Xi'an Jiaotong-Liverpool University

西交利物浦大學

INT305 Machine Learning

Lecture 6

Convolutional Neural Network

Sichen Liu

Department Intelligence Science

Sichen.Liu@xjtlu.edu.cn

The problem

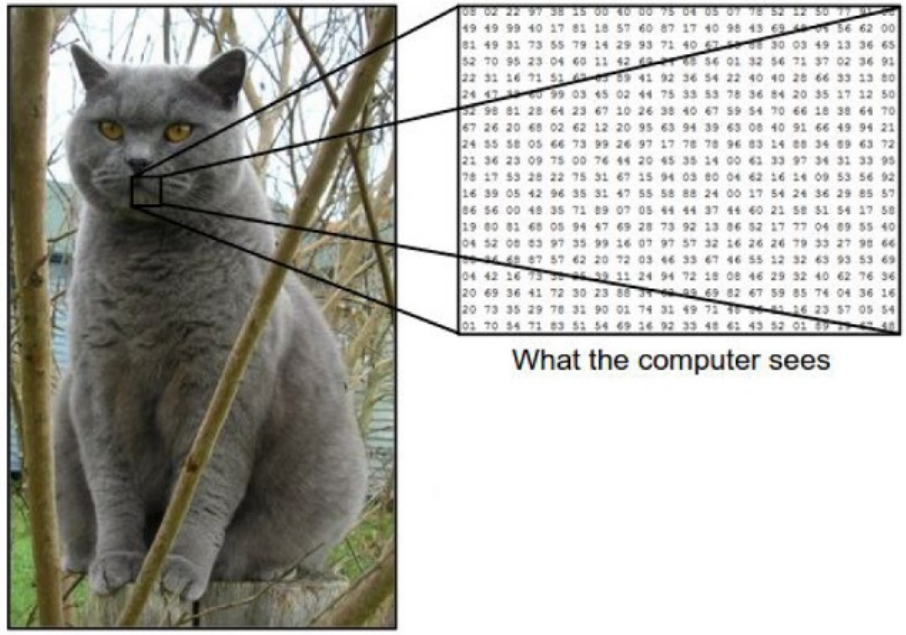
The problem:

semantic gap

Images are represented as 3D arrays of numbers, with integers between [0, 255].

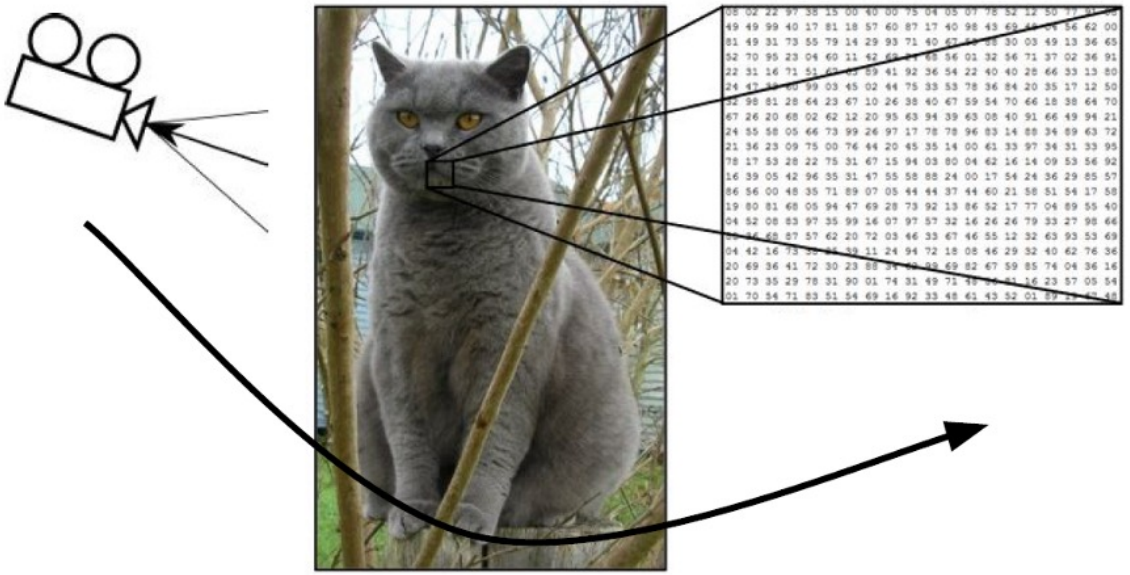
E.g.,
300 x 100 x 3

(3 for 3 color channels RGB)



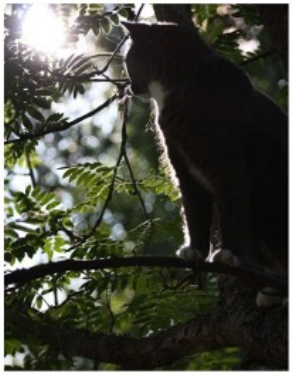
Challenges (1)

Challenges: Viewpoint Variation



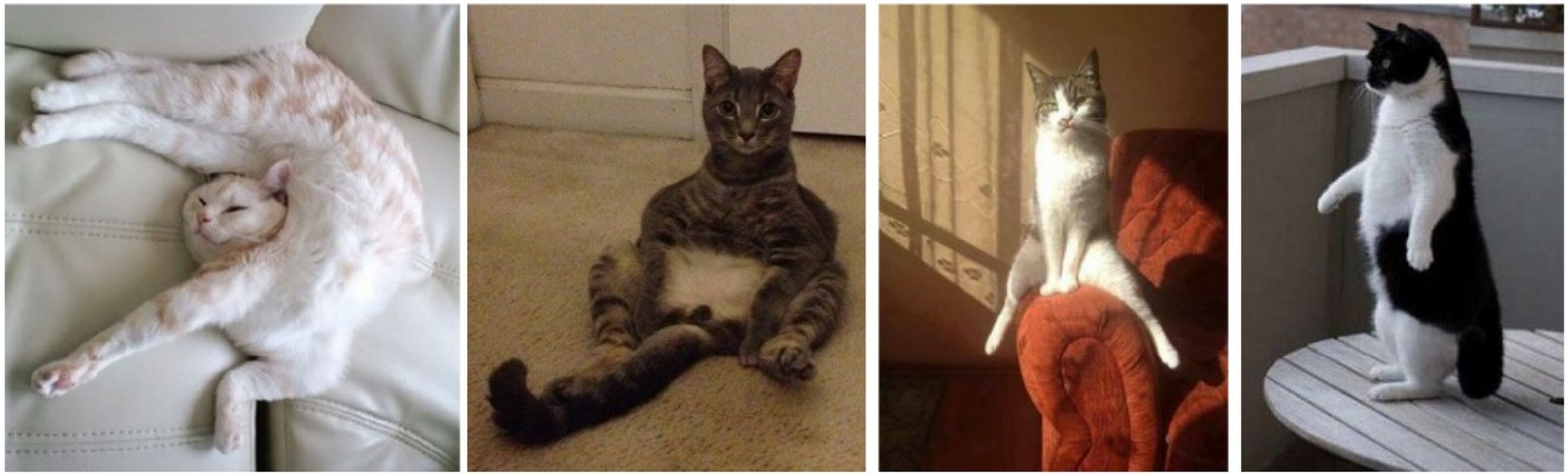
Challenges (2)

Challenges: Illumination



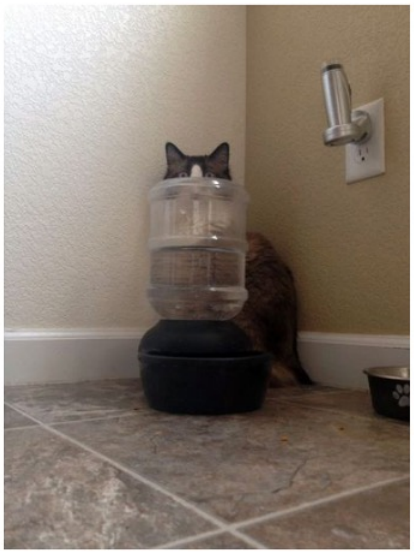
Challenges (3)

Challenges: Deformation



Challenges (4)

Challenges: Occlusion


Challenges (5)

Challenges: Background Clutter

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

Challenges (6)

Challenges: Intraclass variation

An image classifier

An image classifier

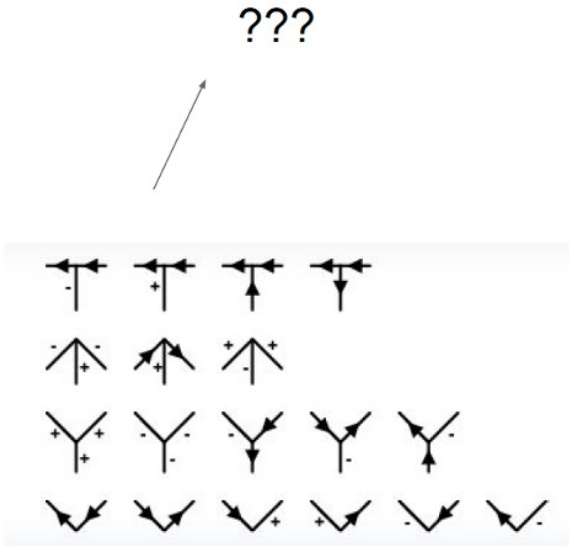
```
def predict(image):
    # ????
    return class_label
```

Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

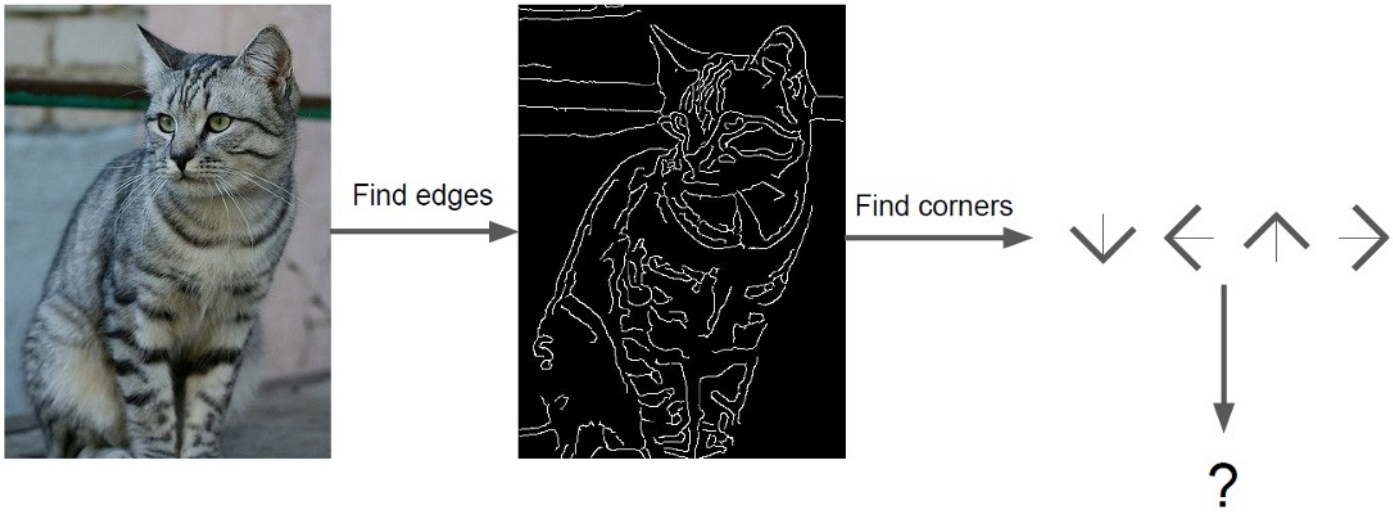
Attempts

Attempts have been made



Attempts

Attempts have been made



Data-driven approach

Data-driven approach:

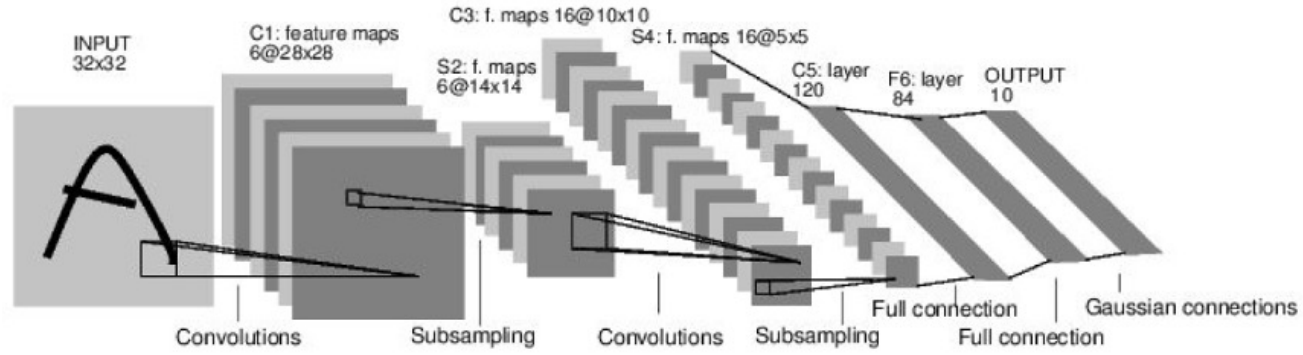
1. Collect a dataset of images and labels
2. Use Machine Learning to train an image classifier
3. Evaluate the classifier on a withheld set of test images

```
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

CNN

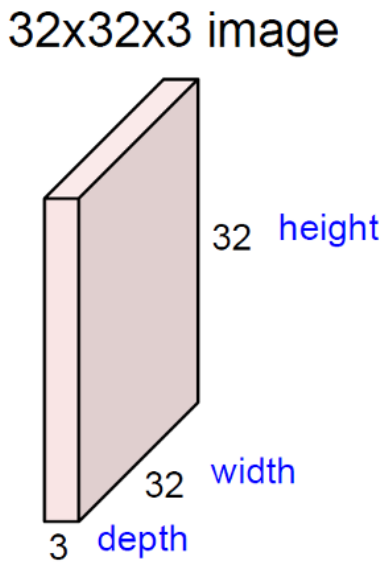
Convolutional Neural Networks



[LeNet-5, LeCun 1980]

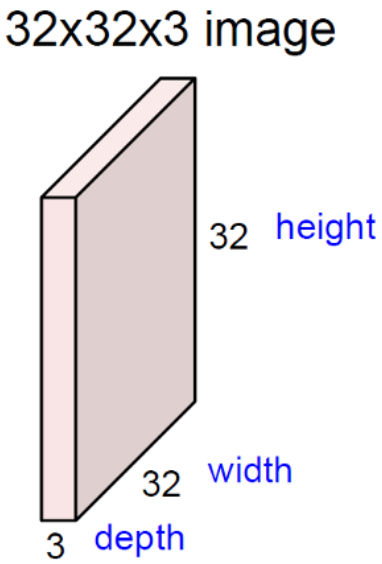
CNN

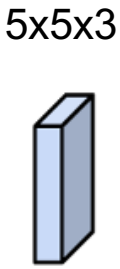
Convolution Layer



CNN

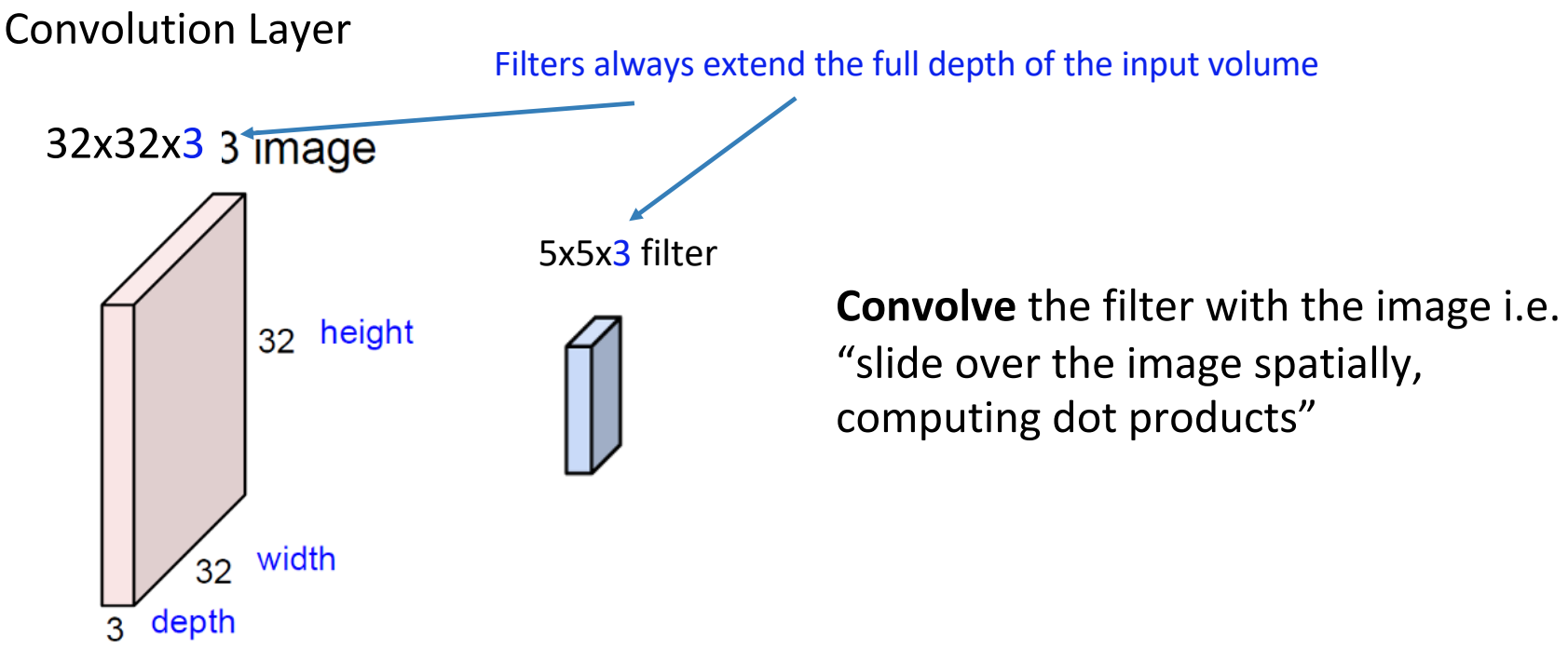
Convolution Layer





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

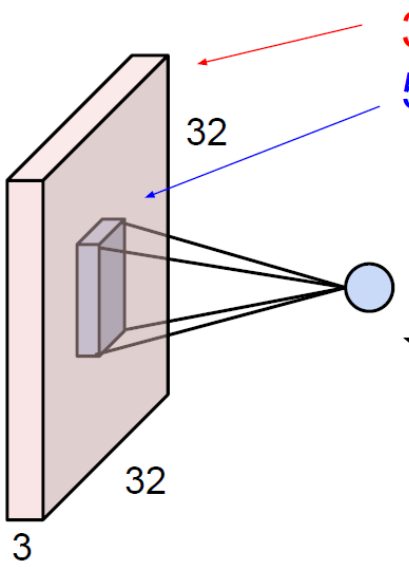
CNN



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

CNN

Convolution Layer



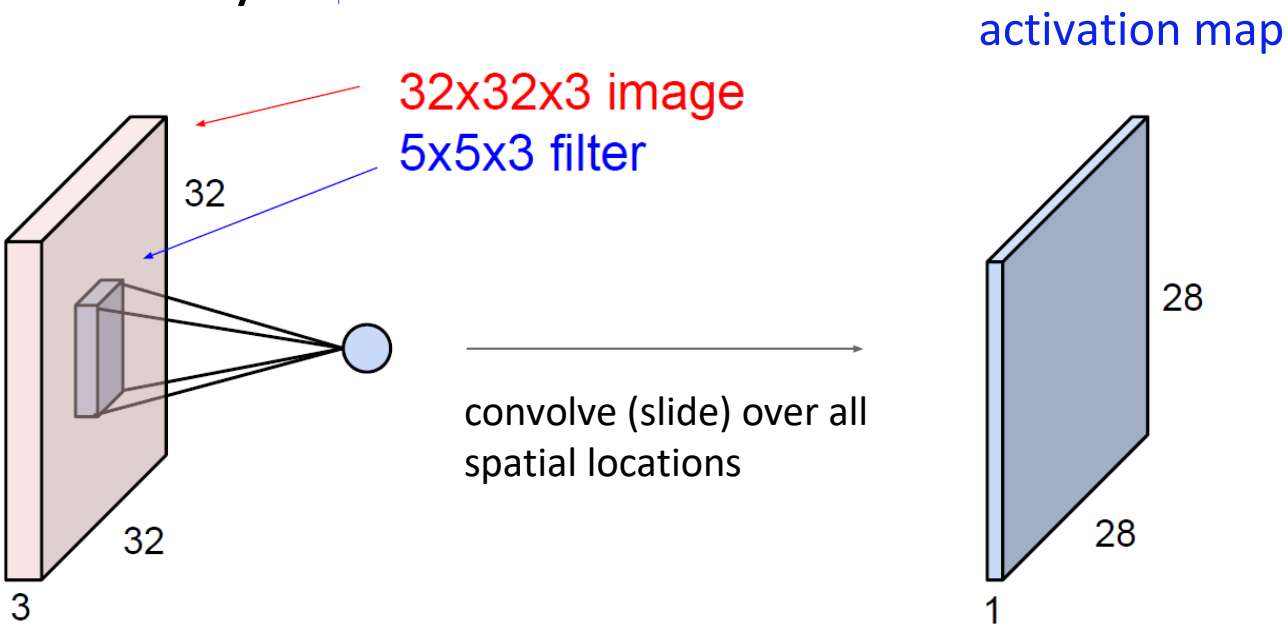
1 number:

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)

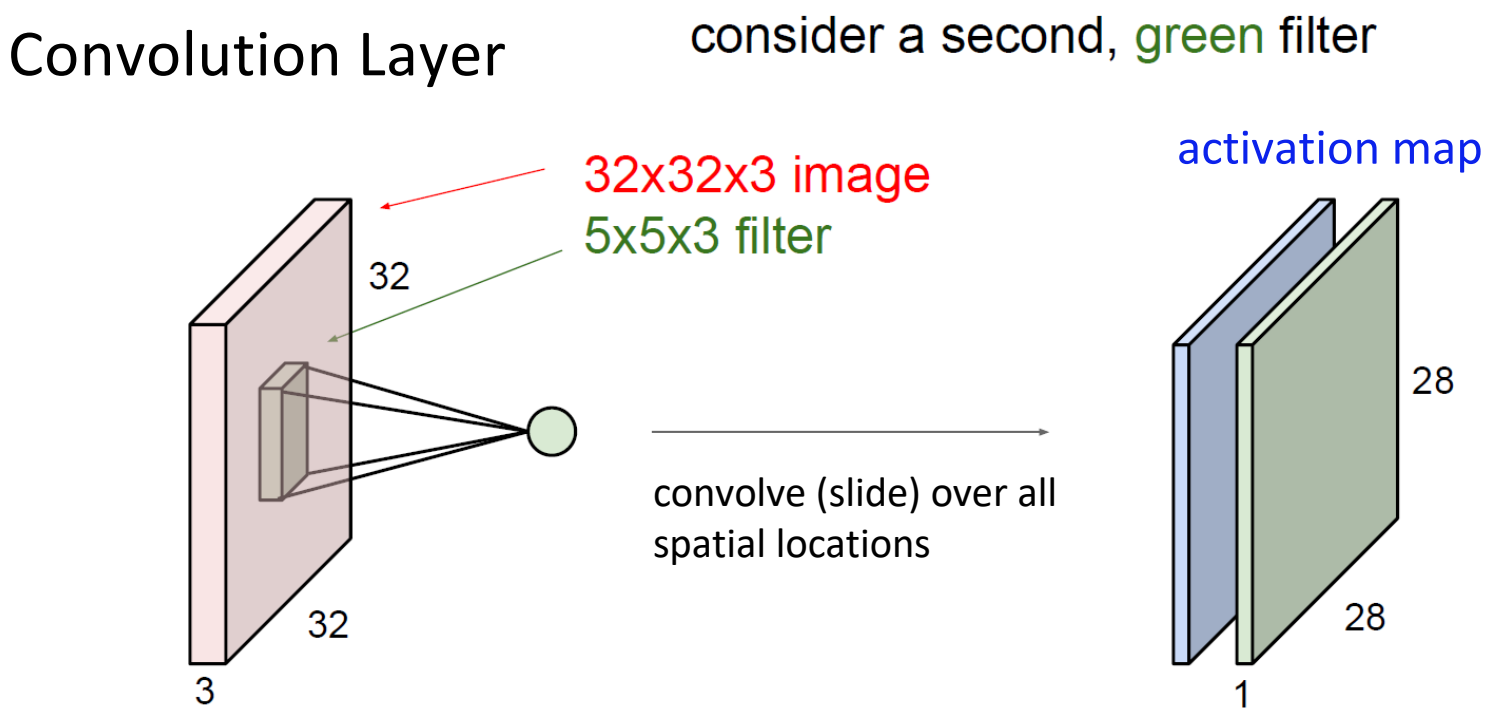
$$w^T x + b$$

CNN

Convolution Layer

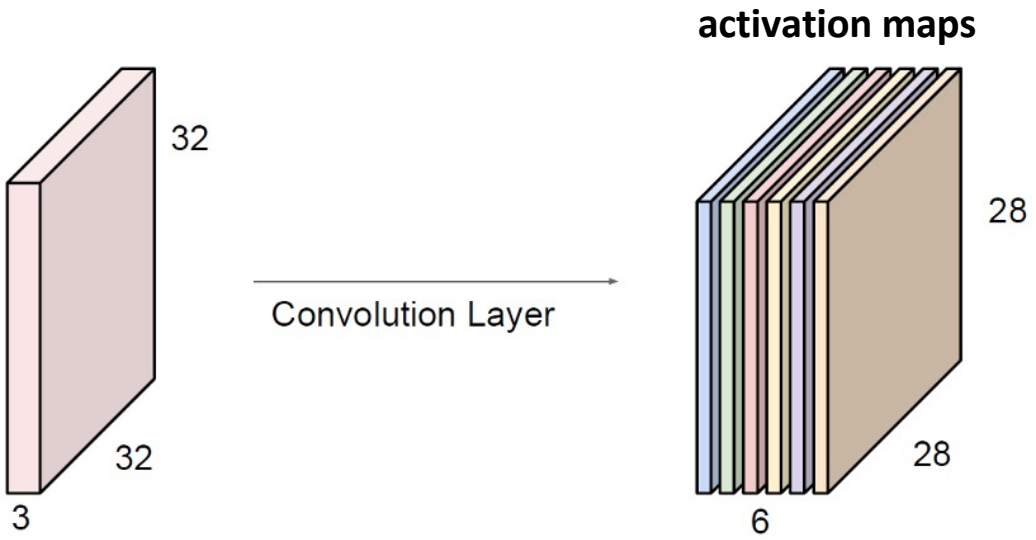


CNN


Convolution Layer
consider a second, green filter
32x32x3 image
5x5x3 filter
activation map
32
32
3
convolve (slide) over all
spatial locations
28
28
1

CNN

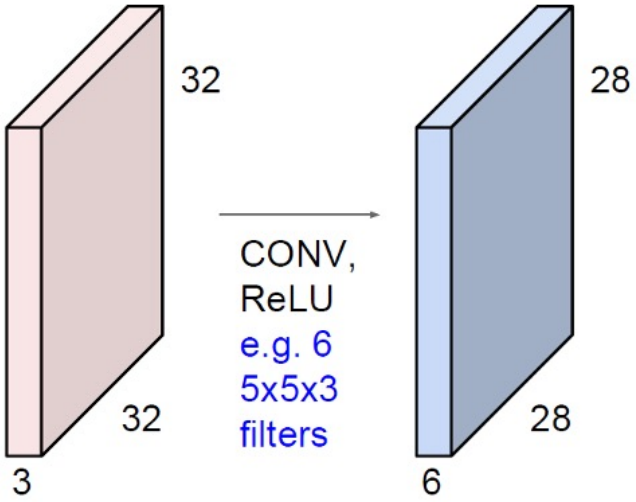
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

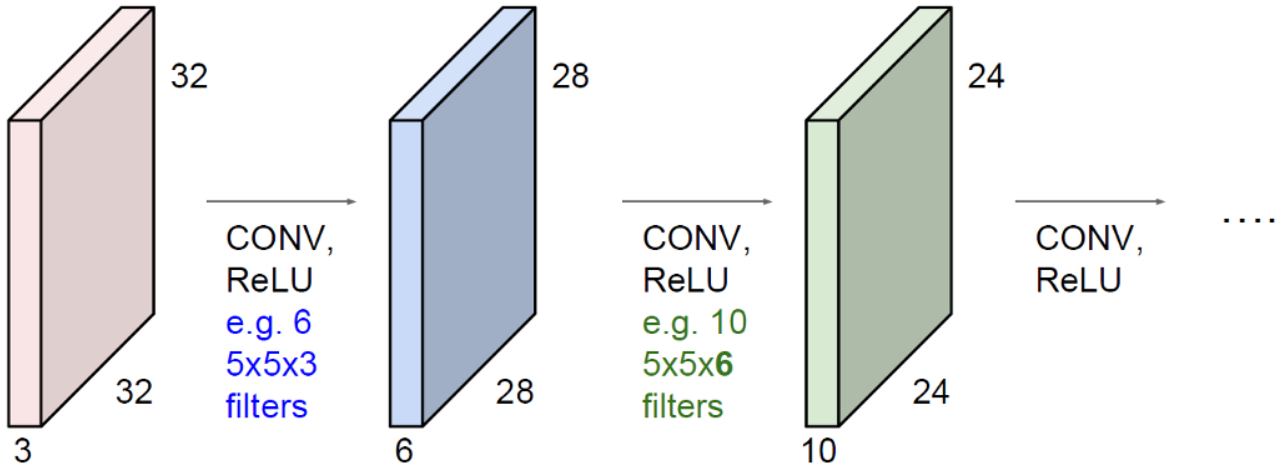
CNN

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



CNN

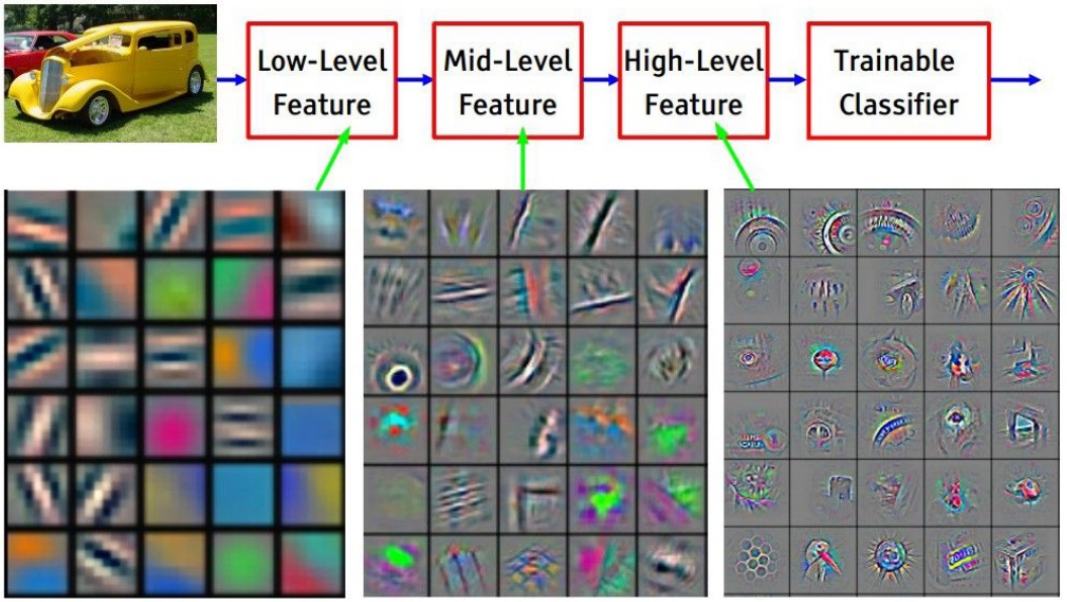
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



CNN

Preview

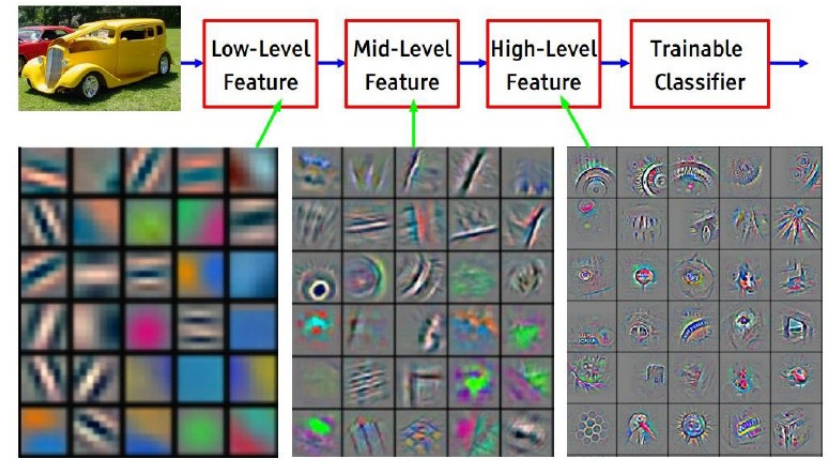
[From recent Yann LeCun slides]



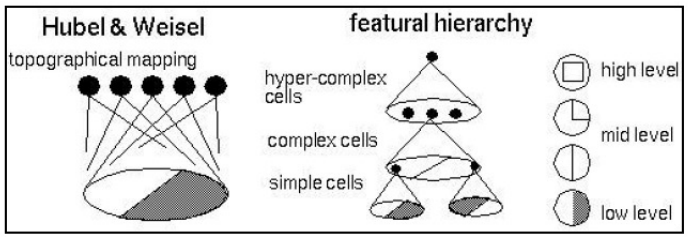
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

CNN

Preview

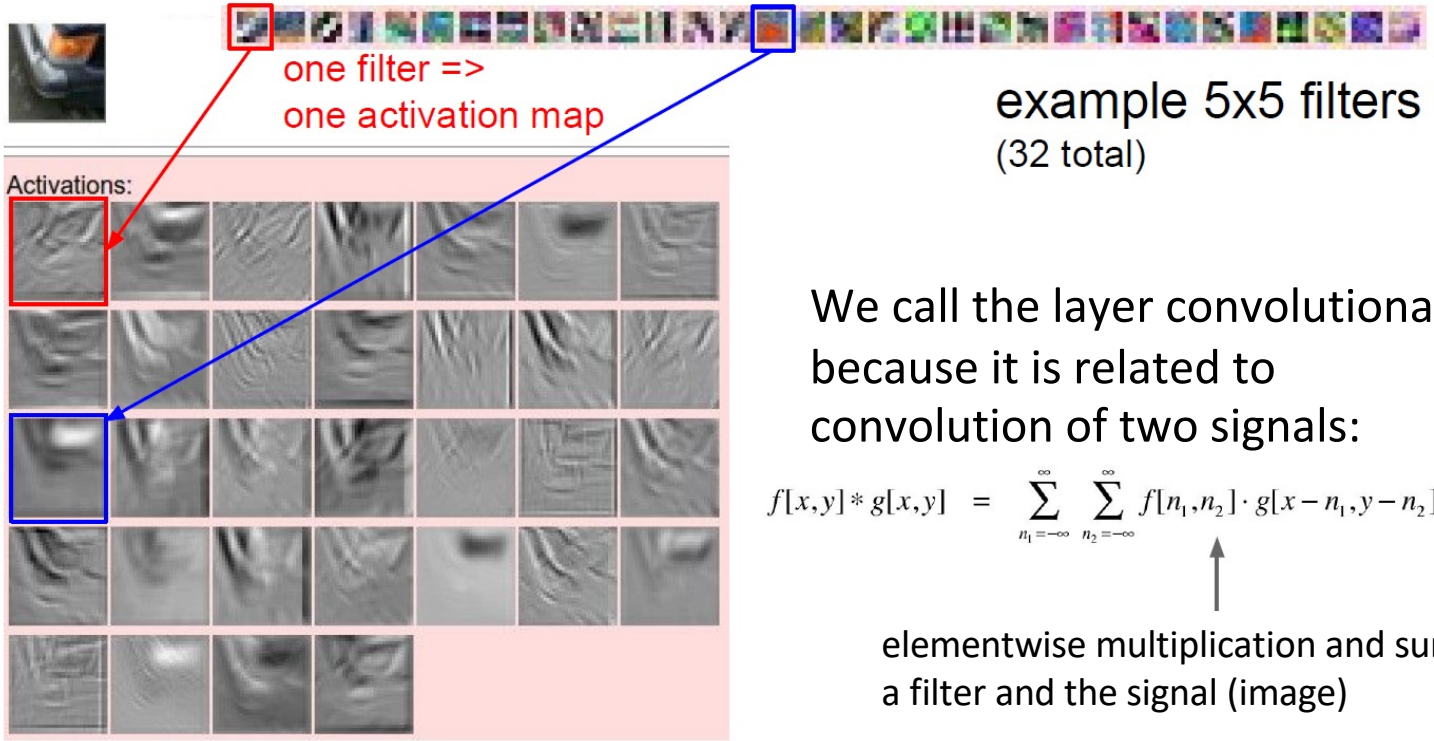


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]



[From recent Yann LeCun slides]

CNN



