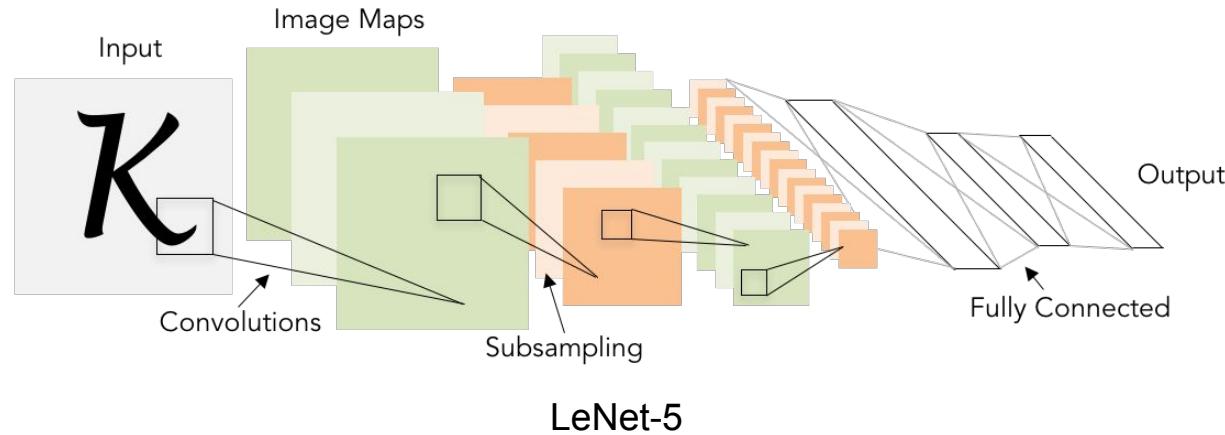


Lecture 5: Convolutional Neural Networks

A bit of history: Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]



A bit of history: ImageNet Classification with Deep Convolutional Neural Networks *[Krizhevsky, Sutskever, Hinton, 2012]*

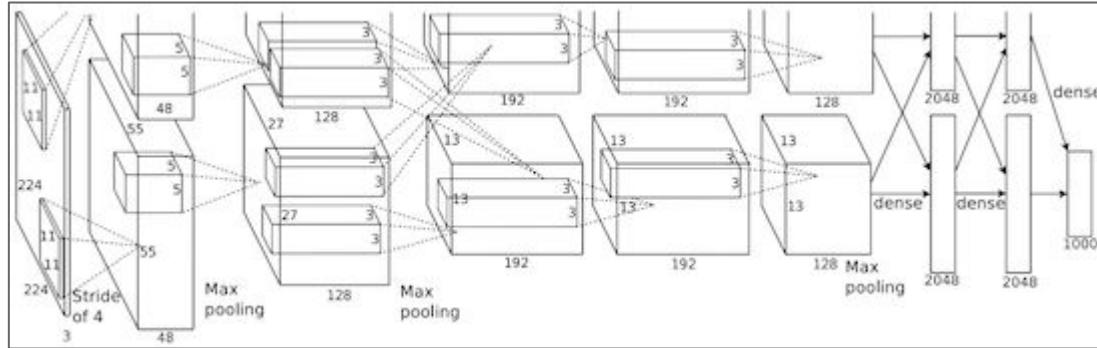


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

“AlexNet”

Fast-forward to today: ConvNets are everywhere

Classification



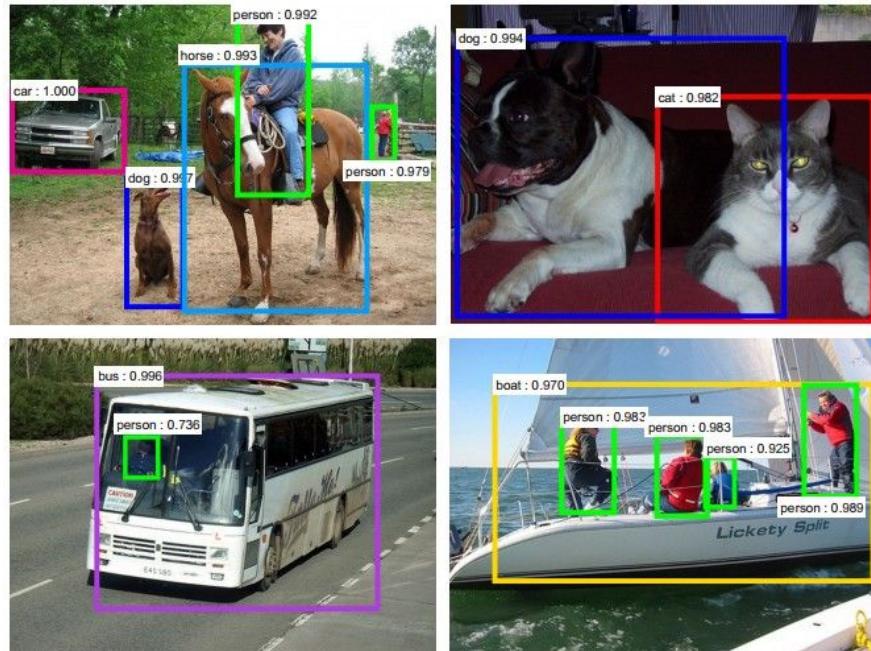
Retrieval



Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Fast-forward to today: ConvNets are everywhere

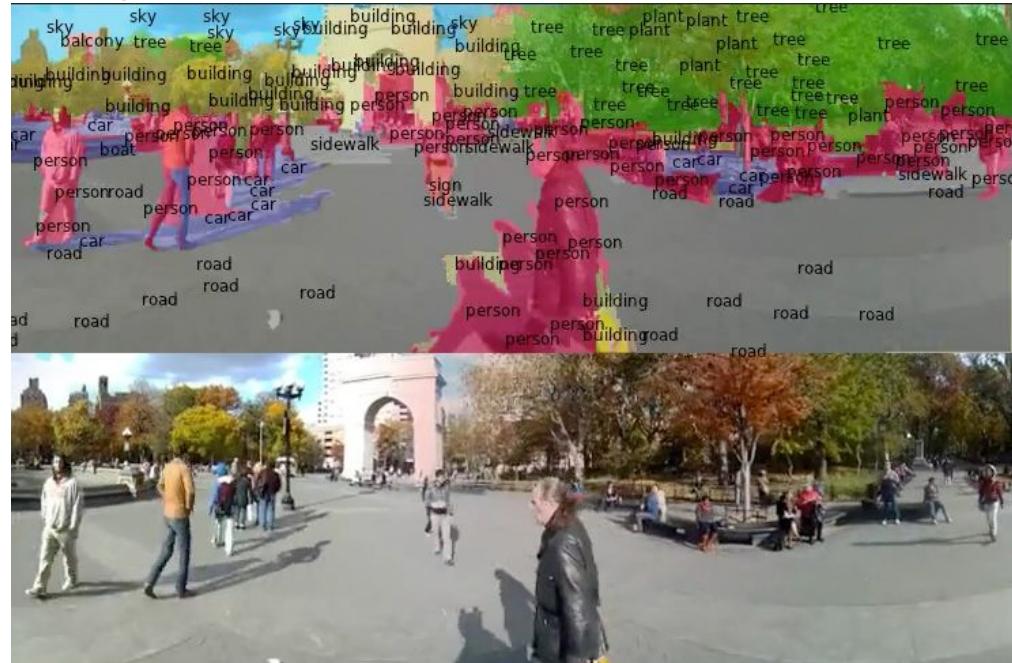
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, 2015. Reproduced with permission.

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

Segmentation



Figures copyright Clement Farabet, 2012.
Reproduced with permission.

[*Farabet et al., 2012*]

Fast-forward to today: ConvNets are everywhere



self-driving cars

Photo by Lane McIntosh. Copyright CS231n 2017.



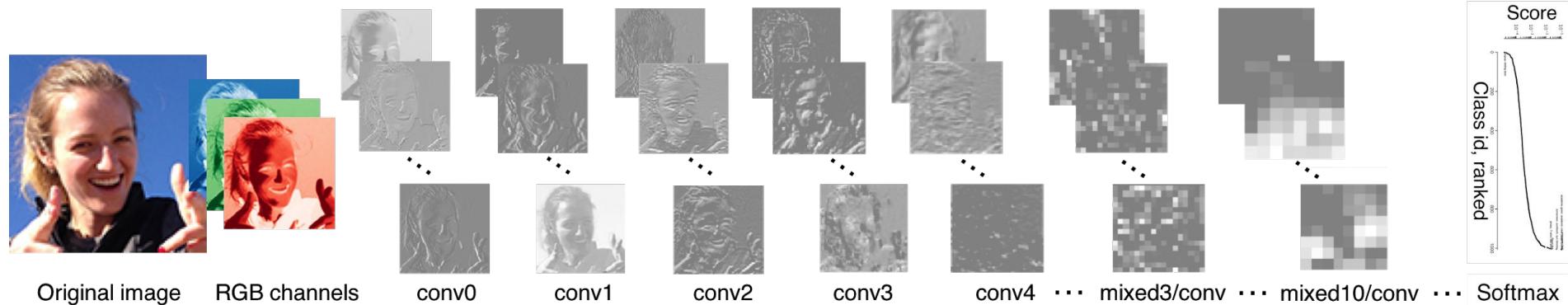
[This image](#) by GBPublic_PR is licensed under [CC-BY 2.0](#)

NVIDIA Tesla line

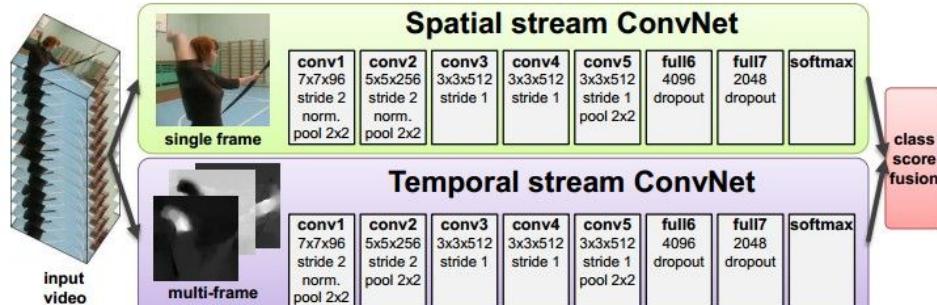
(these are the GPUs on rye01.stanford.edu)

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



Activations of [inception-v3 architecture](#) [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.



[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014.
Reproduced with permission.

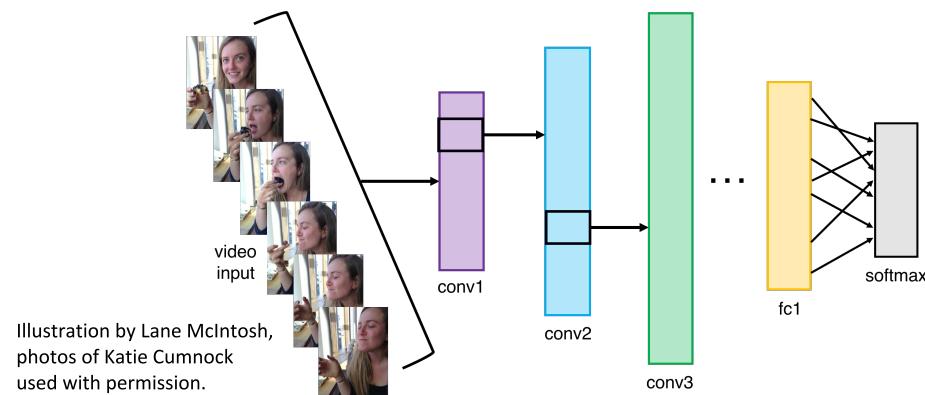


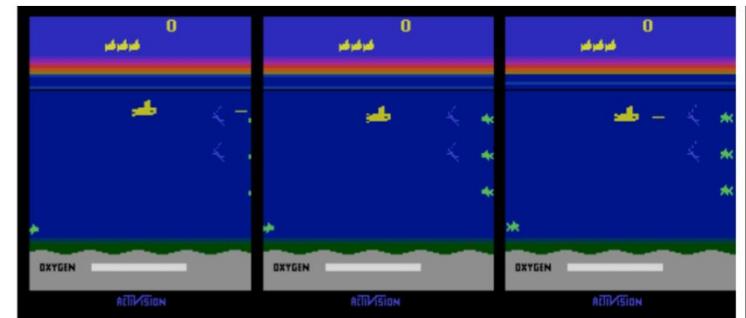
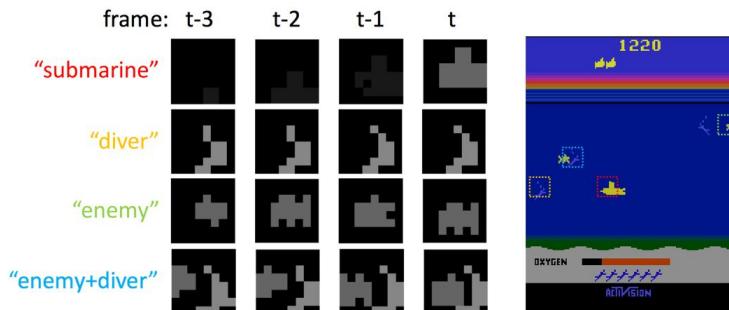
Illustration by Lane McIntosh,
photos of Katie Cumnock
used with permission.

