CSCI567 Machine Learning (Spring 2021)

Sirisha Rambhatla

University of Southern California

Feb 17, 2021

Outline

Outline

Logistics

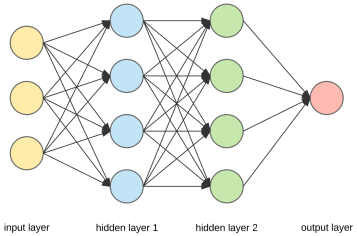
- Sign-up with your group members for the project!

Outline

Math formulation of neural nets

An L-layer neural net can be written as

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{h}_{\mathsf{L}}\left(\boldsymbol{W}_L \boldsymbol{h}_{\mathsf{L}-1}\left(\boldsymbol{W}_{L-1} \cdots \boldsymbol{h}_1\left(\boldsymbol{W}_1 \boldsymbol{x}\right)\right)\right)$$



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

$$\boldsymbol{o}_0 = \boldsymbol{x}, \qquad \boldsymbol{a}_\ell = \boldsymbol{W}_\ell \boldsymbol{o}_{\ell-1}, \qquad \boldsymbol{o}_\ell = \boldsymbol{h}_\ell(\boldsymbol{a}_\ell) \qquad (\ell = 1, \ldots, \mathsf{L})$$

where

- $\boldsymbol{W}_\ell \in \mathbb{R}^{\mathsf{D}_\ell \times \mathsf{D}_{\ell-1}}$ is the weights for layer ℓ
- $\mathsf{D}_0 = \mathsf{D}, \mathsf{D}_1, \ldots, \mathsf{D}_{\mathsf{L}}$ are numbers of neurons at each layer
- $\boldsymbol{a}_\ell \in \mathbb{R}^{\mathsf{D}_\ell}$ is input to layer ℓ
- $\boldsymbol{o}_\ell \in \mathbb{R}^{\mathsf{D}_\ell}$ is output to layer ℓ
- $\boldsymbol{h} : \mathbb{R}^{\mathsf{D}_\ell} \to \mathbb{R}^{\mathsf{D}_\ell}$ is activation functions at layer ℓ

Backprop = SGD for neural nets

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

Initialize $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_{\mathsf{L}}$ (all $\mathbf{0}$ or randomly). Repeat:

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

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

   - update weights

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

Think about how to do the last two steps properly!

Outline

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`

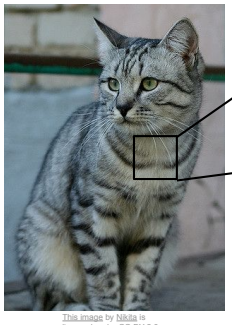
Both website provides tons of useful resources: notes, demos, videos, etc.

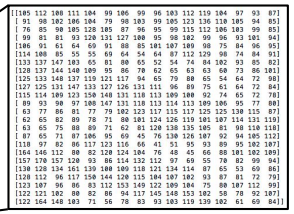
Image Classification: A core task in Computer Vision

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

→ cat

The Problem: Semantic Gap



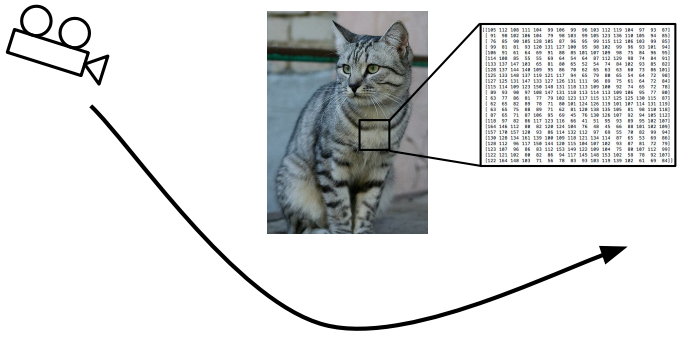


What the computer sees

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

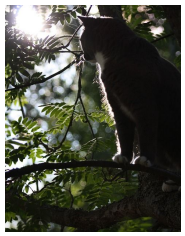
e.g. 800 x 600 x 3
(3 channels RGB)

