

Lecture - 7

Convolutional Neural Network (CNN)

So far ...

- MLP are universal approximators.
- Backpropagation is used to compute gradients
- Variants of GD is used to find optimal weights in NN

How do we process images?



A

OR



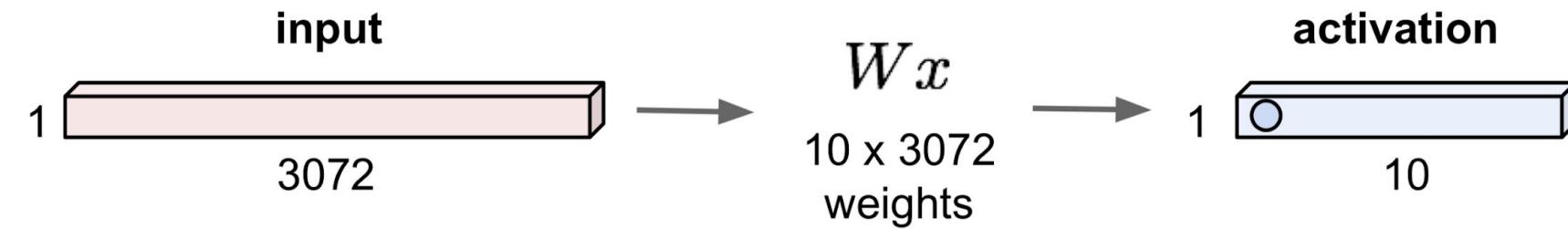
B

Slide credits:

<http://cs231n.stanford.edu/syllabus.html>

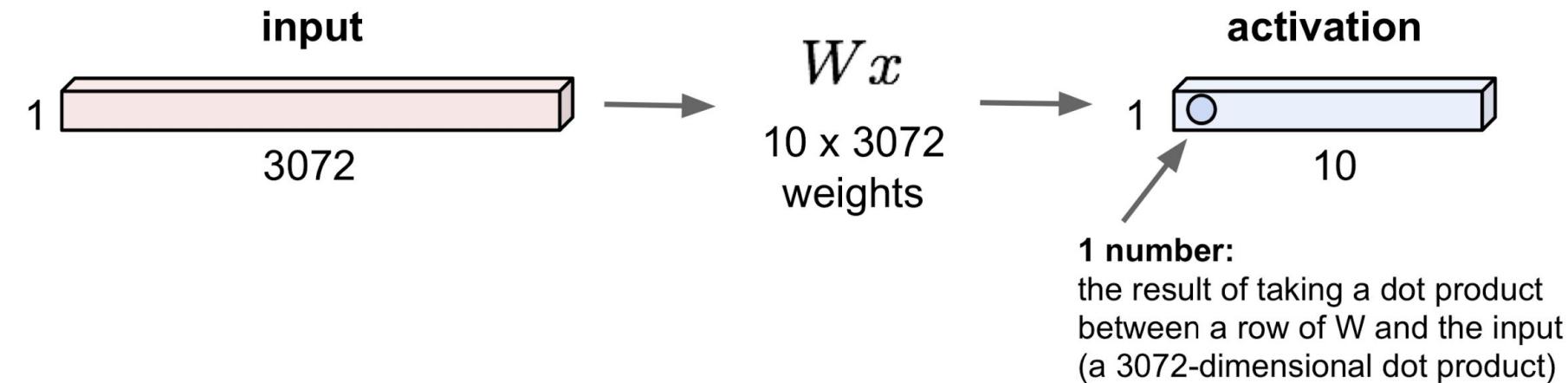
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



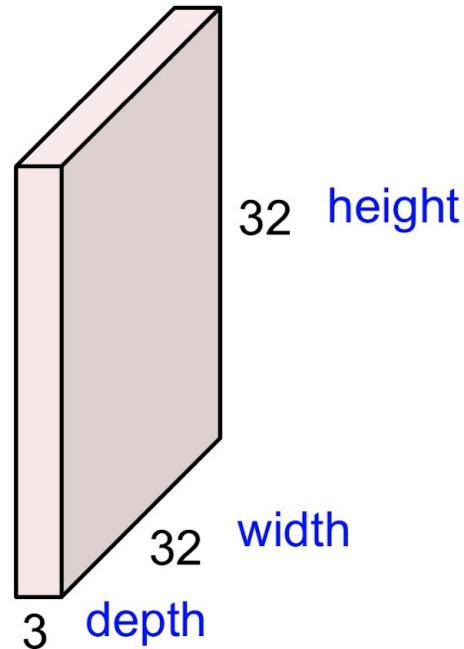
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