Fast-forward to today: ConvNets are everywhere



Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

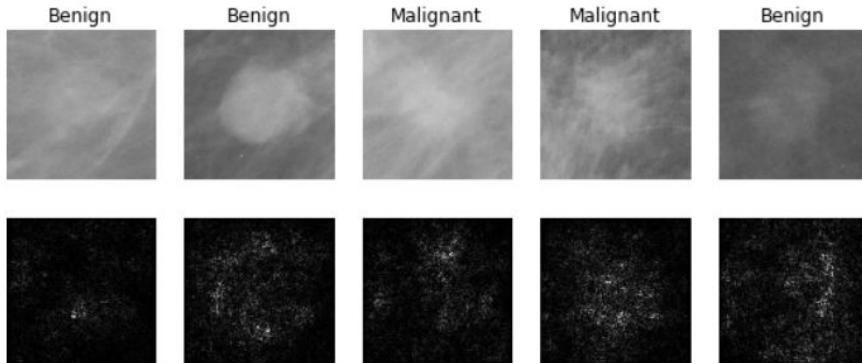
[Toshev, Szegedy 2014]



[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

Fast-forward to today: ConvNets are everywhere



[Levy et al. 2016]

Figure copyright Levy et al. 2016.
Reproduced with permission.



[Dieleman et al. 2014]

From left to right: [public domain by NASA](#), usage [permitted](#) by
ESA/Hubble, [public domain by NASA](#), and [public domain](#).



Photos by Lane McIntosh.
Copyright CS231n 2017.

[Sermanet et al. 2011]
[Ciresan et al.]

[This image](#) by Christin Khan is in the public domain and originally came from the U.S. NOAA.



Whale recognition, Kaggle Challenge

Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.



Mnih and Hinton, 2010

No errors



A white teddy bear sitting in the grass



A man riding a wave on top of a surfboard

Minor errors



A man in a baseball uniform throwing a ball



A cat sitting on a suitcase on the floor

Somewhat related



A woman is holding a cat in her hand



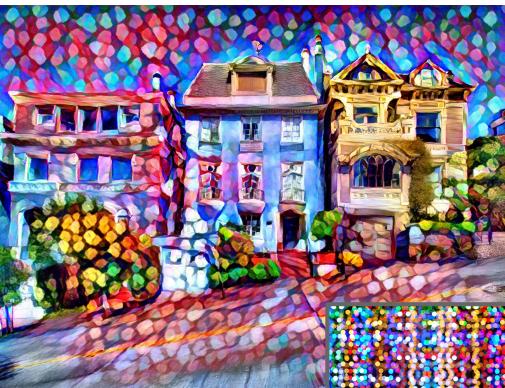
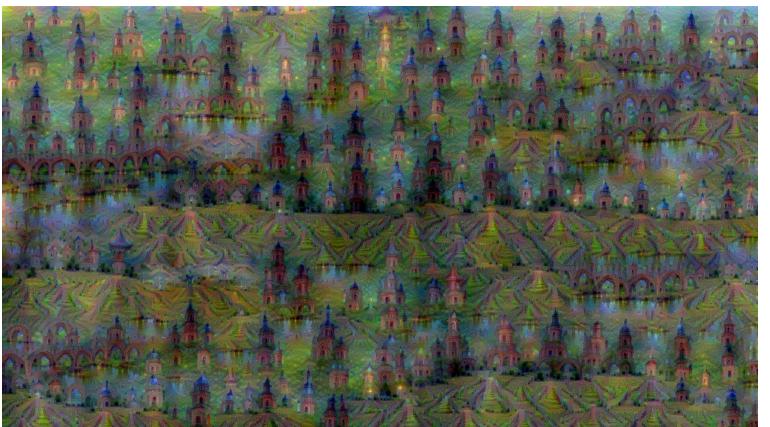
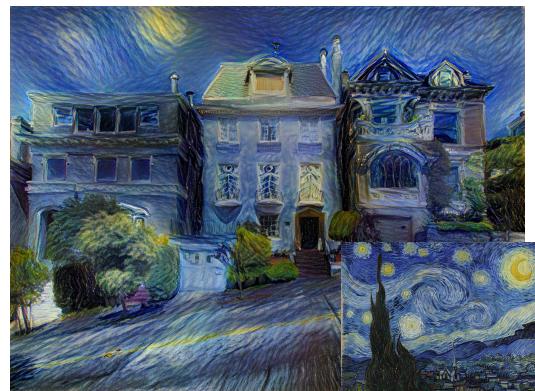
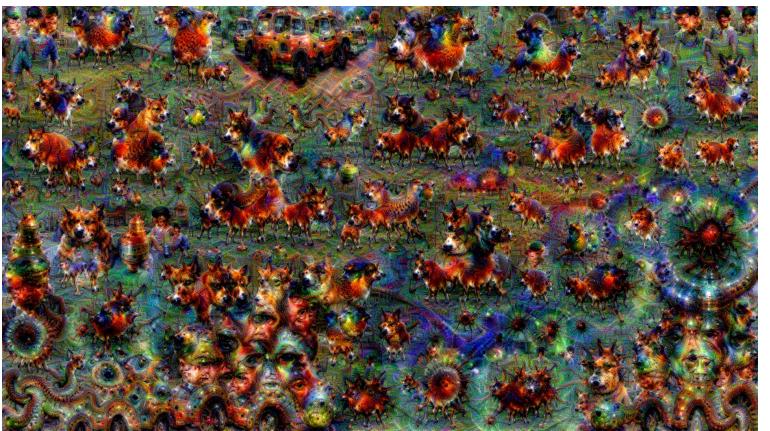
A woman standing on a beach holding a surfboard

Image Captioning

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

All images are CC0 Public domain:
<https://pixabay.com/en/luggage-antique-cat-1643010/>
<https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/>
<https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/>
<https://pixabay.com/en/woman-female-model-portrait-adult-983967/>
<https://pixabay.com/en/handstand-lake-meditation-496008/>
<https://pixabay.com/en/baseball-player-shortstop-infield-1045263/>

Captions generated by Justin Johnson using [Neuraltalk2](#)



Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a [blog post](#) by Google Research.

[Original image](#) is CC0 public domain

[Starry Night](#) and [Tree Roots](#) by Van Gogh are in the public domain

[Bokeh image](#) is in the public domain

Stylized images copyright Justin Johnson, 2017;
reproduced with permission

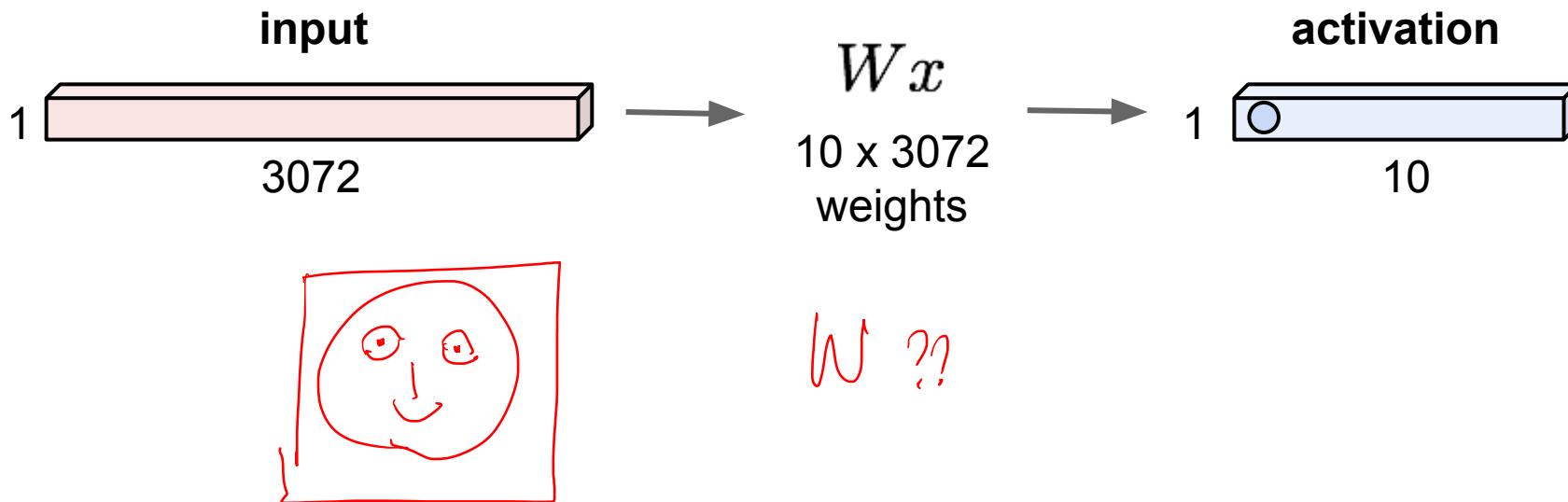
Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

Convolutional Neural Networks

(First without the brain stuff)

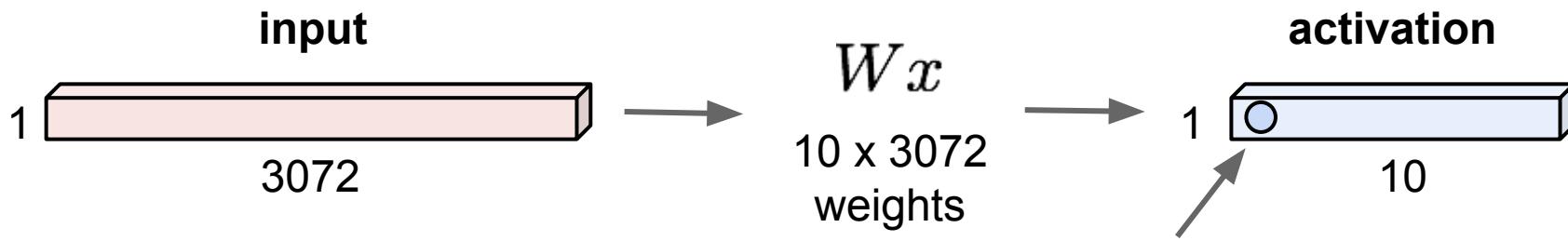
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

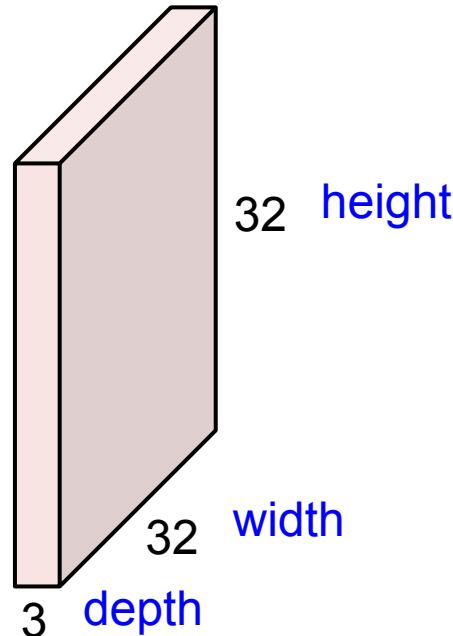


1 number:

the result of taking a dot product
between a row of W and the input
(a 3072-dimensional dot product)

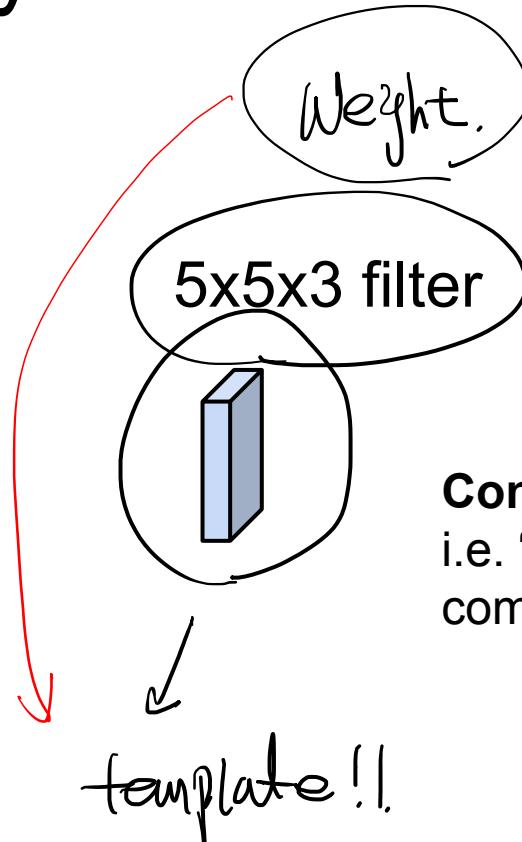
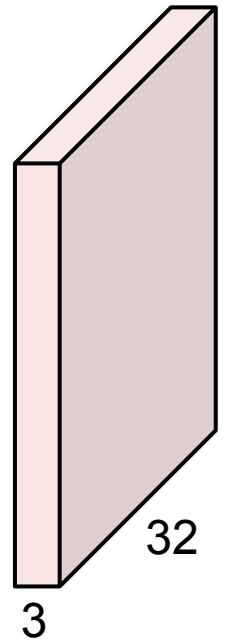
Convolution Layer

