

# Lecture 5: Convolutional Neural Networks

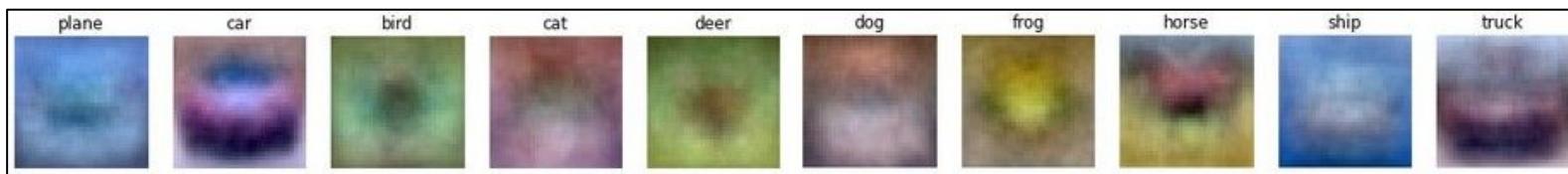
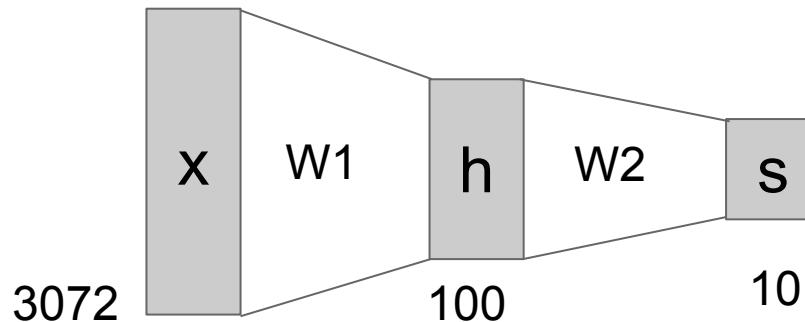
# Wrapping up: Neural Networks

Linear score function:

$$f = Wx$$

2-layer Neural Network

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



# Next: Convolutional Neural Networks

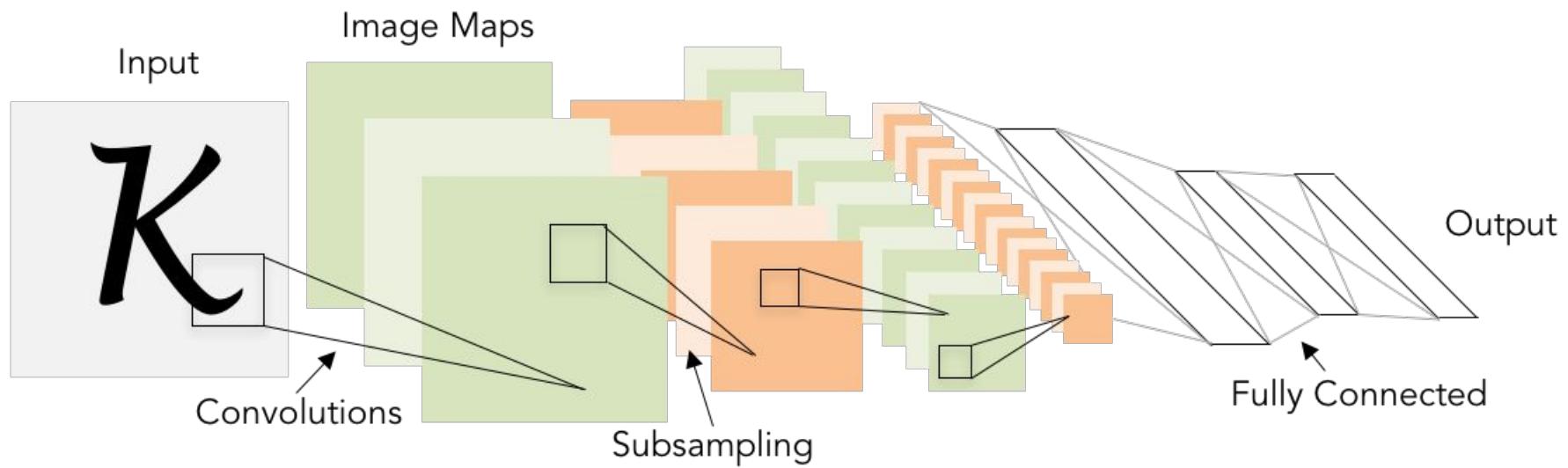
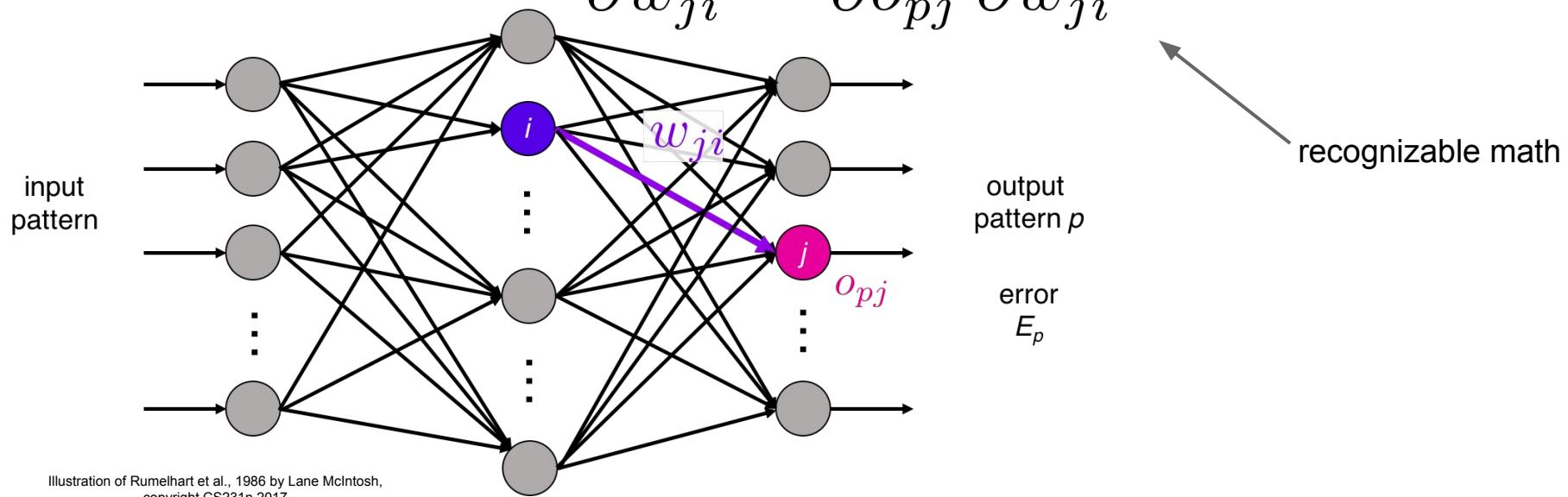


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

# A bit of history...

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}$$



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

# A bit of history...

[Hinton and Salakhutdinov 2006]

Reinvigorated research in  
Deep Learning

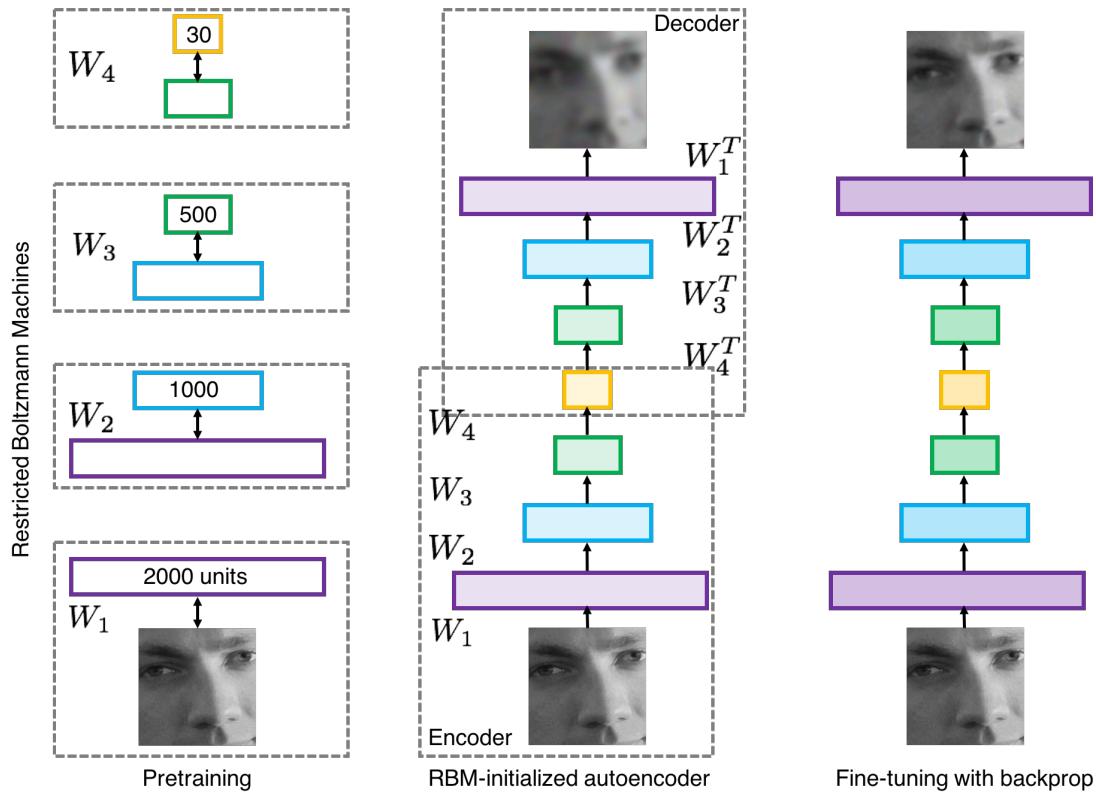


Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

# First strong results

## **Acoustic Modeling using Deep Belief Networks**

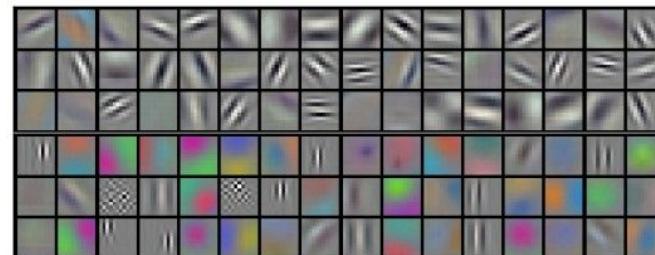
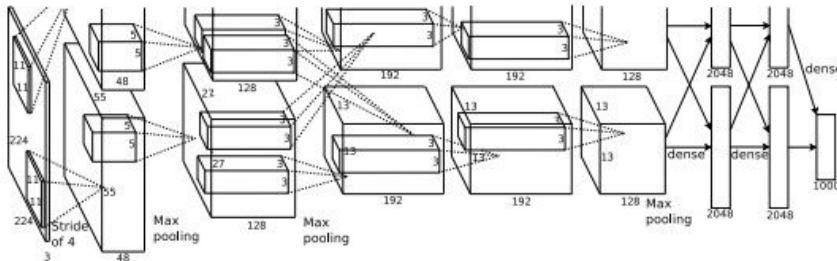
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010

## **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



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

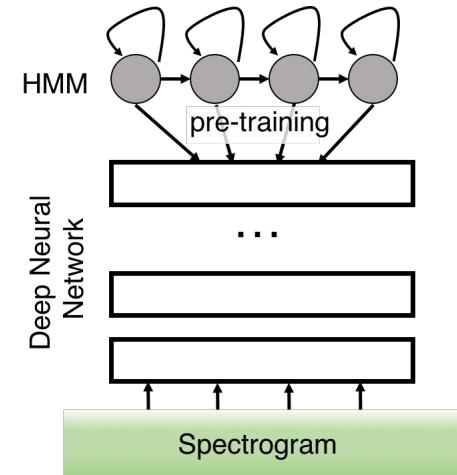


Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

# A bit of history:

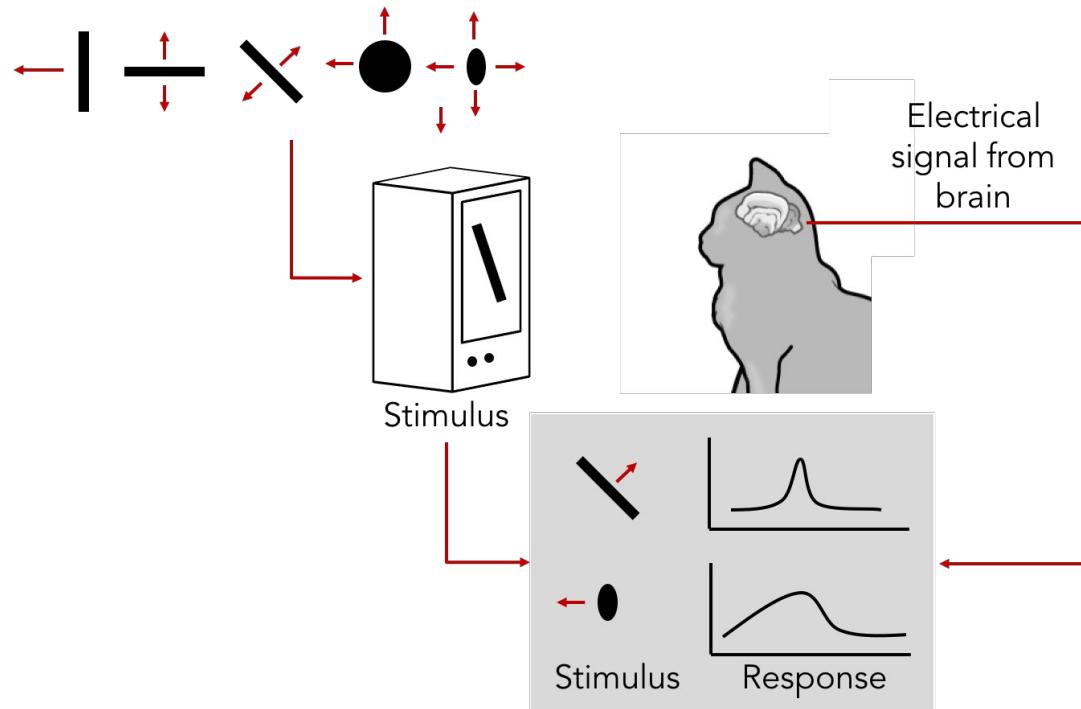
**Hubel & Wiesel,  
1959**

RECEPTIVE FIELDS OF SINGLE  
NEURONES IN  
THE CAT'S STRIATE CORTEX

**1962**

RECEPTIVE FIELDS, BINOCULAR  
INTERACTION  
AND FUNCTIONAL ARCHITECTURE IN  
THE CAT'S VISUAL CORTEX

**1968...**

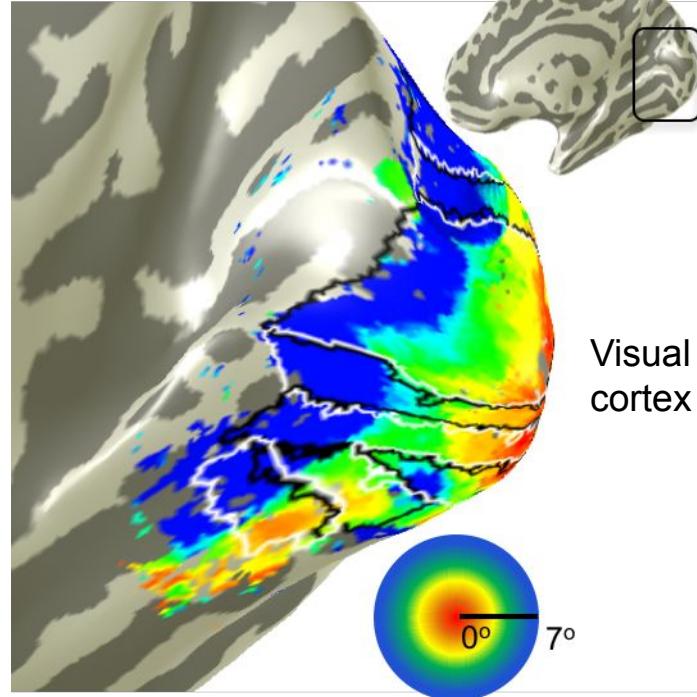
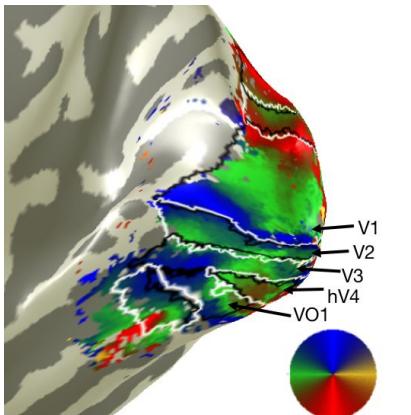


[Cat image](#) by CNX OpenStax is licensed under CC BY 4.0; changes made

# A bit of history

Human brain

**Topographical mapping in the cortex:**  
nearby cells in cortex represent  
nearby regions in the visual field



Retinotopy images courtesy of Jesse Gomez in the  
Stanford Vision & Perception Neuroscience Lab.

