

# Lecture 5: Image Classification with CNNs

Dr. José Ramón Iglesias

DSP-ASIC BUILDER GROUP

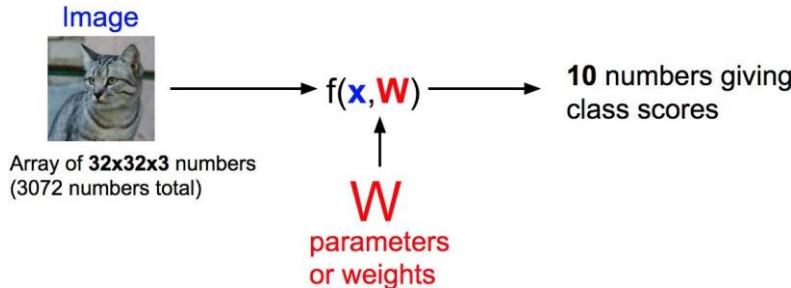
Director Semillero TRIAC

Ingenieria Electronica

Universidad Popular del Cesar

Image Classification with CNNs

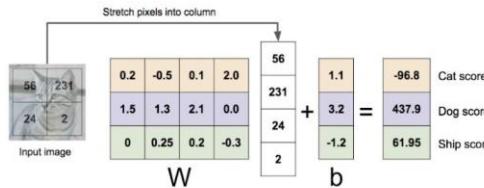
# Recap: Image Classification with Linear Classifier



$$f(x, W) = Wx + b$$

## Algebraic Viewpoint

$$f(x, W) = Wx$$



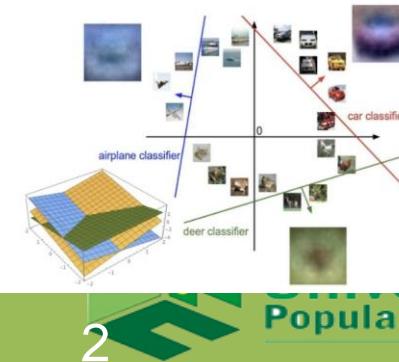
## Visual Viewpoint

One template per class



## Geometric Viewpoint

Hyperplanes cutting up space



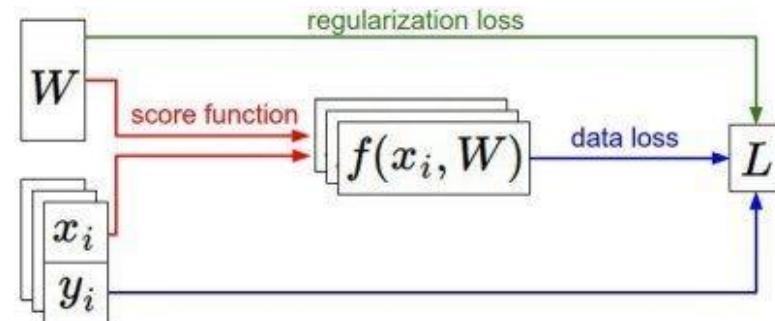
# Recap: Loss Function

- We have some dataset of  $(x, y)$
- We have a **score function**:  $s = f(x; W) = Wx$
- We have a **loss function**:

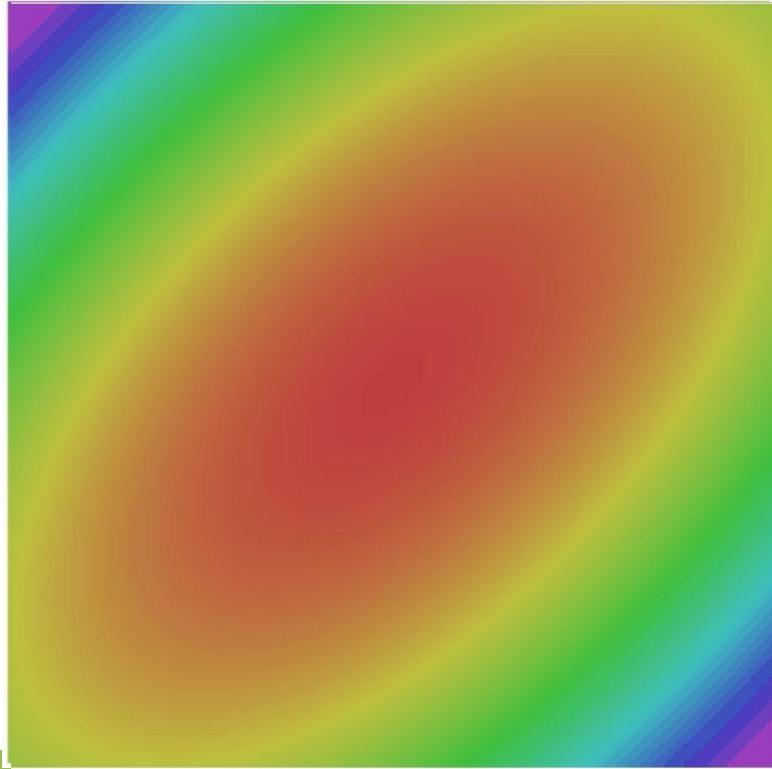
$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{sj}}\right) \text{ Softmax}$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \text{ SVM}$$

$$L = \frac{1}{N} \sum_{i=1}^N L_i + R(W) \text{ Full loss}$$



# Recap: Optimization



- SGD
- SGD+Momentum
- RMSProp
- Adam

Image Classification with CNNs

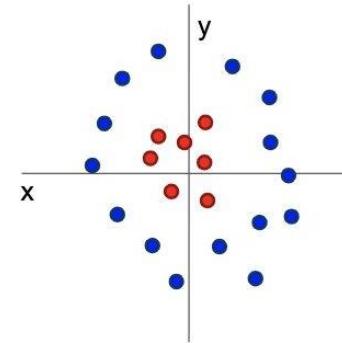
# Problem: Linear Classifiers are not very powerful

## Visual Viewpoint



Linear classifiers learn  
one template per class

## Geometric Viewpoint



Linear classifiers  
can only draw linear  
decision boundaries

# Last time: Neural Networks

Linear score function:

$$f = Wx$$

2-layer Neural Network

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

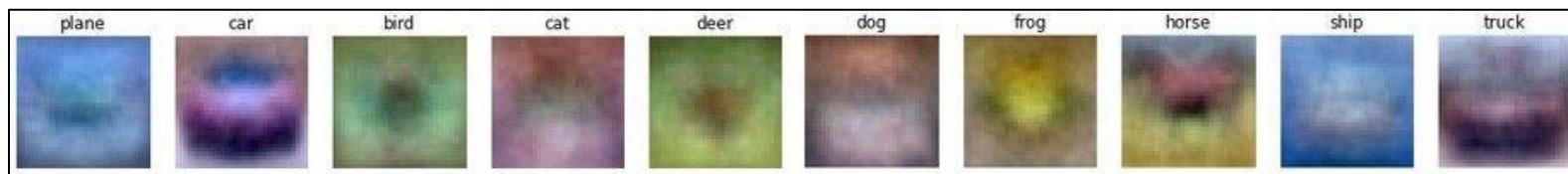
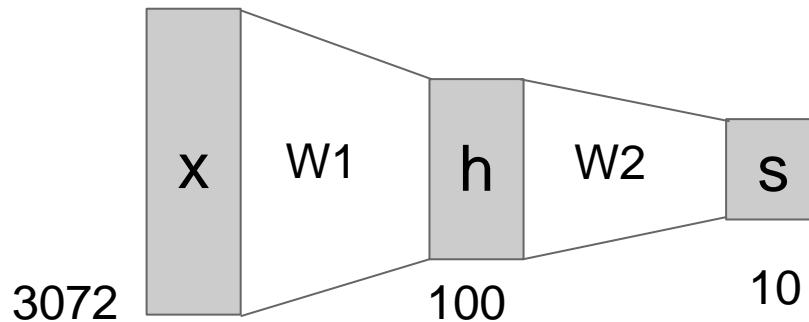
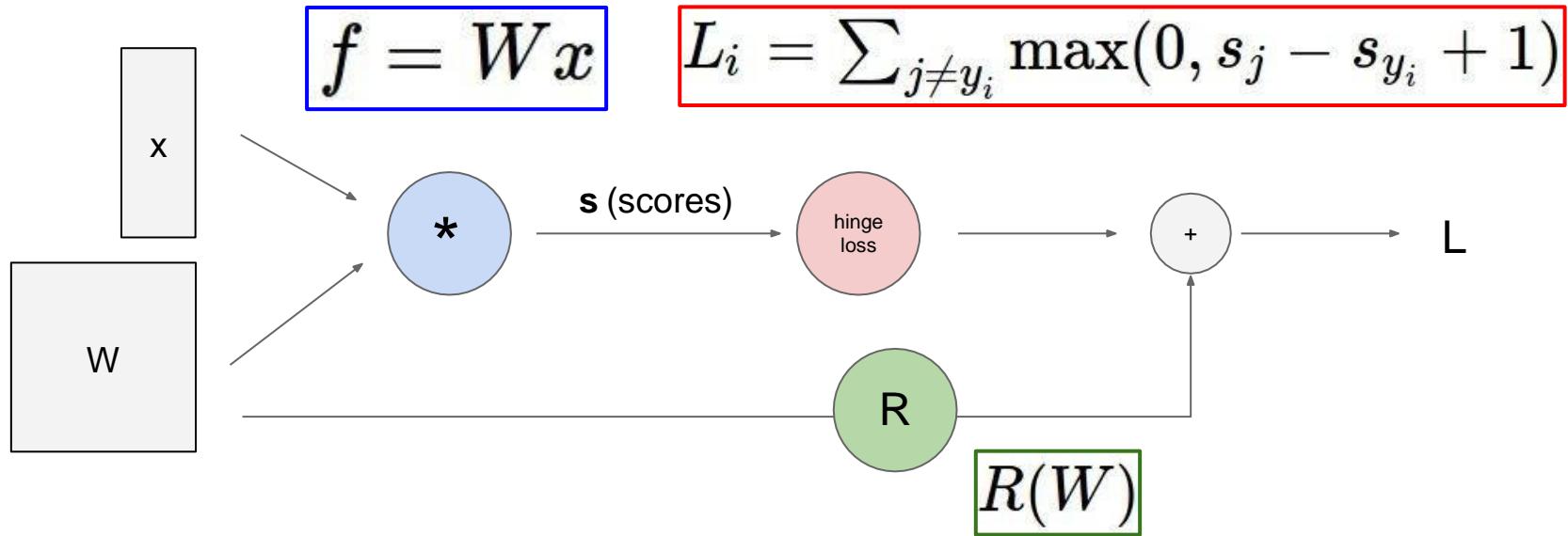
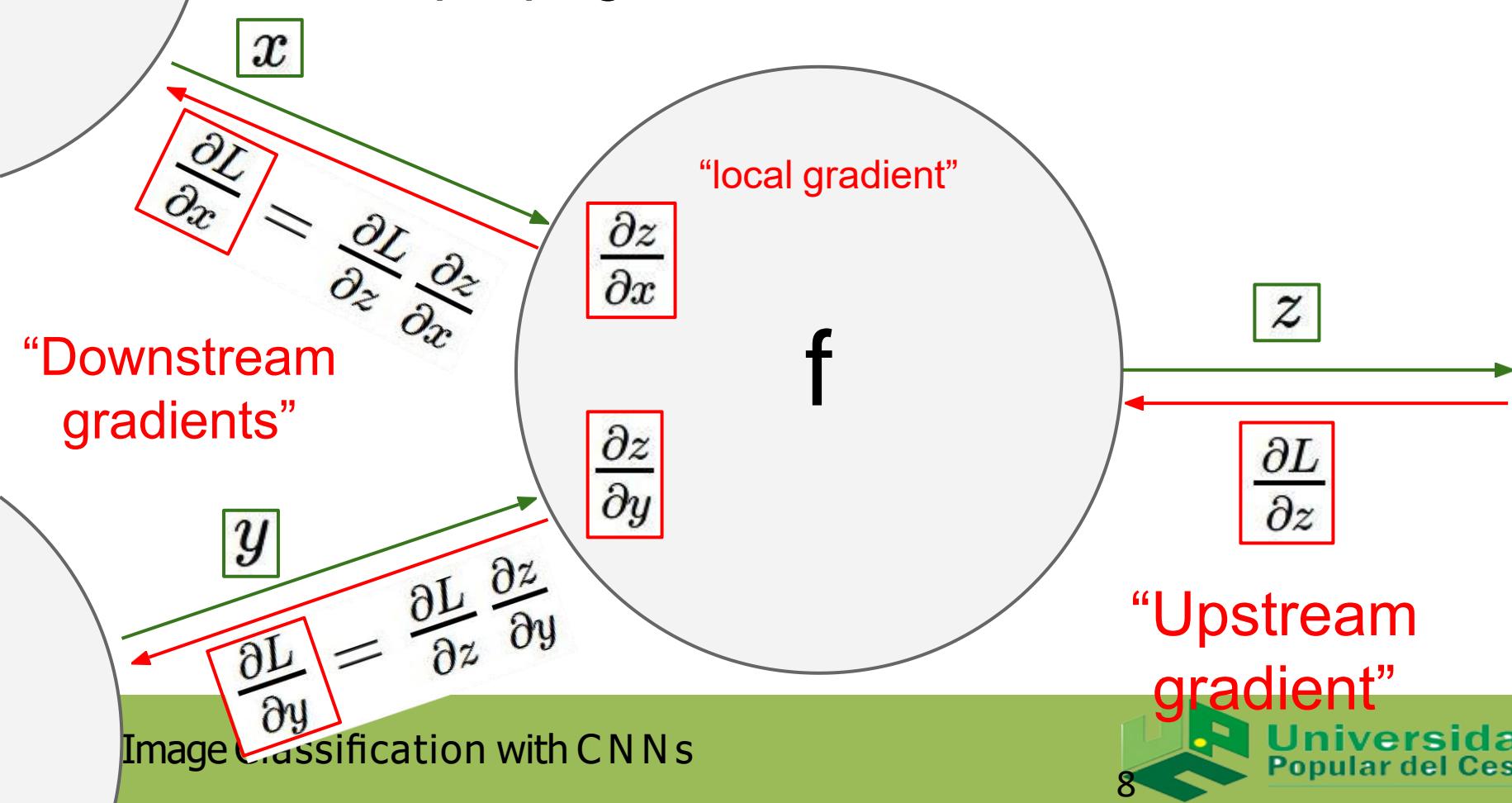


Image Classification with CNNs

# Last time: Computation Graph



# Last time: Backpropagation



# Backprop with Vectors

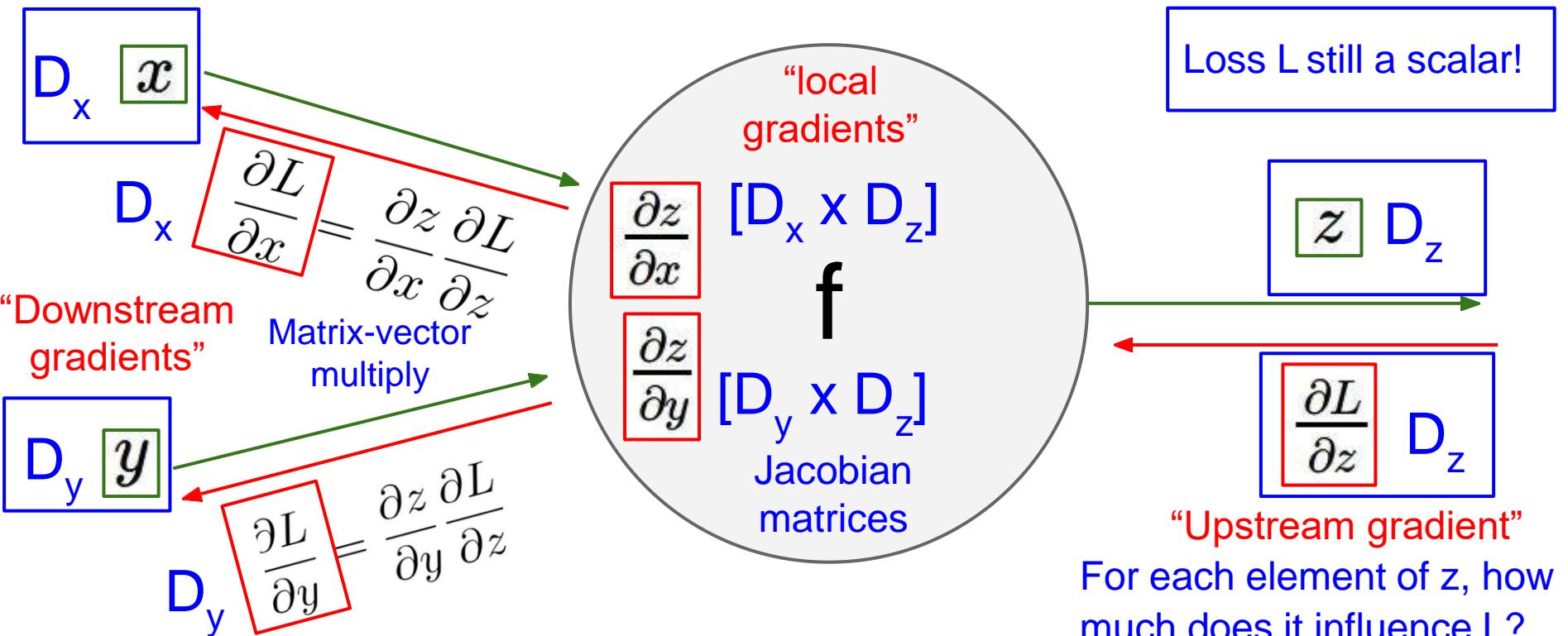
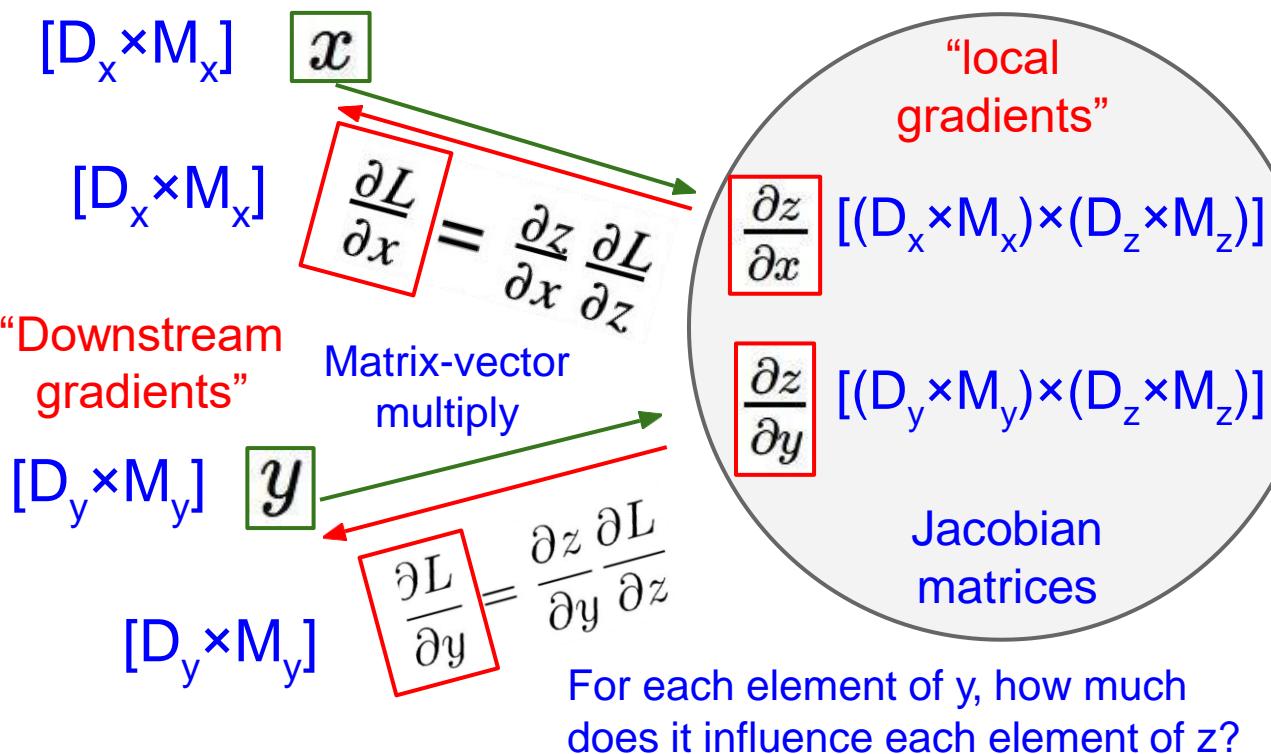


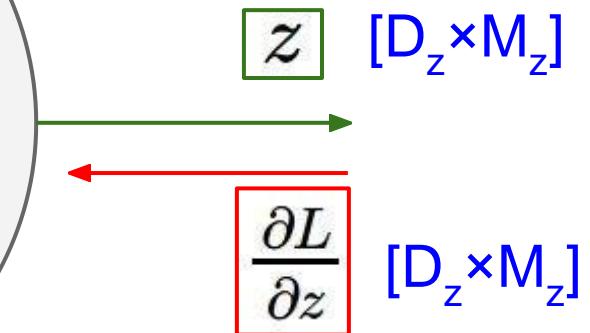
Image Classification with C N N s

# Backprop with Matrices (or Tensors)



Loss  $L$  still a scalar!

$dL/dx$  always has the same shape as  $x$ !



"Upstream gradient"  
For each element of  $z$ , how much does it influence  $L$ ?