32x32x3 image -> preserve spatial structure



Convolution Layer

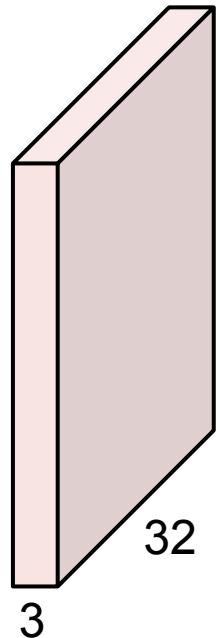
32x32x3 image



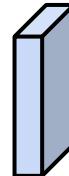
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolution Layer

32x32x3 image



5x5x3 filter

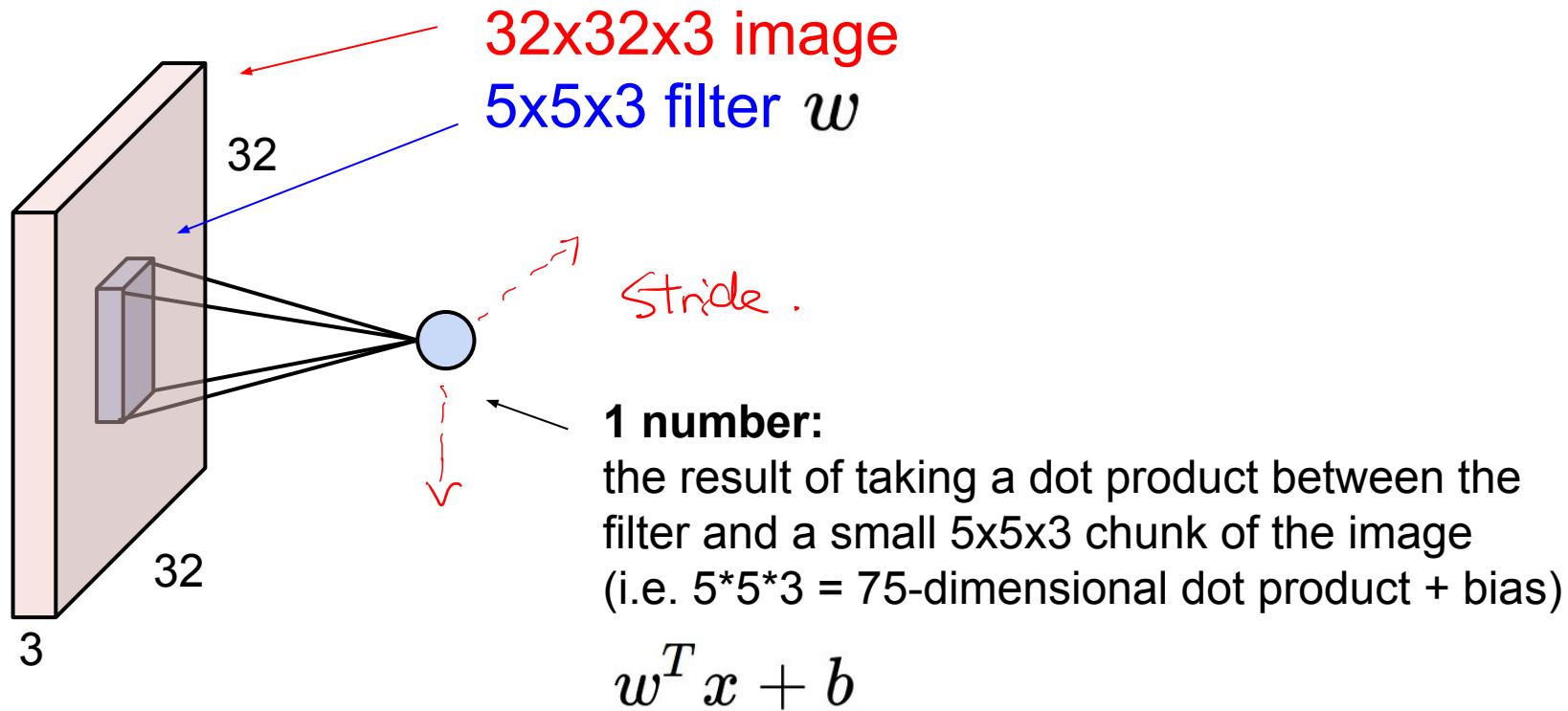


Filters always extend the full depth of the input volume

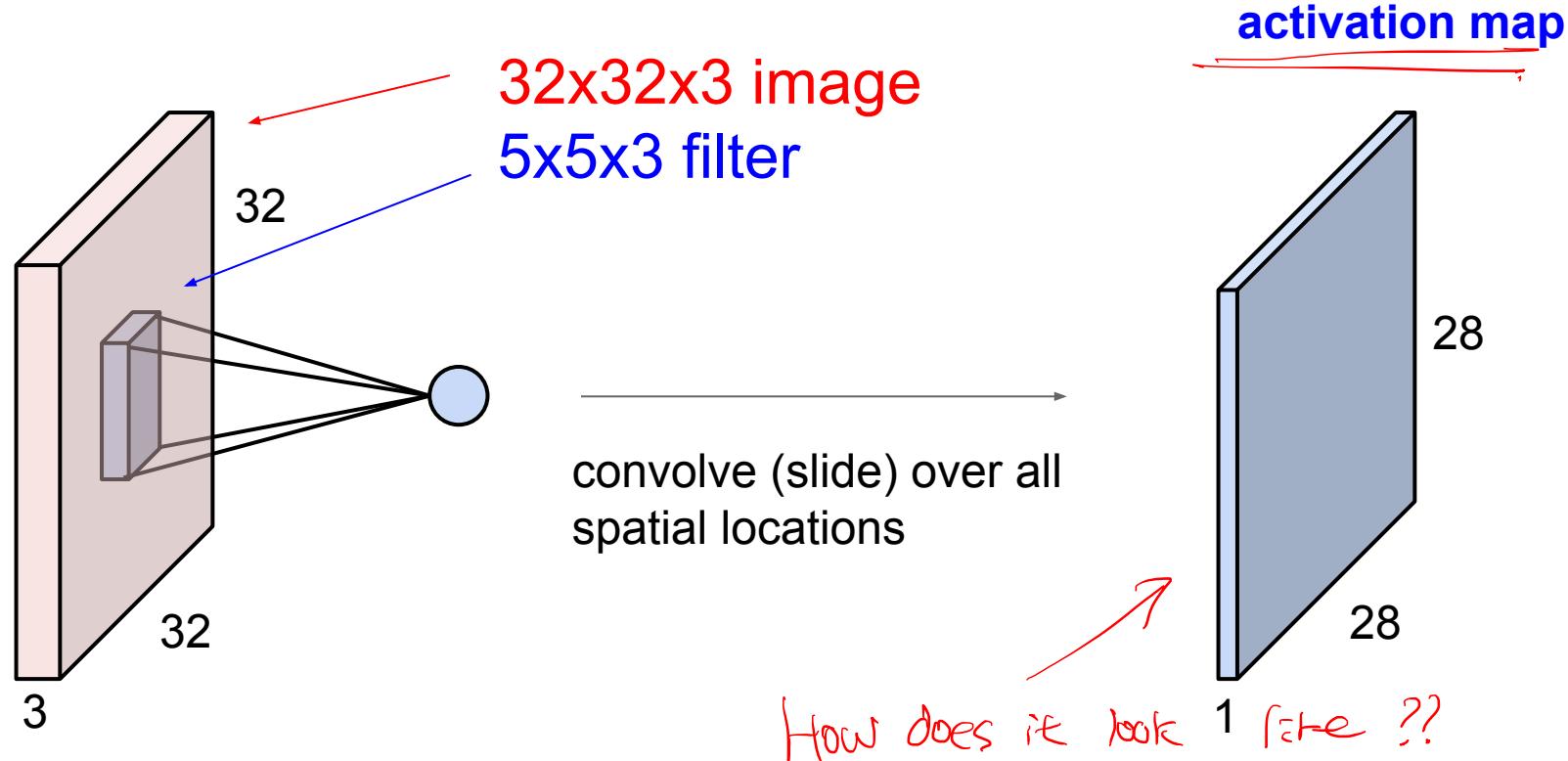
Consider Colors

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

Convolution Layer

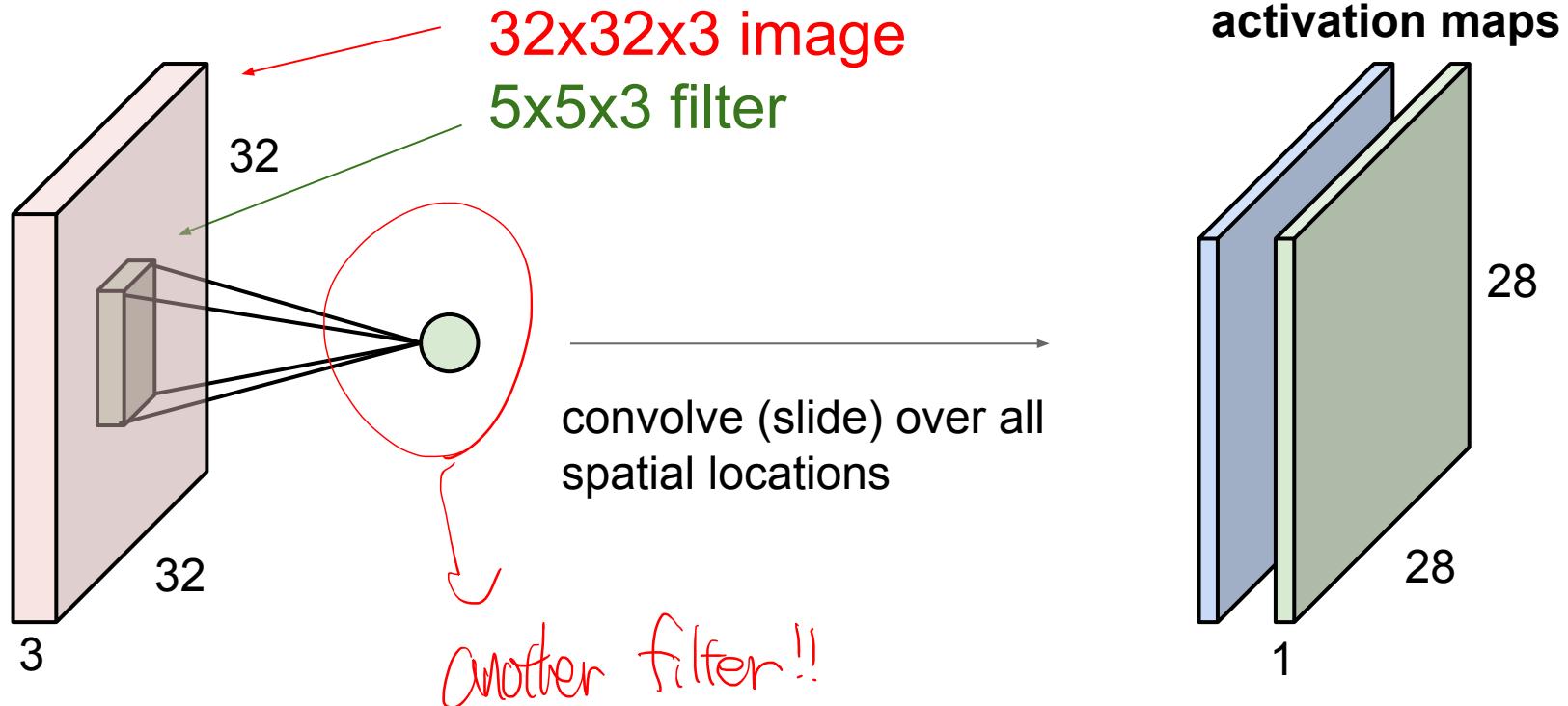


Convolution Layer

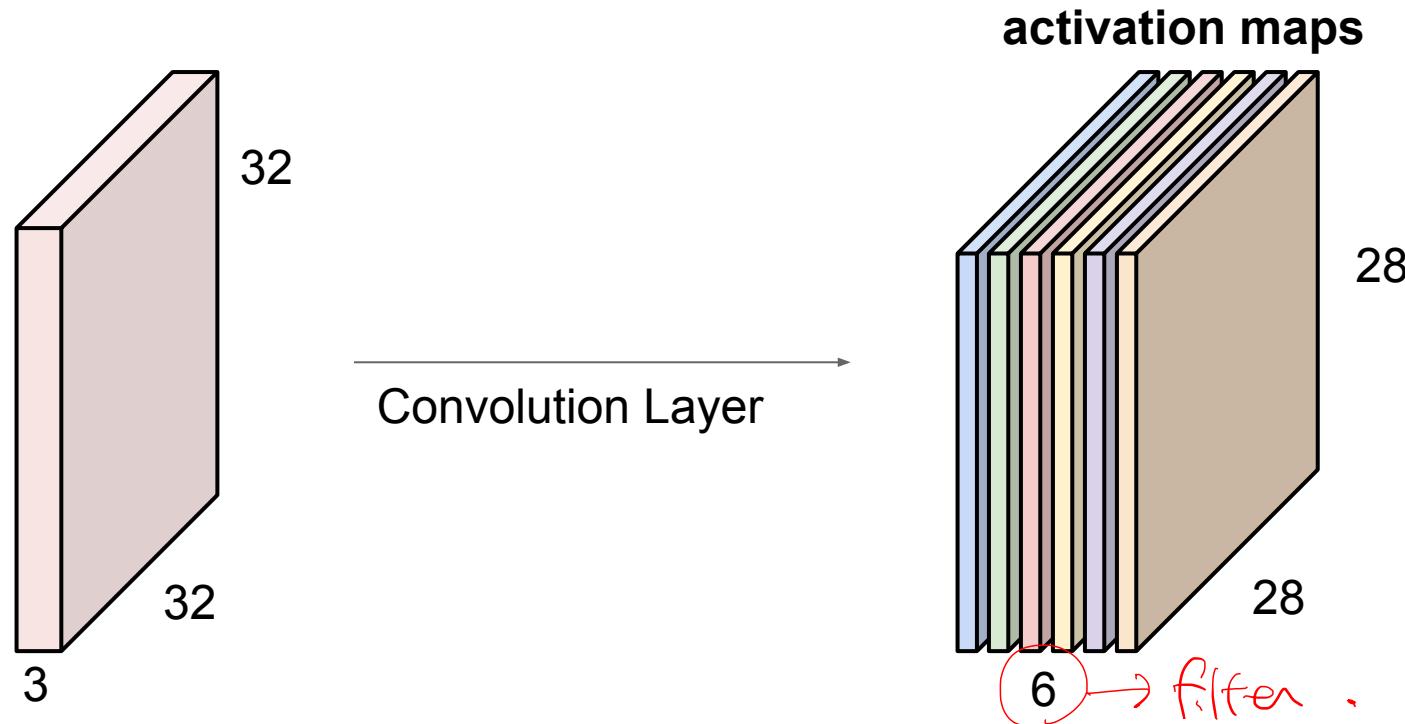