# Hierarchical organization

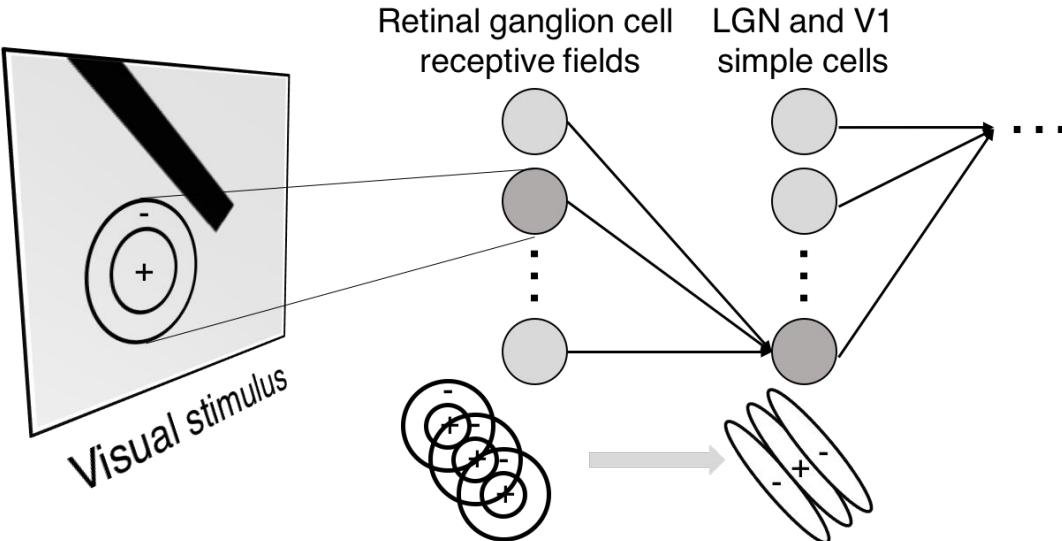
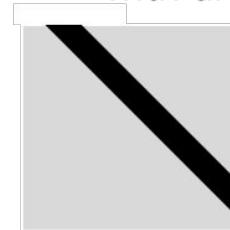


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

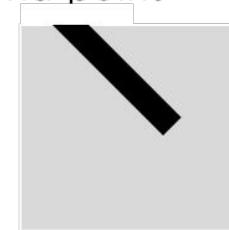
Simple cells:  
Response to light orientation

Complex cells:  
Response to light orientation and movement

Hypercomplex cells:  
response to movement with an end point



No response



Response  
(end point)

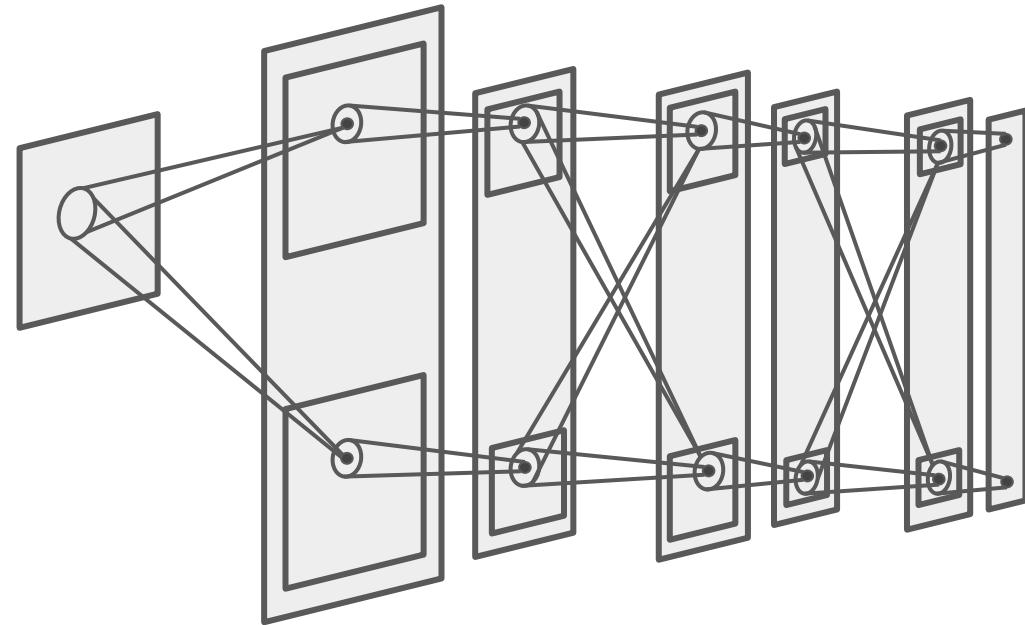
# A bit of history:

## Neocognitron [Fukushima 1980]

“sandwich” architecture (SCSCSC...)

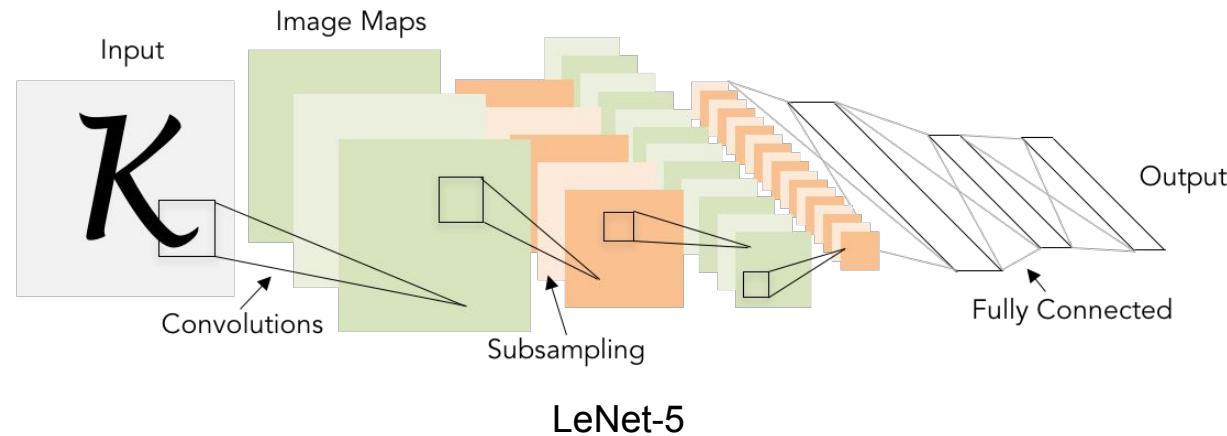
simple cells: modifiable parameters

complex cells: perform pooling



# 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]*

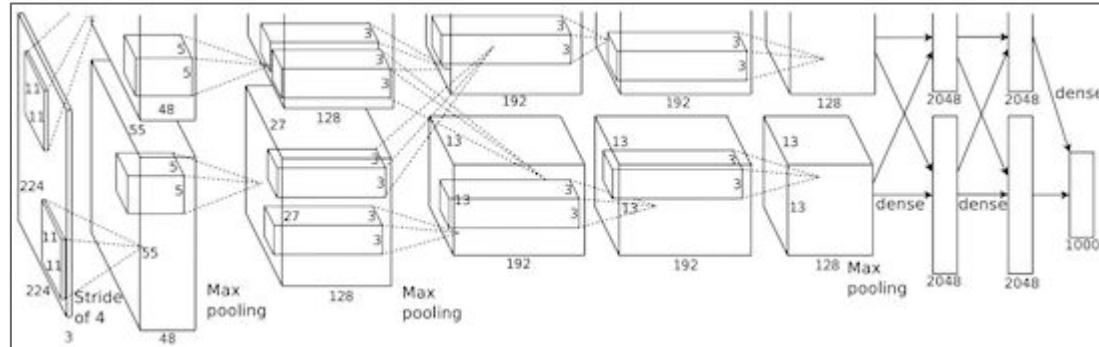


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

“AlexNet”

# Fast-forward to today: ConvNets are everywhere

Classification



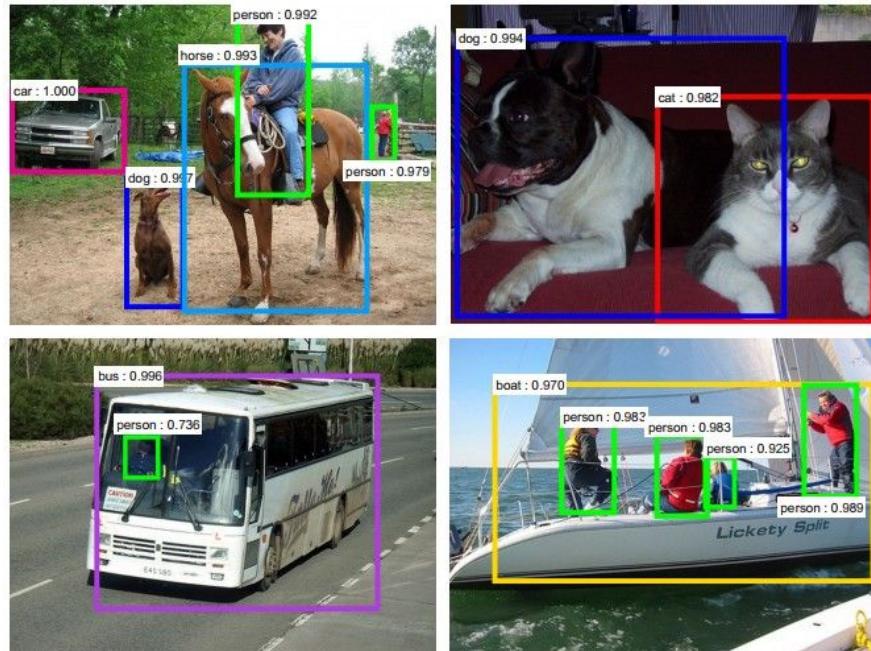
Retrieval



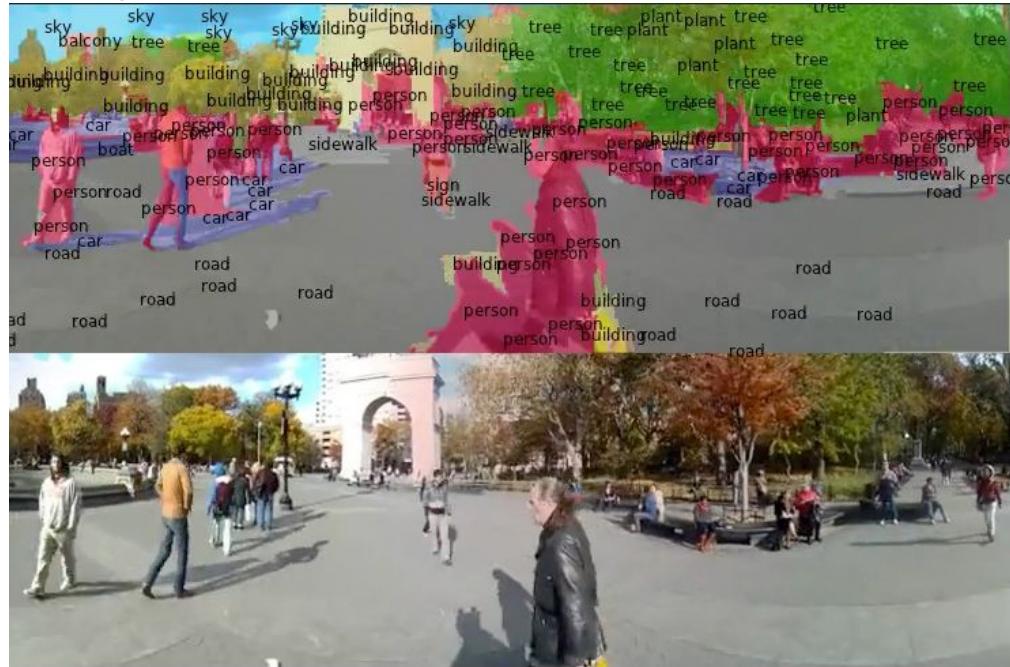
Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

# Fast-forward to today: ConvNets are everywhere

## Detection



## Segmentation



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

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

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.



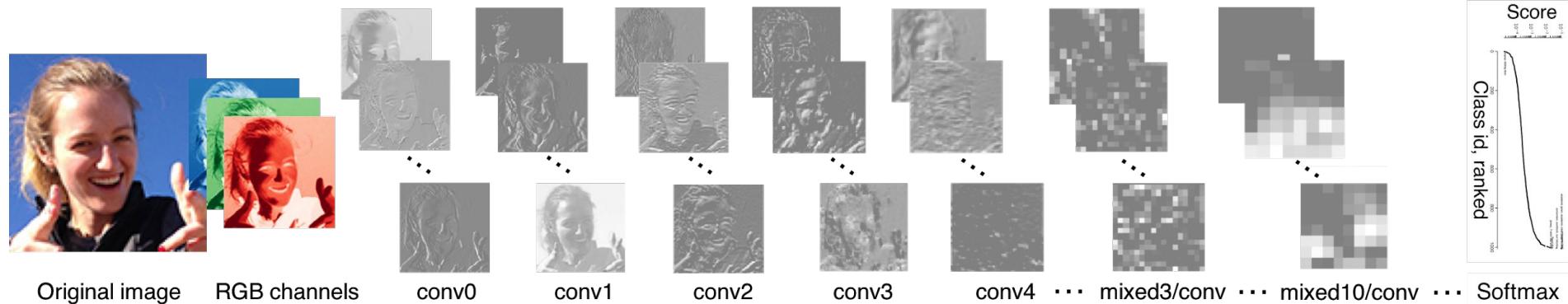
[This image](#) by GBPublic\_PR is licensed under [CC-BY 2.0](#)

## NVIDIA Tesla line

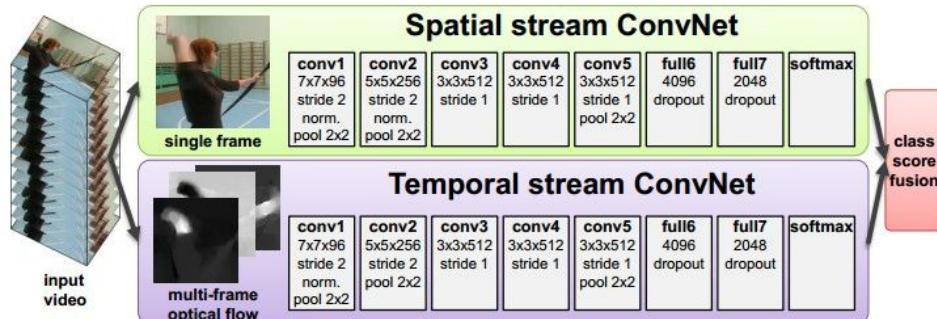
(these are the GPUs on rye01.stanford.edu)

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



[Taigman et al. 2014]



[Simonyan et al. 2014]

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

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.

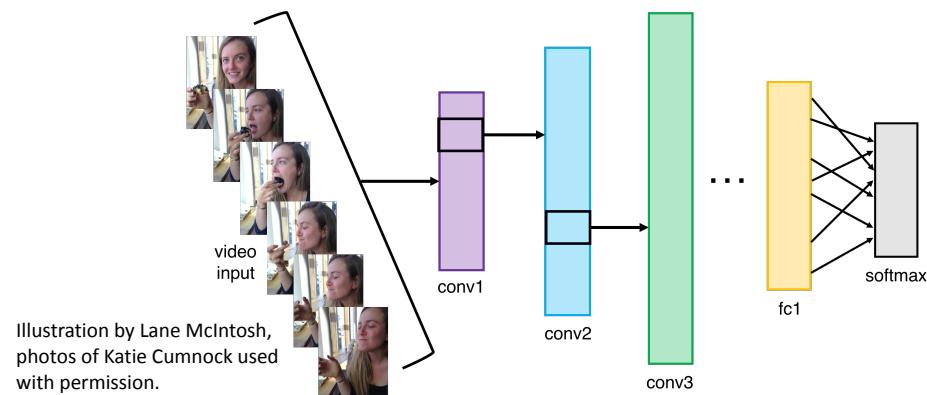


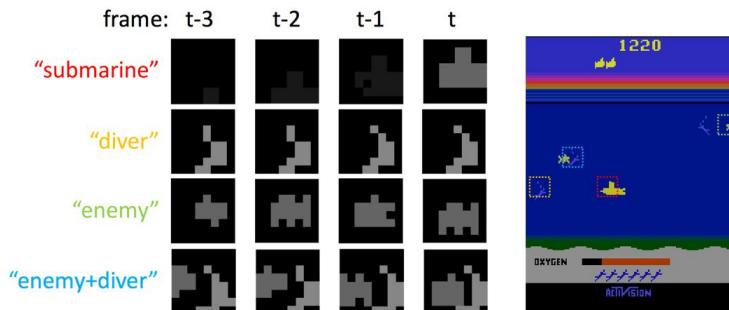
Illustration by Lane McIntosh,  
photos of Katie Cumnock used  
with permission.

