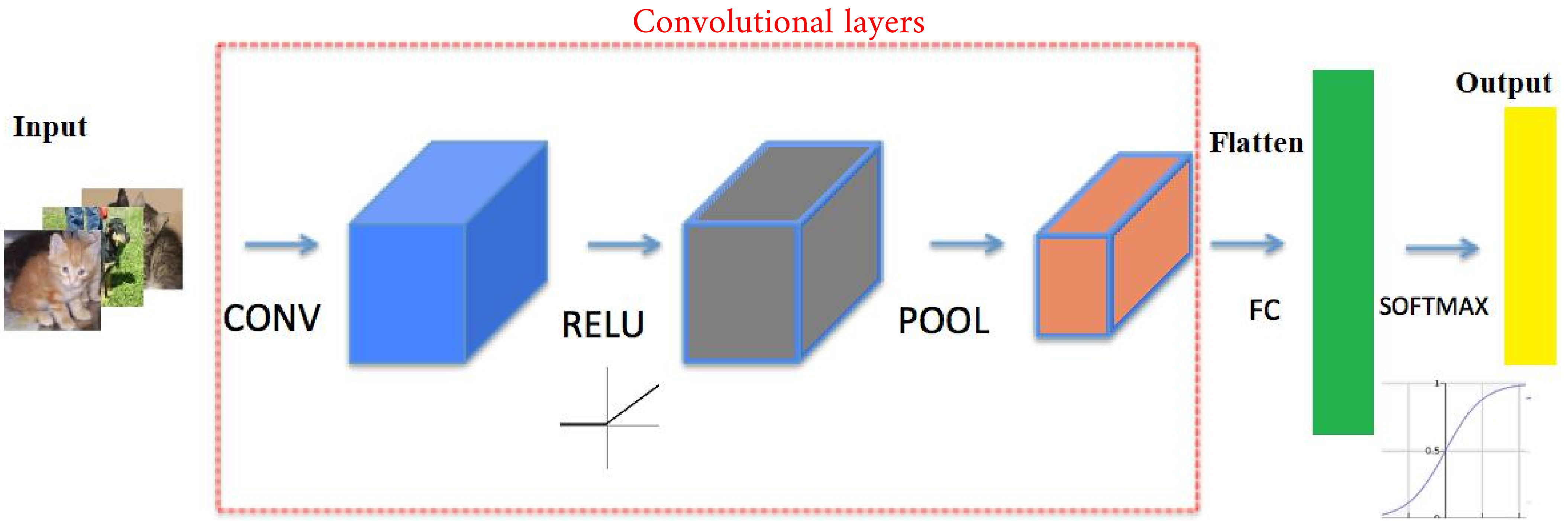

Biological Neuron versus Artificial Neural Network
dendrites
nucleus
cell body
impulses carried toward cell body
axon
impulses carried away from cell body
branches of axon
axon terminals
Inputs
Sum
Activation Function
Output


Deep Learning
Hidden Layers
Input Layer
Output Layer


DEEP
LEARNING

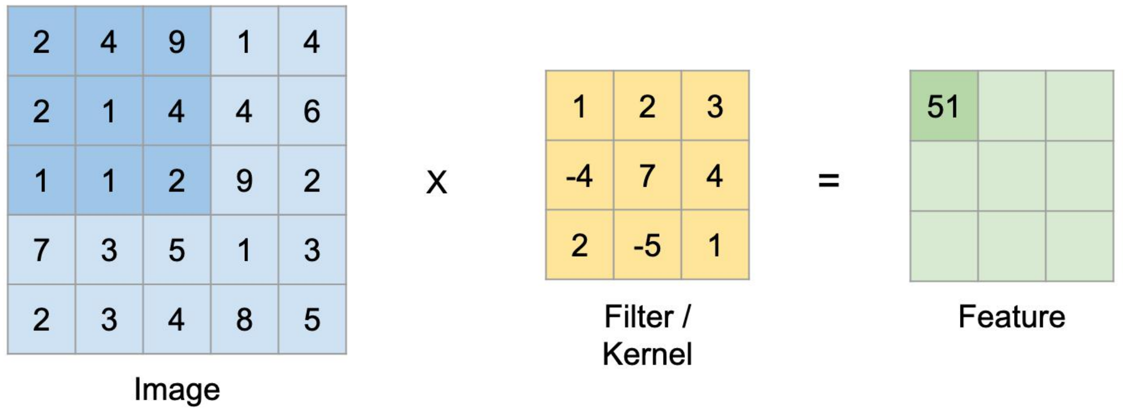
A Convolutional Neural Networks (CNNs)

- **CNNs** are a special kind of multi-layer neural networks, designed to recognize visual patterns directly from pixel images with minimal preprocessing

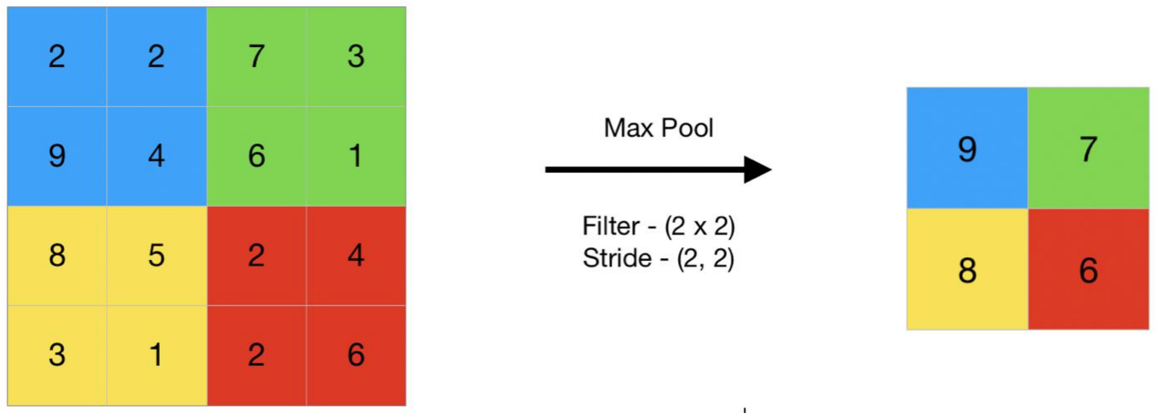


*Convolution: summarize/learn the presence of features in an input image
*Pooling: A fixed operation that down sample the features

A Convolutional Neural Networks (CNNs)- Examples

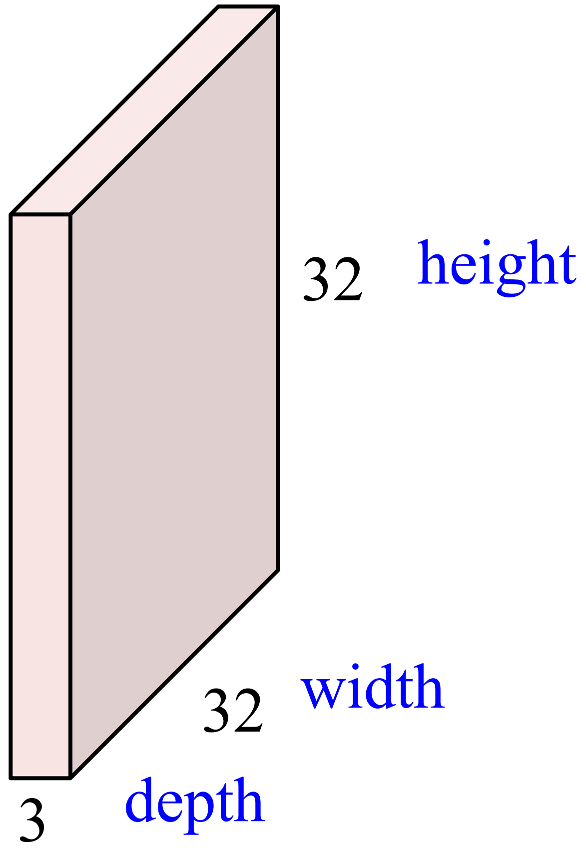


(CONV) uses filters that perform convolution operations as it is scanning the input with respect to its dimensions. Its hyperparameters include the filter size and stride . The resulting output called feature map/activation map



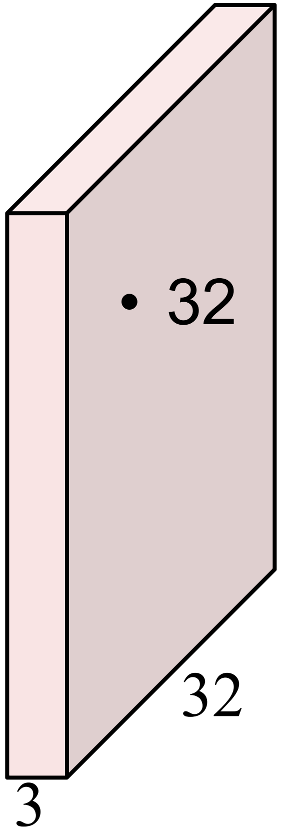
A Convolutional Neural Networks (CNNs)- Examples

32x32x3 image

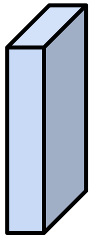


A Convolutional Neural Networks (CNNs)- Examples

32x32x3 image

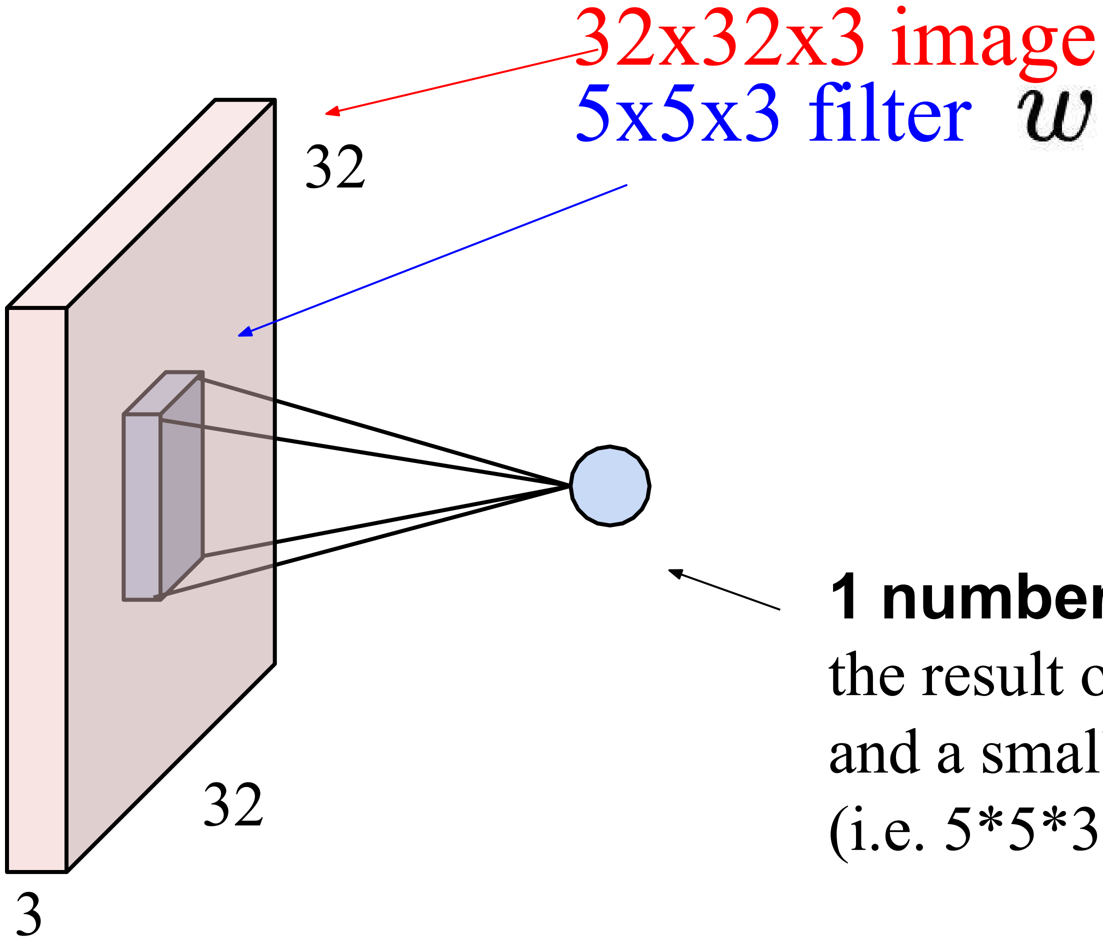


5x5x3 filter



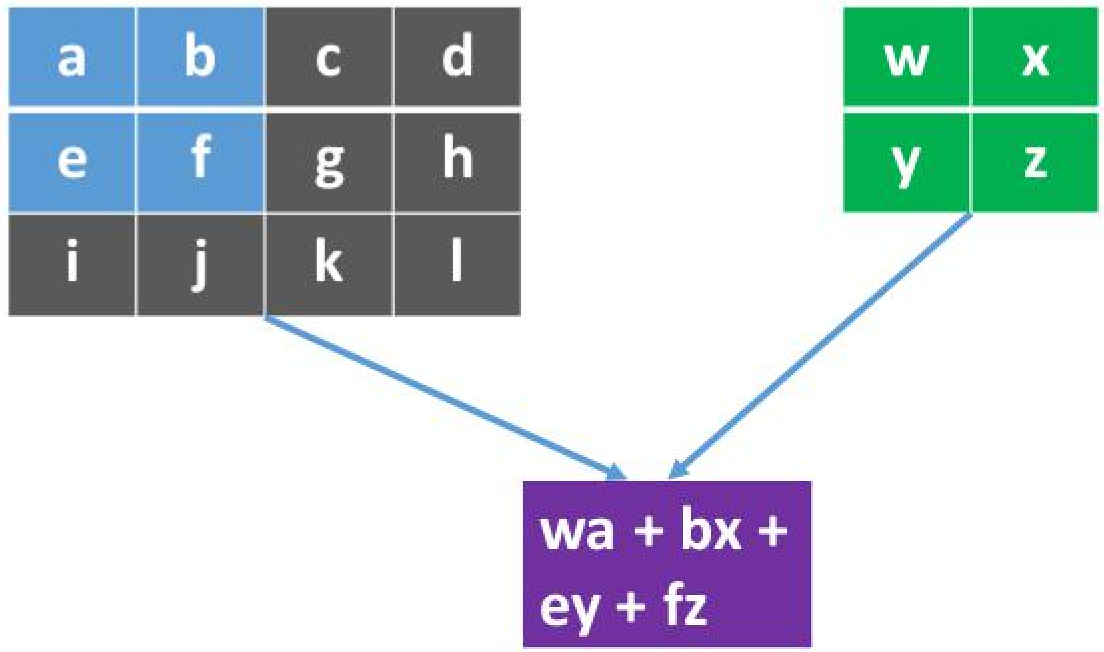
- **Convolve** the filter with the image
- i.e. “slide over the image spatially, computing dot products”