# Image Classification: A core task in Computer Vision



This image by Nikita is  
licensed under CC-BY 2.0

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



cat  
dog  
bird  
deer  
truck

# Pixel space



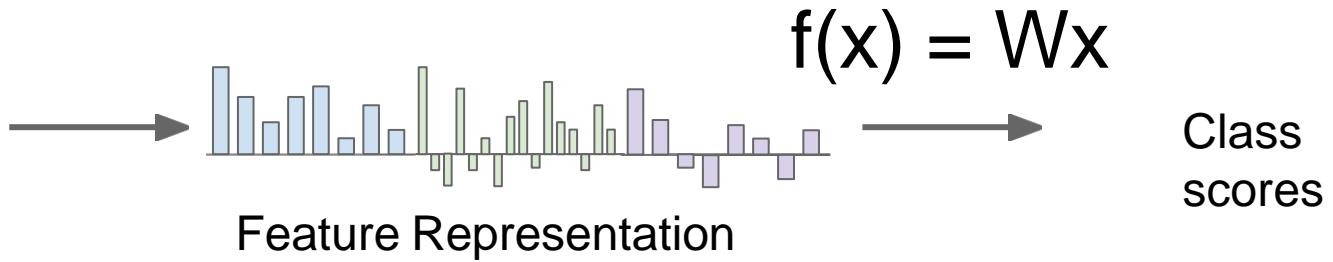
$$f(x) = Wx$$

Class  
scores



Image Classification with C N N s

# Image features



# Example: Color Histogram

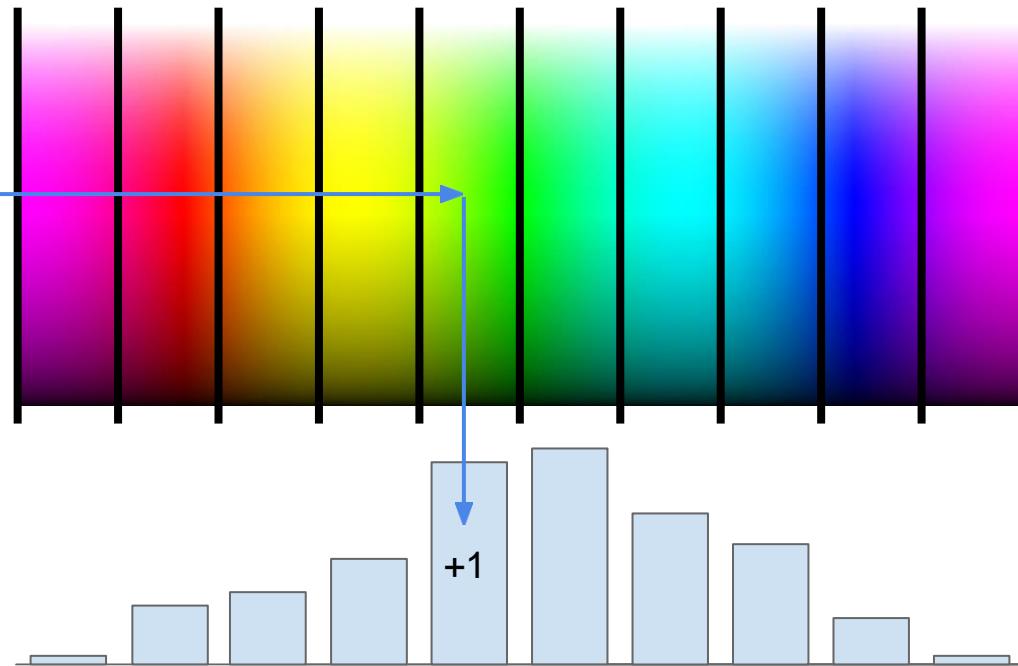
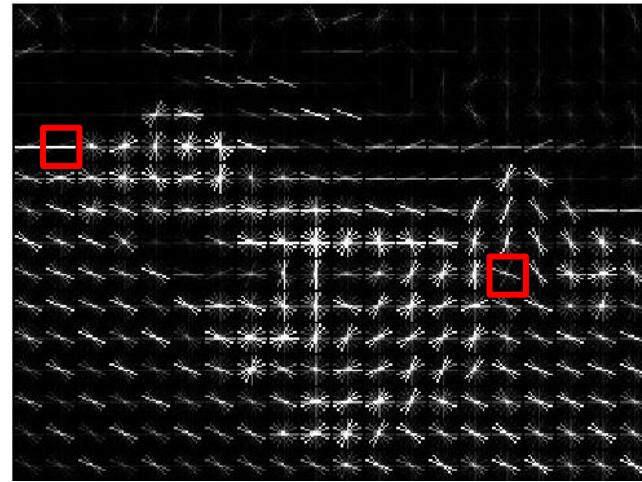


Image Classification with C N N s

# Example: Histogram of Oriented Gradients (HoG)



Divide image into 8x8 pixel regions  
Within each region quantize edge  
direction into 9 bins



Example: 320x240 image gets divided  
into 40x30 bins; in each bin there are  
9 numbers so feature vector has  
 $30 \times 40 \times 9 = 10,800$  numbers

Lowe, "Object recognition from local scale-invariant features", ICCV 1999

Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005

Image Classification with CNNs

# Example: Bag of Words

## Step 1: Build codebook



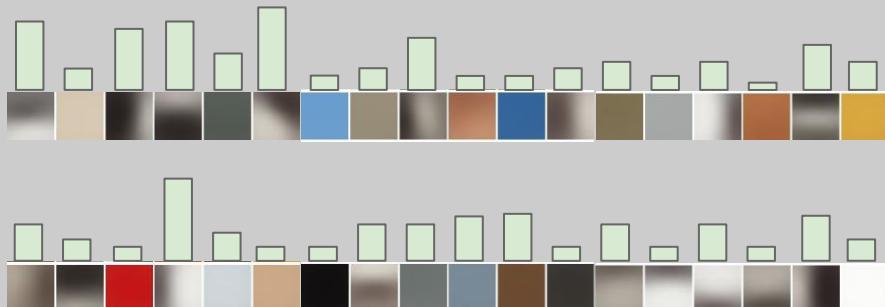
Extract random patches



Cluster patches to form “codebook” of “visual words”



## Step 2: Encode images



Fei-Fei and Perona, "A bayesian hierarchical model for learning natural scene categories", CVPR 2005

Image Classification with CNNs

# Image Features

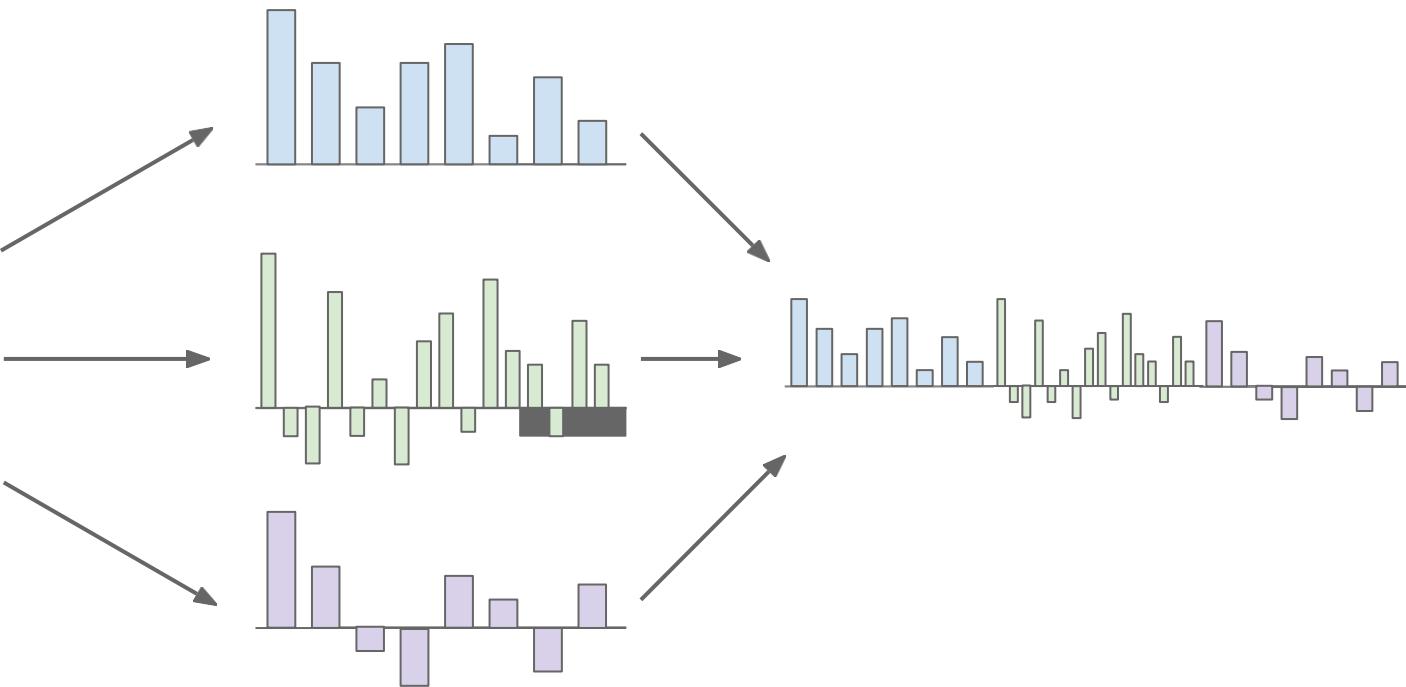
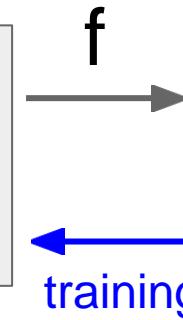
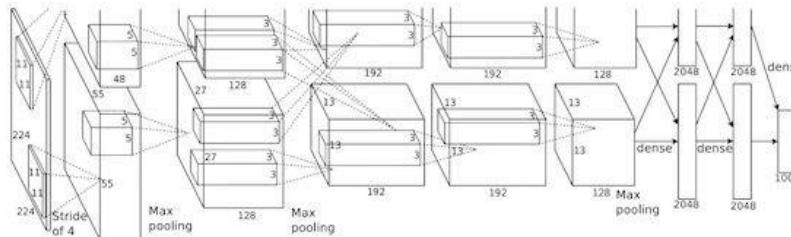


Image Classification with C N Ns

# Image features vs. ConvNets



10 numbers giving scores for classes



Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012.  
Figure copyright Krizhevsky, Sutskever, and Hinton, 2012.  
Reproduced with permission.



10 numbers giving scores for classes

# Last Time: Neural Networks

Linear score function:

$$f = Wx$$

2-layer Neural Network

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

The spatial structure of images is destroyed!

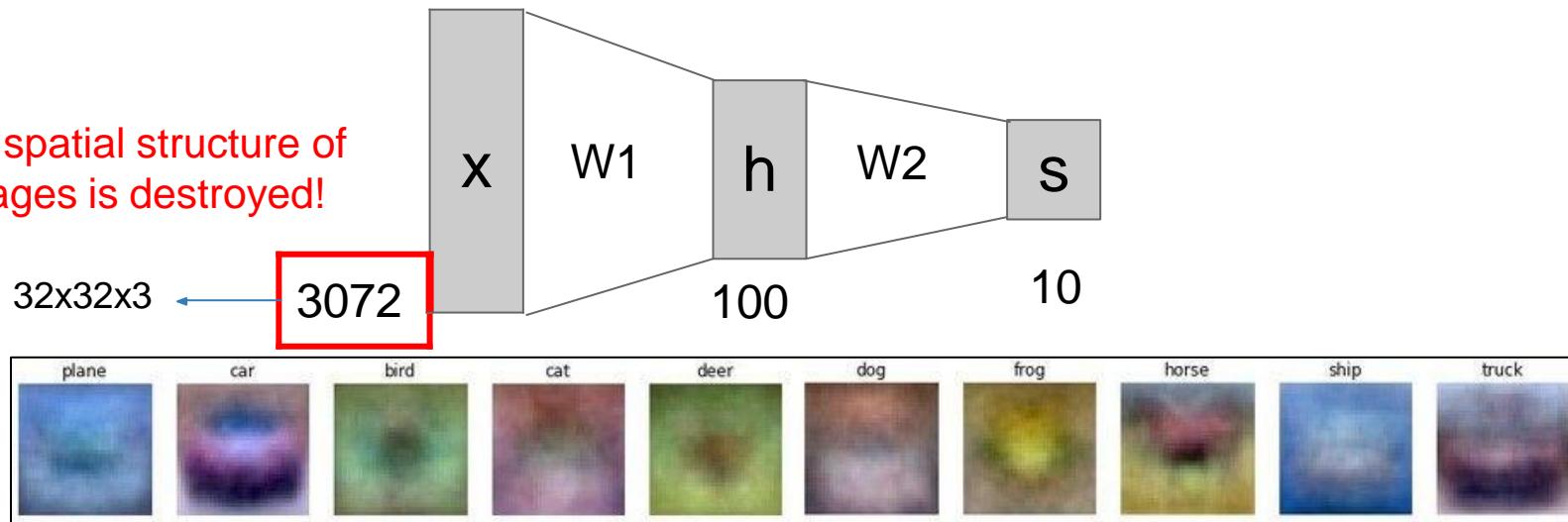


Image Classification with CNNs

# Next: Convolutional Neural Networks

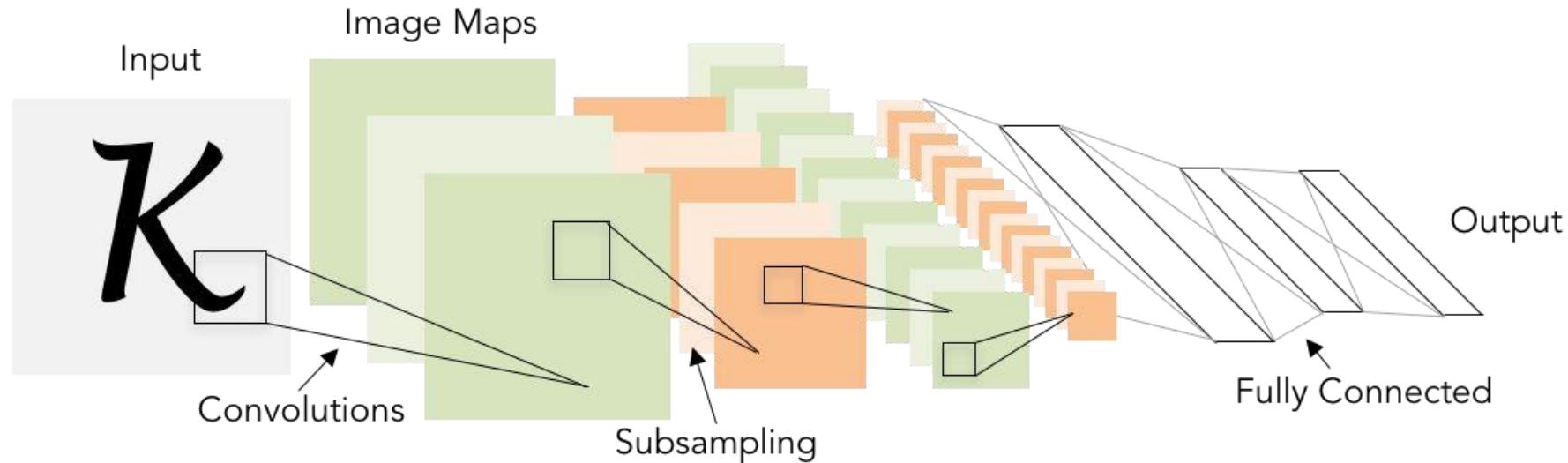


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Image Classification with C N N s

# A bit of history...

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

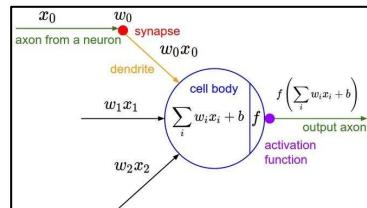
The machine was connected to a camera that used  $20 \times 20$  cadmium sulfide photocells to produce a 400-pixel image.

recognized  
letters of the alphabet

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

update rule:

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

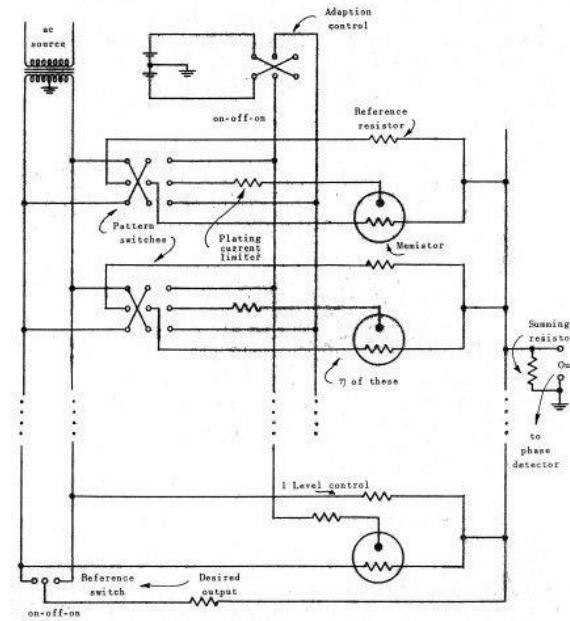
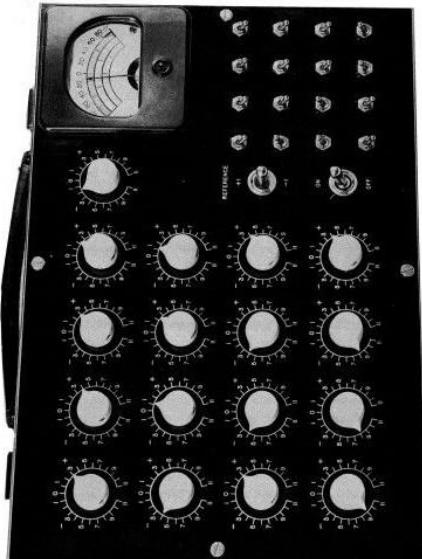
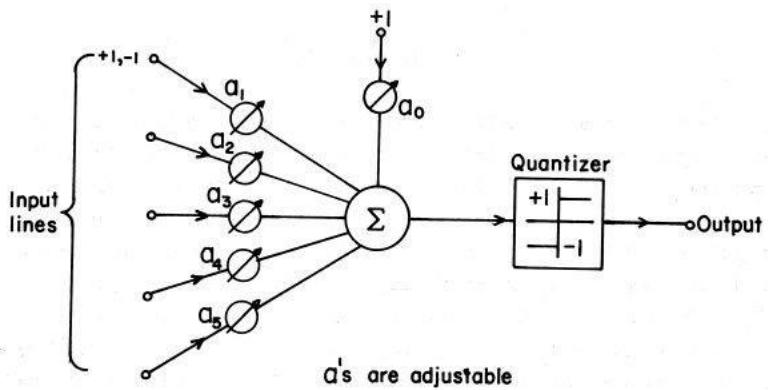


Frank Rosenblatt, ~1957: Perceptron



[This image](#) by Rocky Acosta is licensed under [CC-BY 3.0](#)

# A bit of history...



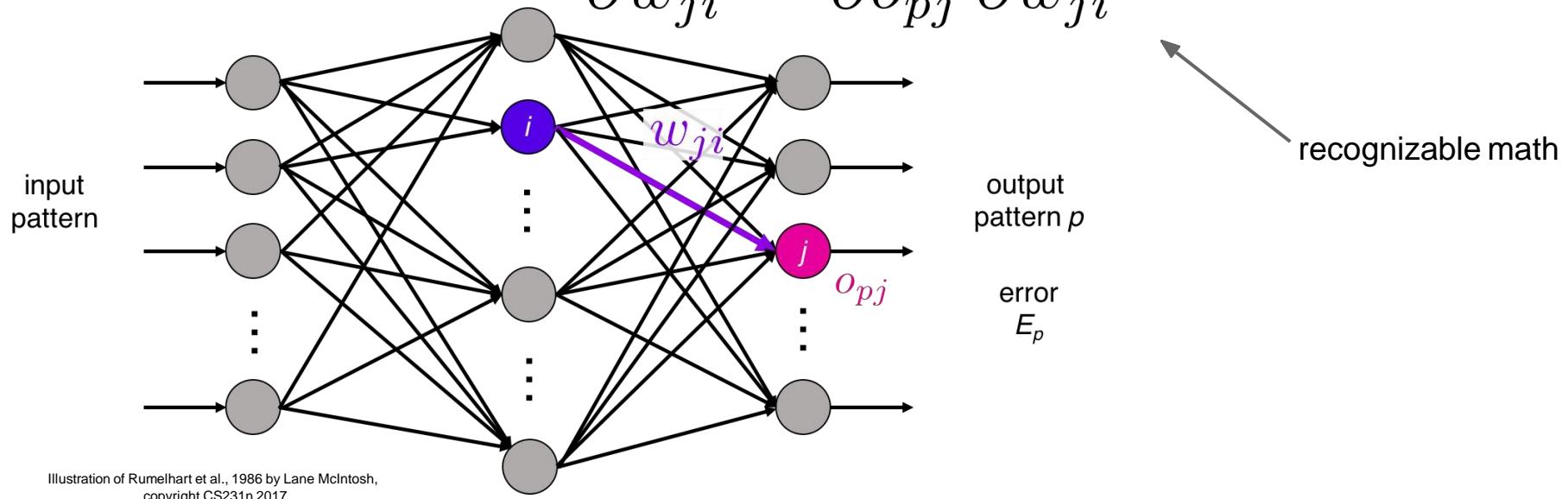
Widrow and Hoff, ~1960: Adaline/Madaline

These figures are reproduced from [Widrow 1960, Stanford Electronics Laboratories Technical Report](#) with permission from [Stanford University Special Collections](#).

Image Classification with CNNs

# 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

Image Classification with CN Ns

# A bit of history...

[Hinton and Salakhutdinov 2006]

Reinvigorated research in  
Deep Learning

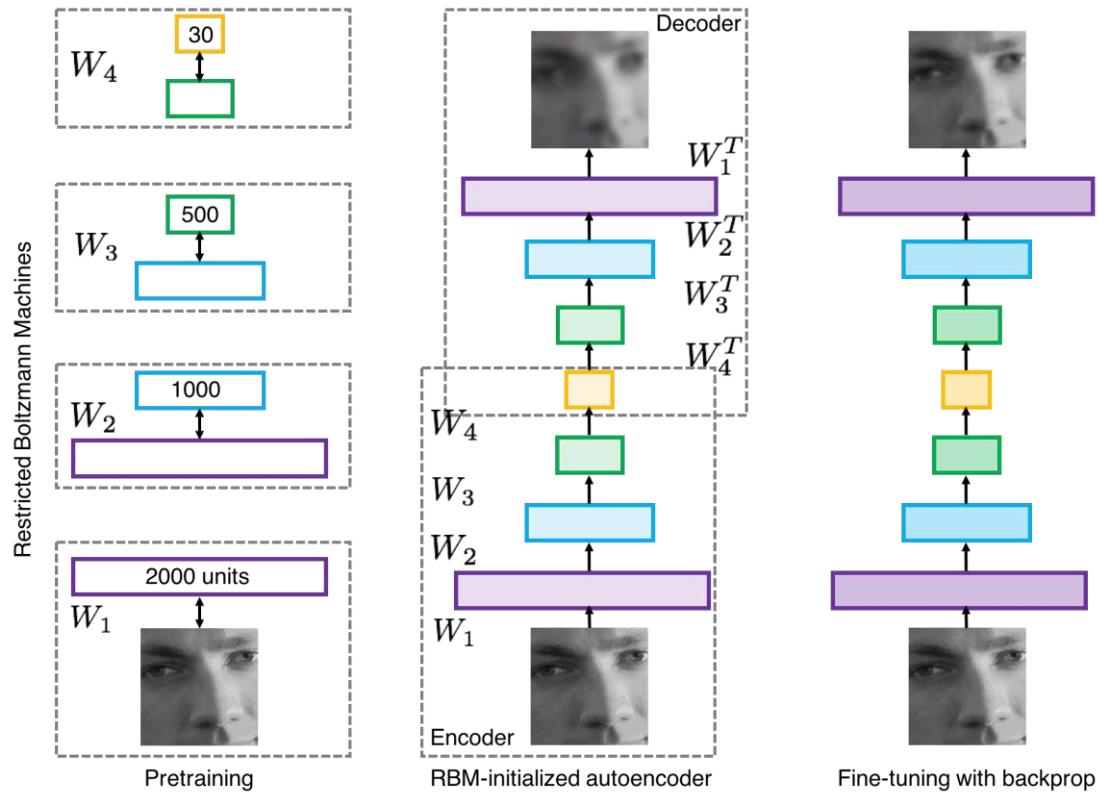


Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

Image Classification with CNNs

# 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

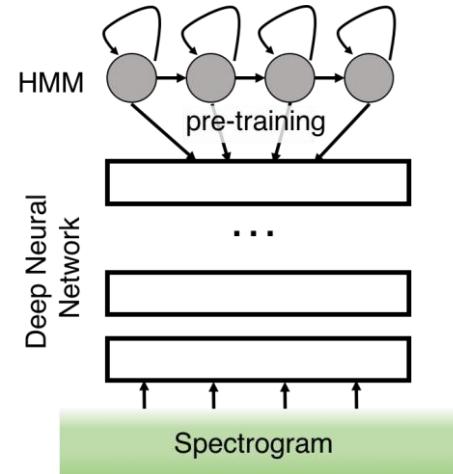
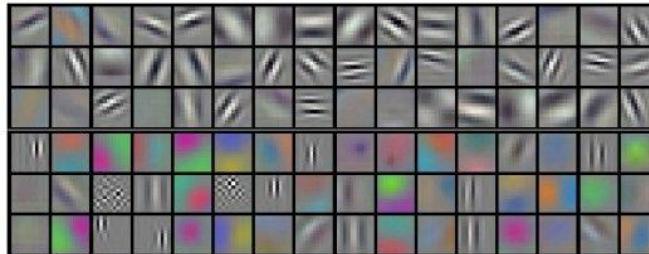
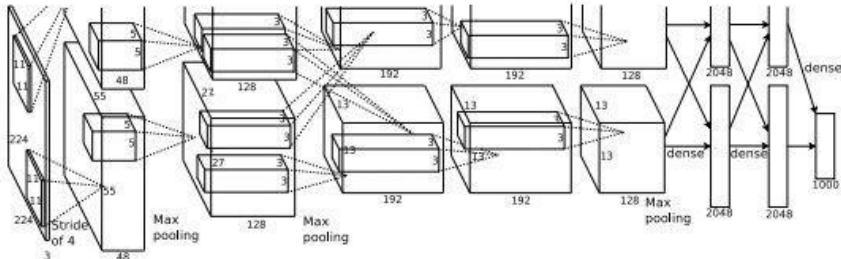


Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

## **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.

## Image Classification with C N N s

# A bit of history:

**Hubel & Wiesel,  
1959**

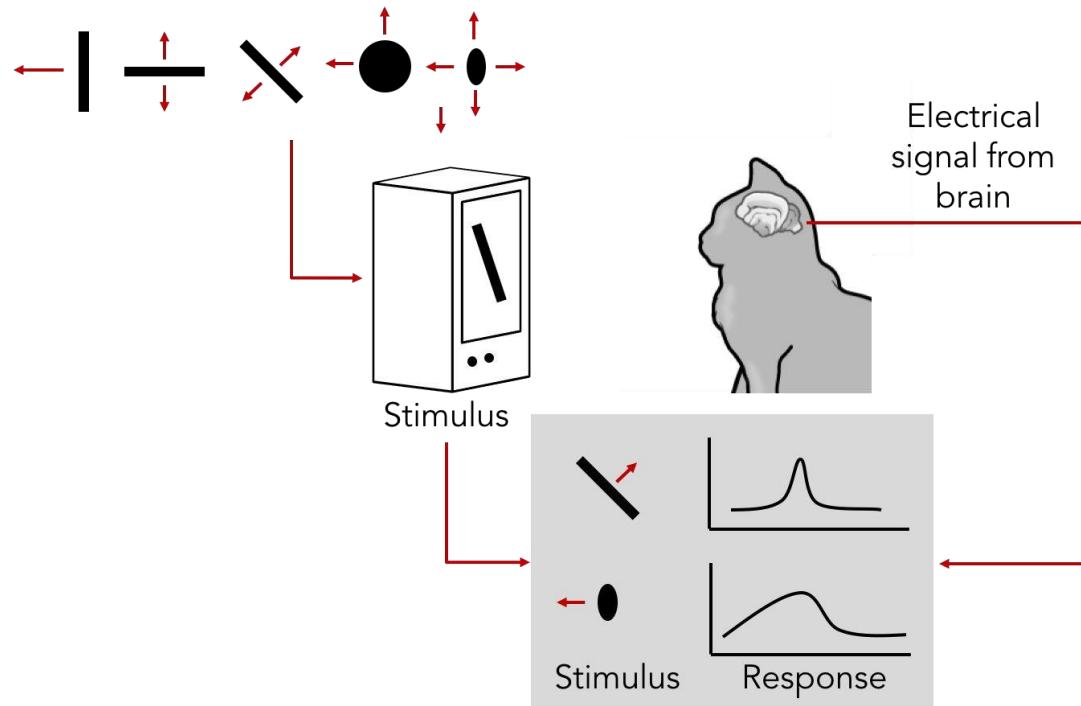
RECEPTIVE FIELDS OF SINGLE  
NEURONES IN  
THE CAT'S STRIATE CORTEX

**1962**

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

**1968...**

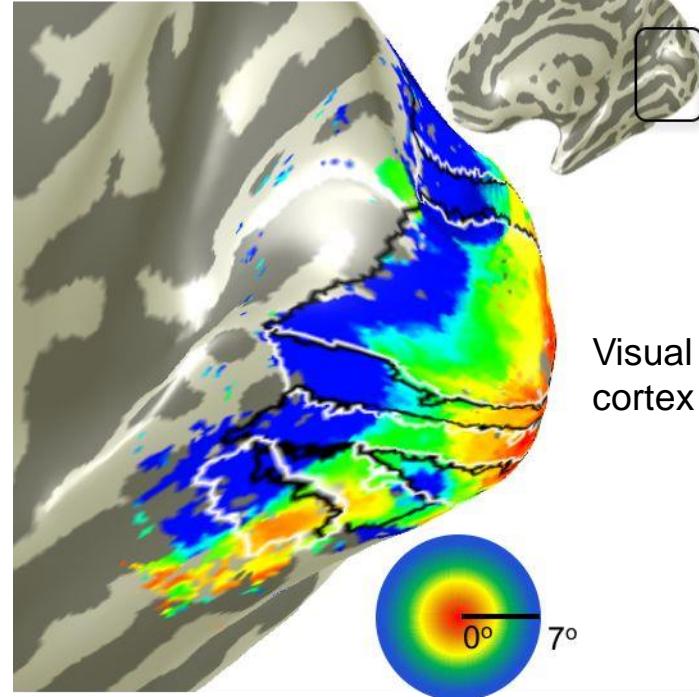
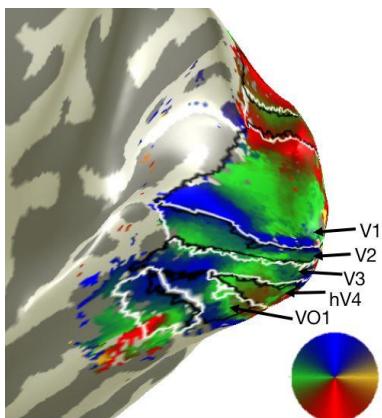
Image Classification with CNNs



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

# A bit of history

**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.

Image Classification with C N N s

# Hierarchical organization

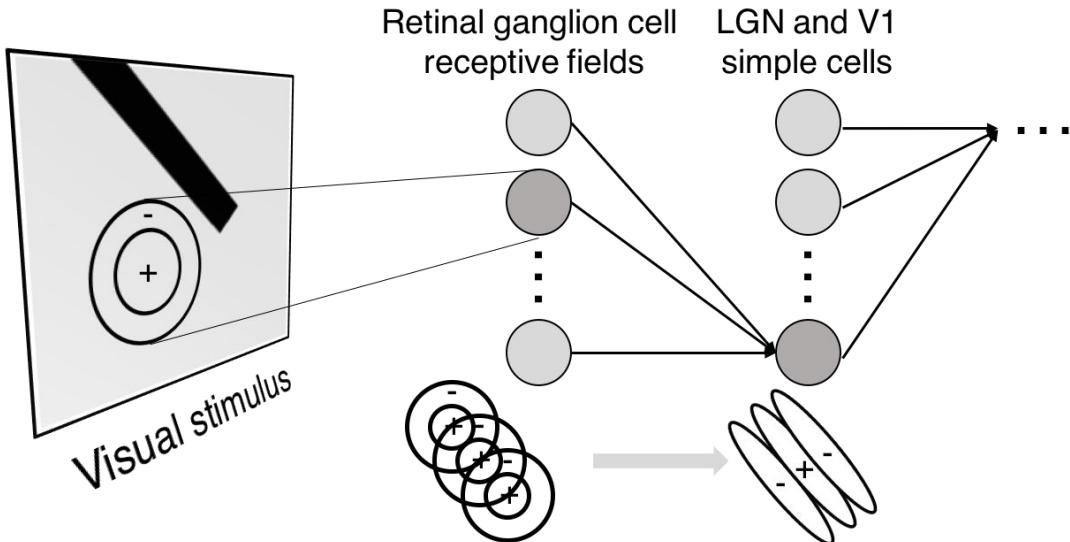


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

**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)

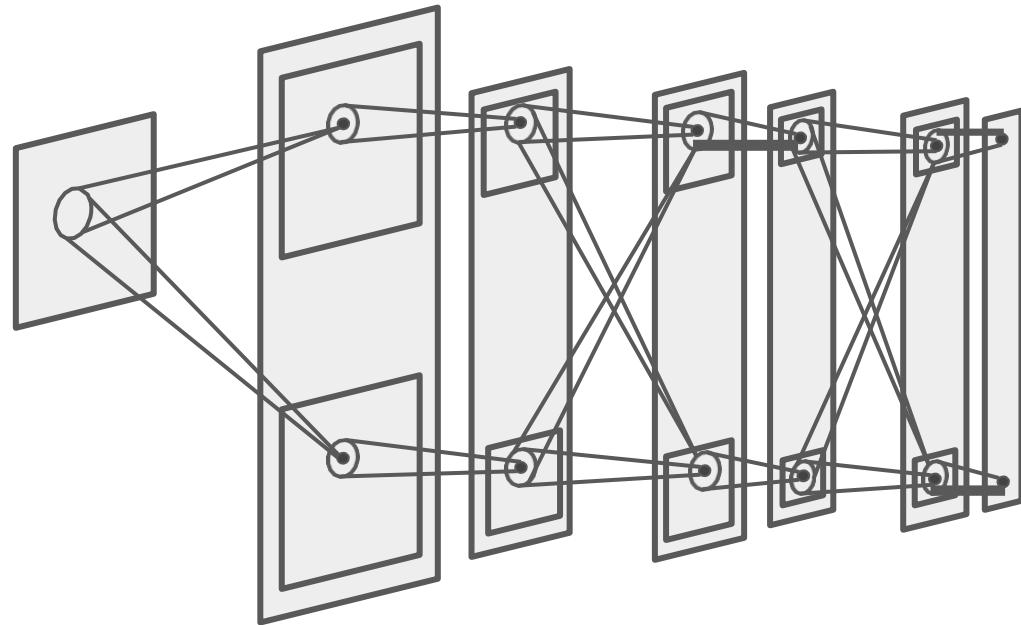
# A bit of history:

## Neocognitron [Fukushima 1980]

“sandwich” architecture (SCSCSC...)

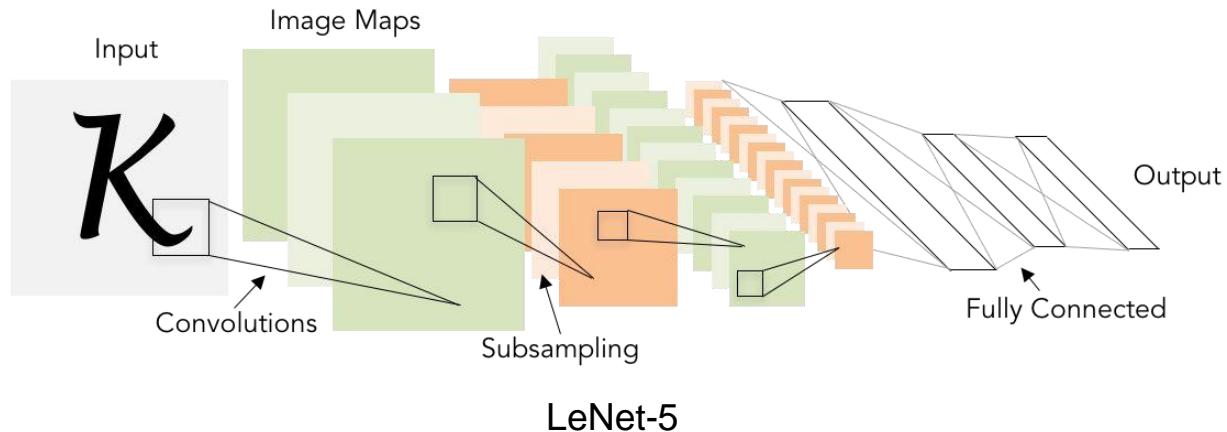
simple cells: modifiable parameters

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]*

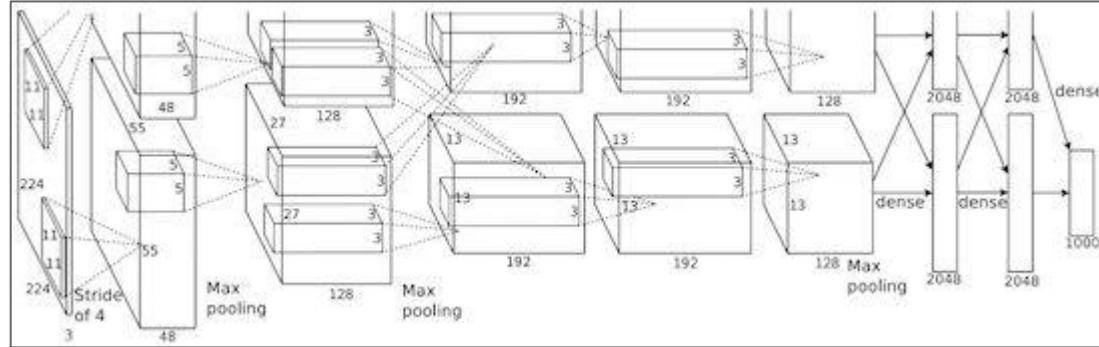
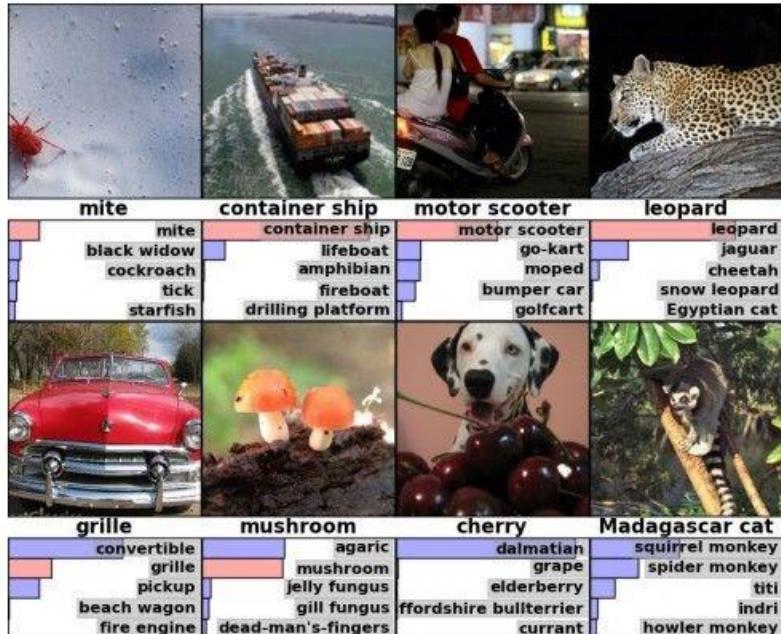


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

“AlexNet”

# Fast-forward to today: ConvNets are everywhere

Classification



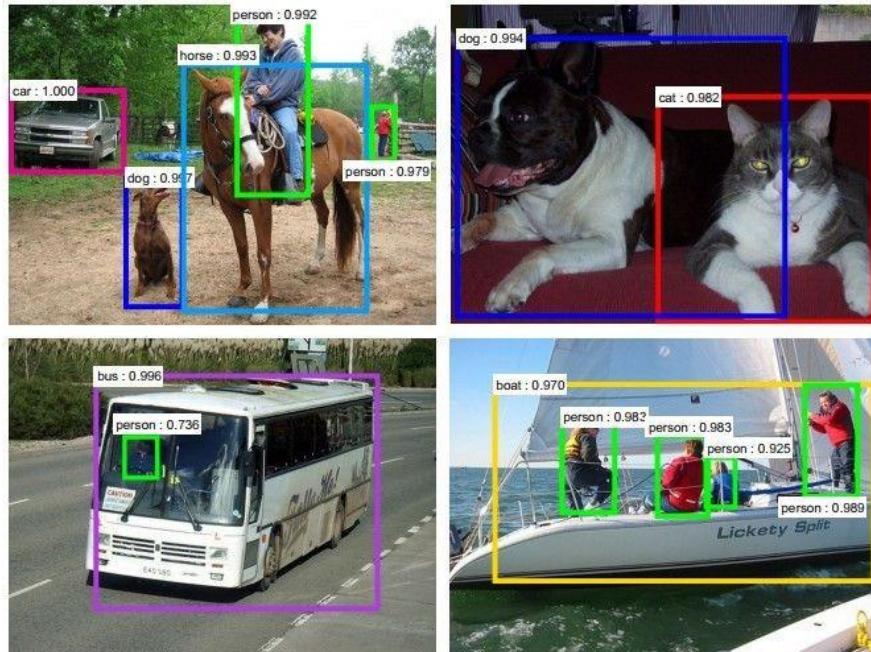
Retrieval



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

# Fast-forward to today: ConvNets are everywhere

## Detection

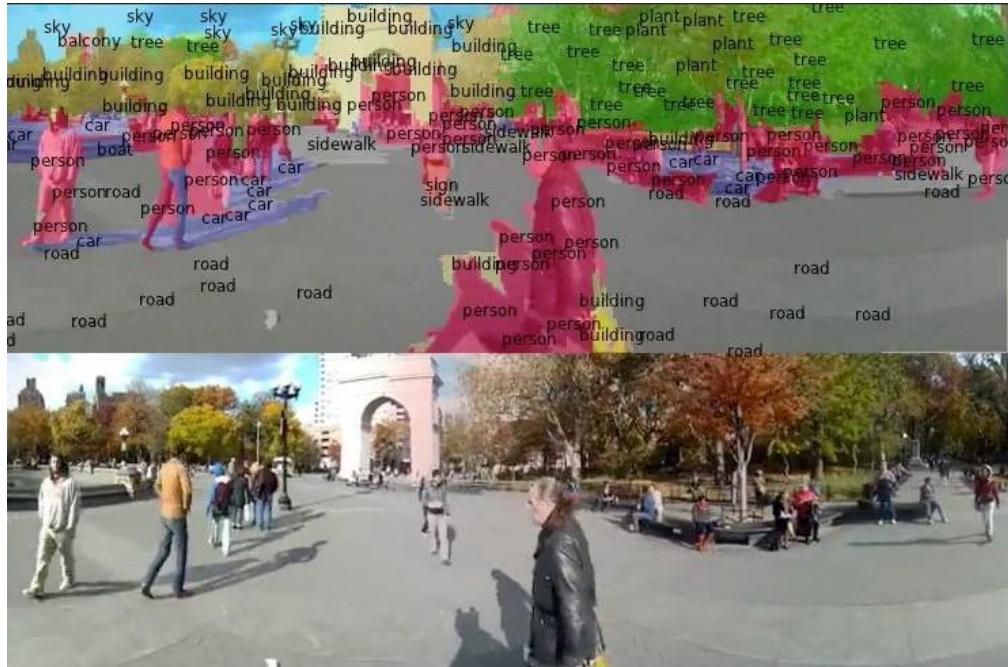


Figures copyright Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, 2015. Reproduced with permission.

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

Image Classification with CN Ns

## 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.



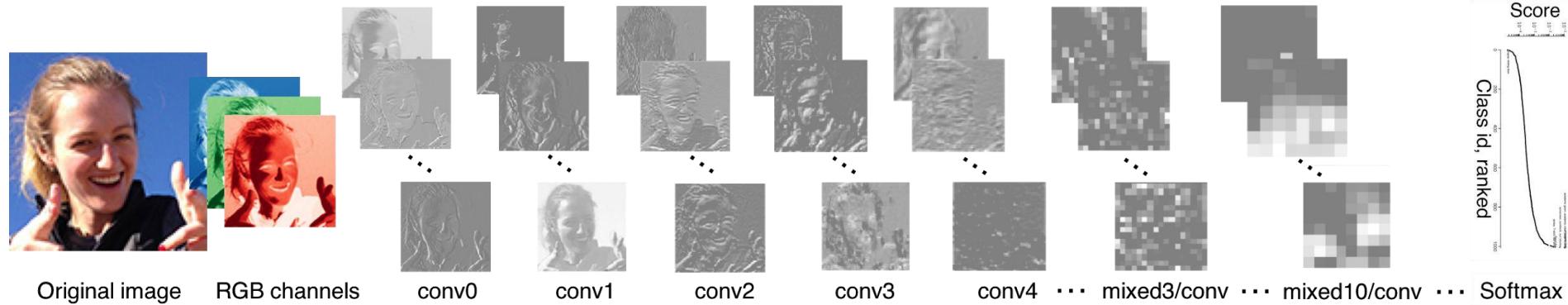
[This image](#) by GBPublic\_PR is  
licensed under [CC-BY 2.0](#)

## NVIDIA Tesla line

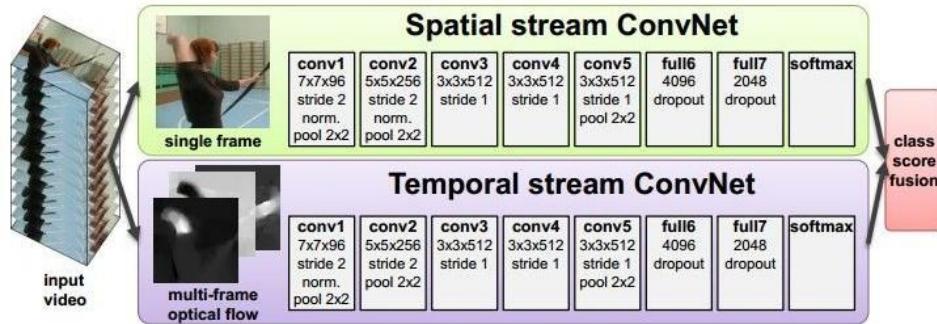
(these are the GPUs on rye01.stanford.edu)

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



[Taigman et al. 2014]

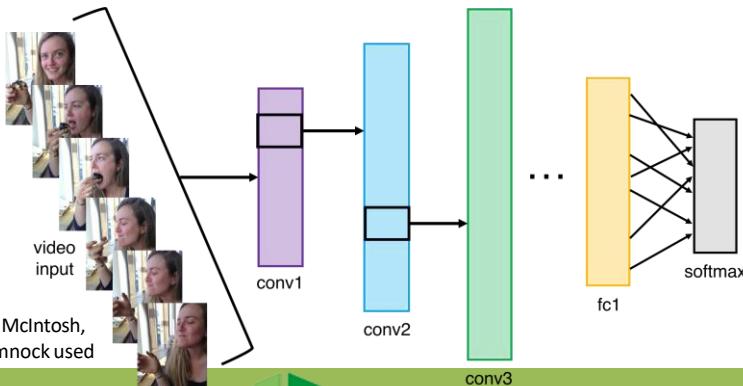


[Simonyan et al. 2014]

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 Curnock used  
with permission.