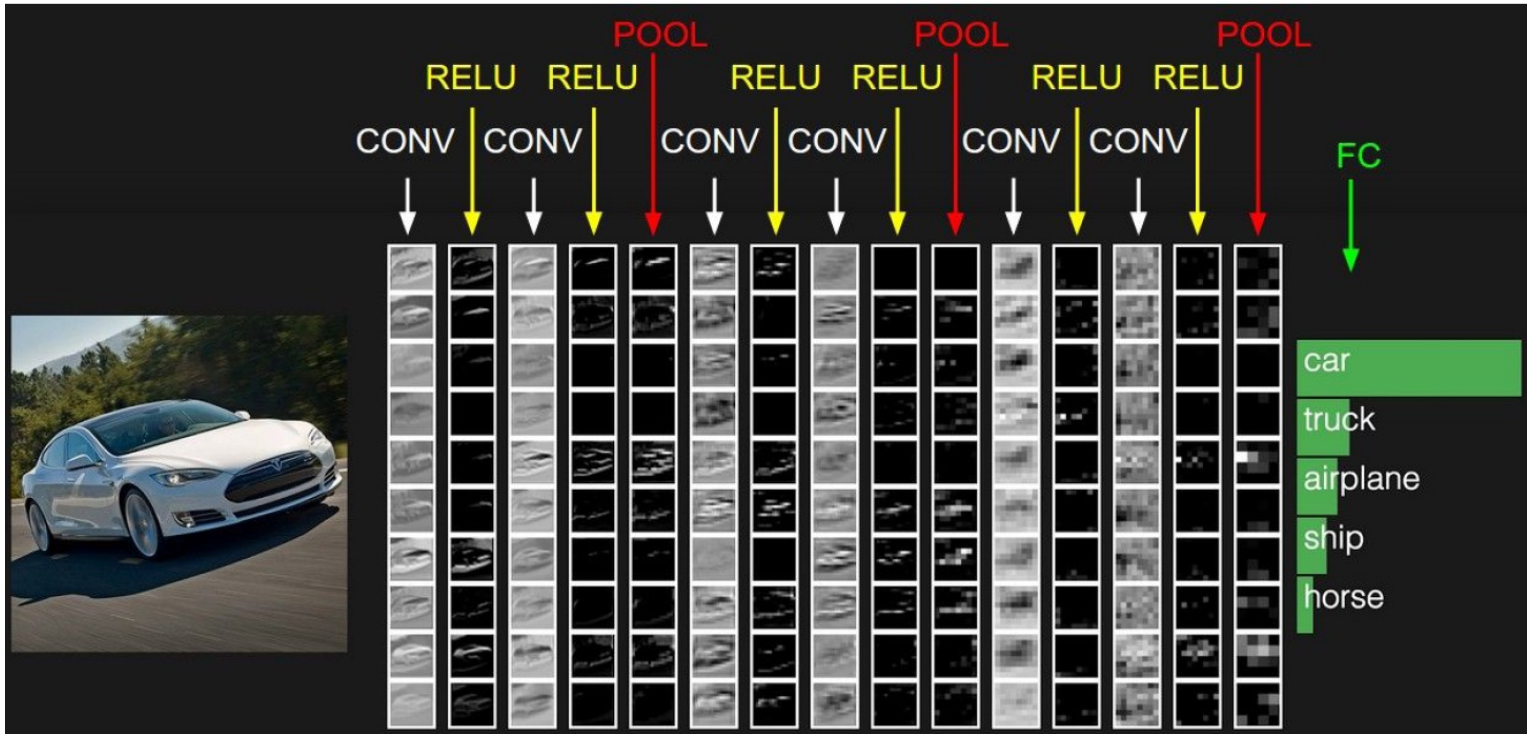
We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

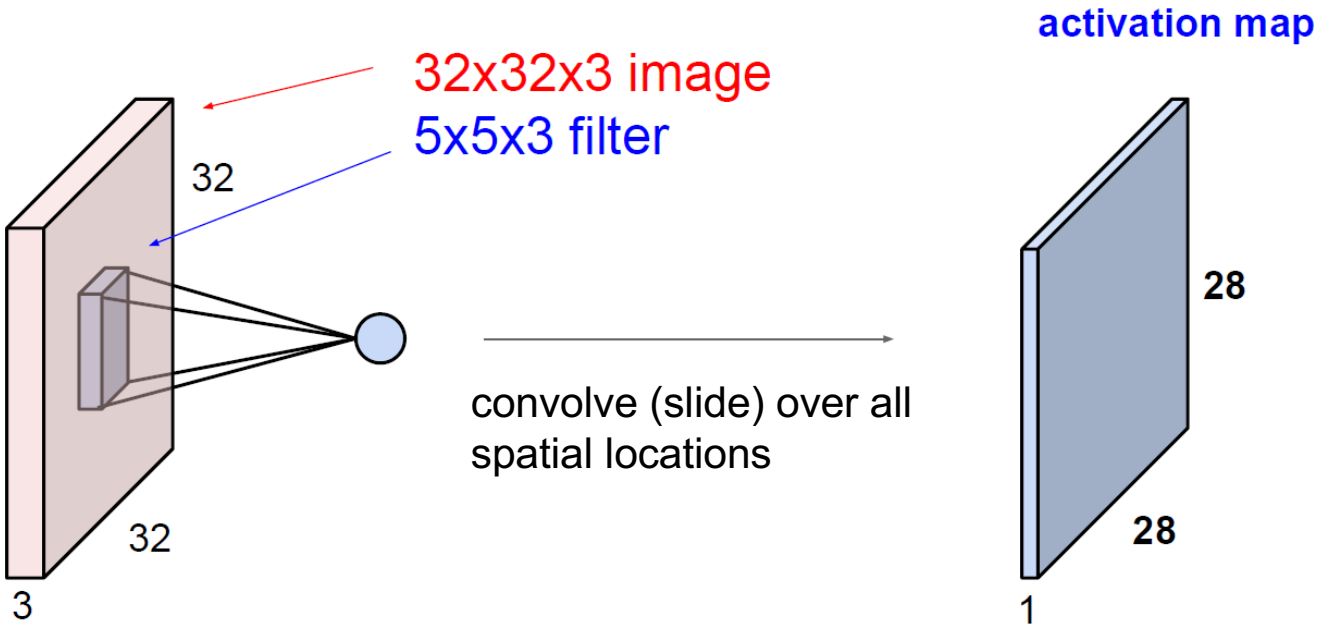
CNN

Preview:


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

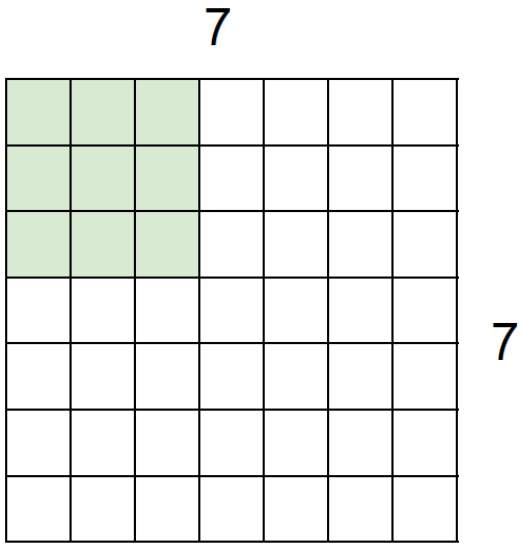
CNN

A closer look at spatial dimensions:



CNN

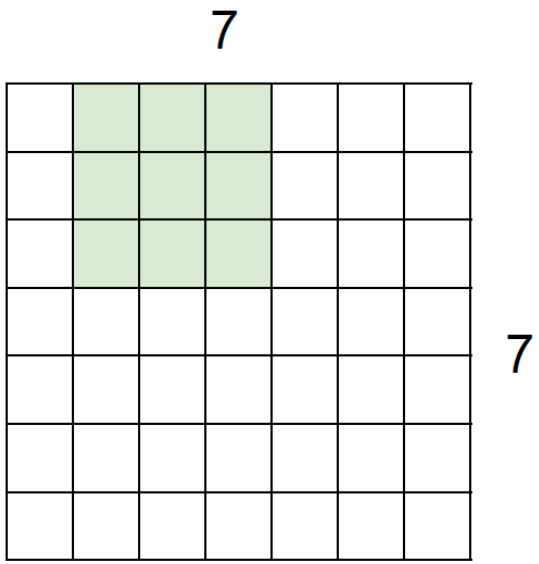
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

CNN

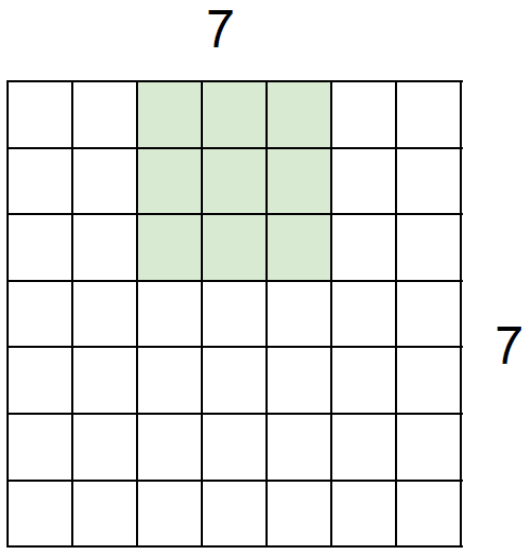
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

CNN

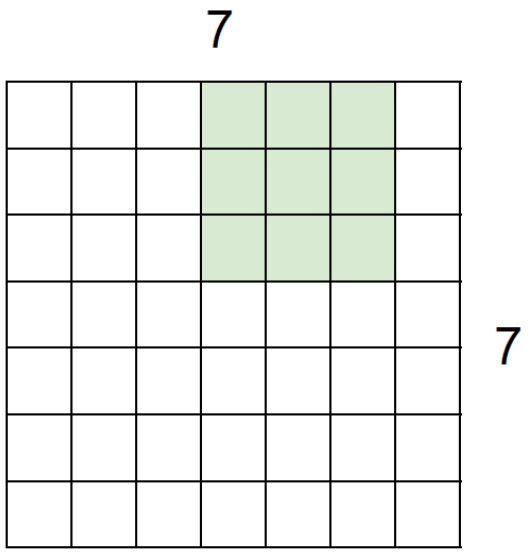
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

CNN

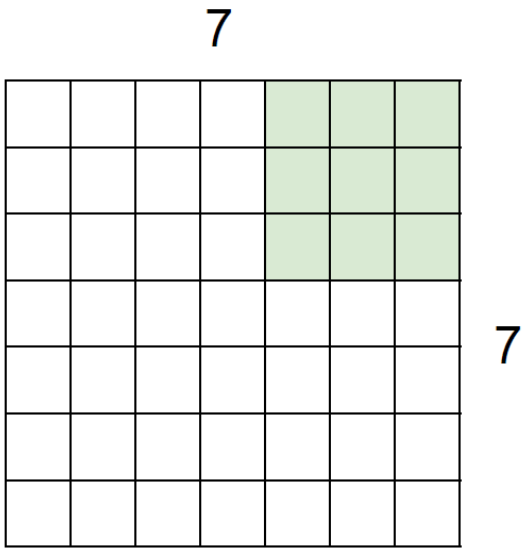
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

CNN

A closer look at spatial dimensions:

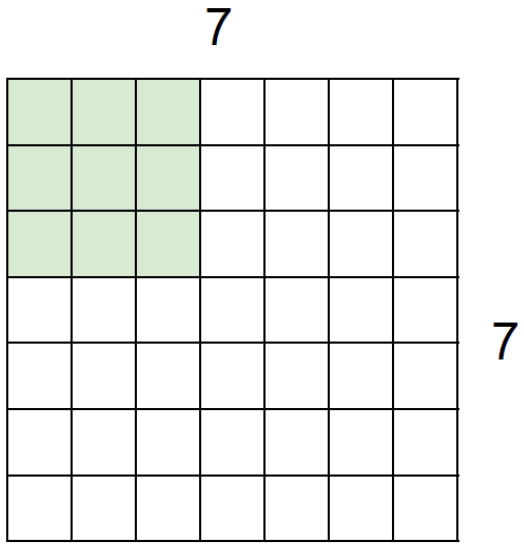


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

CNN

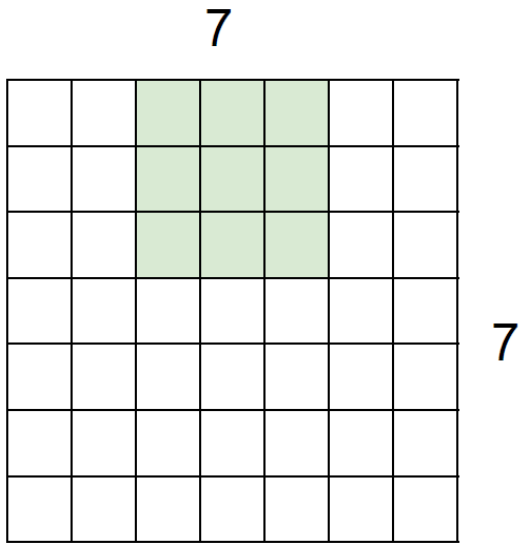
A closer look at spatial dimensions:



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

CNN

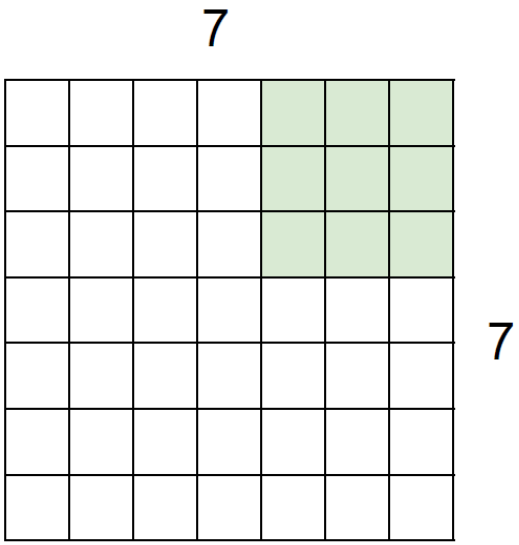
A closer look at spatial dimensions:



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

CNN

A closer look at spatial dimensions:

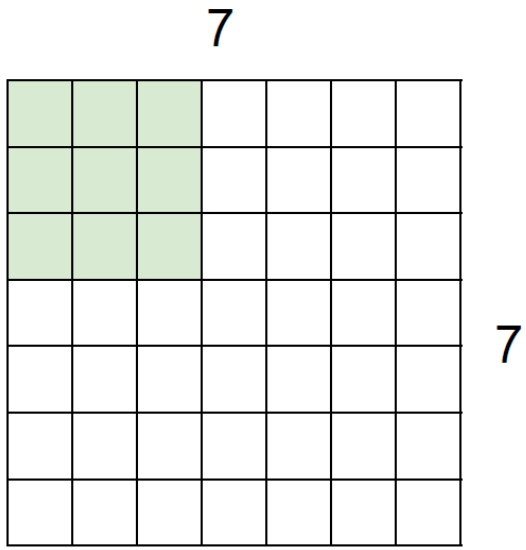


7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

=> 3x3 output!

CNN

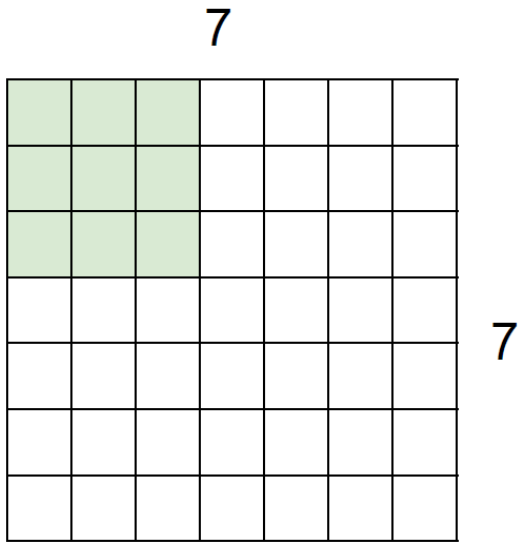
A closer look at spatial dimensions:



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

CNN

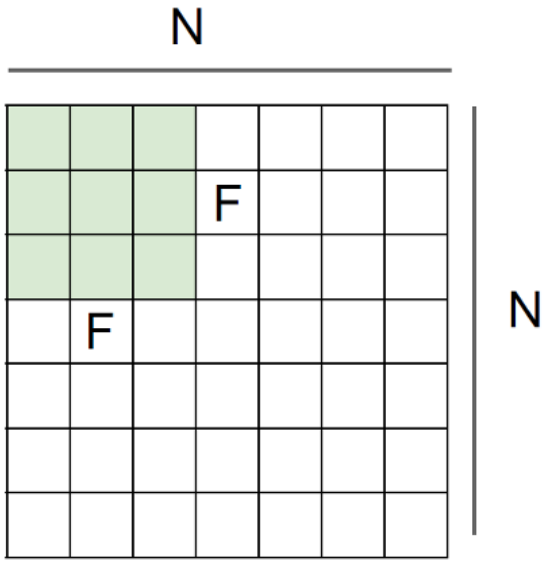
A closer look at spatial dimensions:



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

doesn‘t fit!
Cannot apply 3x3 filter on 7x7 input with stride 3.

CNN

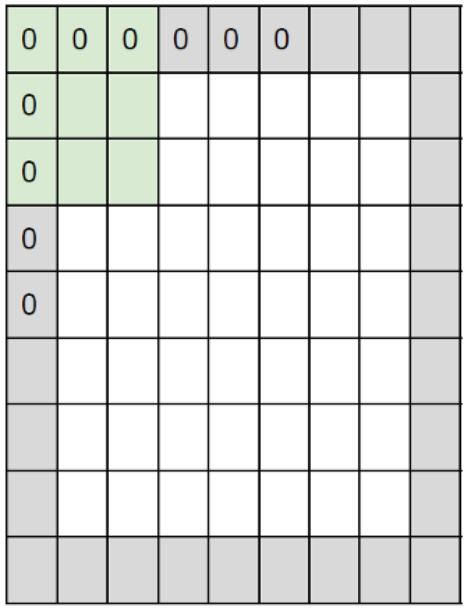


Output size:
(N - F)/stride +1

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 :\

CNN

In practice: Common to zero pad the border