Convolution Layer

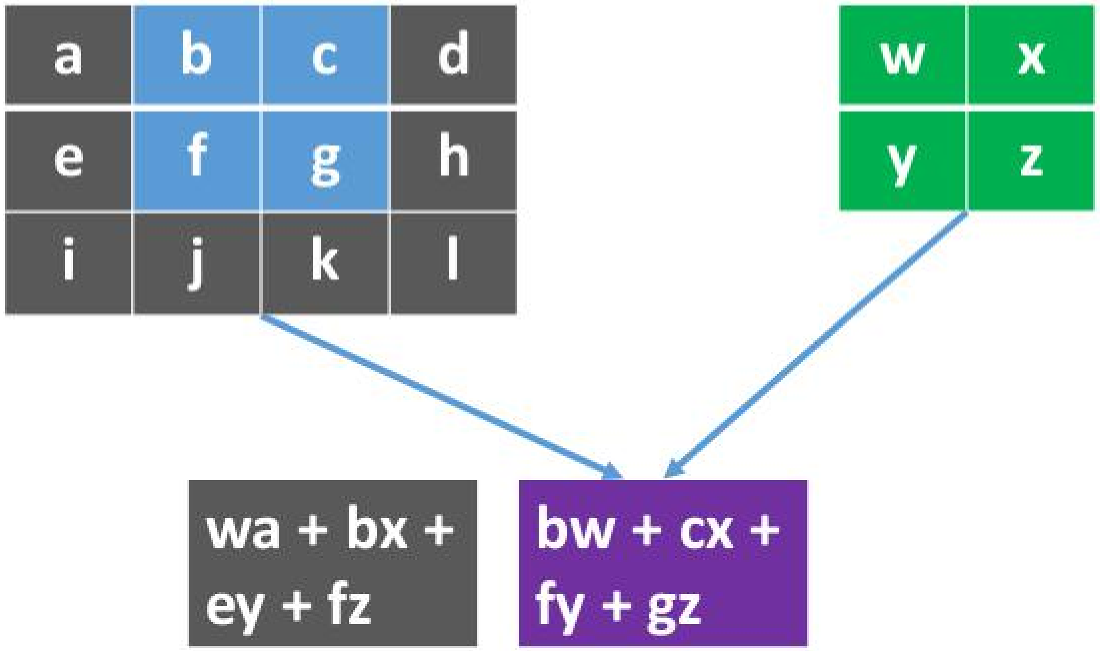


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

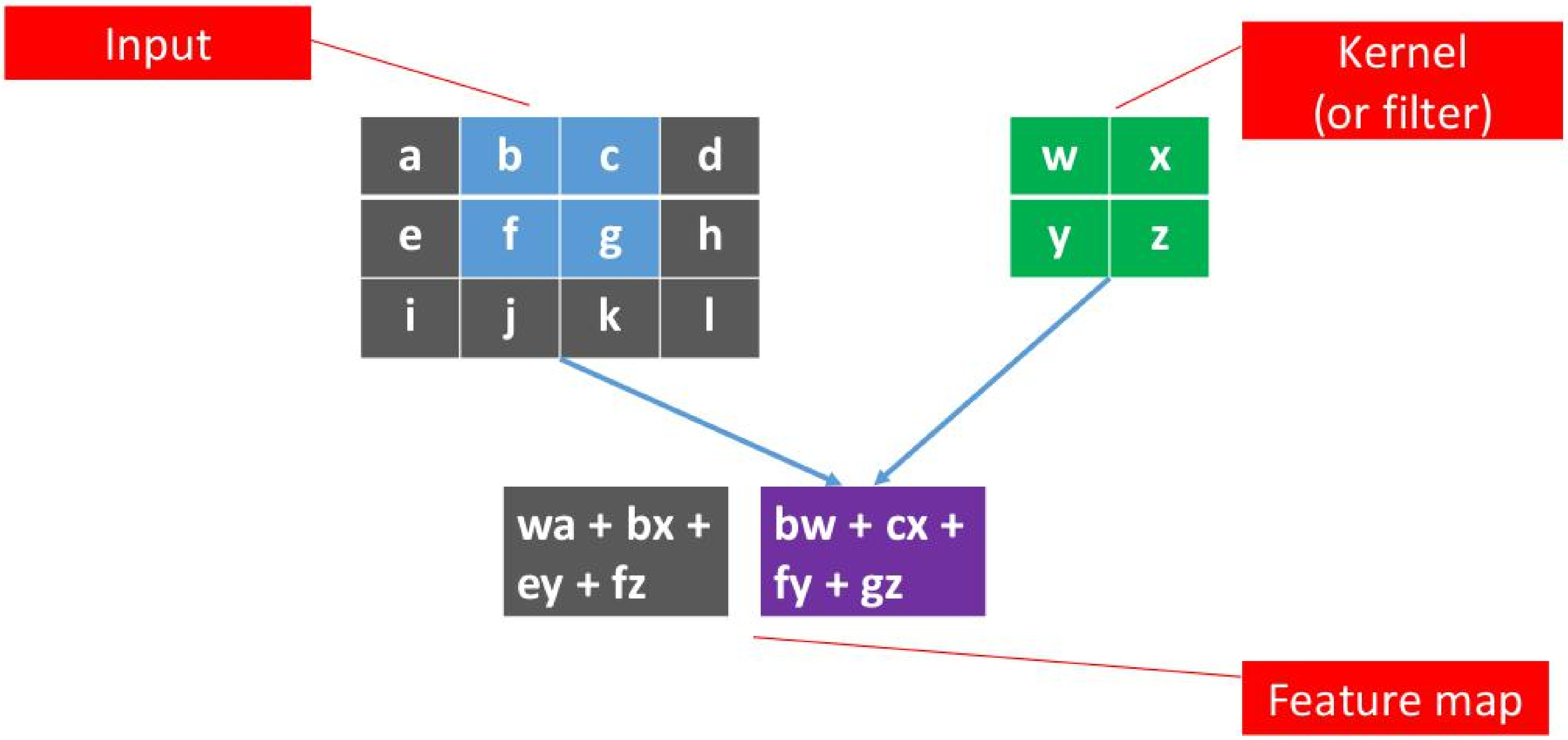
Illustration



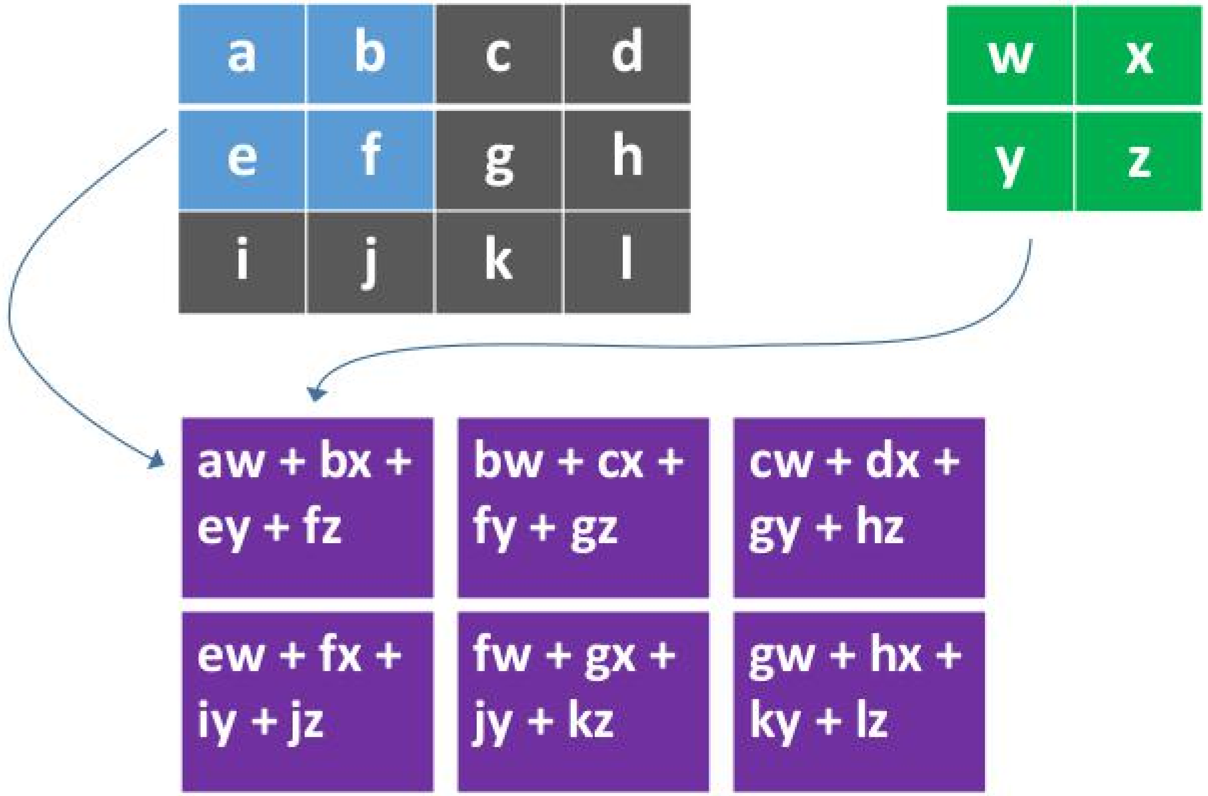
Illustration



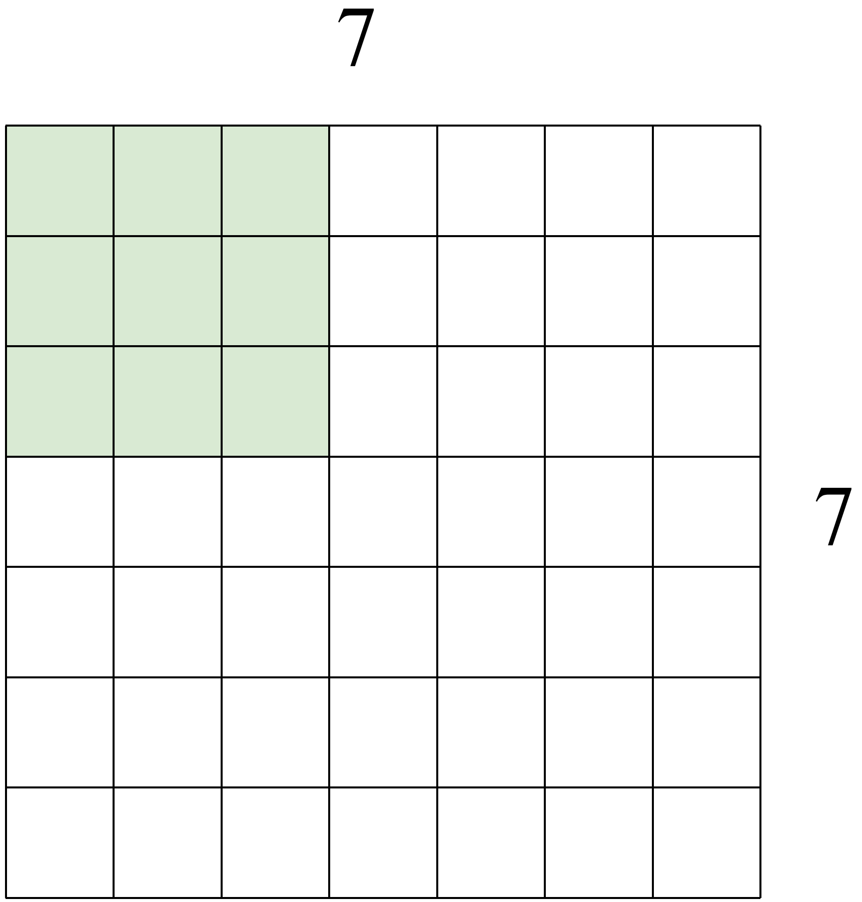
Illustration



Illustration

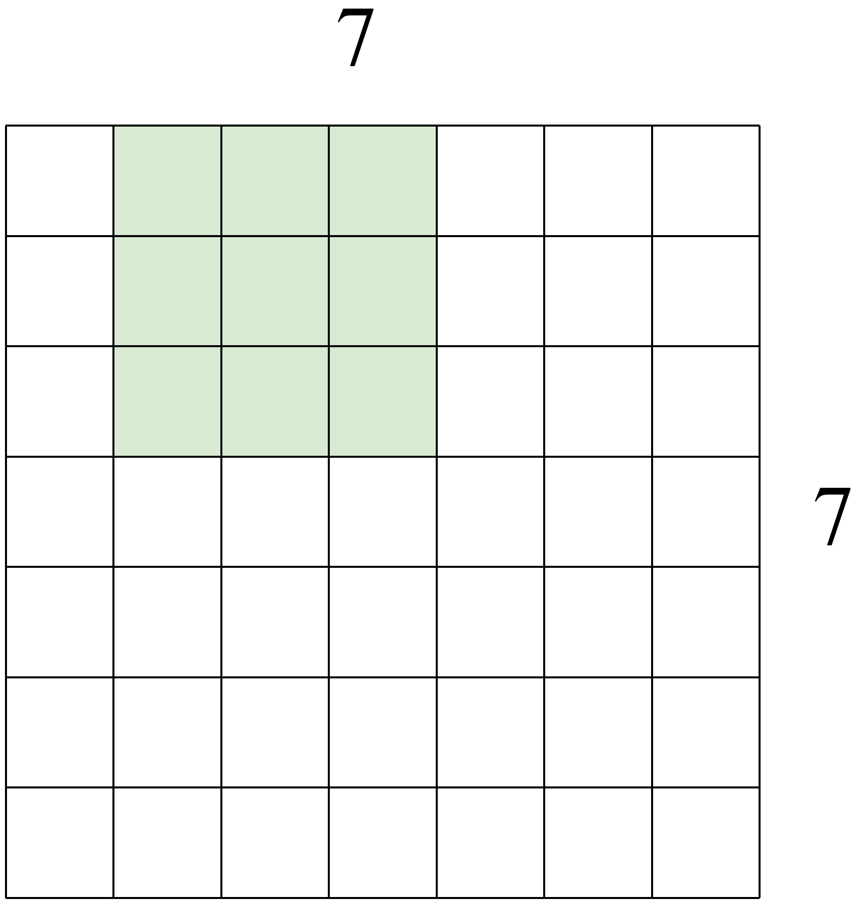


A closer look at spatial dimensions:



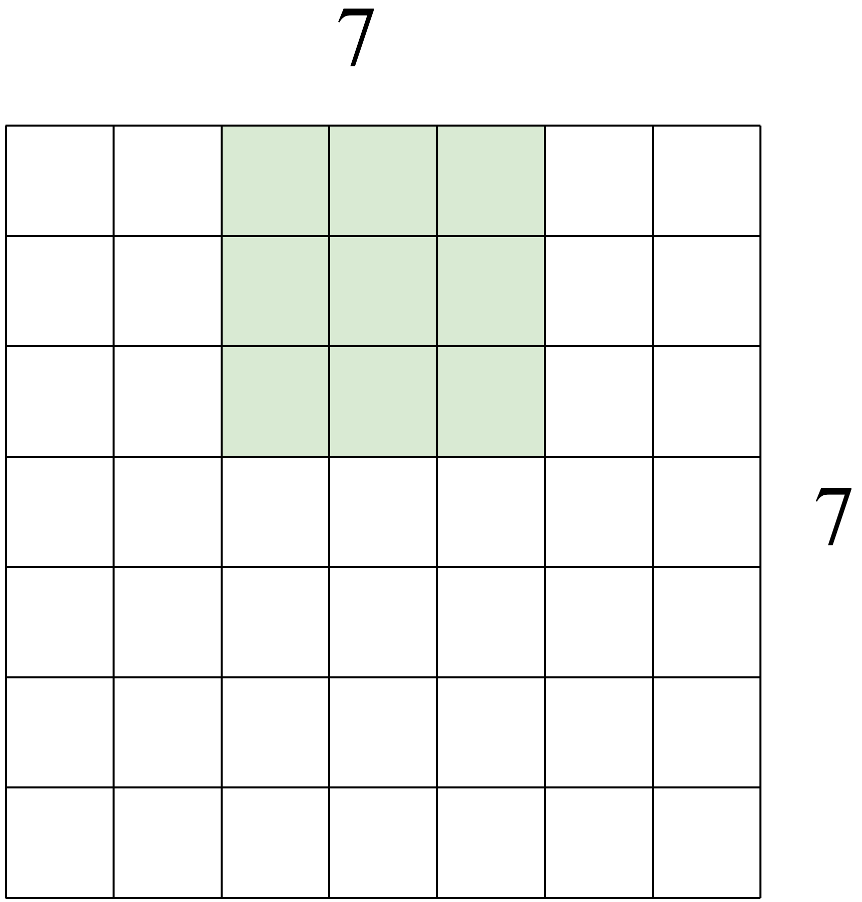
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

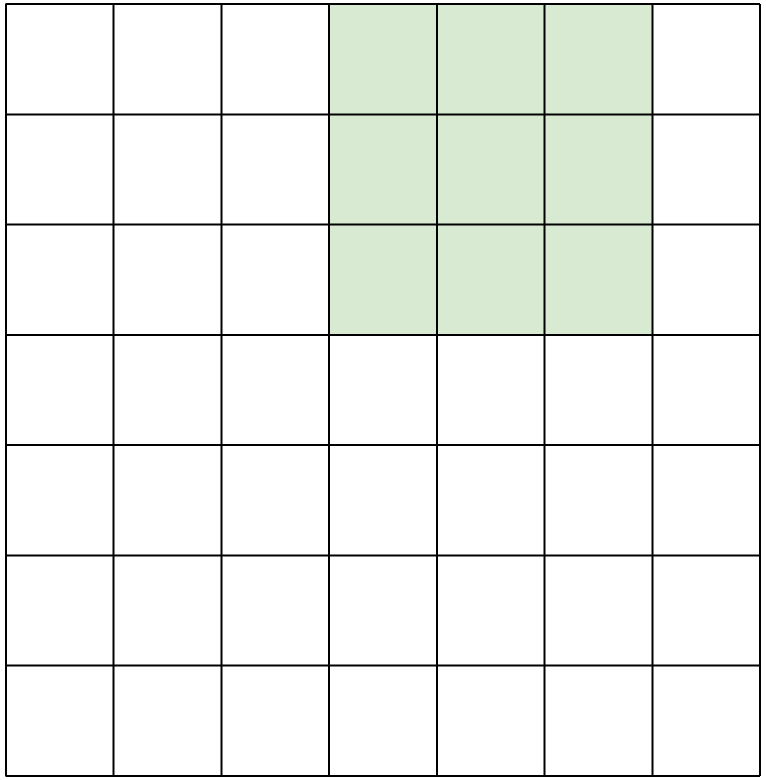
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

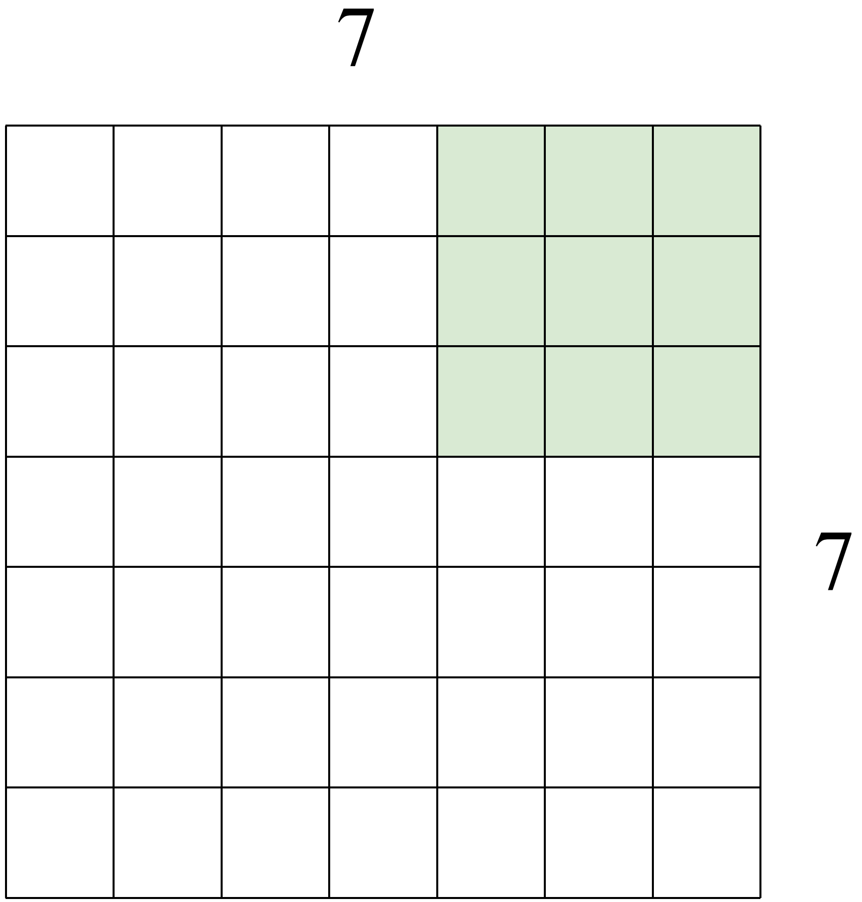
7



7

7x7 input (spatially)
assume 3x3 filter

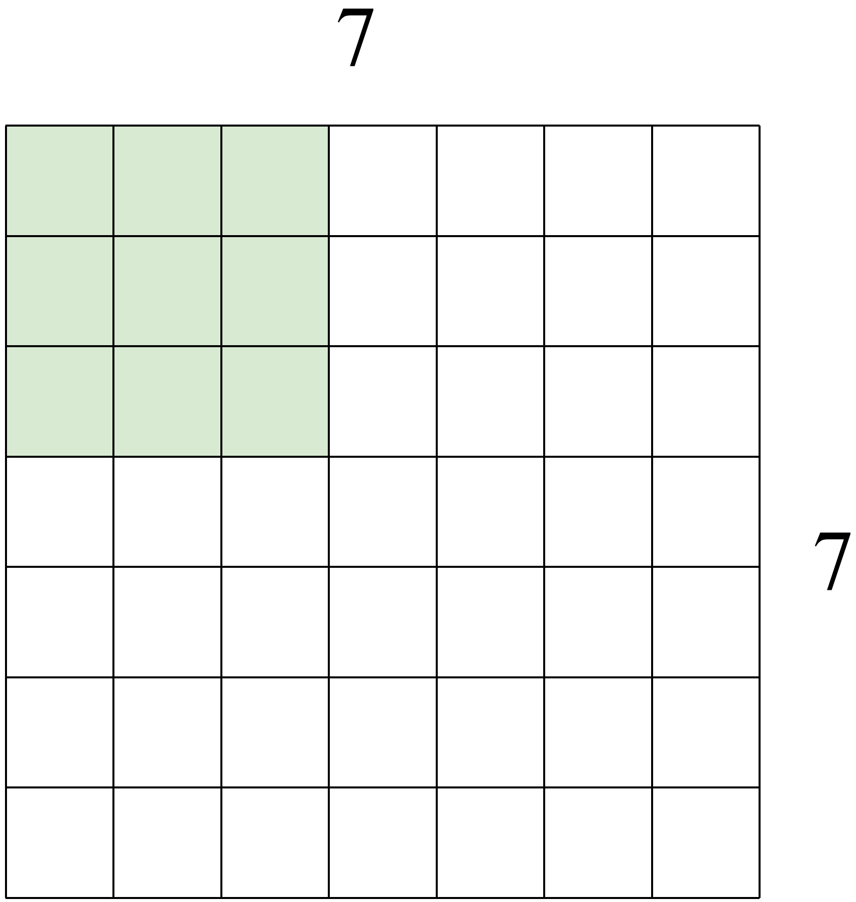
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

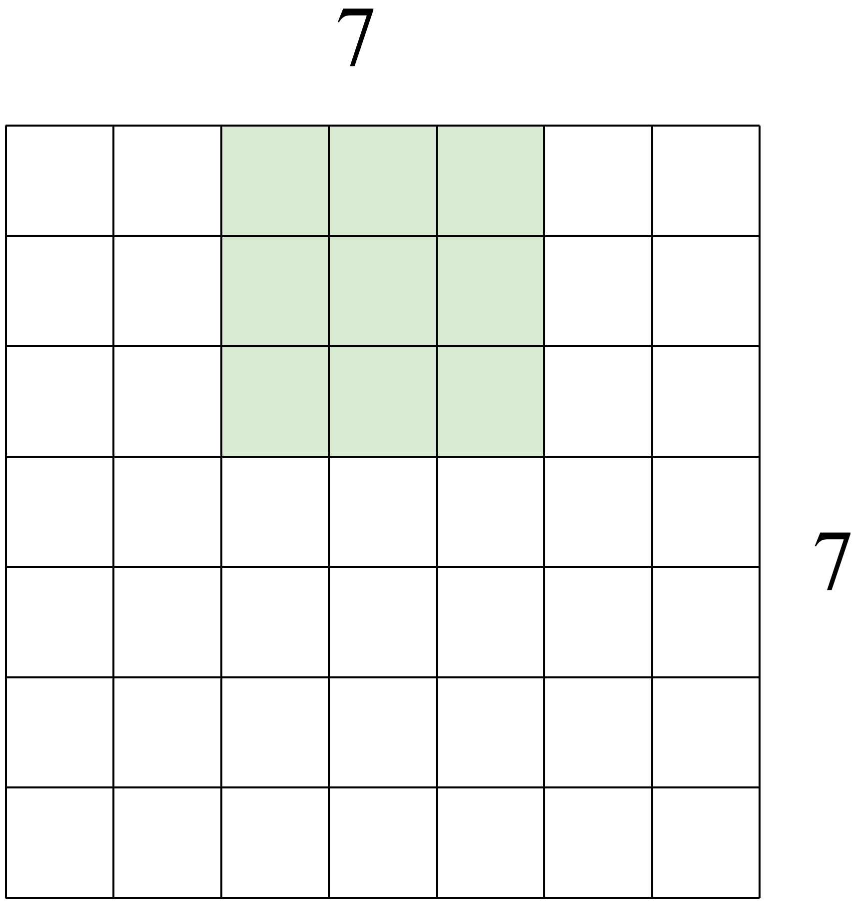
=> 5x5 output

A closer look at spatial dimensions:



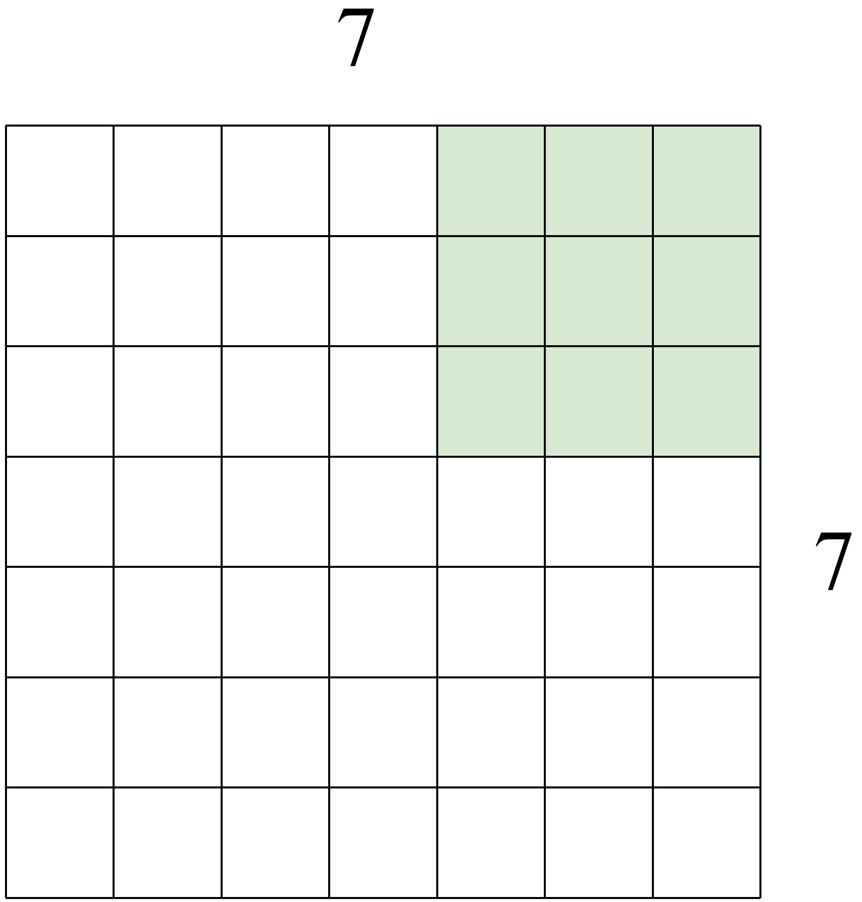
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



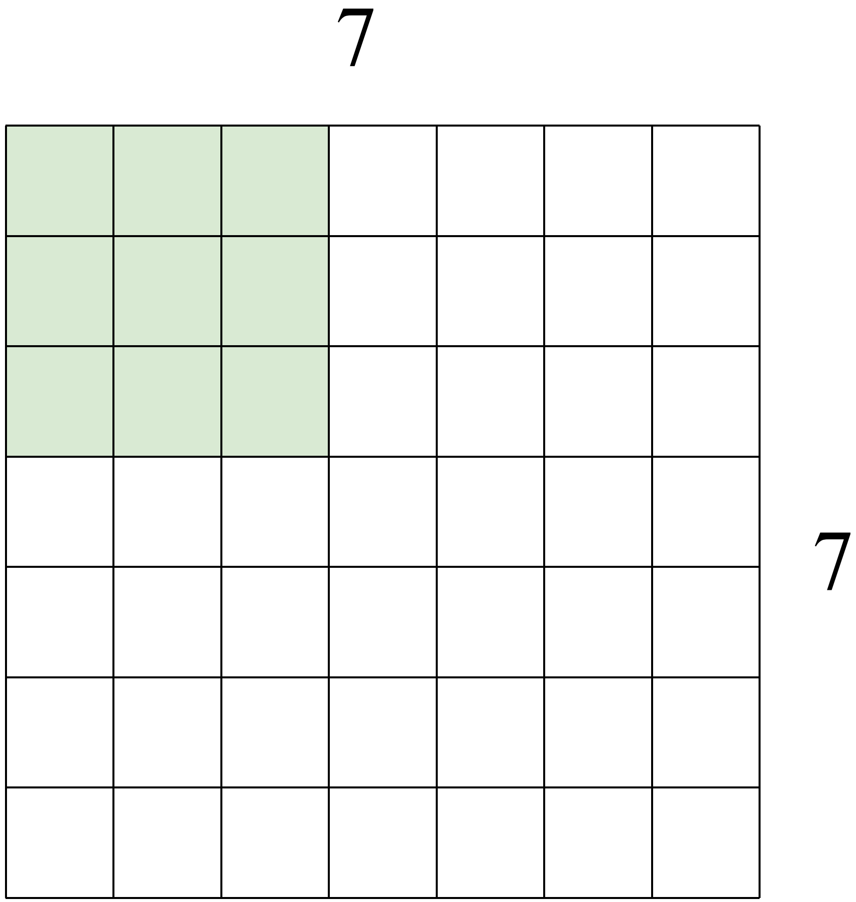
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



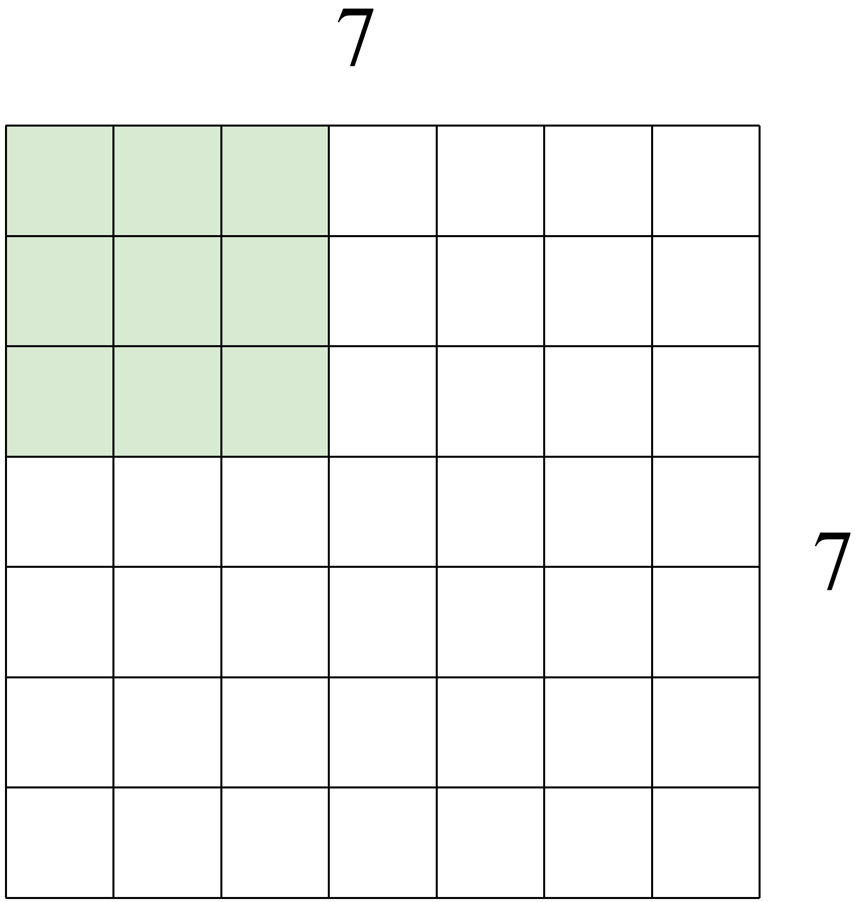
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



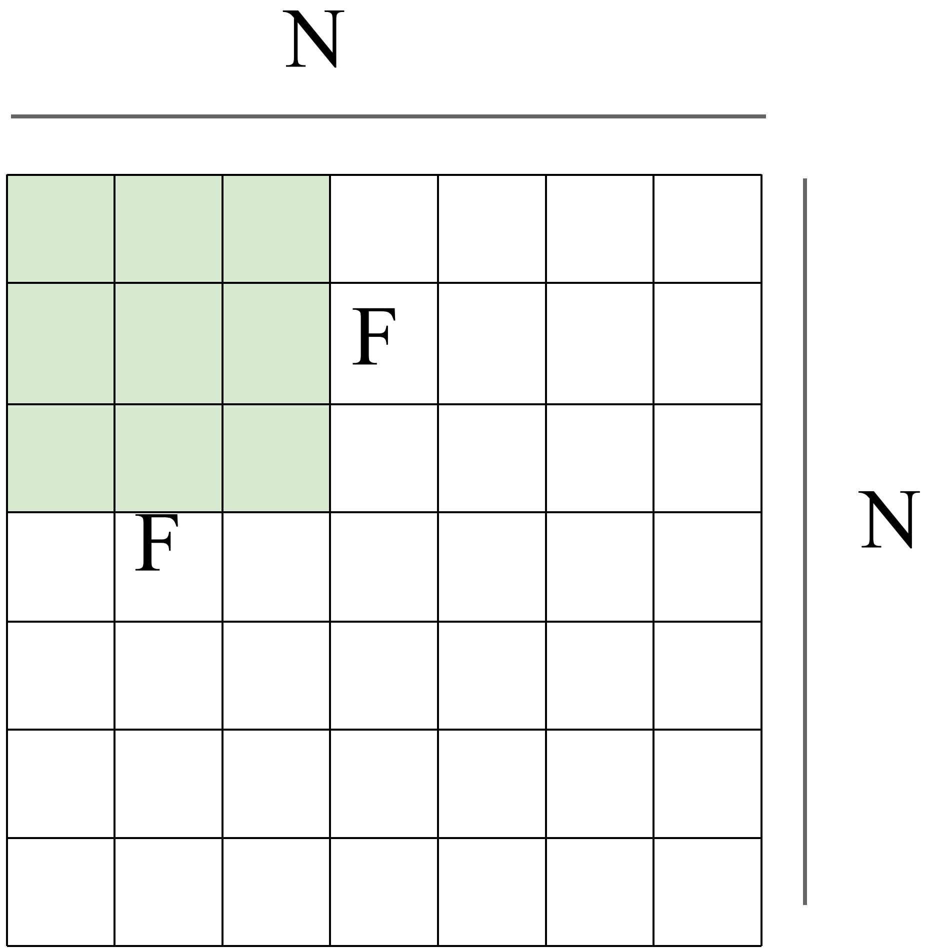
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

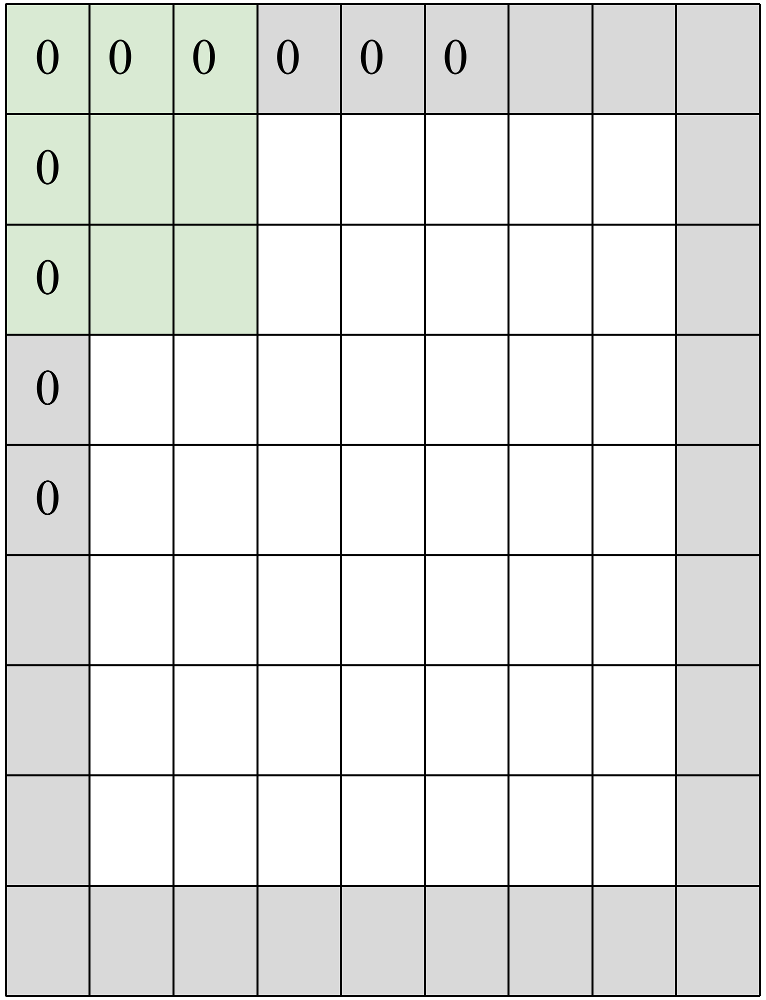
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