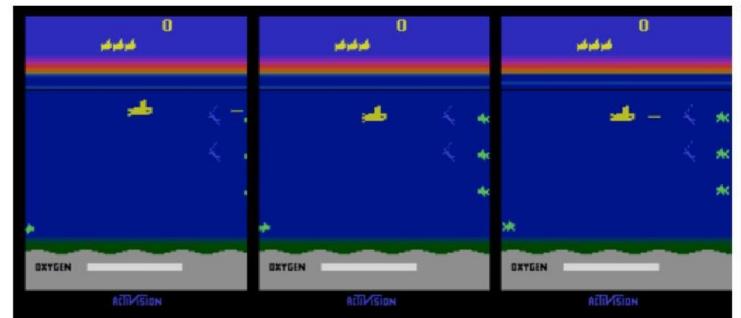
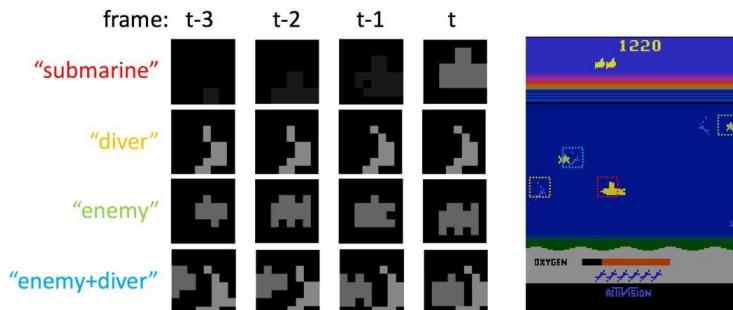
## Image Classification with CNNs

# 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]

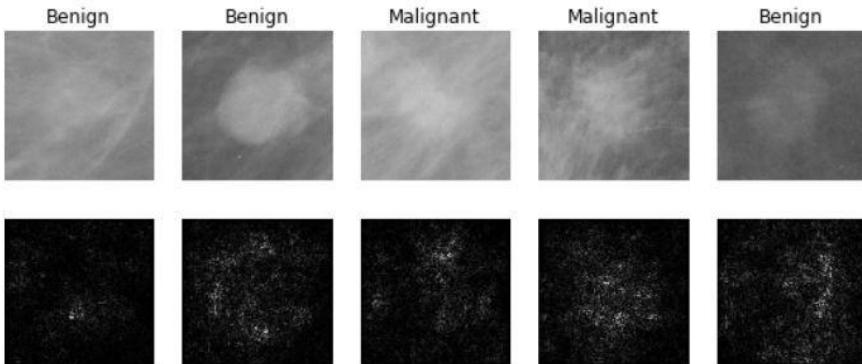


[Guo et al. 2014]

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

Image Classification with C N N s

# 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.]

Image Classification with ConvNets

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



*Whale recognition, Kaggle Challenge*

Image Classification with C N Ns

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*



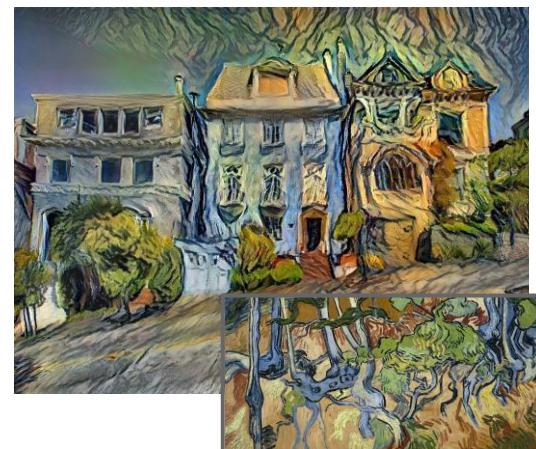
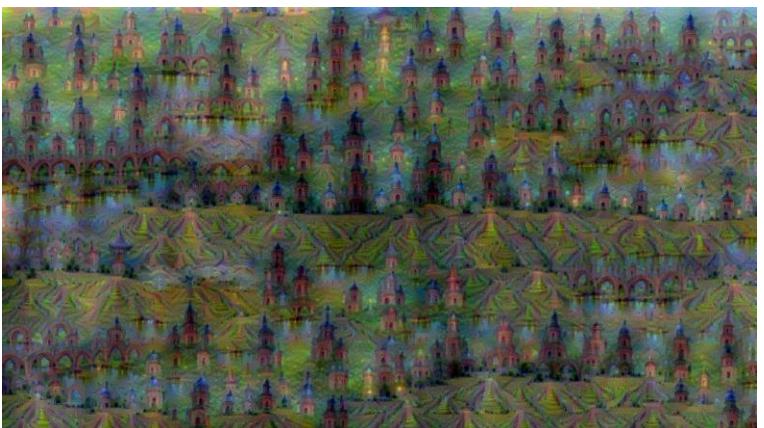
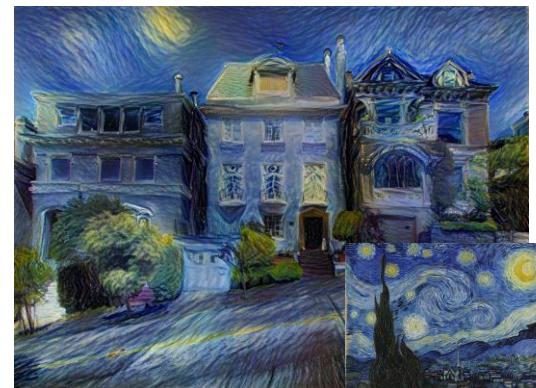
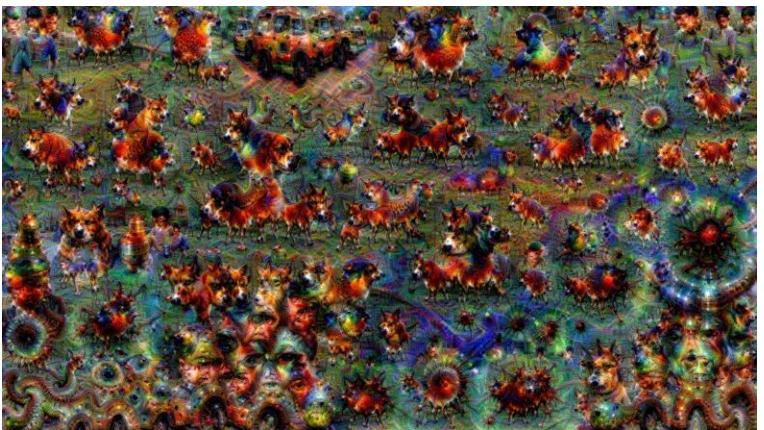
*A woman standing on a beach holding a surfboard*

# 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

Styled 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

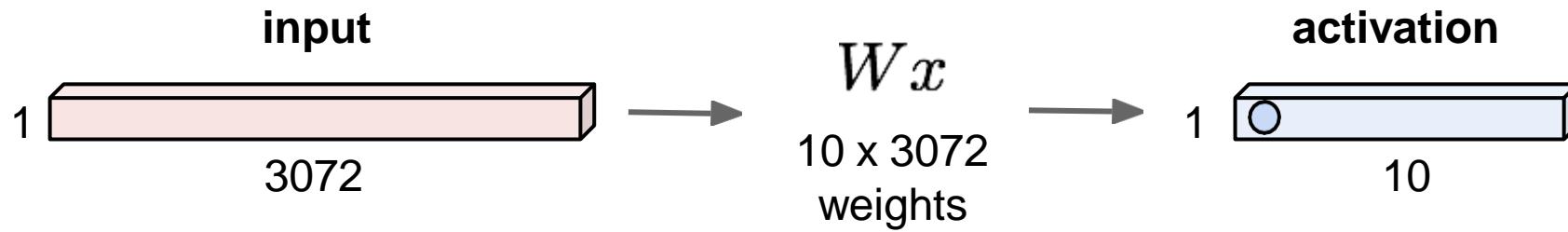
## Image Classification with CNNs

# Convolutional Neural Networks

Image Classification with C N N s

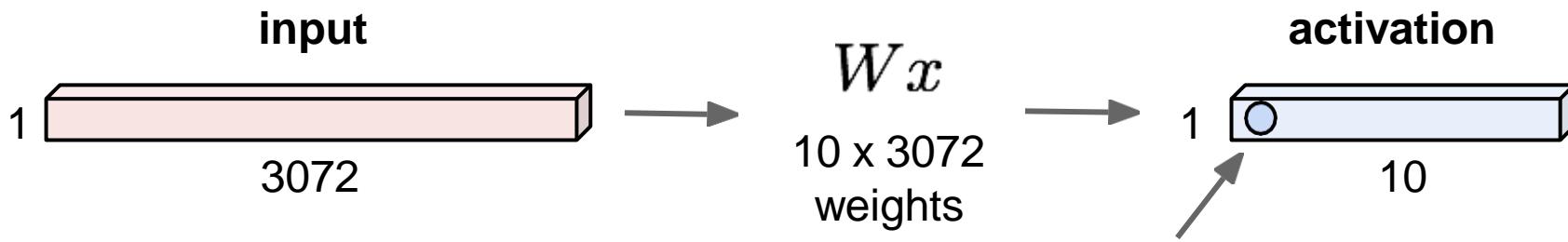
# Recap: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



# Fully Connected Layer

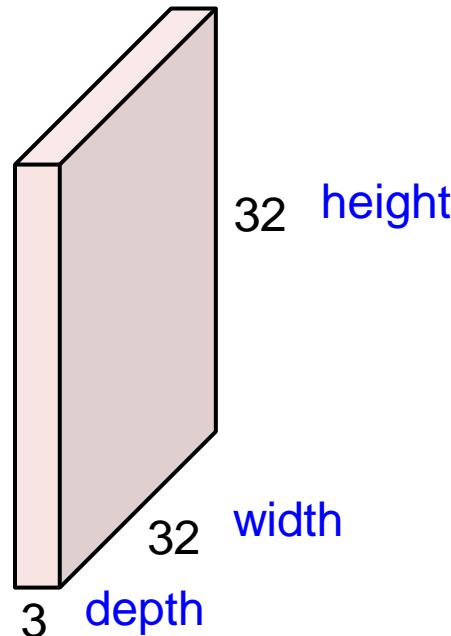
32x32x3 image -> stretch to 3072 x 1



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

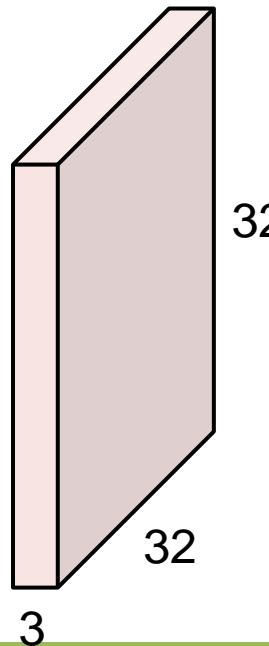
# Convolution Layer

32x32x3 image -> preserve spatial structure

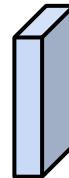


# Convolution Layer

32x32x3 image



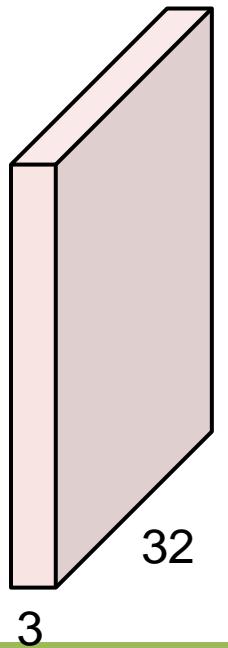
5x5x3 filter



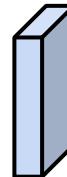
**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

# Convolution Layer

32x32x3 image



5x5x3 filter

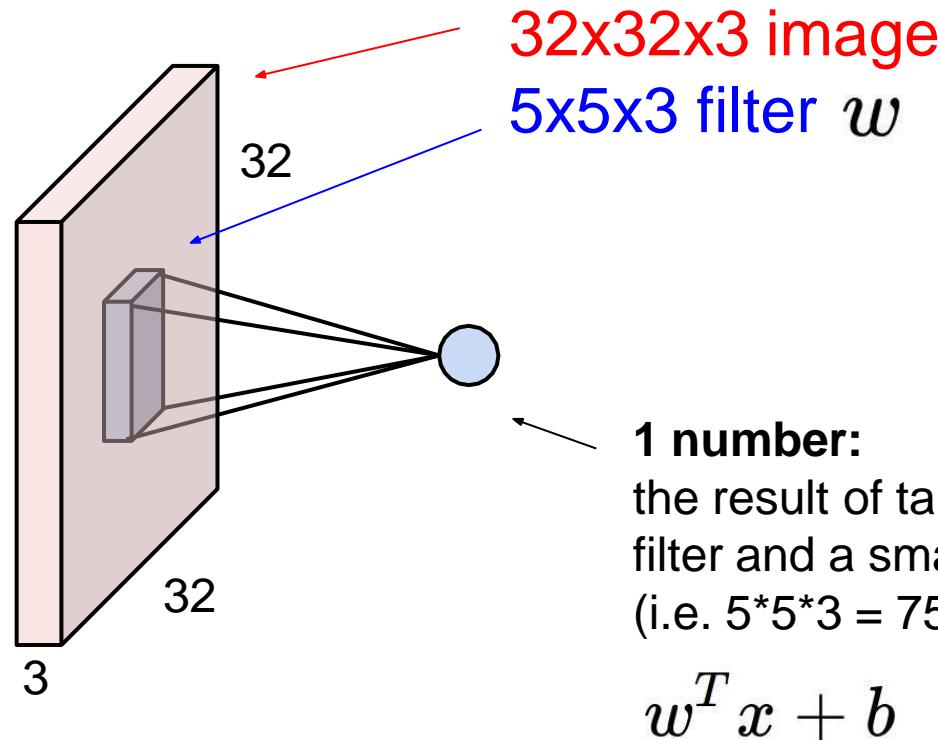


Filters always extend the full depth of the input volume



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

# Convolution Layer



# Convolution Layer

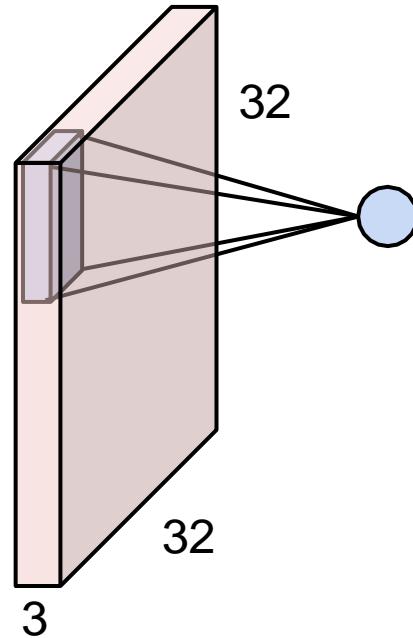


Image Classification with C N N s

# Convolution Layer

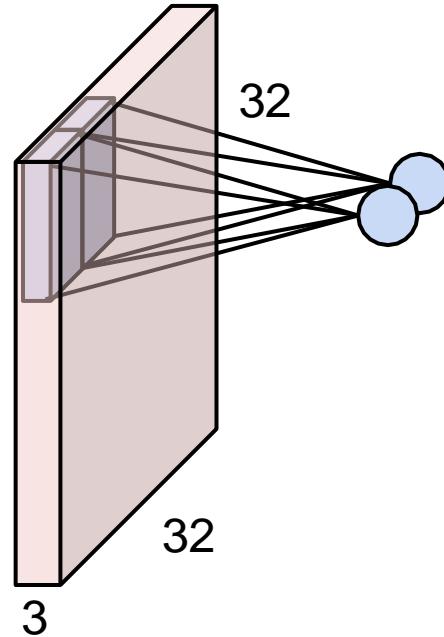


Image Classification with C N N s

# Convolution Layer

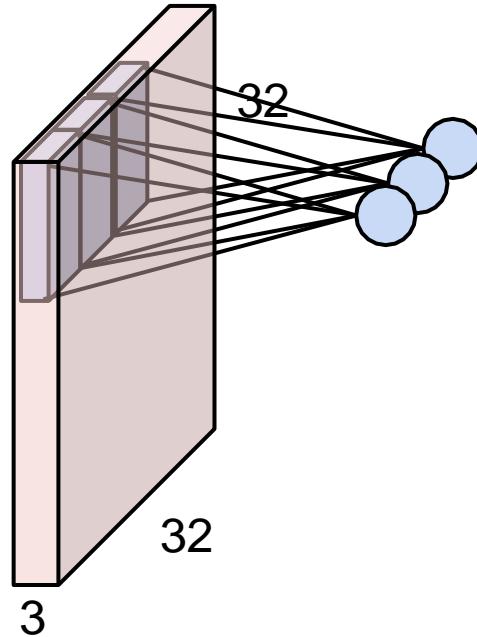


Image Classification with C N N s

# Convolution Layer

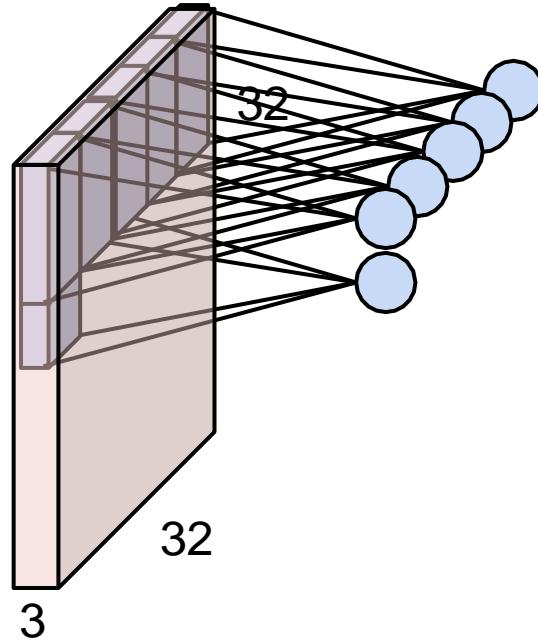
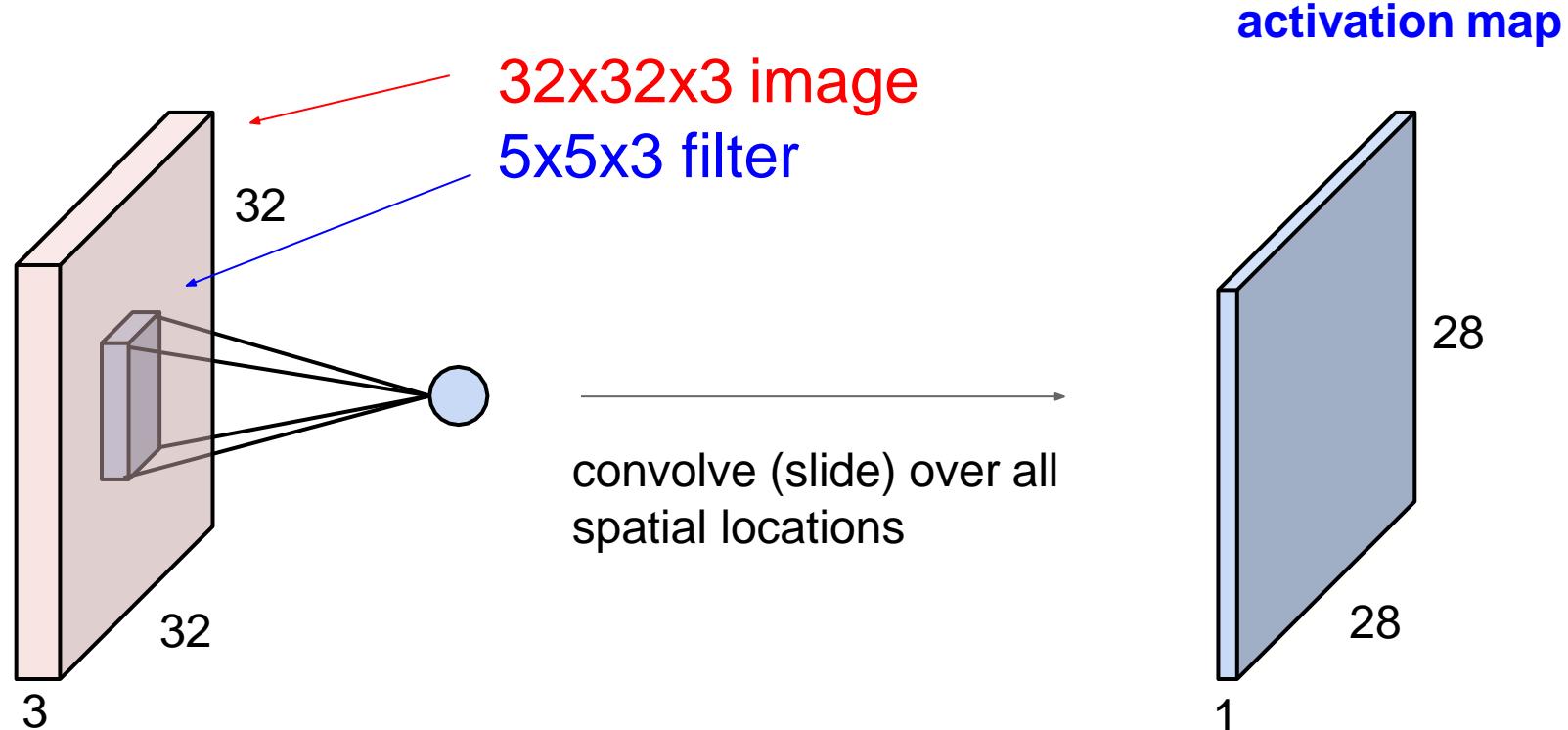


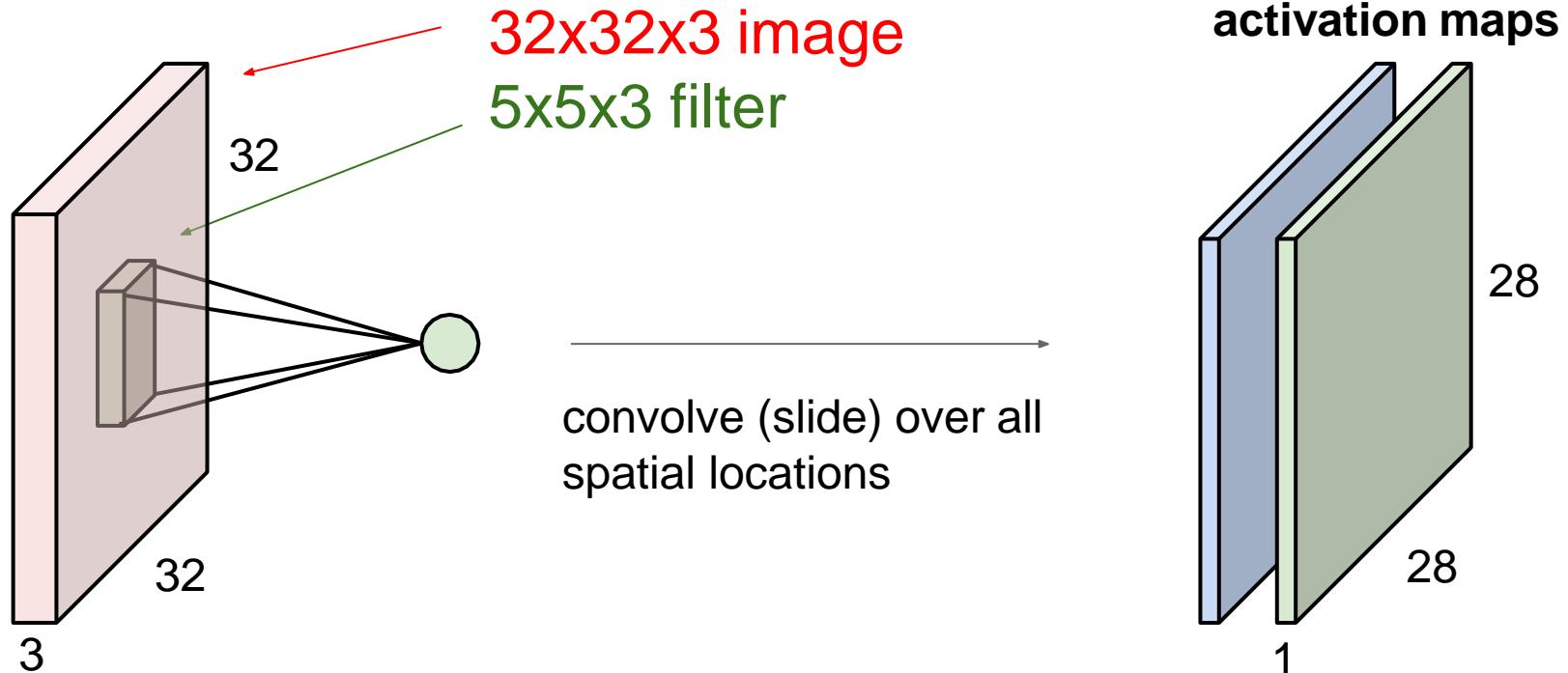
Image Classification with C N N s

# Convolution Layer



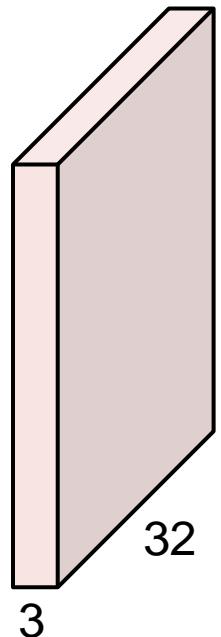
# Convolution Layer

consider a second, green filter

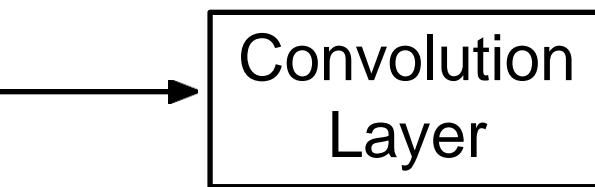


# Convolution Layer

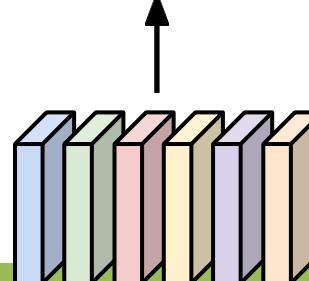
3x32x32 image



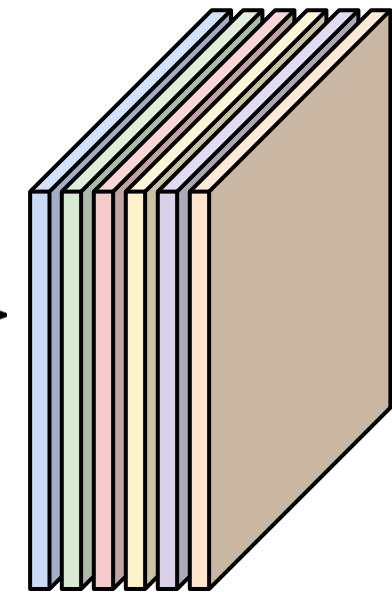
Consider 6 filters,  
each 3x5x5



6x3x5x5  
filters



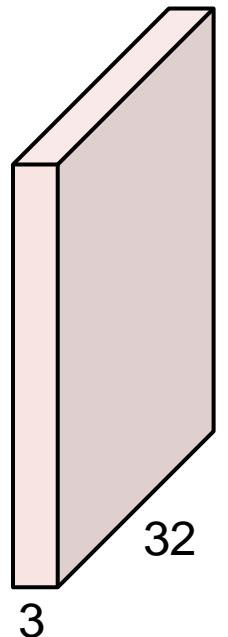
6 activation maps,  
each 1x28x28



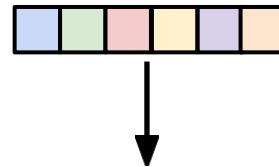
Stack activations to get a  
6x28x28 output image!