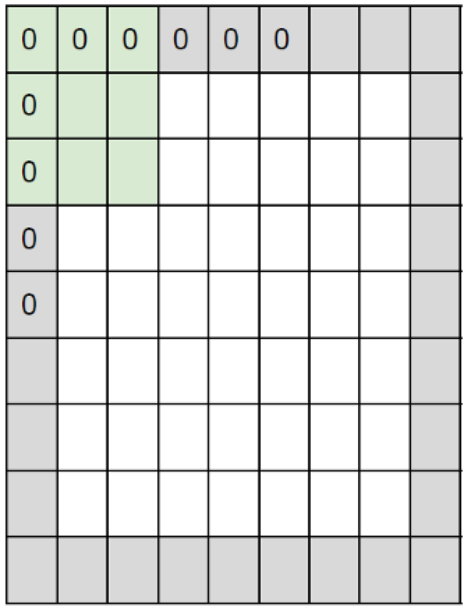


e.g., input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

CNN

In practice: Common to zero pad the border

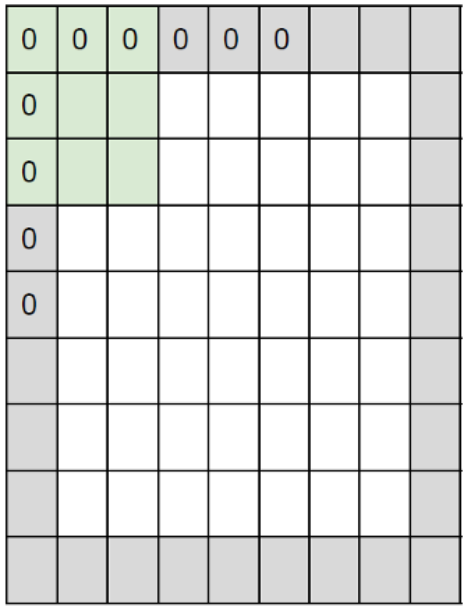


e.g., input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

CNN

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!

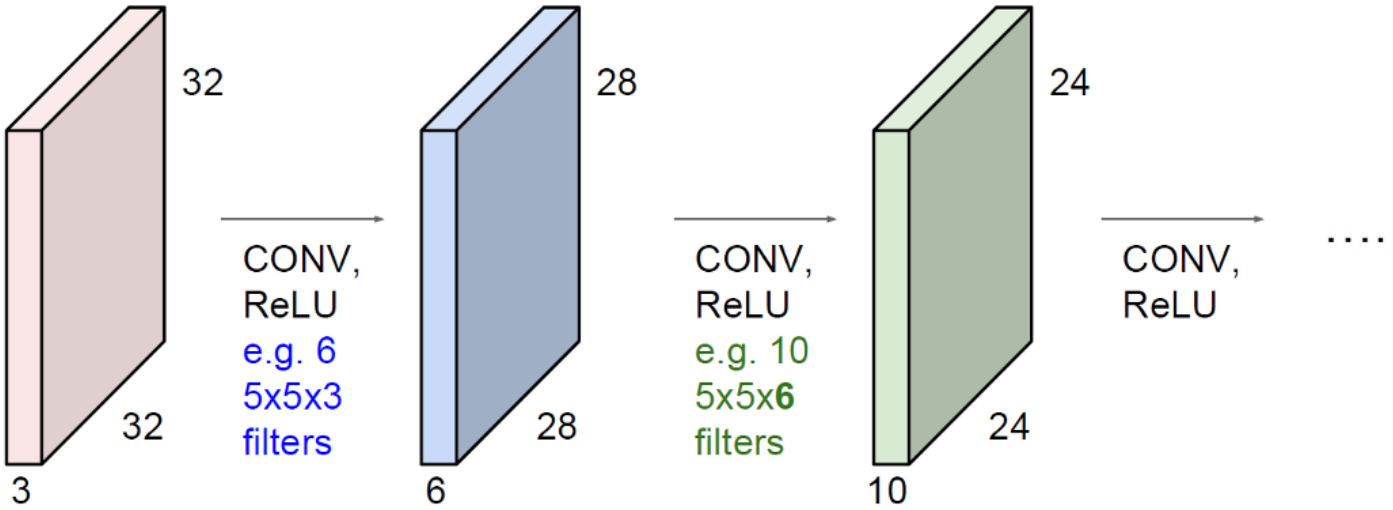
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)

e.g., F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

CNN

Remember back to …

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 …). Shrinking too fast is not good, doesn't work well.

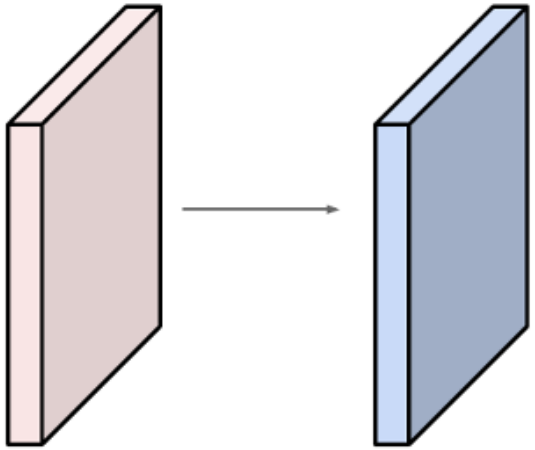


CNN

Examples time:

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

Output volume size:?

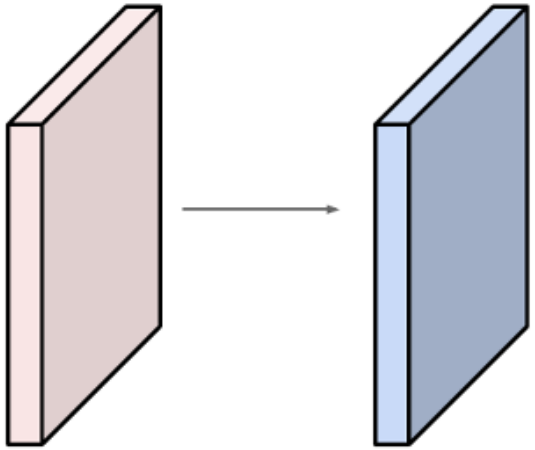


CNN

Examples time:

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

Output volume size:
(32+2*2-5)/1+1=32 spatially, so
32x32x10

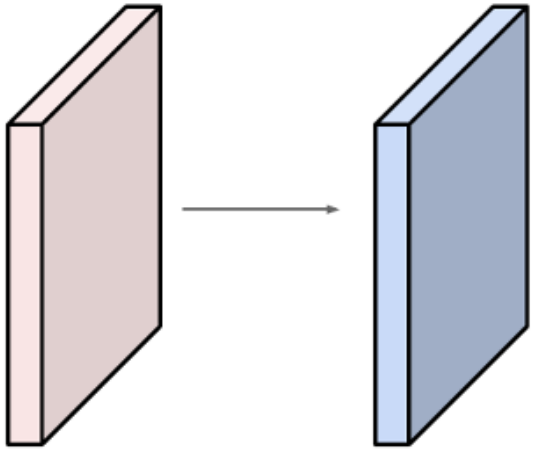


CNN

Examples time:

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

Number of parameters in this layer?

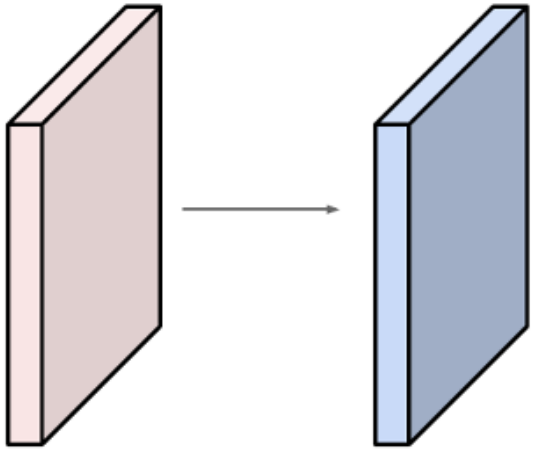


CNN

Examples time:

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
=> 76*10=760



(+1 for bias)

CNN

Summary. To summarize, the Conv layer:

- Accepts a volume of size $\boldsymbol{W_1 \times H_1 \times D_1}$
- Requires four hyperparameters:
  - Number of filters $\boldsymbol{K}$,
  - Their spatial extent $\boldsymbol{F}$,
  - The stride $\boldsymbol{S}$,
  - The amount of zero padding $\boldsymbol{P}$.

- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $\boldsymbol{W_2 = (W_1 - F + 2P)/S + 1}$
  - $\boldsymbol{H_2 = (H_1 - F + 2P)/S + 1}$ (i.e. width and height are computed equally by symmetry)