# Fast-forward to today: ConvNets are everywhere

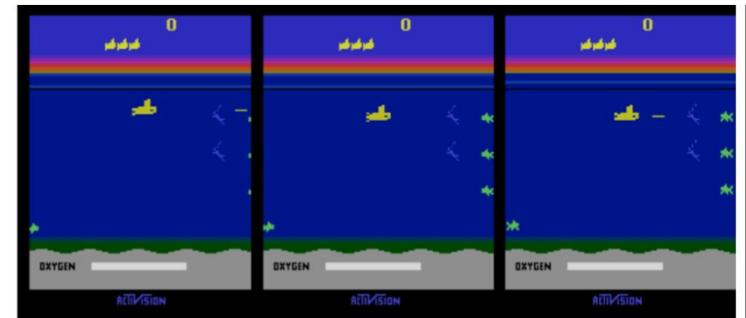


Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]

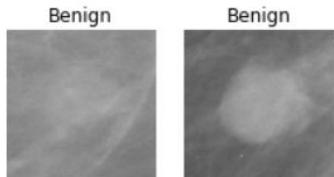


[Guo et al. 2014]



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]

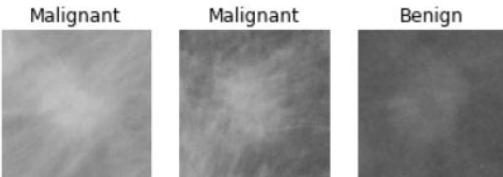


Figure copyright Levy et al. 2016.  
Reproduced with permission.



[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*



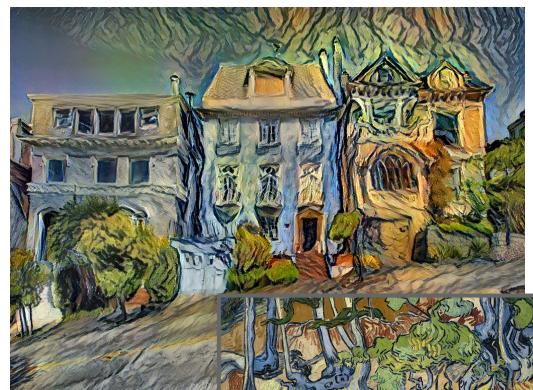
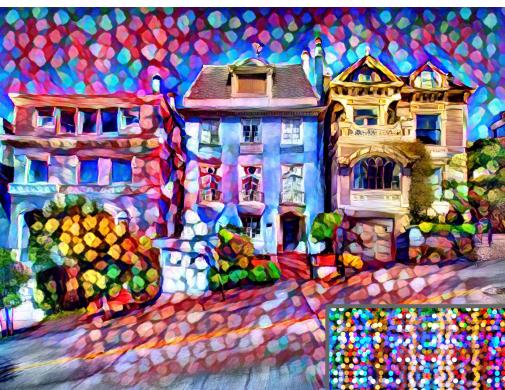
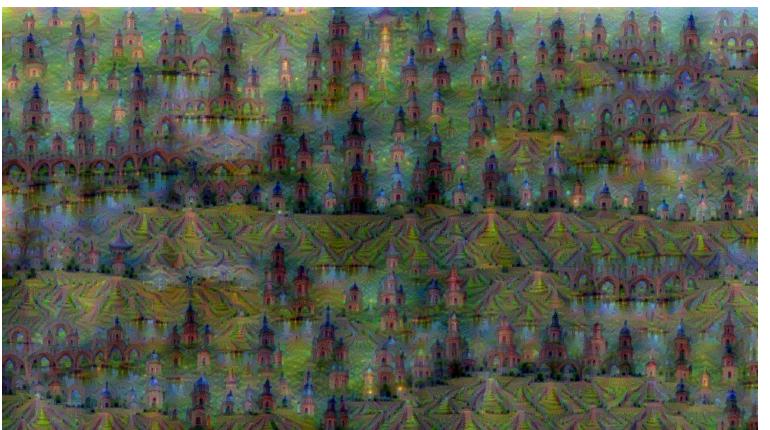
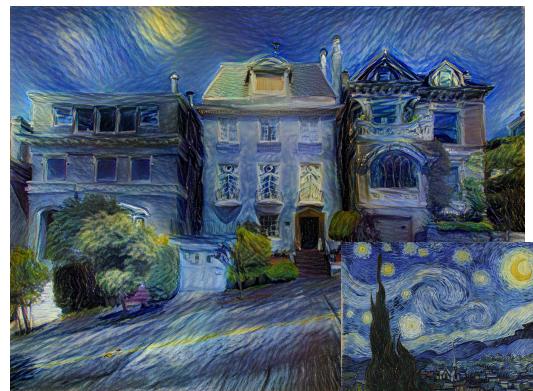
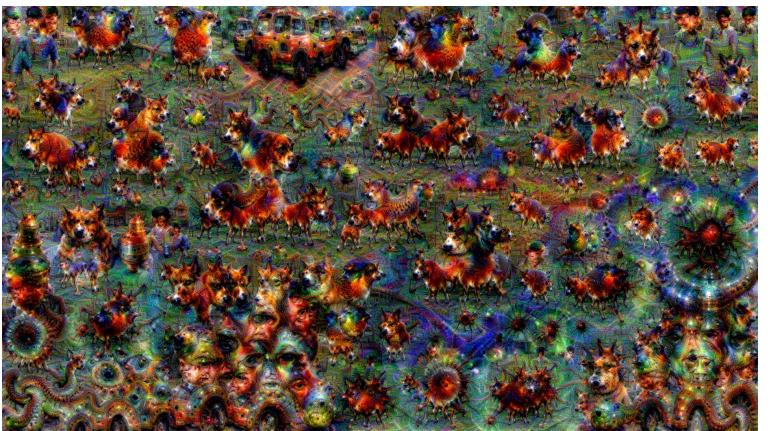
*A woman standing on a beach holding a surfboard*

# 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

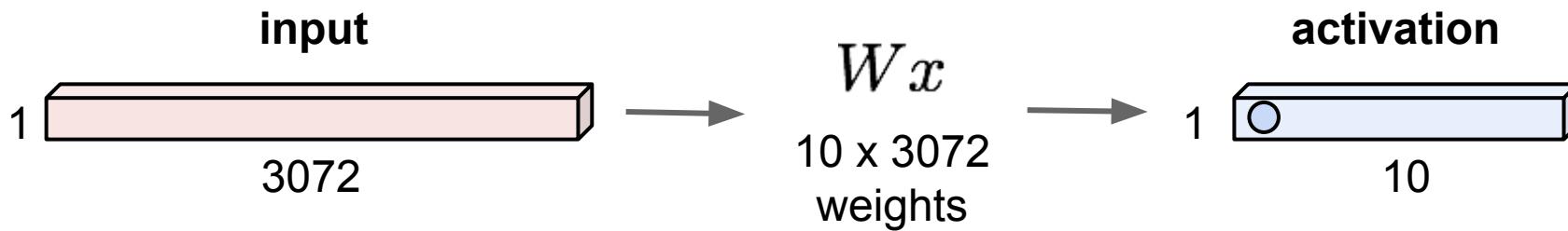
Stylized images copyright Justin Johnson, 2017;  
reproduced with permission

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

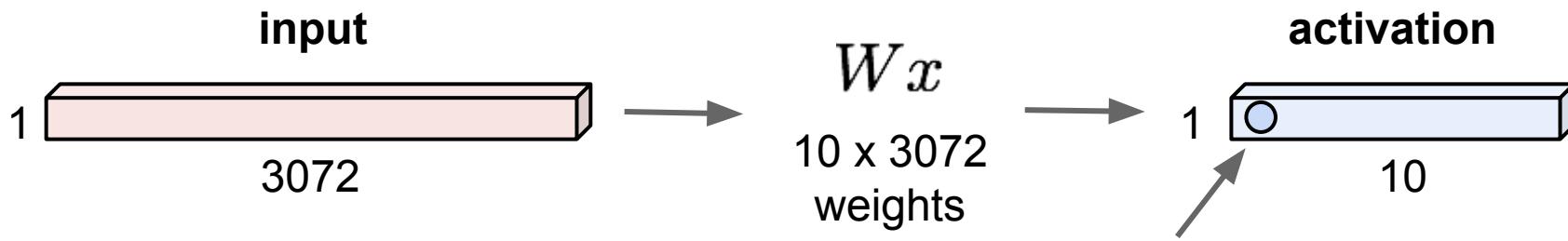
# Recap: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



# Fully Connected Layer

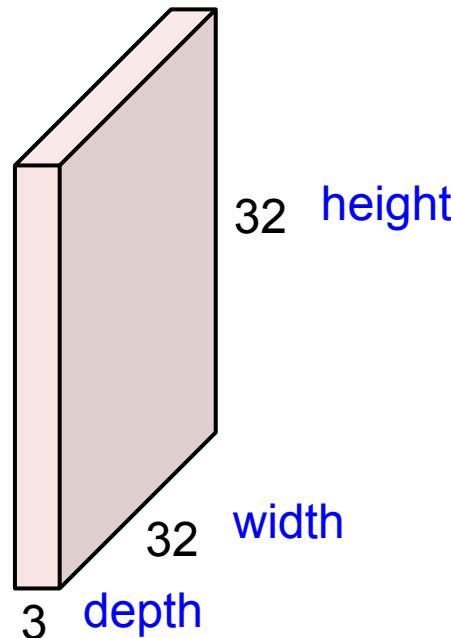
32x32x3 image -> stretch to  $3072 \times 1$



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

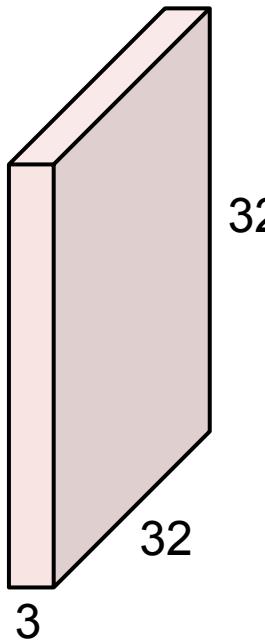
# Convolution Layer

32x32x3 image -> preserve spatial structure

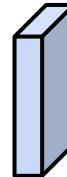


# Convolution Layer

32x32x3 image



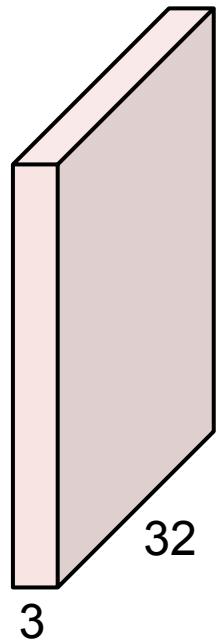
5x5x3 filter



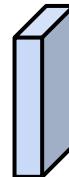
**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

# Convolution Layer

32x32x3 image



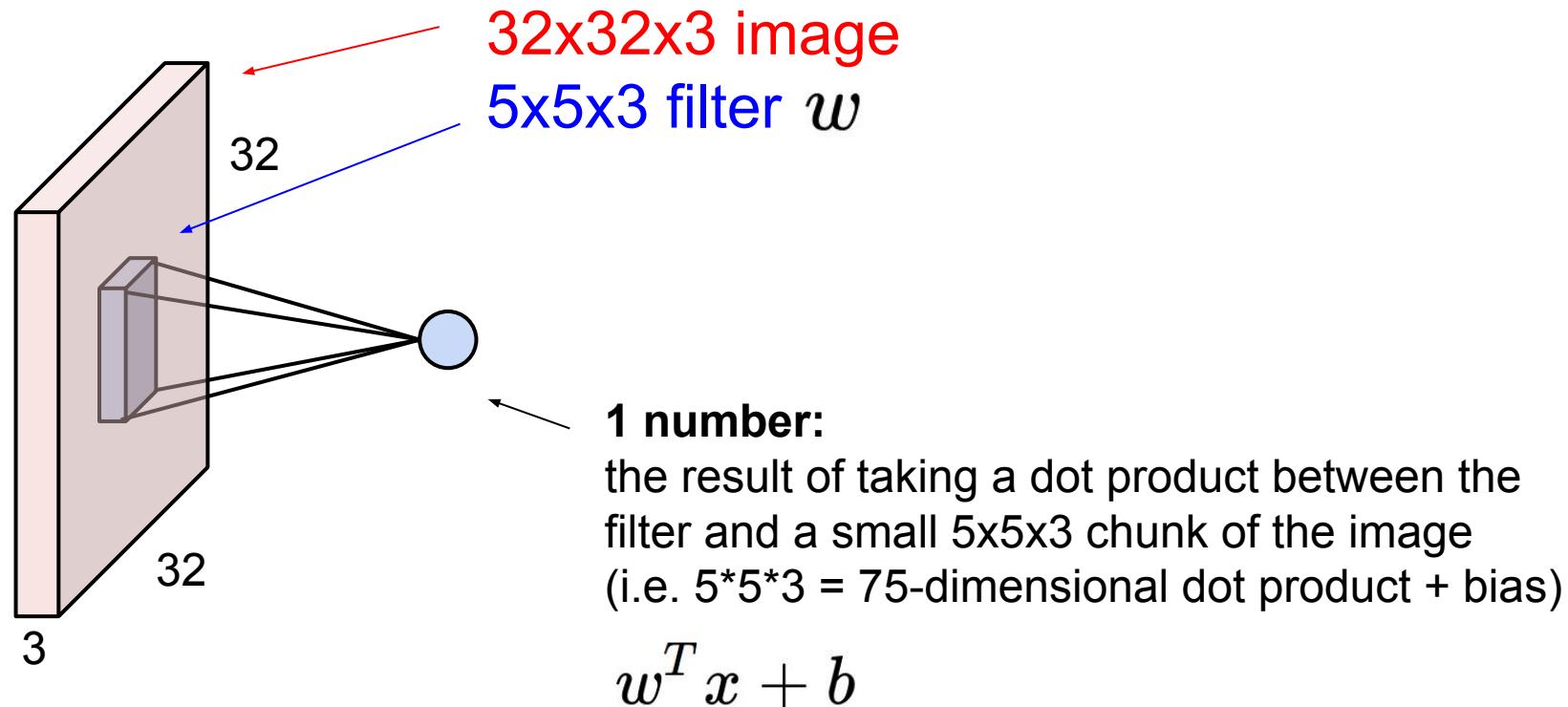
5x5x3 filter



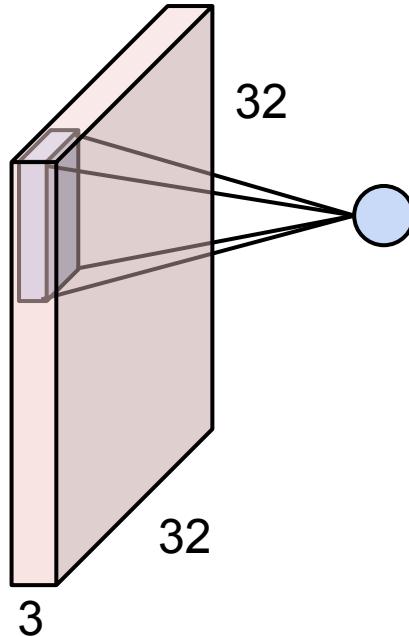
Filters always extend the full depth of the input volume