# Convolution Layer

3x32x32 image

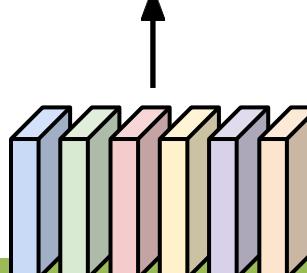


Also 6-dim bias vector:

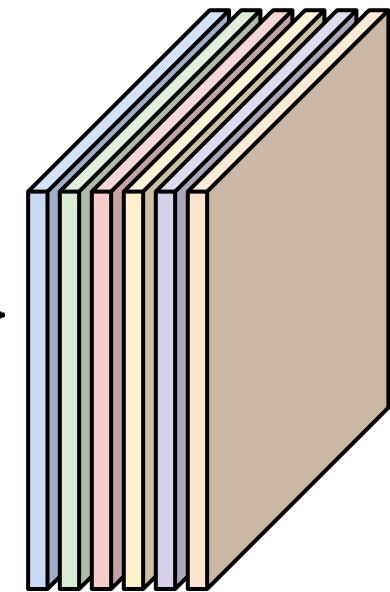


Convolution  
Layer

6x3x5x5  
filters



6 activation maps,  
each 1x28x28

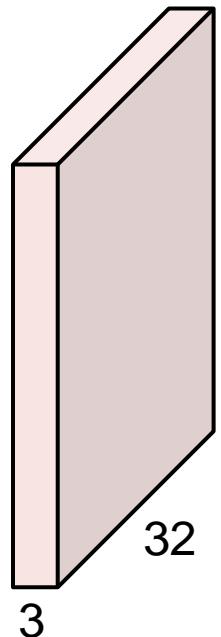


Stack activations to get a  
6x28x28 output image!

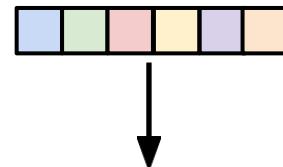
# Convolution Layer

28x28 grid, at each point a 6-dim vector

3x32x32 image

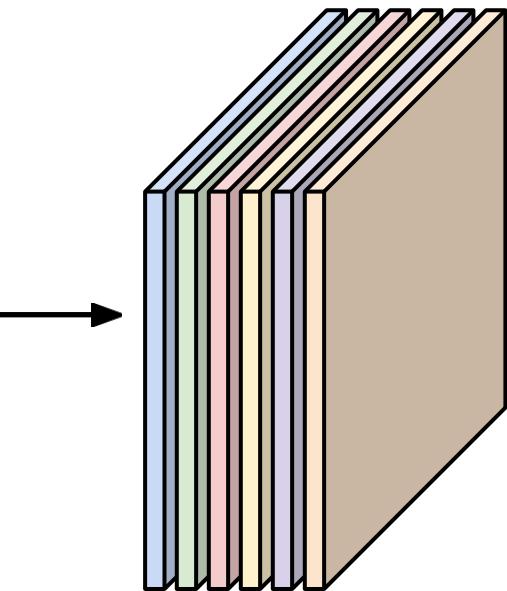
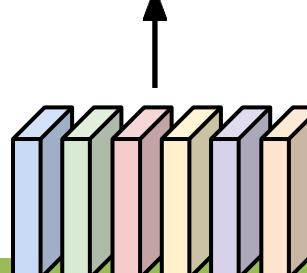


Also 6-dim bias vector:



Convolution  
Layer

6x3x5x5  
filters



Stack activations to get a  
6x28x28 output image!

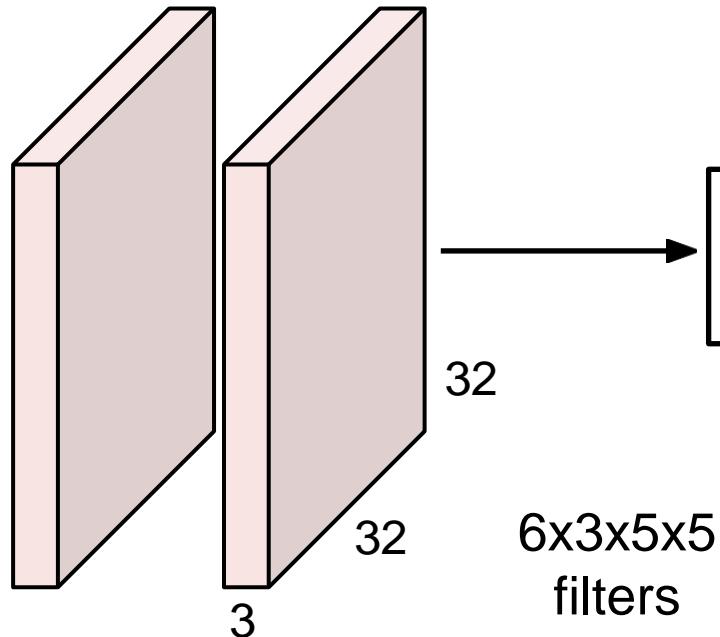
Slide inspiration: Justin Johnson

Image Classification with C N N s

# Convolution Layer

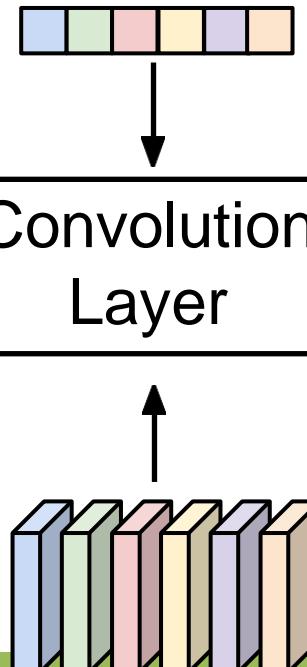
2x3x32x32

Batch of images



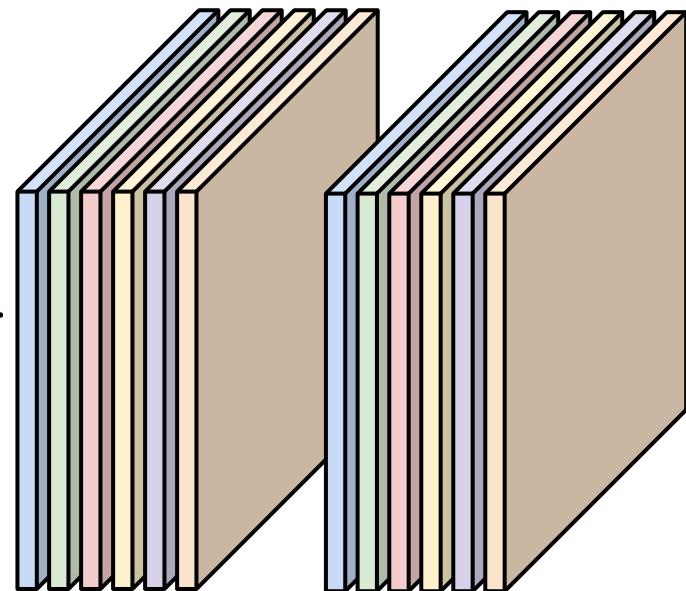
6x3x5x5  
filters

Also 6-dim bias vector:



2x6x28x28

Batch of outputs



Slide inspiration: Justin Johnson

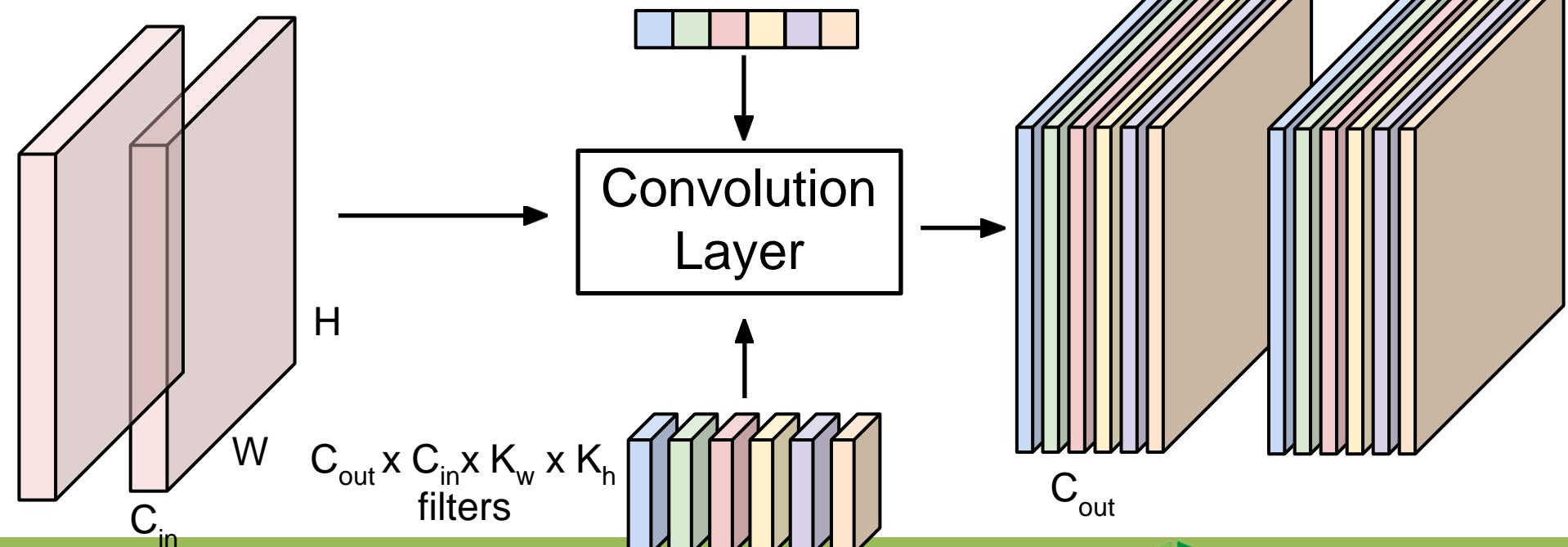
Image Classification with C N N s

# Convolution Layer

$N \times C_{in} \times H \times W$   
Batch of images

$N \times C_{out} \times H' \times W'$   
Batch of outputs

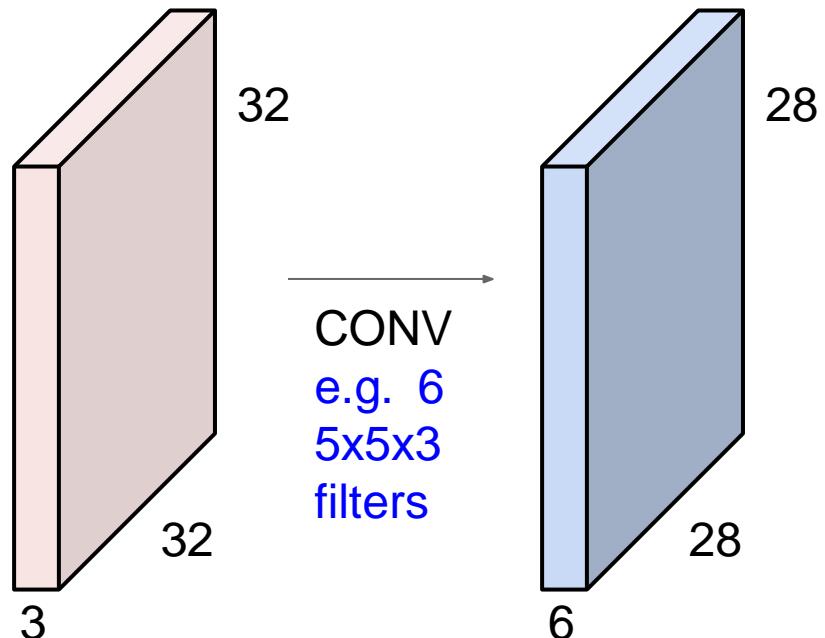
Also  $C_{out}$ -dim bias vector:



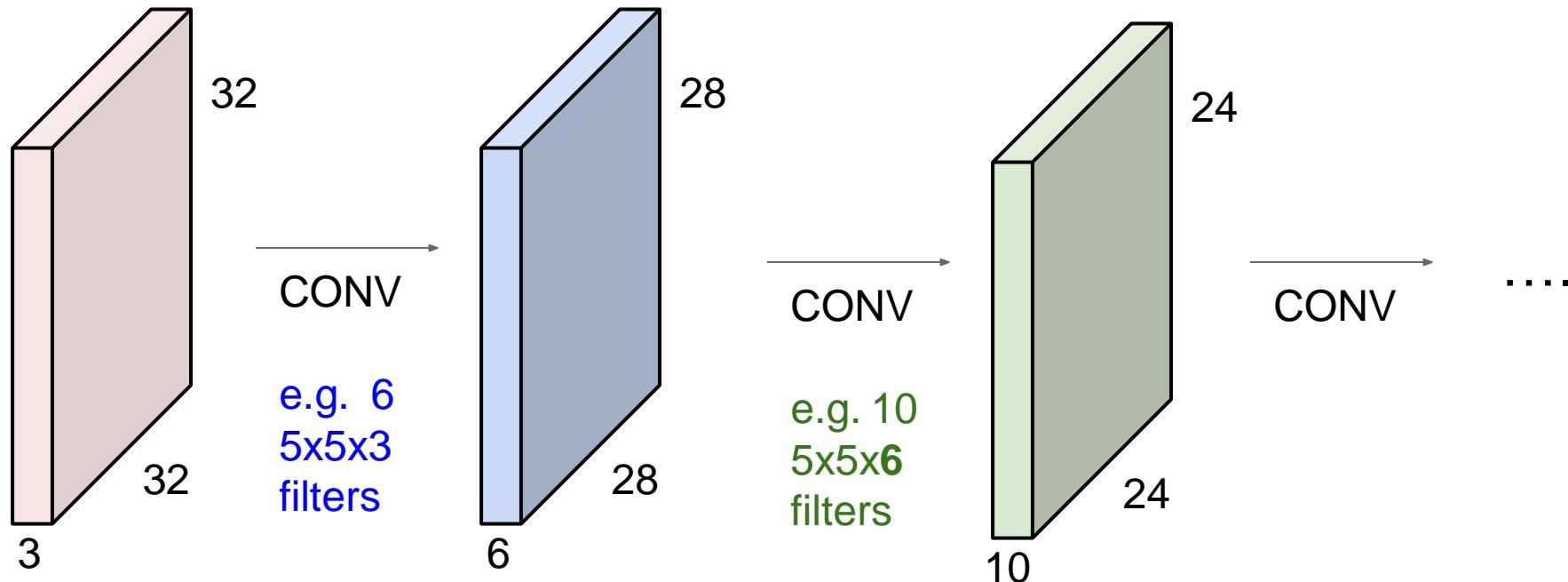
Slide inspiration: Justin Johnson

Image Classification with CNNs

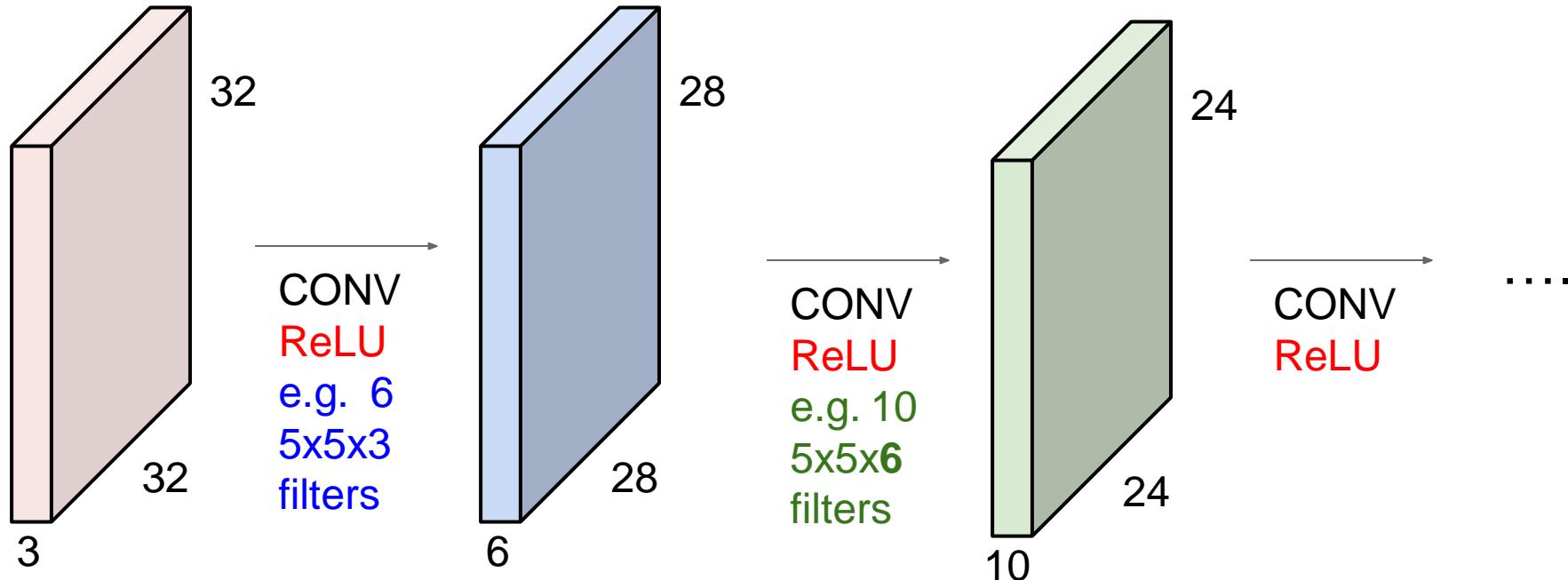
## Preview: ConvNet is a sequence of Convolution Layers



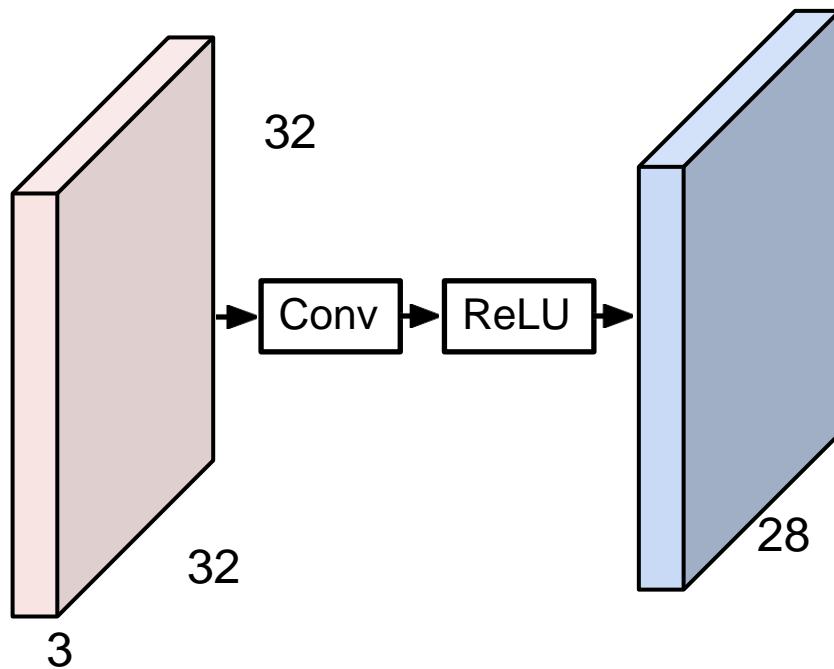
## Preview: ConvNet is a sequence of Convolution Layers



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



## Preview: What do convolutional filters learn?

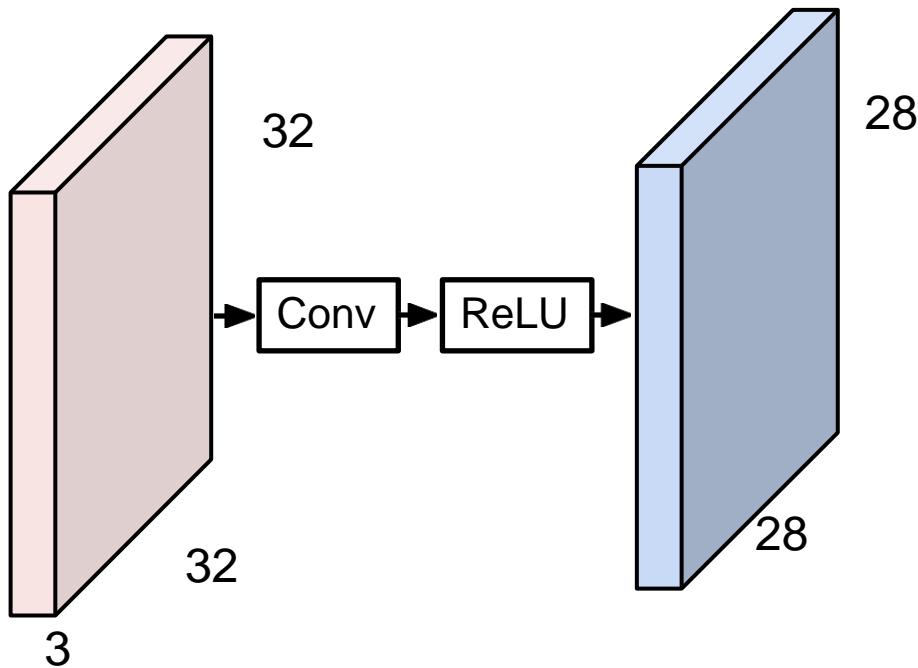


28

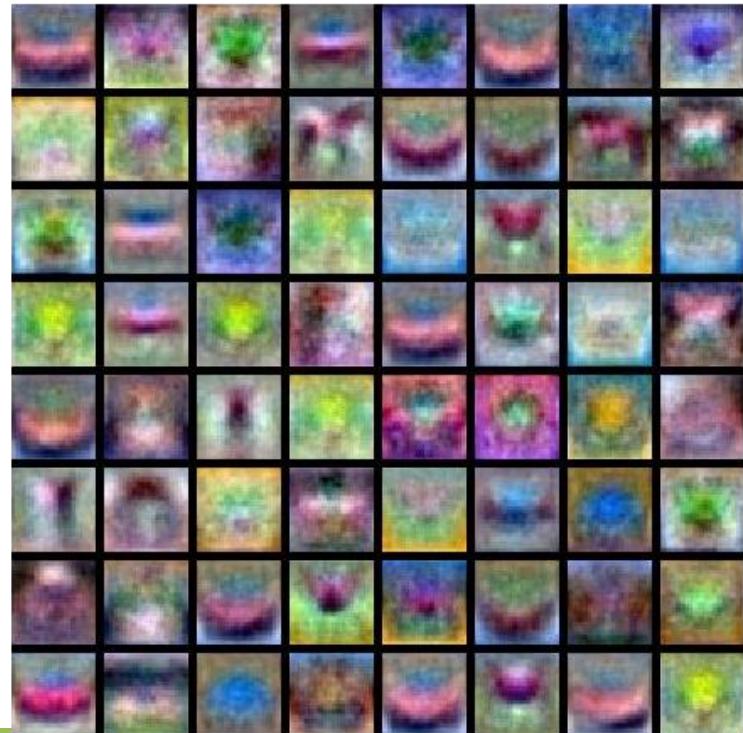
Linear classifier: One template per class



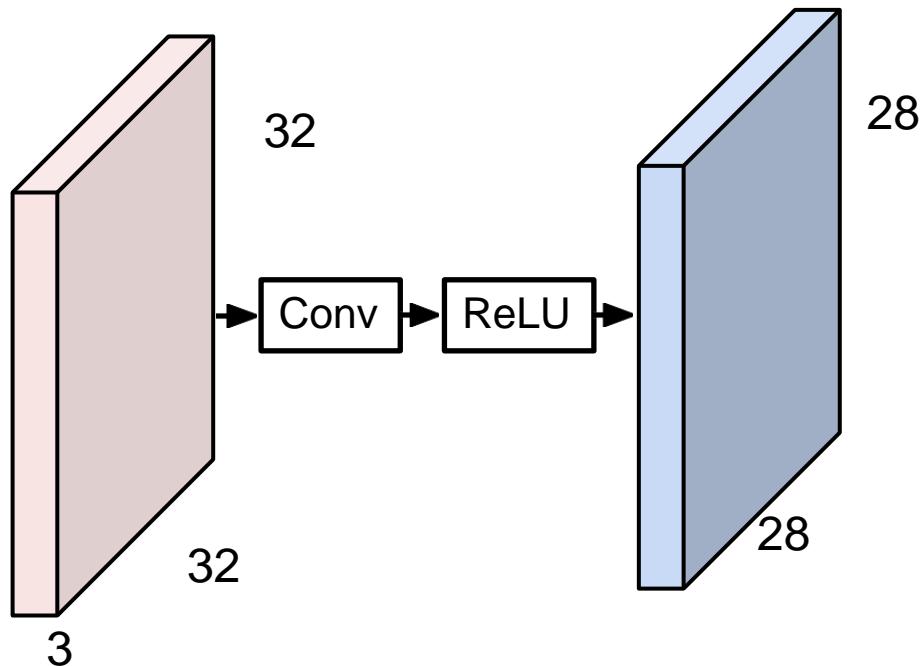
## Preview: What do convolutional filters learn?



