

CSCI567 Machine Learning (Spring 2021)

Sirisha Rambhatla

University of Southern California

Feb 17, 2021

Outline

1 Logistics

2 Review of last lecture

3 Convolutional neural networks (ConvNets/CNNs)

1 / 24

2 / 24

Logistics

Outline

Logistics

Logistics

1 Logistics

2 Review of last lecture

- Sign-up with your group members for the project!

3 Convolutional neural networks (ConvNets/CNNs)

3 / 24

4 / 24

Outline

1 Logistics

2 Review of last lecture

3 Convolutional neural networks (ConvNets/CNNs)

Backprop = SGD for neural nets

The **backpropagation** algorithm (**Backprop**)

Initialize $\mathbf{W}_1, \dots, \mathbf{W}_L$ (all **0** or randomly). Repeat:

- 1 randomly pick one data point $n \in [N]$
- 2 **forward propagation**: for each layer $\ell = 1, \dots, L$
 - compute $\mathbf{a}_\ell = \mathbf{W}_\ell \mathbf{o}_{\ell-1}$ and $\mathbf{o}_\ell = \mathbf{h}_\ell(\mathbf{a}_\ell)$ $(\mathbf{o}_0 = \mathbf{x}_n)$
- 3 **backward propagation**: for each $\ell = L, \dots, 1$
 - compute

$$\frac{\partial \mathcal{E}_n}{\partial \mathbf{a}_\ell} = \begin{cases} \left(\mathbf{W}_{\ell+1}^T \frac{\partial \mathcal{E}_n}{\partial \mathbf{a}_{\ell+1}} \right) \circ \mathbf{h}'_\ell(\mathbf{a}_\ell) & \text{if } \ell < L \\ 2(\mathbf{h}_L(\mathbf{a}_L) - \mathbf{y}_n) \circ \mathbf{h}'_L(\mathbf{a}_L) & \text{else} \end{cases}$$

- update weights

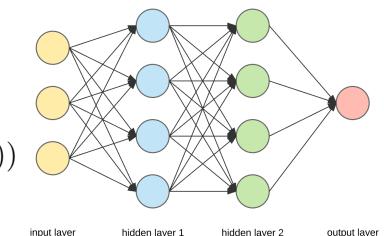
$$\mathbf{W}_\ell \leftarrow \mathbf{W}_\ell - \eta \frac{\partial \mathcal{E}_n}{\partial \mathbf{W}_\ell} = \mathbf{W}_\ell - \eta \frac{\partial \mathcal{E}_n}{\partial \mathbf{a}_\ell} \mathbf{o}_{\ell-1}^T$$

Think about how to do the last two steps properly!

Math formulation of neural nets

An L -layer neural net can be written as

$$\mathbf{f}(\mathbf{x}) = \mathbf{h}_L(\mathbf{W}_L \mathbf{h}_{L-1}(\mathbf{W}_{L-1} \cdots \mathbf{h}_1(\mathbf{W}_1 \mathbf{x})))$$



To ease notation, for a given input \mathbf{x} , define recursively

$$\mathbf{o}_0 = \mathbf{x}, \quad \mathbf{a}_\ell = \mathbf{W}_\ell \mathbf{o}_{\ell-1}, \quad \mathbf{o}_\ell = \mathbf{h}_\ell(\mathbf{a}_\ell) \quad (\ell = 1, \dots, L)$$

where

- $\mathbf{W}_\ell \in \mathbb{R}^{D_\ell \times D_{\ell-1}}$ is the weights for layer ℓ
- $D_0 = D, D_1, \dots, D_L$ are numbers of neurons at each layer
- $\mathbf{a}_\ell \in \mathbb{R}^{D_\ell}$ is input to layer ℓ
- $\mathbf{o}_\ell \in \mathbb{R}^{D_\ell}$ is output to layer ℓ
- $\mathbf{h} : \mathbb{R}^{D_\ell} \rightarrow \mathbb{R}^{D_\ell}$ is activation functions at layer ℓ

Outline

1 Logistics

2 Review of last lecture

3 Convolutional neural networks (ConvNets/CNNs)

- Motivation
- Architecture

Acknowledgements

Not much math, a lot of empirical intuitions

The materials borrow heavily from the following sources:

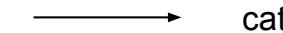
- Stanford Course Cs231n: <http://cs231n.stanford.edu/>
- Dr. Ian Goodfellow's lectures on deep learning:
<http://deeplearningbook.org>

Both website provides tons of useful resources: notes, demos, videos, etc.