Challenges: Viewpoint variation

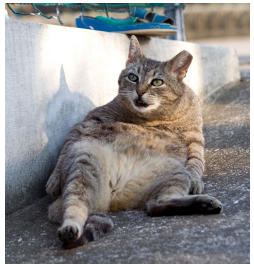


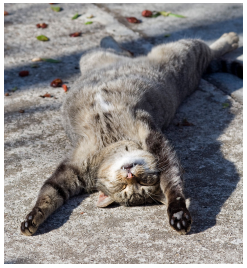
All pixels change when the camera moves!

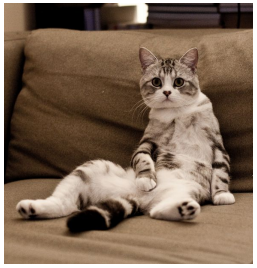
Challenges: Illumination



Challenges: Deformation







Challenges: Occlusion

Challenges: Background Clutter

Challenges: Intraclass variation

This image is CC0 1.0 public domain

Fundamental problems in vision

The key challenge

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

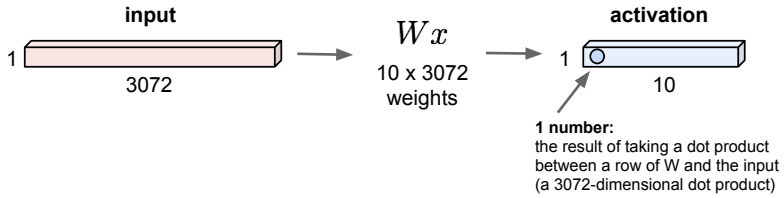
Main ideas

- 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 -> stretch to 3072 x 1

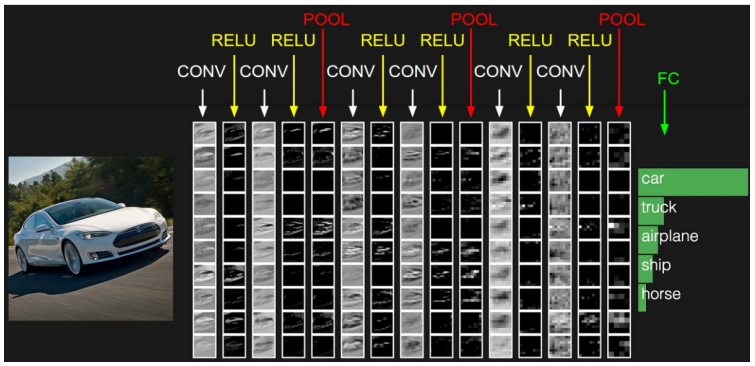


Spatial structure is lost!