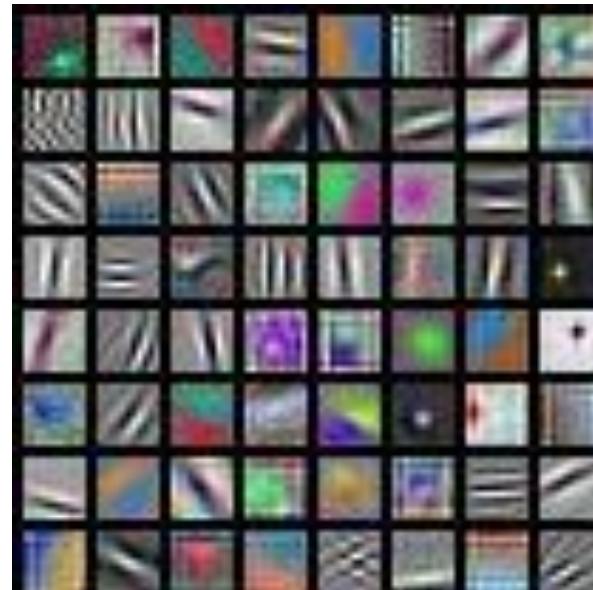
MLP: Bank of whole-image templates



## Preview: What do convolutional filters learn?



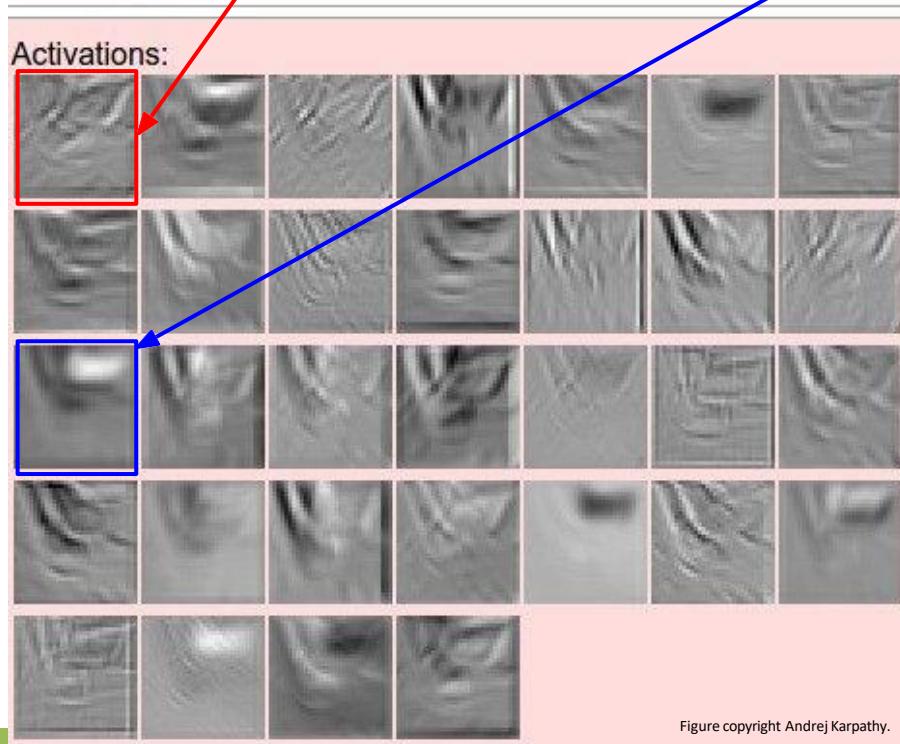
First-layer conv filters: local image templates  
(Often learns oriented edges, opposing colors)



AlexNet: 64 filters, each 3x11x11



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:

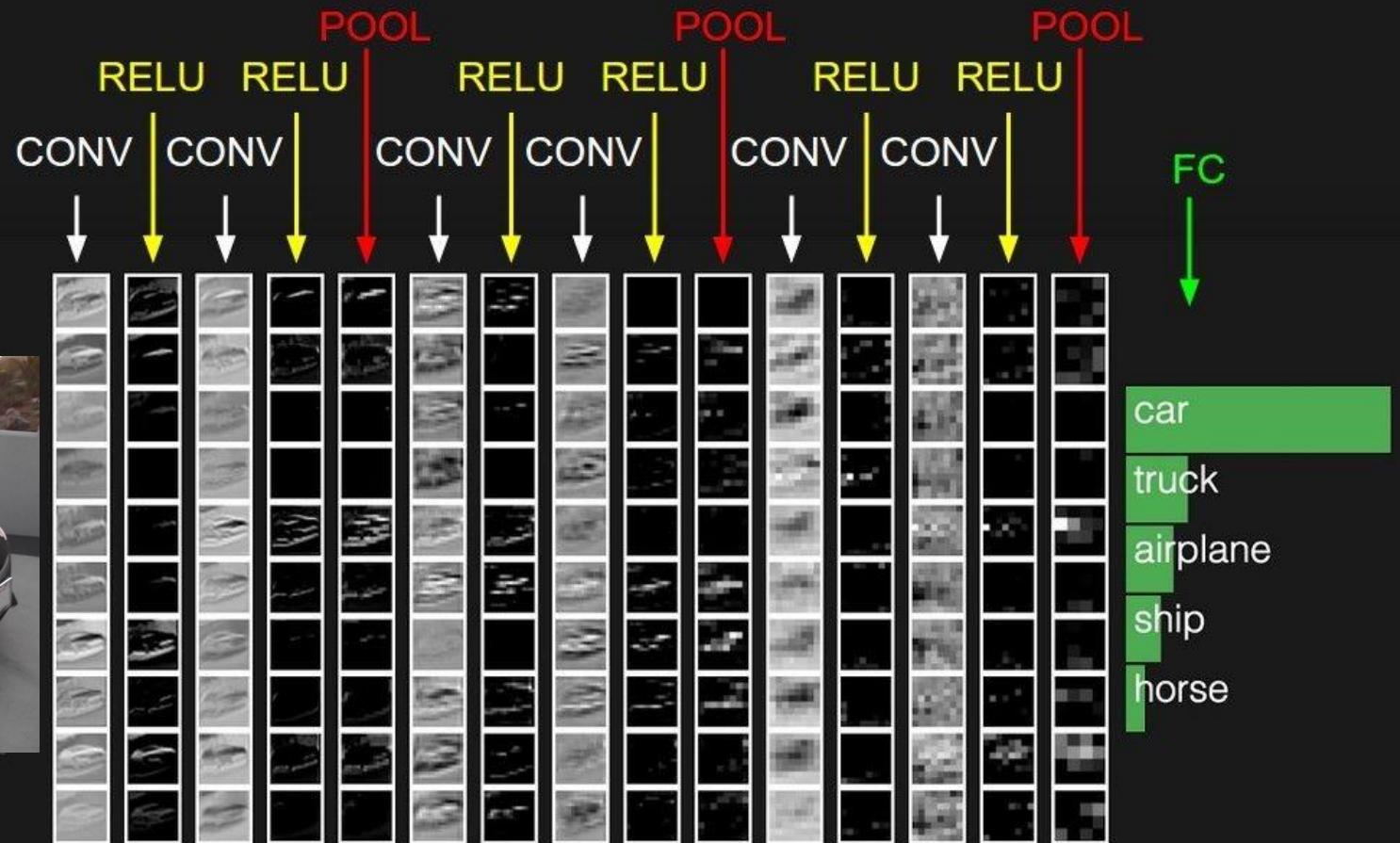
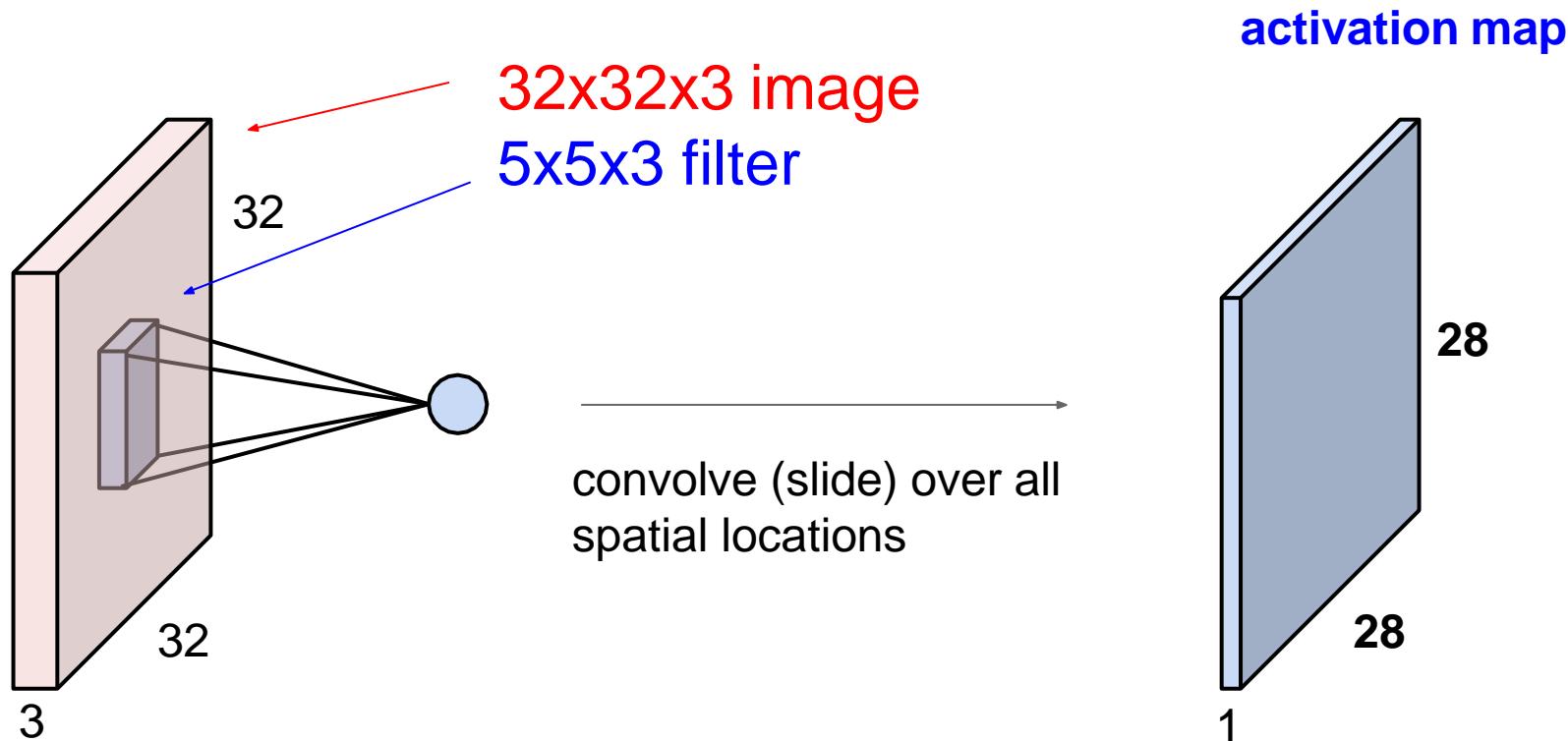


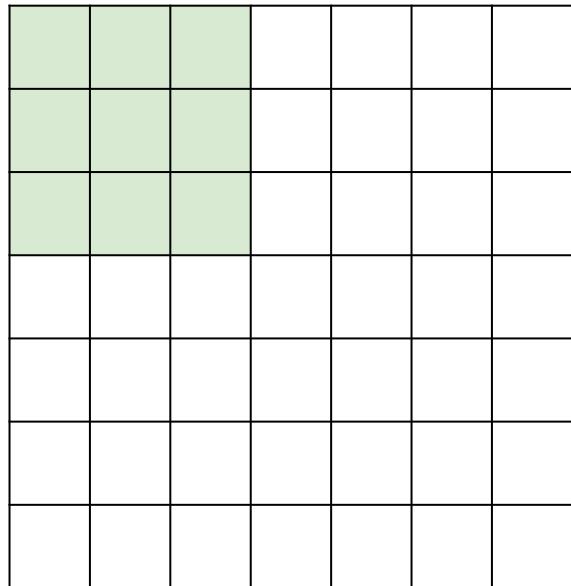
Image Classification with C N N s

## A closer look at spatial dimensions:



## A closer look at spatial dimensions:

7

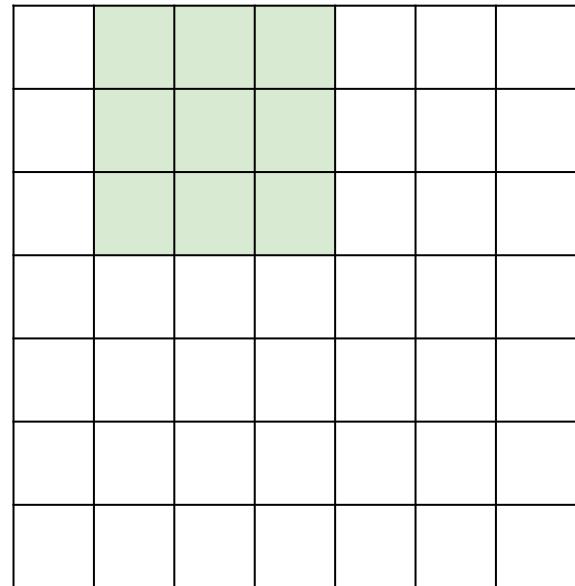


7

7x7 input (spatially)  
assume 3x3 filter

## A closer look at spatial dimensions:

7

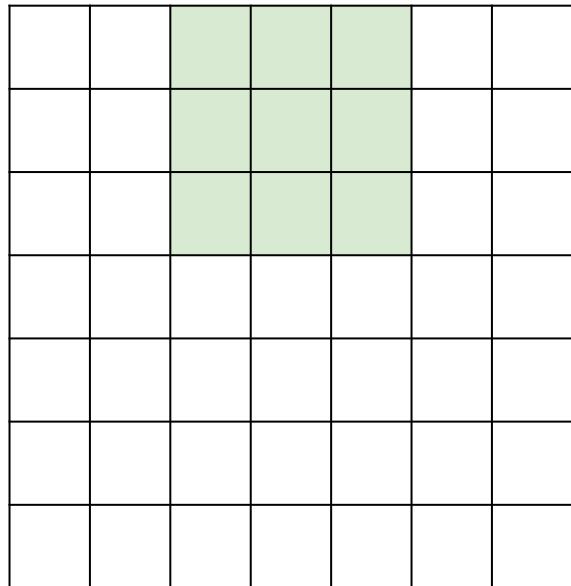


7

7x7 input (spatially)  
assume 3x3 filter

## A closer look at spatial dimensions:

7

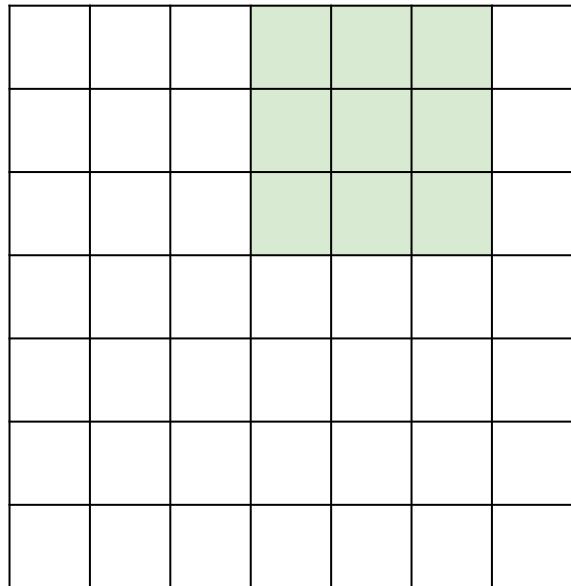


7x7 input (spatially)  
assume 3x3 filter

7

## A closer look at spatial dimensions:

7

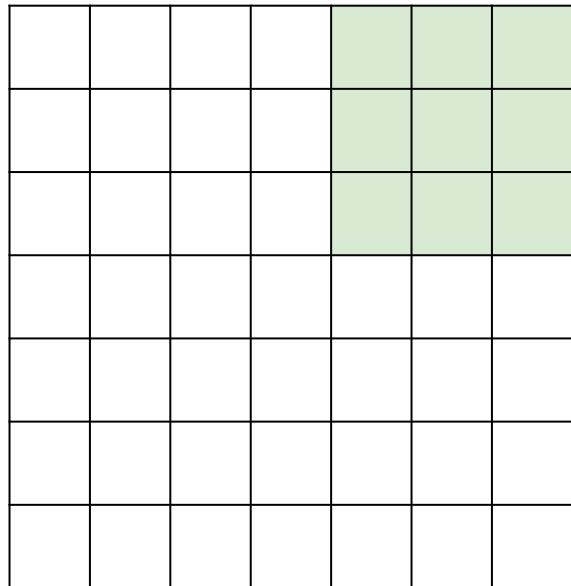


7x7 input (spatially)  
assume 3x3 filter

7

## A closer look at spatial dimensions:

7



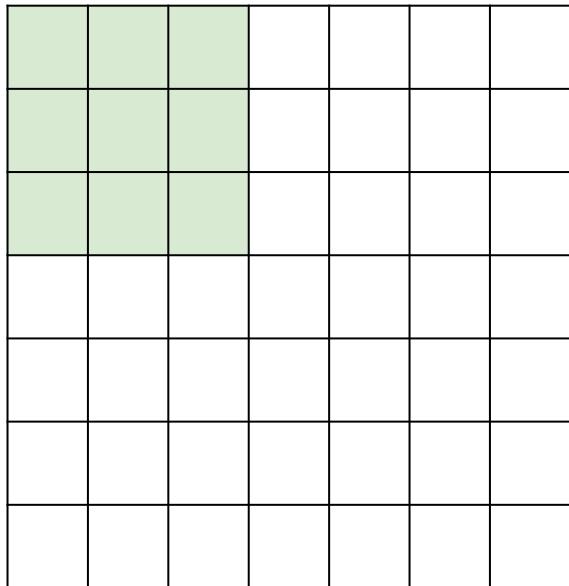
7x7 input (spatially)  
assume 3x3 filter

**=> 5x5 output**

7

## A closer look at spatial dimensions:

7

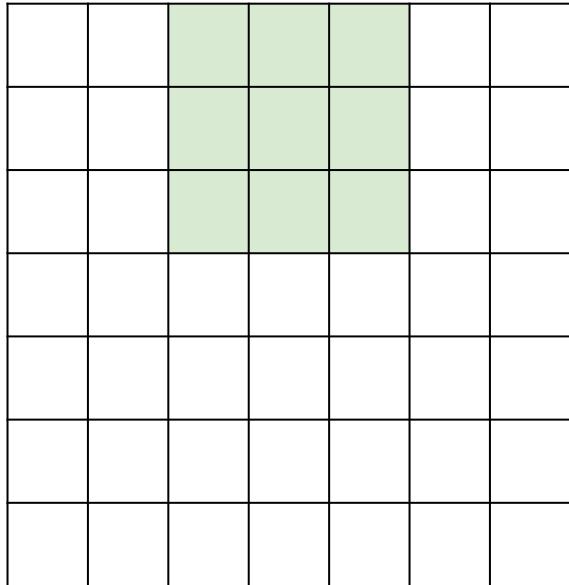


7

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

## A closer look at spatial dimensions:

7

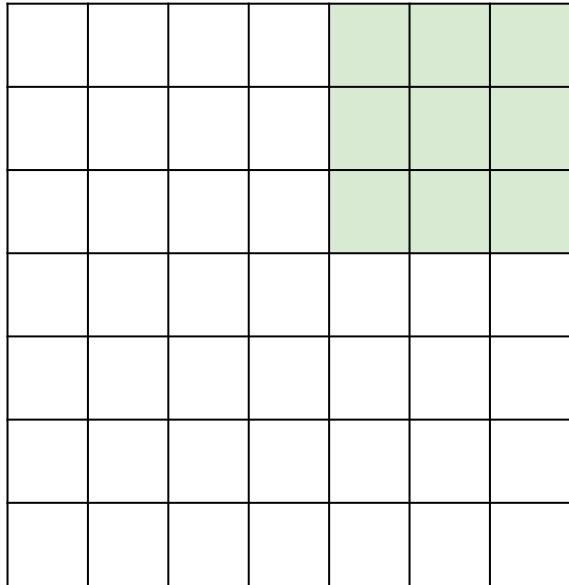


7

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

## A closer look at spatial dimensions:

7

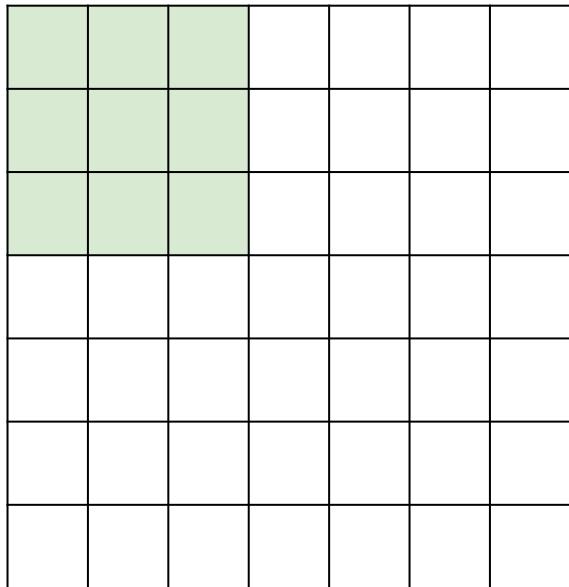


7

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

## A closer look at spatial dimensions:

7

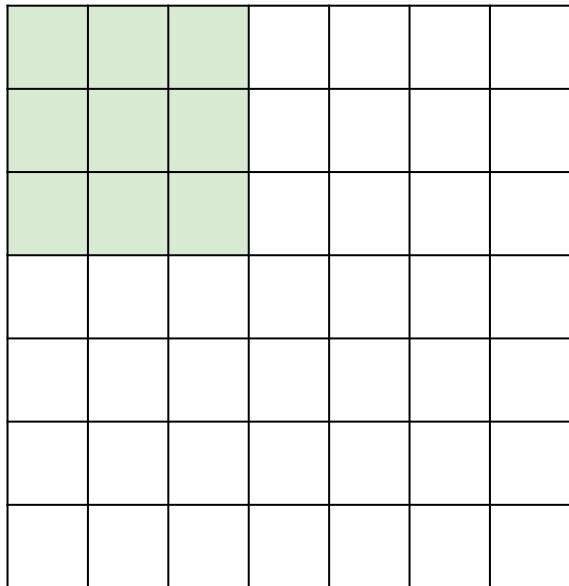


7

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

## A closer look at spatial dimensions:

7

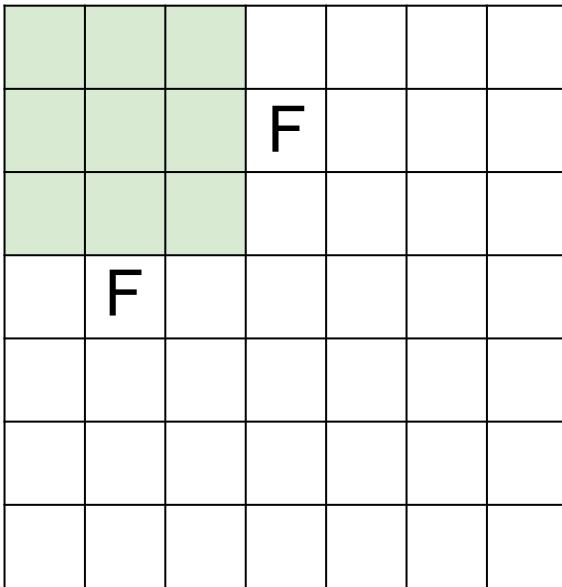


7

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

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

N



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!**

(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 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.