Image Classification: A core task in Computer Vision



(assume given set of discrete labels)
{dog, cat, truck, plane, ...}



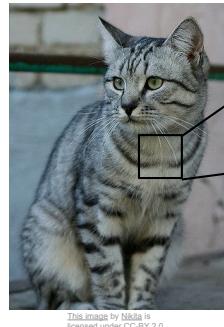
Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 2 - 6

April 6, 2017

9 / 24

The Problem: Semantic Gap



[1] 185 132 188 111 184 99 186 99 96 183 112 119 184 97 93 87]

What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

Challenges: Viewpoint variation



[1] 185 132 188 111 184 99 186 99 96 183 112 119 184 97 93 87]
[2] 74 85 98 185 128 185 87 96 99 99 115 112 186 183 99 85]
[3] 99 81 81 93 128 131 127 188 95 98 182 99 96 93 181 84]
[4] 188 132 188 111 184 99 186 99 96 183 112 119 184 97 93 85]
[5] 111 85 55 55 60 64 54 64 87 112 120 94 74 84 91]
[6] 132 137 144 148 189 93 89 78 62 65 63 63 68 73 86 181]
[7] 125 132 148 137 119 121 117 94 65 79 88 65 54 64 72 98]
[8] 127 132 148 137 119 121 117 94 65 79 88 65 54 64 72 98]
[9] 115 118 189 123 158 149 111 118 113 189 188 92 74 65 72 78]
[10] 132 132 148 137 119 121 117 94 65 79 88 65 54 64 72 98]
[11] 63 77 86 81 77 79 182 123 117 115 117 125 125 138 115 87]
[12] 63 65 75 88 89 71 62 91 128 138 135 183 91 98 118 118]
[13] 67 65 71 87 186 90 69 45 76 138 124 187 92 94 185 121]
[14] 125 132 148 137 119 121 117 94 65 79 88 65 54 64 72 98]
[15] 146 132 188 82 120 121 184 76 48 45 66 88 181 182 189]
[16] 128 132 148 137 119 121 117 94 65 79 88 65 54 64 72 98]
[17] 130 128 134 161 139 180 180 118 121 134 114 87 65 53 60 86]
[18] 112 96 117 158 140 128 115 184 187 182 93 88 104 72 79]
[19] 122 123 182 88 82 86 94 117 145 148 153 182 58 78 92 187]
[20] 122 154 149 183 71 56 76 83 93 180 119 129 182 51 69 84]

All pixels change when the camera moves!

Challenges: Illumination



[This image is CC0 1.0 public domain](#)

Challenges: Deformation



[This image by Umberto Salvagnin is licensed under CC-BY 2.0](#)

[This image by Umberto Salvagnin is licensed under CC-BY 2.0](#)

[This image by sara_bear is licensed under CC-BY 2.0](#)

[This image by Tom.Thai is licensed under CC-BY 2.0](#)

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 2 - 9

April 6, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 2 - 10

April 6, 2017

Challenges: Occlusion



[This image is CC0 1.0 public domain](#)

[This image is CC0 1.0 public domain](#)

[This image by jonsson is licensed under CC-BY 2.0](#)

Challenges: Background Clutter



[This image is CC0 1.0 public domain](#)

[This image is CC0 1.0 public domain](#)

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 2 - 11

April 6, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 2 - 12

April 6, 2017

Fundamental problems in vision

Challenges: Intraclass variation



The key challenge

How to train a model that can tolerate all those variations?