Convolution Layer

consider a second, green filter

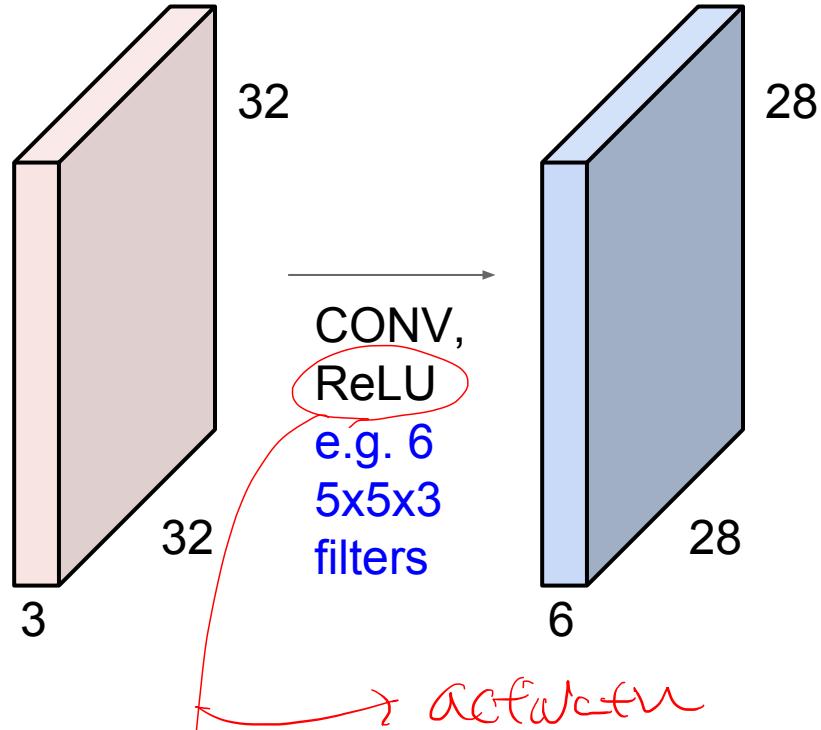


For example, if we had 6 5×5 filters, we'll get 6 separate activation maps:

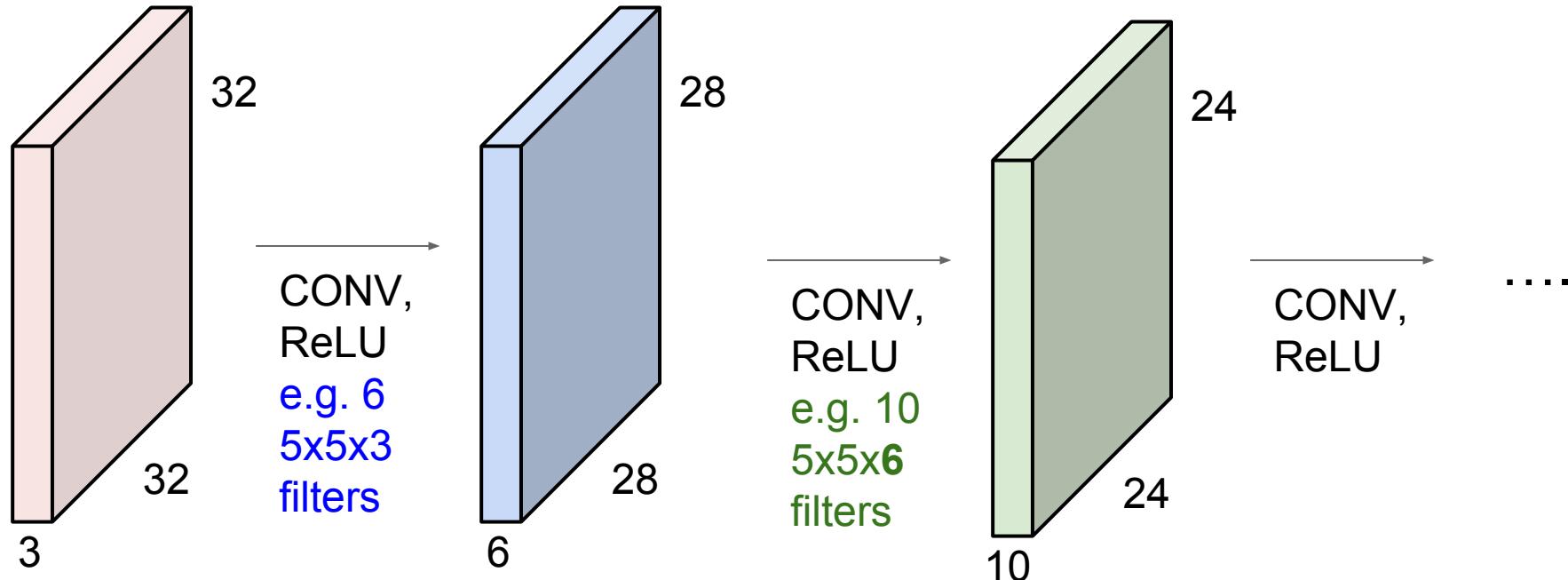


We stack these up to get a “new image” of size $28 \times 28 \times 6$!

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



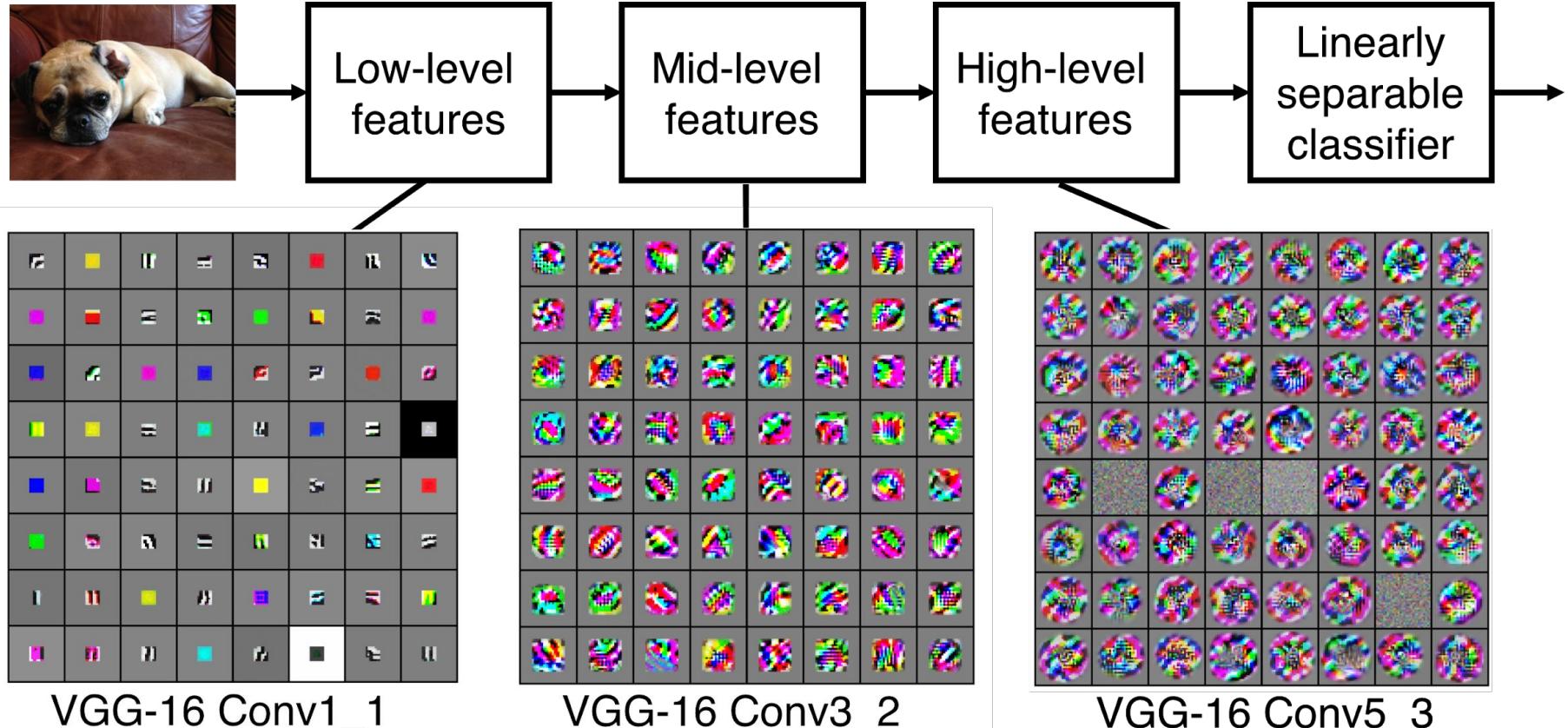
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Preview

[Zeiler and Fergus 2013]

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

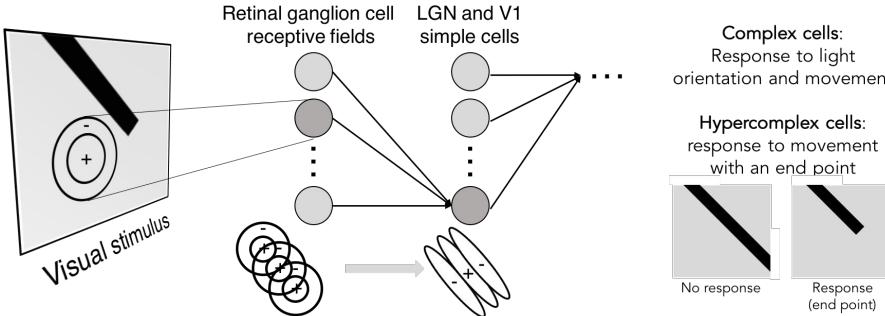
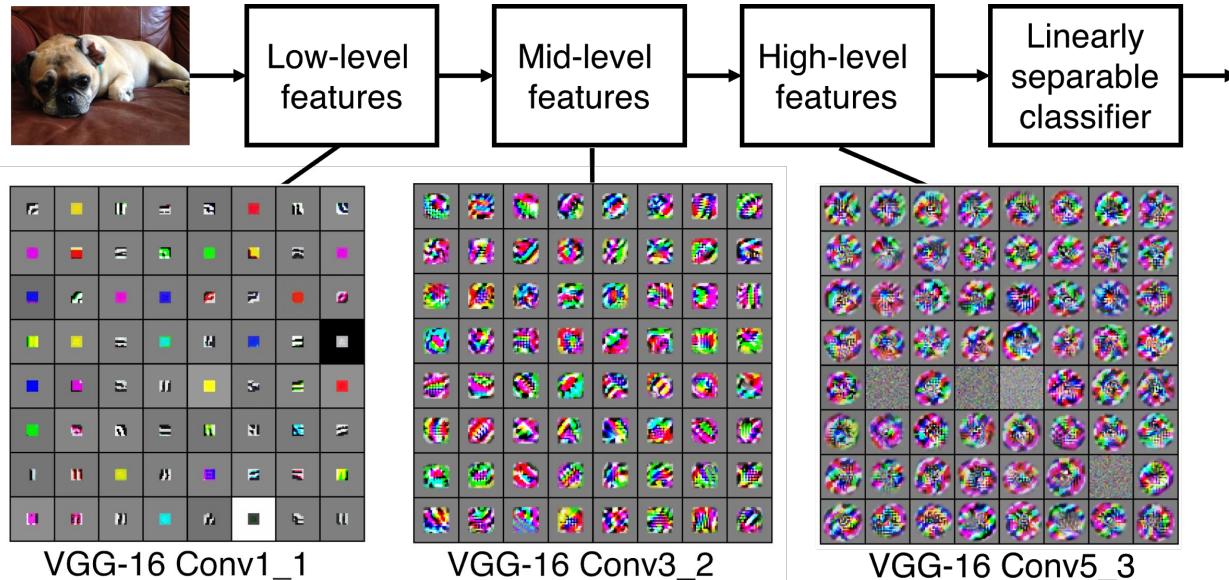