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*

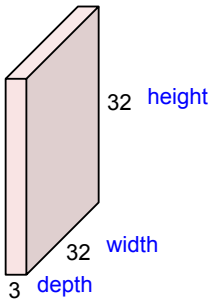


Convolution layer

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

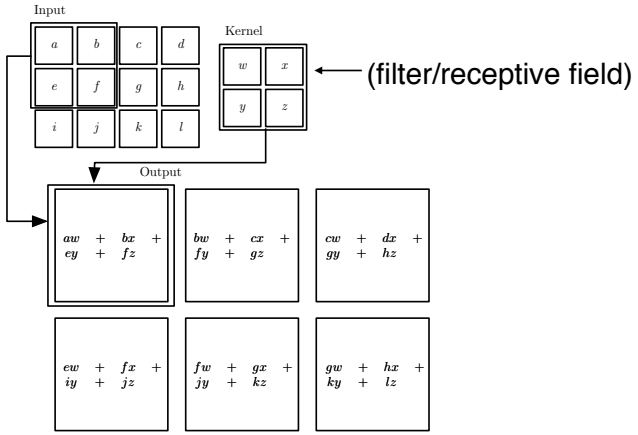
Convolution Layer

32x32x3 image -> preserve spatial structure



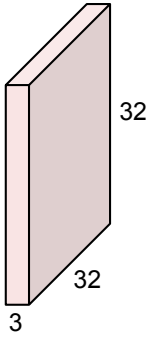
Convolution

2D Convolution



Convolution Layer

32x32x3 image



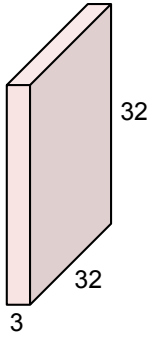
5x5x3 filter

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

Convolution Layer

Filters always extend the full depth of the input volume

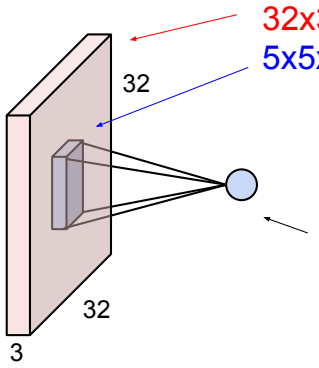
32x32x3 image



5x5x3 filter

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

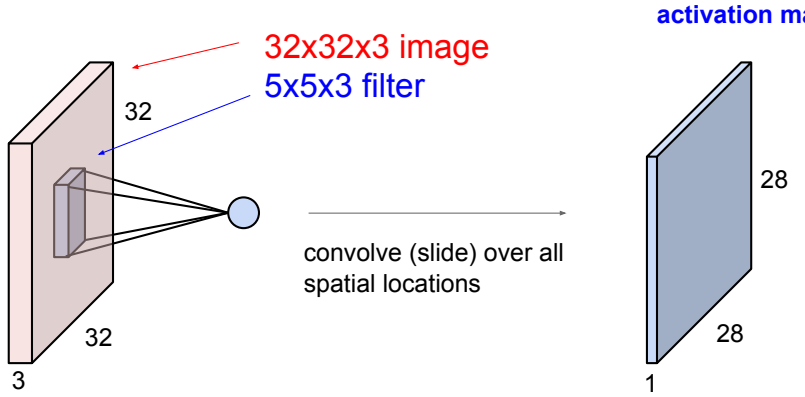
Convolution Layer



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)