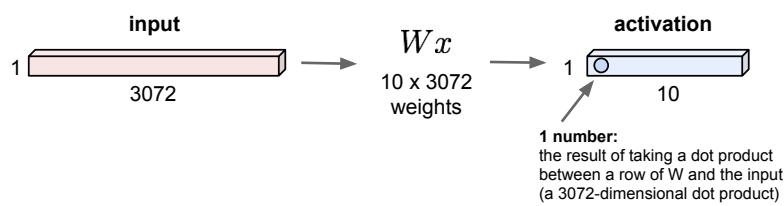
Main ideas

- need a lot of data that exhibits those variations
- need more specialized models to capture the invariance

Issues of standard NN for image inputs

Fully Connected Layer

32x32x3 image \rightarrow stretch to 3072 x 1

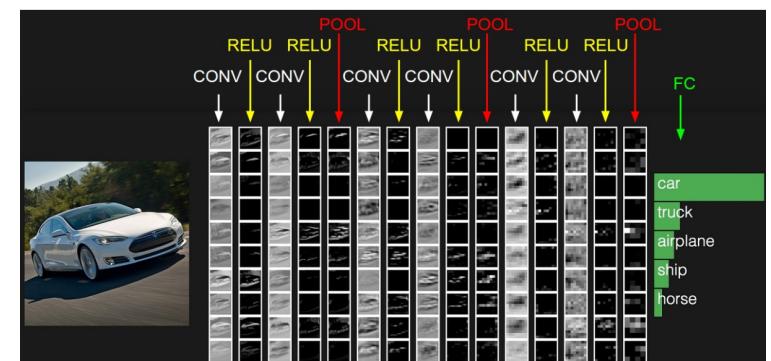


Spatial structure is lost!

Solution: Convolutional Neural Net (ConvNet/CNN)

A special case of fully connected neural nets

- usually consist of **convolution layers**, ReLU layers, **pooling layers**, and regular fully connected layers
- key idea: *learning from low-level to high-level features*