VGG-16 Conv1_1

VGG-16 Conv3_2

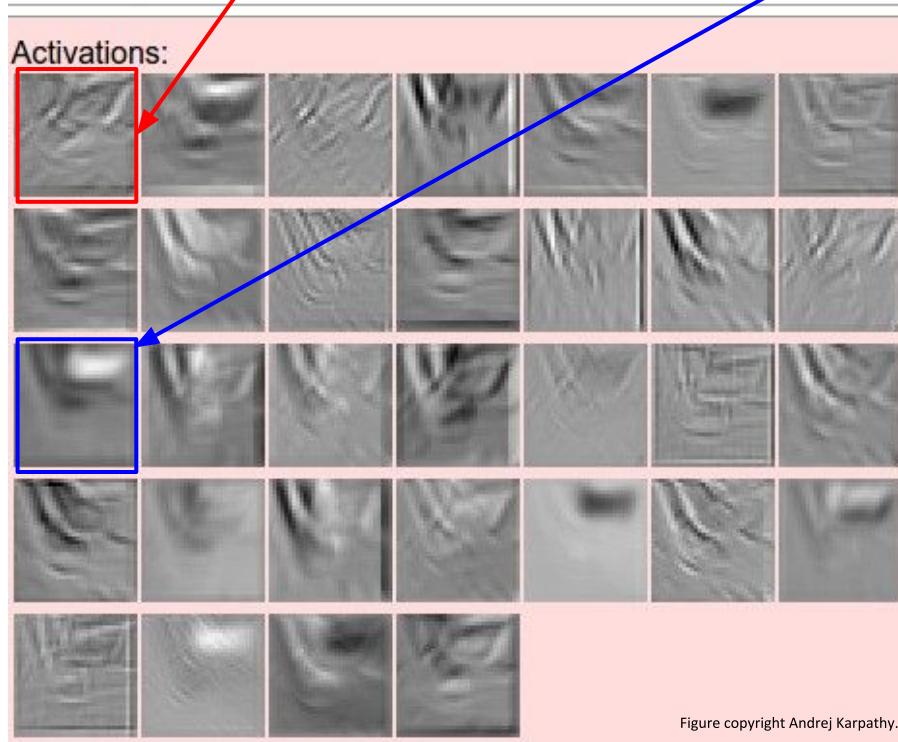
VGG-16 Conv5_3

Preview





one filter =>
one activation map



example 5x5 filters
(32 total)

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

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

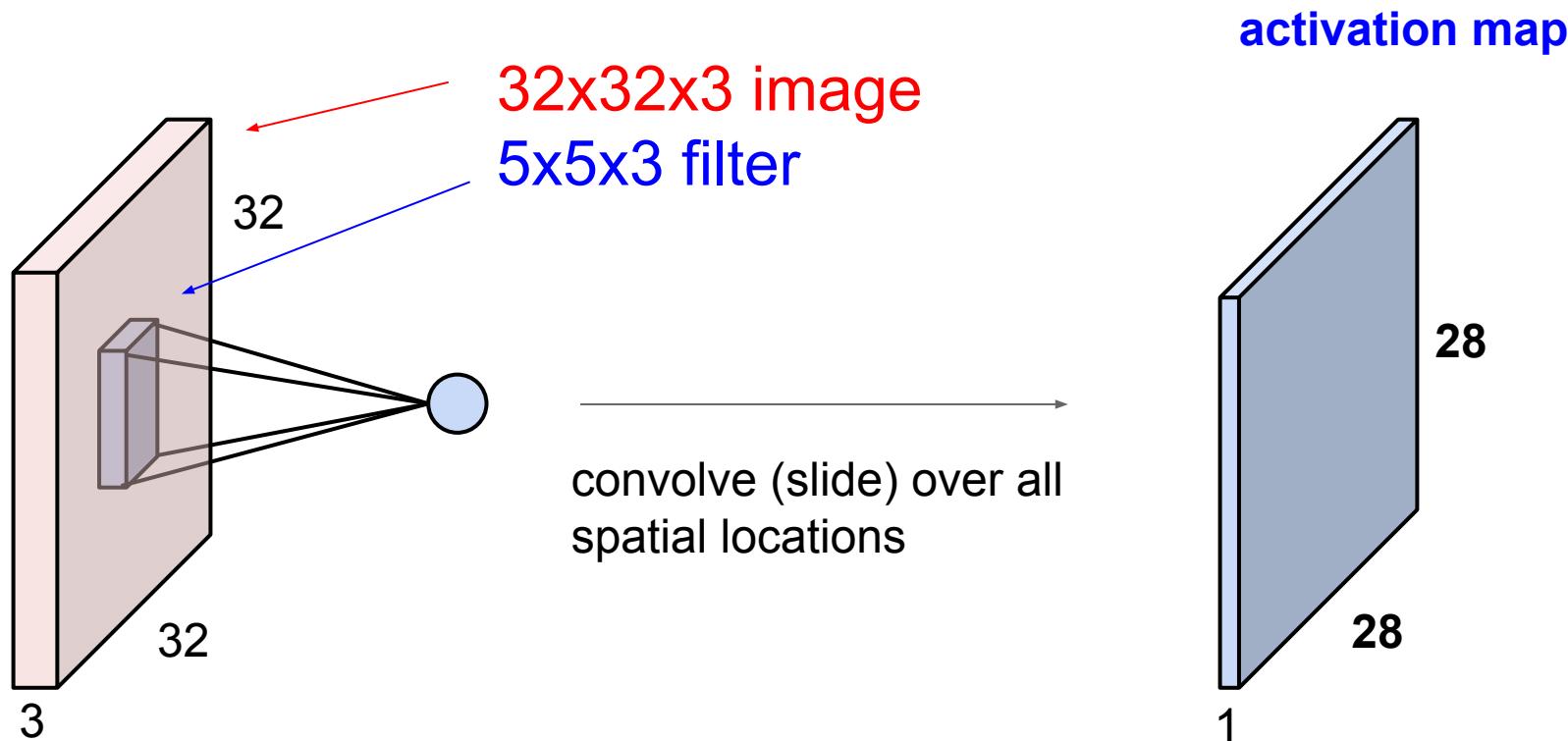


elementwise multiplication and sum of
a filter and the signal (image)

preview:

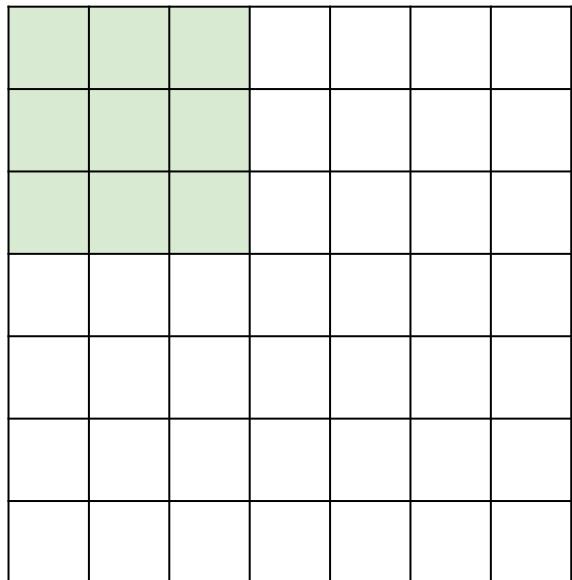


A closer look at spatial dimensions:



A closer look at spatial dimensions:

7

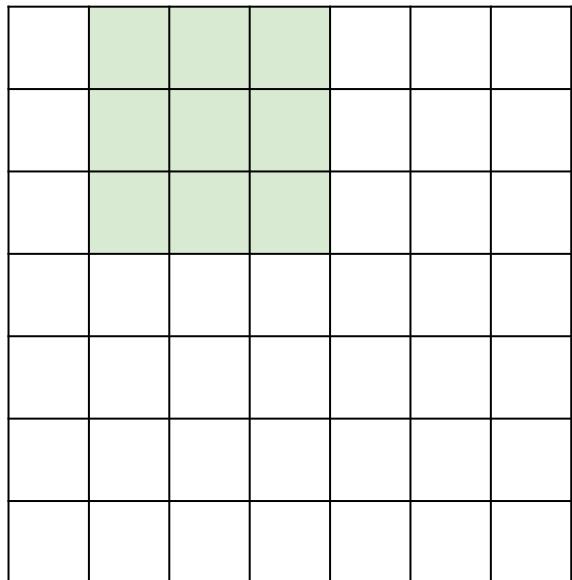


7

7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

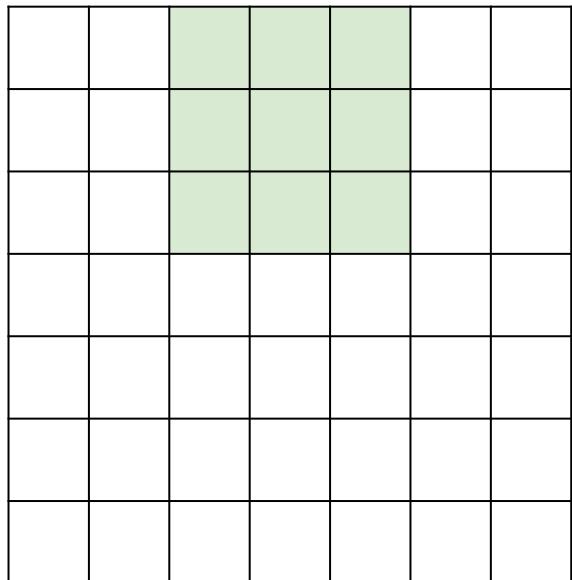
7



7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

7

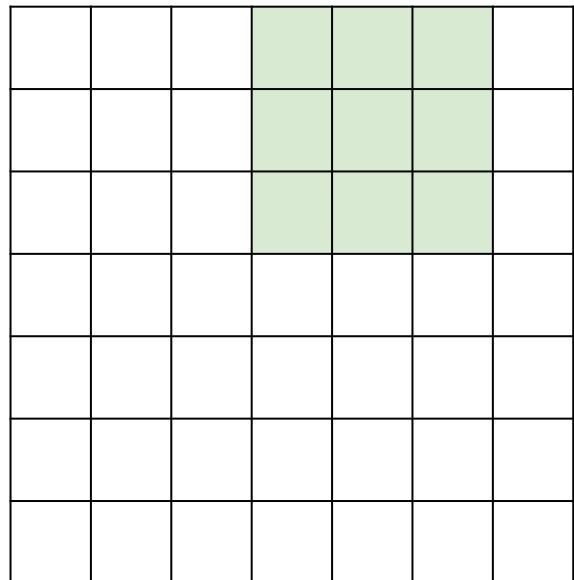


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

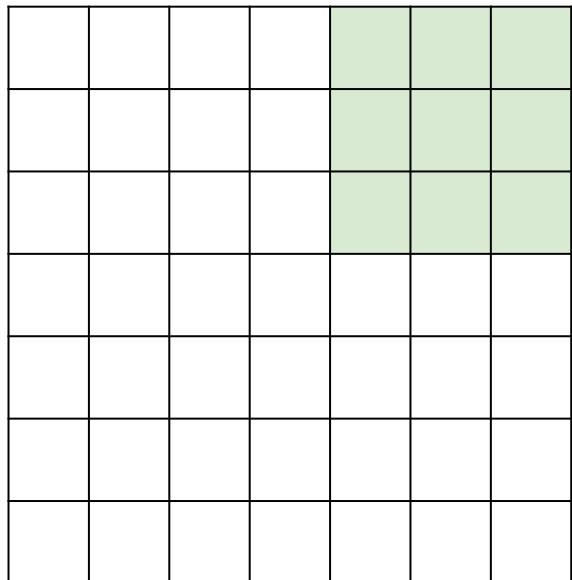


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

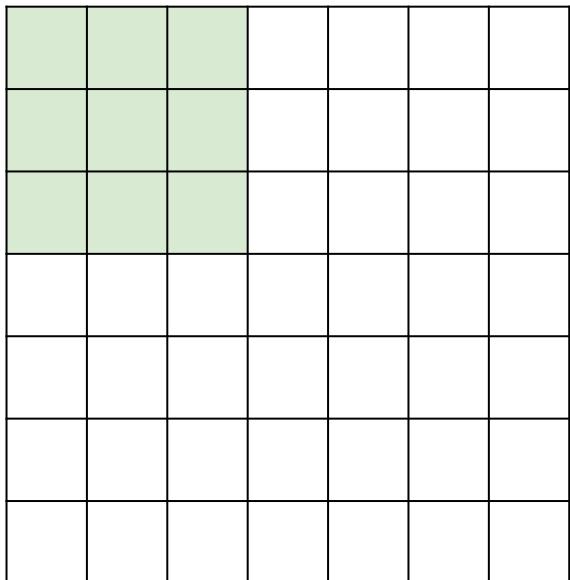


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

A closer look at spatial dimensions:

7

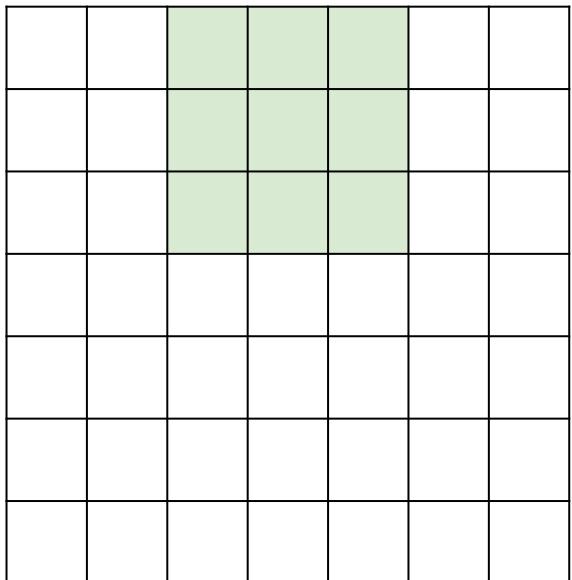


7

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

A closer look at spatial dimensions:

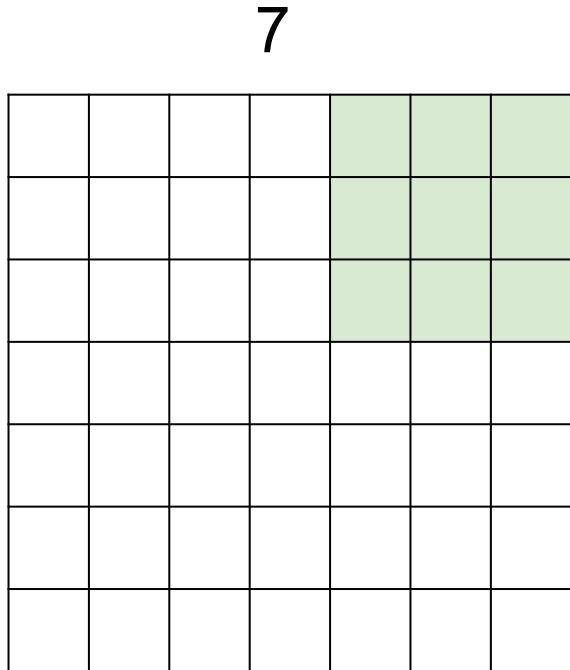
7



7

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

A closer look at spatial dimensions:

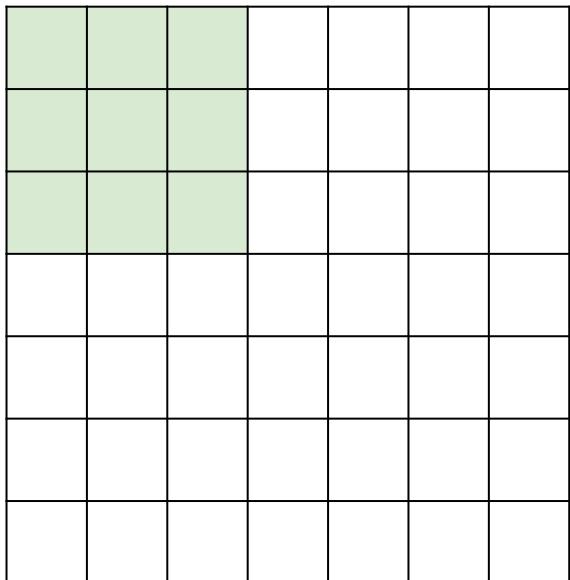


7

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

A closer look at spatial dimensions:

7

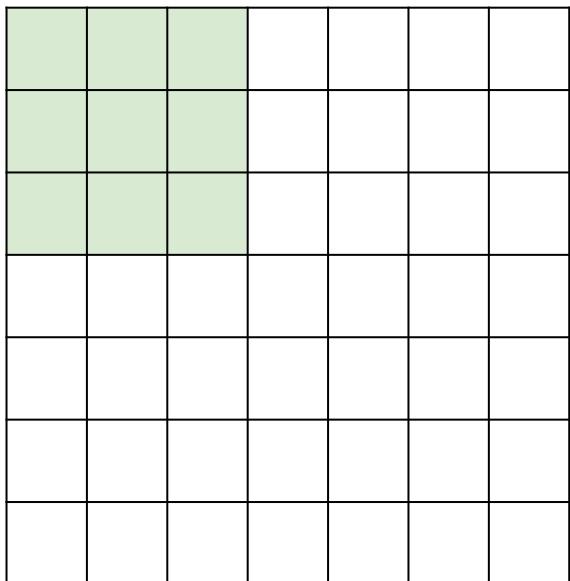


7

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

A closer look at spatial dimensions:

7



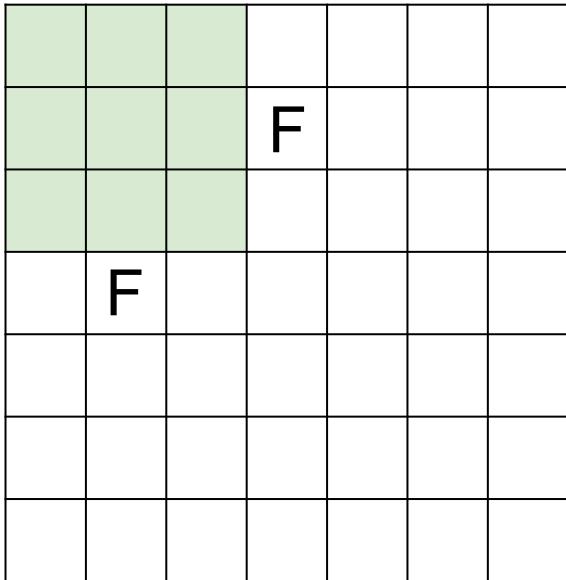
7

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

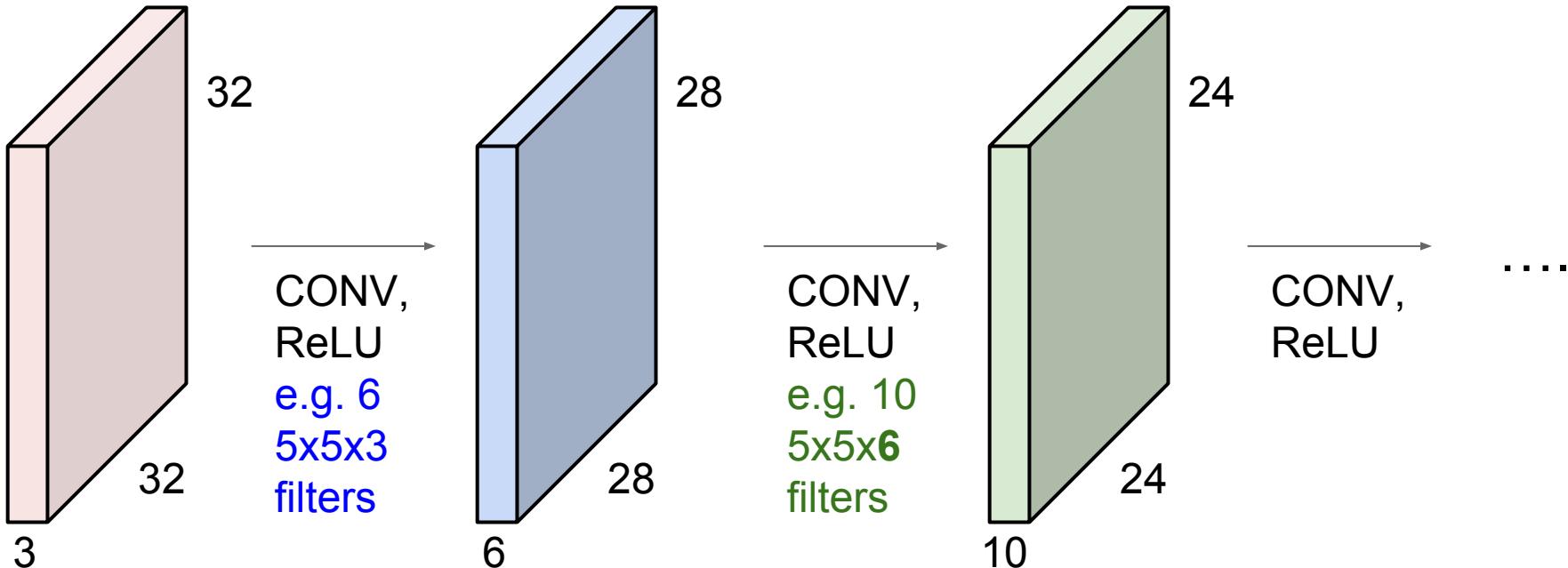
e.g. $F = 3 \Rightarrow$ zero pad with 1

$F = 5 \Rightarrow$ 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 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.

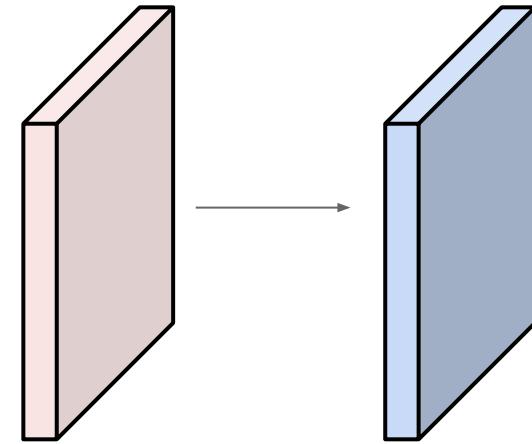


Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

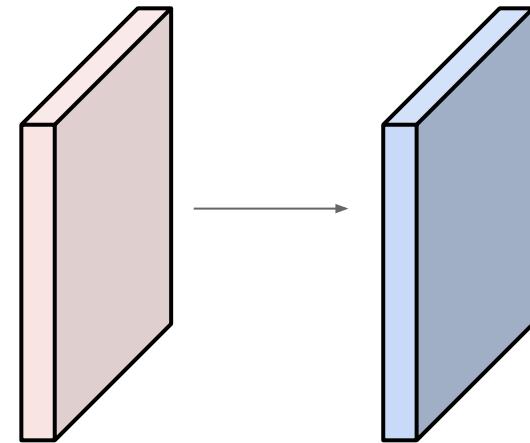
Output volume size: ?



Examples time:

Input volume: **32x32x3**

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



Output volume size:

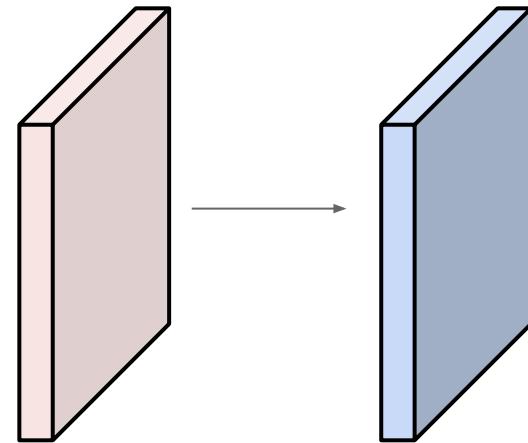
$(32+2*2-5)/1+1 = 32$ spatially, so

32x32x10

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

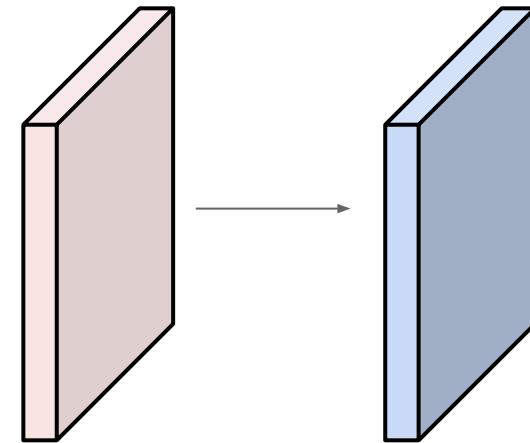


Number of parameters in this layer?