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

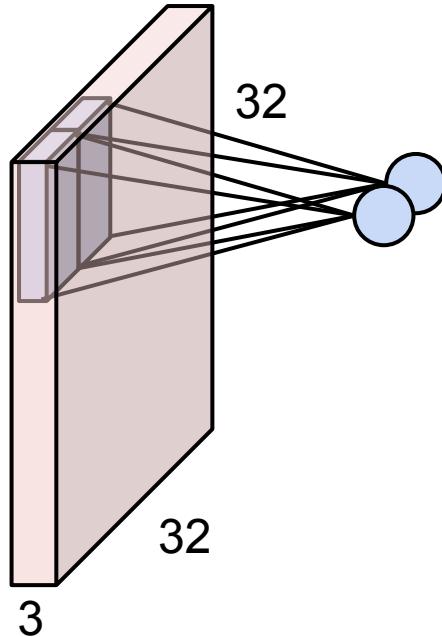
# Convolution Layer



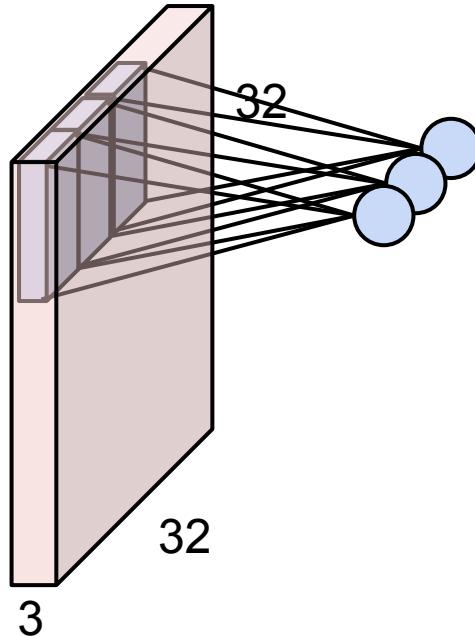
# Convolution Layer



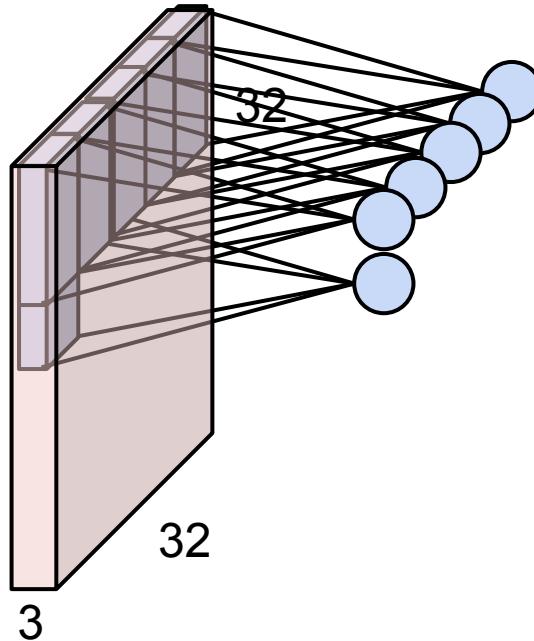
# Convolution Layer



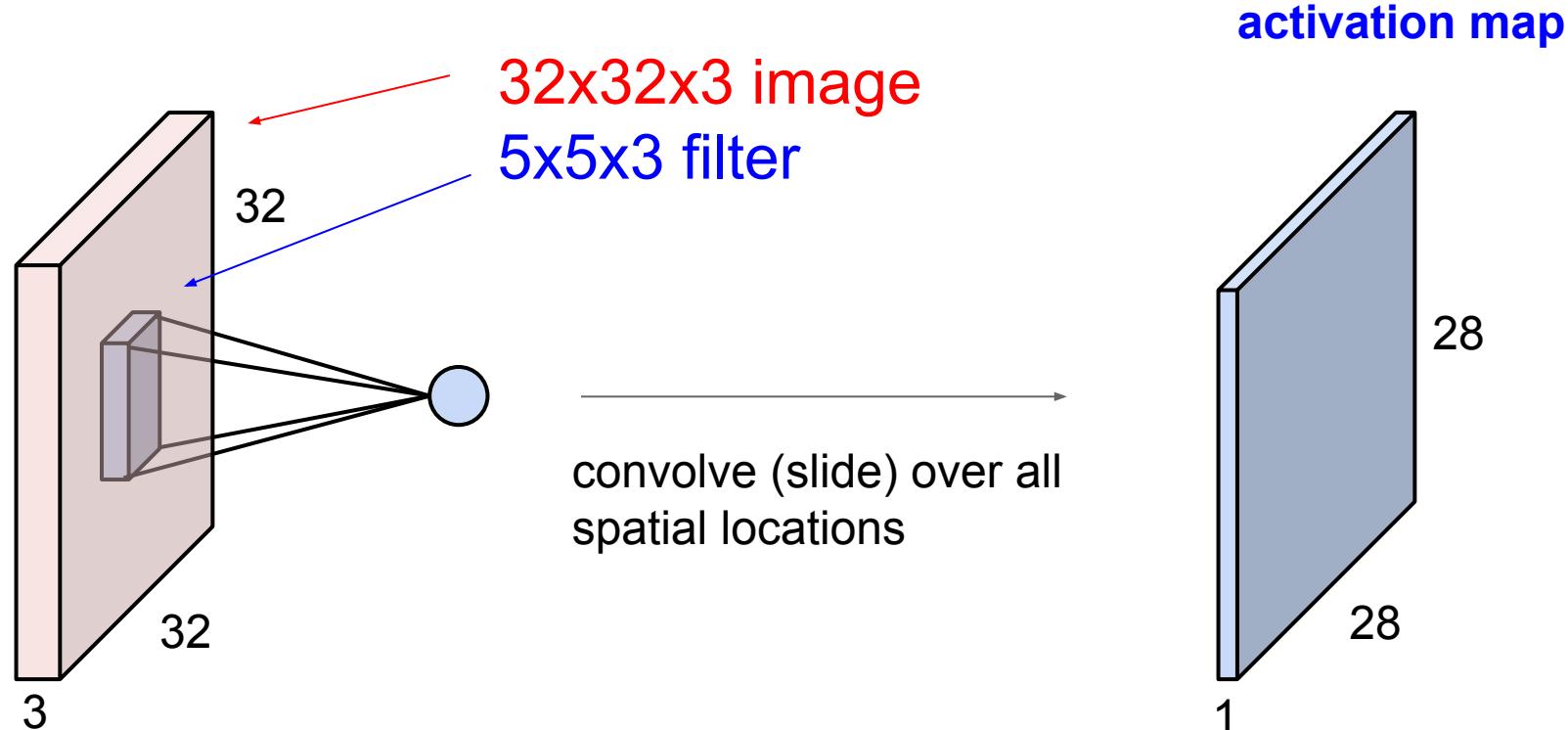
# Convolution Layer



# Convolution Layer

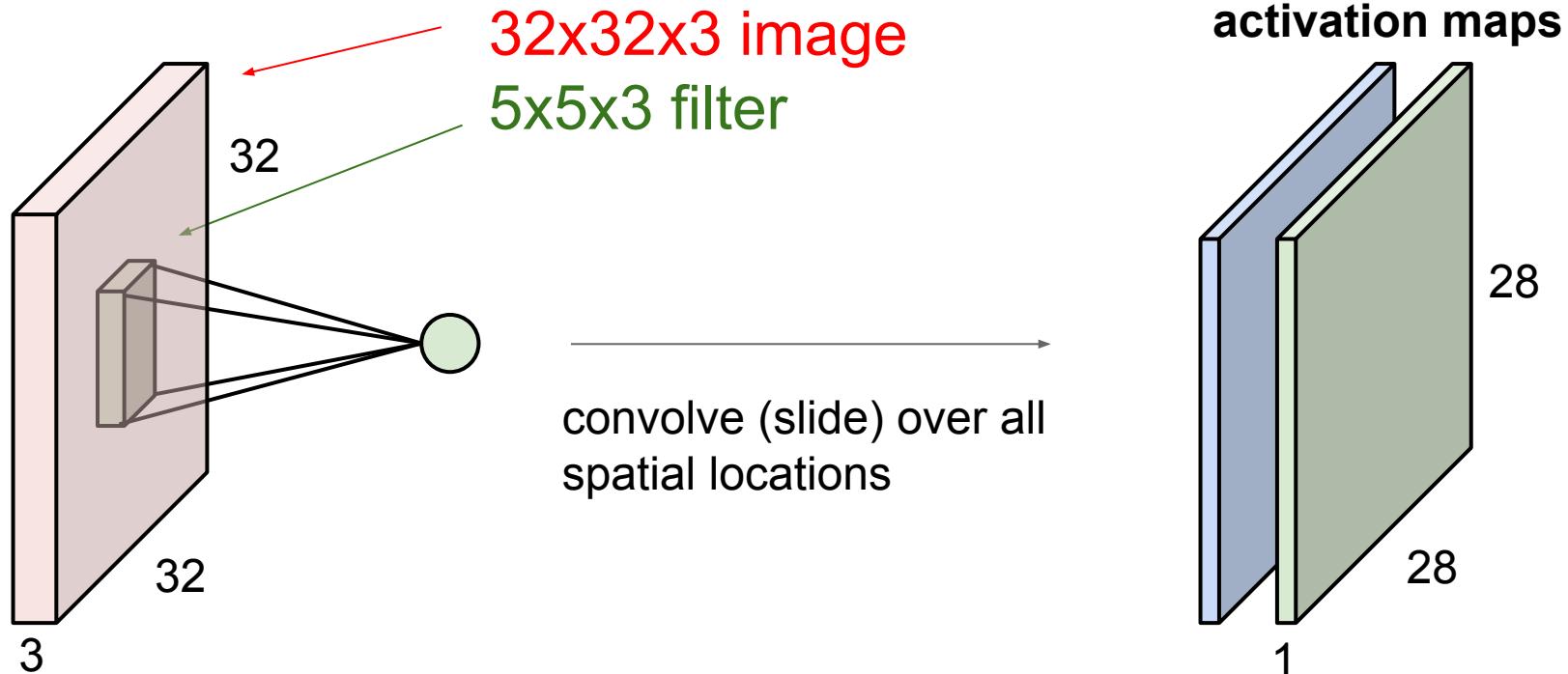


# Convolution Layer

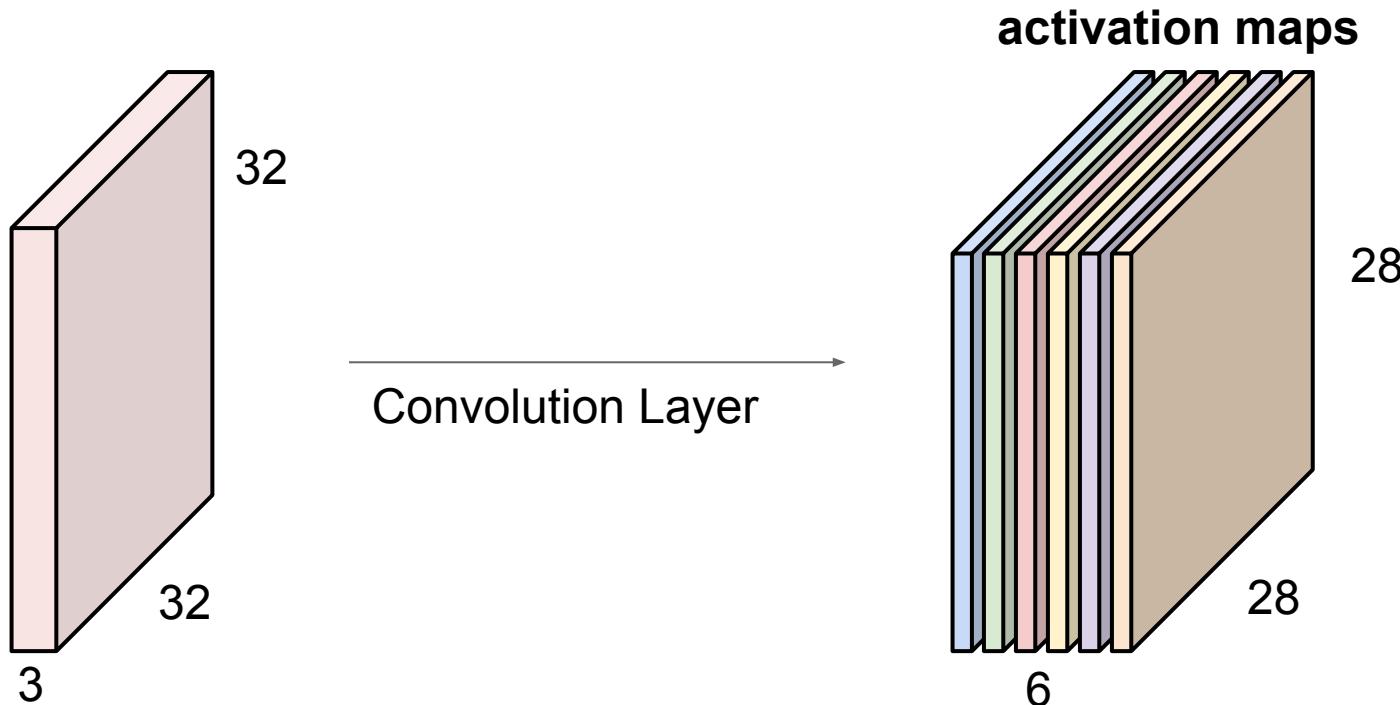


# Convolution Layer

consider a second, green filter

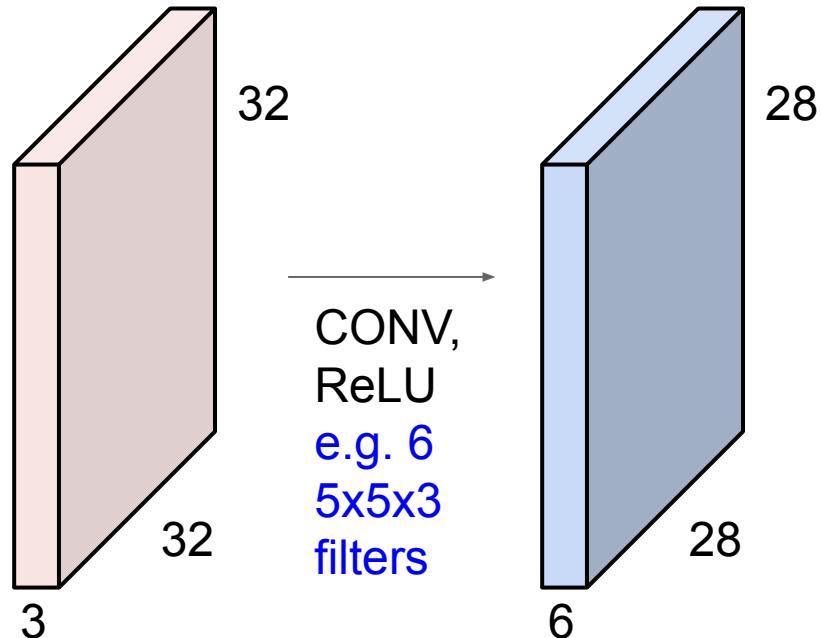


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

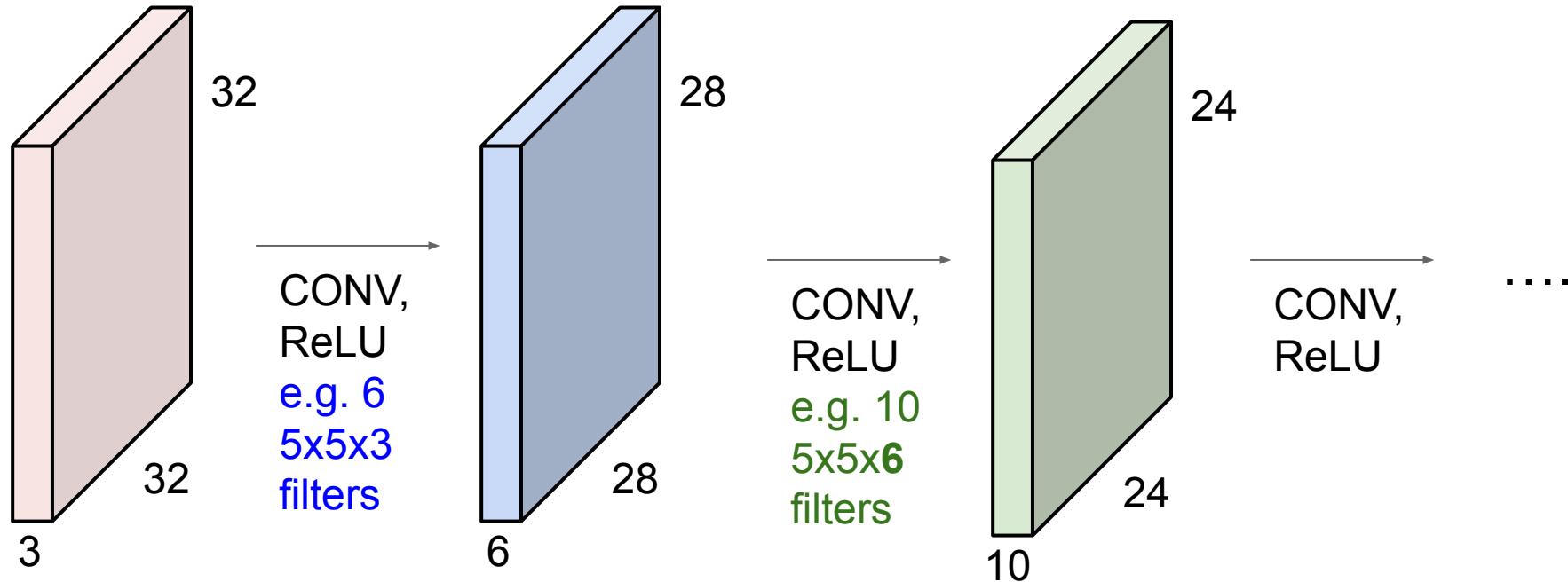


We stack these up to get a “new image” of size  $28 \times 28 \times 6$ !