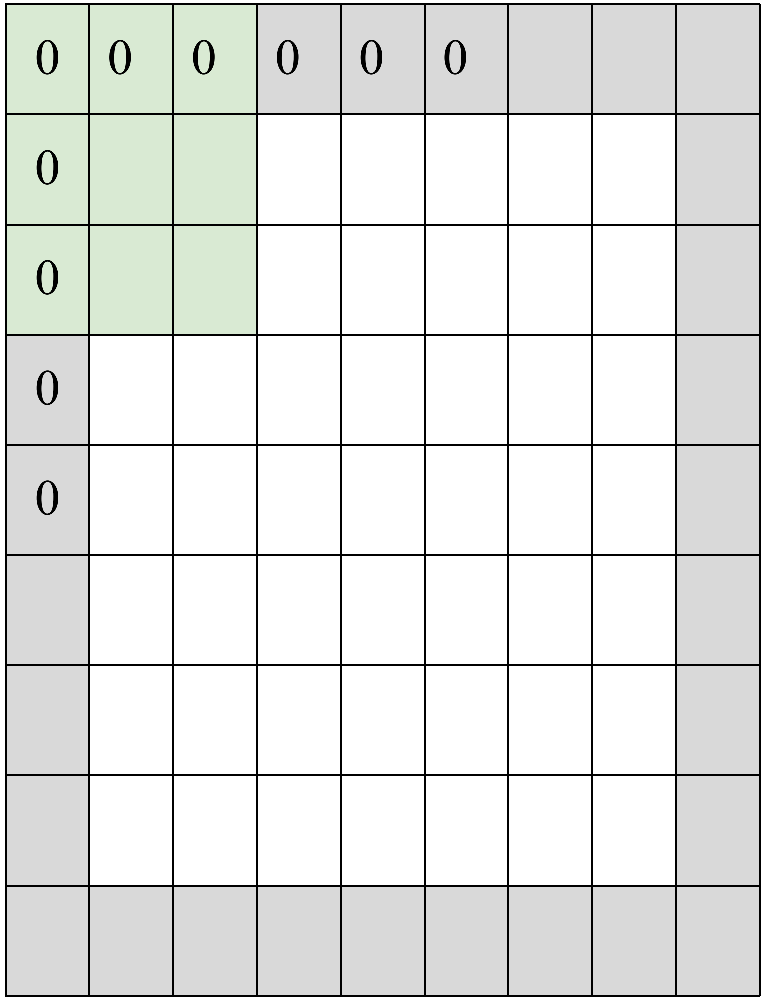
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
$(N - F) / \text{stride} + 1$

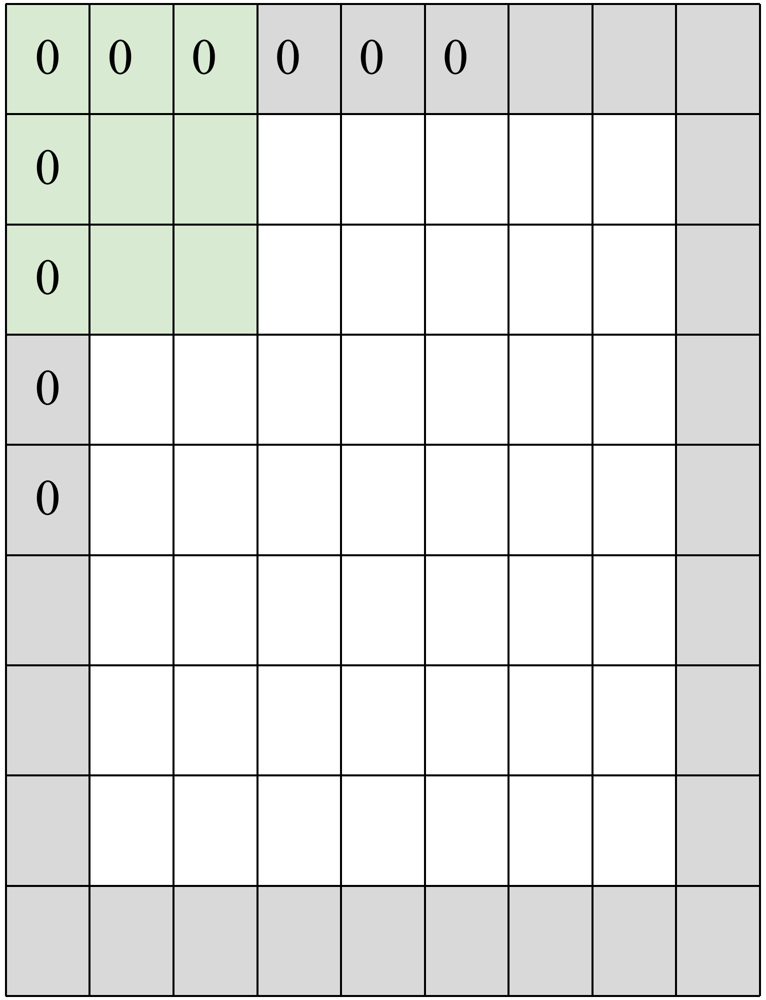
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

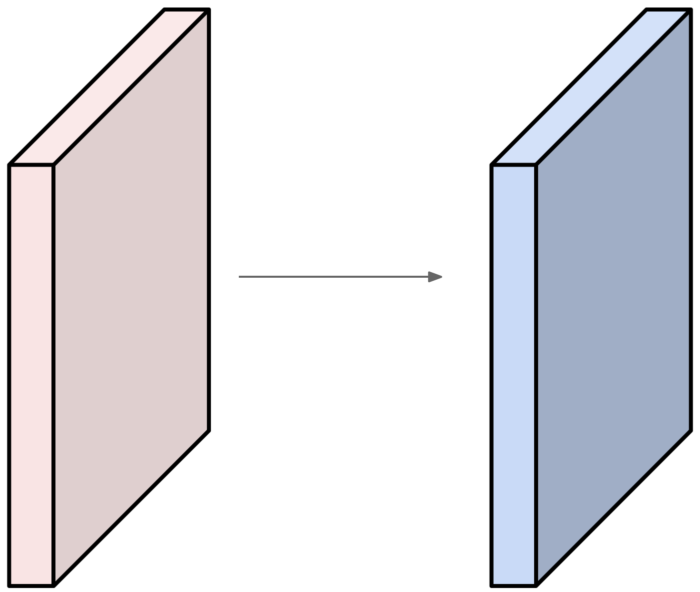
7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1 F = 5 => zero pad with 2 F = 7 => zero pad with 3

- Please read Valid padding and Full padding

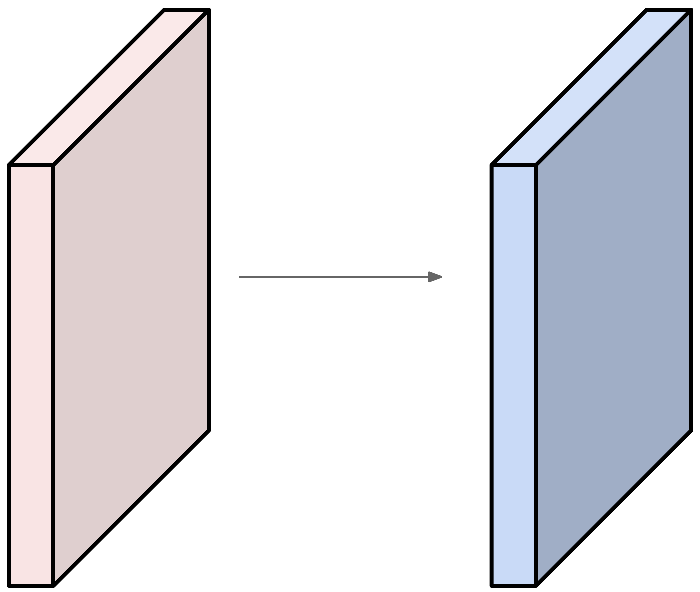
Examples

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?



Examples



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

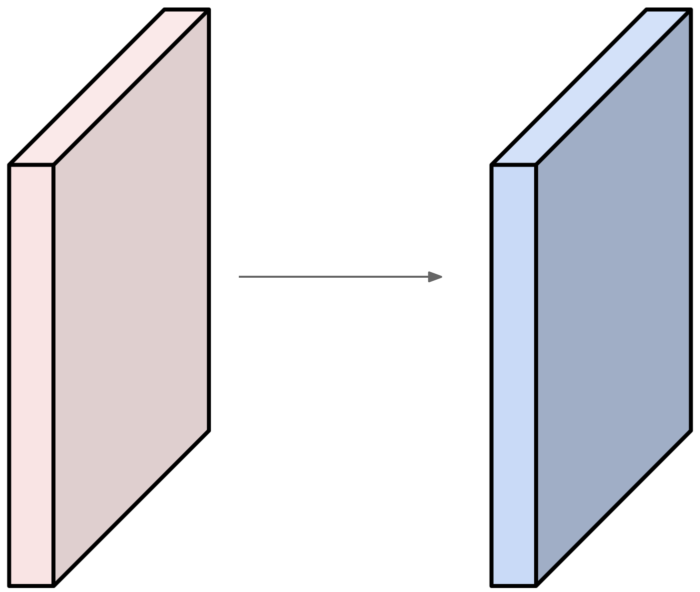
Output volume size:

(32-5+2*2)/1+1 = 32 spatially, so

32x32x10

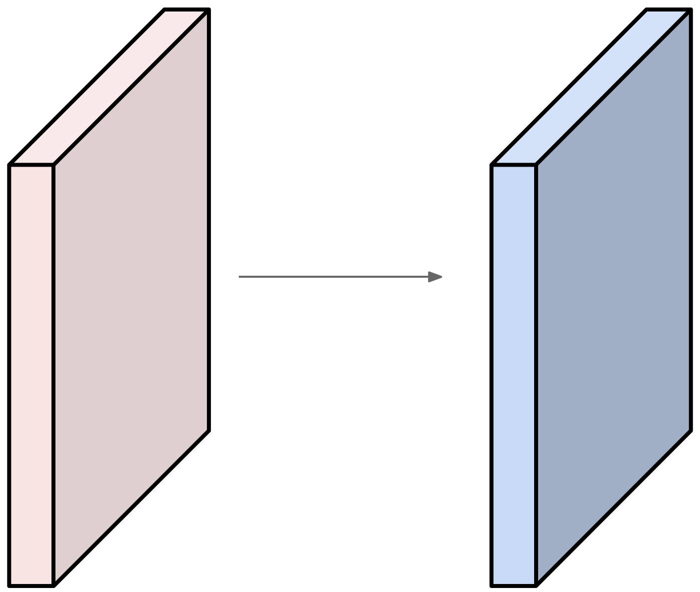
Examples

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?

Examples:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)

=> 76*10 = **760**

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Variants of CNN

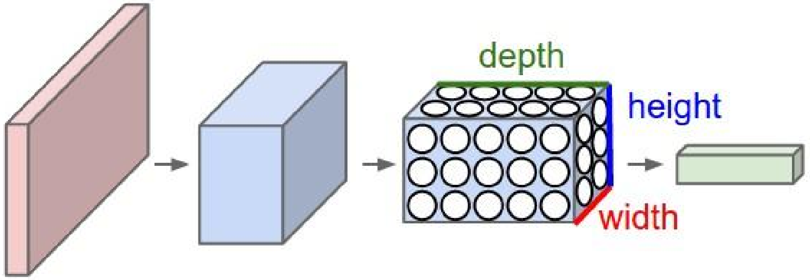
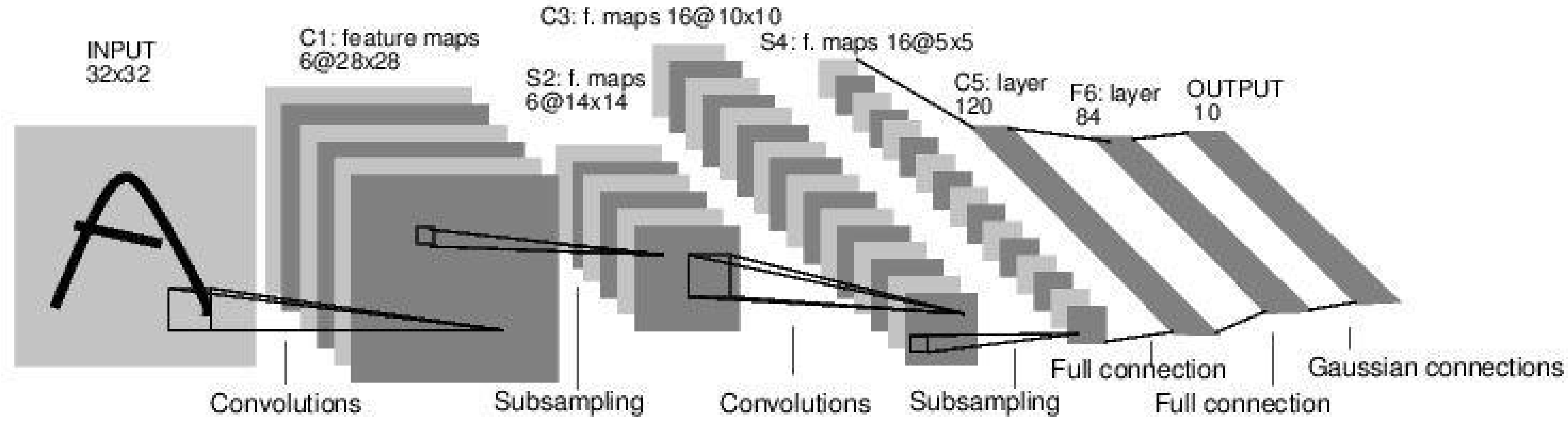


Image → Convolution → Pooling → Flattenning → Fully Connected Layer → Softmax → Loss