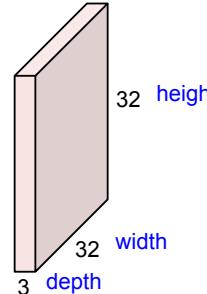


Convolution layer

Arrange neurons as a **3D volume** naturally

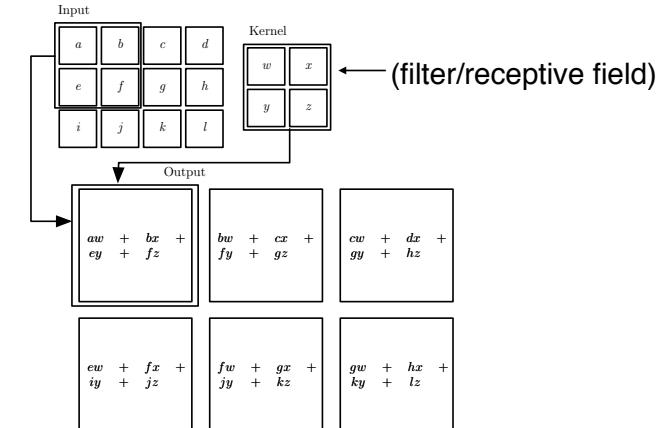
Convolution Layer

32x32x3 image -> preserve spatial structure



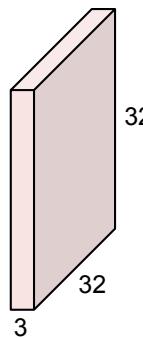
Convolution

2D Convolution

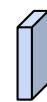


Convolution Layer

32x32x3 image



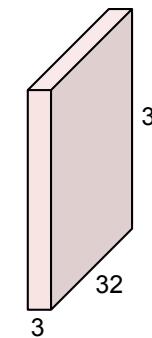
5x5x3 filter



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

Convolution Layer

32x32x3 image

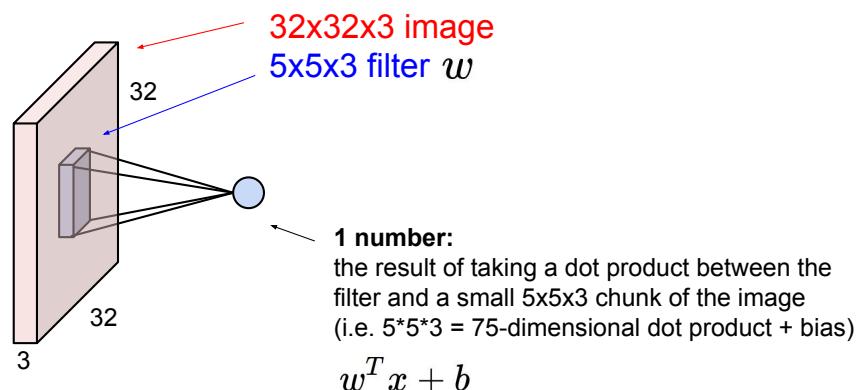


5x5x3 filter



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

Convolution Layer

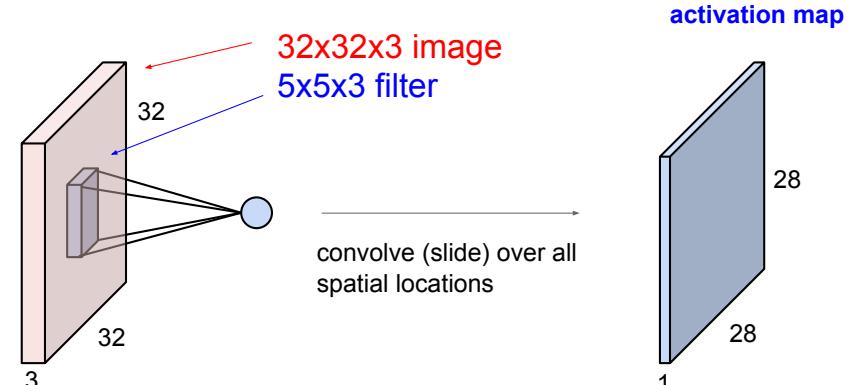


Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 5 - 31

April 18, 2017

Convolution Layer



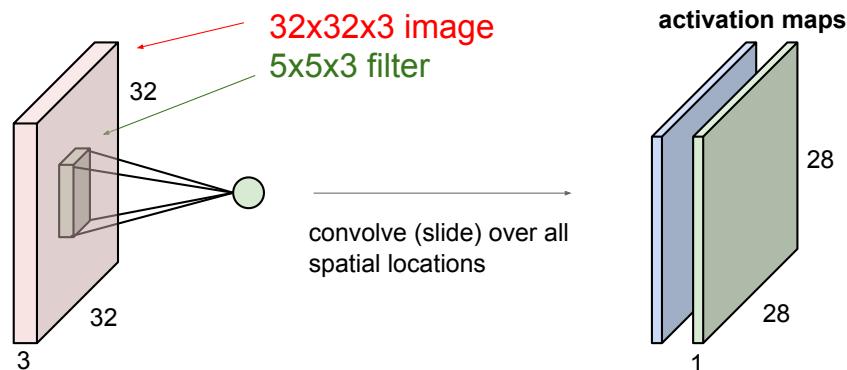
Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 5 - 32

April 18, 2017

Convolution Layer

consider a second, green filter

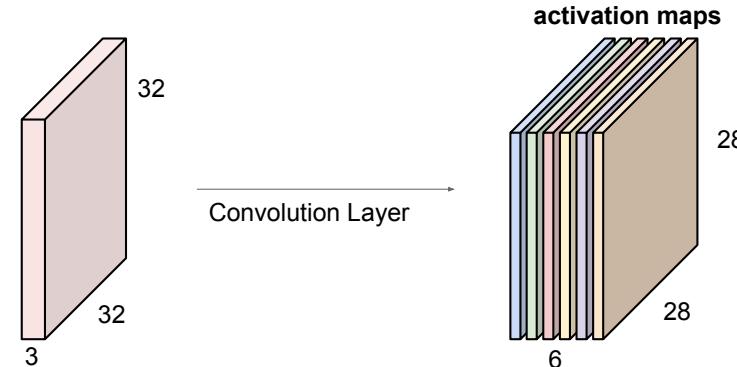


Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 5 - 33

April 18, 2017

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



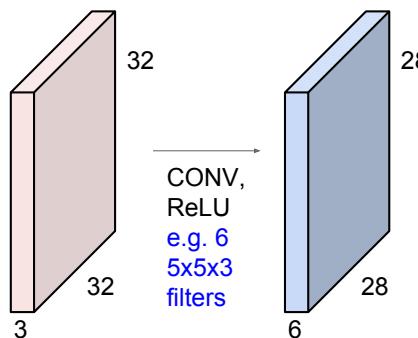
We stack these up to get a "new image" of size 28x28x6!