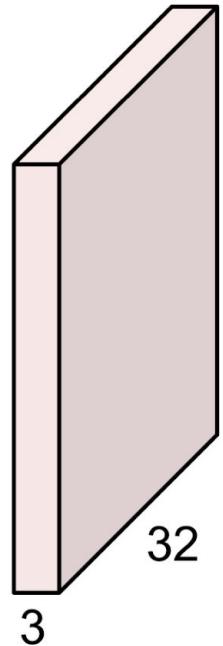
Convolution Layer

32x32x3 image -> preserve spatial structure

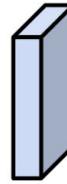


Convolution Layer

32x32x3 image



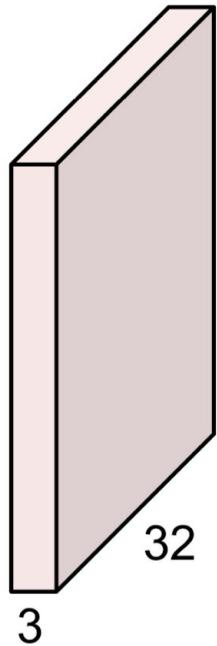
5x5x3 filter



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

Convolution Layer

32x32x3 image



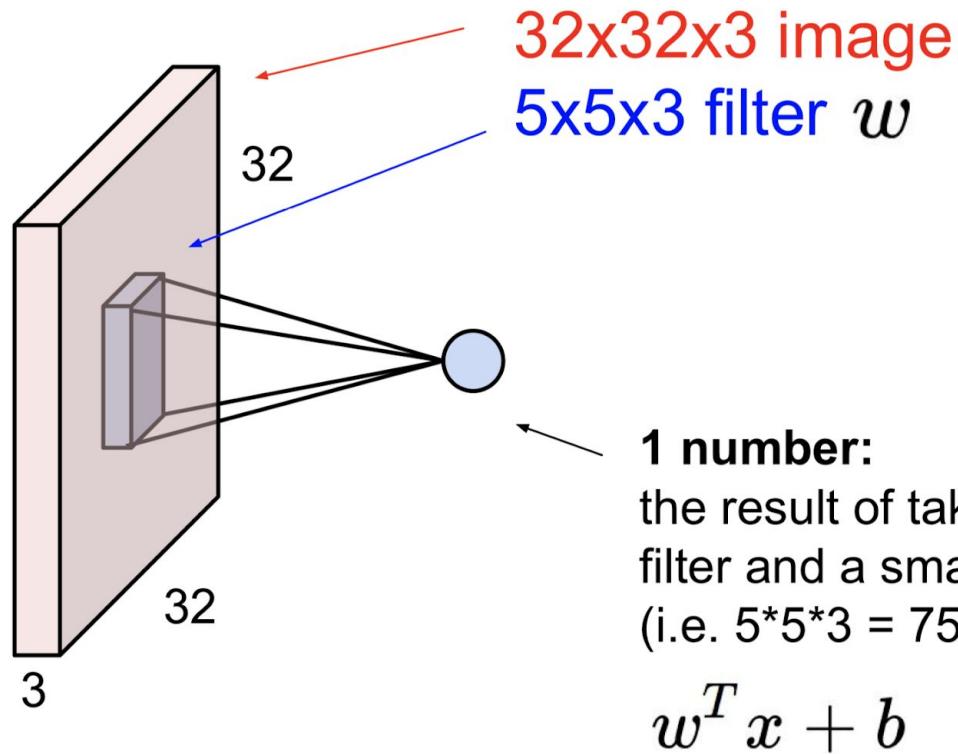
5x5x3 filter



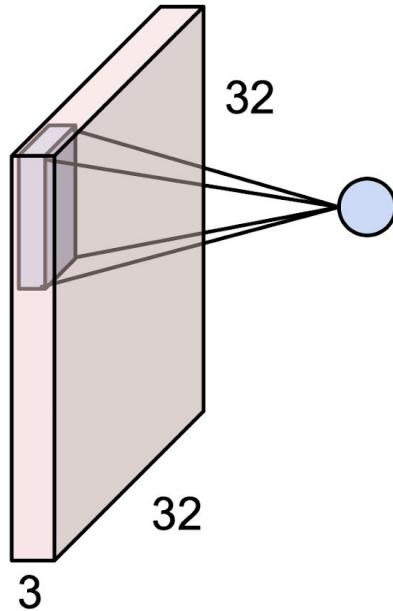
Filters always extend the full depth of the input volume