LeNet-5

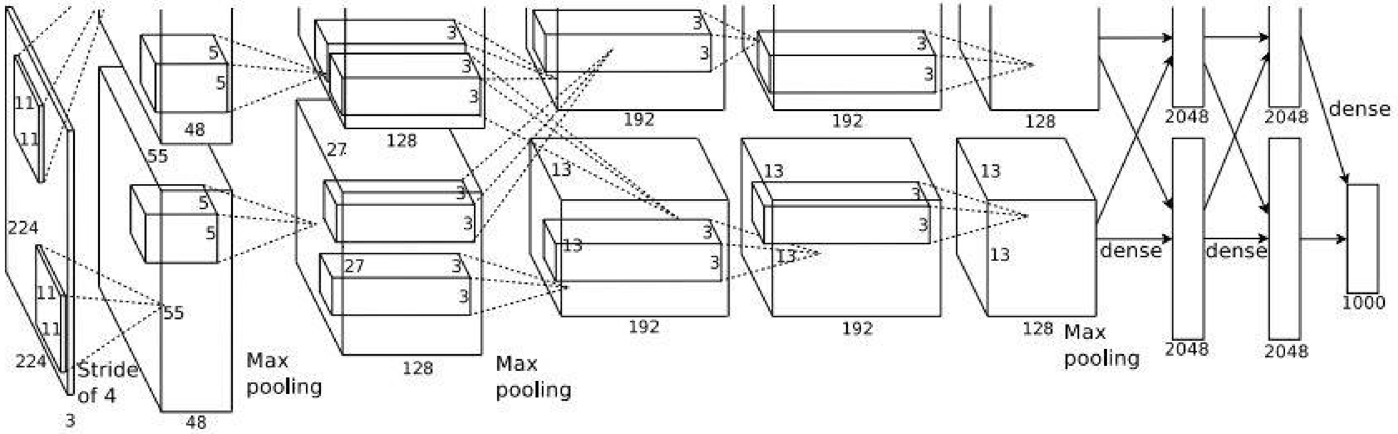
[LeCun et al., 1998]



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

AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

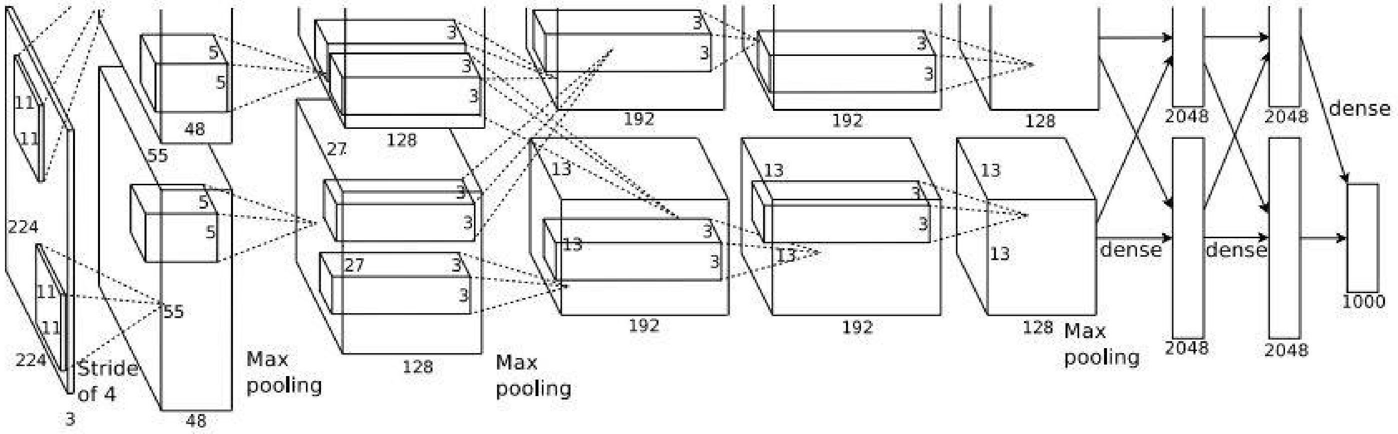
First layer (CONV1): 96 11x11 filters applied at stride 4

=>

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

AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

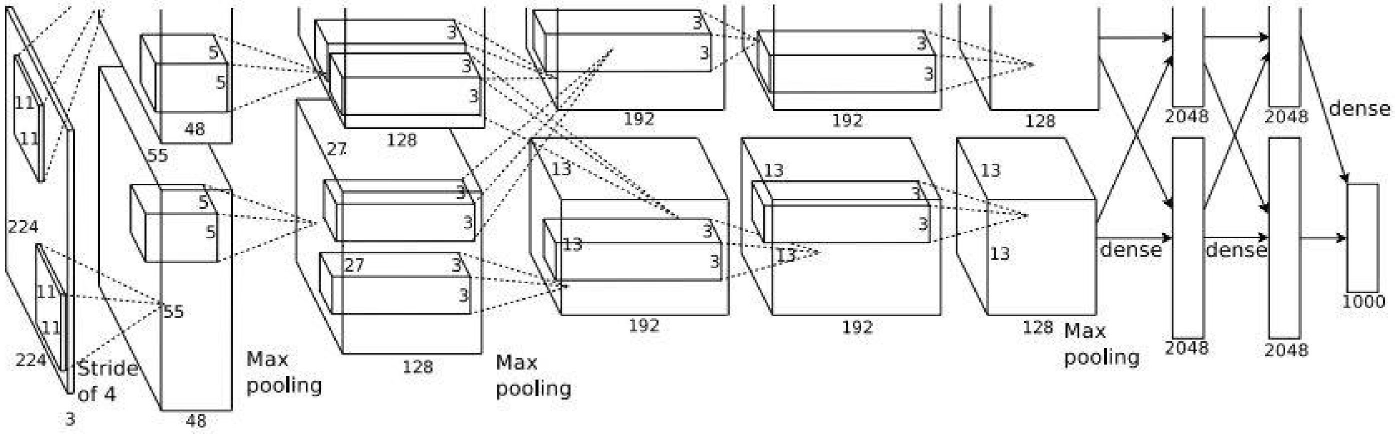
=>

Output volume **[55x55x96]**

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

AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

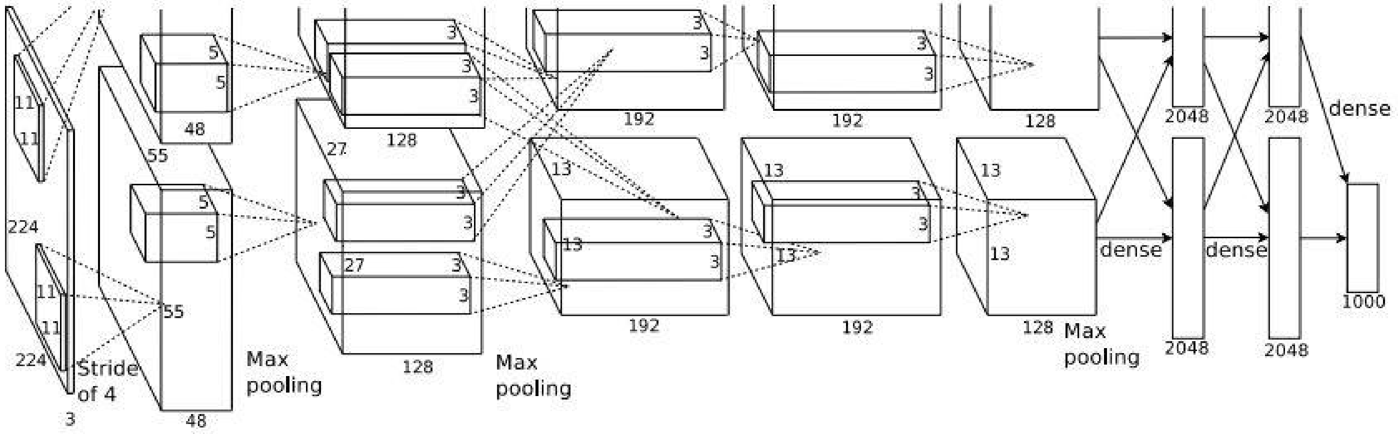
=>

Output volume **[55x55x96]**

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

AlexNet

[Krizhevsky et al. 2012]



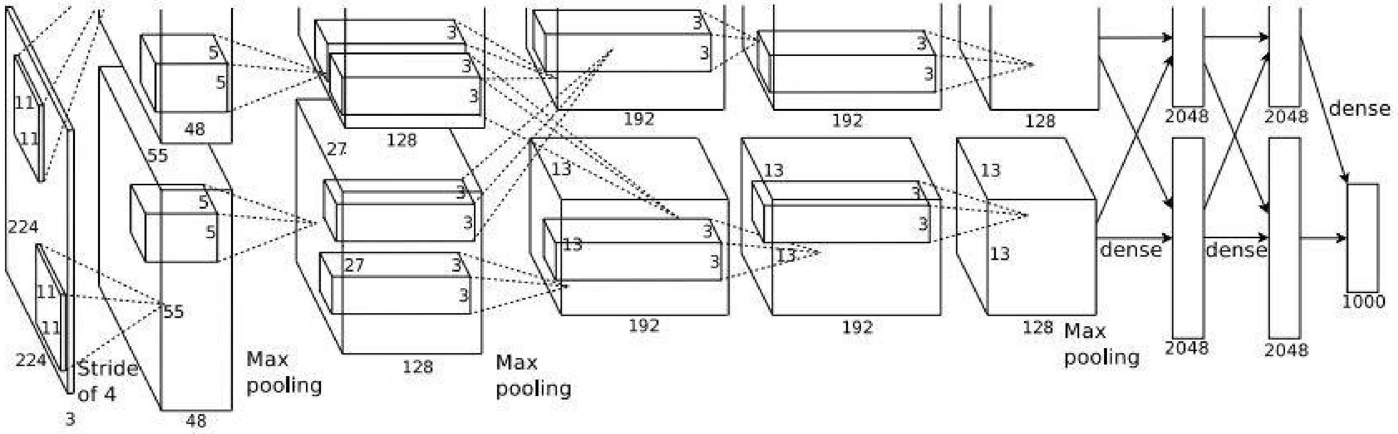
Input: 227x227x3 images After CONV1: 55x55x96

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

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

AlexNet

[Krizhevsky et al. 2012]



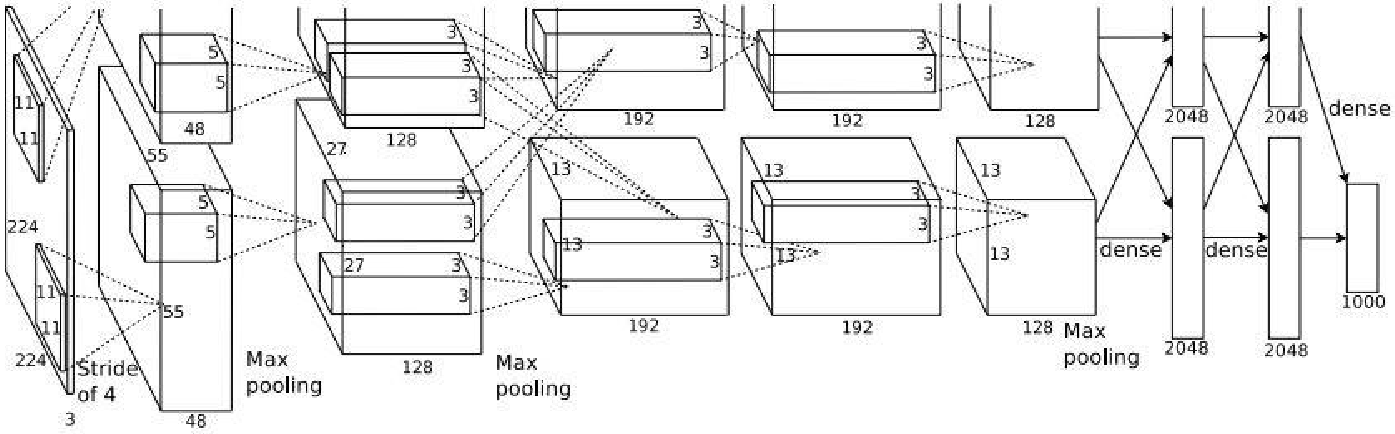
Input: 227x227x3 images After CONV1: 55x55x96

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

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

AlexNet

[Krizhevsky et al. 2012]