Fei-Fei Li & Justin Johnson & Serena Yeung

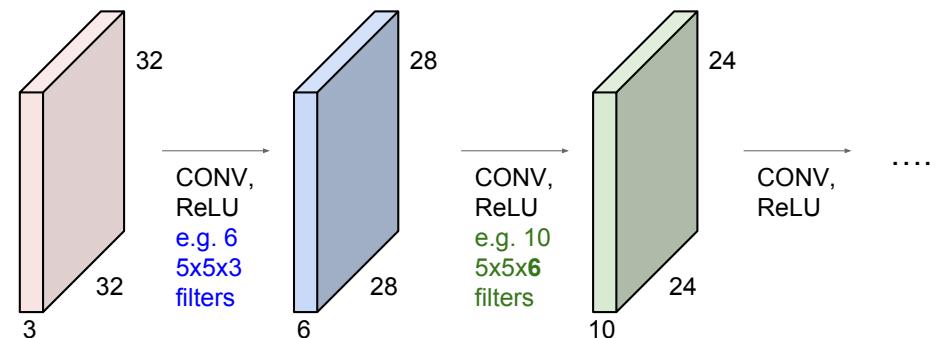
Lecture 5 - 34

April 18, 2017

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



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



Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 5 - 35

April 18, 2017

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 5 - 36

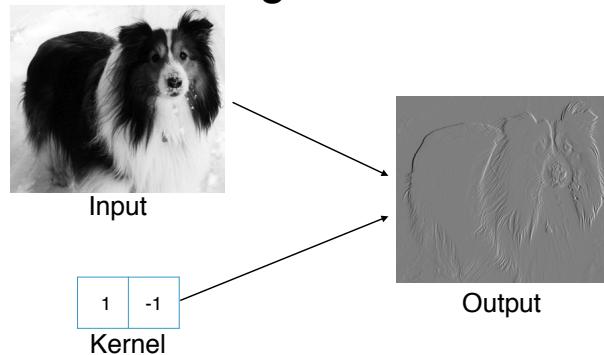
April 18, 2017

Convolutional neural networks (ConvNets/CNNs) Architecture

Why convolution makes sense?

Main idea: **if a filter is useful at one location, it should be useful at other locations.**

A simple example why filtering is useful



Convolutional neural networks (ConvNets/CNNs) Architecture

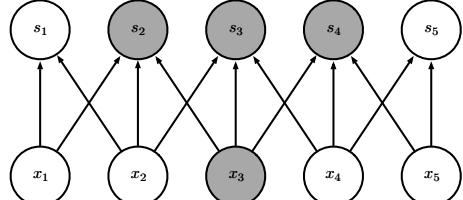
Connection to fully connected NNs

A convolution layer is a special case of a fully connected layer:

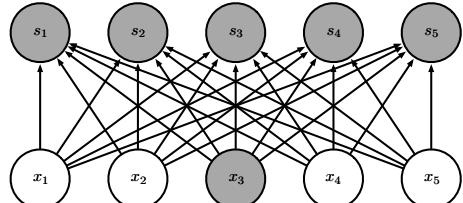
- filter = weights with **sparse connection**

Local Receptive Field Leads to Sparse Connectivity (affects less)

Sparse connections due to small convolution kernel



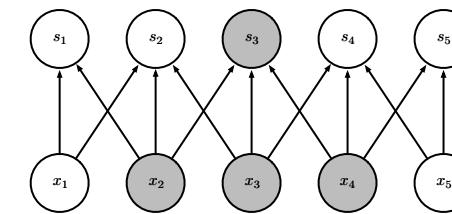
Dense connections



(Goodfellow 2016)