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

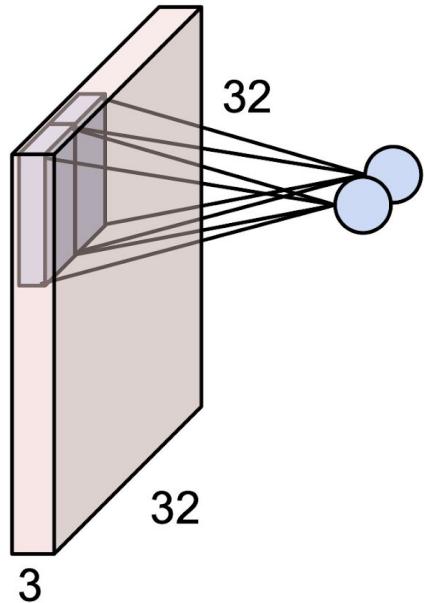
Convolution Layer



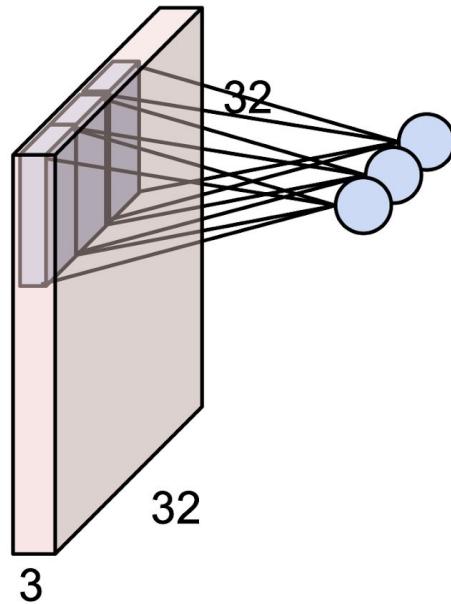
Convolution Layer



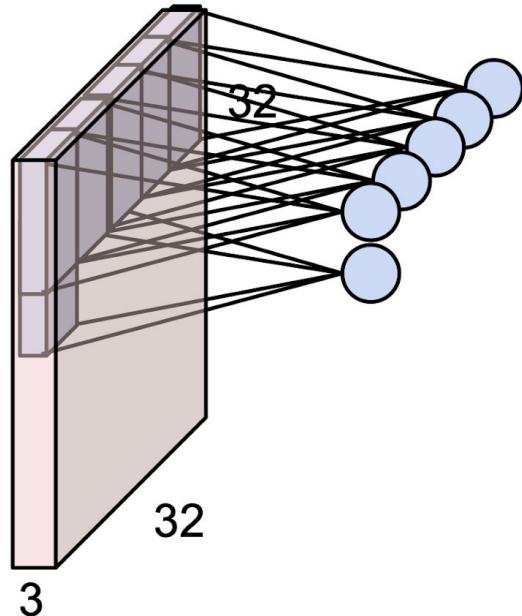
Convolution Layer



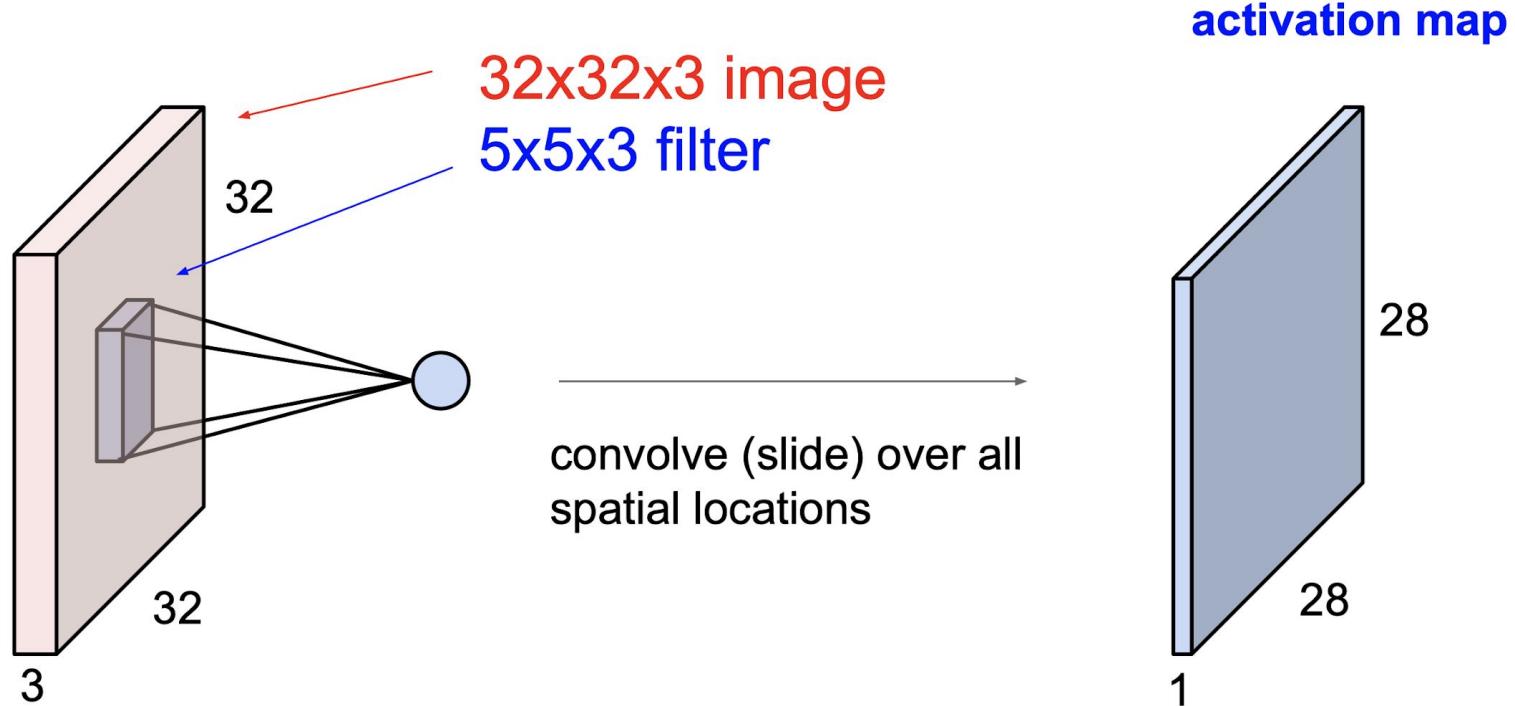
Convolution Layer



Convolution Layer



Convolution Layer

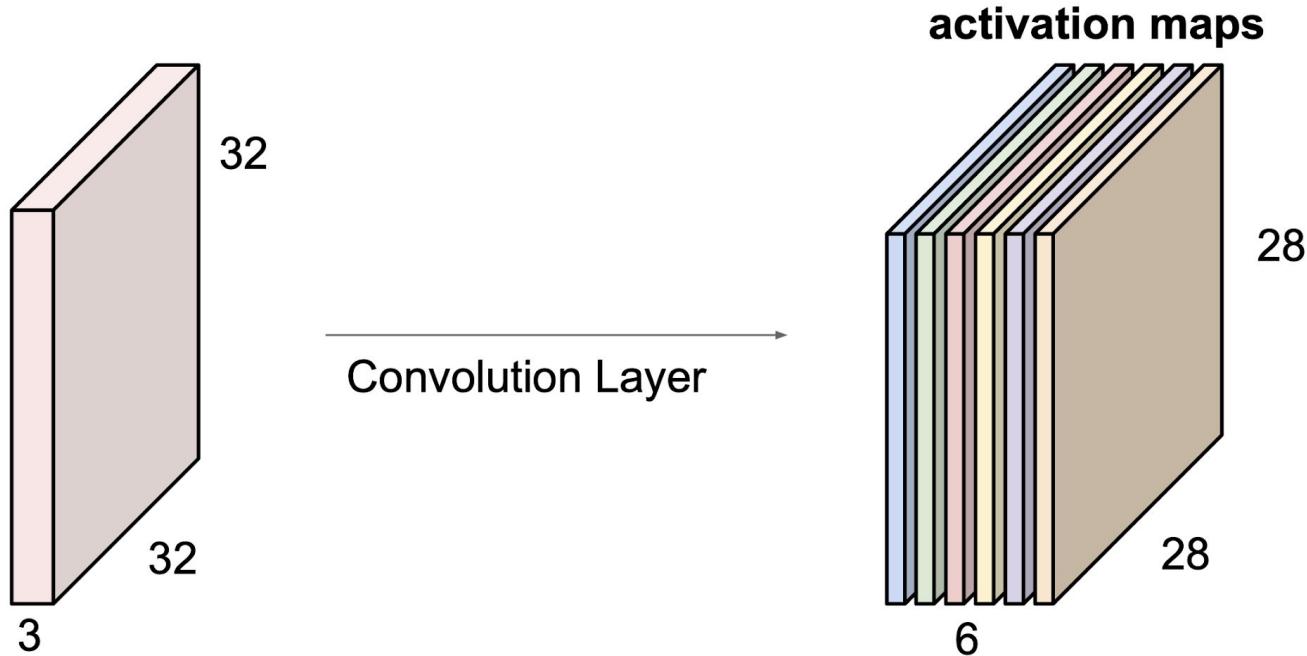


Convolution Layer

consider a second, green filter

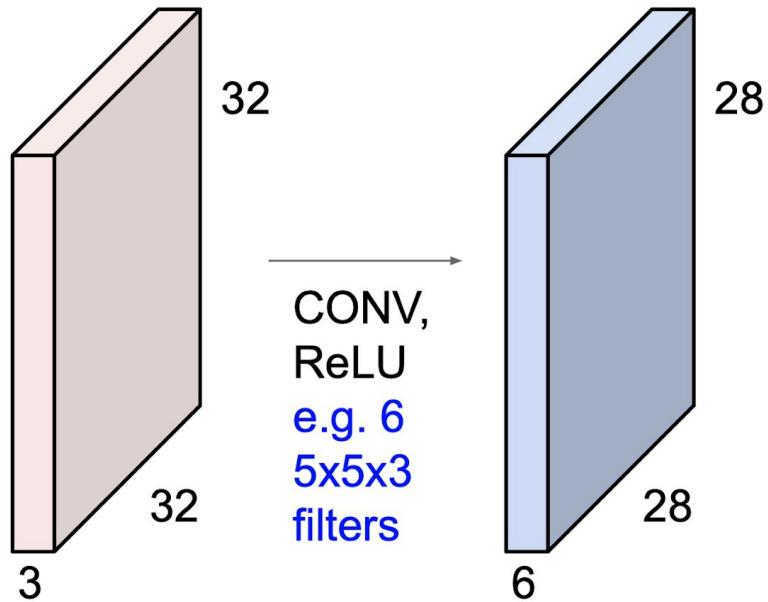


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

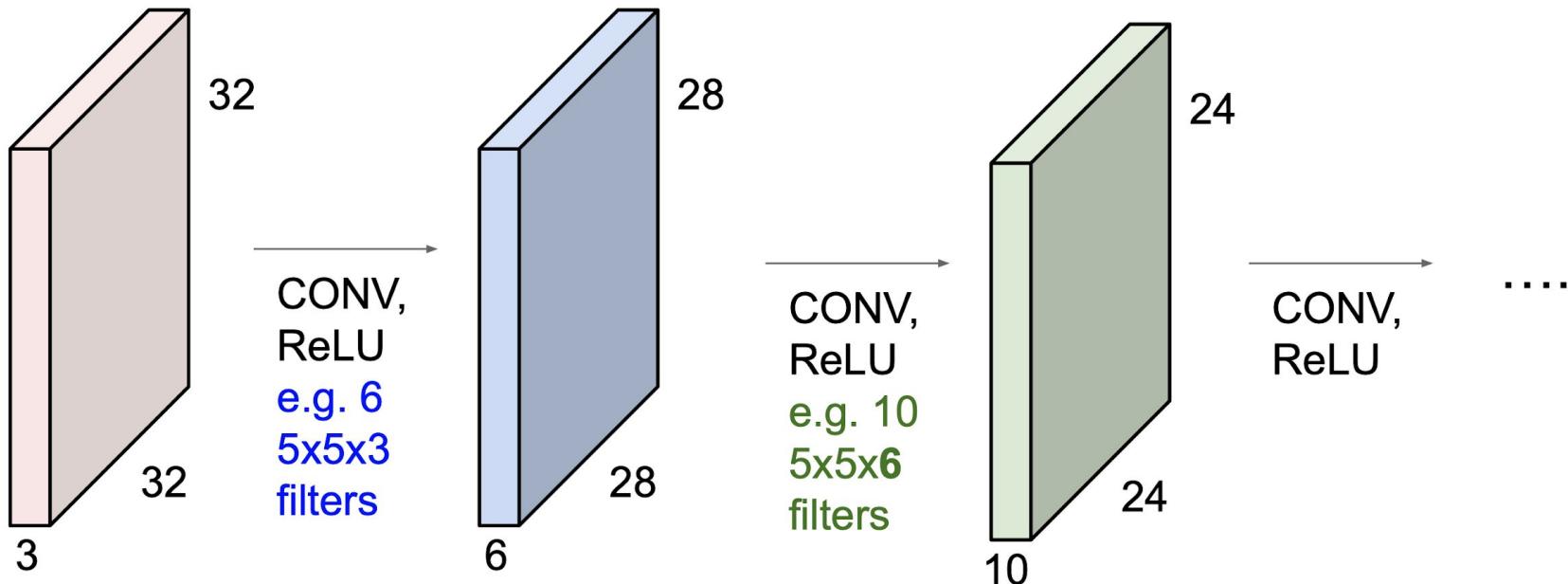


We stack these up to get a “new image” of size $28 \times 28 \times 6$!