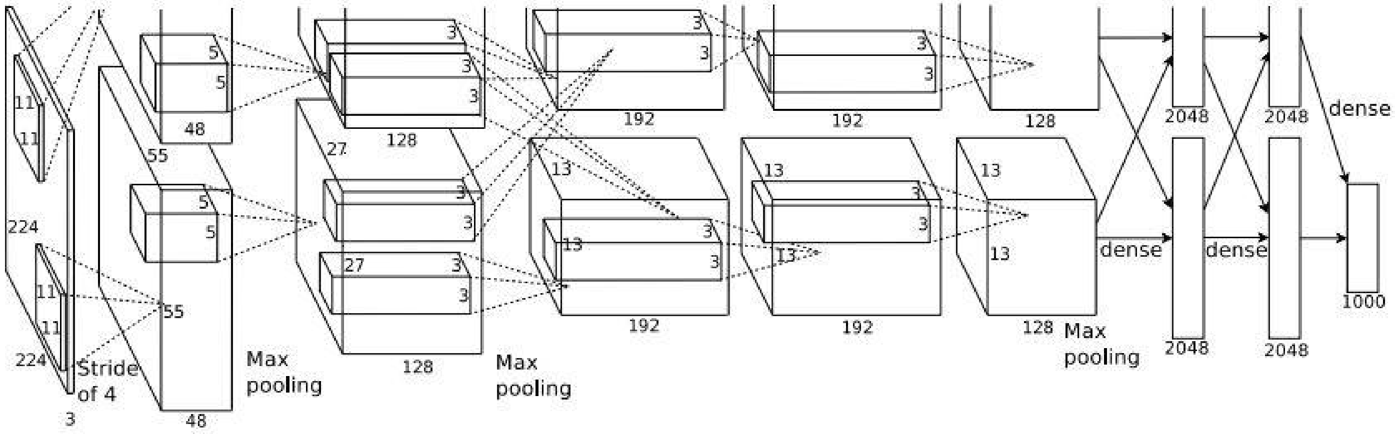


Input: 227x227x3 images After CONV1: 55x55x96

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

AlexNet

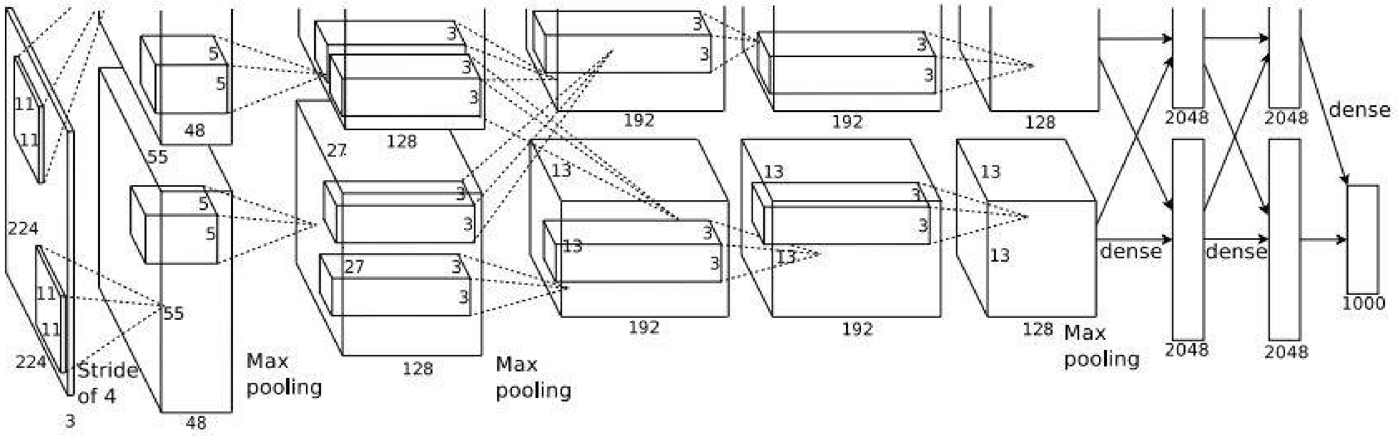
[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

AlexNet

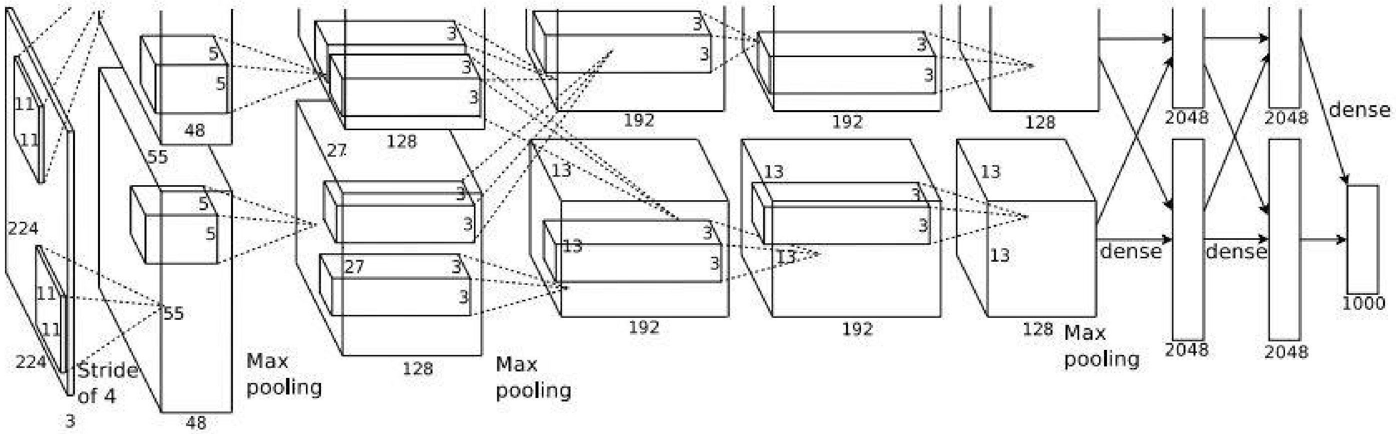
[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture: [227x227x3]
INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer [13x13x384]
CONV3: 384 3x3 filters at stride 1, pad 1 [13x13x384]
CONV4: 384 3x3 filters at stride 1, pad 1 [13x13x256]
CONV5: 256 3x3 filters at stride 1, pad 1 [6x6x256]
MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

AlexNet

[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture: [227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer [13x13x384]
CONV3: 384 3x3 filters at stride 1, pad 1 [13x13x384]
CONV4: 384 3x3 filters at stride 1, pad 1 [13x13x256]
CONV5: 256 3x3 filters at stride 1, pad 1 [6x6x256]
MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

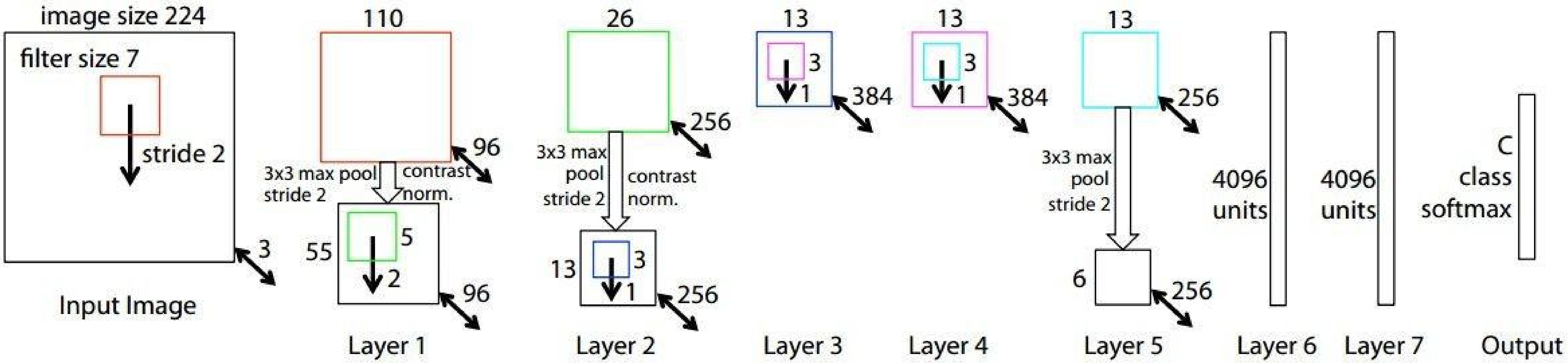
~65M parametres

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2

ImageNet error 15.4%

ZFNet

[Zeiler and Fergus, 2013]



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

ImageNet top 5 error: 15.4% -> 14.8%

VGGNet

[Simonyan and Zisserman, 2014]

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

best model

11.2% top 5 error in ILSVRC 2013

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

INPUT: [224x224x3] memory: 224*224*3=150K params: 0

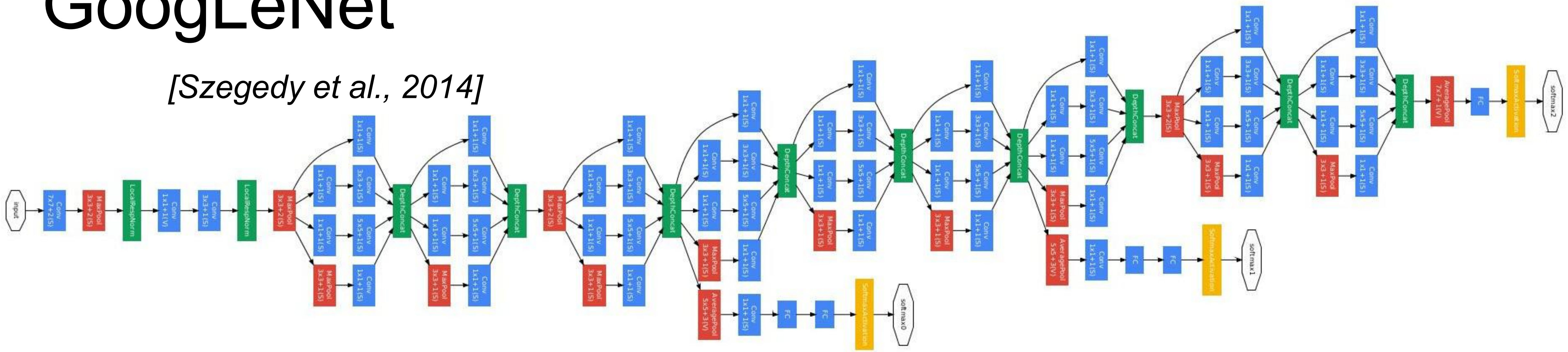
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] 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

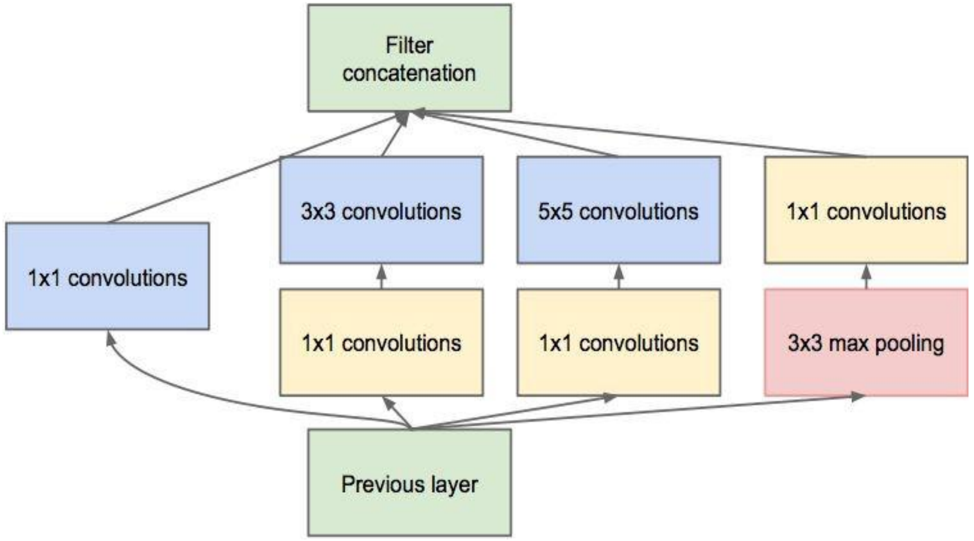
TOTAL params: 138M parameters

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

GoogLeNet

[Szegedy et al., 2014]





Why are there 1x1 convolutions? What are their effects?

Inception module

ILSVRC 2014 winner (6.7% top 5 error)

Fun features:

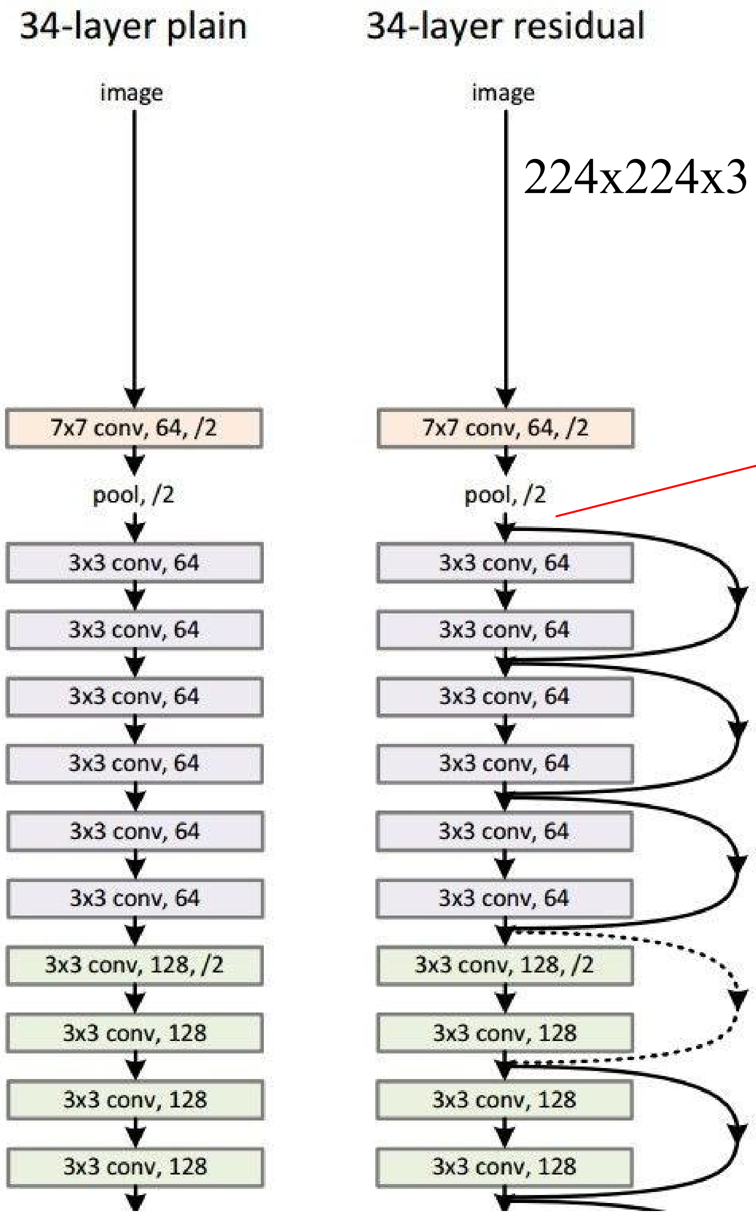
Only 5 million params! (Removes FC layers completely)