Sparse connectivity: being affected by less

Sparse connections due to small convolution kernel



Dense connections

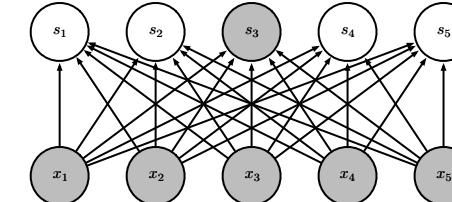


Figure 9.3

(Goodfellow 2016)

Convolutional neural networks (ConvNets/CNNs) Architecture

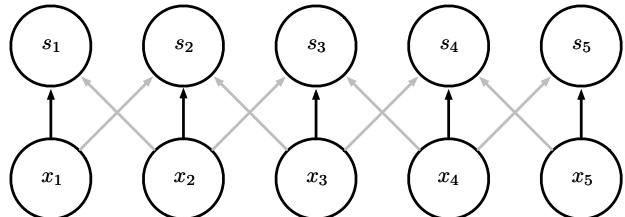
Connection to fully connected NNs

A convolution layer is a special case of a fully connected layer:

- filter = weights with **sparse connection**
- **parameters sharing**

Parameter Sharing

Convolution shares the same parameters across all spatial locations



Traditional matrix multiplication does not share any parameters

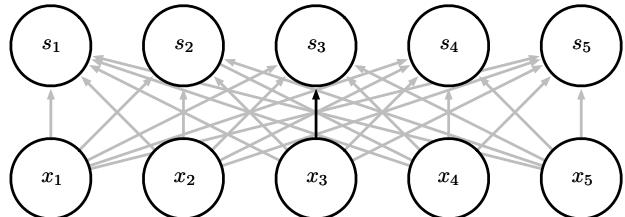


Figure 9.5

(Goodfellow 2016)

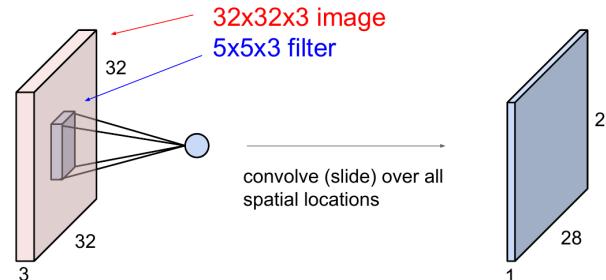
Connection to fully connected NNs

A convolution layer is a special case of a fully connected layer:

- filter = weights with **sparse connection**
- **parameters sharing**

Much less parameters! Example (ignore bias terms):

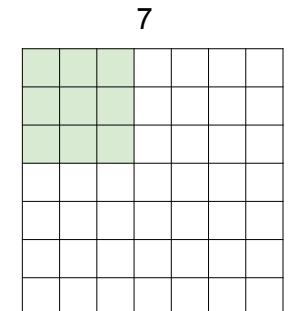
- FC: $(32 \times 32 \times 3) \times (28 \times 28) \approx 2.4M$
- CNN: $5 \times 5 \times 3 = 75$



18 / 24

Spatial arrangement: stride and padding

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

7

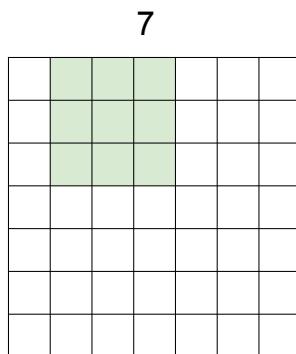
Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 5 - 42

April 18, 2017

19 / 24

A closer look at spatial dimensions:

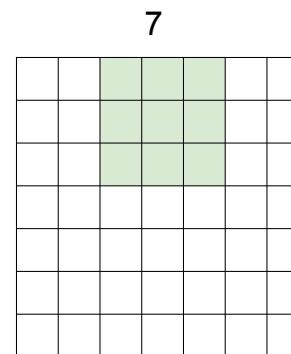


7

7

7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

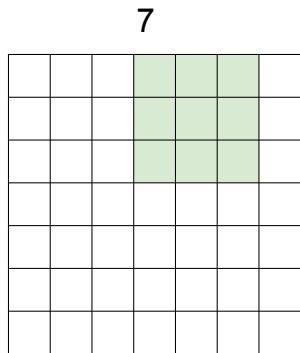


7

7

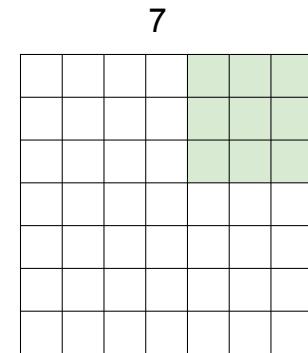
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