  - $\boldsymbol{D_2 = K}$

- With parameter sharing, it introduces $\boldsymbol{F \cdot F \cdot D_1}$ weights per filter, for a total of $\boldsymbol{(F \cdot F \cdot D_1) \cdot K}$ weights and $\boldsymbol{K}$ biases.
- In the output volume, the $\boldsymbol{d}$-th depth slice (of size $\boldsymbol{W_2 \times H_2}$) is the result of performing a valid convolution of the $\boldsymbol{d}$-th filter over the input volume with a stride of $\boldsymbol{S}$, and then offset by $\boldsymbol{d}$-th bias.

CNN

Summary. To summarize, the Conv layer:

- Accepts a volume of size $\boldsymbol{W_1 \times H_1 \times D_1}$
- Requires four hyperparameters:
  - Number of filters $\boldsymbol{K}$,
  - Their spatial extent $\boldsymbol{F}$,
  - The stride $\boldsymbol{S}$,
  - The amount of zero padding $\boldsymbol{P}$.

Common settings:

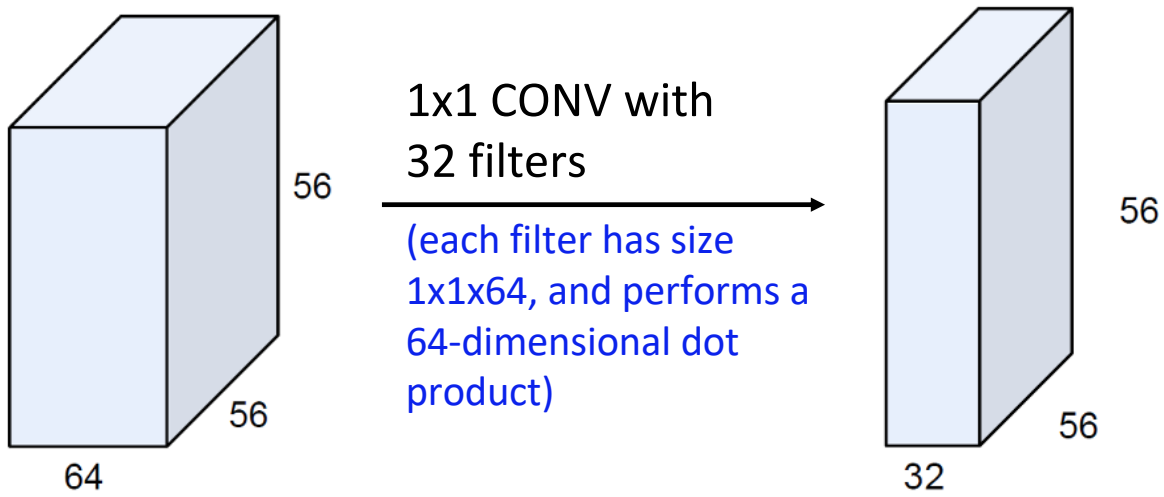
K = (powers of 2, e.g., 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $\boldsymbol{W_2 = (W_1 - F + 2P)/S + 1}$
  - $\boldsymbol{H_2 = (H_1 - F + 2P)/S + 1}$ (i.e. width and height are computed equally by symmetry)
  - $\boldsymbol{D_2 = K}$

- With parameter sharing, it introduces $\boldsymbol{F \cdot F \cdot D_1}$ weights per filter, for a total of $\boldsymbol{(F \cdot F \cdot D_1) \cdot K}$ weights and $\boldsymbol{K}$ biases.
- In the output volume, the $\boldsymbol{d}$-th depth slice (of size $\boldsymbol{W_2 \times H_2}$) is the result of performing a valid convolution of the $\boldsymbol{d}$-th filter over the input volume with a stride of $\boldsymbol{S}$, and then offset by $\boldsymbol{d}$-th bias.

CNN

(btw, 1x1 convolution layers make perfect sense)



CNN

Example: CONV layer in Torch

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.

SpatialConvolution