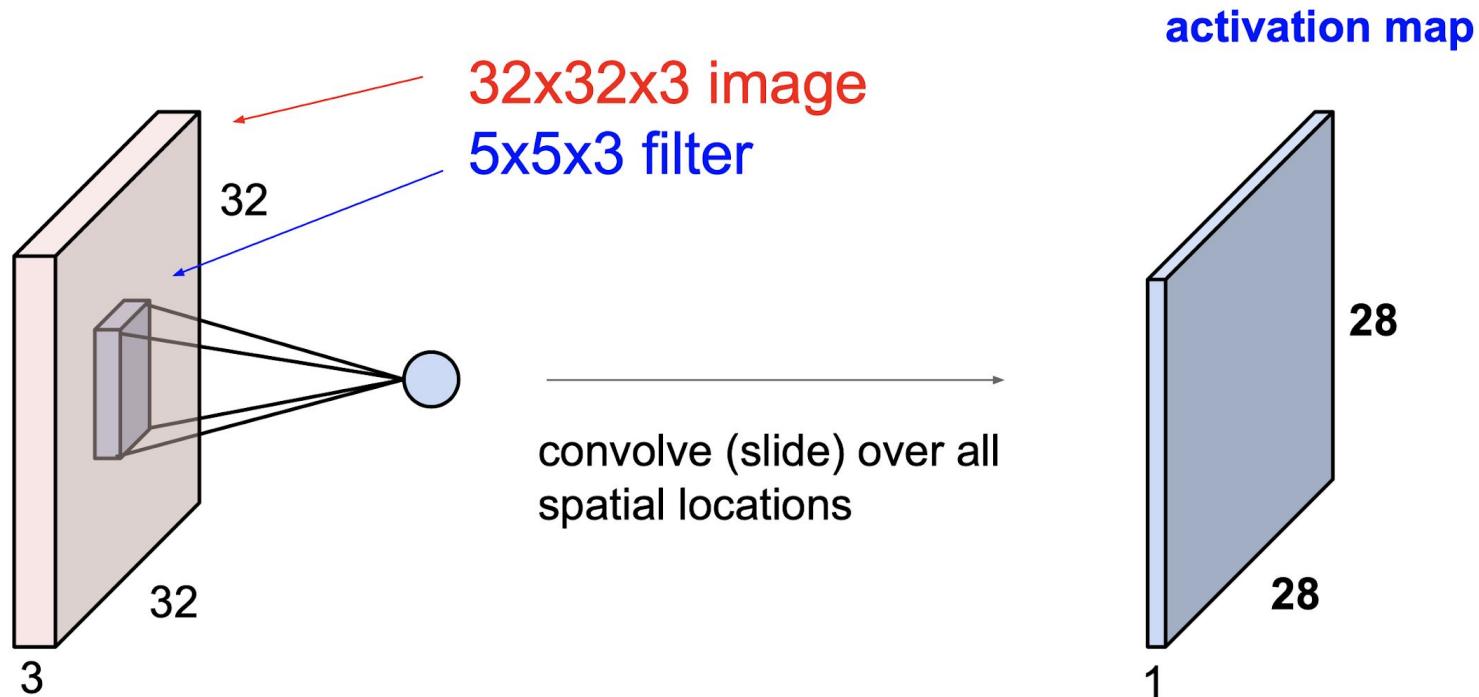
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



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

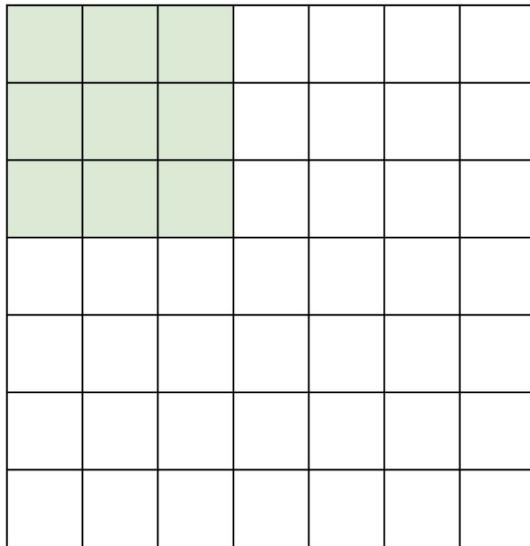


A closer look at spatial dimensions:



A closer look at spatial dimensions:

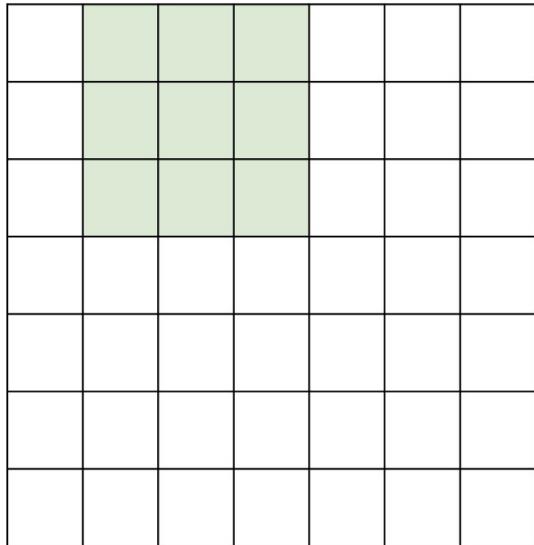
7



7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

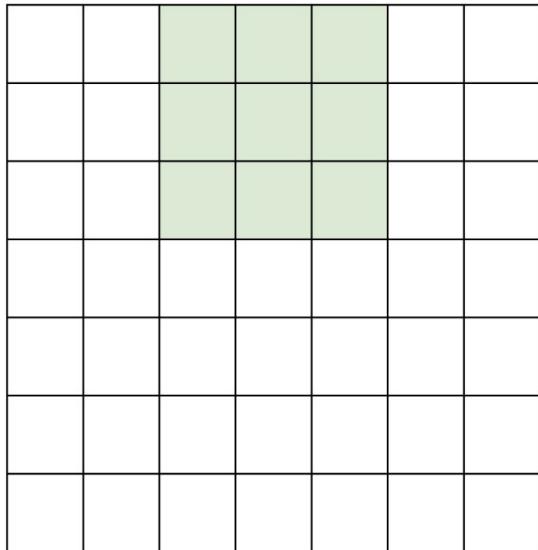
7



7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

7

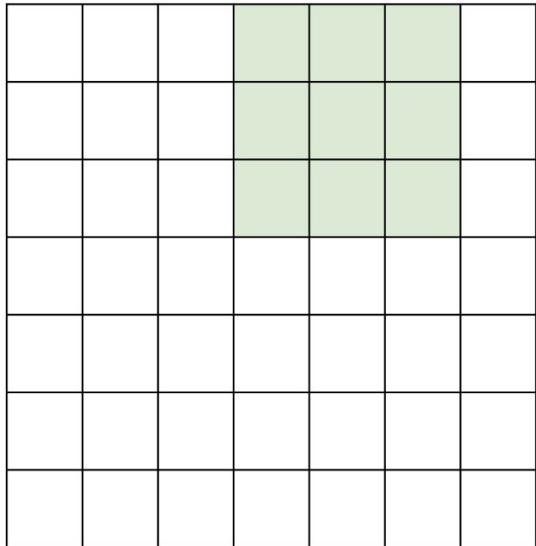


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

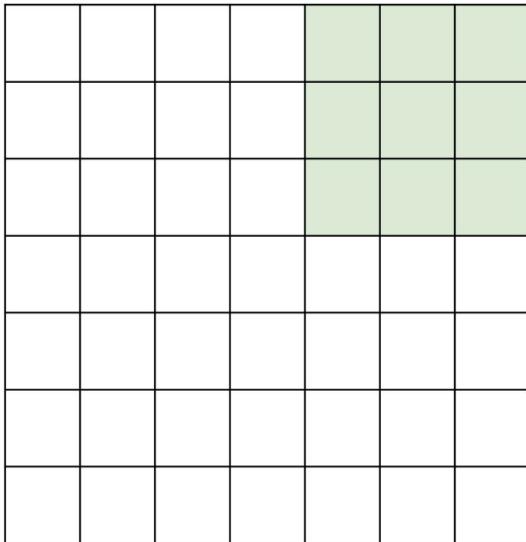


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

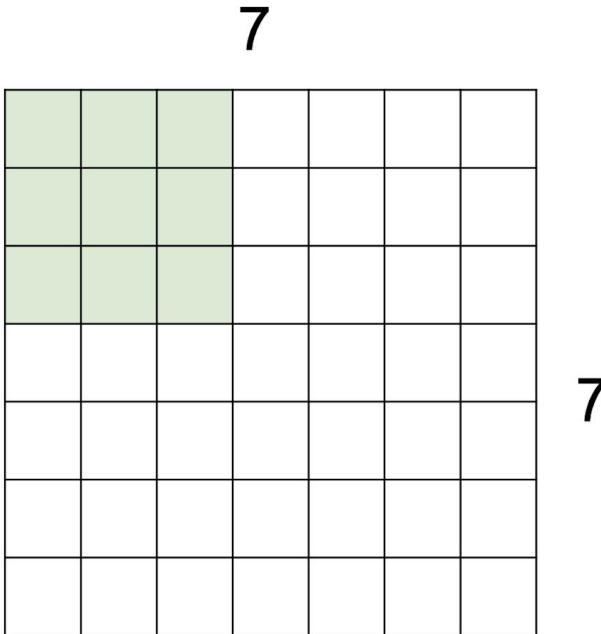


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

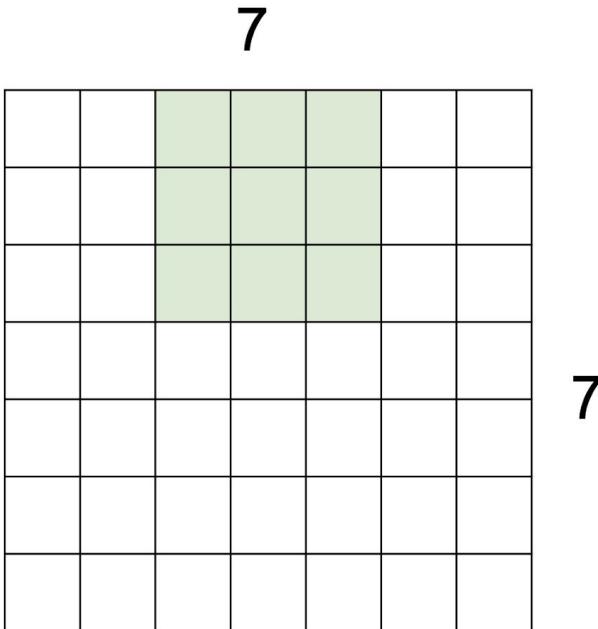
7

A closer look at spatial dimensions:



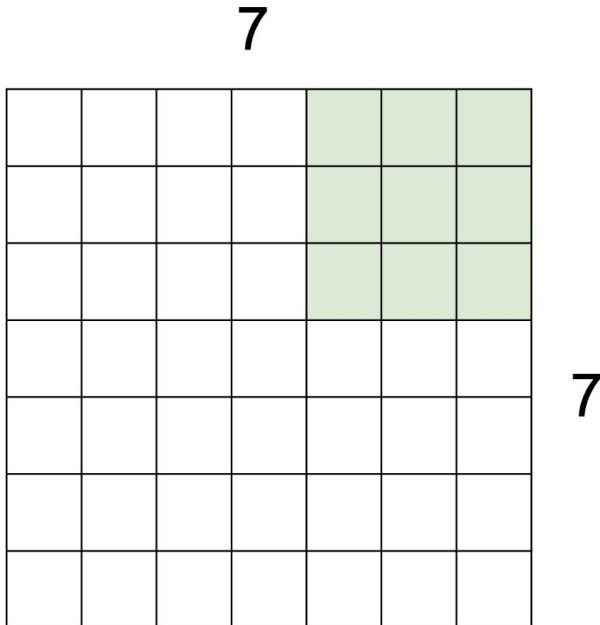
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



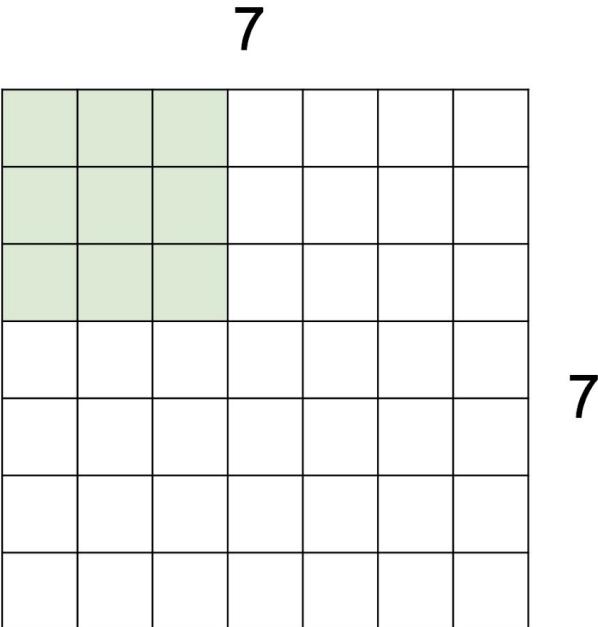
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



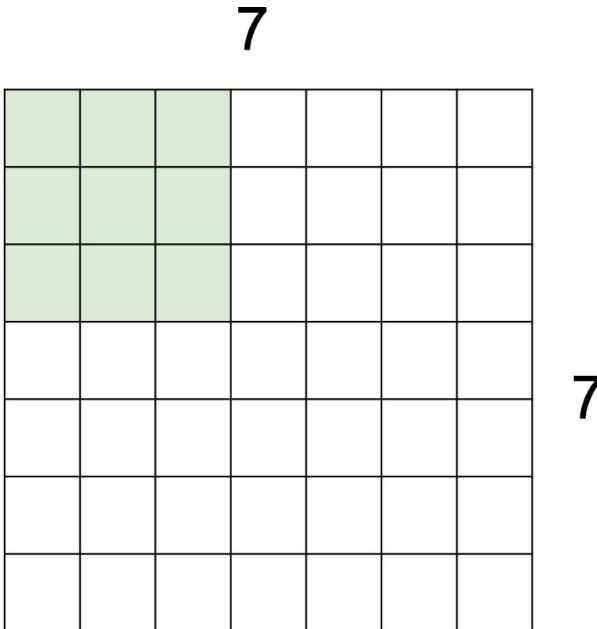
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



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

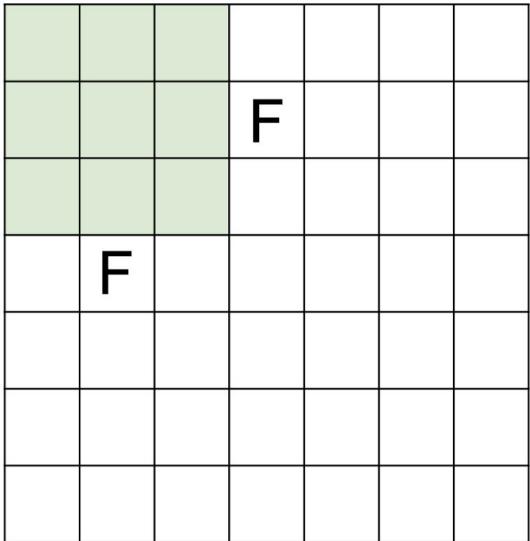
A closer look at spatial dimensions:



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

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

N



N

Output size:

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

e.g. $N = 7, F = 3$:

$$\text{stride } 1 \Rightarrow (7 - 3)/1 + 1 = 5$$

$$\text{stride } 2 \Rightarrow (7 - 3)/2 + 1 = 3$$

$$\text{stride } 3 \Rightarrow (7 - 3)/3 + 1 = 2.33 : \backslash$$

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?