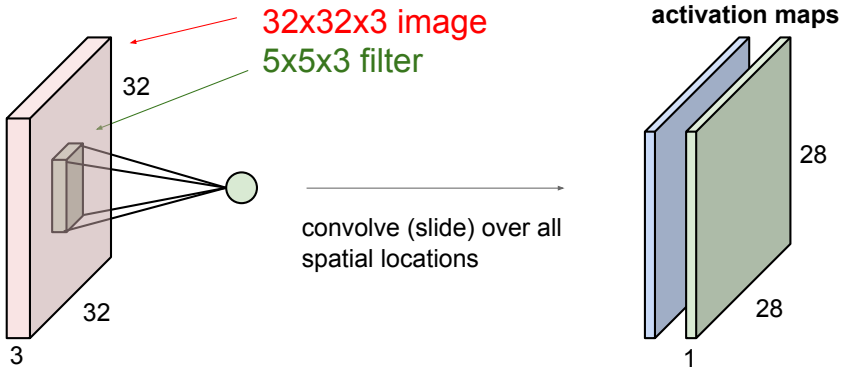
$$w^T x + b$$

Convolution Layer

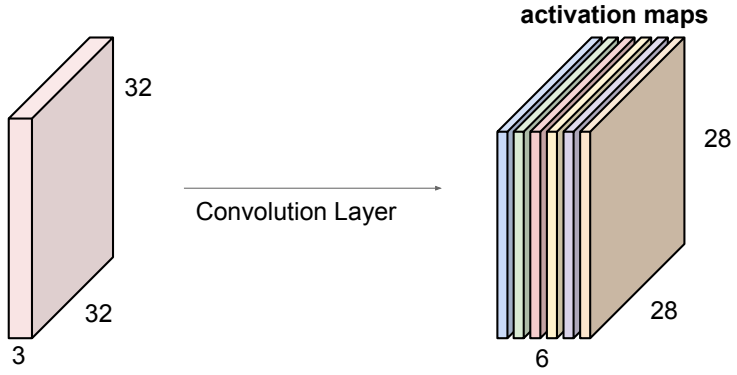


Convolution Layer

consider a second, green filter

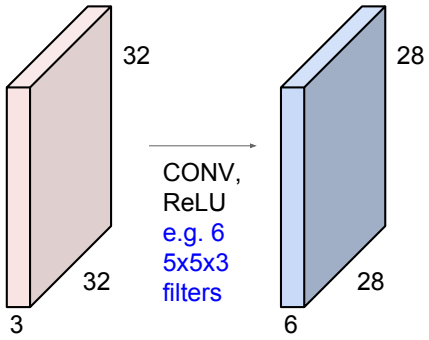


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

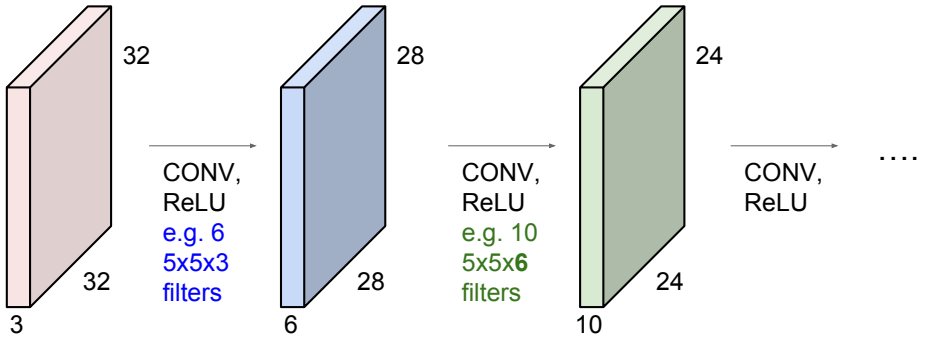


We stack these up to get a "new image" of size 28x28x6!

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



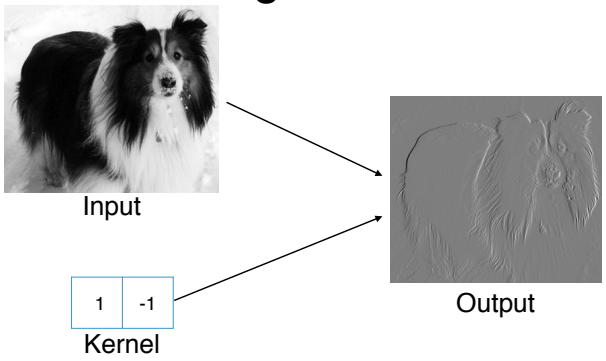
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



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



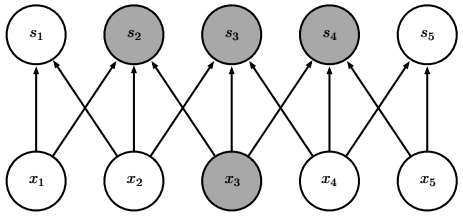
Connection to fully connected NNs

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

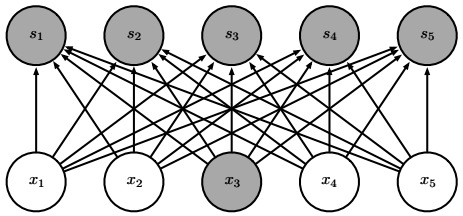
- filter = weights with **sparse connection**

Local Receptive Field Leads to Sparse Connectivity (affects less)

Sparse connections due to small convolution kernel



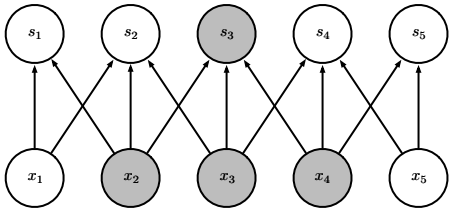
Dense connections



(Goodfellow 2016)