```
module = nn.SpatialConvolution(nInputPlane, nOutputPlane, kW, kH, [dW], [dH], [padW], [padH])
```

Applies a 2D convolution over an input image composed of several input planes. The `input` tensor in `forward(input)` is expected to be a 3D tensor (`nInputPlane x height x width`).

The parameters are the following:

- `nInputPlane` : The number of expected input planes in the image given into `forward()` .
- `nOutputPlane` : The number of output planes the convolution layer will produce.
- `kW` : The kernel width of the convolution
- `kH` : The kernel height of the convolution
- `dW` : The step of the convolution in the width dimension. Default is `1` .
- `dH` : The step of the convolution in the height dimension. Default is `1` .
- `padW` : The additional zeros added per width to the input planes. Default is `0` , a good number is `(kW-1)/2` .
- `padH` : The additional zeros added per height to the input planes. Default is `padW` , a good number is `(kH-1)/2` .

Note that depending of the size of your kernel, several (of the last) columns or rows of the input image might be lost. It is up to the user to add proper padding in images.

If the input image is a 3D tensor `nInputPlane x height x width` , the output image size will be `nOutputPlane x oheight x owidth` where

```
owidth  = floor((width  + 2*padW - kW) / dW + 1)
oheight = floor((height + 2*padH - kH) / dH + 1)
```

CNN

Example: CONV layer in Caffe

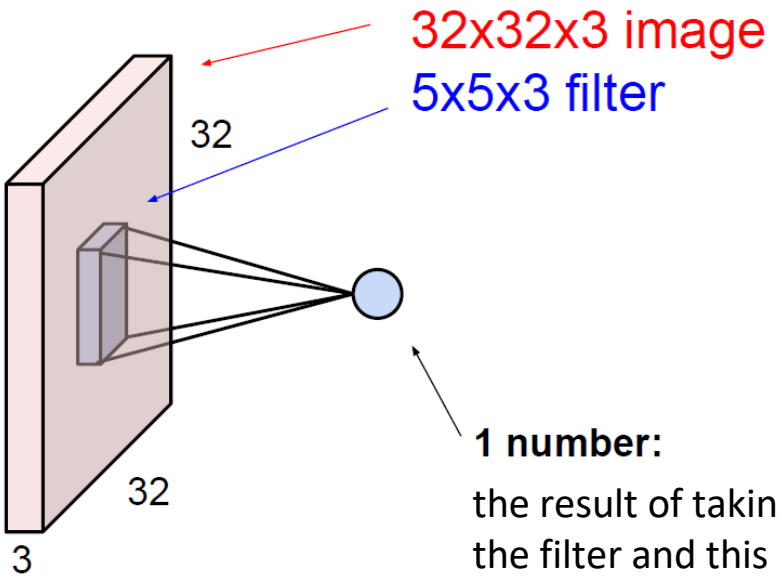
```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }
  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96     # learn 96 filters
    kernel_size: 11    # each filter is 11x11
    stride: 4          # step 4 pixels between each filter application
    weight_filler {
      type: "gaussian" # initialize the filters from a Gaussian
      std: 0.01        # distribution with stdev 0.01 (default mean: 0)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 0
    }
  }
}
```

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.

CNN

The brain/neuron view of CONV Layer

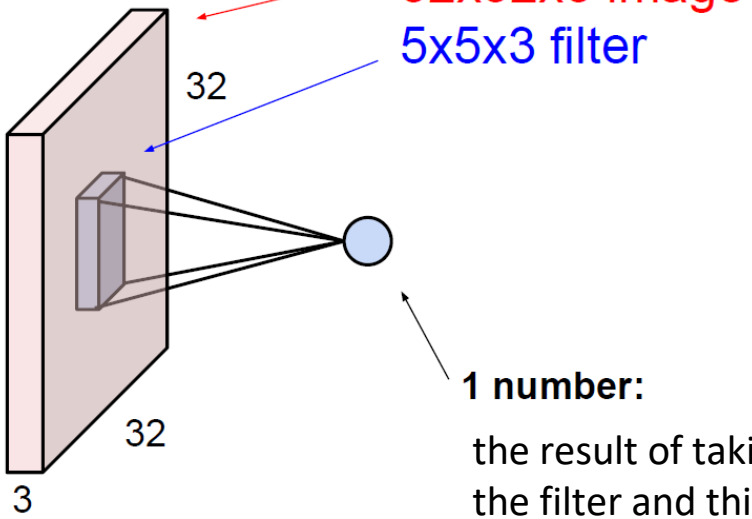


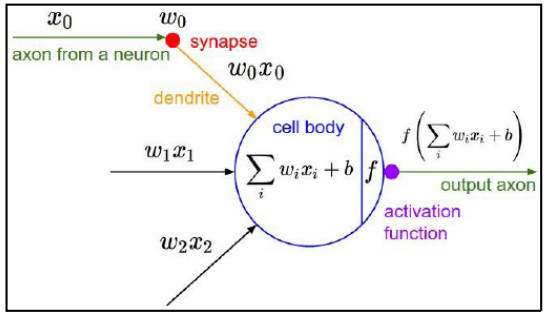
1 number:

the result of taking a dot product between the filter and this part of the image (i.e. 5*5*3 = 75-dimensional dot product)

CNN

The brain/neuron view of CONV Layer





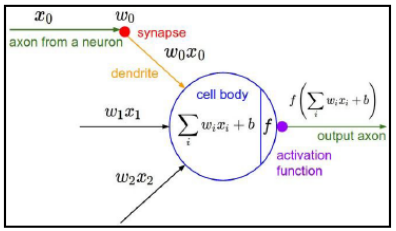
It's just a neuron with local connectivity…

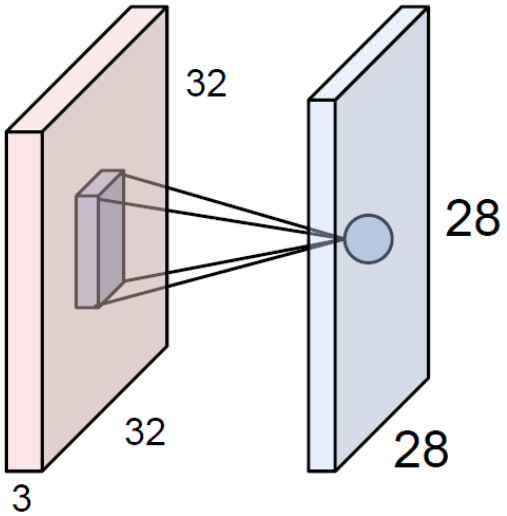
1 number:

the result of taking a dot product between the filter and this part of the image (i.e. 5*5*3 = 75-dimensional dot product)

CNN

The brain/neuron view of CONV Layer





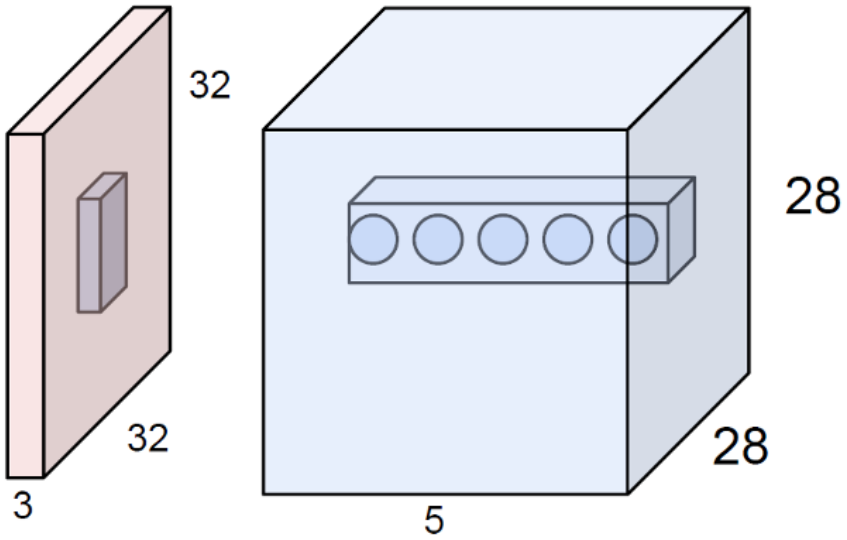
An activation map is a 28x28 sheet of neuron outputs:

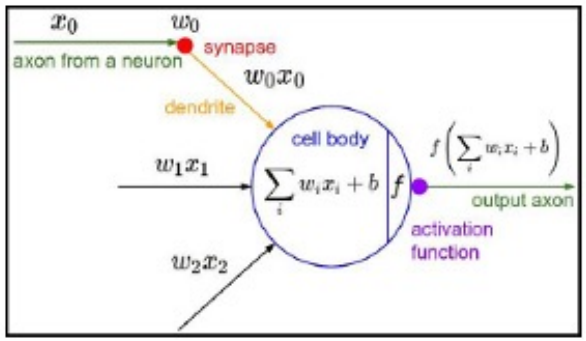
1. Each is connected to a small region in the input
2. All of them share parameters

"5x5 filter" -> "5x5 receptive field for each neuron"

CNN

The brain/neuron view of CONV Layer



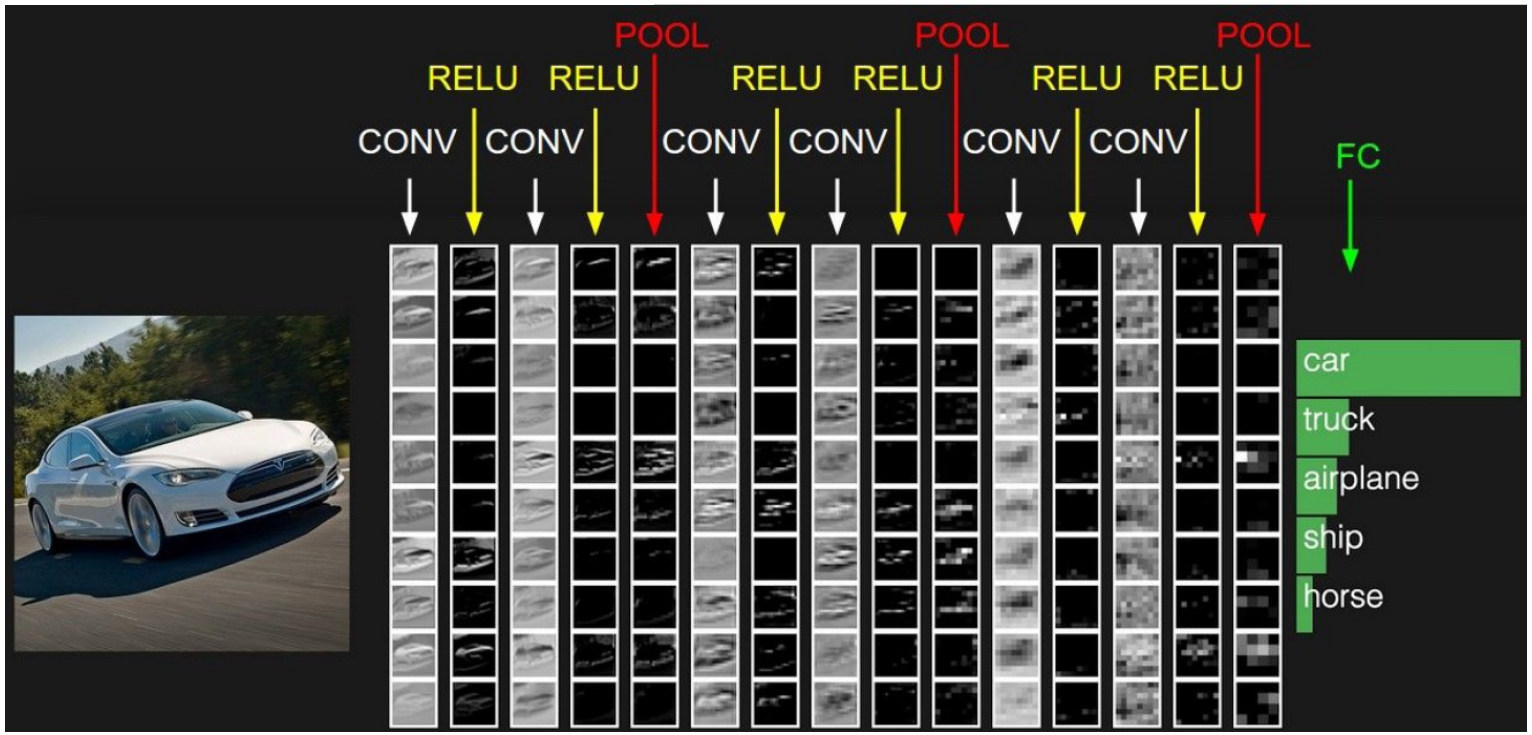