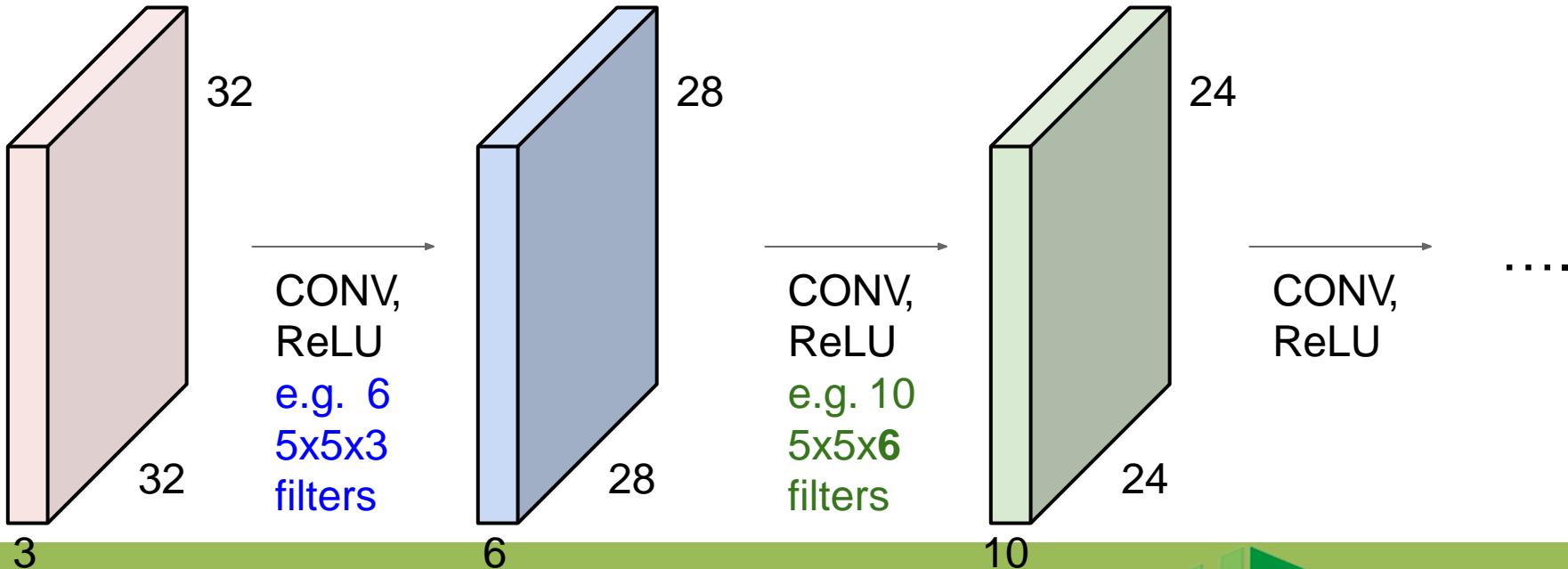


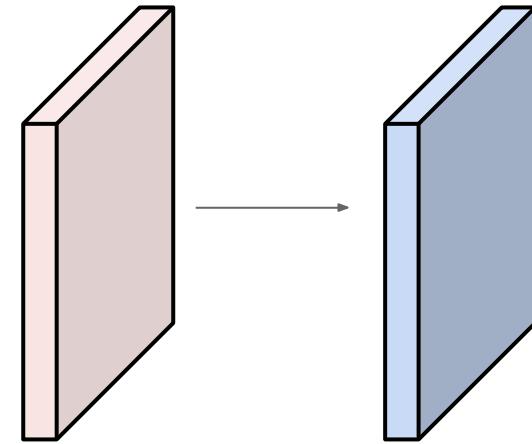
Image Classification with C N N s

# Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

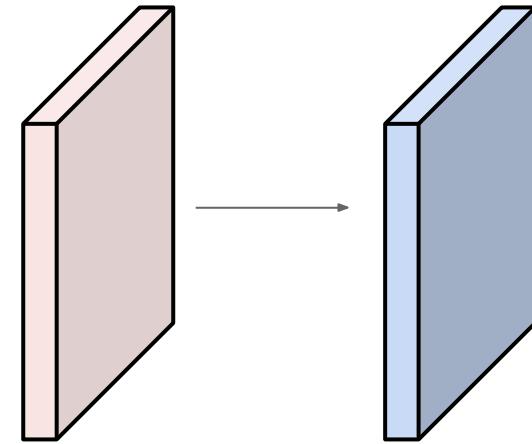
Output volume size: ?



# Examples time:

Input volume: **32x32x3**

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



Output volume size:

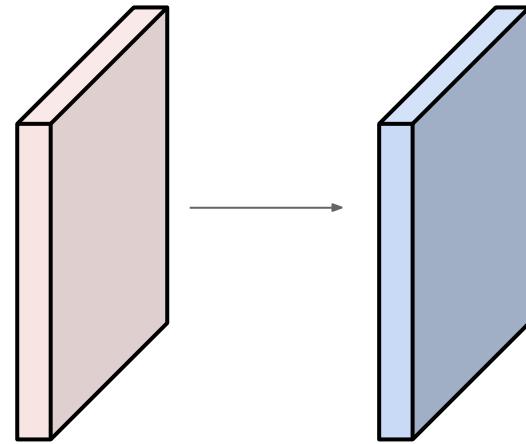
$(32+2*2-5)/1+1 = 32$  spatially, so

**32x32x10**

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

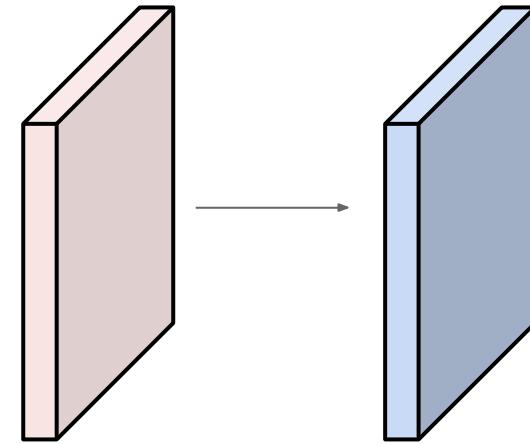


Number of parameters in this layer?

# Examples time:

Input volume: **32x32x3**

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

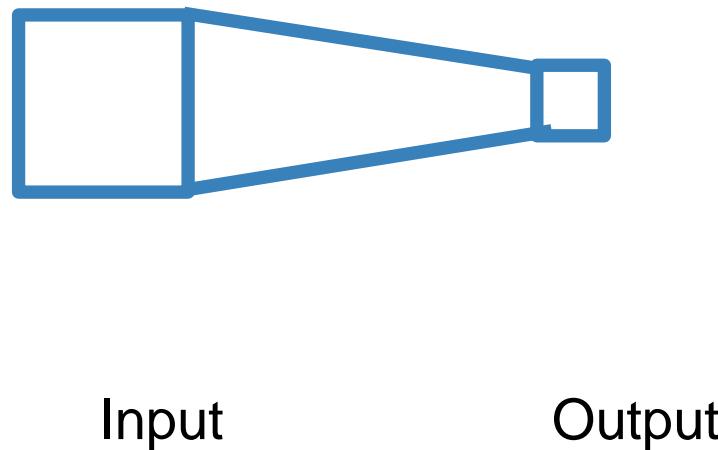


Number of parameters in this layer?

each filter has  $5*5*3 + 1 = 76$  params (+1 for bias)  
 $\Rightarrow 76*10 = 760$

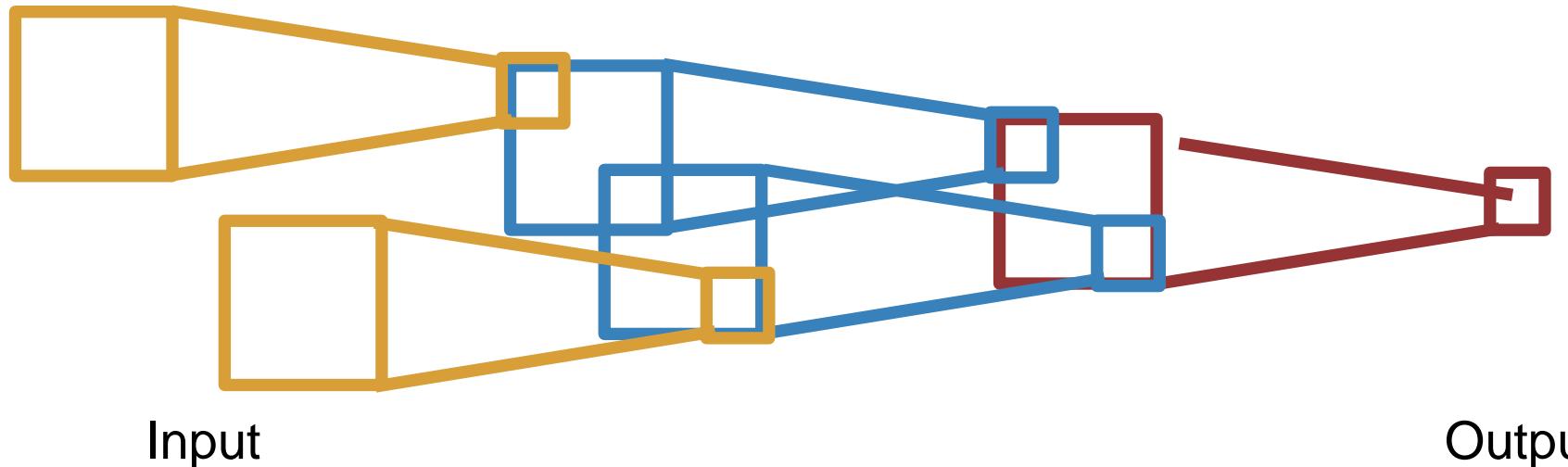
# Receptive Fields

For convolution with kernel size K, each element in the output depends on a  $K \times K$  **receptive field** in the input



# Receptive Fields

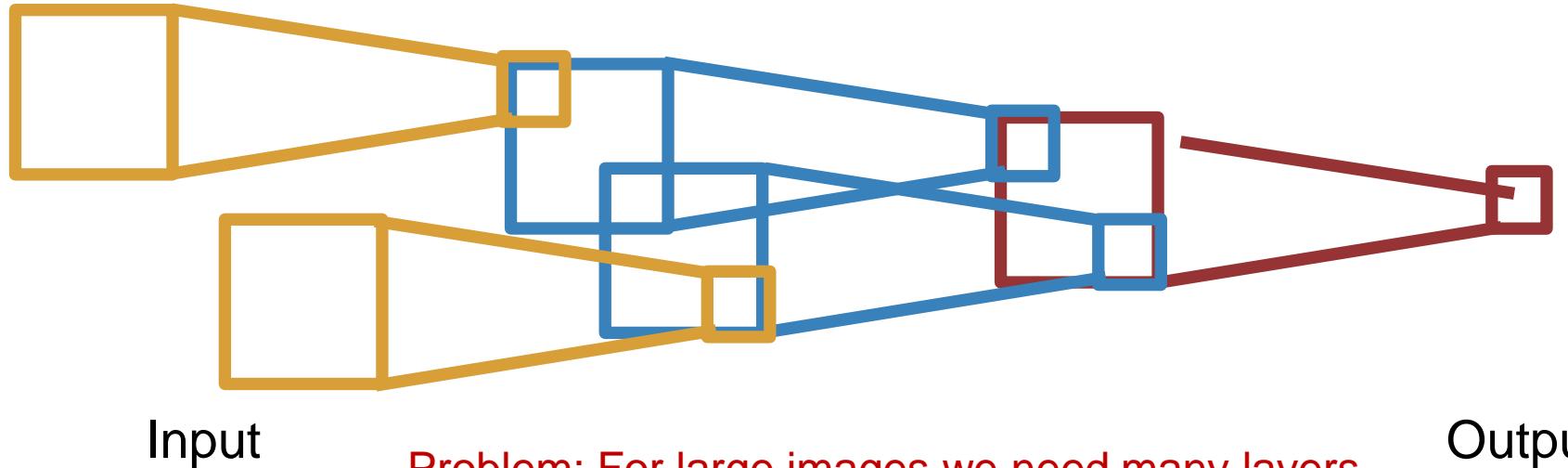
Each successive convolution adds  $K - 1$  to the receptive field size  
With  $L$  layers the receptive field size is  $1 + L * (K - 1)$



Be careful – “receptive field in the input” vs. “receptive field in the previous layer”

# Receptive Fields

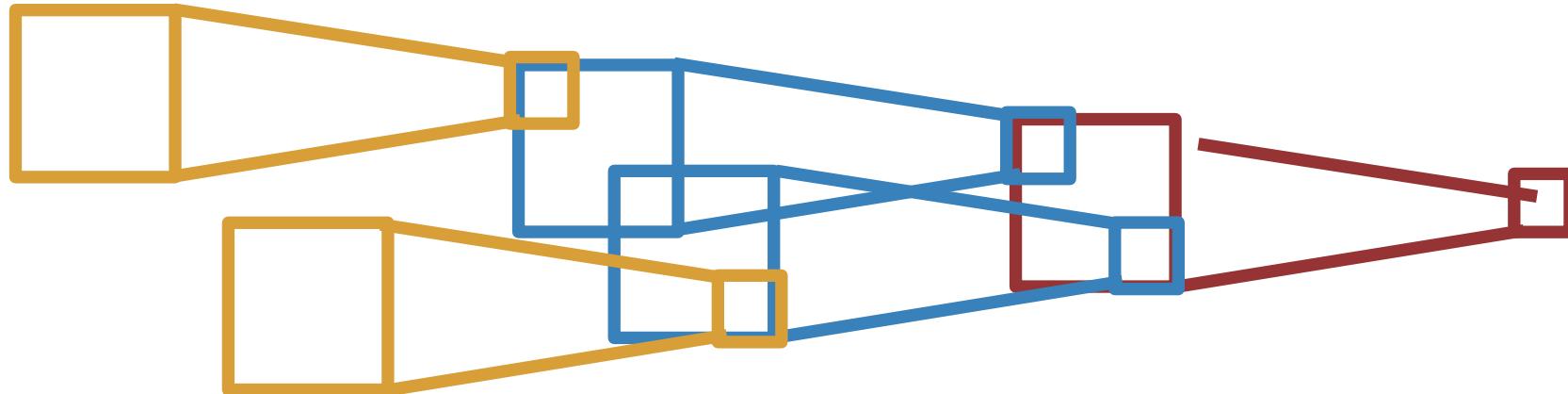
Each successive convolution adds  $K - 1$  to the receptive field size  
With  $L$  layers the receptive field size is  $1 + L * (K - 1)$



Problem: For large images we need many layers for each output to “see” the whole image

# Receptive Fields

Each successive convolution adds  $K - 1$  to the receptive field size  
With  $L$  layers the receptive field size is  $1 + L * (K - 1)$



Input

Problem: For large images we need many layers  
for each output to “see” the whole image

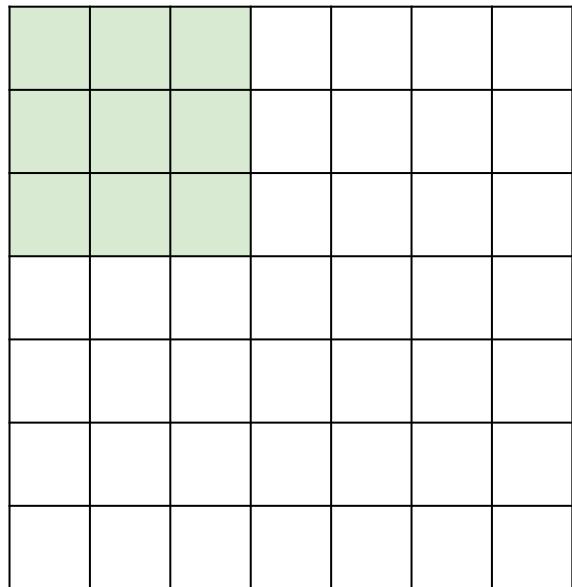
Output

Solution: Downsample inside the network

Image Classification with C N N s

# Solution: Strided Convolution

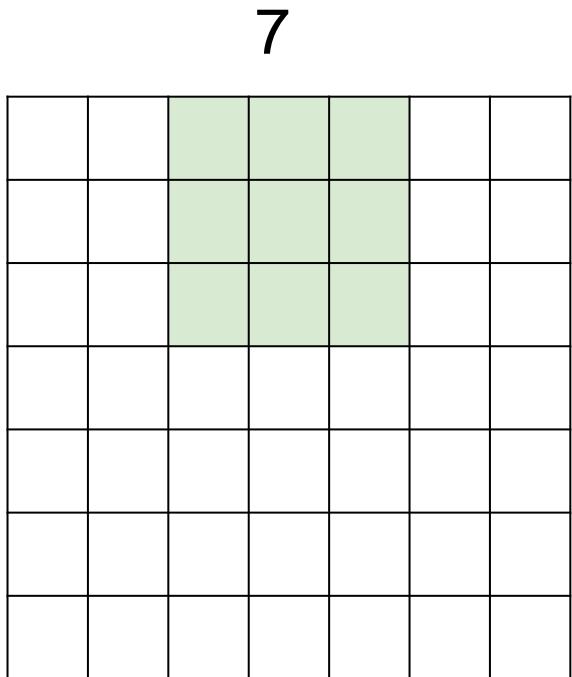
7



7

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

# Solution: Strided Convolution



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

=> 3x3 output!

# 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  
Image Classification with CNNs

# Convolution layer: summary

Common settings:

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**

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

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

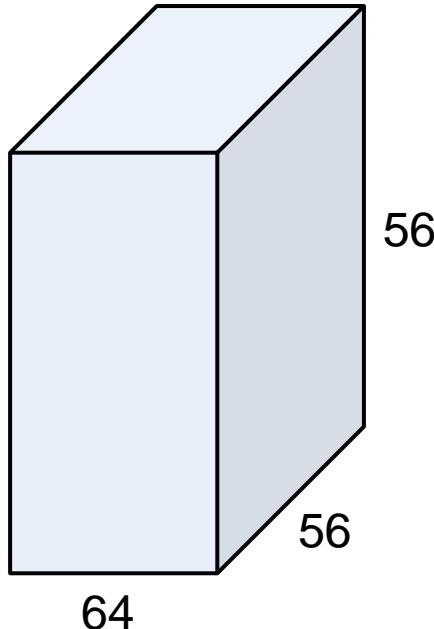
where:

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

Number of parameters:  $F^2CK$  and  $K$  biases

Image Classification with CNNs

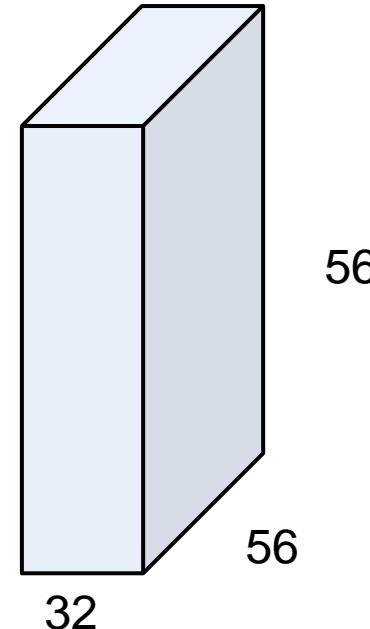
(btw, 1x1 convolution layers make perfect sense)



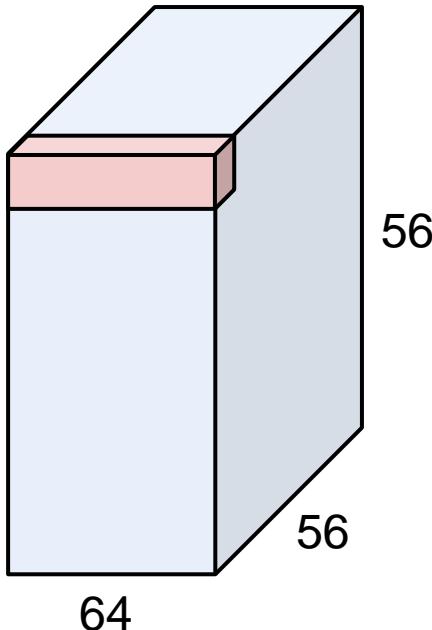
1x1 CONV  
with 32 filters

---

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



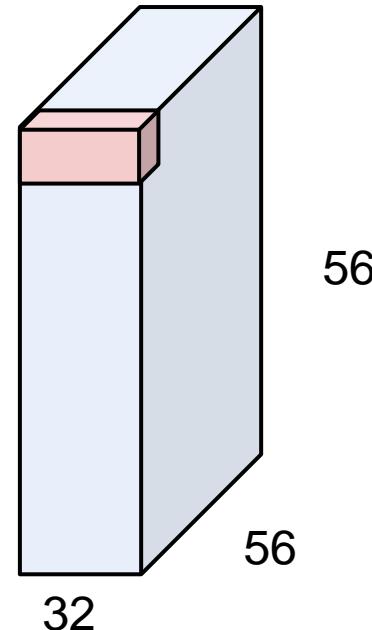
(btw, 1x1 convolution layers make perfect sense)



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.