Sparse connectivity: being affected by less

Sparse connections due to small convolution kernel



Dense connections

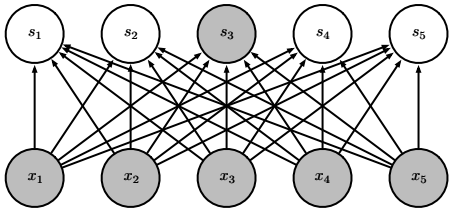


Figure 9.3

(Goodfellow 2016)

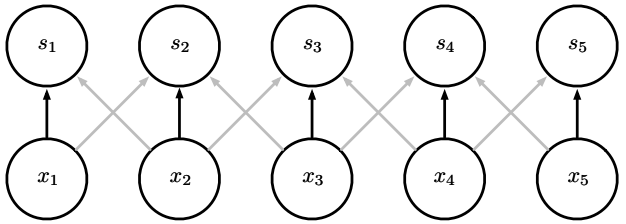
Connection to fully connected NNs

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

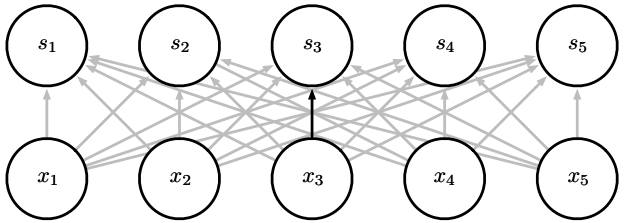


Figure 9.5

(Goodfellow 2016)

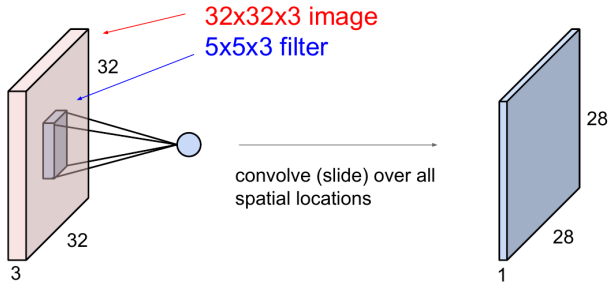
Connection to fully connected NNs

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

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

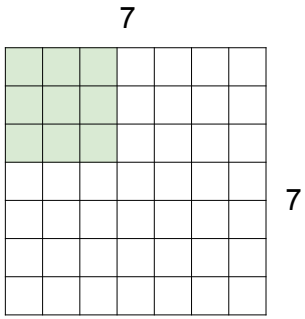
Much less parameters! Example (ignore bias terms):

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



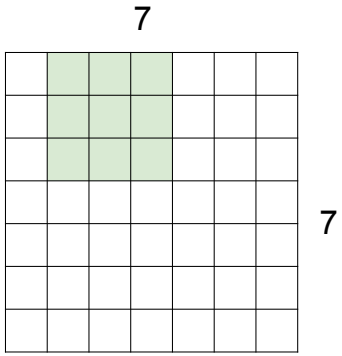
Spatial arrangement: stride and padding

A closer look at spatial dimensions:



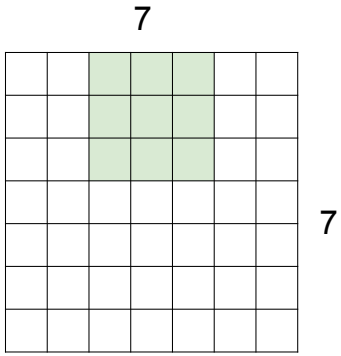
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



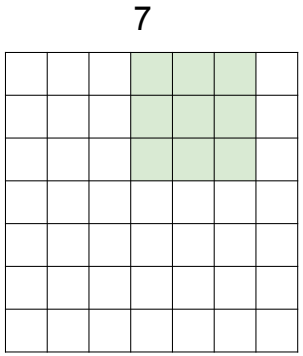
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