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



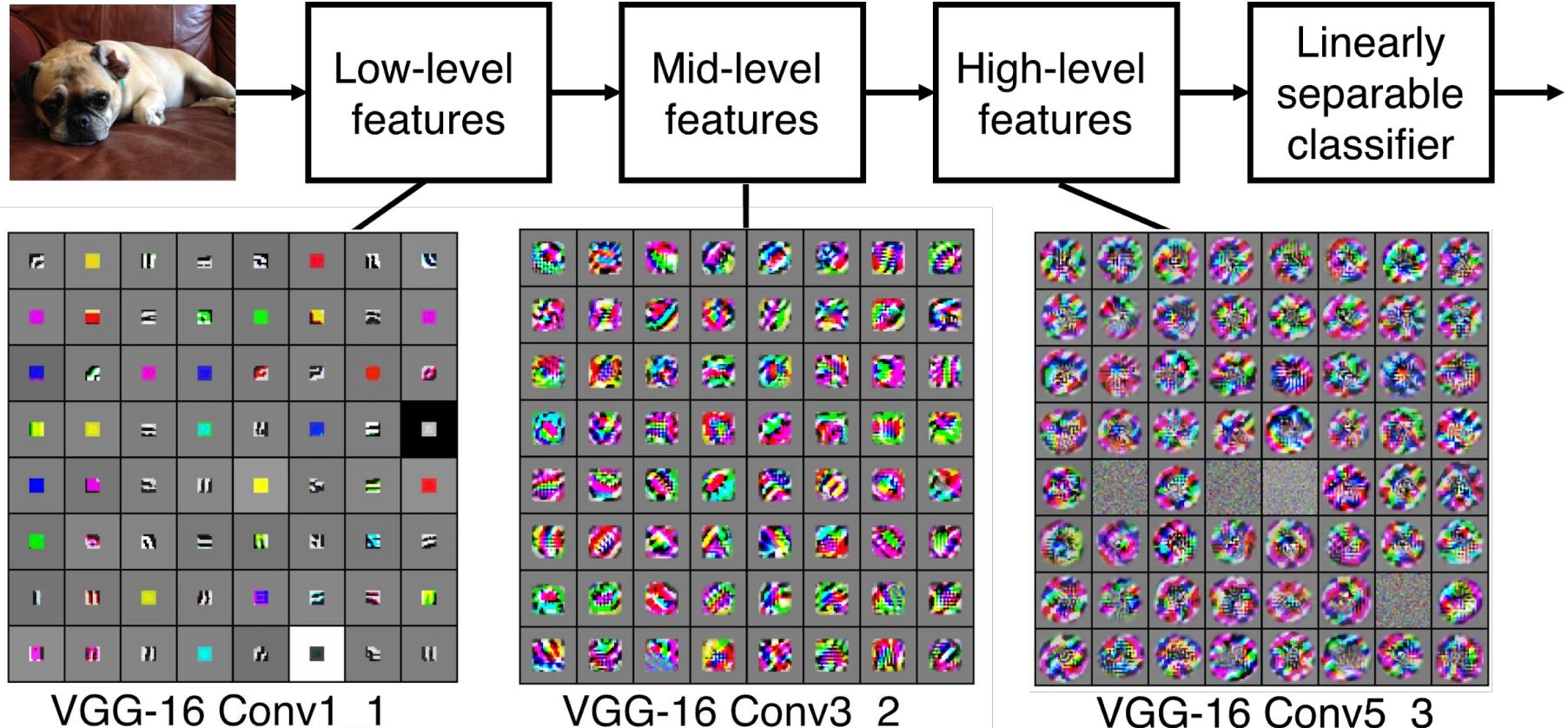
**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions



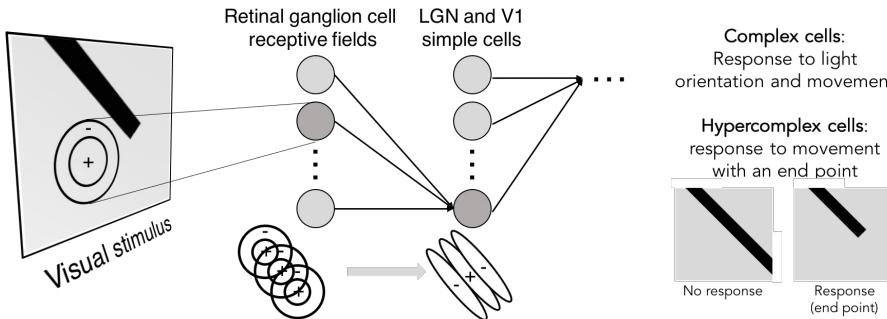
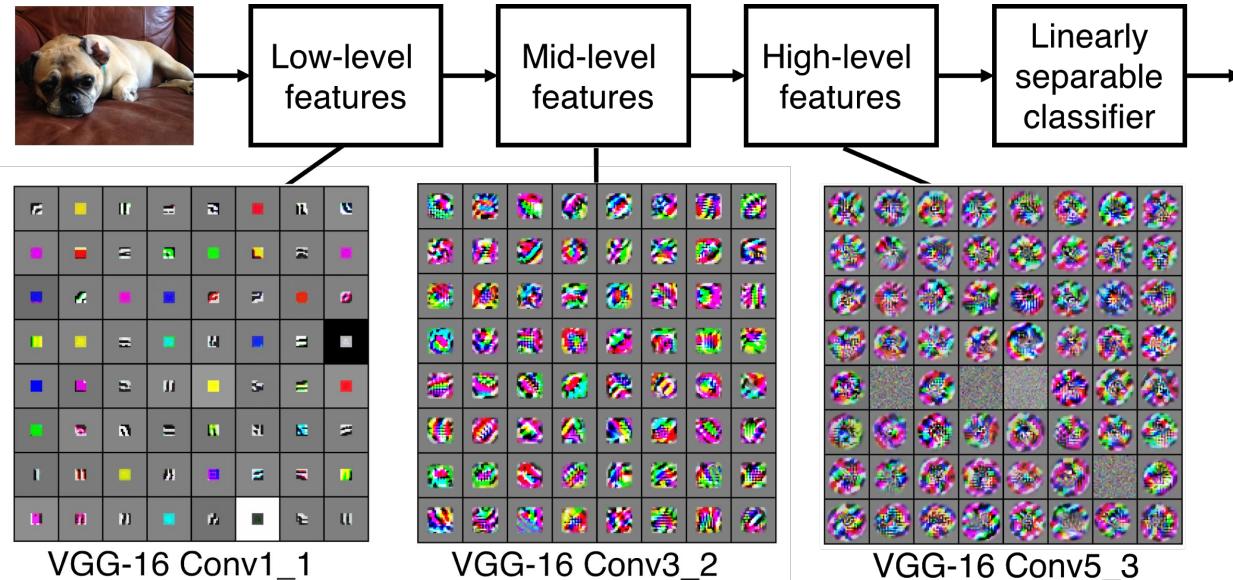
## Preview

[Zeiler and Fergus 2013]

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

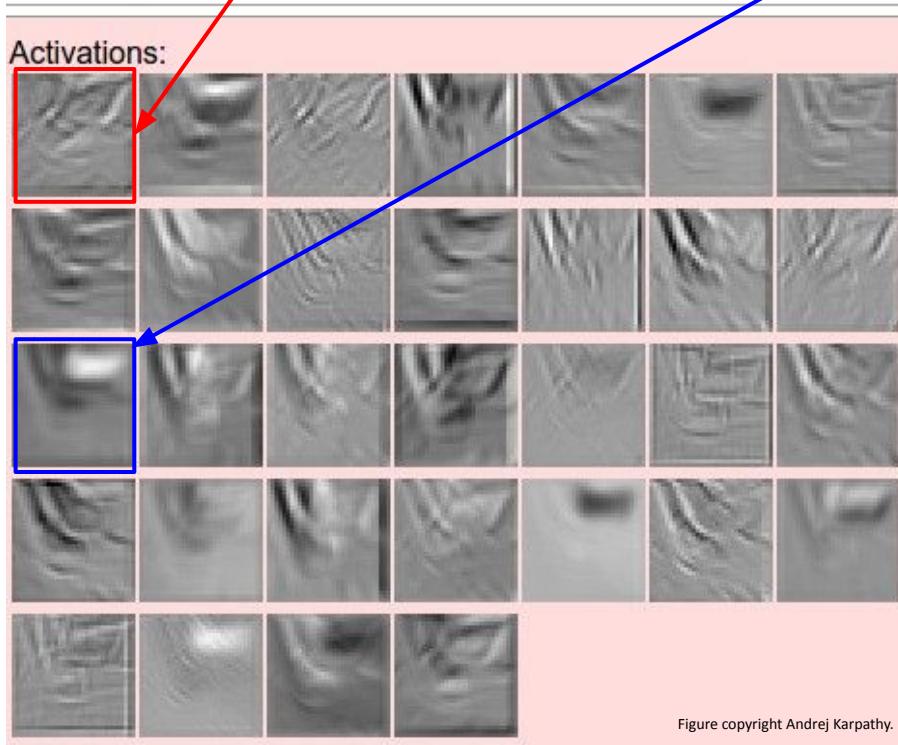


## Preview





one filter =>  
one activation map



example 5x5 filters  
(32 total)

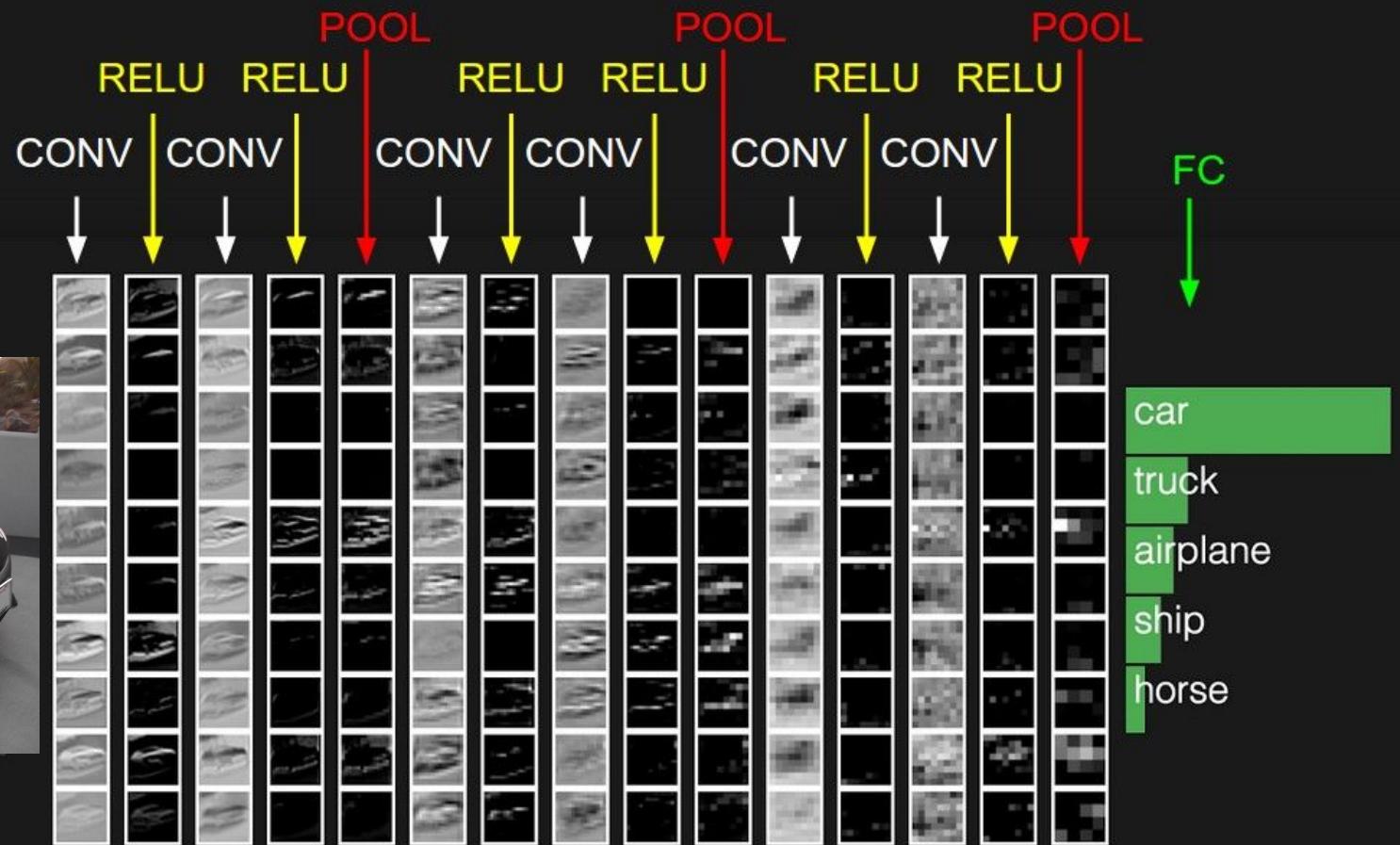
We call the layer convolutional  
because it is related to convolution  
of two signals:

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

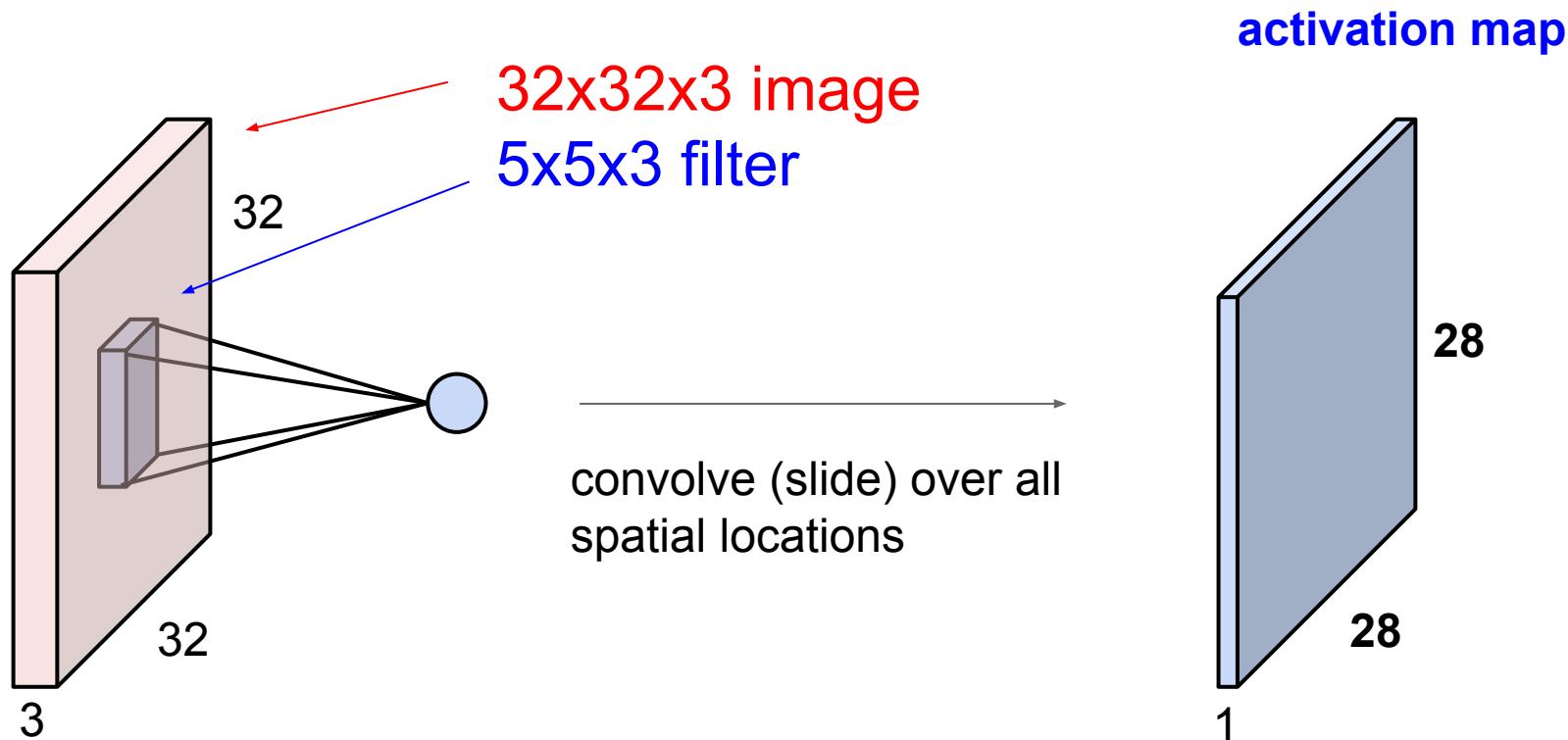


elementwise multiplication and sum of  
a filter and the signal (image)

preview:

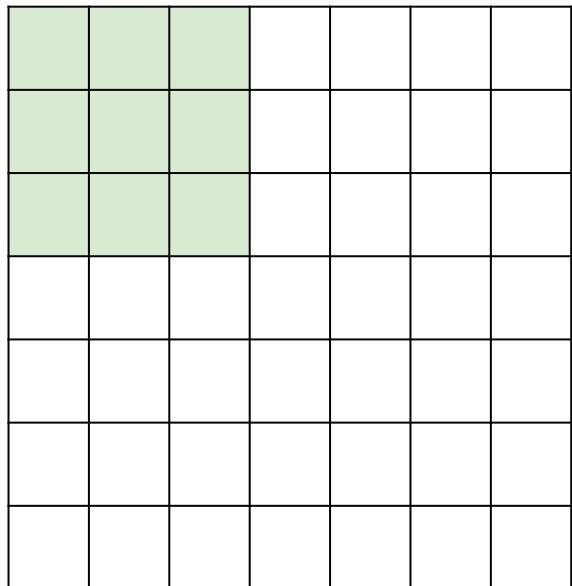


## A closer look at spatial dimensions:



## A closer look at spatial dimensions:

7

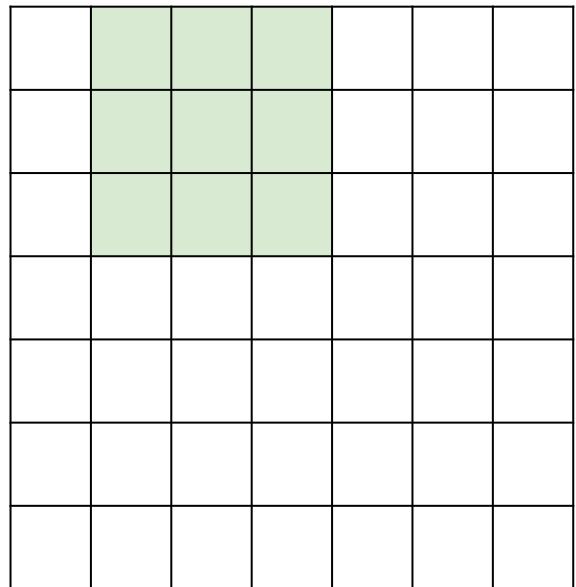


7x7 input (spatially)  
assume 3x3 filter

7

## A closer look at spatial dimensions:

7

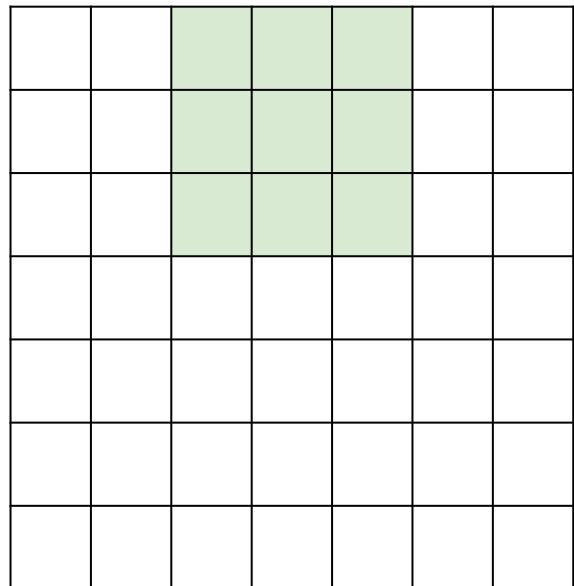


7x7 input (spatially)  
assume 3x3 filter

7

## A closer look at spatial dimensions:

7

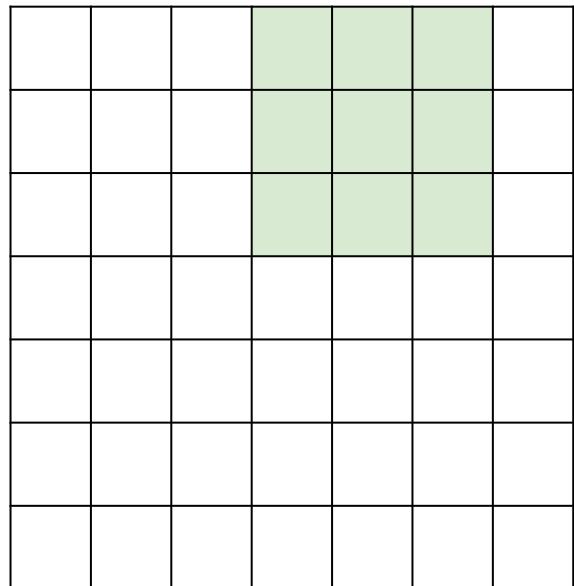


7x7 input (spatially)  
assume 3x3 filter

7

## A closer look at spatial dimensions:

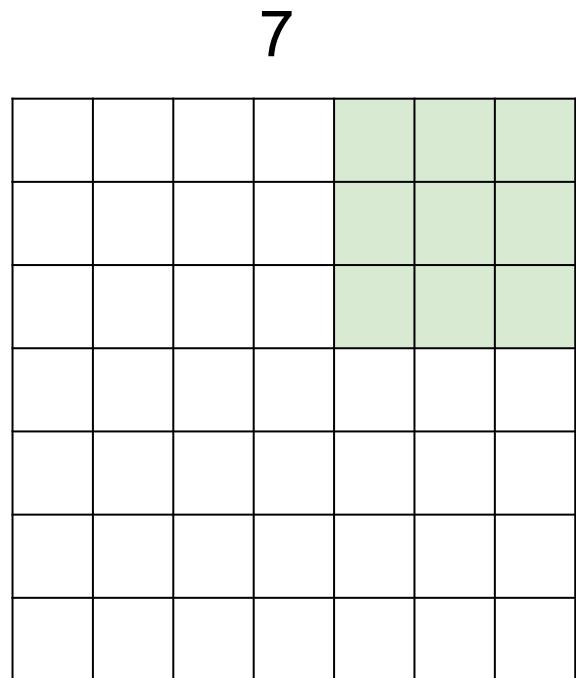
7



7x7 input (spatially)  
assume 3x3 filter

7

## A closer look at spatial dimensions:

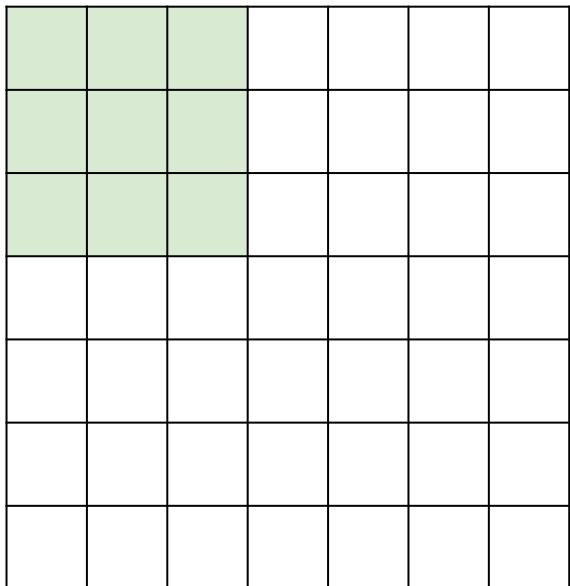


7x7 input (spatially)  
assume 3x3 filter

**=> 5x5 output**

## A closer look at spatial dimensions:

7

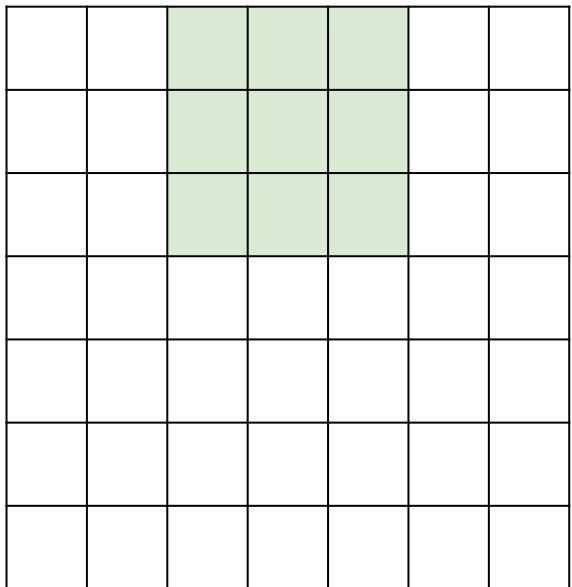


7

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

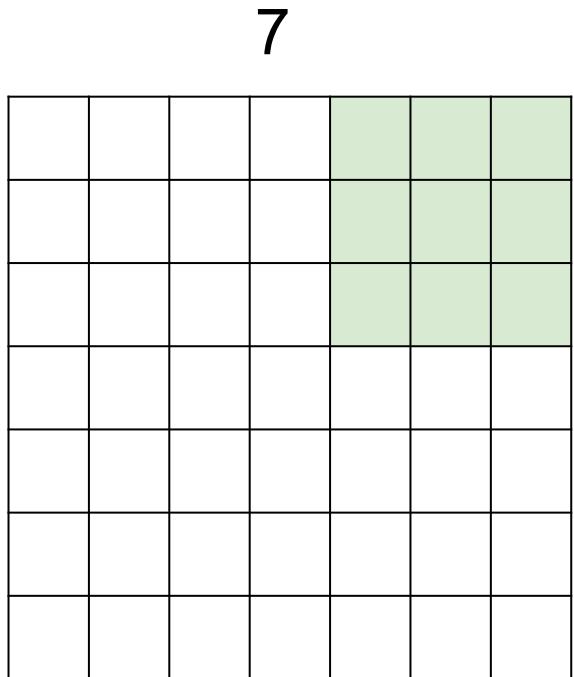
## A closer look at spatial dimensions:

7



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

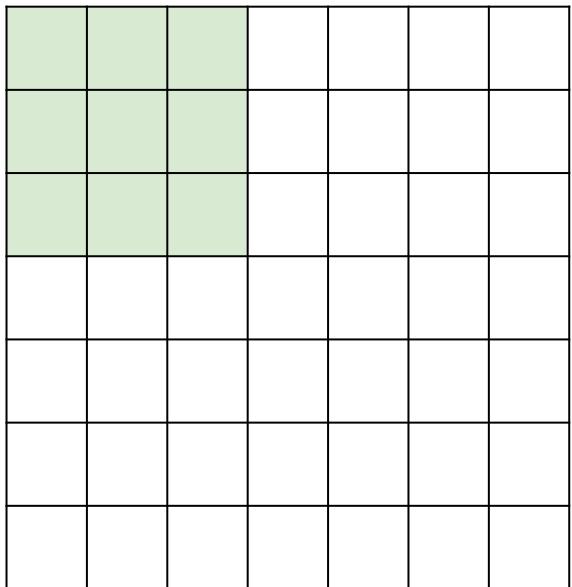
## A closer look at spatial dimensions:



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

## A closer look at spatial dimensions:

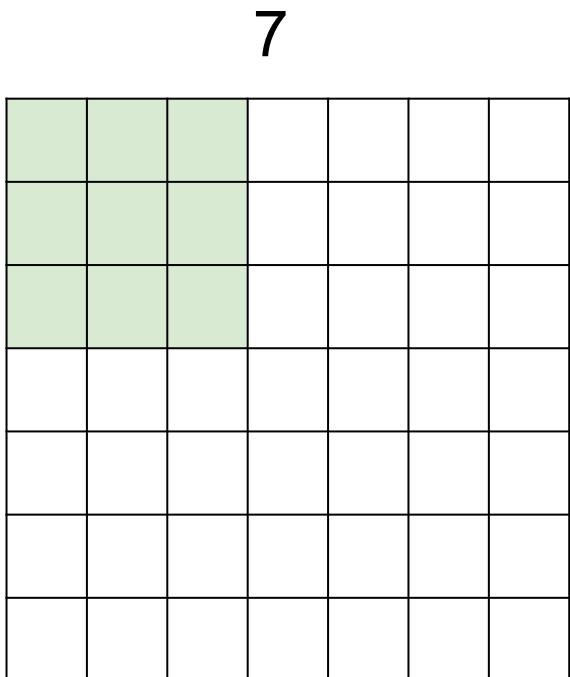
7



7

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

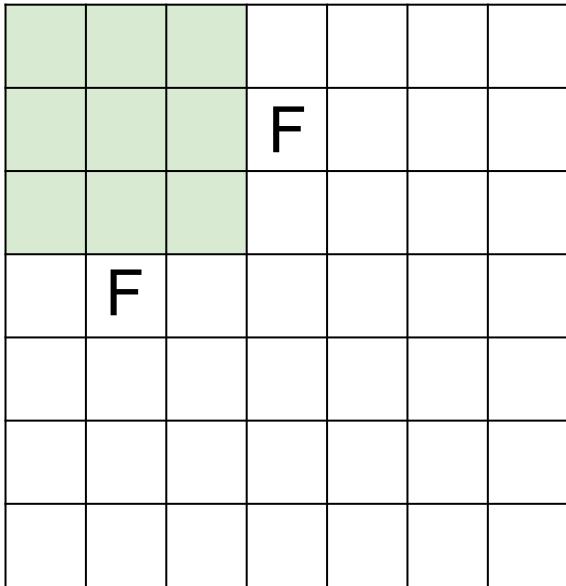
## A closer look at spatial dimensions:



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) / 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!**

(recall:)

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

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

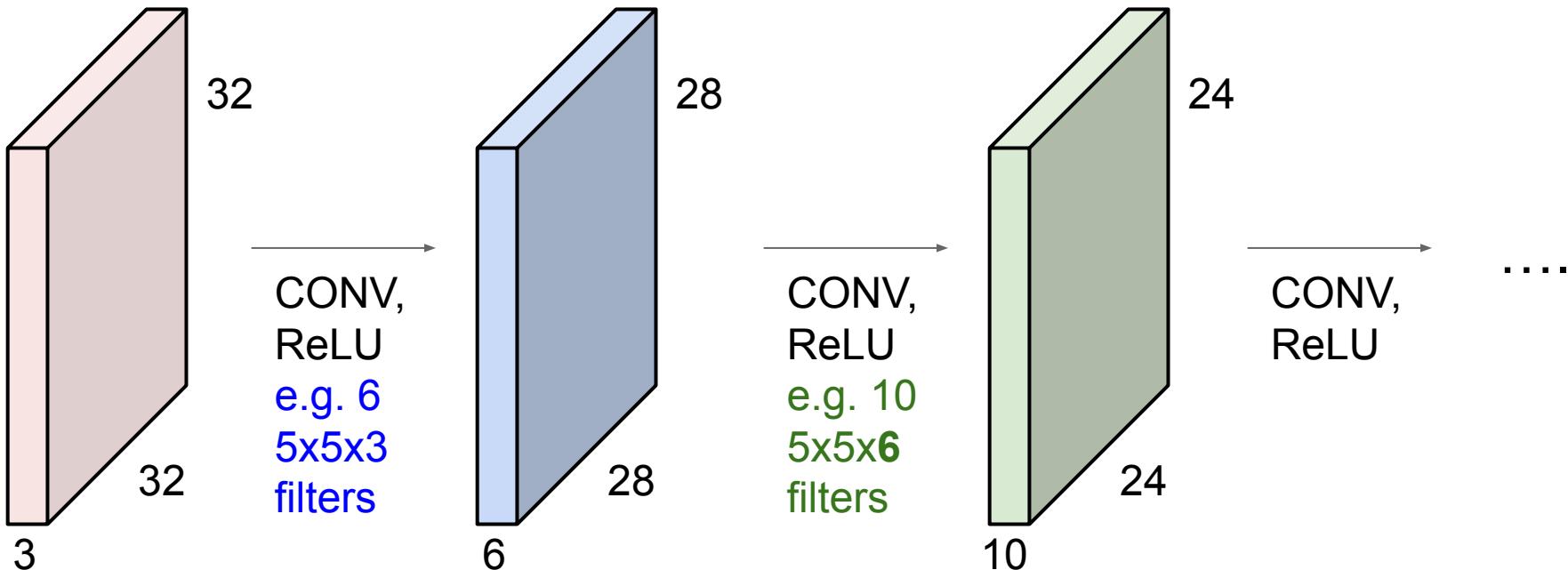
e.g.  $F = 3 \Rightarrow$  zero pad with 1

$F = 5 \Rightarrow$  zero pad with 2

$F = 7 \Rightarrow$  zero pad with 3

## Remember back to...

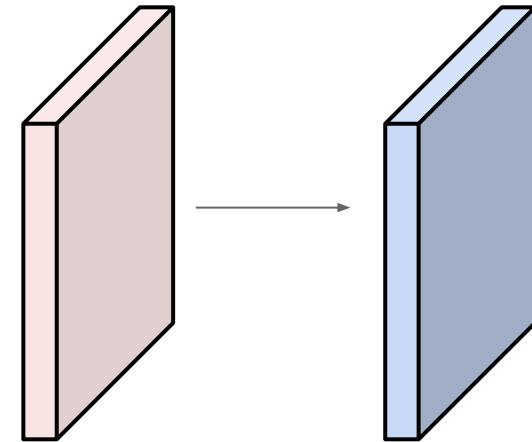
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!  
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

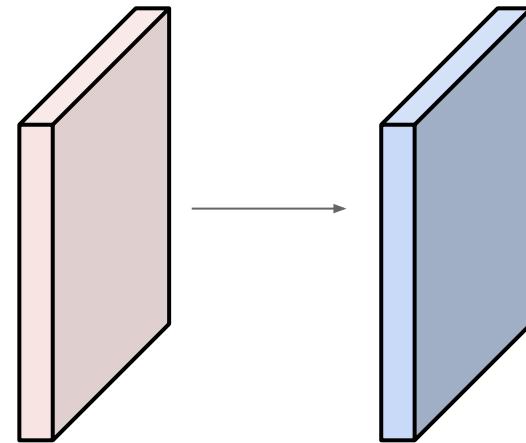


Output volume size: ?

