

Neurónové siete pre počítačové videnie

šk.r. 2019-20

4 Konvolučné neurónové siete

RNDr. Zuzana Černeková, PhD.

Ing. Viktor Kocur, PhD.

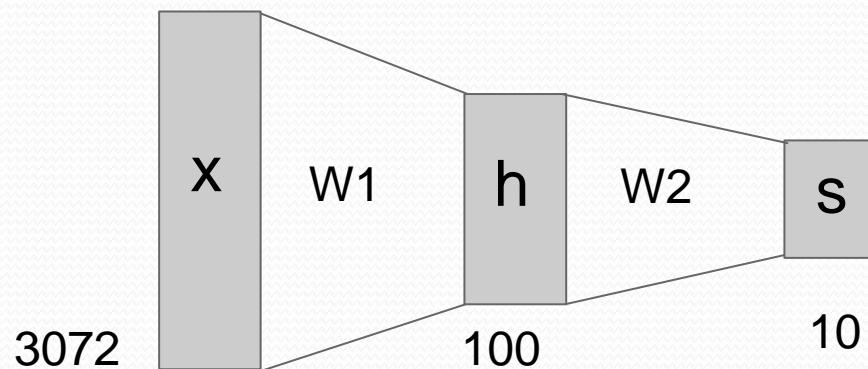
Opakovanie: Neural Networks

Linear score function:

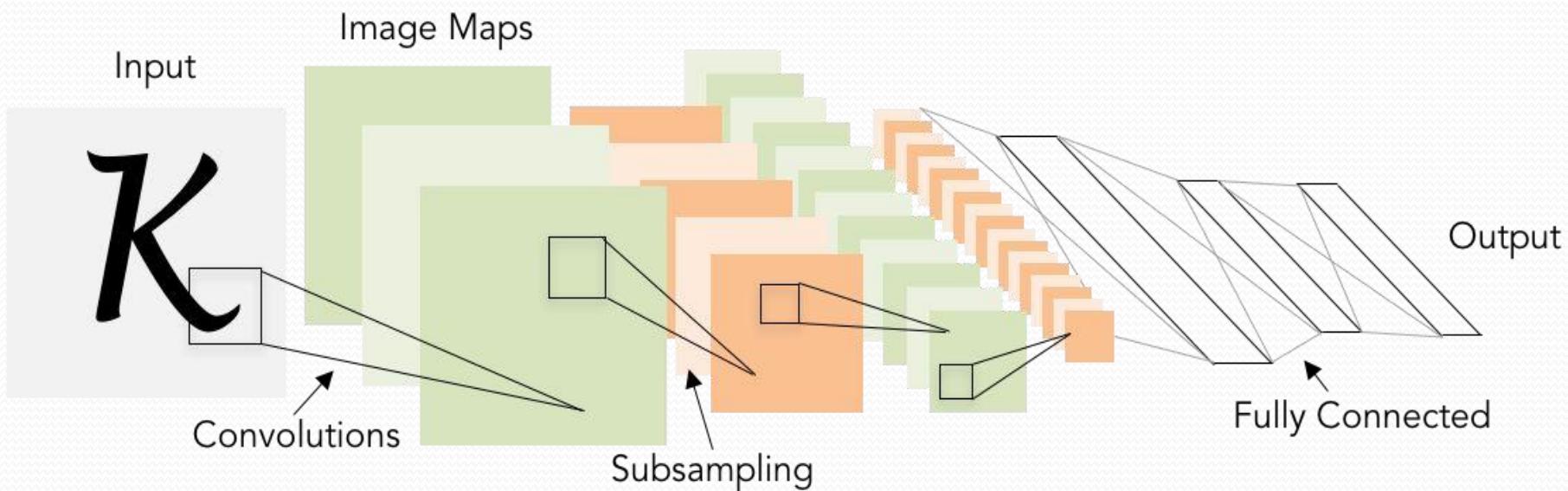
2-layer Neural Network

$$f = Wx$$

$$f = W_2 \max(0, W_1 x)$$



Next: Convolutional Neural Networks



A bit of history...

The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.

The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.

recognized
letters of the alphabet

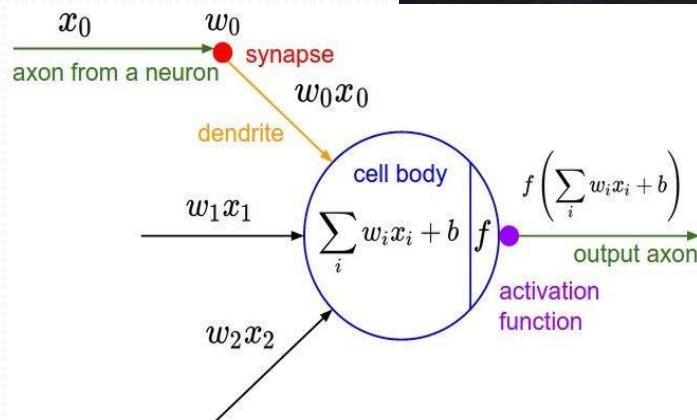
$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

update rule:

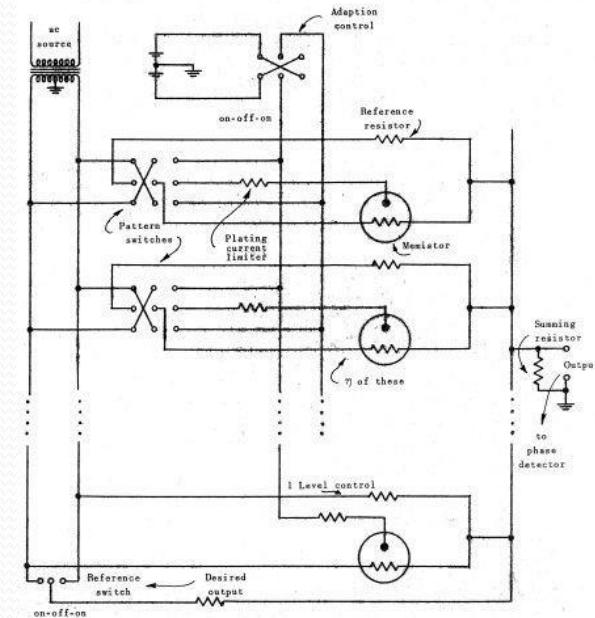
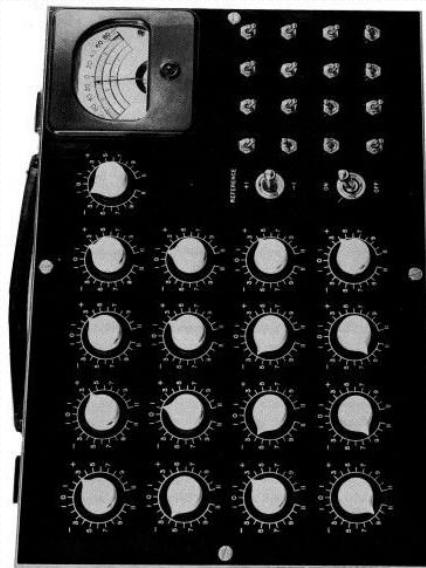
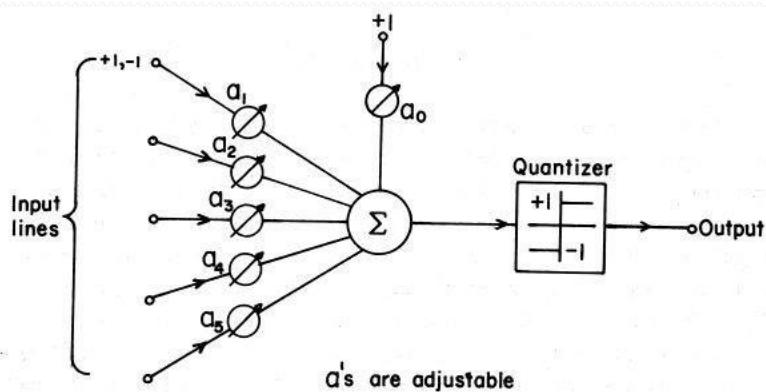
$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i},$$

Frank Rosenblatt, ~1957: Perceptron

Update rule similar to back prop but
No back propagation yet



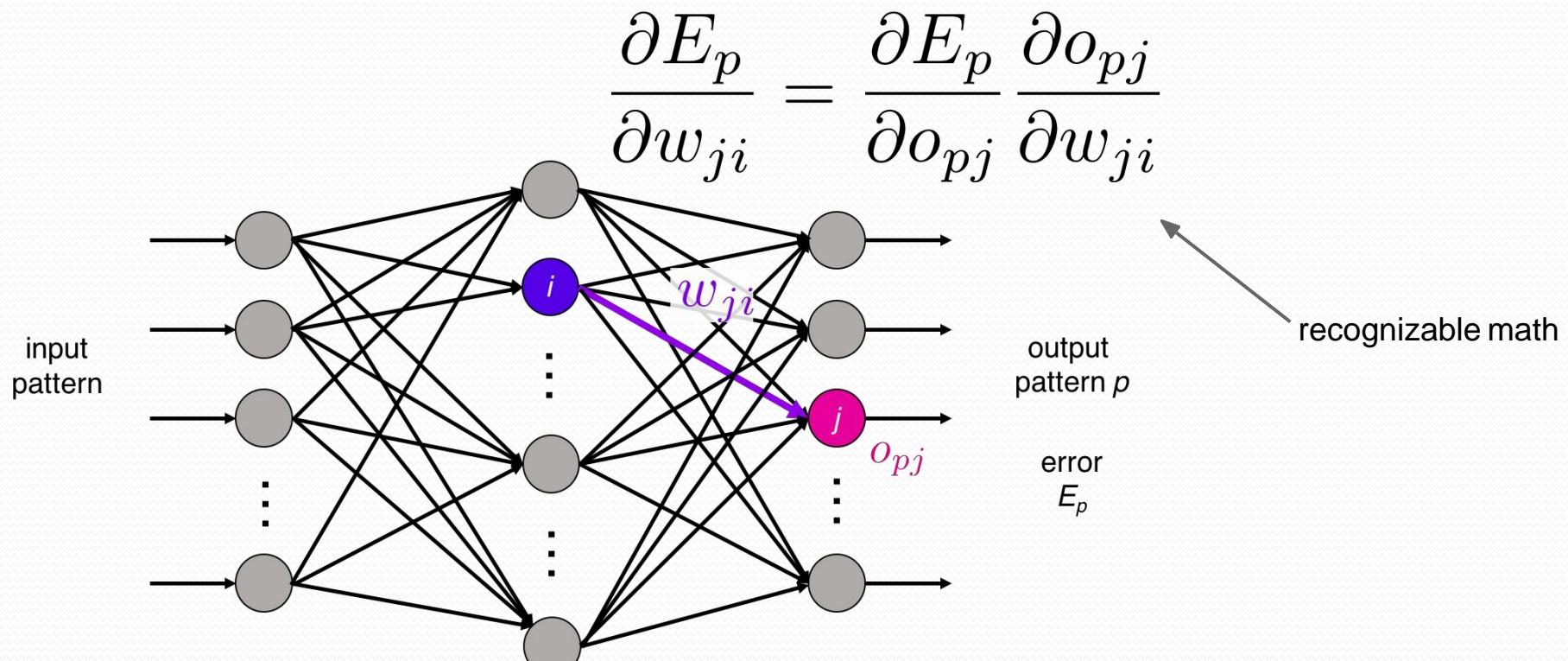
A bit of history...



Widrow and Hoff, ~1960: Adaline/Madaline

- Multilayer perceptron
- Starting to look like NN layers
- Still no back prop.

A bit of history...

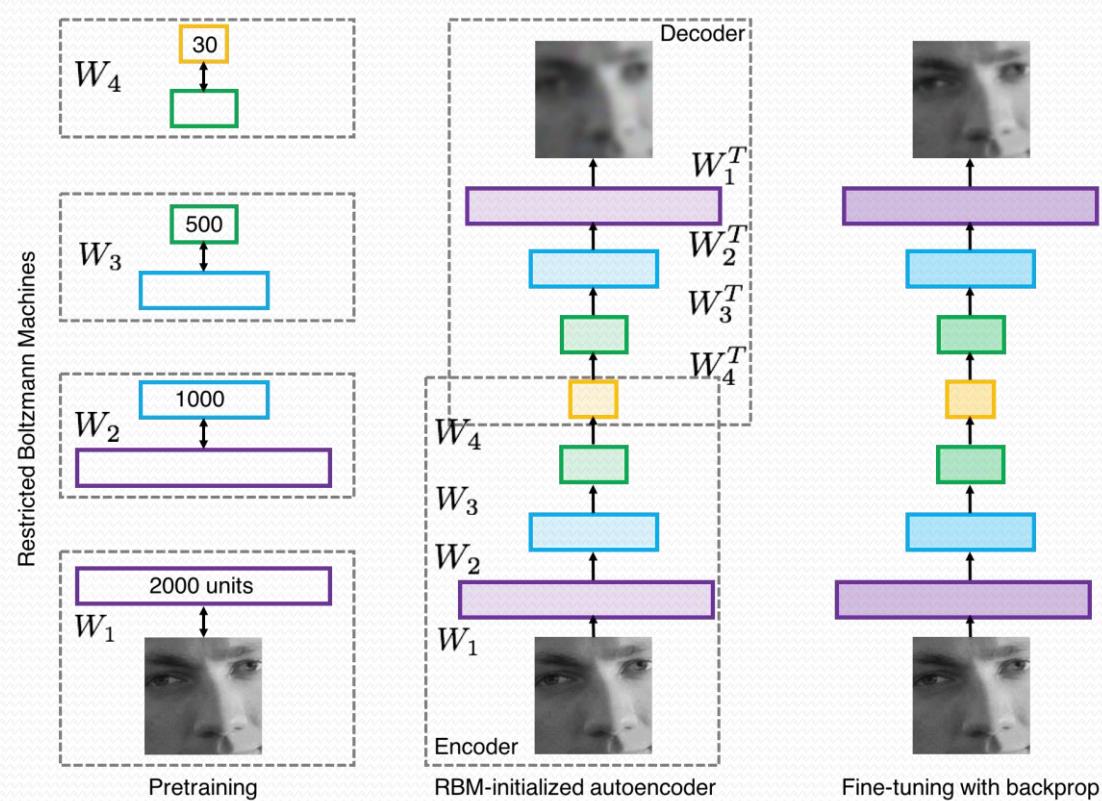


Rumelhart et al., 1986: First time back-propagation became popular

A bit of history...