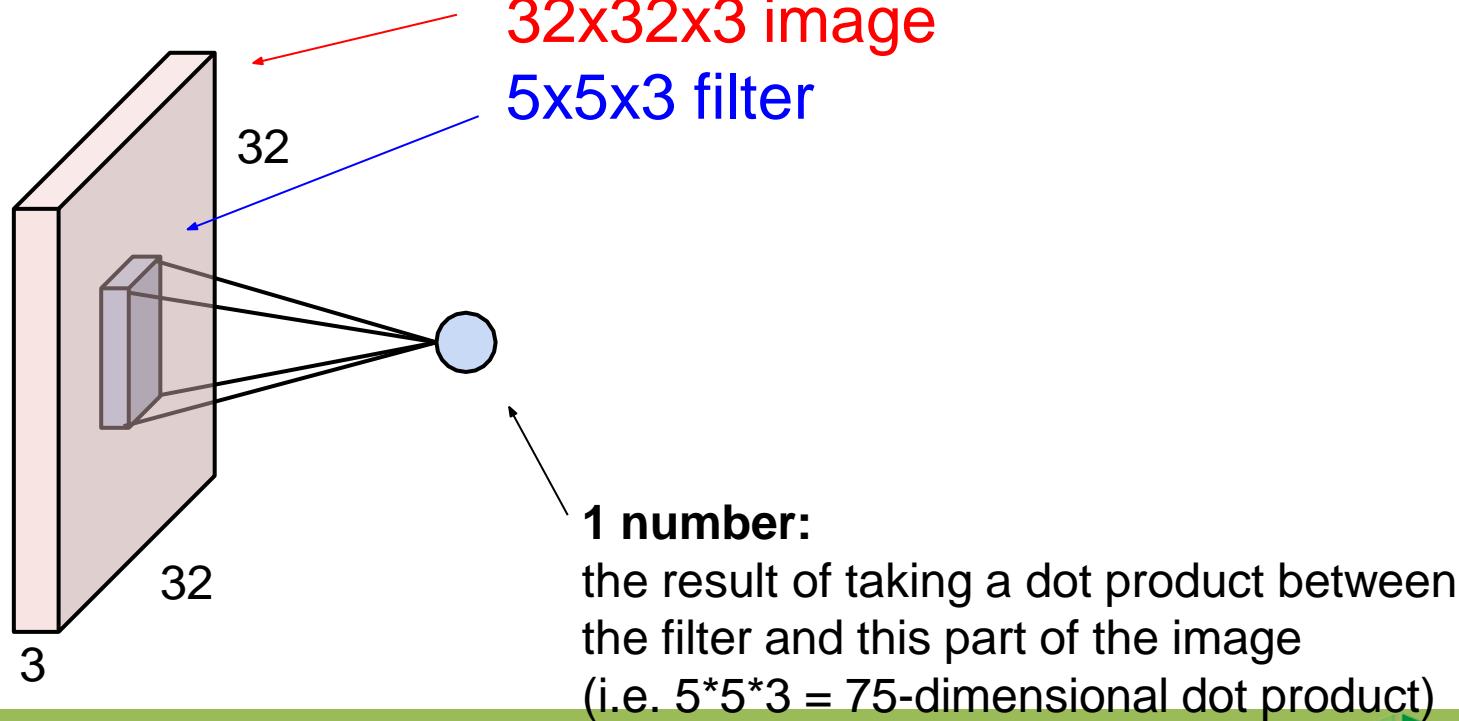
- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. 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:

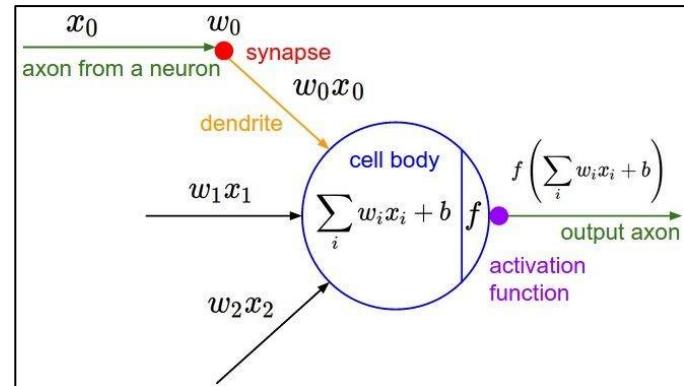
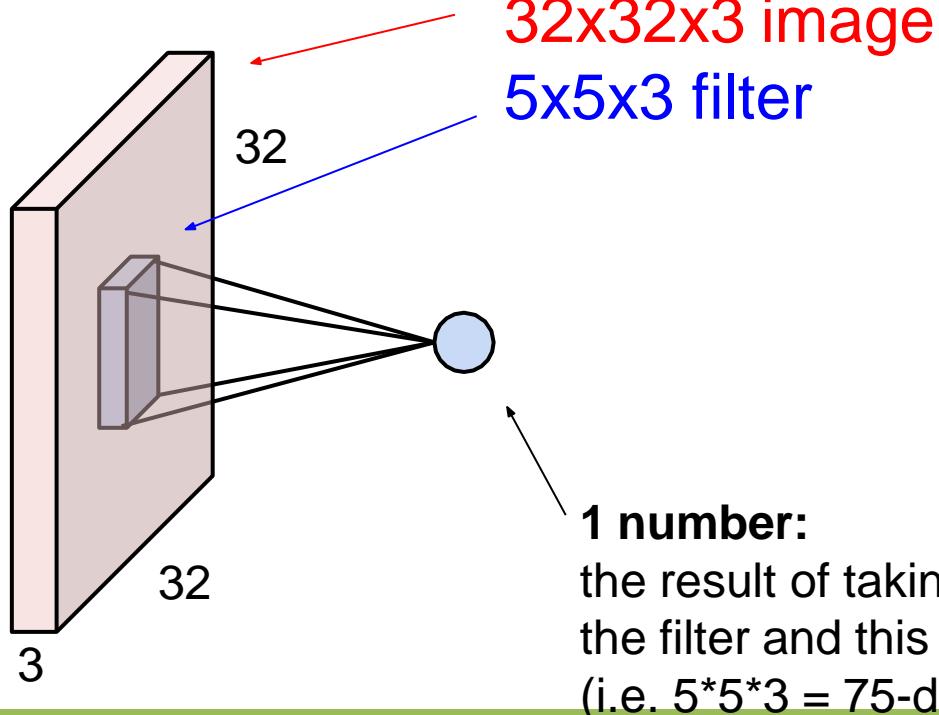
- a single `int` – in which case the same value is used for the height and width dimension
- a `tuple` of two `ints` – in which case, the first `int` is used for the height dimension, and the second `int` for the width dimension

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

# The brain/neuron view of CONV Layer



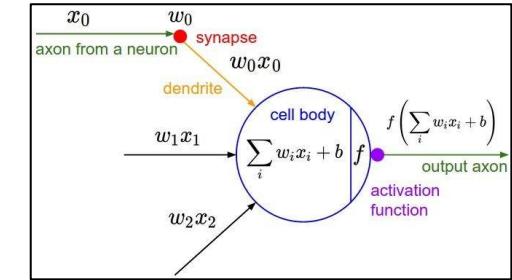
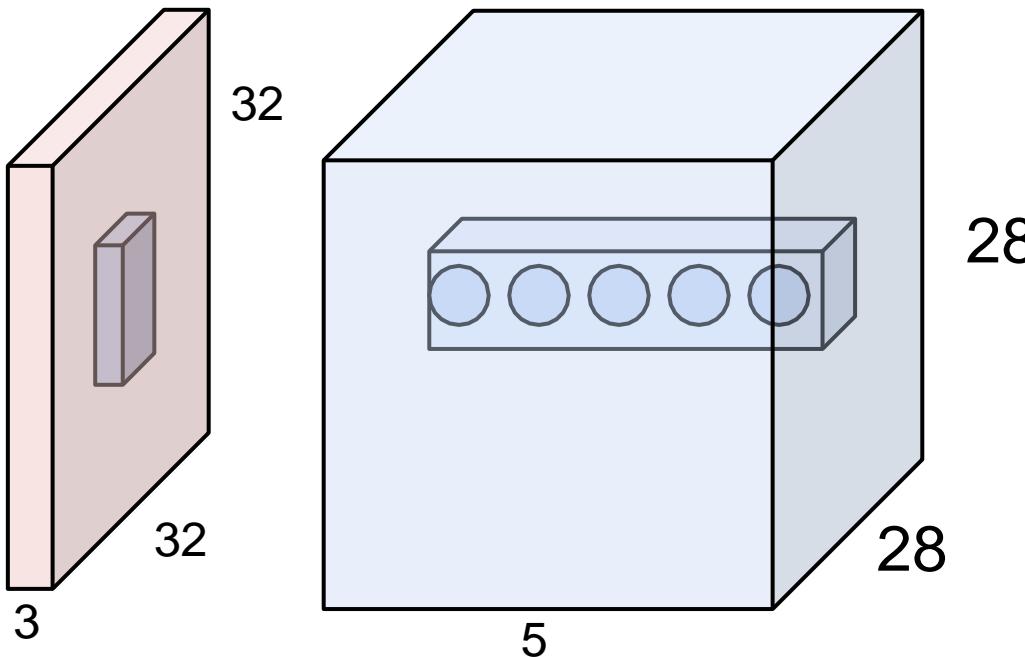
# The brain/neuron view of CONV Layer



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

Image Classification with C N N s

# The brain/neuron view of CONV Layer



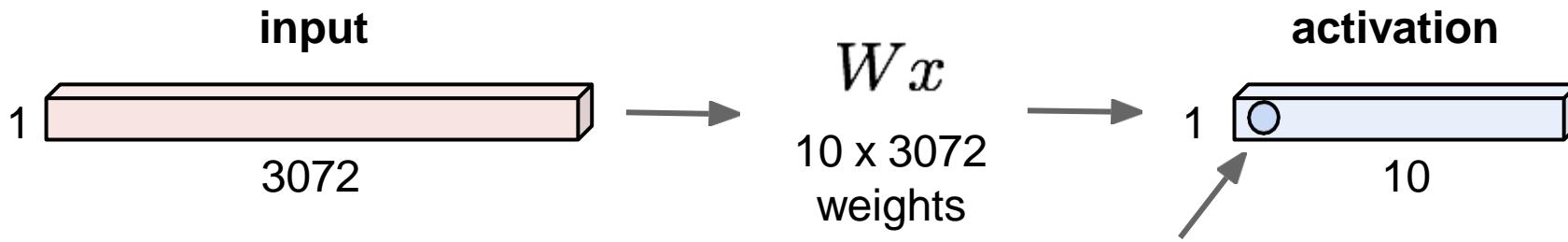
E.g. with 5 filters,  
CONV layer consists of  
neurons arranged in a 3D grid  
( $28 \times 28 \times 5$ )

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 x 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)

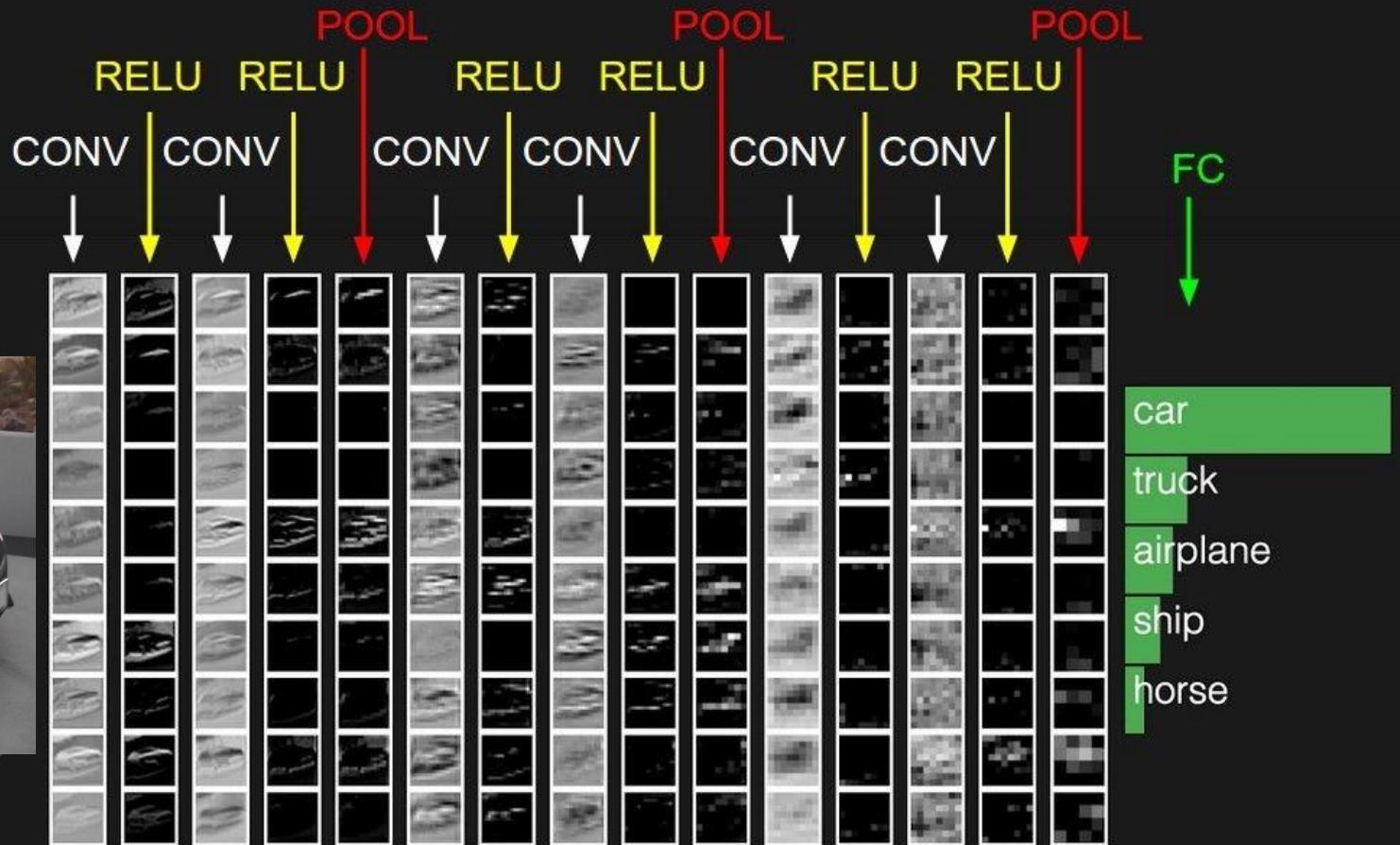
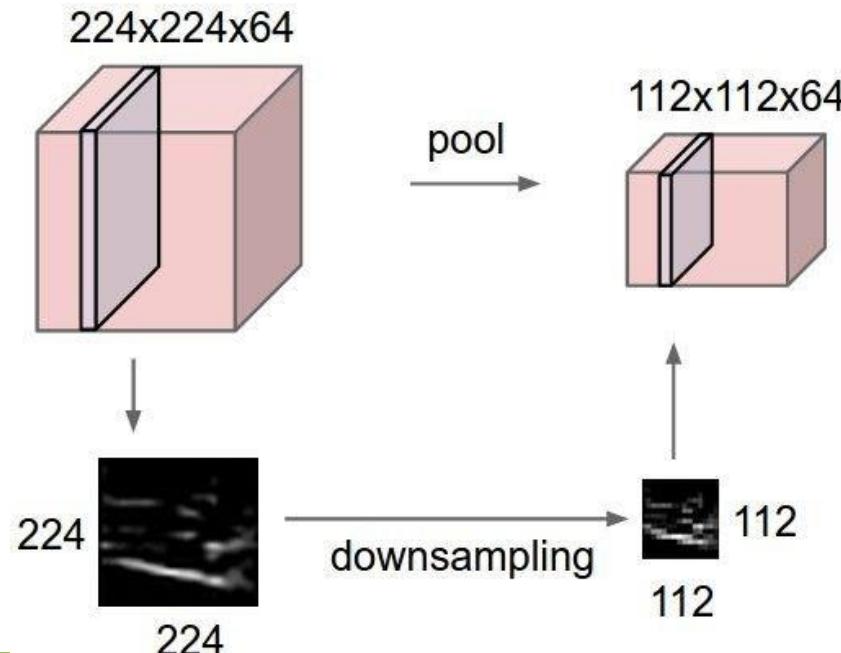


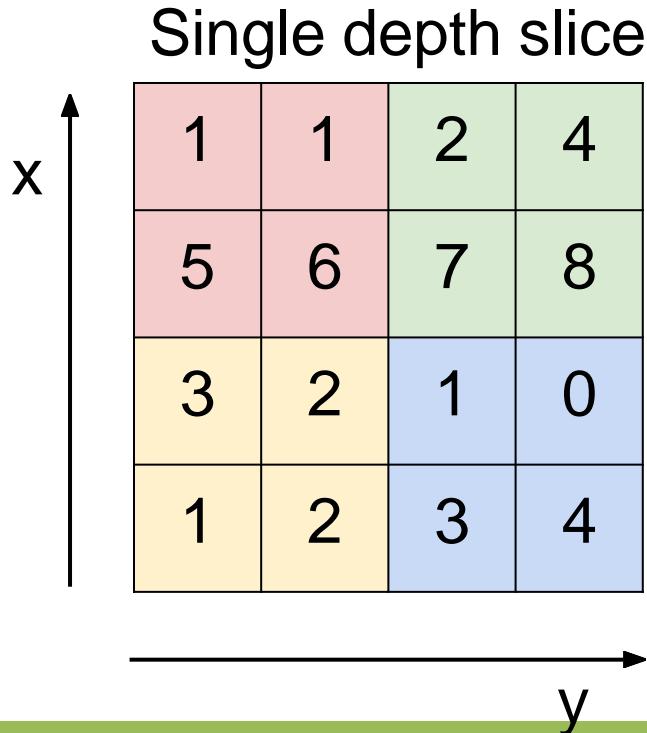
Image Classification with C N N s

# Pooling layer

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



# MAX POOLING

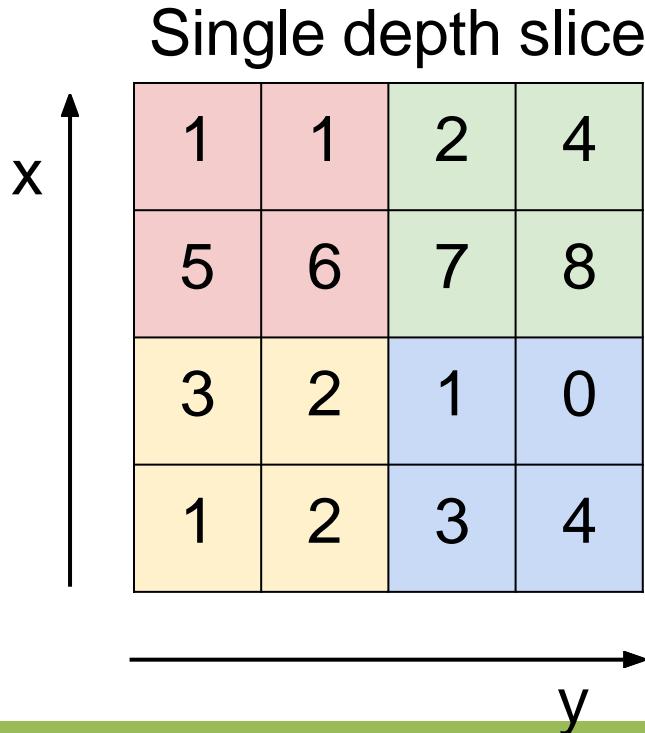


max pool with 2x2 filters  
and stride 2

The output matrix has the following values:

6	8
3	4

# MAX POOLING



max pool with 2x2 filters  
and stride 2

6	8
3	4

- No learnable parameters
- Introduces spatial invariance

# 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

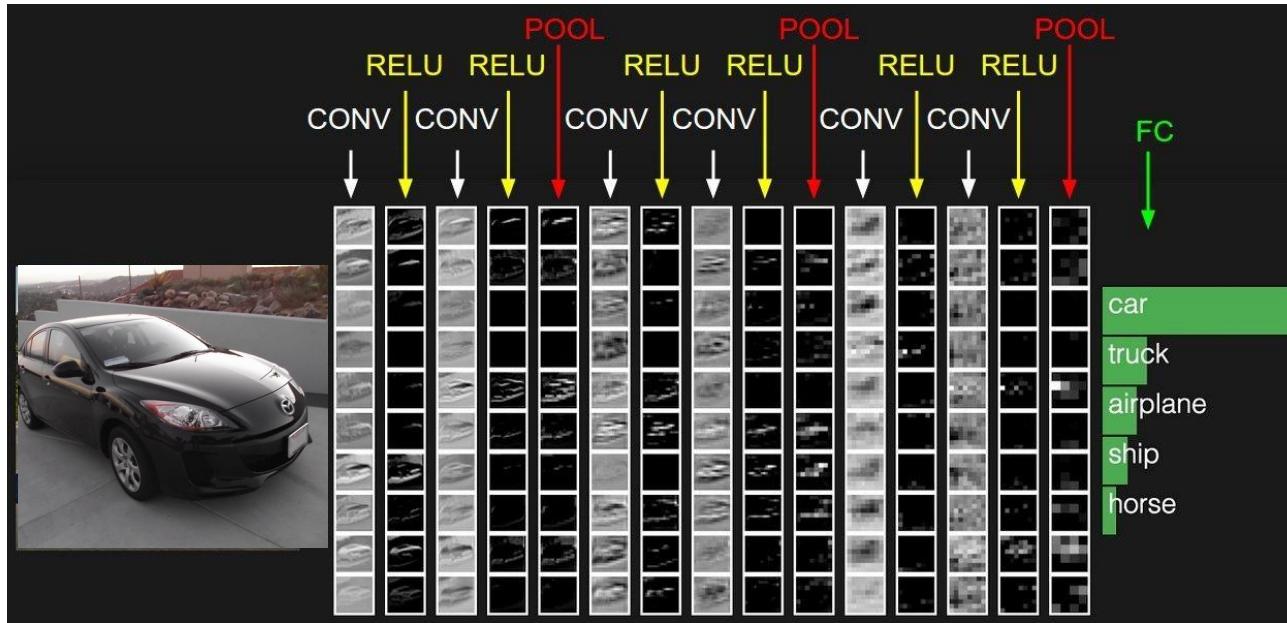


Image Classification with C N N s

# [ConvNetJS demo: training on CIFAR-10]

## ConvNetJS CIFAR-10 demo

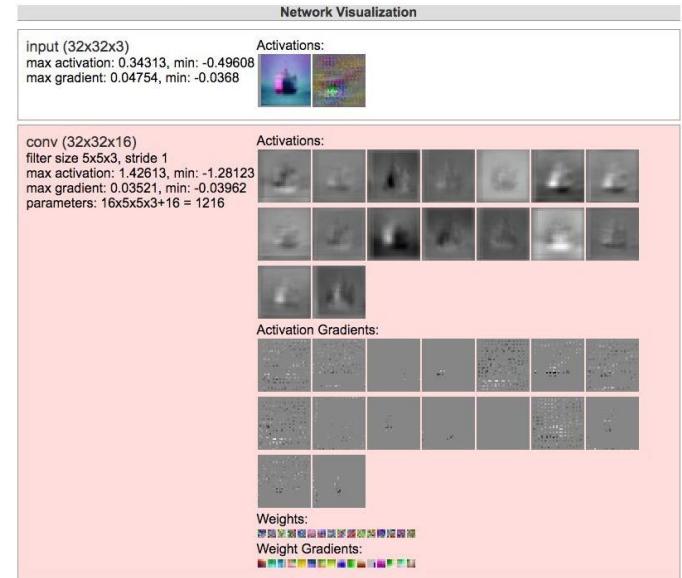
### Description

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

# Next time: CNN Architectures

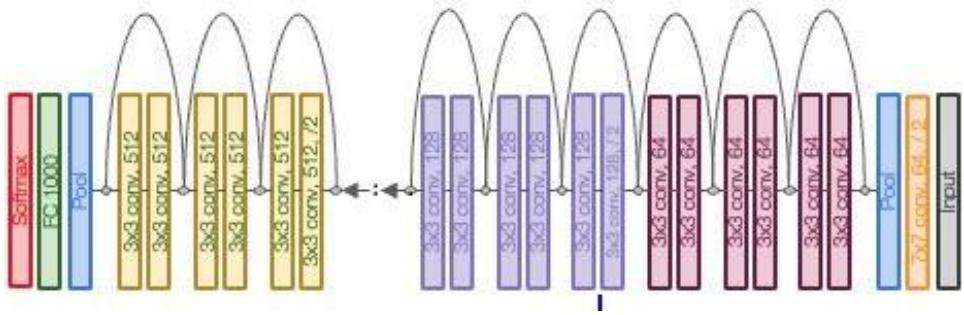
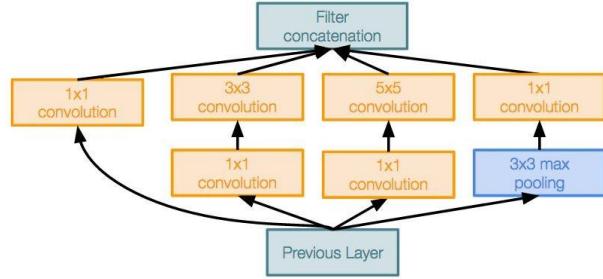
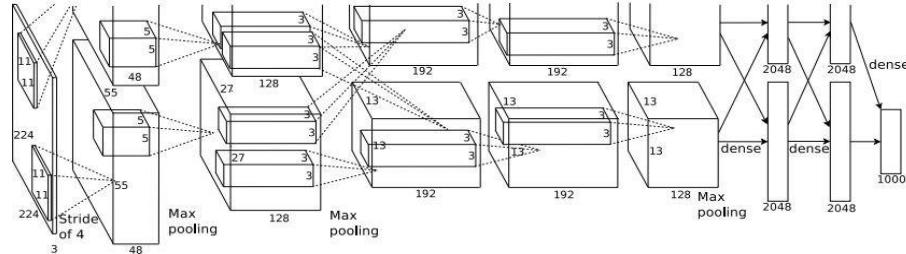
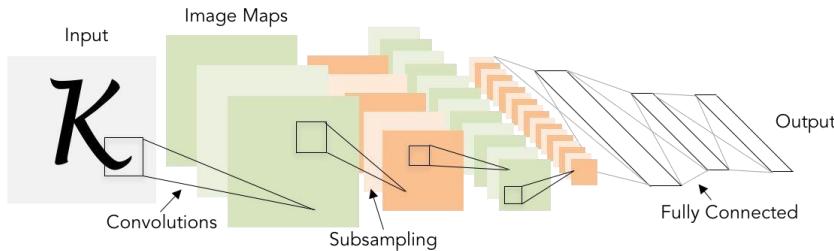


Image Classification with CNNs