Examples time:

Input volume: **32x32x3**

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



Output volume size:

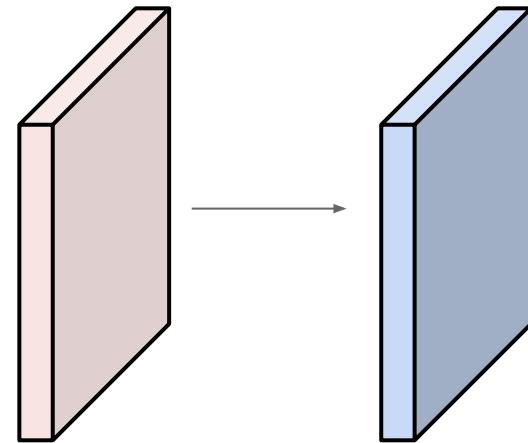
$(32+2*2-5)/1+1 = 32$  spatially, so

**32x32x10**

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

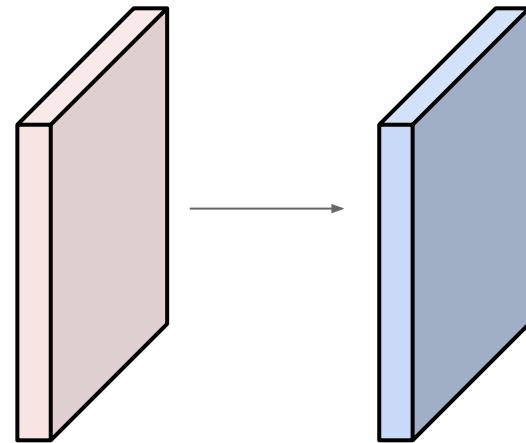


Number of parameters in this layer?

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)  
=>  $76*10 = 760$

# 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:

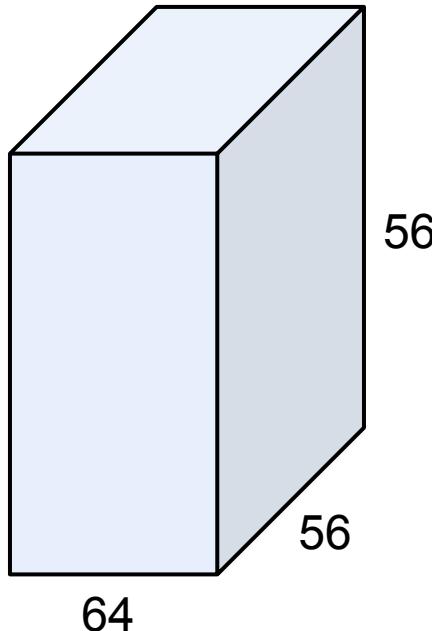
- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

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)
  - $F = 1, S = 1, P = 0$

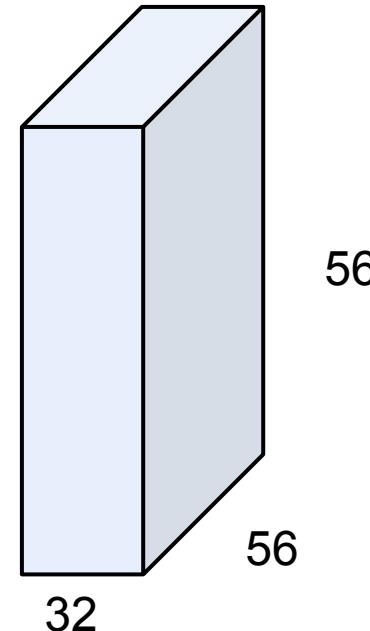
(btw, 1x1 convolution layers make perfect sense)



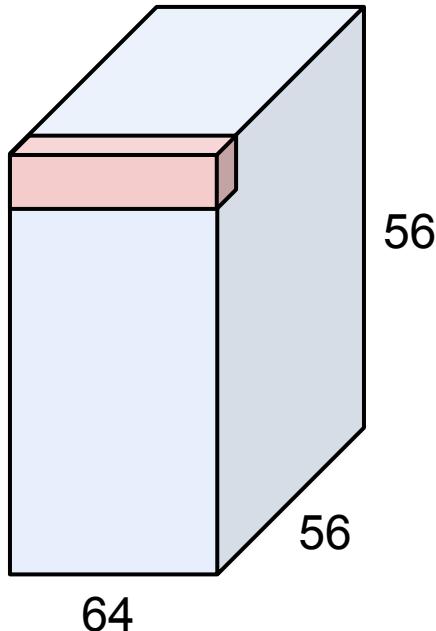
1x1 CONV  
with 32 filters

---

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



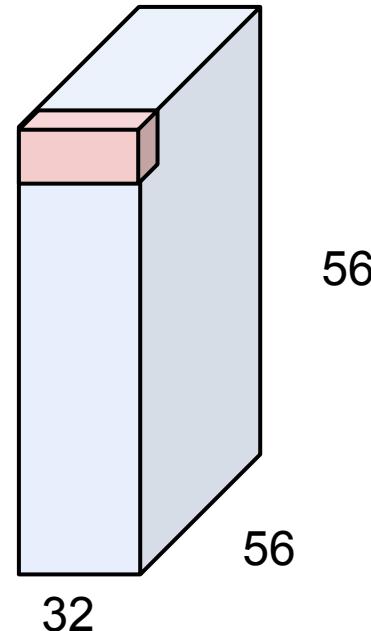
(btw, 1x1 convolution layers make perfect sense)



1x1 CONV  
with 32 filters

→

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



# Example: CONV layer in PyTorch

## Conv2d

```
CLASS torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,  
dilation=1, groups=1, bias=True)
```

[SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size  $(N, C_{\text{in}}, H, W)$  and output  $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$  can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) * \text{input}(N_i, k)$$

where  $*$  is the valid 2D cross-correlation operator,  $N$  is a batch size,  $C$  denotes a number of channels,  $H$  is a height of input planes in pixels, and  $W$  is width in pixels.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this [link](#) has a nice visualization of what `dilation` does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be divisible by `groups`. For example,
  - At `groups=1`, all inputs are convolved to all outputs.
  - At `groups=2`, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels, and producing half the output channels, and both subsequently concatenated.
  - At `groups= in_channels`, each input channel is convolved with its own set of filters, of size:  $\left\lfloor \frac{C_{\text{out}}}{C_{\text{in}}} \right\rfloor$ .

The parameters `kernel_size`, `stride`, `padding`, `dilation` can either be:

- a single `int` – in which case the same value is used for the height and width dimension
- a `tuple` of two `ints` – in which case, the first `int` is used for the height dimension, and the second `int` for the width dimension

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

# Example: CONV layer in Keras

Conv layer needs 4 hyperparameters:

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

## Conv2D

[source]

```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, d:
```

2D convolution layer (e.g. spatial convolution over images).

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

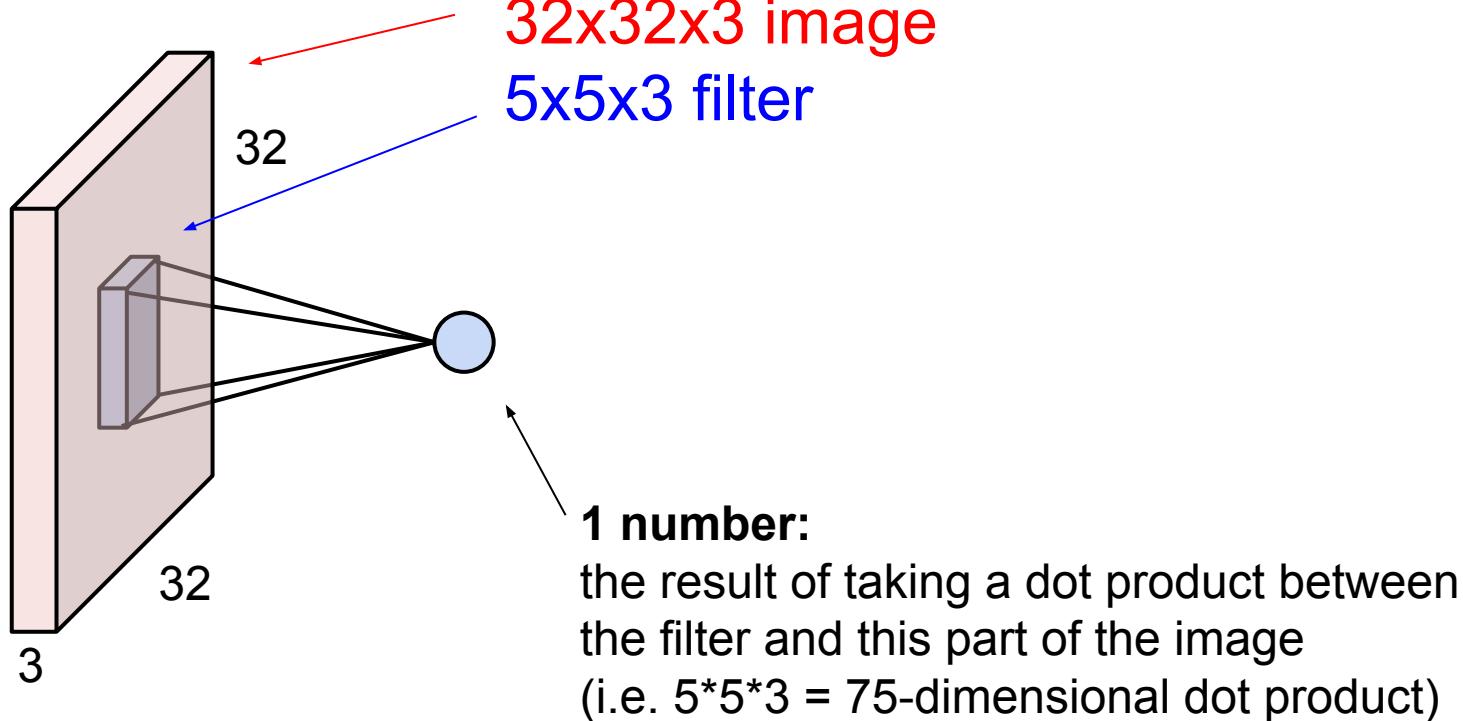
When using this layer as the first layer in a model, provide the keyword argument `input_shape` (tuple of integers, does not include the batch axis), e.g. `input_shape=(128, 128, 3)` for 128x128 RGB pictures in `data_format="channels_last"`.

### Arguments

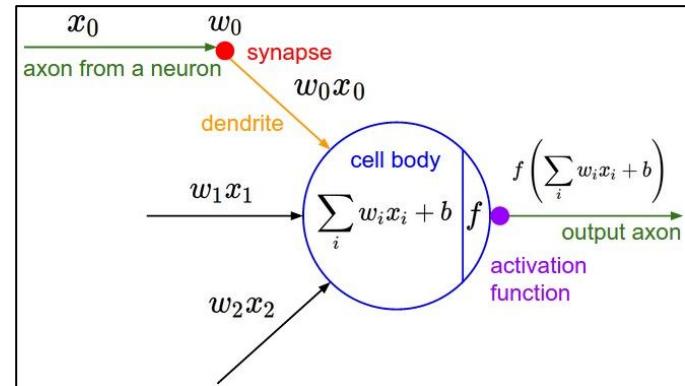
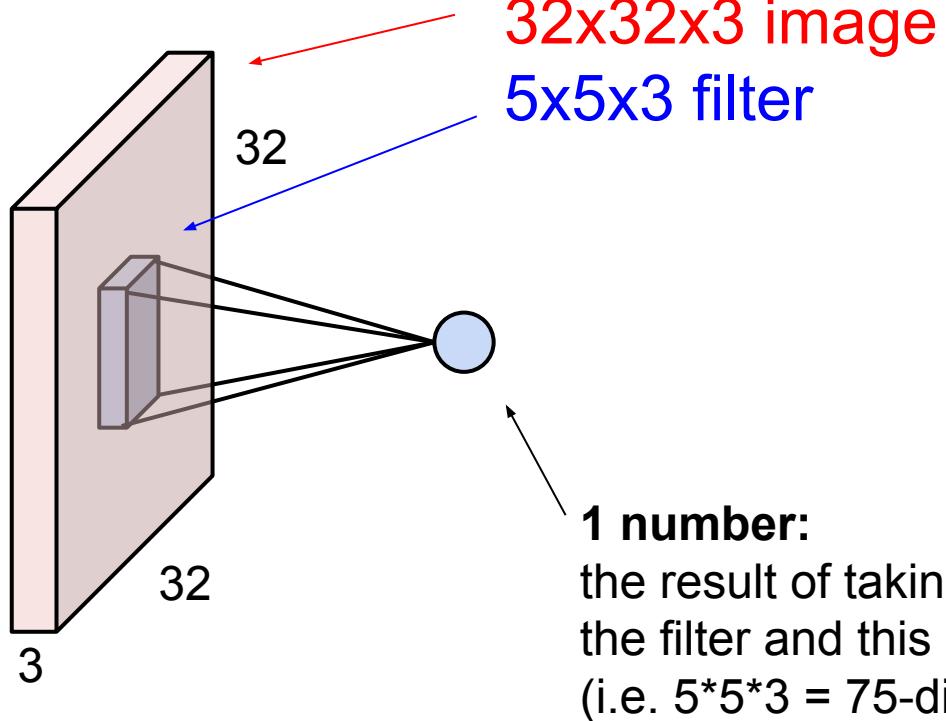
- **filters**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel\_size**: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides**: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- **padding**: one of `"valid"` or `"same"` (case-insensitive). Note that `"same"` is slightly inconsistent across backends with `strides` != 1, as described here
- **data\_format**: A string, one of `"channels_last"` or `"channels_first"`. The ordering of the dimensions in the inputs. `"channels_last"` corresponds to inputs with shape `(batch, height, width, channels)` while `"channels_first"` corresponds to inputs with shape `(batch, channels, height, width)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be `"channels_last"`.

Keras is licensed under the [MIT license](#).