[Hinton and Salakhutdinov 2006]

Reinvigorated research in
Deep Learning



First strong results

Acoustic Modeling using Deep Belief Networks

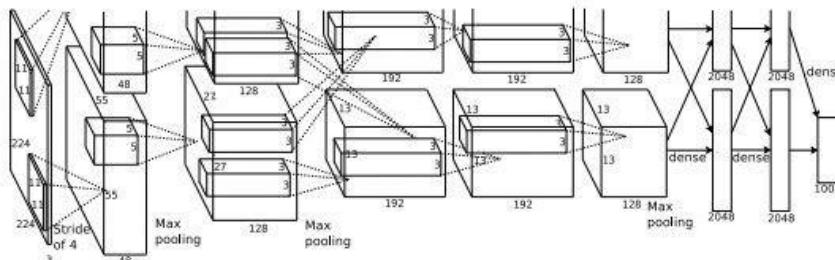
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010

Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition

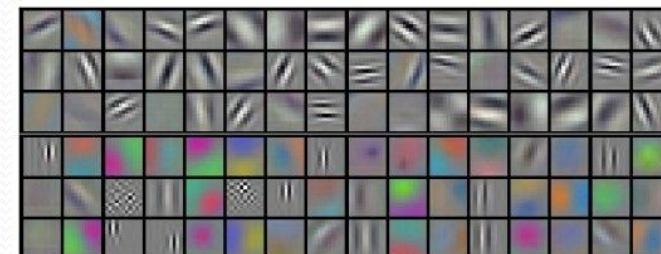
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

Imagenet classification with deep convolutional neural networks

Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012

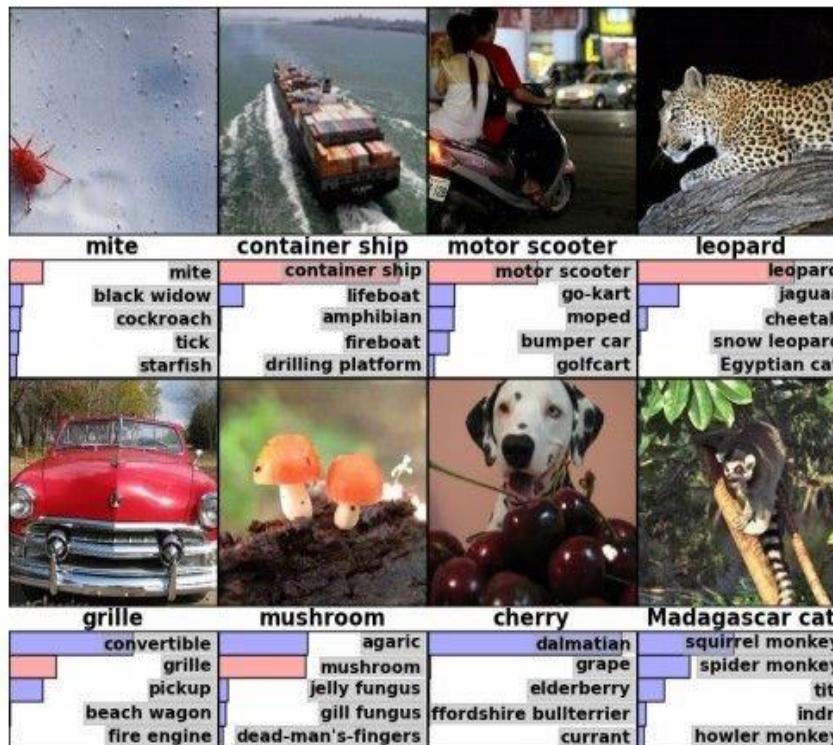


“AlexNet”



Fast-forward to today: ConvNets are everywhere

Classification

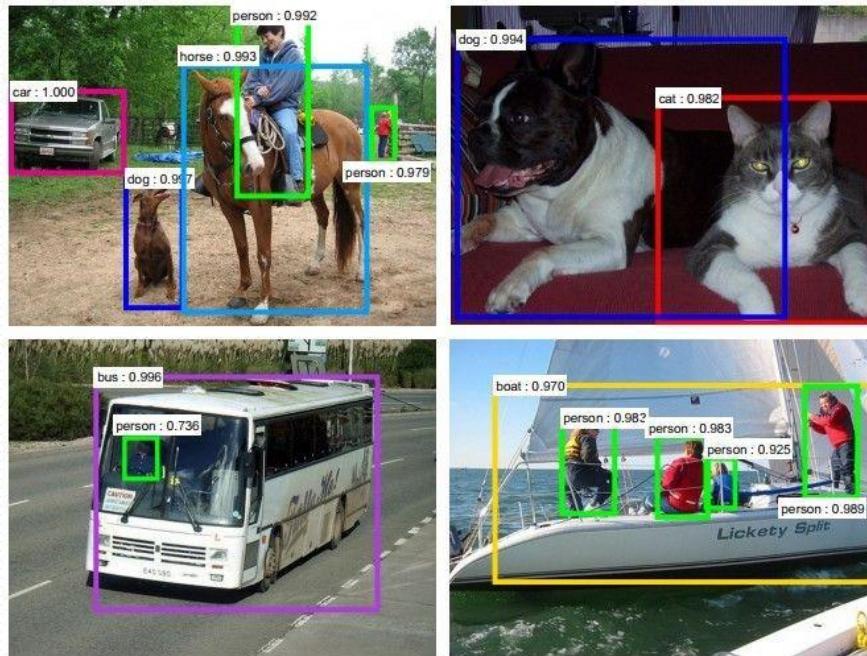


Retrieval



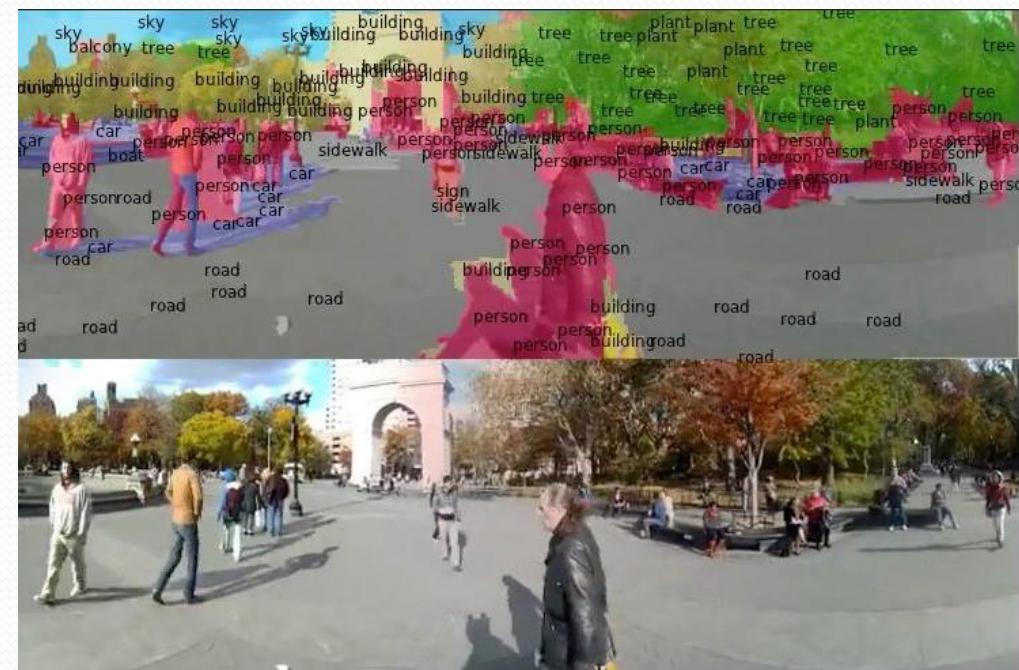
Fast-forward to today: ConvNets are everywhere

Detection



[Faster R-CNN: Ren, He, Girshick, Sun 2015]

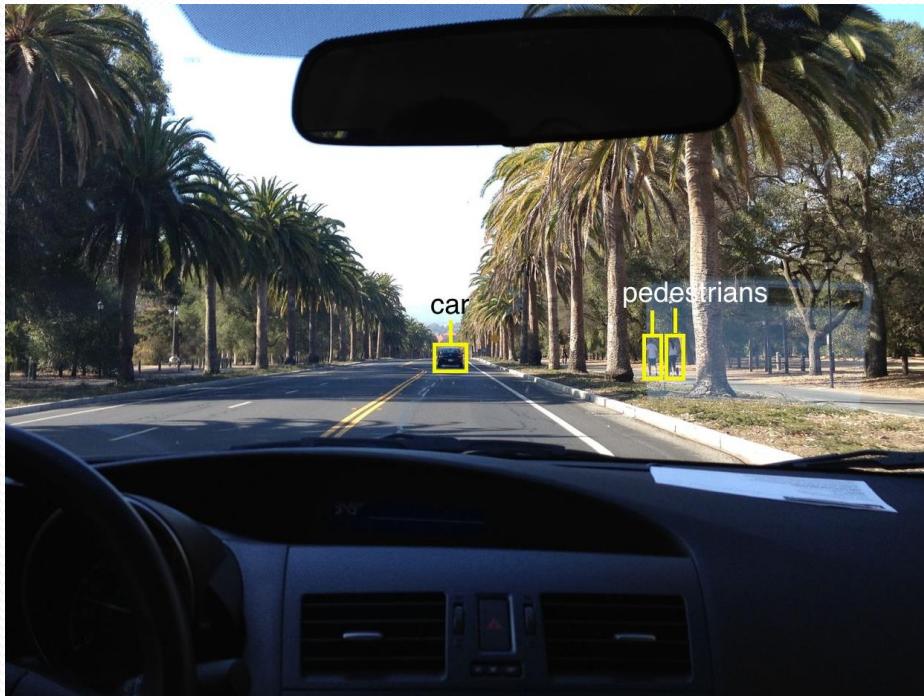
Segmentation



[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere

self-driving cars



NVIDIA Tesla line

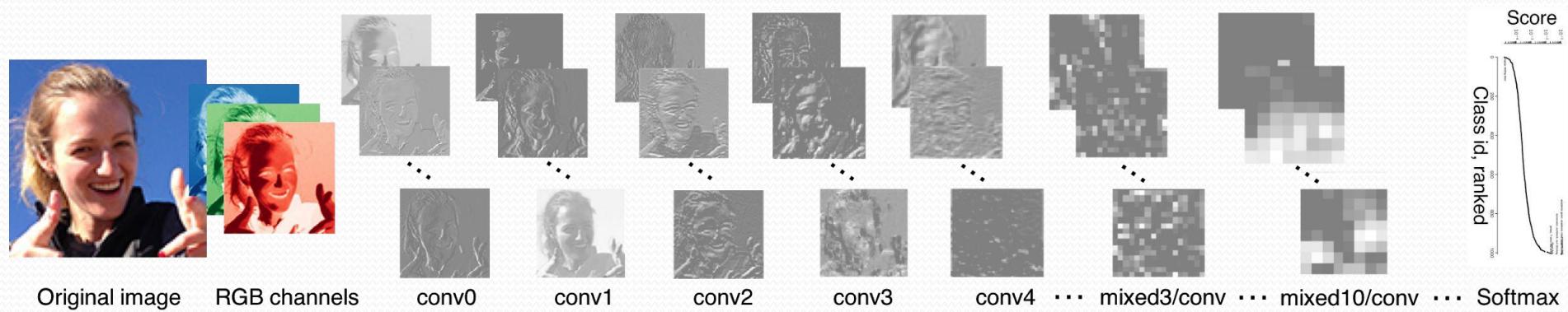
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

Fast-forward to today: ConvNets are everywhere

Pose recognition – even non standard poses

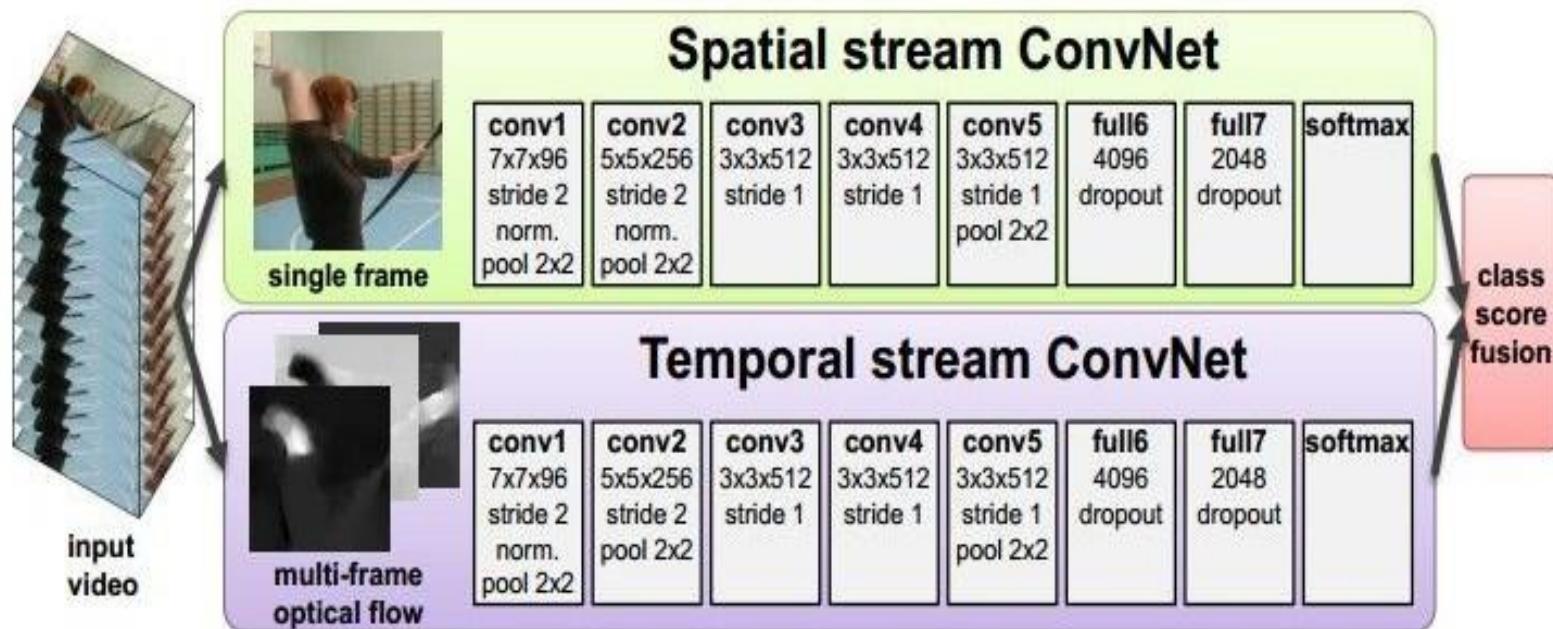


Face recognition from an image



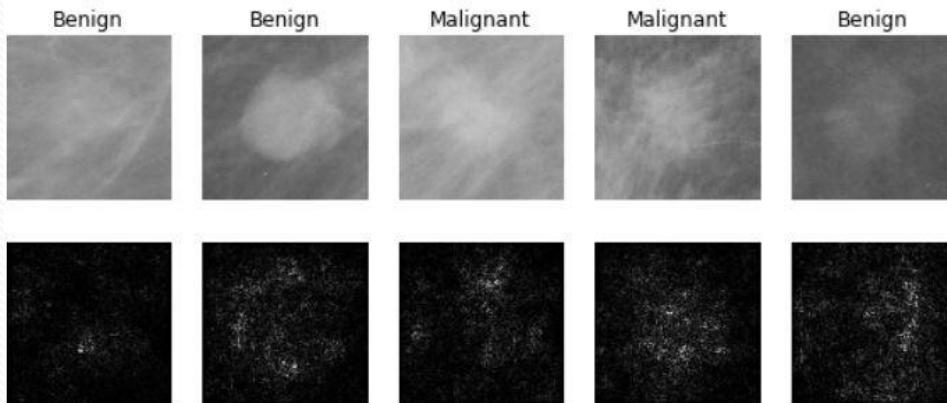
Fast-forward to today: ConvNets are everywhere

Video classification using temporal information



Fast-forward to today: ConvNets are everywhere

Interpretation and diagnostics of medical images



[Levy et al. 2016]

Street sign recognition



[Sermanet et al. 2011]

[Ciresan et al.]