E.g., with 5 filters, CONV layer consists of neurons arranged in a 3D grid (28x28x5)

There will be 5 different neurons all looking at the same region in the input volume

CNN

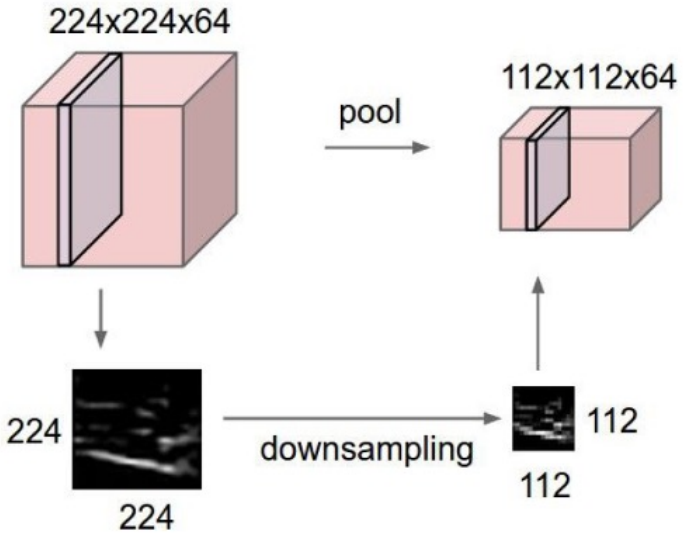
two more layers to go: POOL/FC



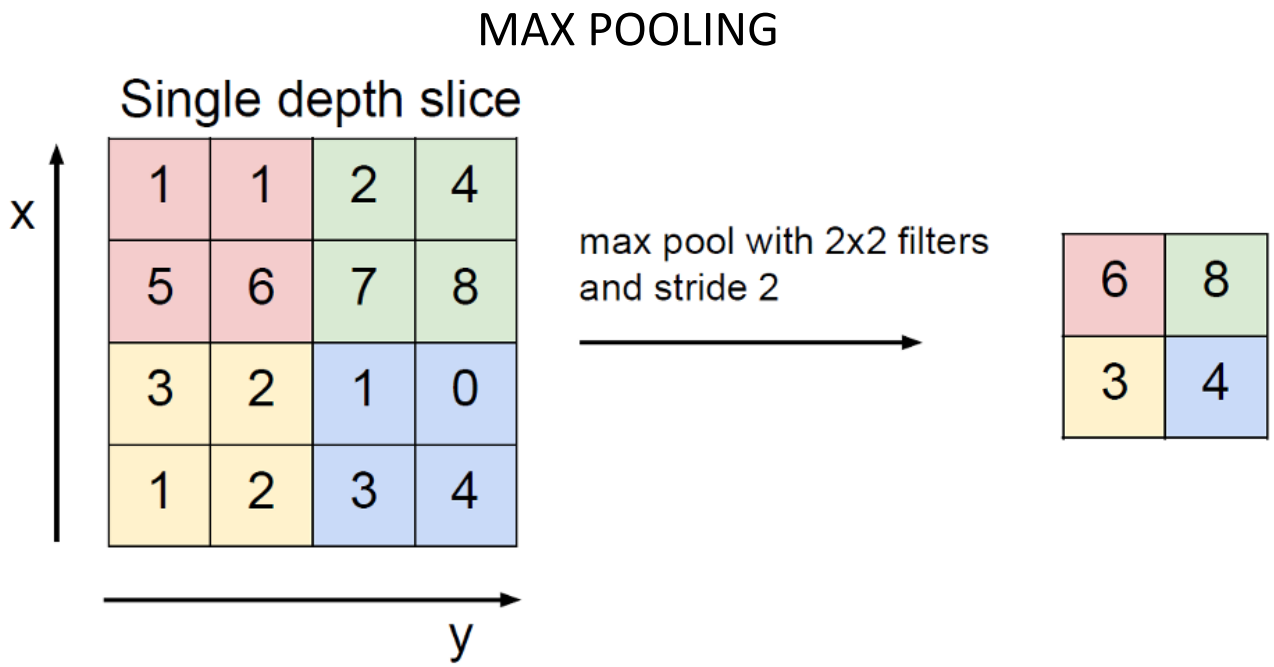
CNN

Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



CNN


MAX POOLING
Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

CNN

- Accepts a volume of size $\boldsymbol{W_1 \times H_1 \times D_1}$
- Requires three hyperparameters:
  - Their spatial extent $\boldsymbol{F}$,
  - The stride $\boldsymbol{S}$,
- Produces a volume of size $\boldsymbol{W_2 \times H_2 \times D_2}$ where:
  - $\boldsymbol{W_2 = (W_1 - F)/S + 1}$
  - $\boldsymbol{H_2 = (H_1 - F)/S + 1}$
  - $\boldsymbol{D_2 = D_1}$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

CNN

- Accepts a volume of size $\boldsymbol{W_1 \times H_1 \times D_1}$
- Requires three hyperparameters:
  - Their spatial extent $\boldsymbol{F}$,
  - The stride $\boldsymbol{S}$,
- Produces a volume of size $\boldsymbol{W_2 \times H_2 \times D_2}$ where:
  - $\boldsymbol{W_2 = (W_1 - F)/S + 1}$
  - $\boldsymbol{H_2 = (H_1 - F)/S + 1}$
  - $\boldsymbol{D_2 = D_1}$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

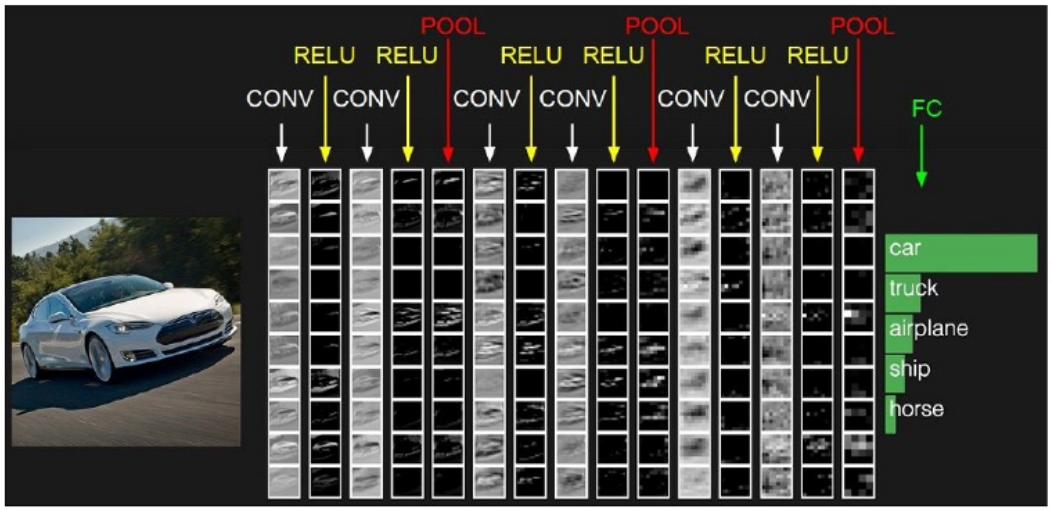
F = 2, S = 2

F = 3, S = 2

CNN

Fully Connected Layer (FC layer)

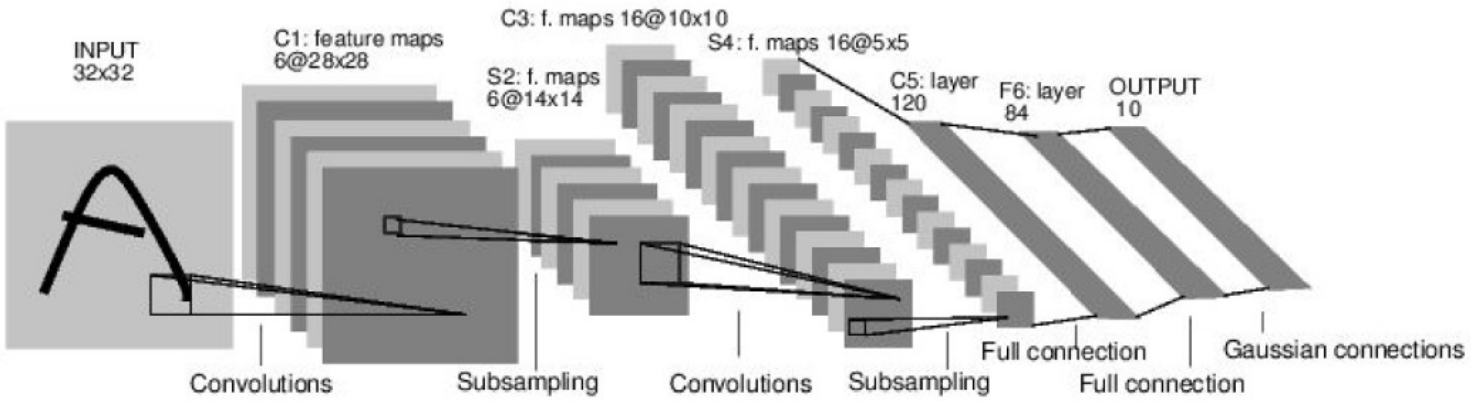
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



CNN

Case Study: LeNet-5

[LeCun et al., 1998]

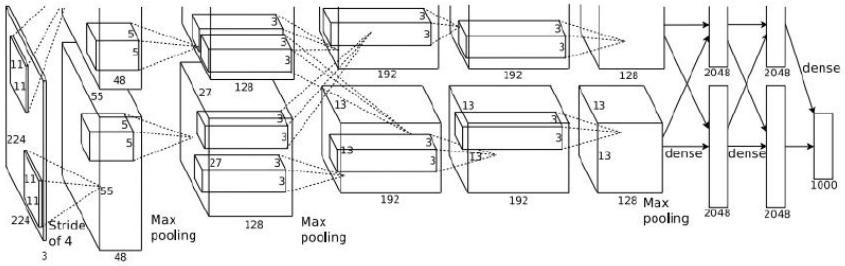


Conv filters were 5x5, applied at strid 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

CNN

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3

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

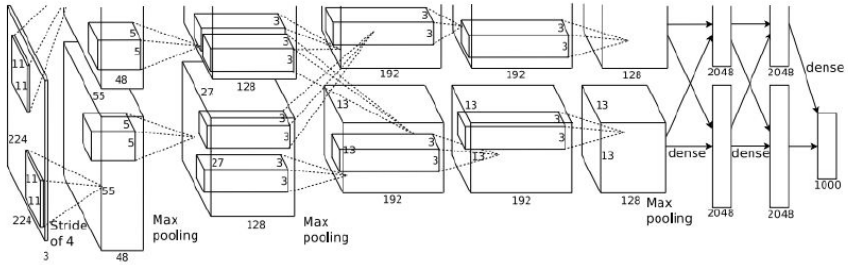
=>

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

CNN

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3

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

=>

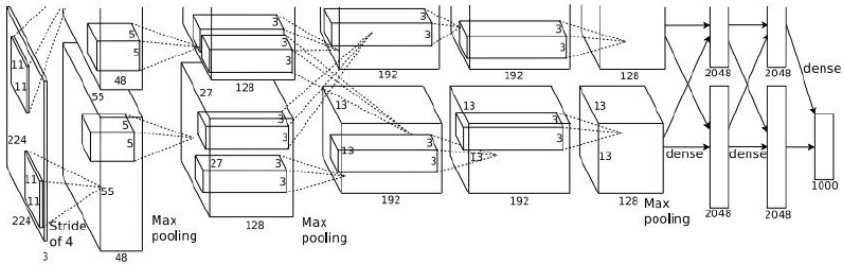
Output volume [55x55x96]

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

CNN

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3

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

=>

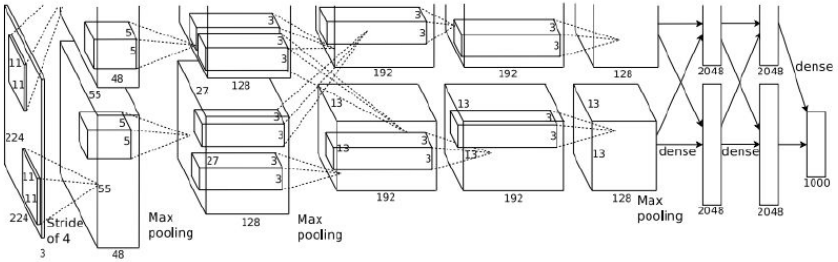
Output volume **[55x55x96]**

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

CNN

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 image
After CONV1: 55x55x96

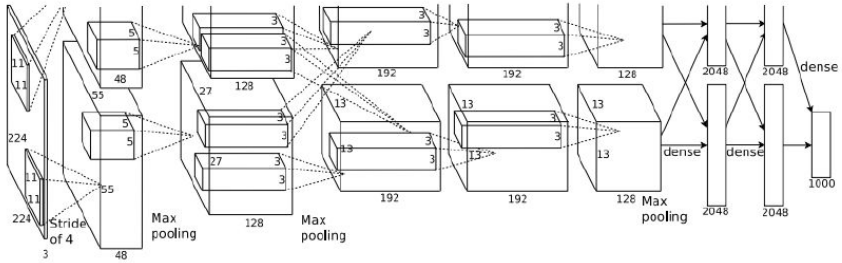
Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

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

CNN

Case Study: AlexNet

[Krizhevsky et al. 2012]



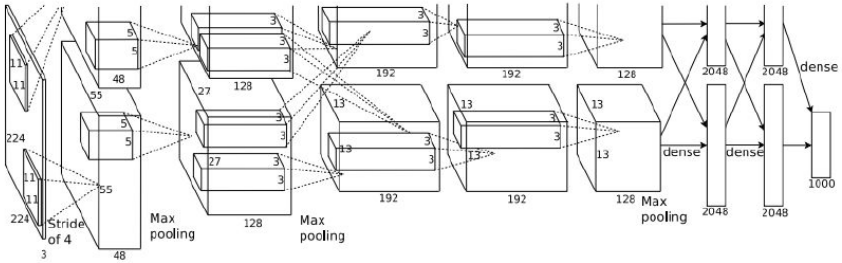
Input: 227x227x3 image
After CONV1: 55x55x96

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

CNN

Case Study: AlexNet

[Krizhevsky et al. 2012]

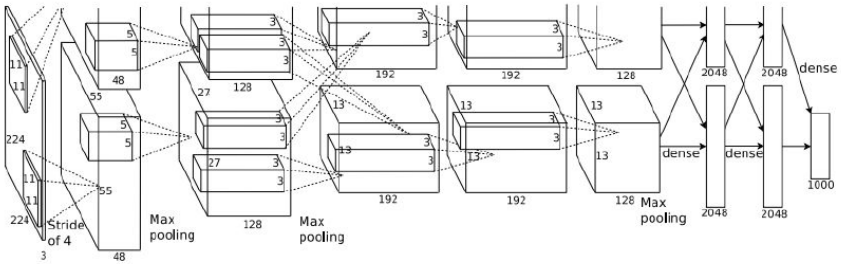


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256]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

CNN