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)
=> $76*10 = 760$

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:

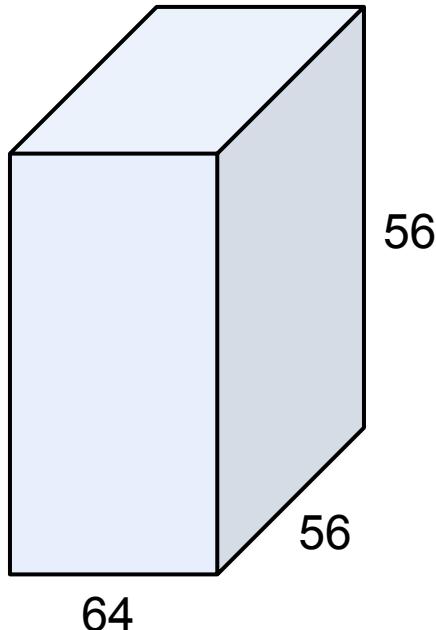
Summary. To summarize, the Conv Layer:

- 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.

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

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

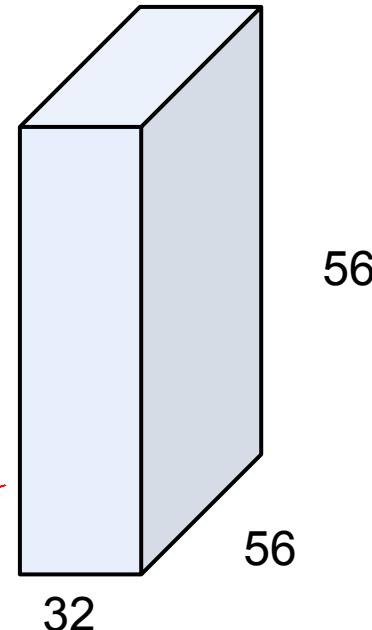
(btw, 1x1 convolution layers make perfect sense)



1x1 CONV
with 32 filters

→

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



Example: CONV layer in Torch

SpatialConvolution

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

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

The parameters are the following:

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

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

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

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

[Torch](#) is licensed under [BSD 3-clause](#).

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 .