Classification of galaxies



[Dieleman et al. 2014]

Fast-forward to today: ConvNets are everywhere

*Whale recognition,
Kaggle Challenge*



Streets and buildings recognition in
areal maps



Fast-forward to today: ConvNets are everywhere

Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei,
2015]

No errors



*A white teddy bear sitting in
the grass*

Minor errors



*A man in a baseball
uniform throwing a ball*

Somewhat related



*A woman is holding a
cat in her hand*



*A man riding a wave on
top of a surfboard*



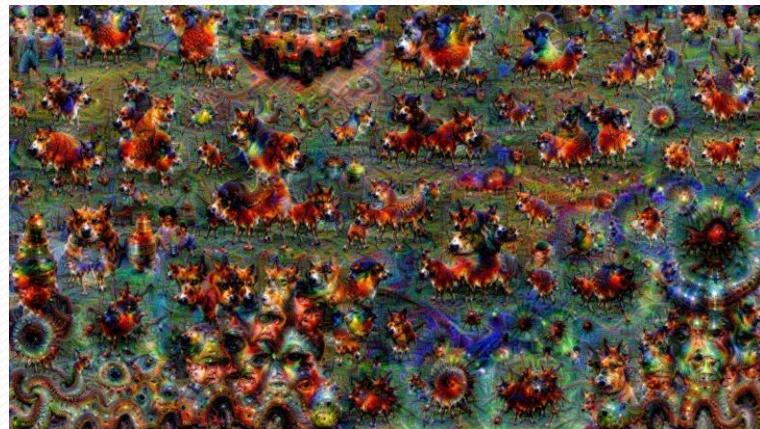
*A cat sitting on a
suitcase on the floor*



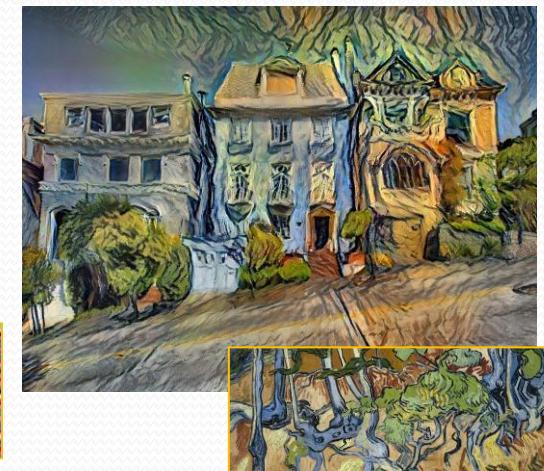
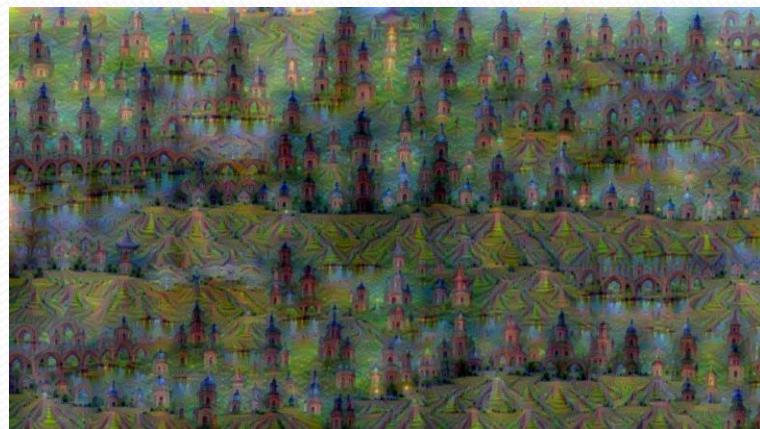
*A woman standing on a
beach holding a surfboard*

Fast-forward to today: ConvNets are everywhere

Generated art work



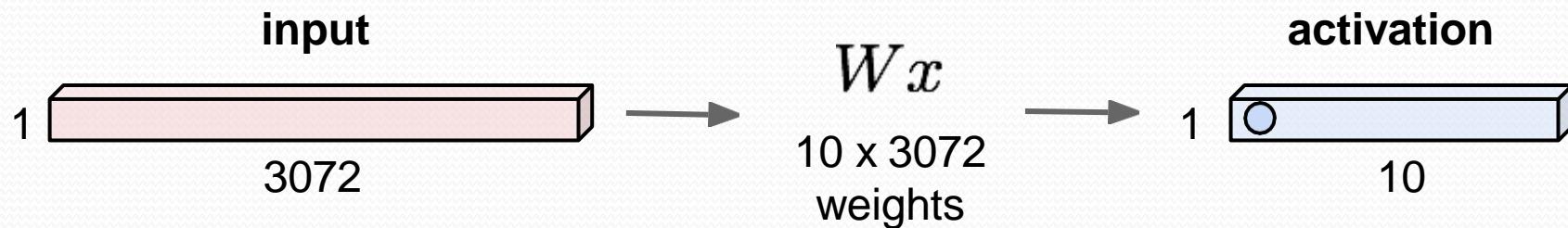
Art style transfer



Convolutional Neural Networks

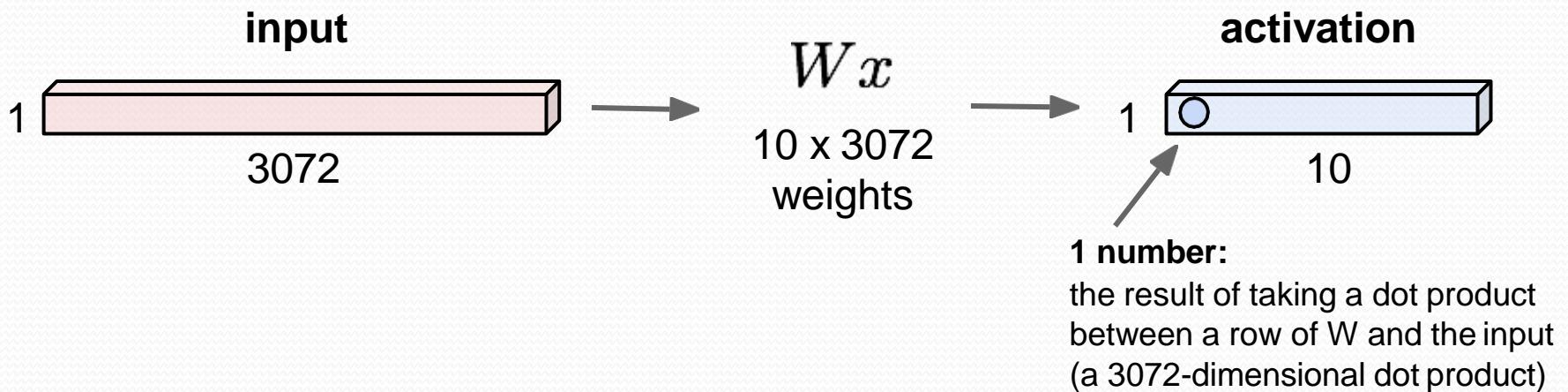
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



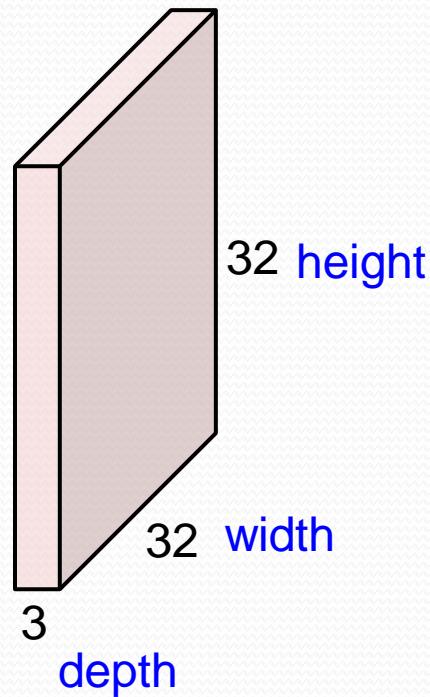
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