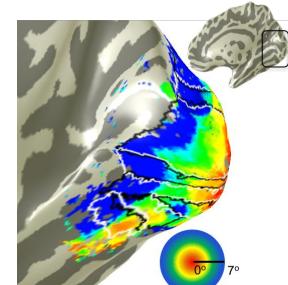
# The brain/neuron view of CONV Layer



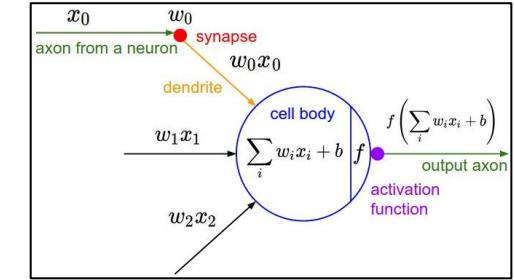
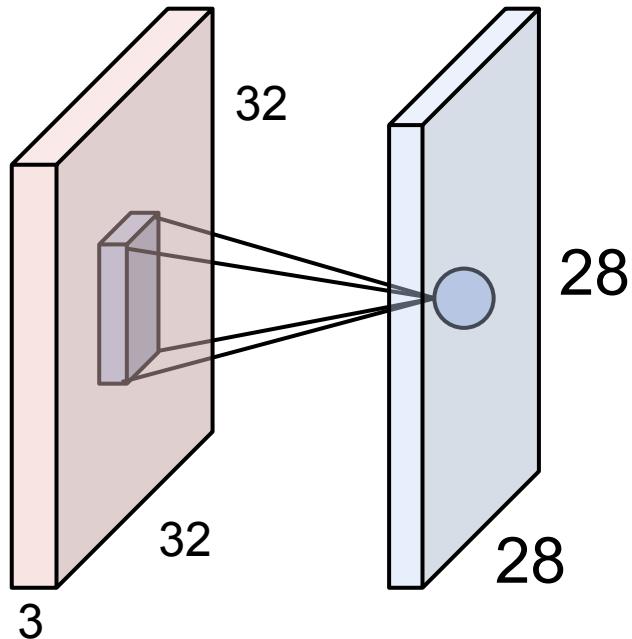
# The brain/neuron view of CONV Layer



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



# Receptive field

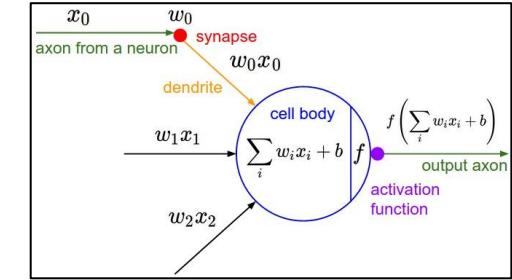
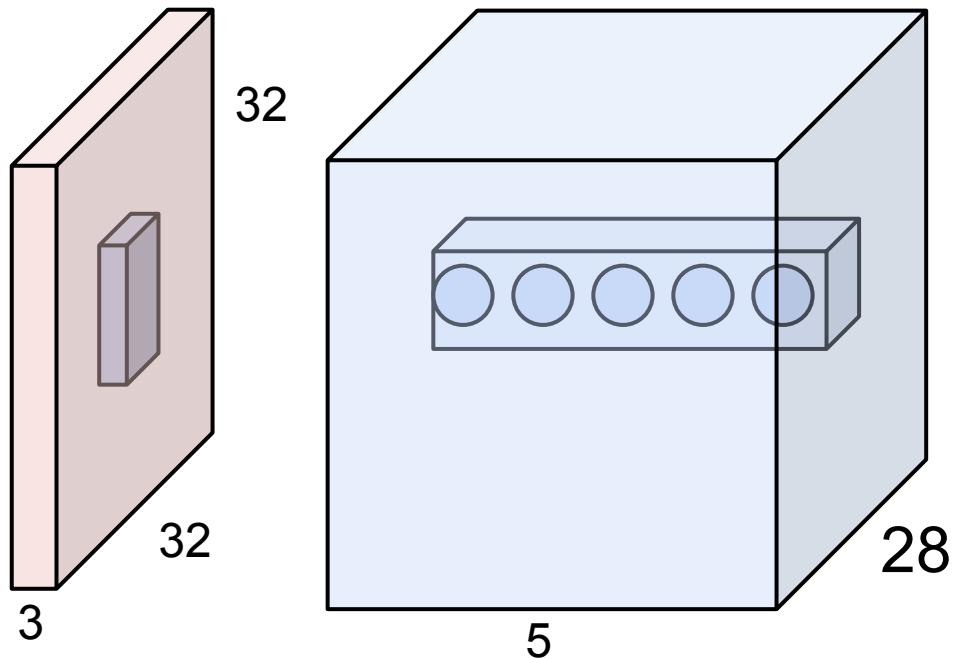


An activation map is a 28x28 sheet of neuron outputs:

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



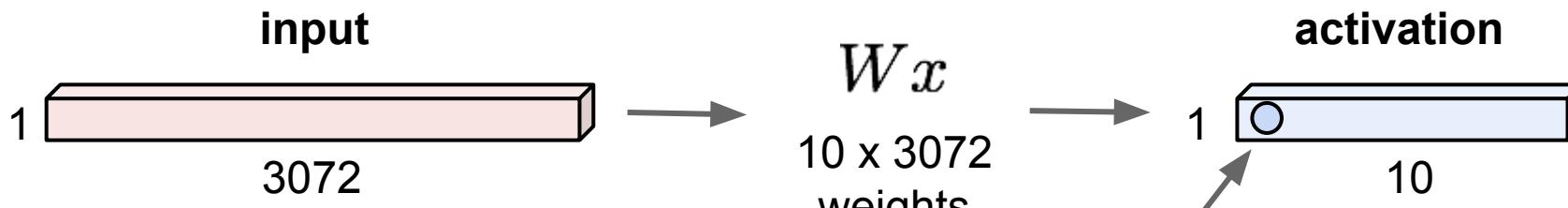
E.g. with 5 filters,  
CONV layer consists of  
neurons arranged in a 3D grid  
( $28 \times 28 \times 5$ )

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

# Reminder: Fully Connected Layer

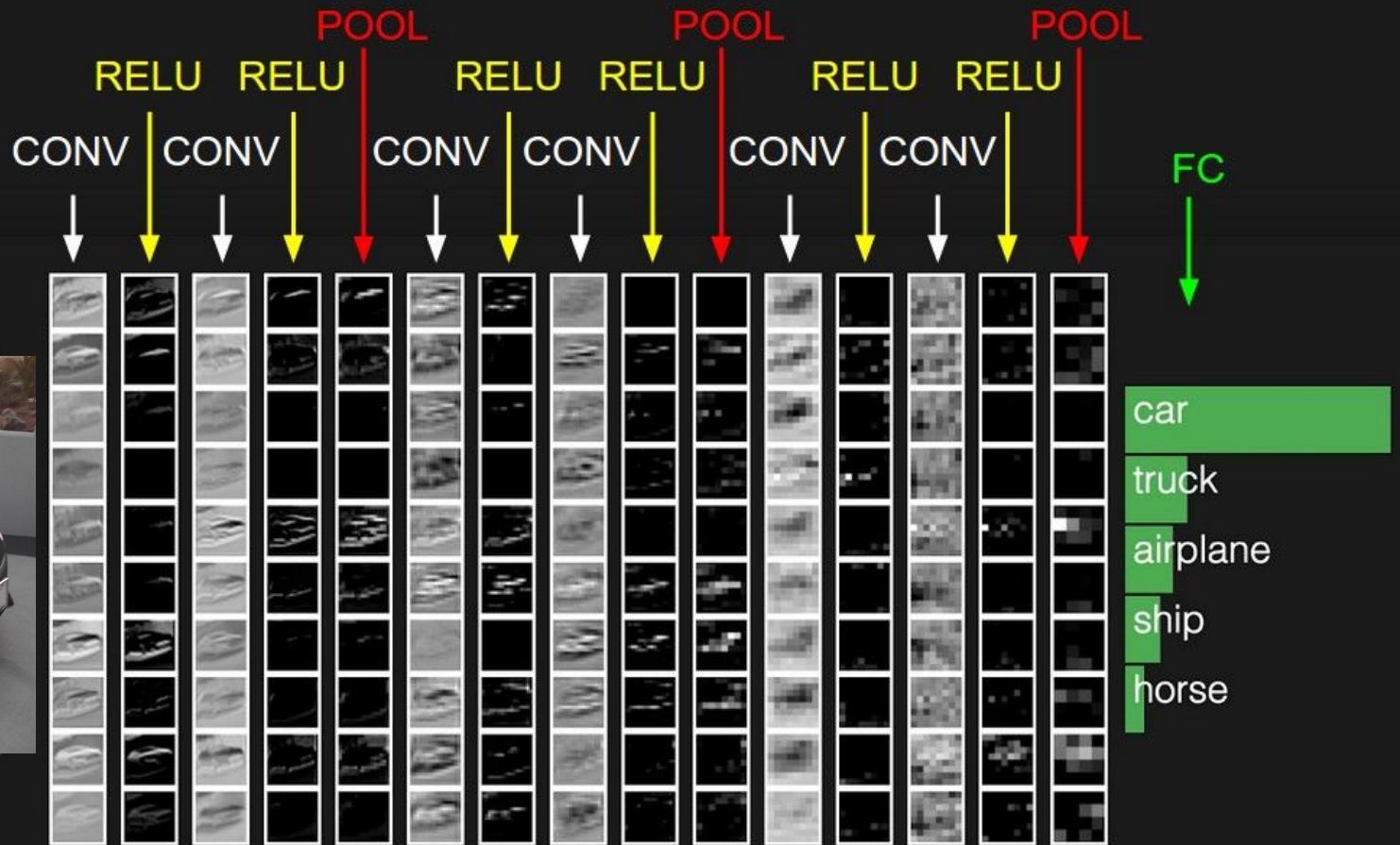
32x32x3 image -> stretch to  $3072 \times 1$

Each neuron  
looks at the full  
input volume



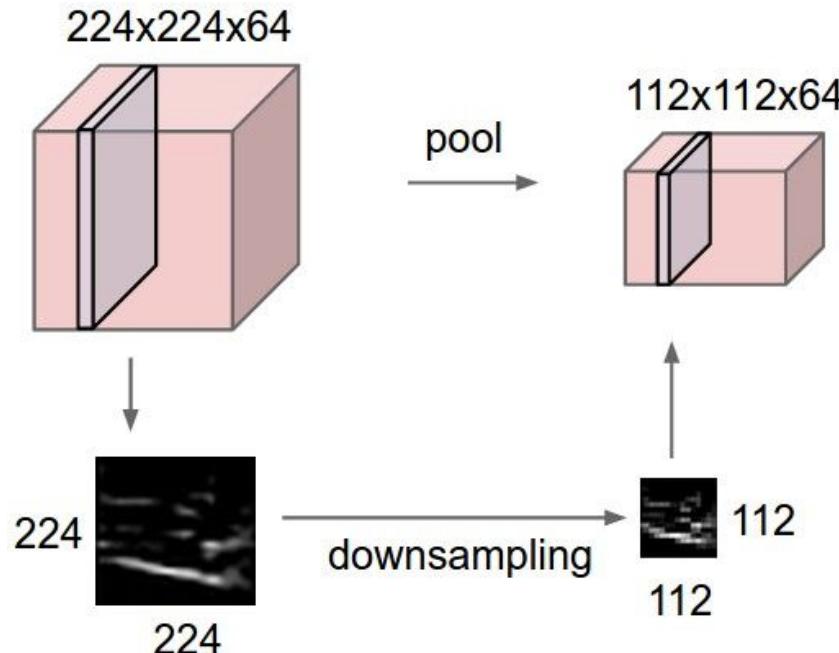
**1 number:**  
the result of taking a dot product  
between a row of  $W$  and the input  
(a 3072-dimensional dot product)

two more layers to go: POOL/FC

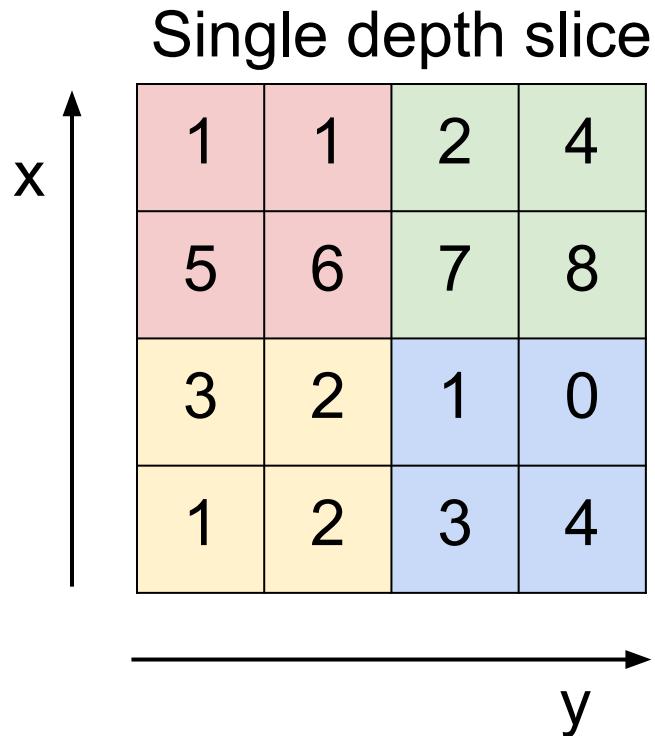


# Pooling layer

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



# MAX POOLING



max pool with 2x2 filters  
and stride 2

6	8
3	4

# Pooling layer: summary

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

Conv layer needs 2 hyperparameters:

- The spatial extent **F**
- The stride **S**

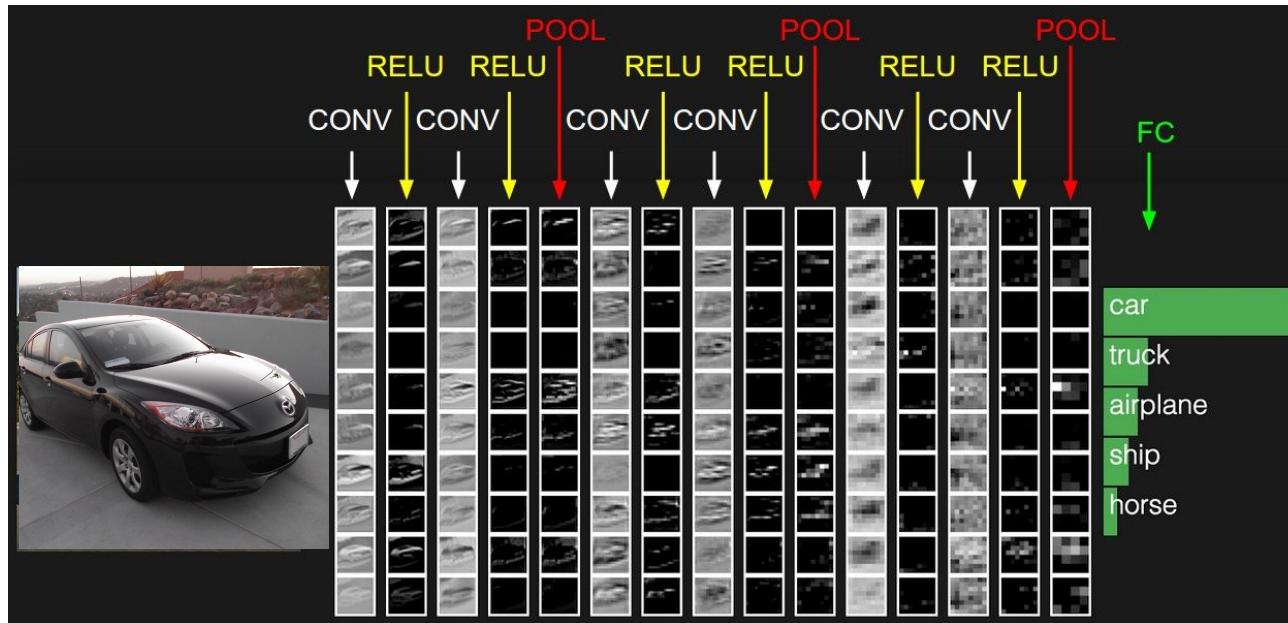
This will produce an output of  $W_2 \times H_2 \times C$  where:

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

Number of parameters: 0

# 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

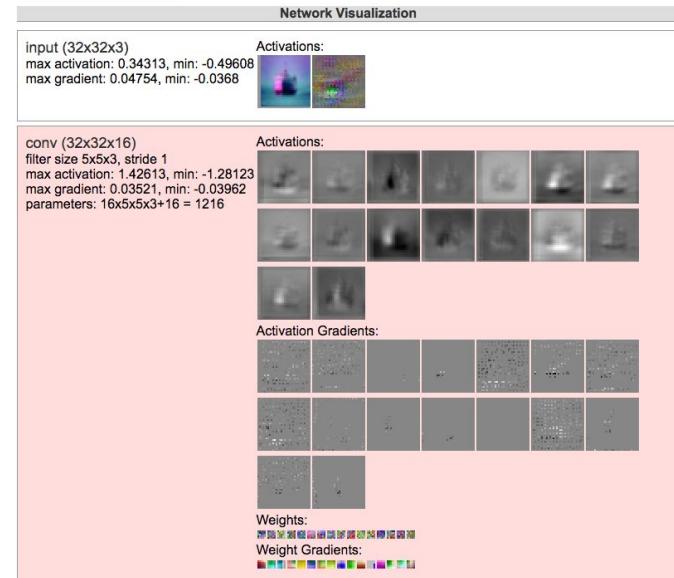
### Description

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)
- Historically architectures looked like  
 **$[(CONV-RELU)^*N-POOL?]^*M-(FC-RELU)^*K, SOFTMAX$**   
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