Example: CONV layer in Caffe

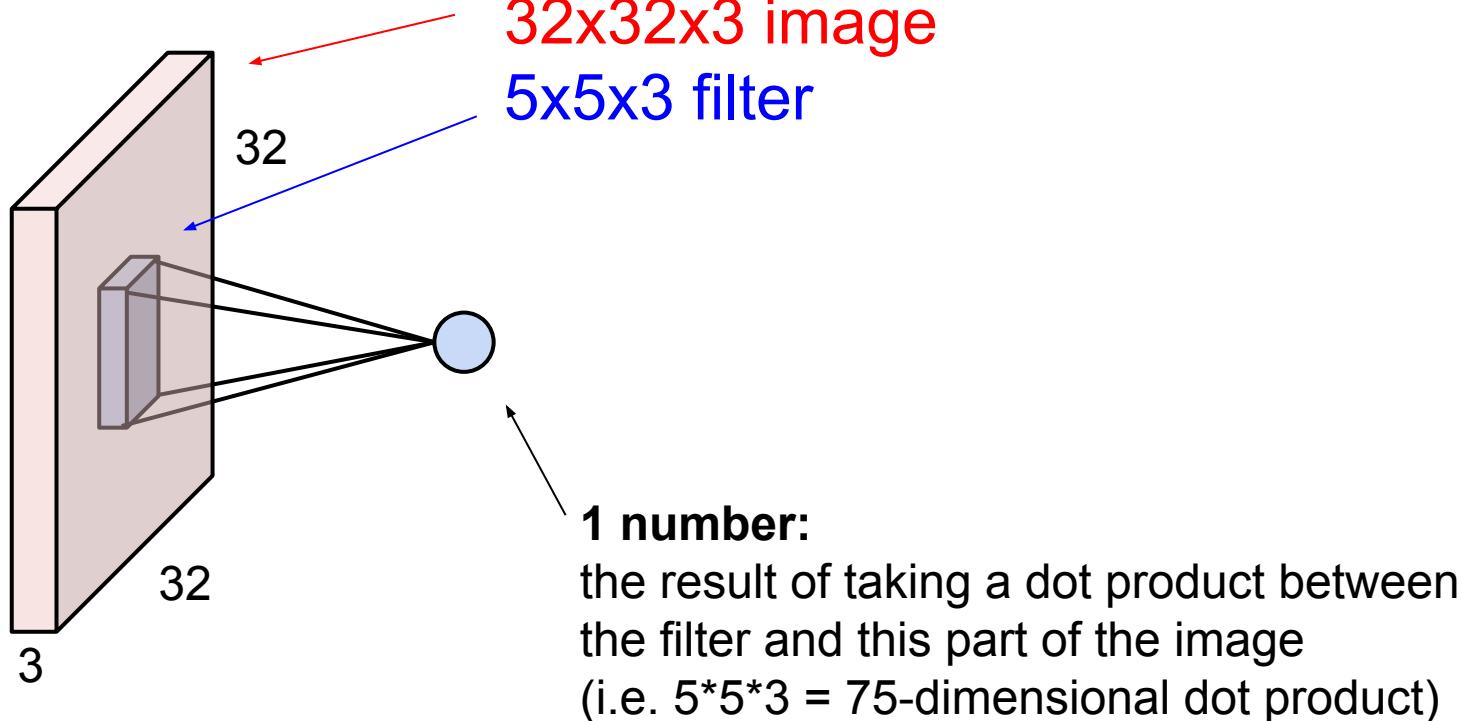
```
layer {
    name: "convl"
    type: "Convolution"
    bottom: "data"
    top: "convl"
    # learning rate and decay multipliers for the filters
    param { lr_mult: 1 decay_mult: 1 }
    # learning rate and decay multipliers for the biases
    param { lr_mult: 2 decay_mult: 0 }
    convolution_param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

Summary. To summarize, the Conv Layer:

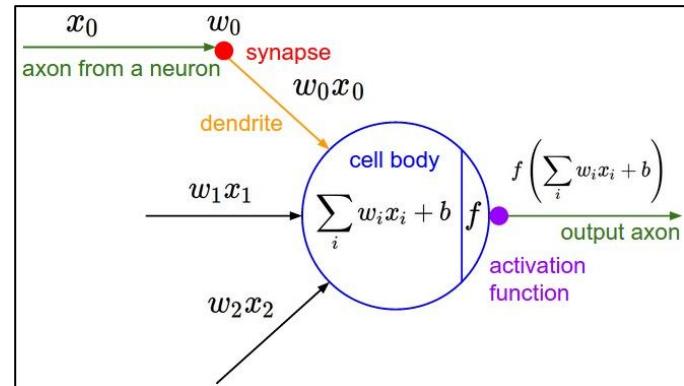
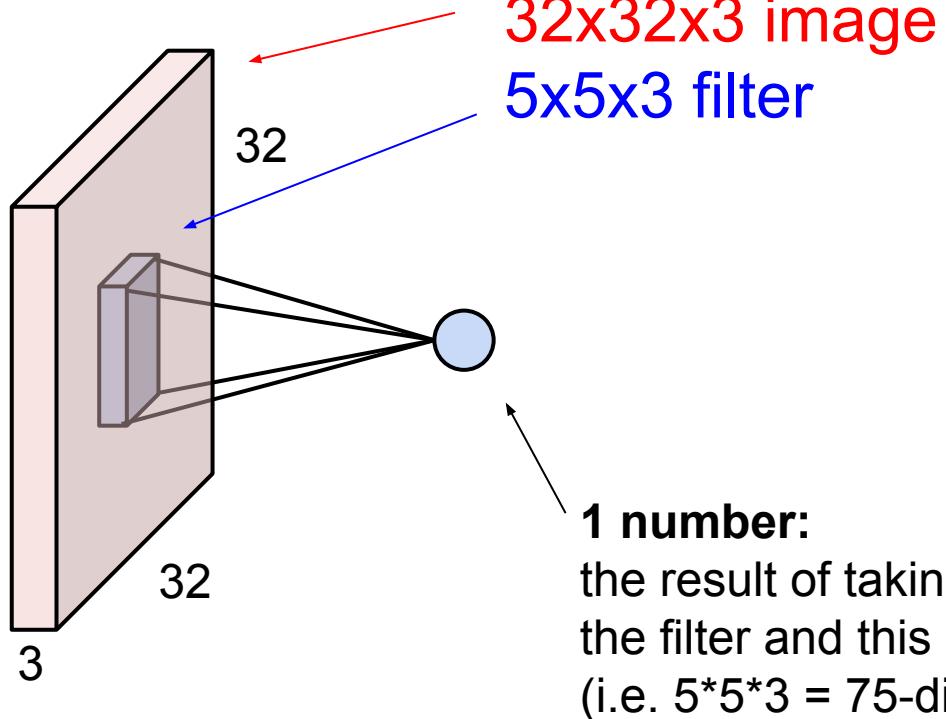
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
 - Number of filters K ,
 - their spatial extent F ,
 - the stride S ,
 - the amount of zero padding P .

[Caffe](#) is licensed under [BSD 2-Clause](#).

The brain/neuron view of CONV Layer

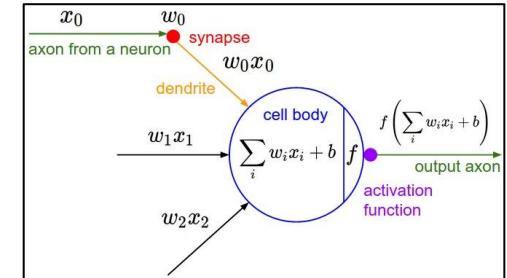
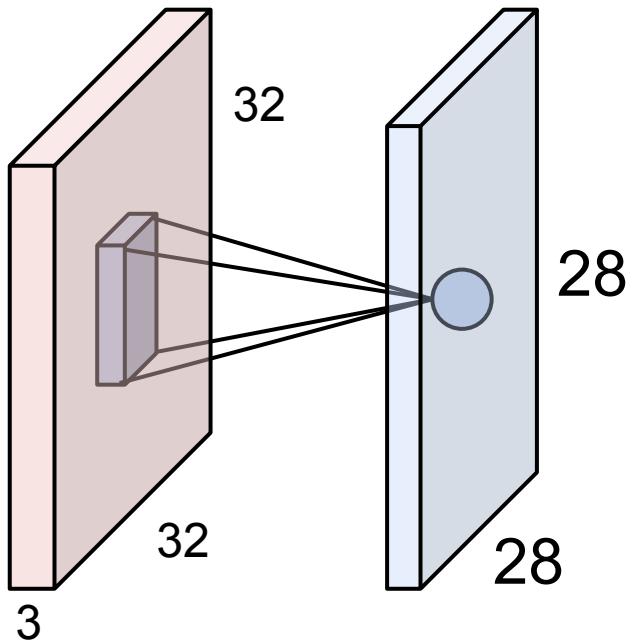


The brain/neuron view of CONV Layer



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

The brain/neuron view of CONV Layer

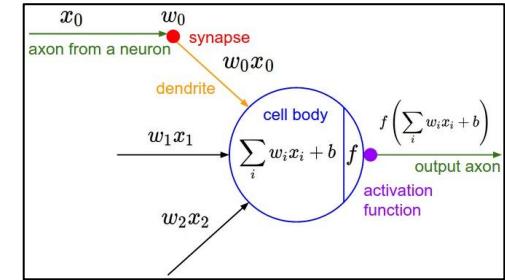
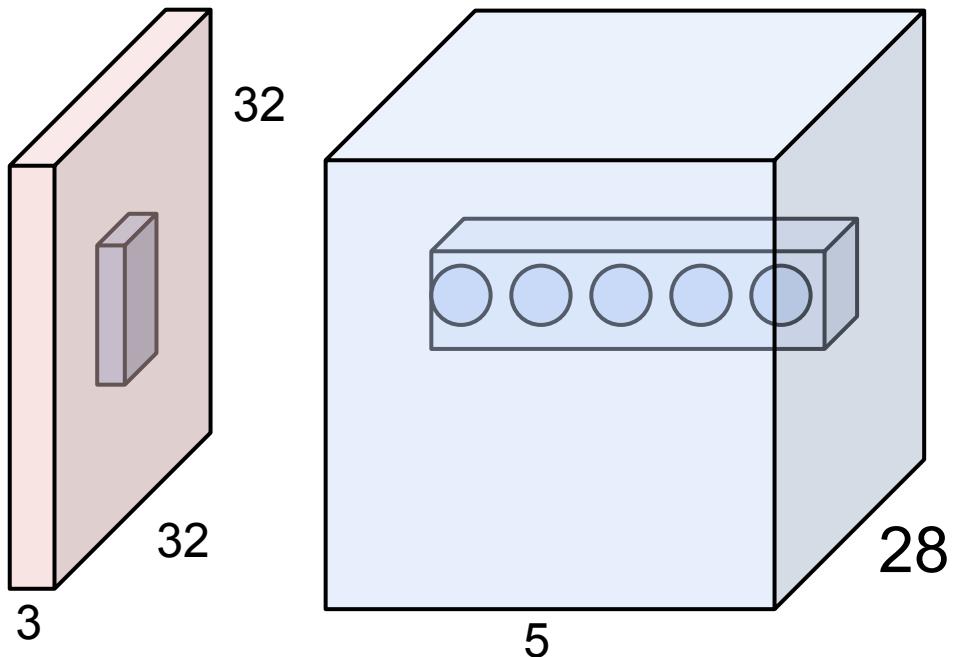


An activation map is a 28x28 sheet of neuron outputs:

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

“5x5 filter” -> “5x5 receptive field for each neuron”

The brain/neuron view of CONV Layer



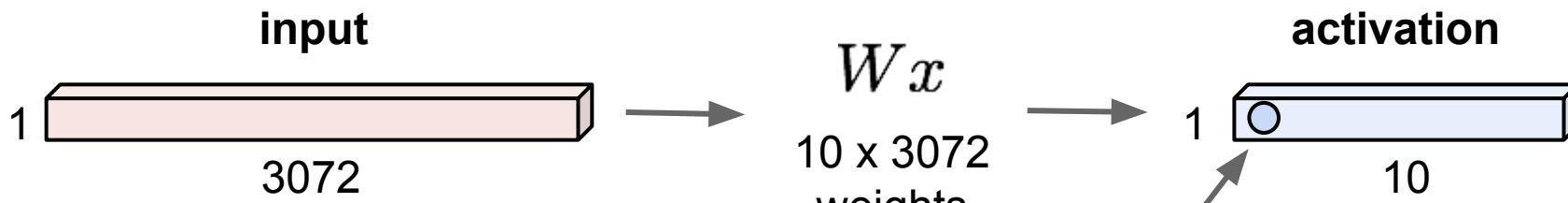
E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
($28 \times 28 \times 5$)

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

Reminder: Fully Connected Layer

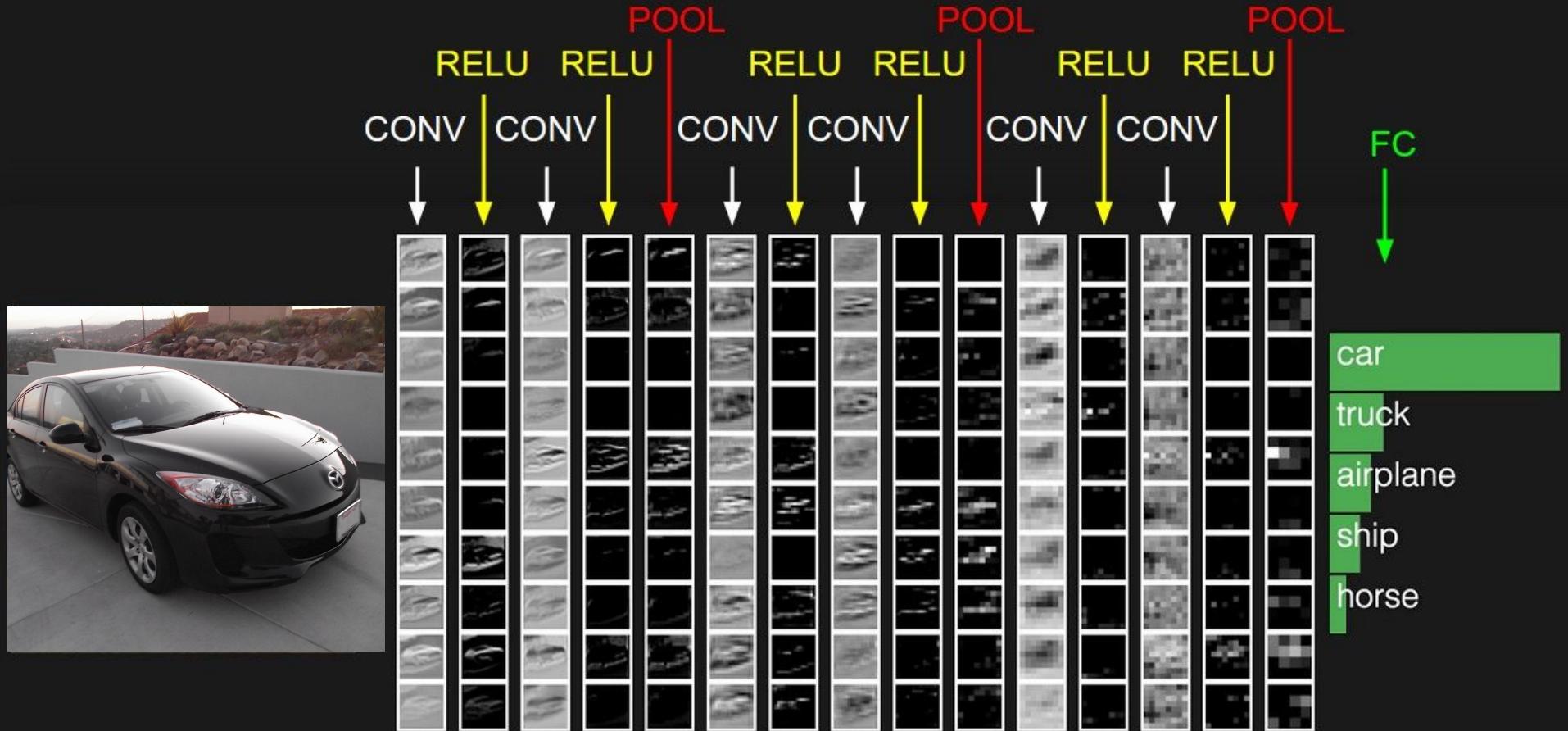
32x32x3 image -> stretch to 3072×1

Each neuron
looks at the full
input volume



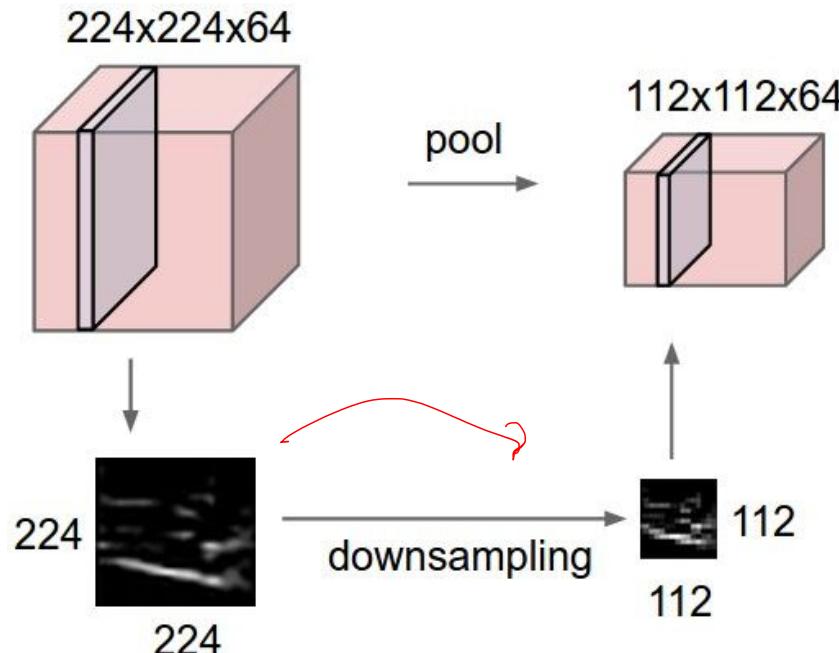
1 number:
the result of taking a dot product
between a row of W and the input
(a 3072-dimensional dot product)

two more layers to go: POOL/FC



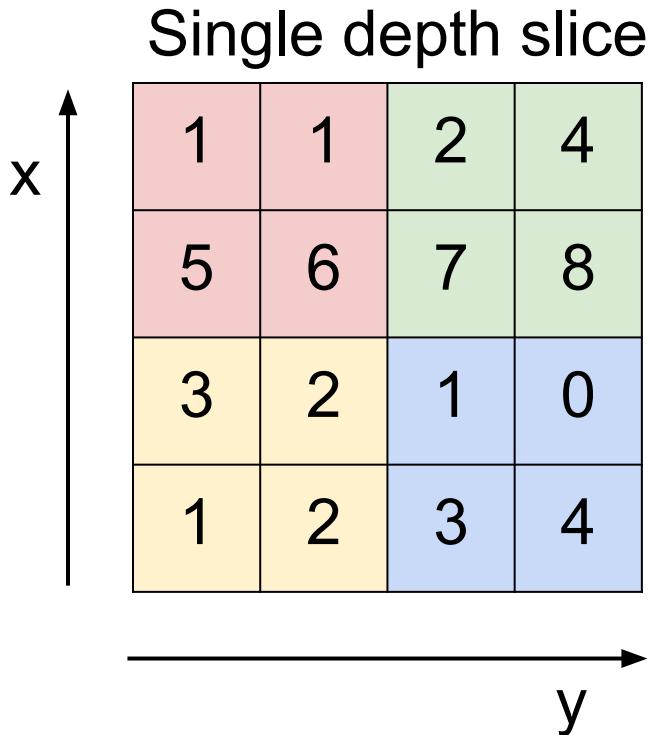
Pooling layer

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



Pooling !!

MAX POOLING



max pool with 2x2 filters
and stride 2

An arrow points from the input matrix to the output matrix, indicating the transformation process.

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

Common settings:

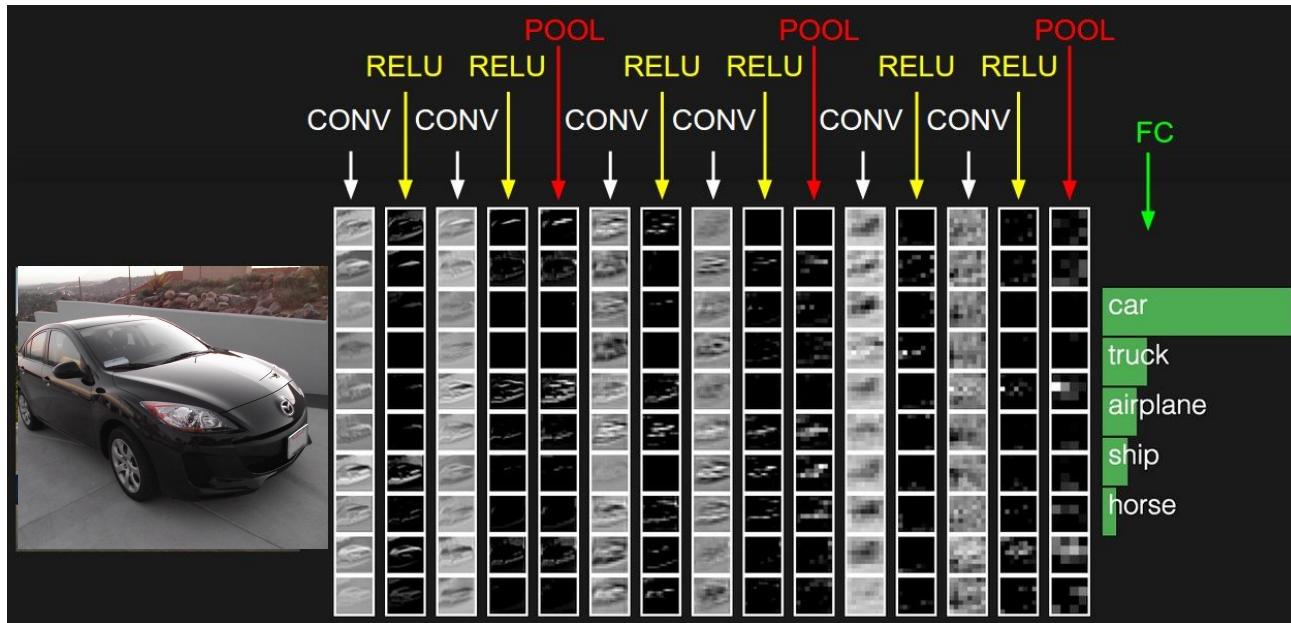
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
 - their spatial extent F ,
 - the stride S ,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F)/S + 1$
 - $H_2 = (H_1 - F)/S + 1$
 - $D_2 = D_1$
- Introduces zero parameters since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

$F = 2, S = 2$

$F = 3, S = 2$

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

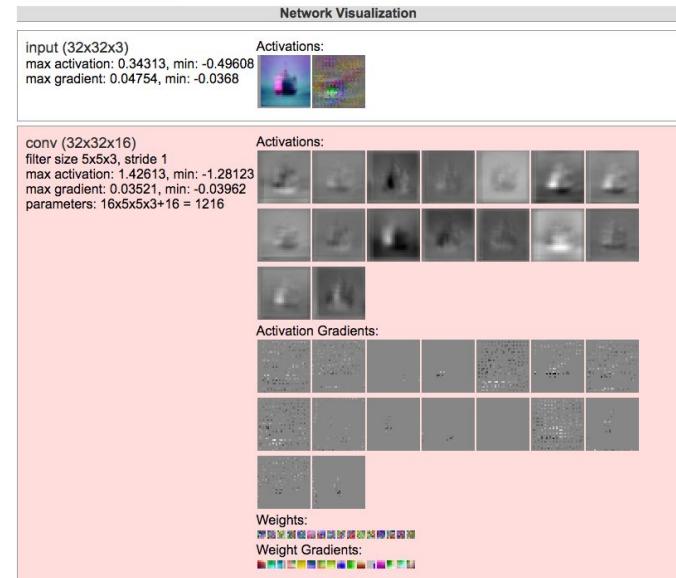
Description

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