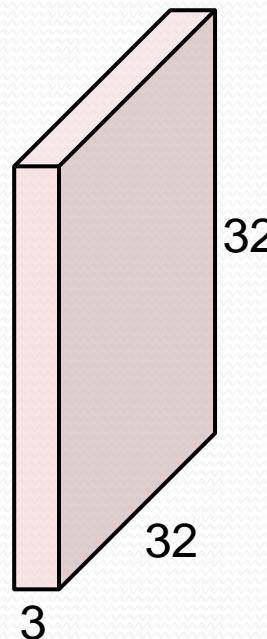
Convolution Layer

32x32x3 image -> preserve spatial structure

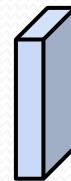


Convolution Layer

32x32x3 image

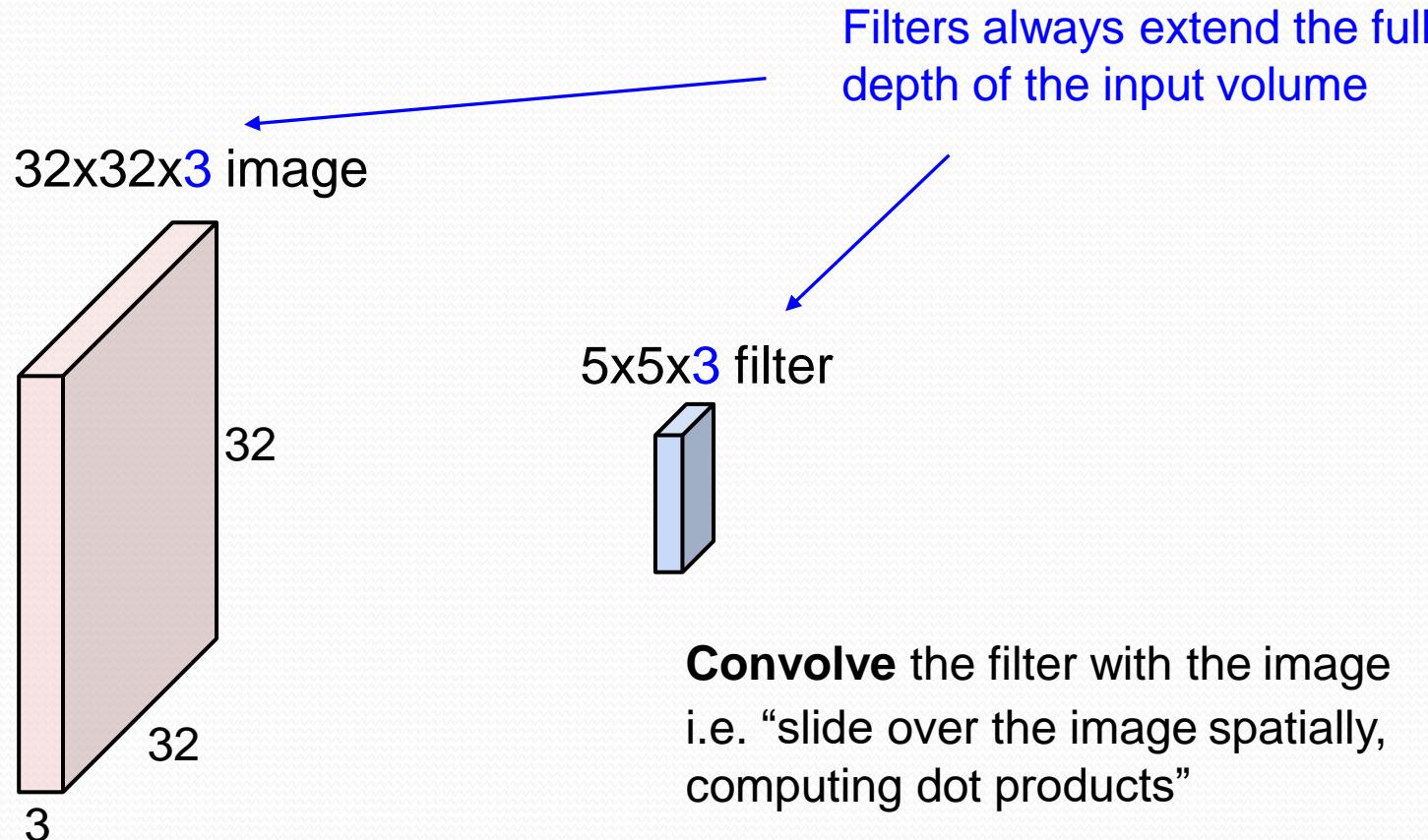


5x5x3 filter

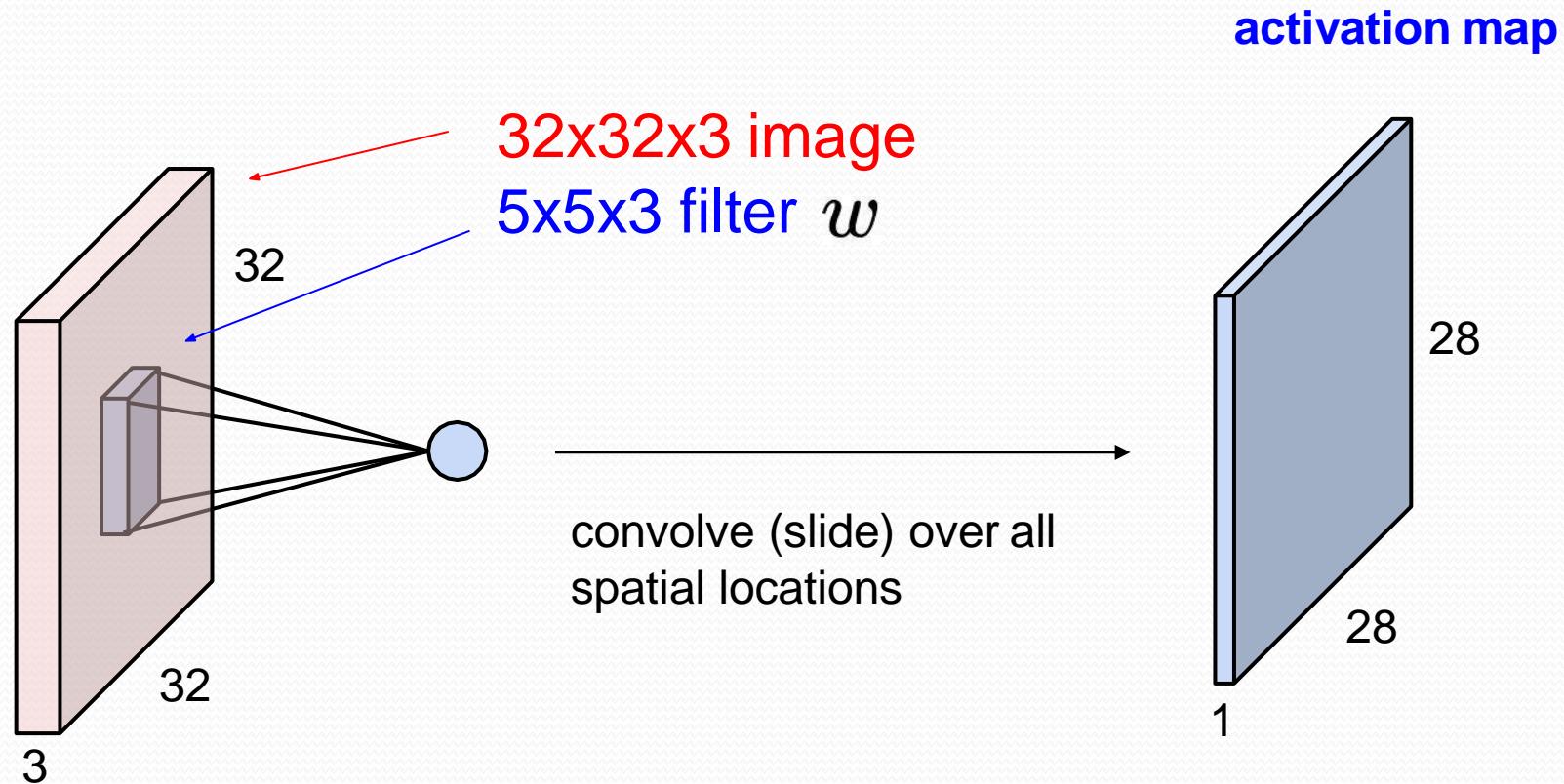


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolution Layer

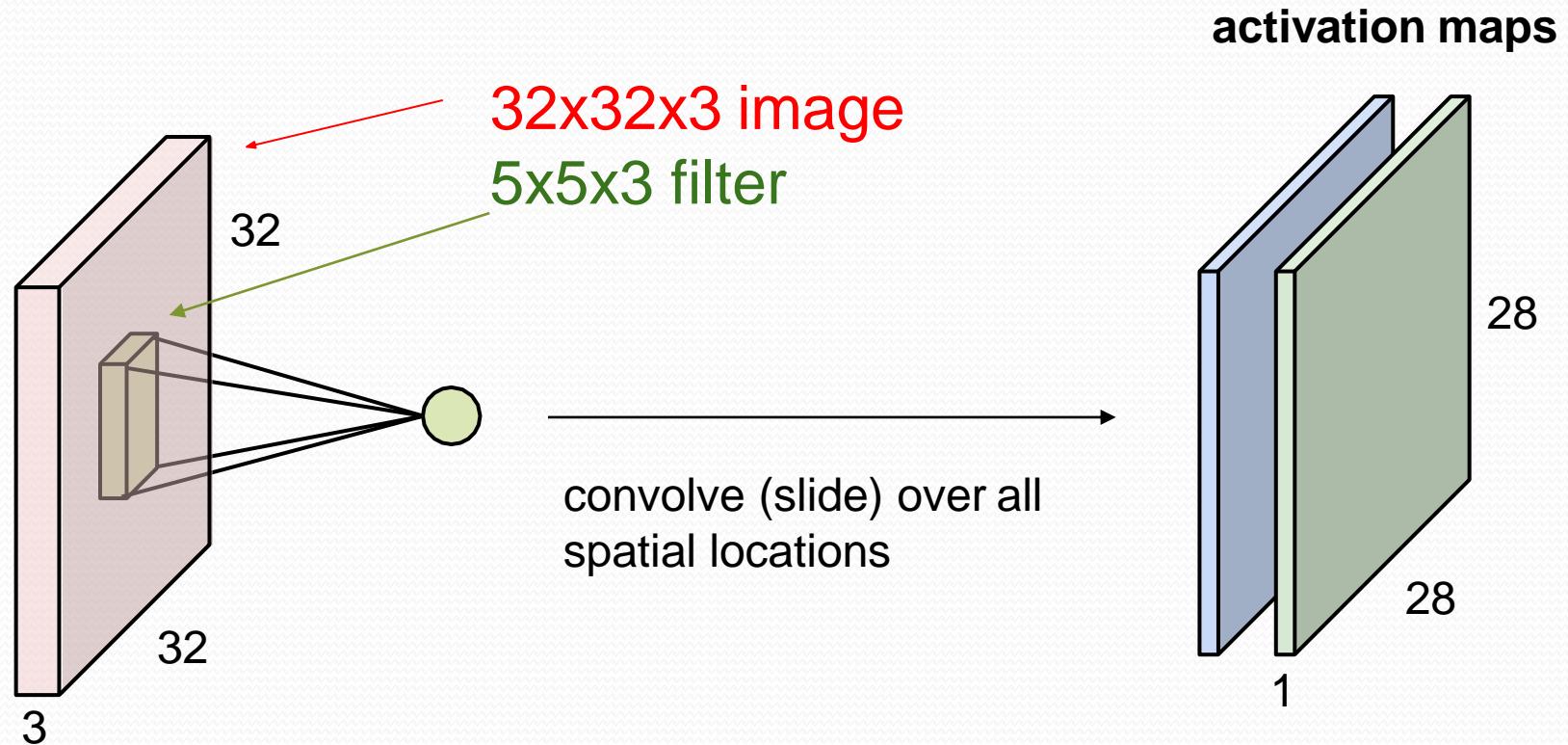


Convolution Layer

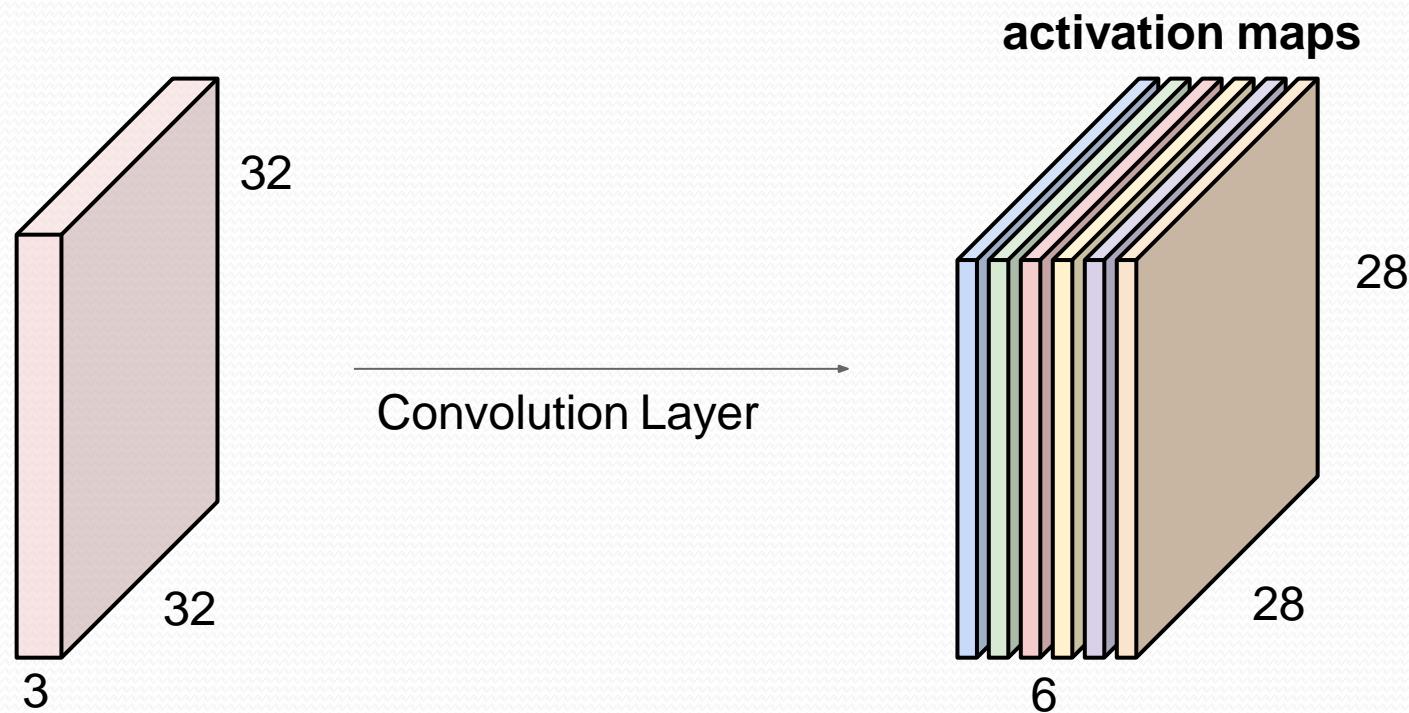


Convolution Layer

consider a second, green filter

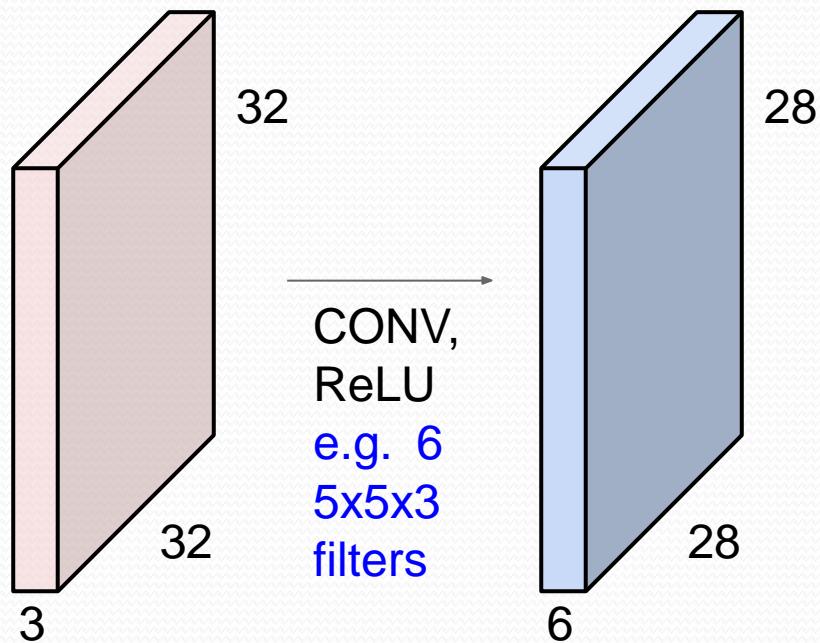


For example, if we had 6 $5 \times 5 \times 3$ filters, we'll get 6 separate activation maps:

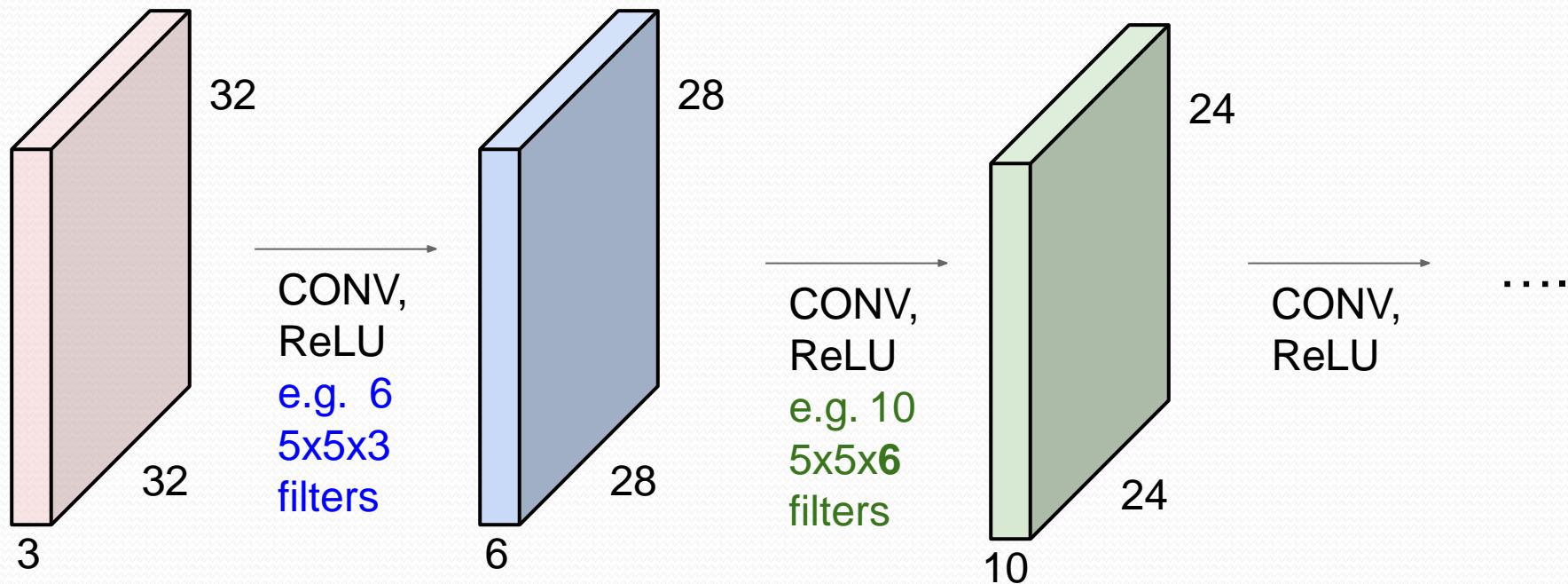


We stack these up to get a “new image” of size $28 \times 28 \times 6$!