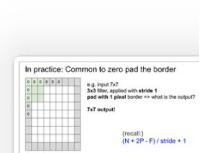
7x7 output!

in 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. $F = 3 \Rightarrow$ zero pad with 1

$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3



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!

(recall:)

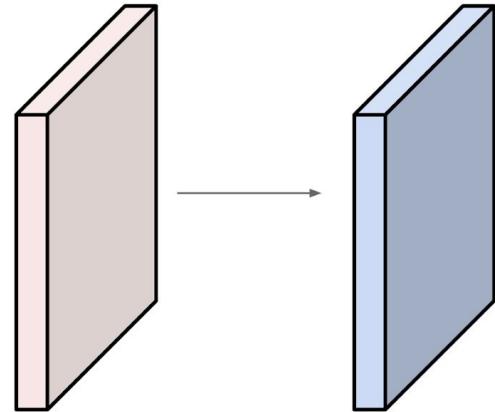
$$(N + 2P - F) / \text{stride} + 1$$

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

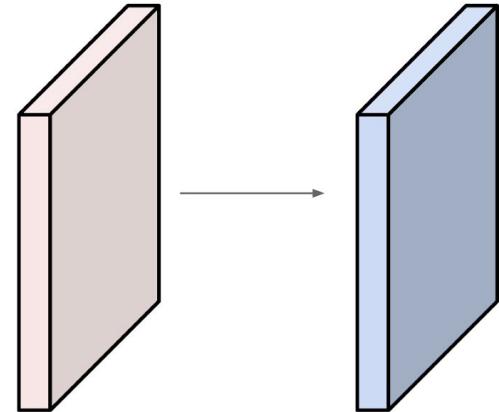
Output volume size: ?



Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride **1**, pad **2**



Output volume size:

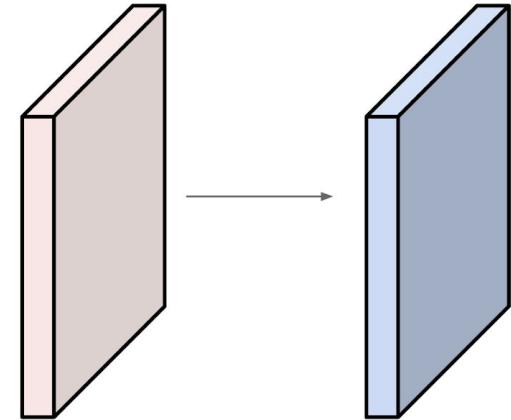
$(32+2*2-5)/1+1 = 32$ spatially, so

32x32x10

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

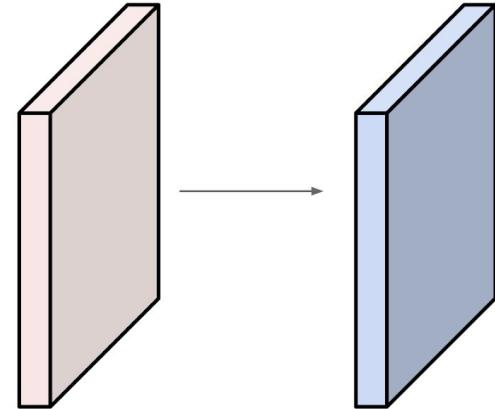


Number of parameters in this layer?

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)

$$\Rightarrow 76 * 10 = 760$$

Convolution layer: summary

Let's assume input is $W_1 \times H_1 \times C$

Conv layer needs 4 hyperparameters:

- Number of filters K
- The filter size F
- The stride S
- The zero padding P

This will produce an output of $W_2 \times H_2 \times K$

where:

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

Number of parameters: F^2CK and K biases

Convolution layer: summary

Let's assume input is $W_1 \times H_1 \times C$

Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of $W_2 \times H_2 \times K$

where:

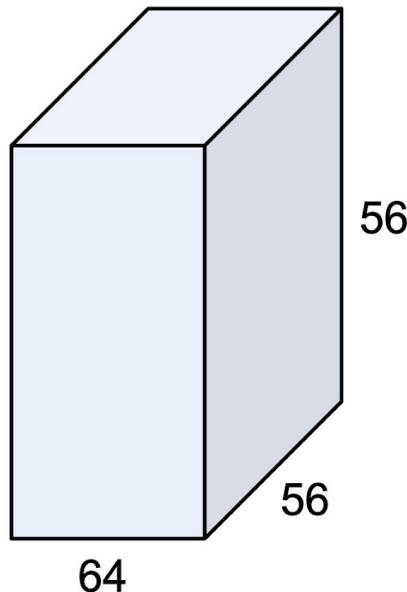
- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

Common settings:

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

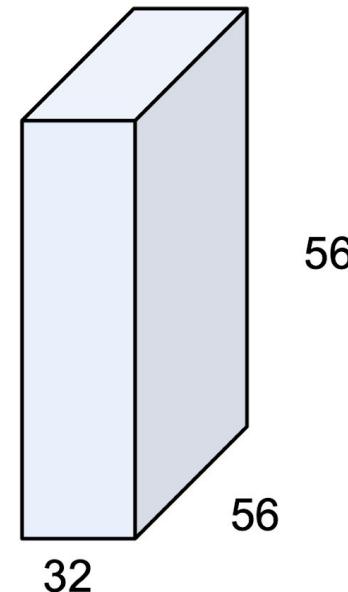
(btw, 1x1 convolution layers make perfect sense)



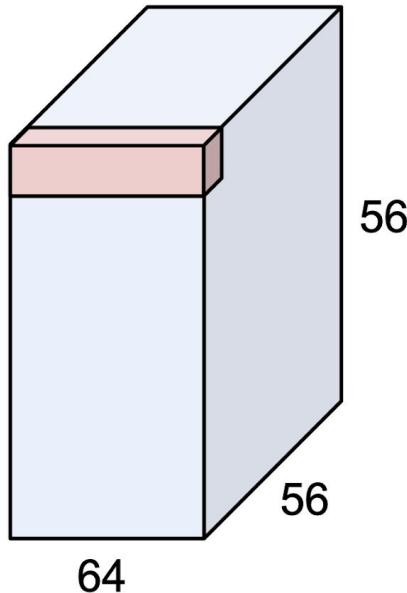
1x1 CONV
with 32 filters

→

(each filter has size
 $1 \times 1 \times 64$, and performs a
64-dimensional dot product)



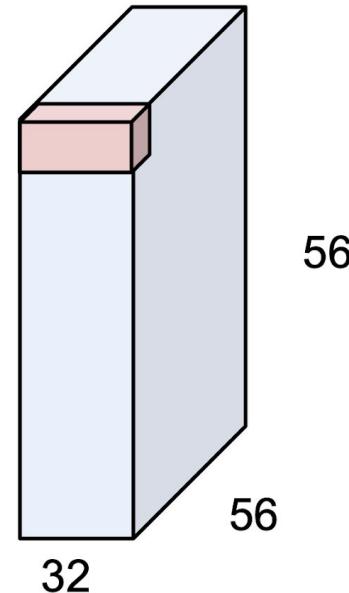
(btw, 1x1 convolution layers make perfect sense)