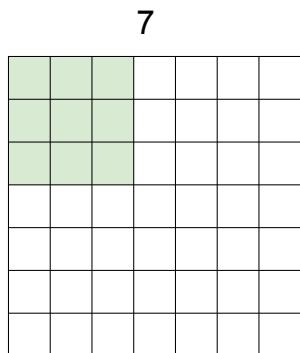
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

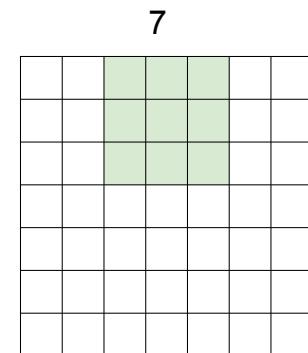
=> **5x5 output**

A closer look at spatial dimensions:



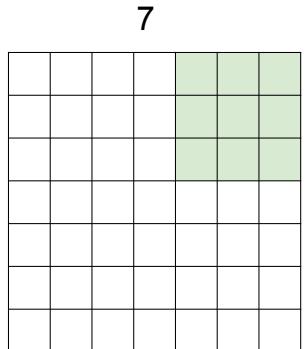
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



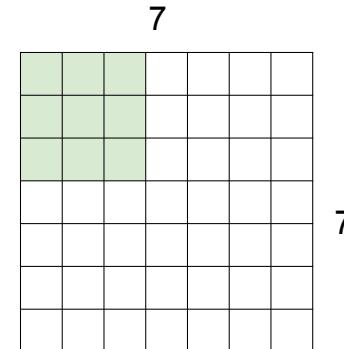
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



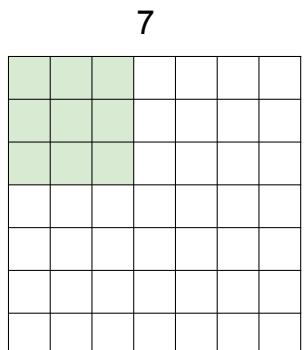
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



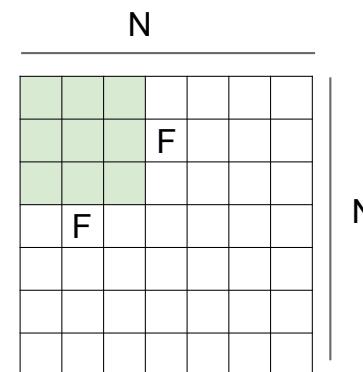
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

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 => $(7 - 3)/3 + 1 = 2.33 \backslash$

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

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

0	0	0	0	0	0			
0								
0								
0								
0								

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

0	0	0	0	0	0			
0								
0								
0								
0								

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 $F \times F$, and zero-padding with $(F-1)/2$. (will preserve size spatially)

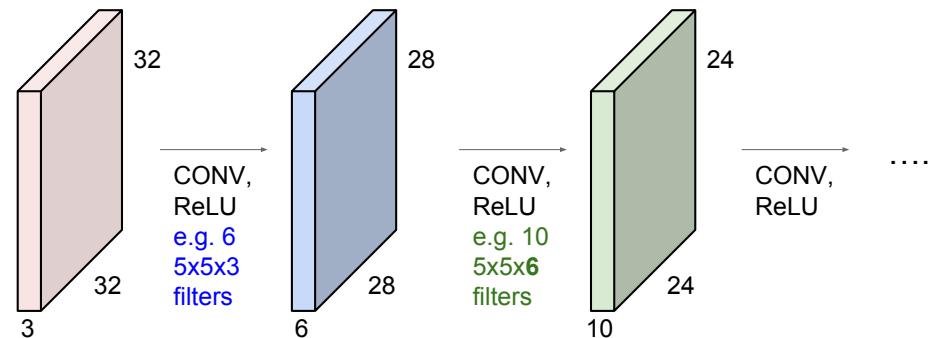
e.g. $F = 3 \Rightarrow$ zero pad with 1

$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

Remember back to...