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



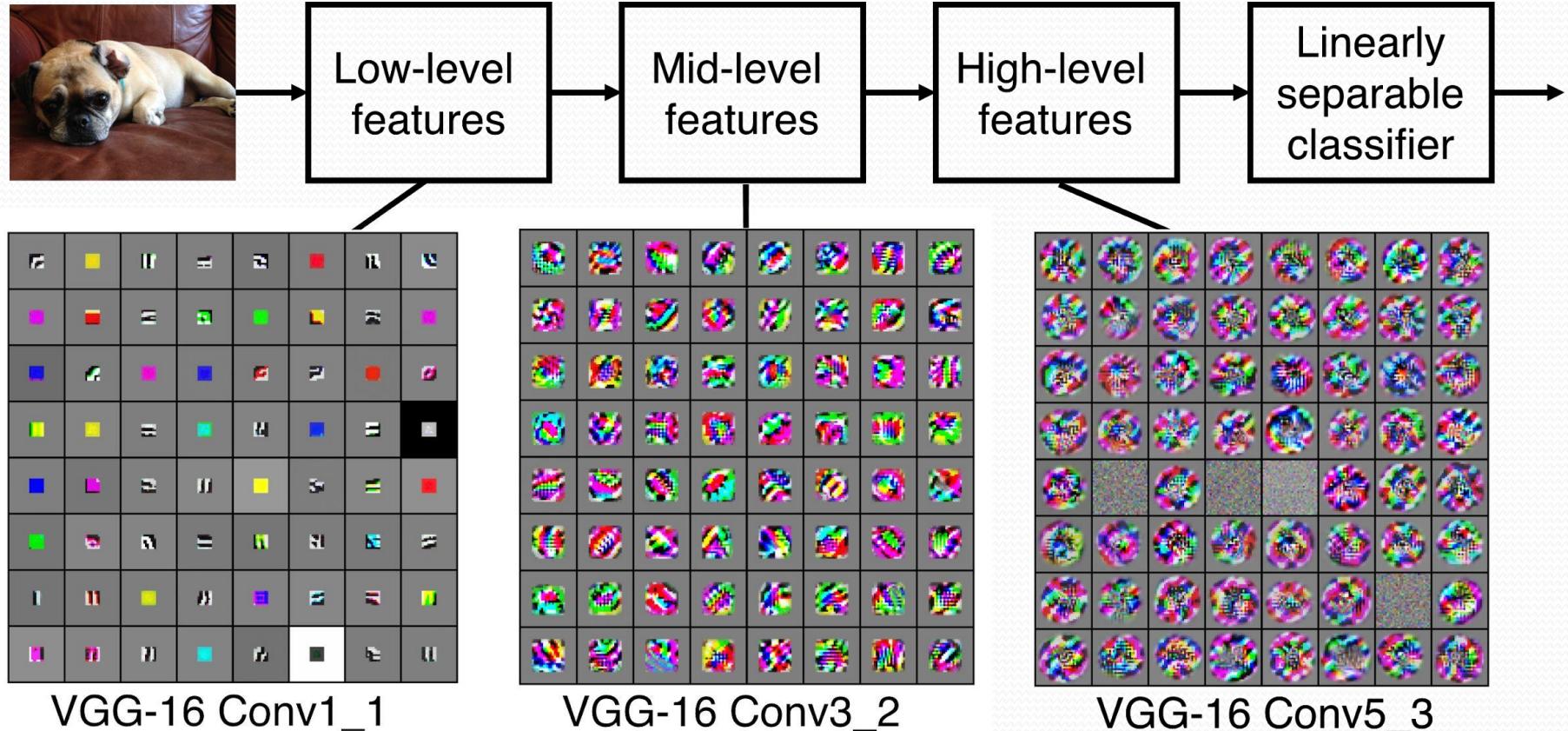
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Preview

[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].

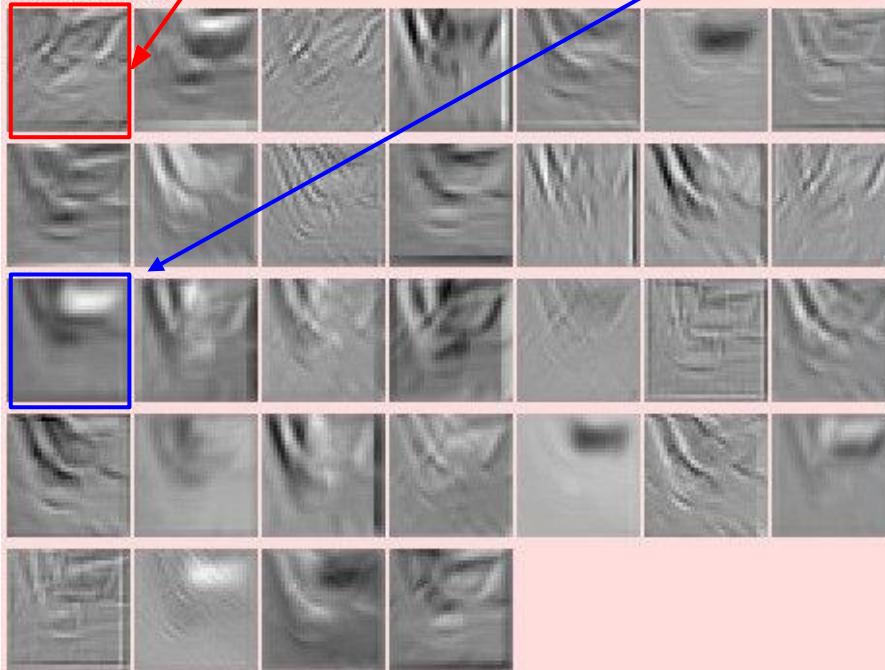


Activations:



one filter =>
one activation map

Activations:



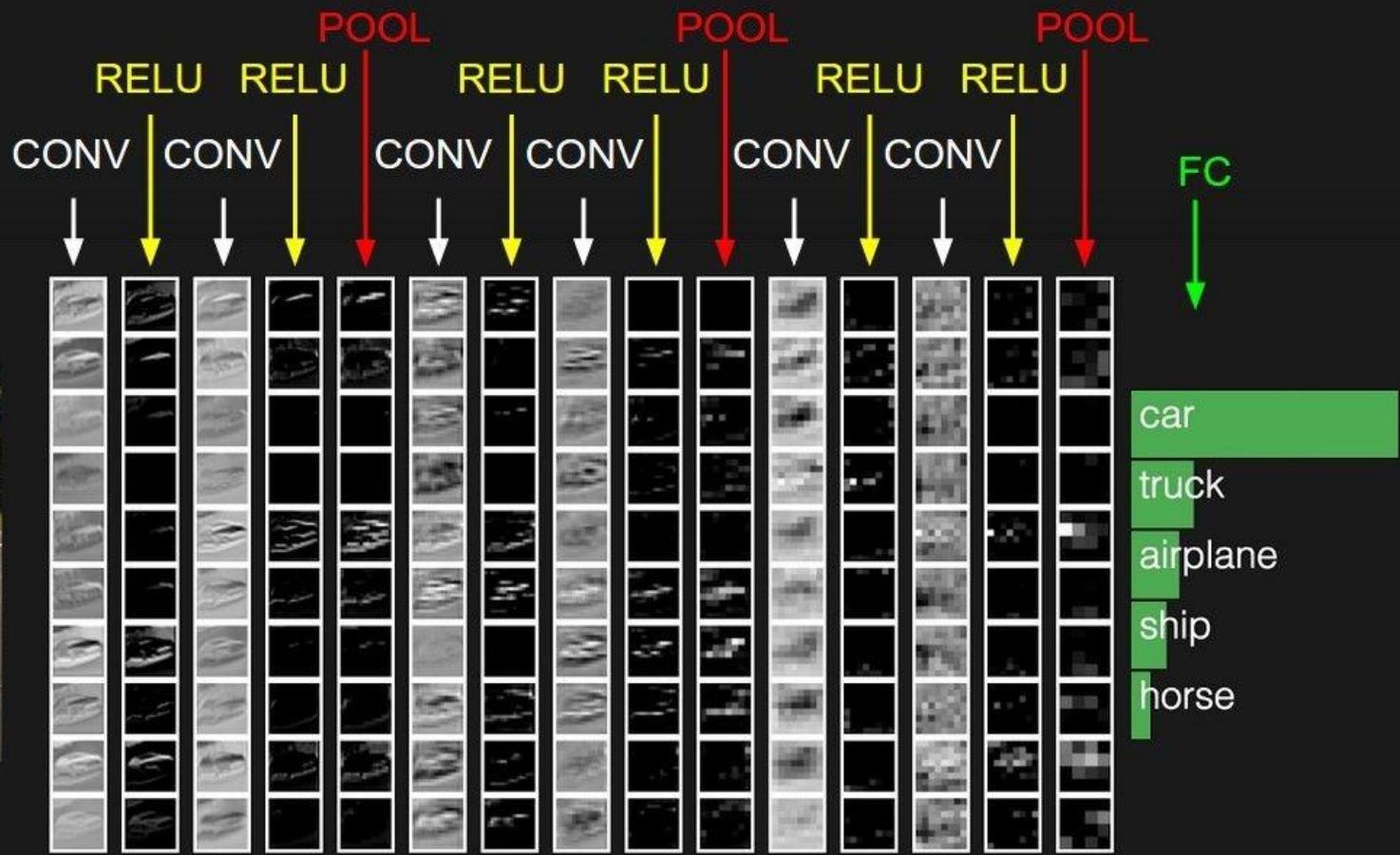
example 5x5 filters
(32 total)

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

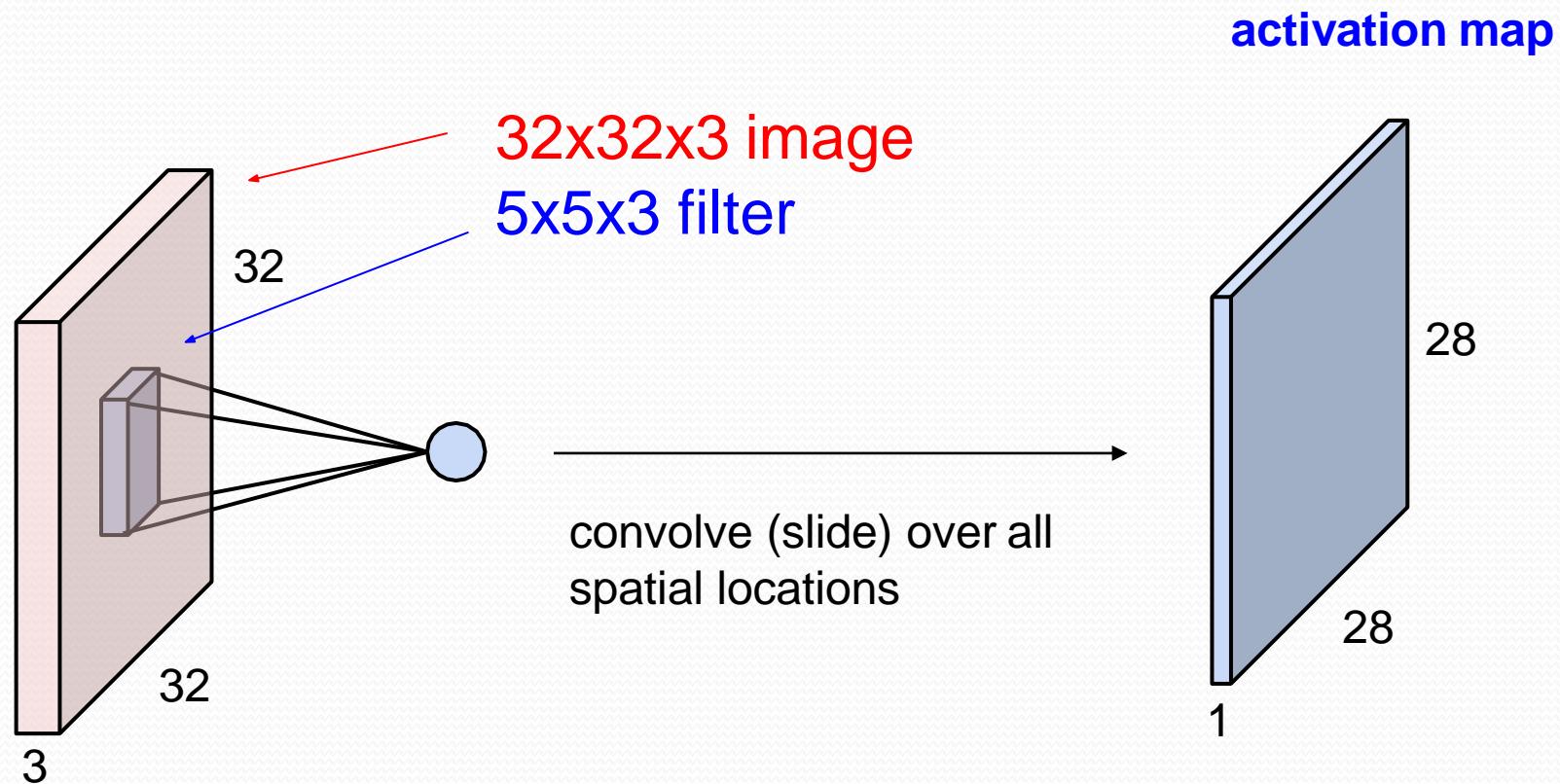
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2]$$

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

Preview



A closer look at spatial dimensions:



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

7

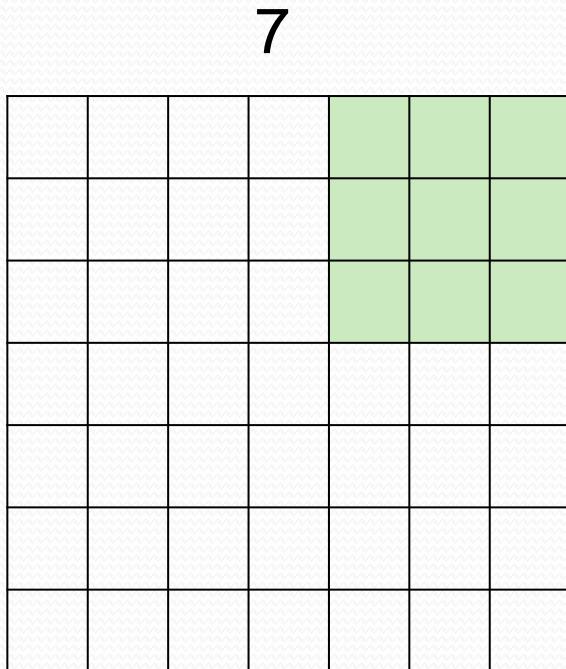
A closer look at spatial dimensions:

7

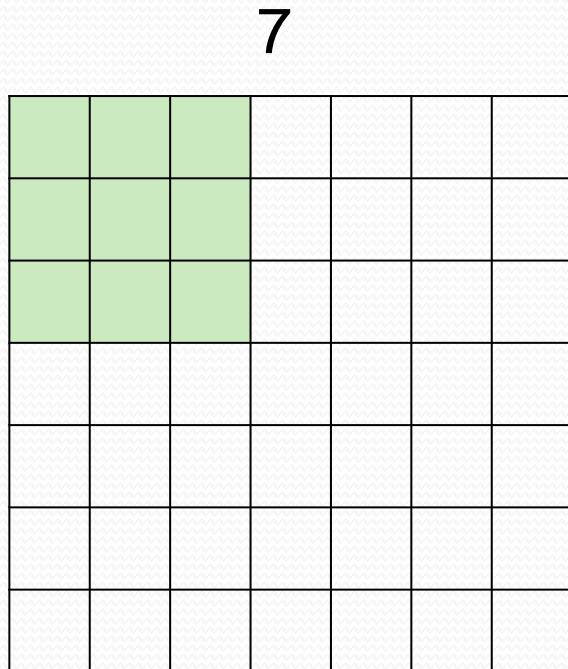
7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:



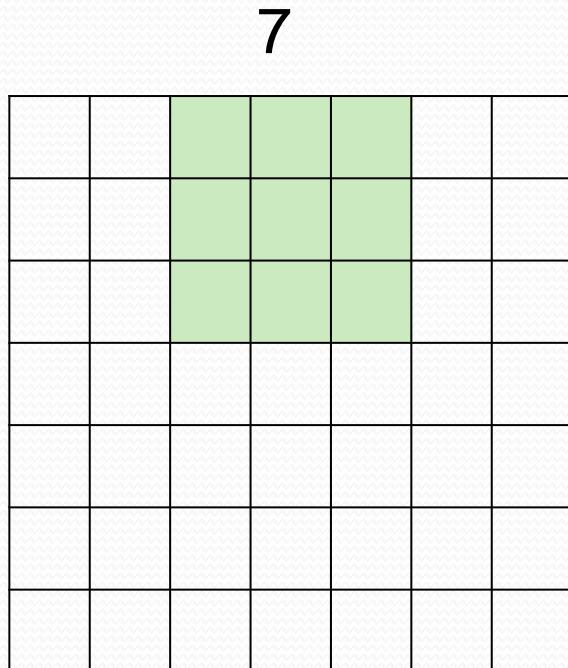
A closer look at spatial dimensions:



7

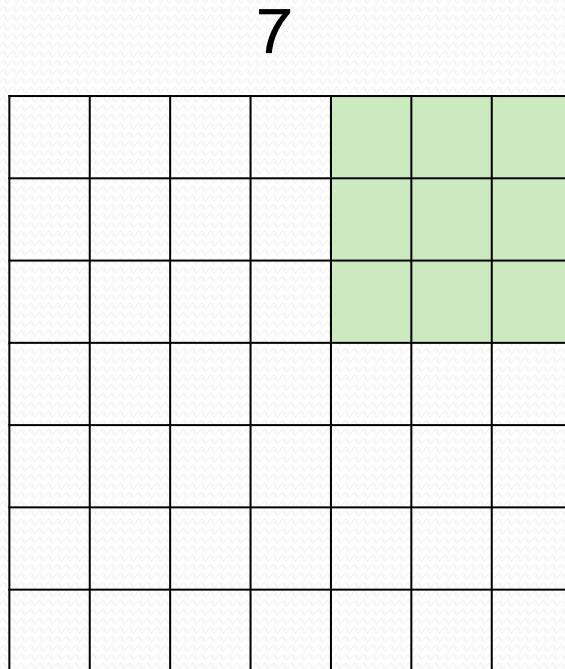
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



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

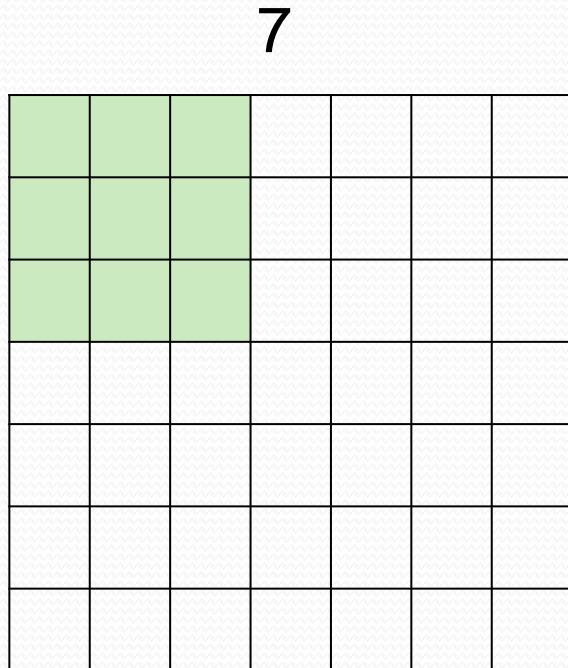
A closer look at spatial dimensions:



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

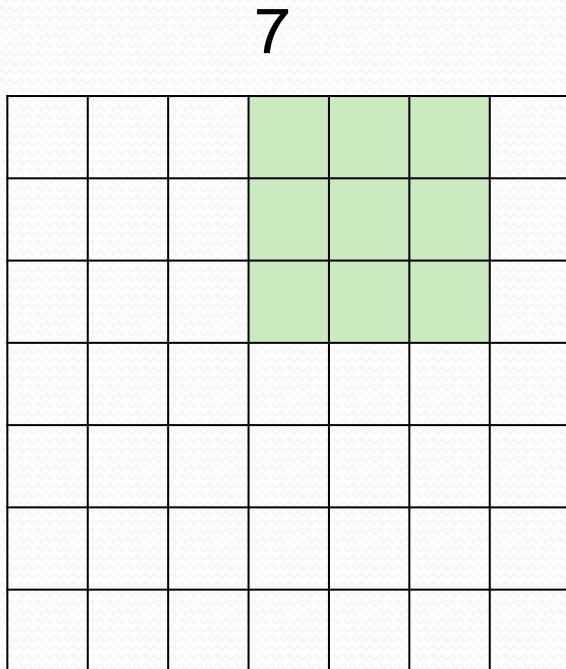
=> **3x3 output!**

A closer look at spatial dimensions:



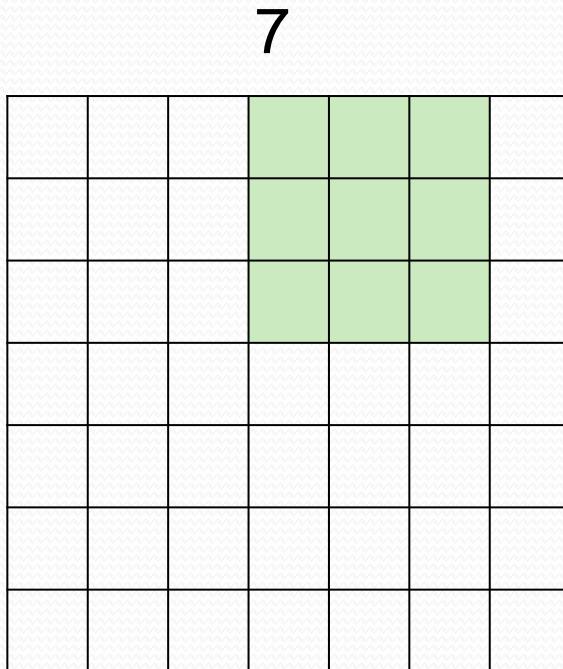
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



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

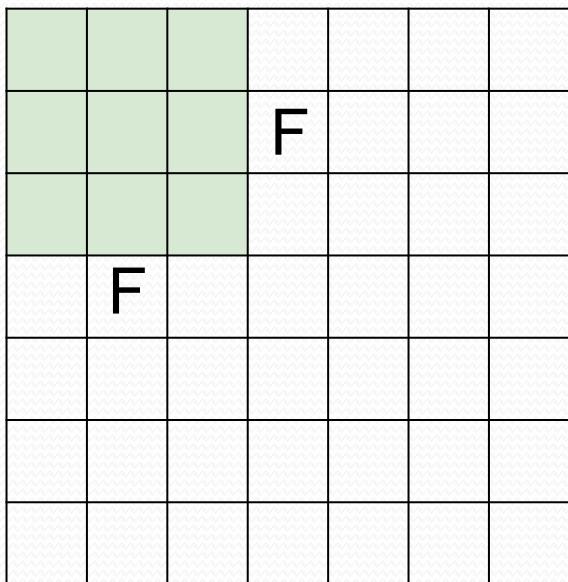
A closer look at spatial dimensions:



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 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)

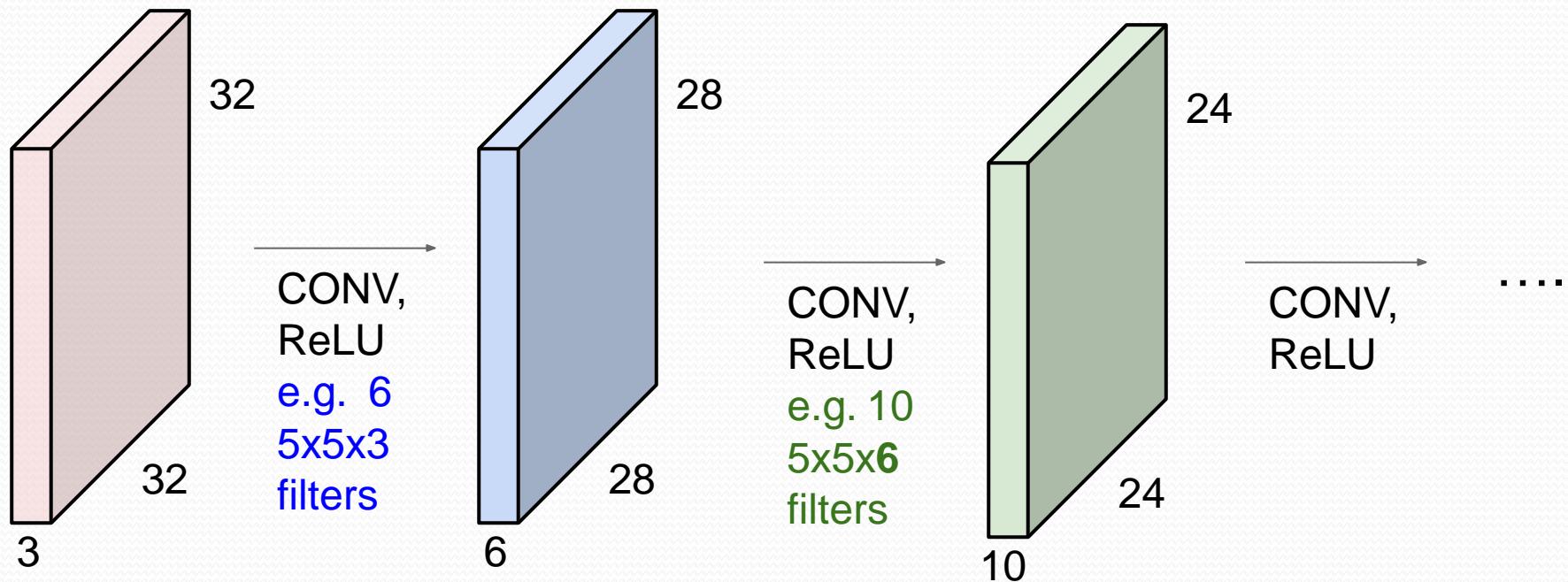
e.g. $F = 3 \Rightarrow$ zero pad with 1

$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

Remember back to...

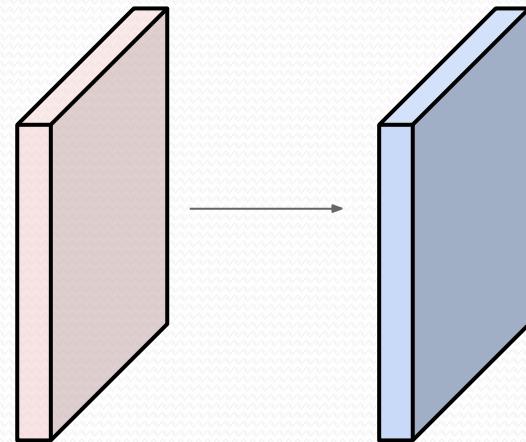
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



Examples time:

Input volume: **32x32x3**

10 5x5x3 filters with stride 1, pad 2

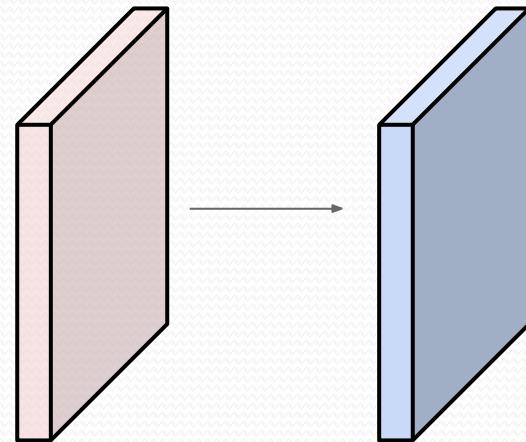


Output volume size?

Examples time:

Input volume: **32x32x3**

10 **5x5x3** filters with stride 1, pad 2



Output volume size:

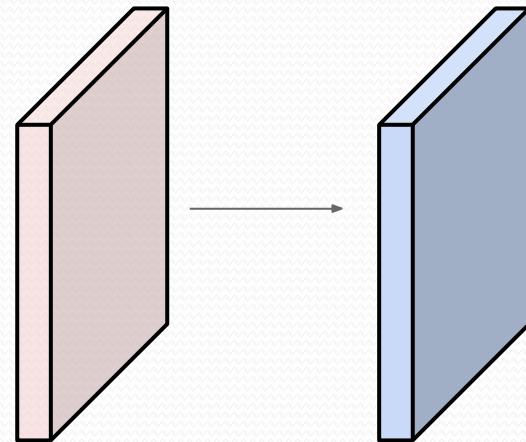
$(32+2*2-5)/1+1 = 32$ spatially, so

32x32x10

Examples time:

Input volume: **32x32x3**

10 5x5x3 filters with stride 1, pad 2

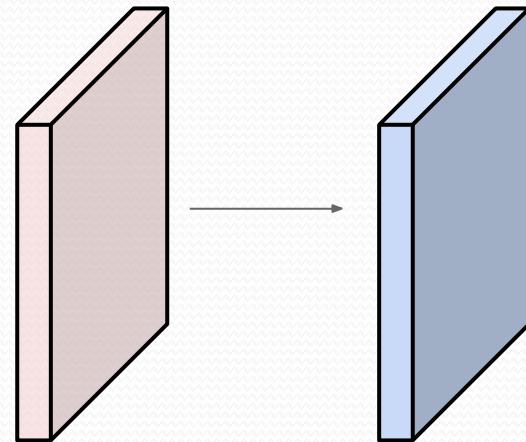


Number of parameters in this layer?