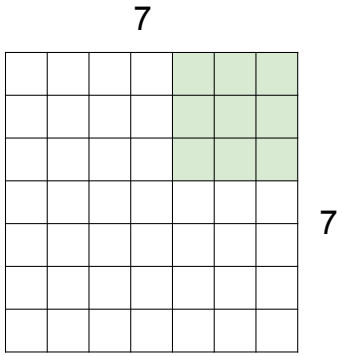
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

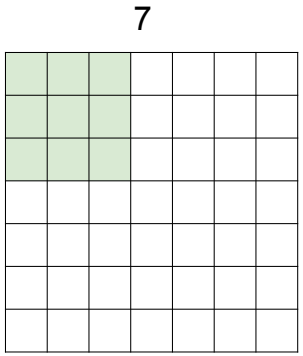
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

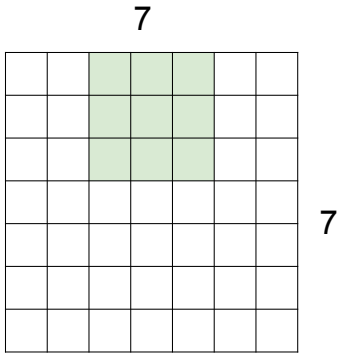
=> 5x5 output

A closer look at spatial dimensions:



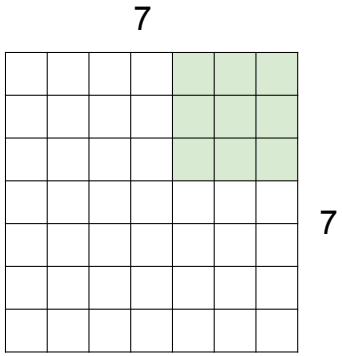
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



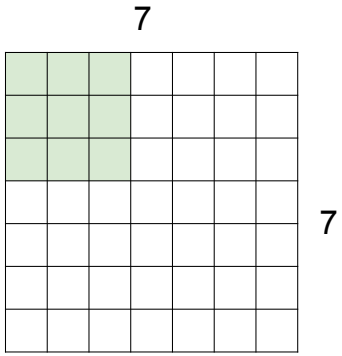
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



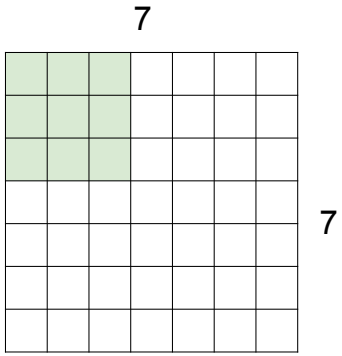
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



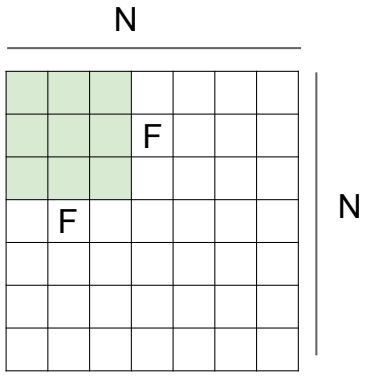
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



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

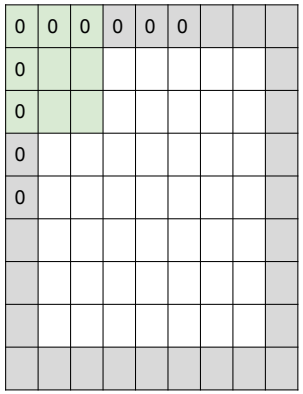
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



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 :\

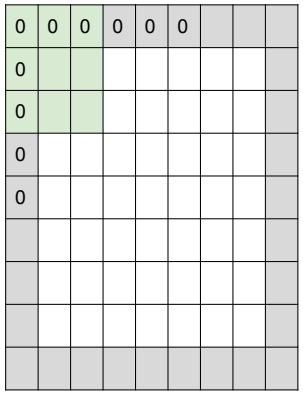
In practice: Common to zero pad the border



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

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

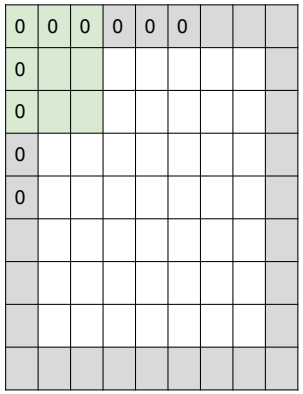
In practice: Common to zero pad the border



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

7x7 output!