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



Summary for convolution layer

Input: a volume of size $W_1 \times H_1 \times D_1$

Hyperparameters:

- K filters of size $F \times F$
- stride S
- amount of zero padding P (for one side)

Output: a volume of size $W_2 \times H_2 \times D_2$ where

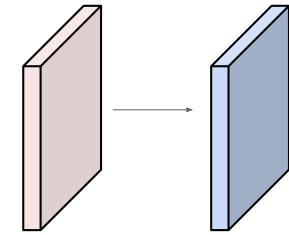
- $W_2 = (W_1 + 2P - F)/S + 1$
- $H_2 = (H_1 + 2P - F)/S + 1$
- $D_2 = K$

#parameters: $(F \times F \times D_1 + 1) \times K$ weights

Common setting: $F = 3, S = P = 1$

20 / 24

Examples time:



Input volume: **32x32x3**

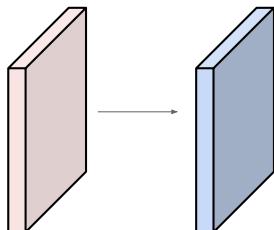
10 5x5 filters with stride 1, pad 2

Output volume size: ?

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 5 - 57 April 18, 2017

Examples time:



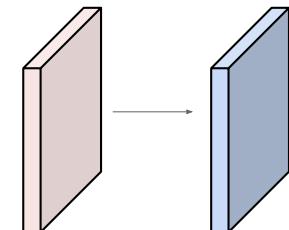
Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size:

$(32+2*2-5)/1+1 = 32$ spatially, so
32x32x10

Examples time:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

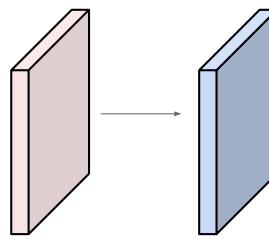
Number of parameters in this layer?

Another element: pooling

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?

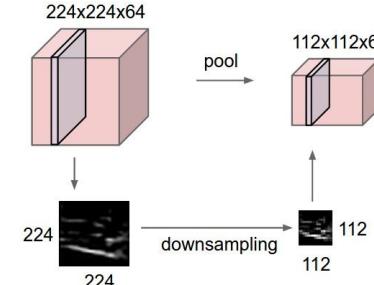
each filter has $5 \times 5 \times 3 + 1 = 76$ params

$$\Rightarrow 76 \times 10 = 760$$

(+1 for bias)

Pooling layer

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



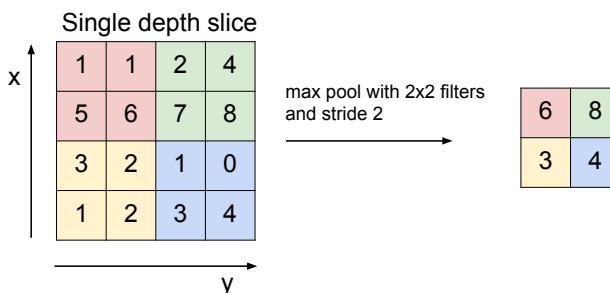
Pooling

Similar to a filter, except

- depth is always 1
- different operations: average, L2-norm, max
- no parameters to be learned

Max pooling with 2×2 filter and stride 2 is very common

MAX POOLING



Putting everything together

Typical architecture for CNNs:

Input \rightarrow [[Conv \rightarrow ReLU]*N \rightarrow Pool?]*M \rightarrow [FC \rightarrow ReLU]*Q \rightarrow FC

Common choices: $N \leq 5, Q \leq 2, M$ is large

Well-known CNNs: LeNet, AlexNet, ZF Net, GoogLeNet, VGGNet, etc.

All achieve excellent performance on image classification tasks.

How to train a CNN?

How do we learn the filters/weights?

Essentially the same as FC NNs: apply **SGD/backpropagation**