Case Study: AlexNet

[Krizhevsky et al. 2012]



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

Details/Retrospectives:

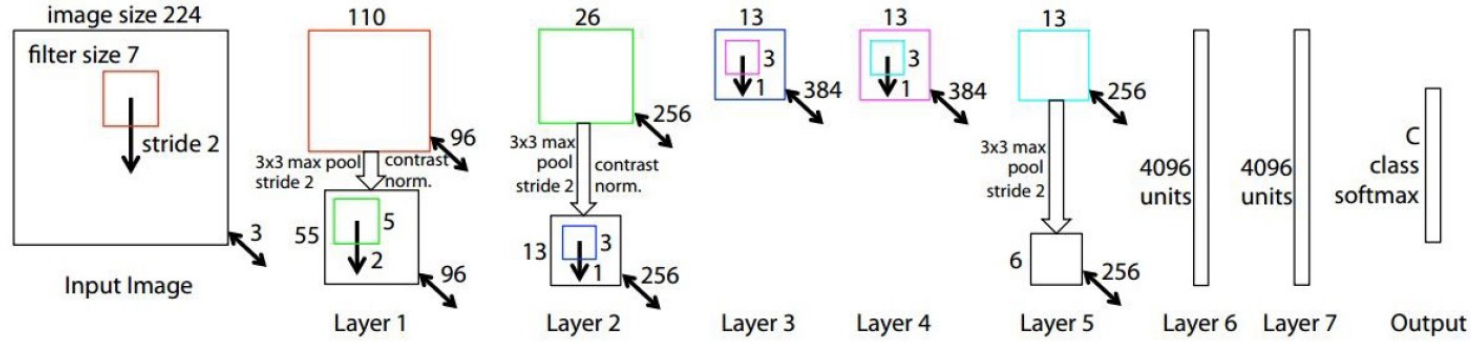
- First use of ReLU
- Used Norm layers (not common anymore)
- Heavy data augmentation
- Dropout 0.5
- Batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10

Manually when val accuracy plateaus

- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

CNN

Case Study: 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%

CNN

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad1
And 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013
->
7.3% top 5 error

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 conv3-25[illegible] **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					

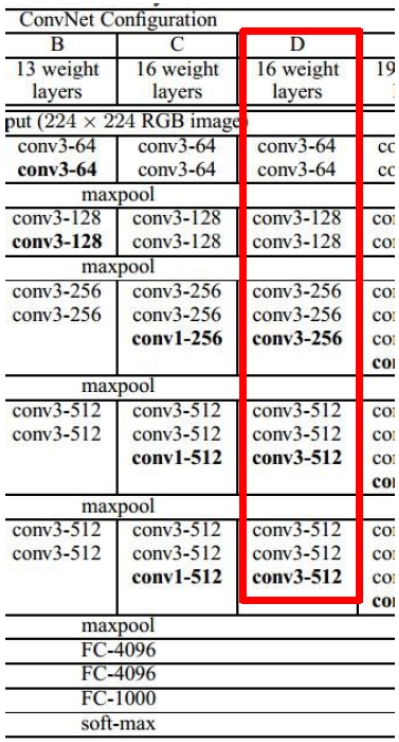
Table 2: **Number of parameters** (in millions).

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

CNN

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000



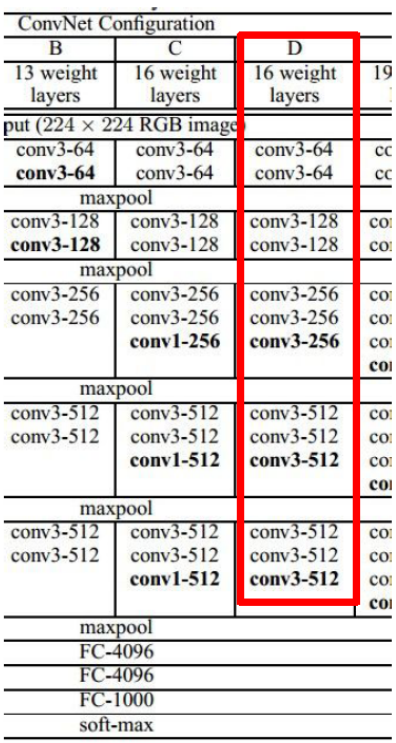
ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64	conv3-64	conv3-64	cc
conv3-64	conv3-64	conv3-64	cc
maxpool			
conv3-128	conv3-128	conv3-128	co
conv3-128	conv3-128	conv3-128	co
maxpool			
conv3-256	conv3-256	conv3-256	co
conv3-256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

CNN

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
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	co co
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			

CNN

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

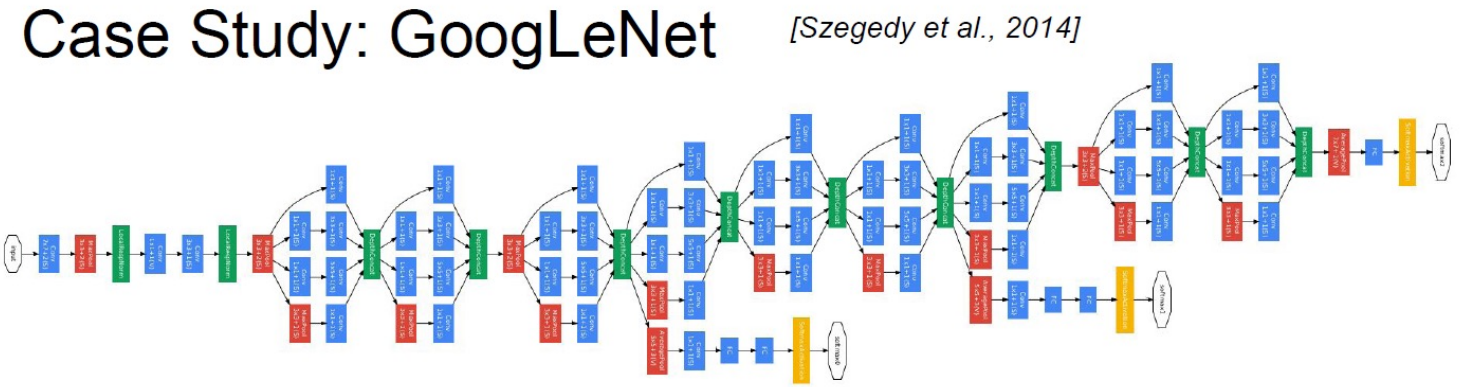
TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

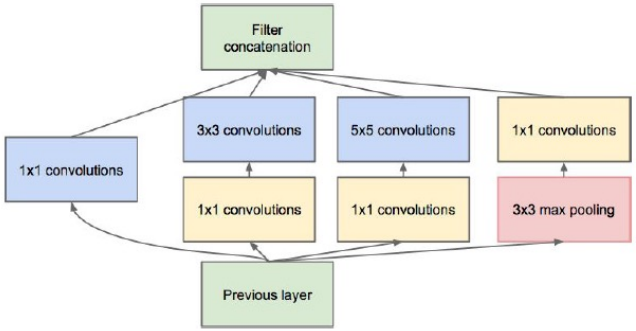
Note:

Most memory is in early CONV

Most params are in late FC

CNN





Inception module

ILSVRC 2014 winner (6.7% top 5 error)

CNN

Case Study: GoogLeNet

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								

Fun features:

- Only 5 million params! (Removes FC layers completely)

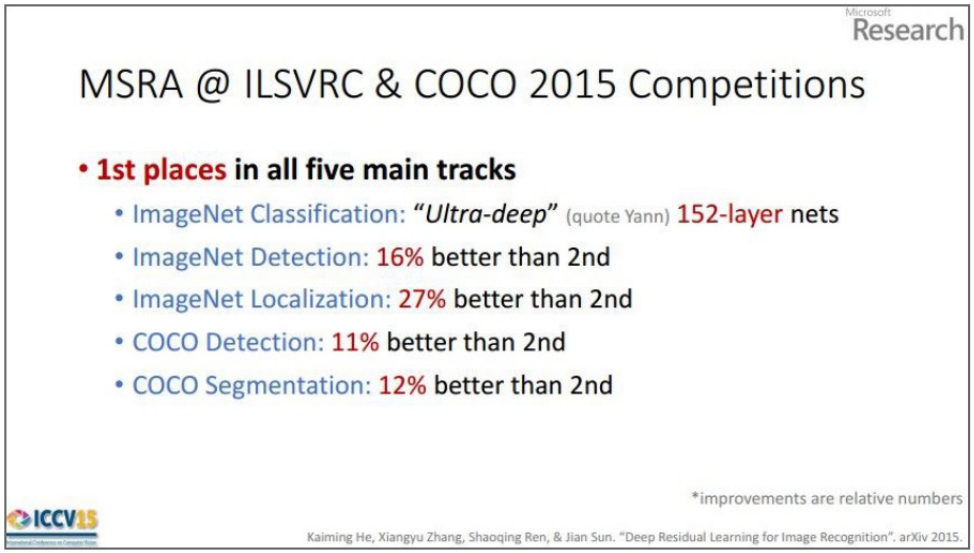
Compared to AlexNet:

- **12X less params**
- **2x more compute**
- **6.67% (vs. 16.4%)**

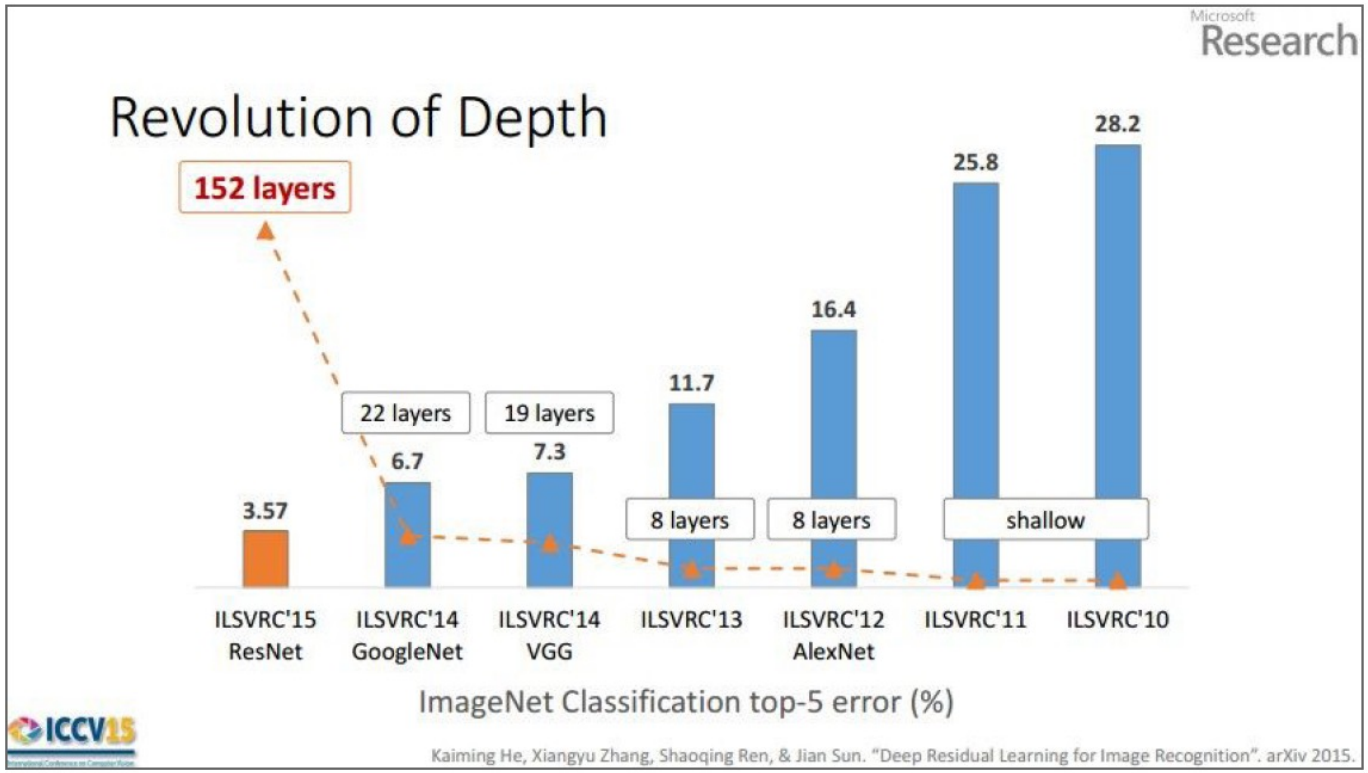
CNN

Case Study: ResNet *[He et al., 2015]*

ILSVRC 2015 winner (3.6% top 5 error)

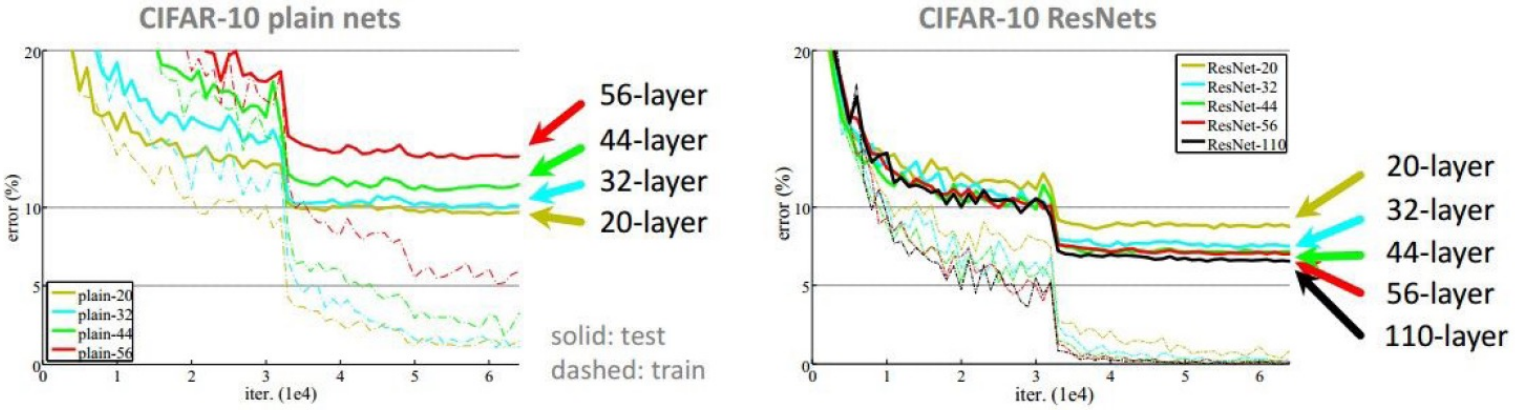


CNN


Microsoft Research
Revolution of Depth
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10
ImageNet Classification top-5 error (%)
ICCV15
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

CNN

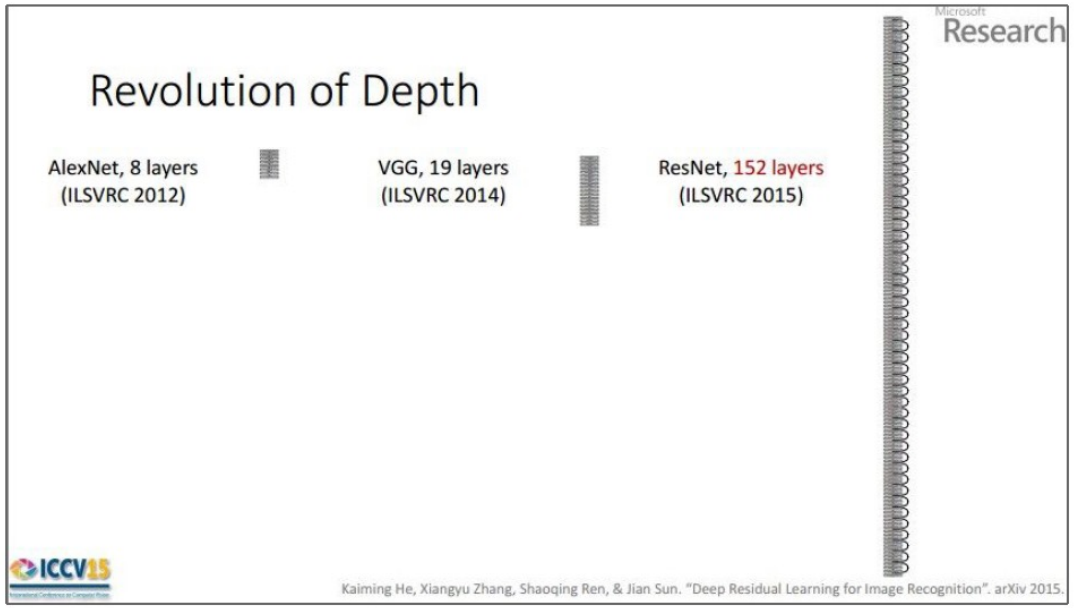
CIFAR-10 experiments



CNN

Case Study: ResNet *[He et al., 2015]*

ILSVRC 2015 winner (3.6% top 5 error)



(slide from Kaiming He's recent presentation)

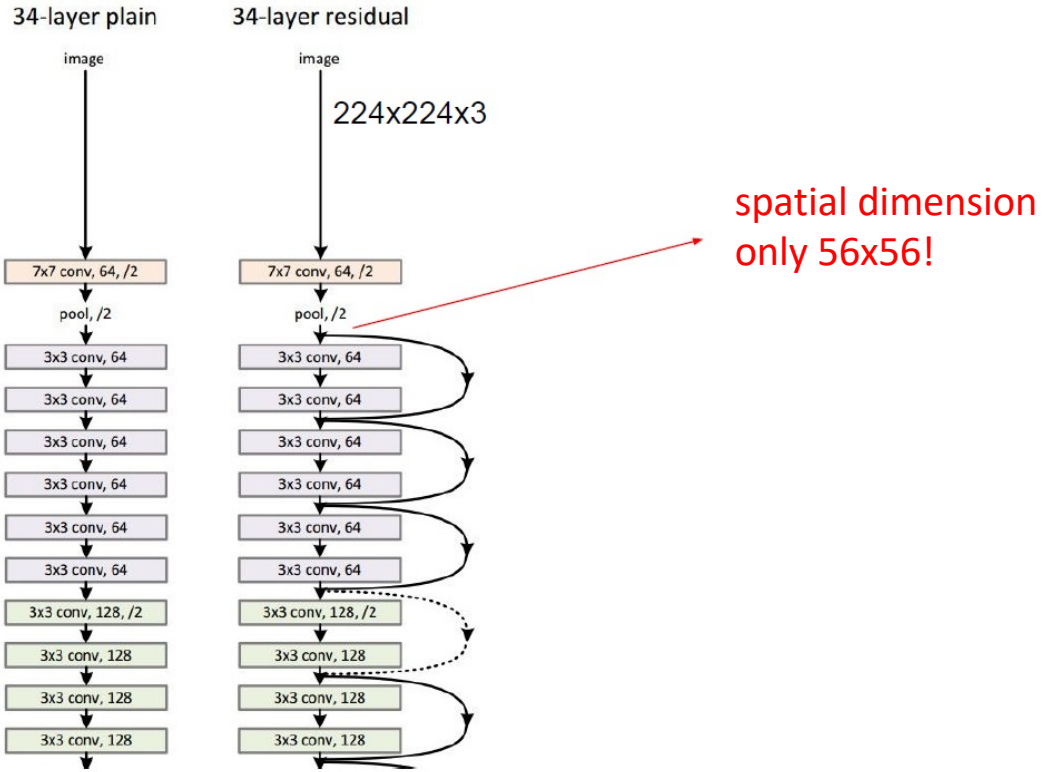
2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet! (even though it has 8x more layers)

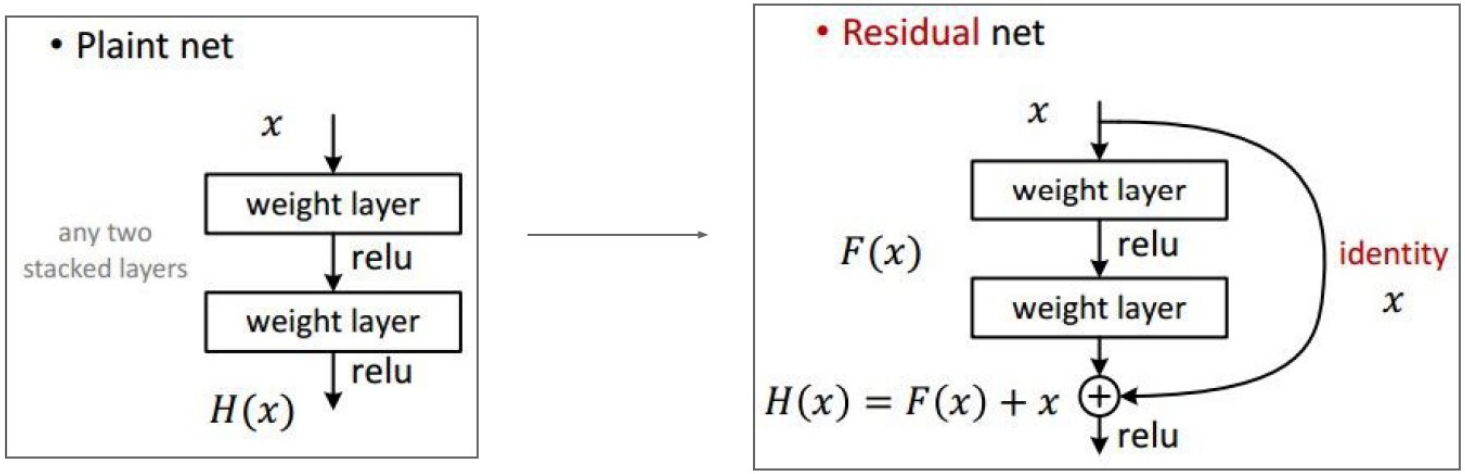
CNN

Case Study: ResNet

[He et al., 2015]



Case Study: ResNet *[He et al., 2015]*



CNN

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