Examples time:

Input volume: **32x32x3**

10 5x5x3 filters with stride 1, pad 2



Number of parameters in this layer?

each filter has $5*5*3 + 1 = 76$ params

(+1 for bias)

$$\Rightarrow 76 * 10 = 760$$

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
 - Number of filters K ,
 - their spatial extent F ,
 - the stride S ,
 - the amount of zero padding P .
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F + 2P)/S + 1$
 - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
 - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d -th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d -th filter over the input volume with a stride of S , and then offset by d -th bias.

Summary. To summarize, the Conv Layer:

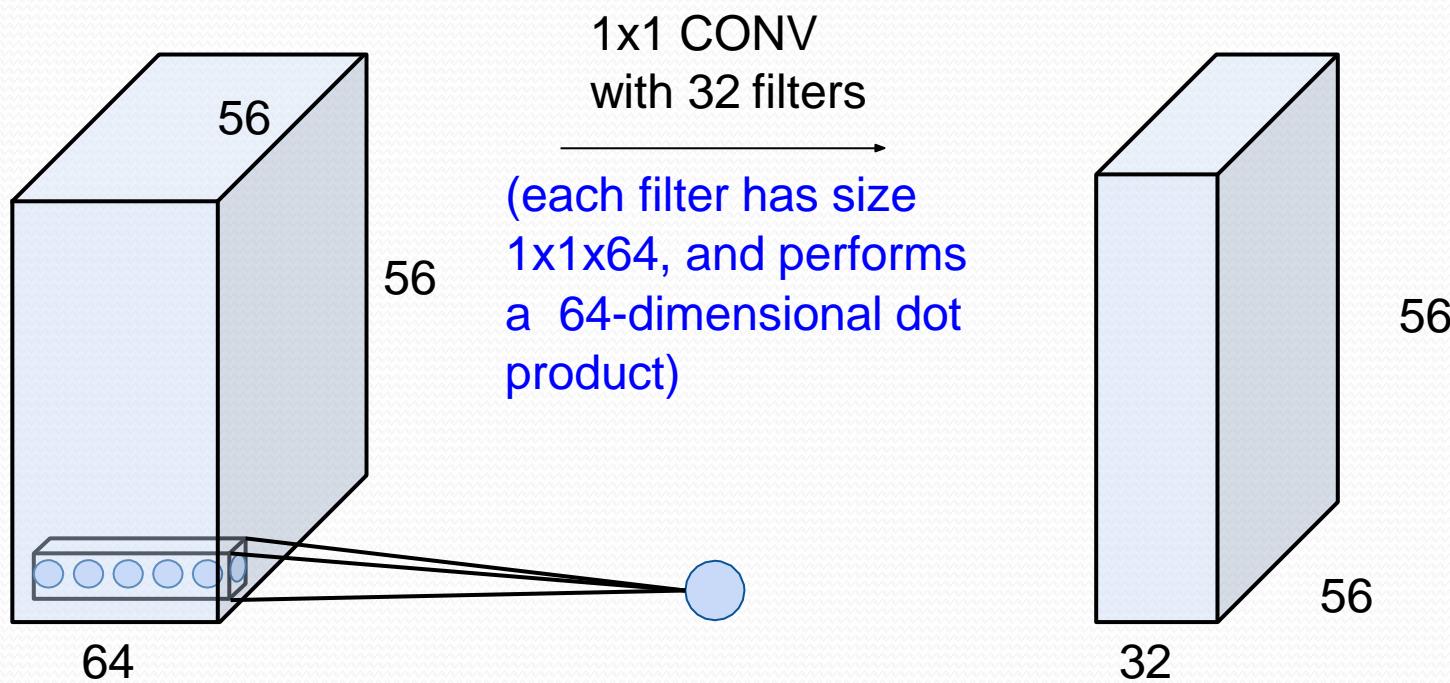
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
 - Number of filters K ,
 - their spatial extent F ,
 - the stride S ,
 - the amount of zero padding P .
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F + 2P)/S + 1$
 - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
 - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d -th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d -th filter over the input volume with a stride of S , and then offset by d -th bias.

Common settings:

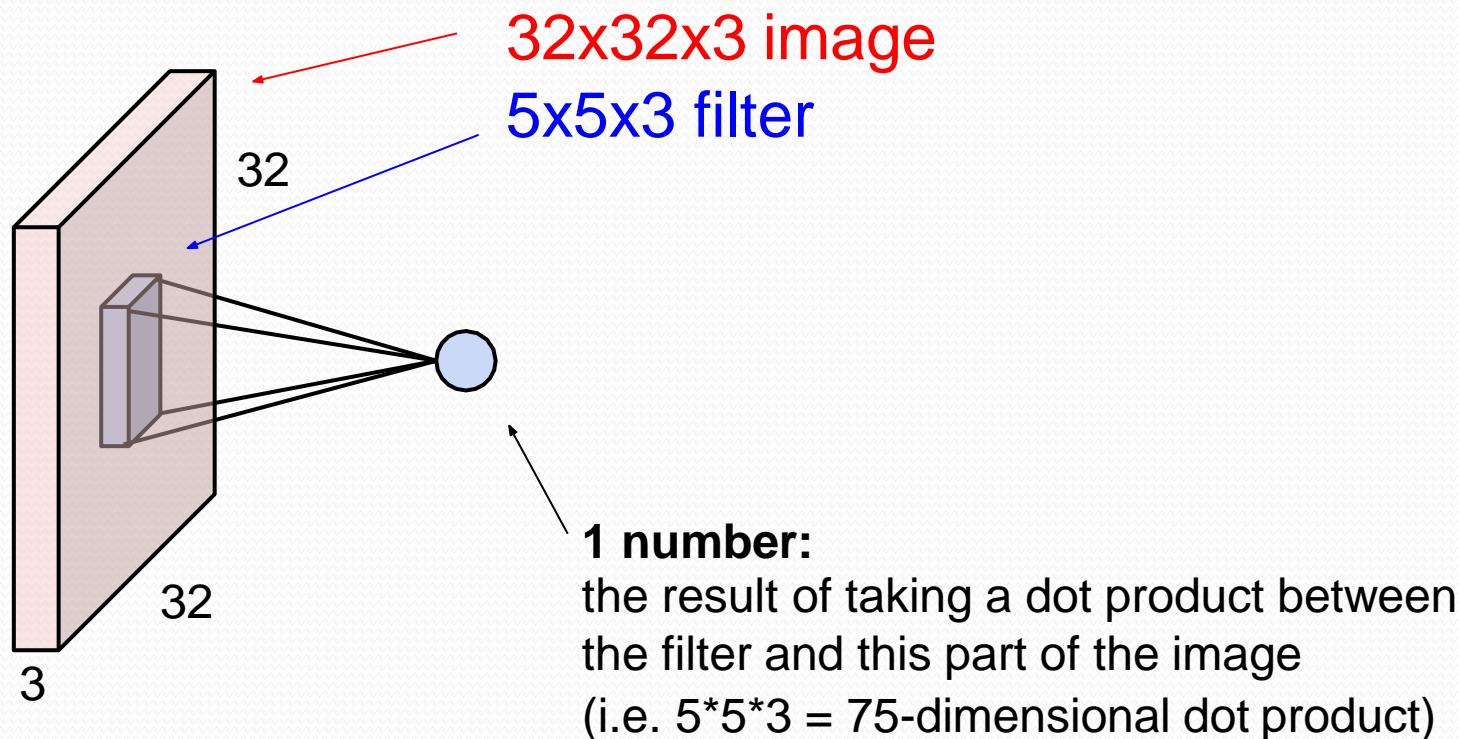
$K = (\text{powers of 2, e.g. } 32, 64, 128, 512)$

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

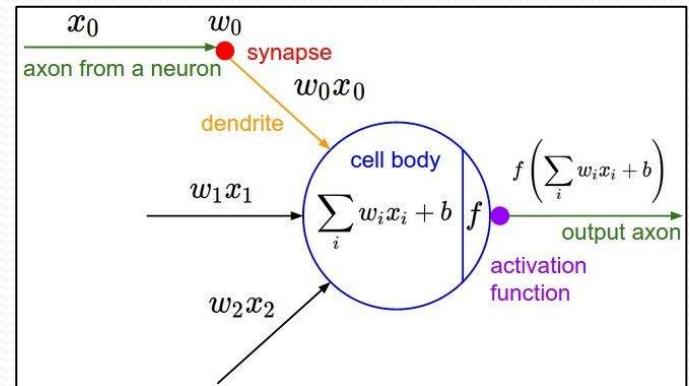
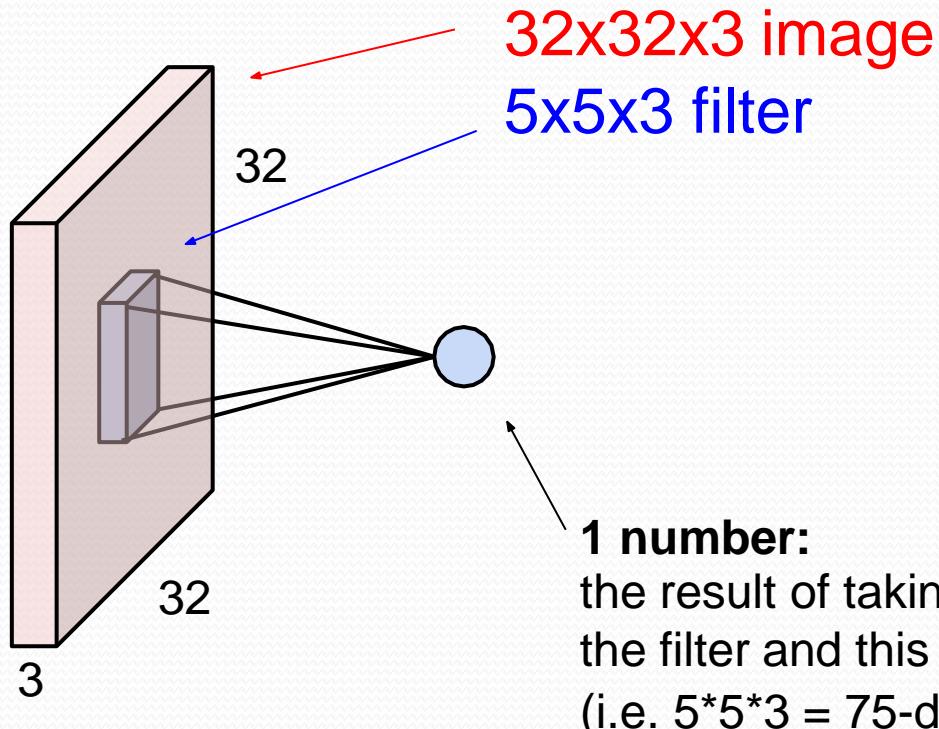
(btw, 1x1 convolution layers make perfect sense)



The brain/neuron view of CONV Layer

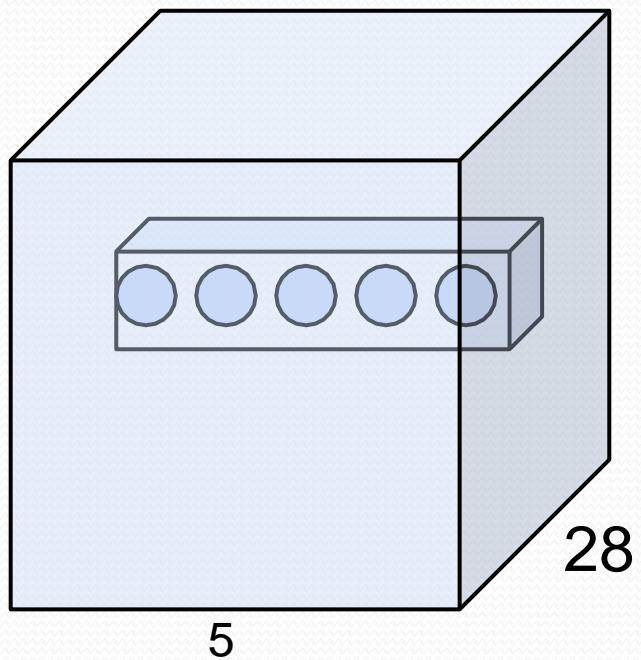
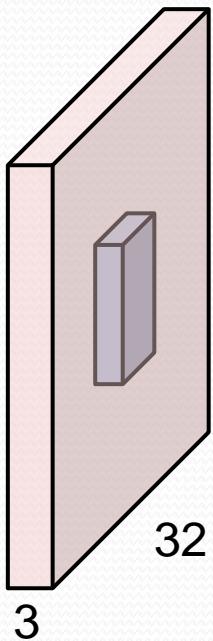


The brain/neuron view of CONV Layer



It's just a neuron with local connectivity...