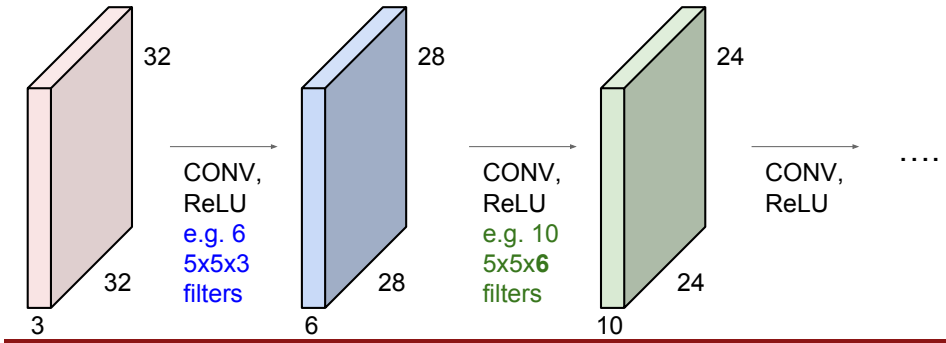
In practice: Common to zero pad the border



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

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.



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

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

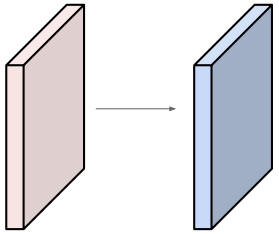
#parameters: $(F \times F \times D_1 + 1) \times K$ weights

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

Examples time:

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

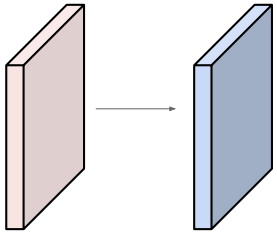
Output volume size: ?



Examples time:

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

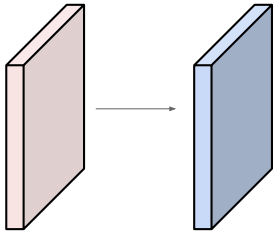
Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10



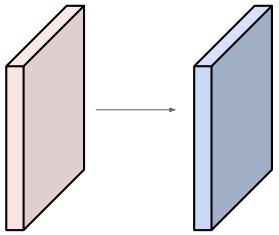
Examples time:

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

Number of parameters in this layer?



Examples time:

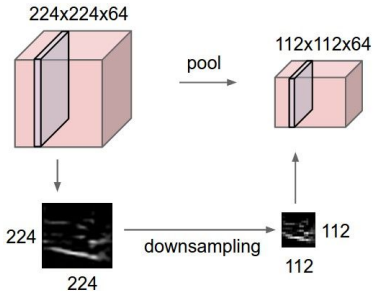


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Another element: pooling

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

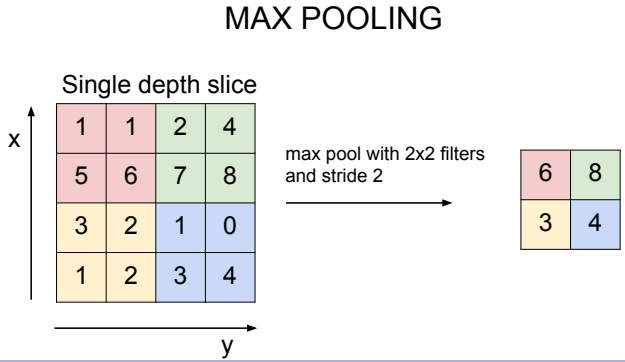


Pooling

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



Putting everything together

Typical architecture for CNNs:

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