1x1 CONV
with 32 filters

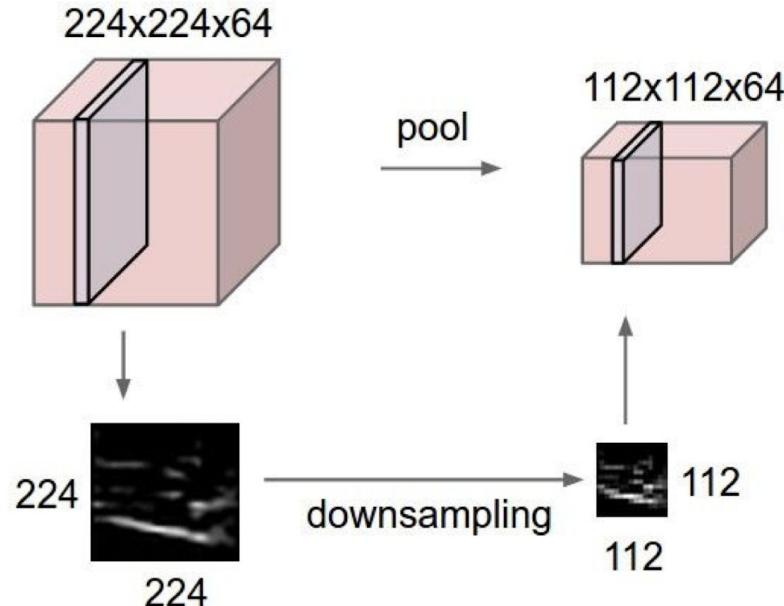
→

(each filter has size
1x1x64, and performs a
64-dimensional dot
product)



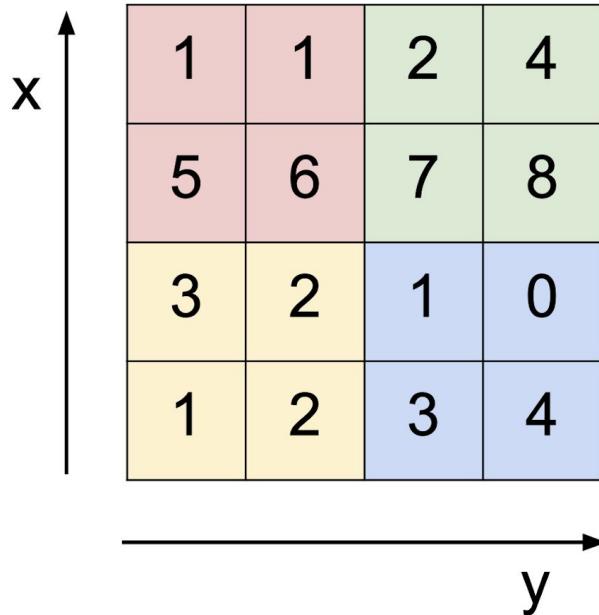
Pooling layer

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



MAX POOLING

Single depth slice



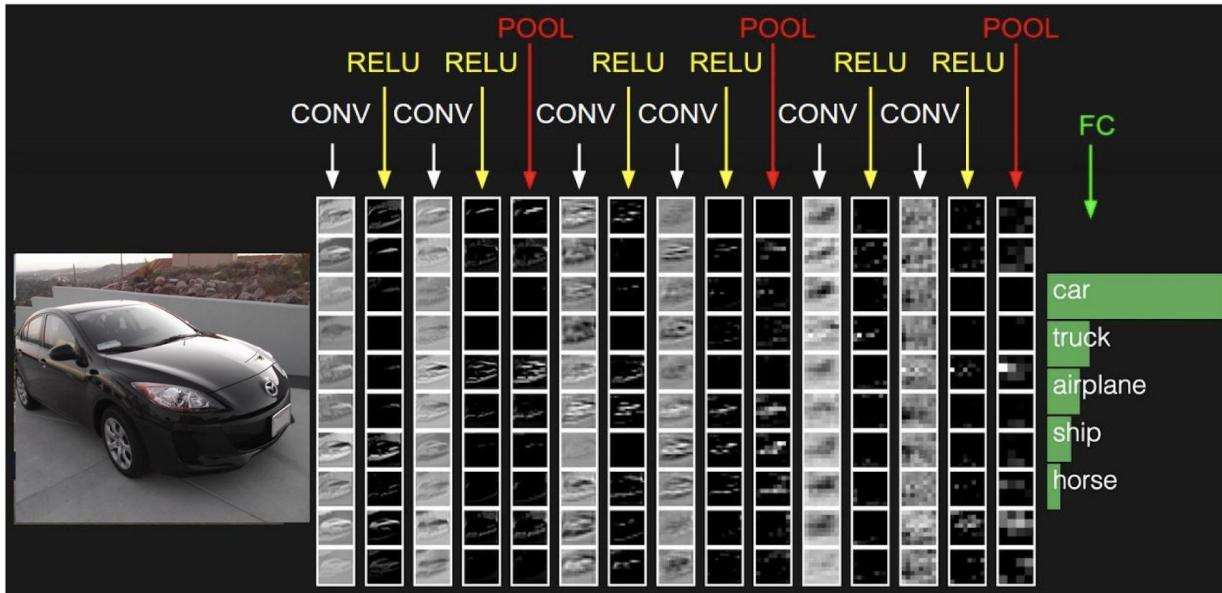
max pool with 2x2 filters
and stride 2

The output tensor is a 2x2 matrix resulting from the max pooling operation. It contains the maximum values from each 2x2 receptive field. The output is color-coded: 6 (light red), 8 (light green), 3 (light yellow), and 4 (light blue).

6	8
3	4

Fully Connected Layer (FC layer)

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



Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Historically architectures looked like
 $[(\text{CONV-RELU})^*N - \text{POOL?}]^*M - (\text{FC-RELU})^*K, \text{SOFTMAX}$
where N is usually up to ~5, M is large, $0 \leq K \leq 2$.
 - but recent advances such as ResNet/GoogLeNet have challenged this paradigm

Implementation

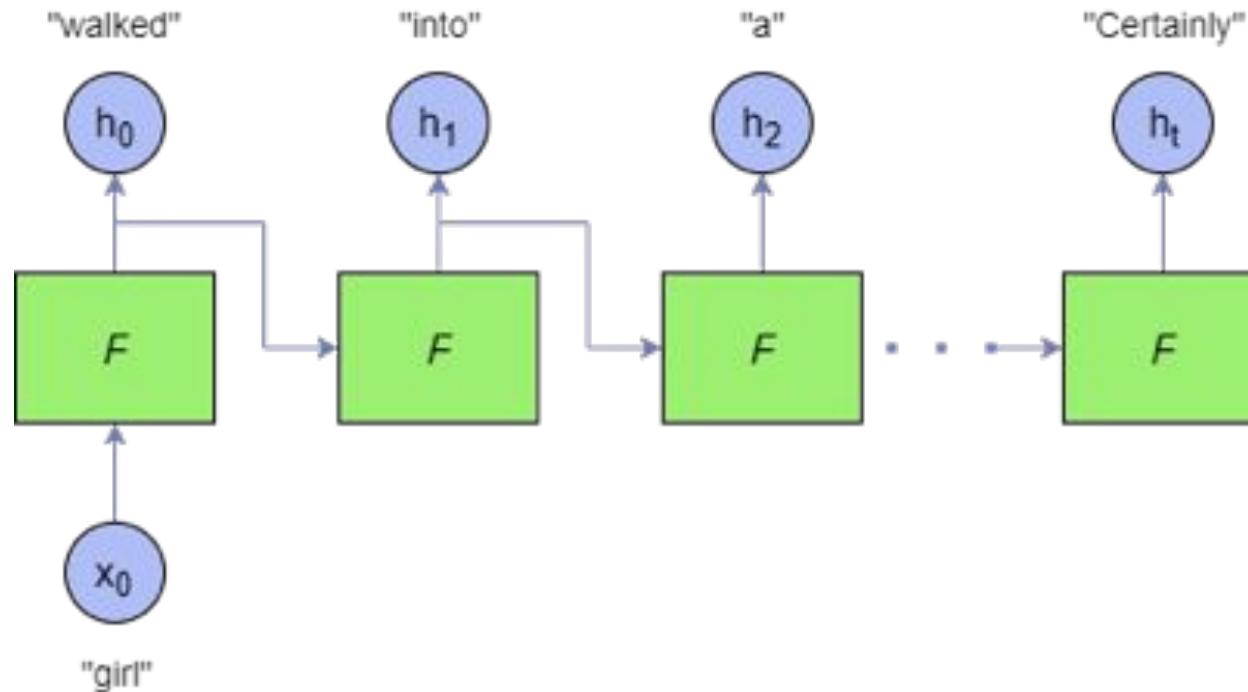
<https://github.com/bentrevett/pytorch-image-classification/blob/master/2%20-%20LeNet.ipynb>

Introduction to RNN

Seq2Seq problems

- Many ML problems can be posed as Seq2Seq problem.
Example?
- In these problems often output depends on current input and previous output.
- Traditional NNs can not capture such relationships.