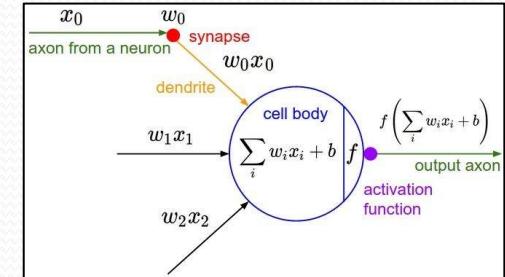
The brain/neuron view of CONV Layer



28

28

5



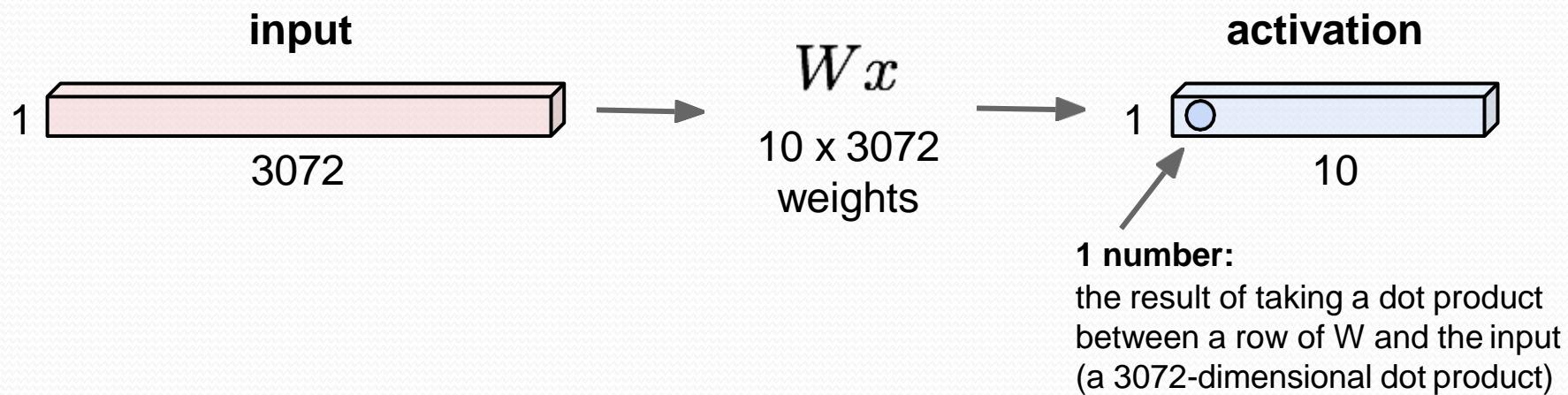
E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
($28 \times 28 \times 5$)

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

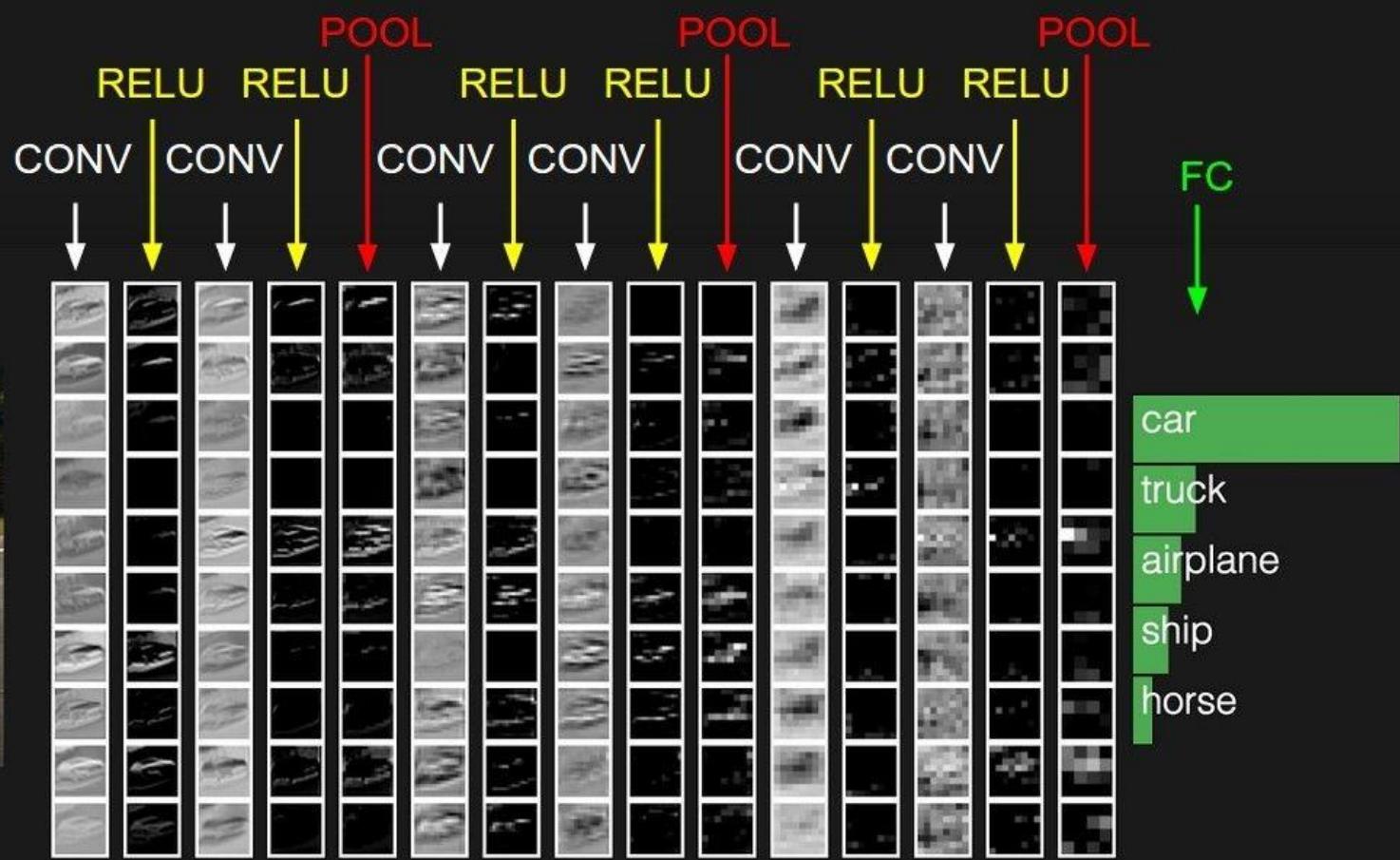
Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

Each neuron
looks at the full
input volume

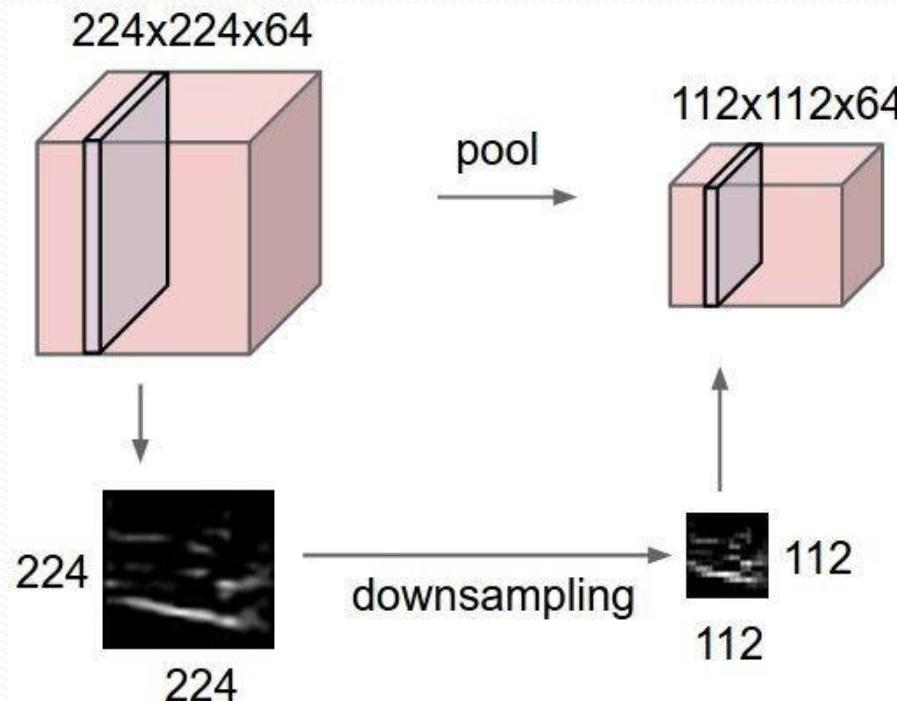


two more layers to go: POOL/FC



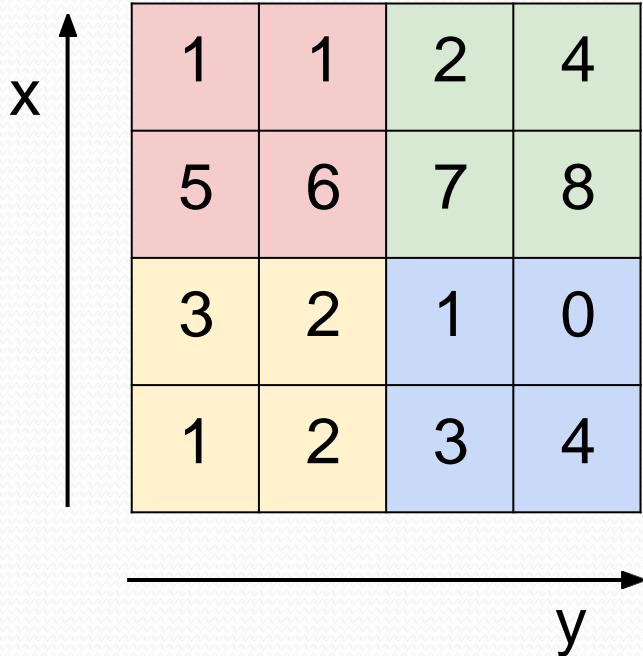
Pooling layer

- 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



6	8
3	4

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
 - their spatial extent F ,
 - the stride S ,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F)/S + 1$
 - $H_2 = (H_1 - F)/S + 1$
 - $D_2 = D_1$
- 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

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
 - their spatial extent F ,
 - the stride S ,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F)/S + 1$
 - $H_2 = (H_1 - F)/S + 1$
 - $D_2 = D_1$
- 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

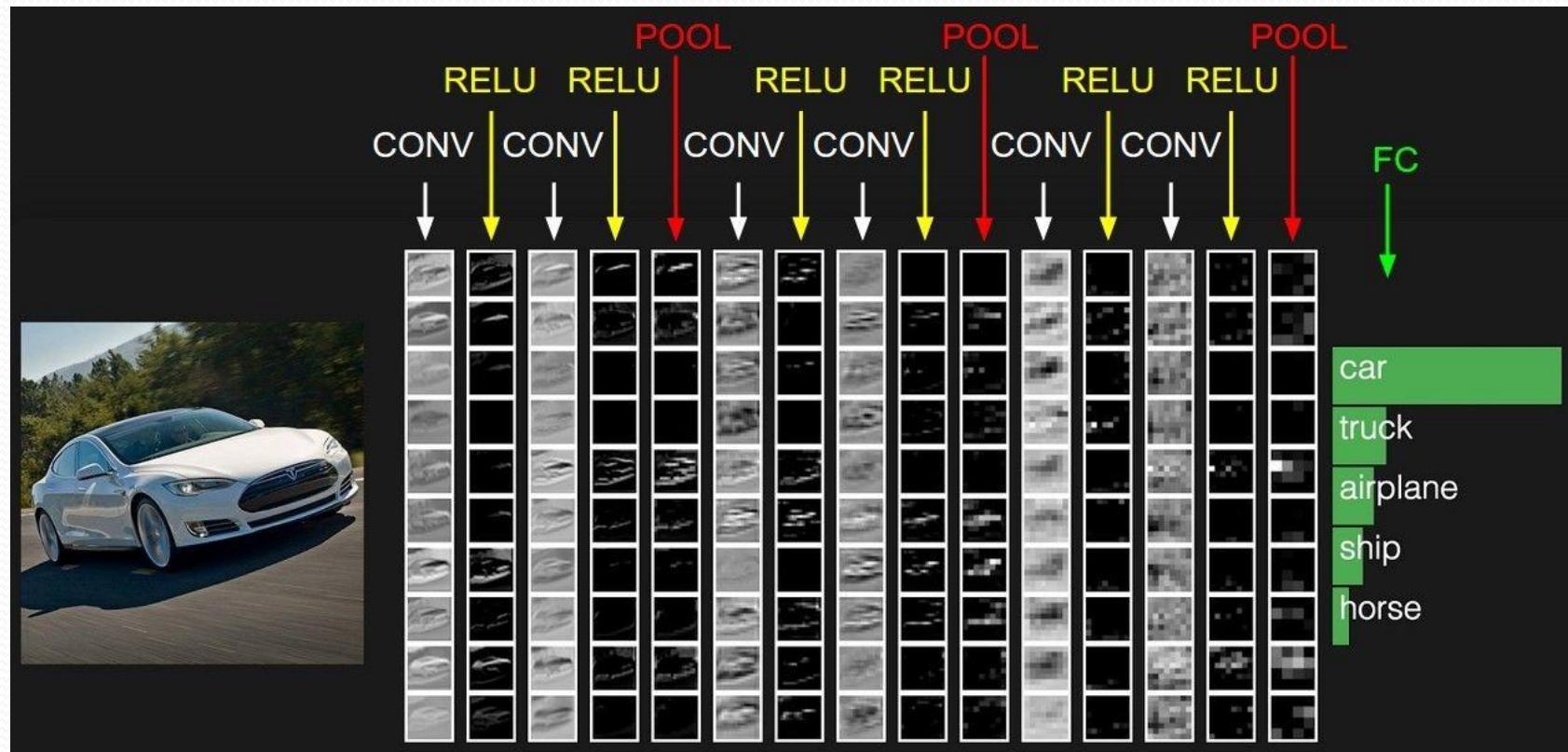
Common settings:

$F = 2, S = 2$

$F = 3, S = 2$

Fully Connected Layer (FC layer)

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



[ConvNetJS demo: training on CIFAR-10]

ConvNetJS CIFAR-10 demo

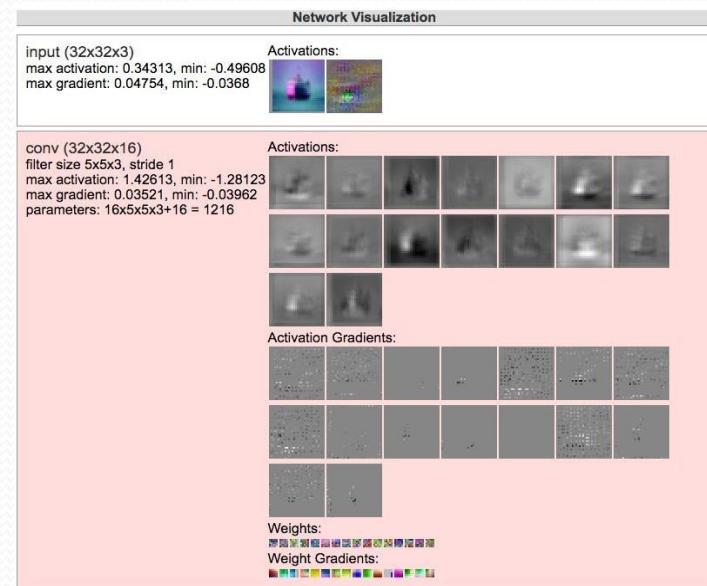
Description

This demo trains a Convolutional Neural Network on the [CIFAR-10 dataset](#) in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used [this python script](#) to parse the [original files](#) (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to [@karpathy](#).



<http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

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

$[(\text{CONV-RELU})^*N - \text{POOL?}]^*M - (\text{FC-RELU})^*K, \text{SOFTMAX}$

where N is usually up to ~5, M is large, $0 \leq K \leq 2$.

! but recent advances such as ResNet/GoogLeNet challenge this paradigm