Compared to AlexNet:

- 12X less params
- 2x more compute
- 6.67% (vs. 15.4%)

ResNet

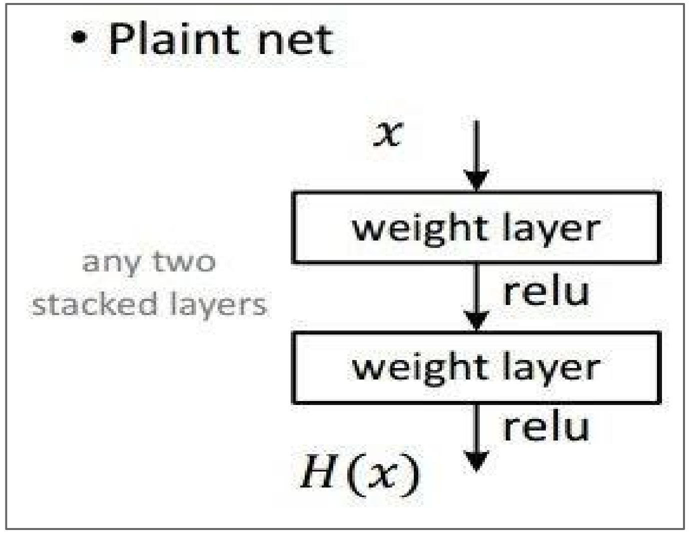
[He et al., 2015]

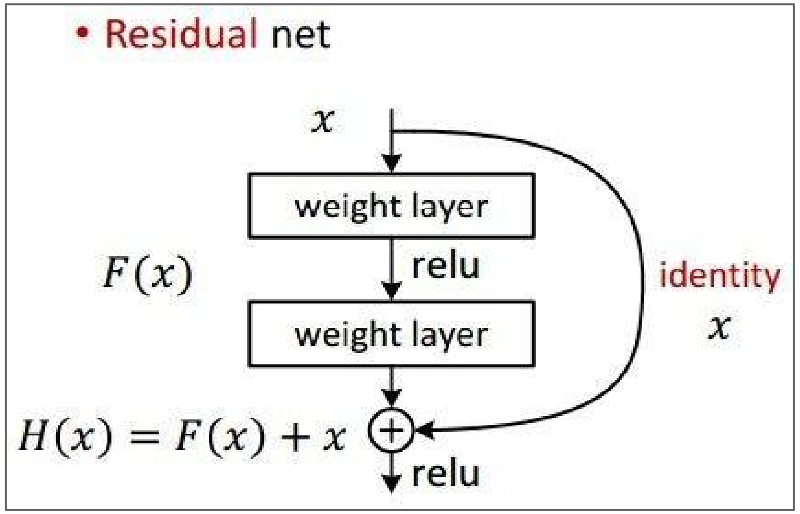


spatial dimension only 56x56!

ResNet

[He et al., 2015]





Two reasons to use skip conncetion:

- They mitigate the problem of vanishing gradient by allowing this alternate shortcut path for gradient to flow through.
- They allow the model to learn an identity function which ensures that the higher layer will perform at least as good as the lower layer, and not worse.

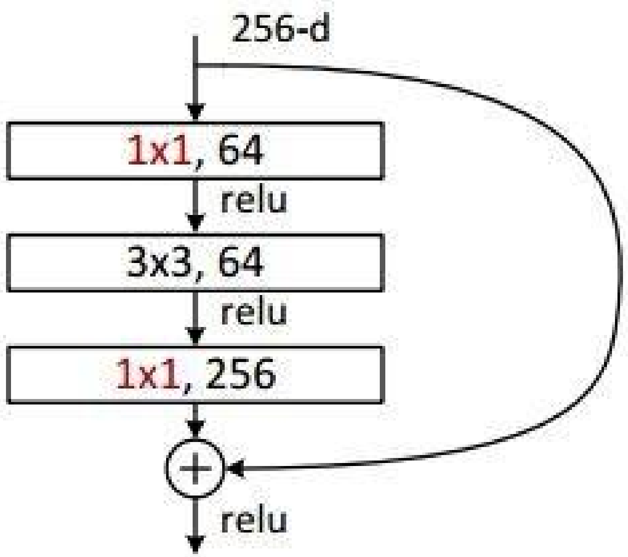
ResNet

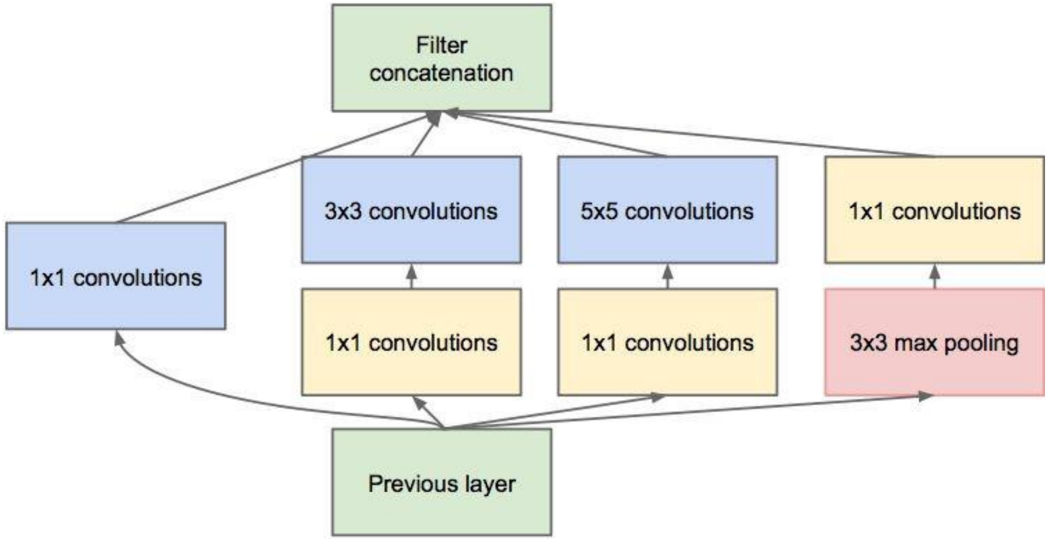
[He et al., 2015]

- 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

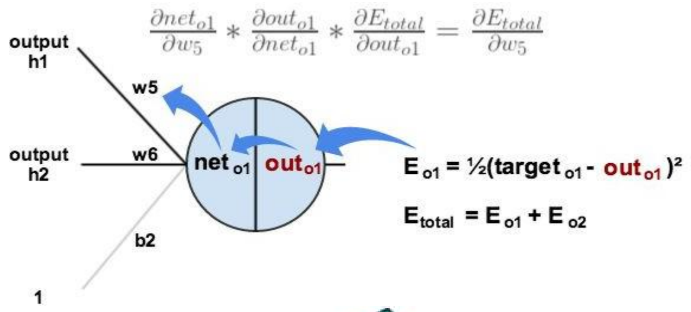
ResNet

[He et al., 2015]

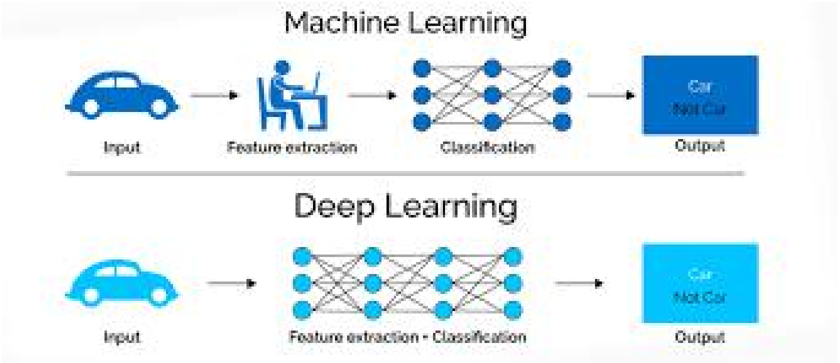




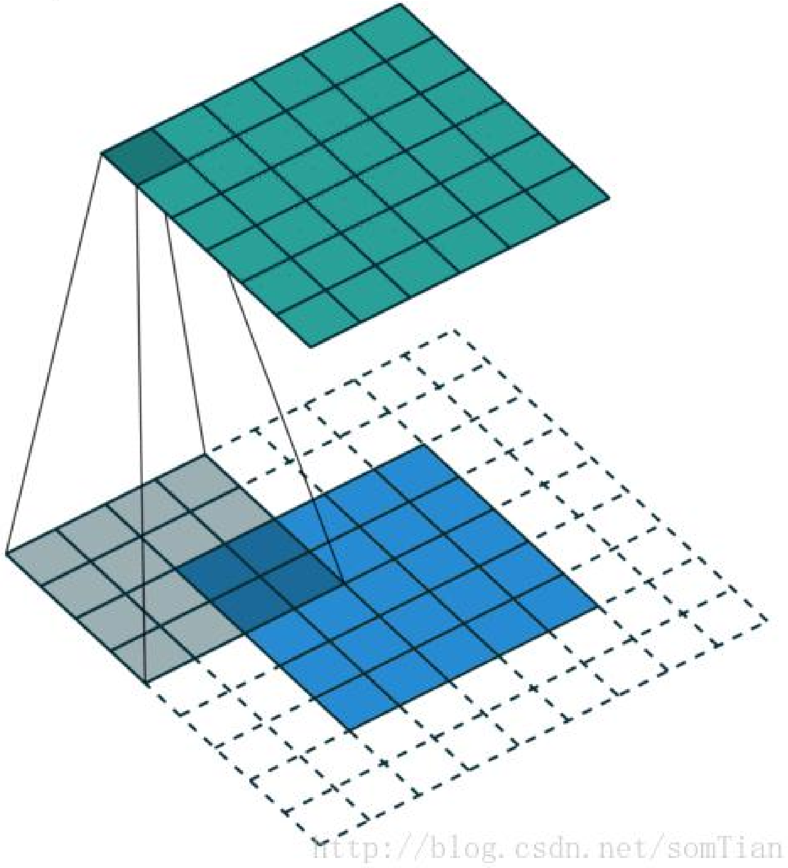
(this trick is also used in GoogLeNet)



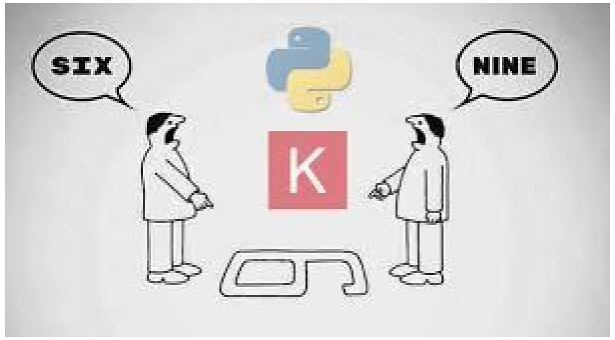
output h1
output h2
w5
w6
b2
1
net o1
out o1
$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$
$E_{o1} = ½(target_{o1} - out_{o1})^2$
$E_{total} = E_{o1} + E_{o2}$



Machine Learning
Input
Feature extraction
Classification
Output
Deep Learning
Input
Feature extraction + Classification
Output



http://blog.csdn.net/somTian



SIX
NINE
K

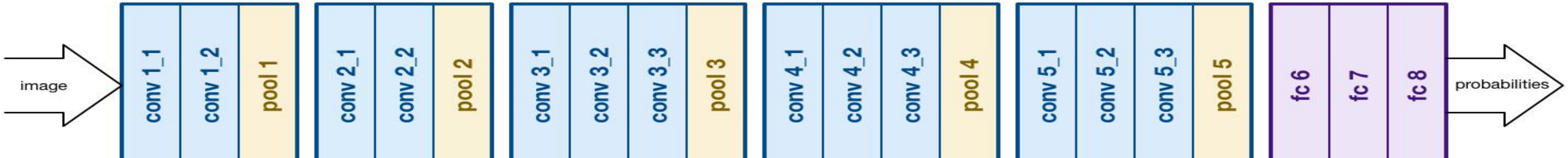


image
conv 1_1
conv 1_2
pool 1
conv 2_1
conv 2_2
pool 2
conv 3_1
conv 3_2
conv 3_3
pool 3
conv 4_1
conv 4_2
conv 4_3
pool 4
conv 5_1
conv 5_2
conv 5_3
pool 5
fc 6
fc 7
fc 8
probabilities

Programming Tasks

Prerequisites

- Python: 3.7+
- One of the following Deep Learning frameworks:
- PyTorch: http://pytorch.org/
- Tensorflow: https://www.tensorflow.org/install/
- Keras (https://keras.io/)

- Build a Convolutional Neural Network
- The main goal of this task is to build a simple Convolutional Neural Network and train it on the MNIST dataset.
- For this part of the task you need to develop the following Conv Net architec ture using the Deep Learning framework of your choice (the layers need to be implemented in the order specified):

- Conv Layer1: num_filters=64, activation=relu, padding=valid, strides=(2, 2), kernel_size=(3, 3),
- Conv Layer2: num_filters=32, kernel_size=(2, 2), activation=relu, padding=same, strides=(1, 1)
- Max Pool Layer: pool_size=(2, 2), strides=(1, 1)
- Dropout: rate=0.35 (Fraction of the input units to drop)
- Flatten
- Dense: num_units=256, activation=tanh
- Dropout: rate=0.5 (Fraction of the input units to drop)
- Dense: num_units=10, activation=softmax

- Train the above architecture on the MNIST dataset with the following optimizers for 10 epochs each and plot their corresponding loss curves.
- Use categorical Cross Entropy as the loss function.
- Optimizers
  - Adam
  - ADADELTA
  - Stochastic Gradient Descent

Successful implementation +5 points