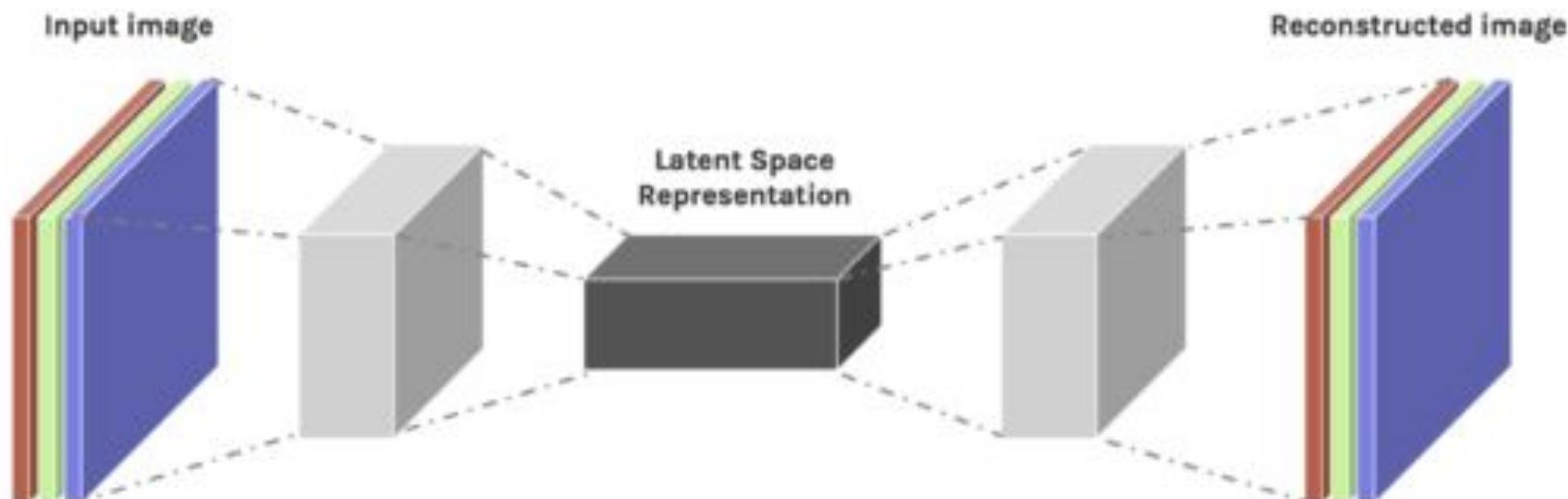
Introduction to RNN



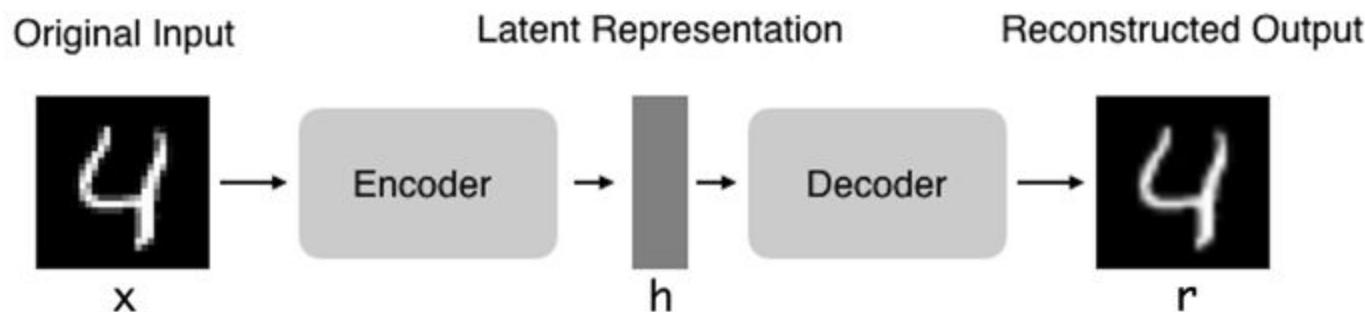
Introduction to Autoencoder

Autoencoder

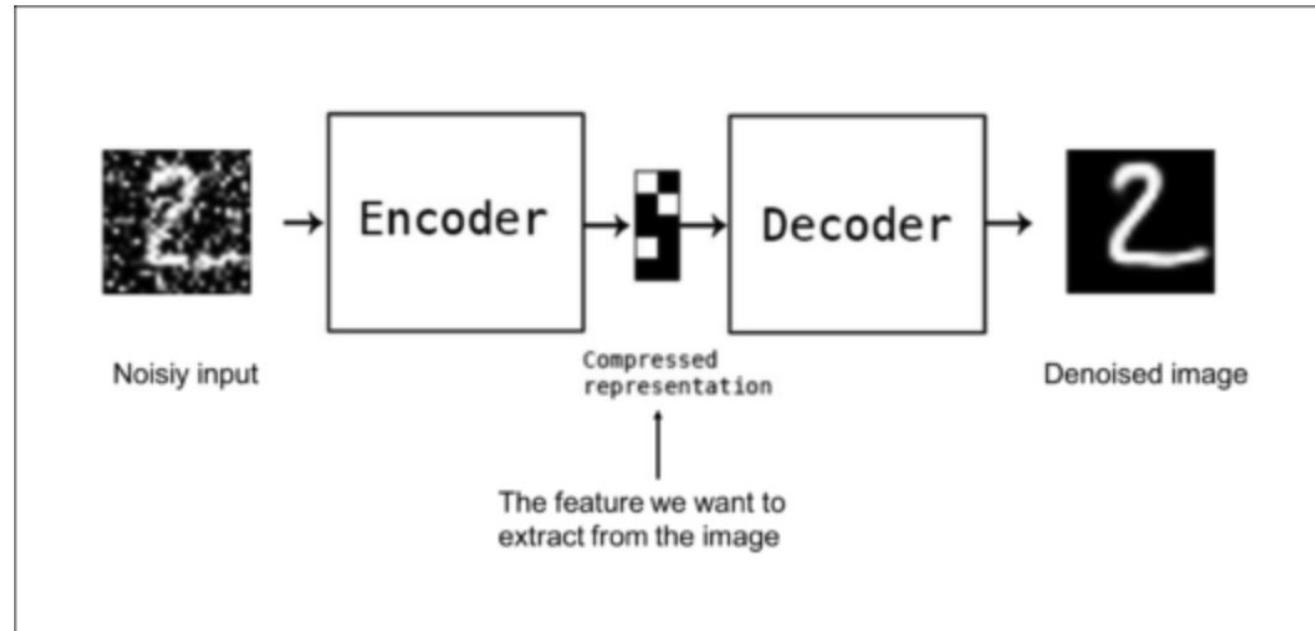
- Autoencoders are designed to reproduce their input



Autoencoder example



Autoencoder: applications (Denoising)



Autoencoder: applications (Image colorization)



Autoencoder: applications (Watermark removal)

