CS 4644-DL / 7643-A: LECTURE 7
DANFEI XU

Topics:

- Convolutional Neural Networks: Past and Present
- Convolution Layers

Administrative:

- Assignment due on Sep 19th (with 48hr grace period)
- Will release proposal template today
- Proposal due Sep 26th 11:59pm (**No Grace Period**)
- Start finding a project team if you haven't!

Recap: Vector derivatives

Scalar to Scalar

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

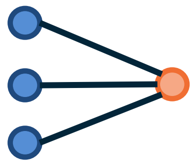
Vector to Scalar

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left(\frac{\partial y}{\partial x}\right)_n = \frac{\partial y}{\partial x_n}$$

For each element of x, if it changes by a small amount, how much will y change?



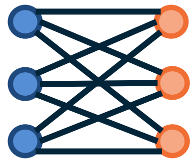
Vector to Vector

$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

Derivative is **Jacobian**:

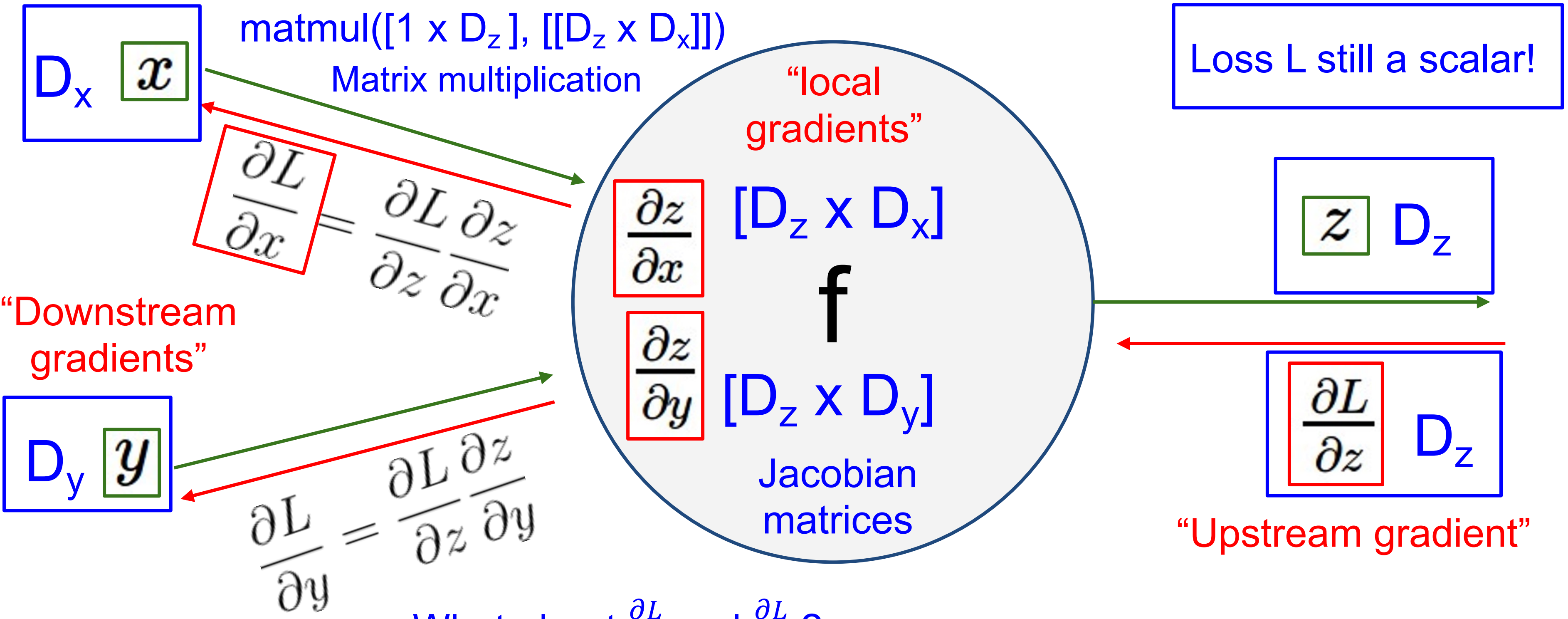
$$\frac{\partial y}{\partial x} \in \mathbb{R}^{M \times N} \quad \left(\frac{\partial y}{\partial x}\right)_{n,m} = \frac{\partial y_n}{\partial x_m}$$

For each element of x, if it changes by a small amount, how much will **each element** of y change?



Slide credit: Stanford CS231n Instructors

Backprop with Vectors



What about $\frac{\partial L}{\partial x}$ and $\frac{\partial L}{\partial y}$?

Slide credit: Stanford CS231n Instructors

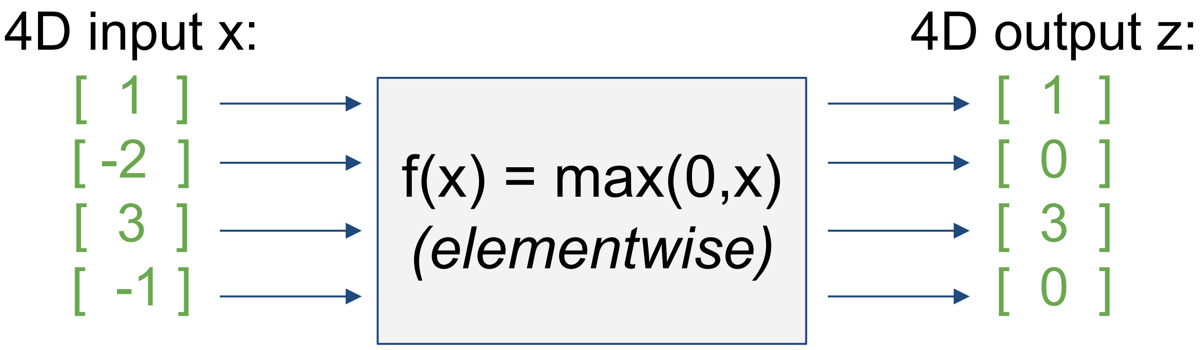
Jacobians

Given a function $f: \mathbb{R}^n \to \mathbb{R}^m$, we have the Jacobian matrix $\mathbf{J}$ of shape $\boldsymbol{m} \times \boldsymbol{n}$, where $\mathbf{J}_{i,j} = \frac{\partial f_i}{\partial x_j}$

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathrm{T}} f_1 \\ \vdots \\ \nabla^{\mathrm{T}} f_m \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

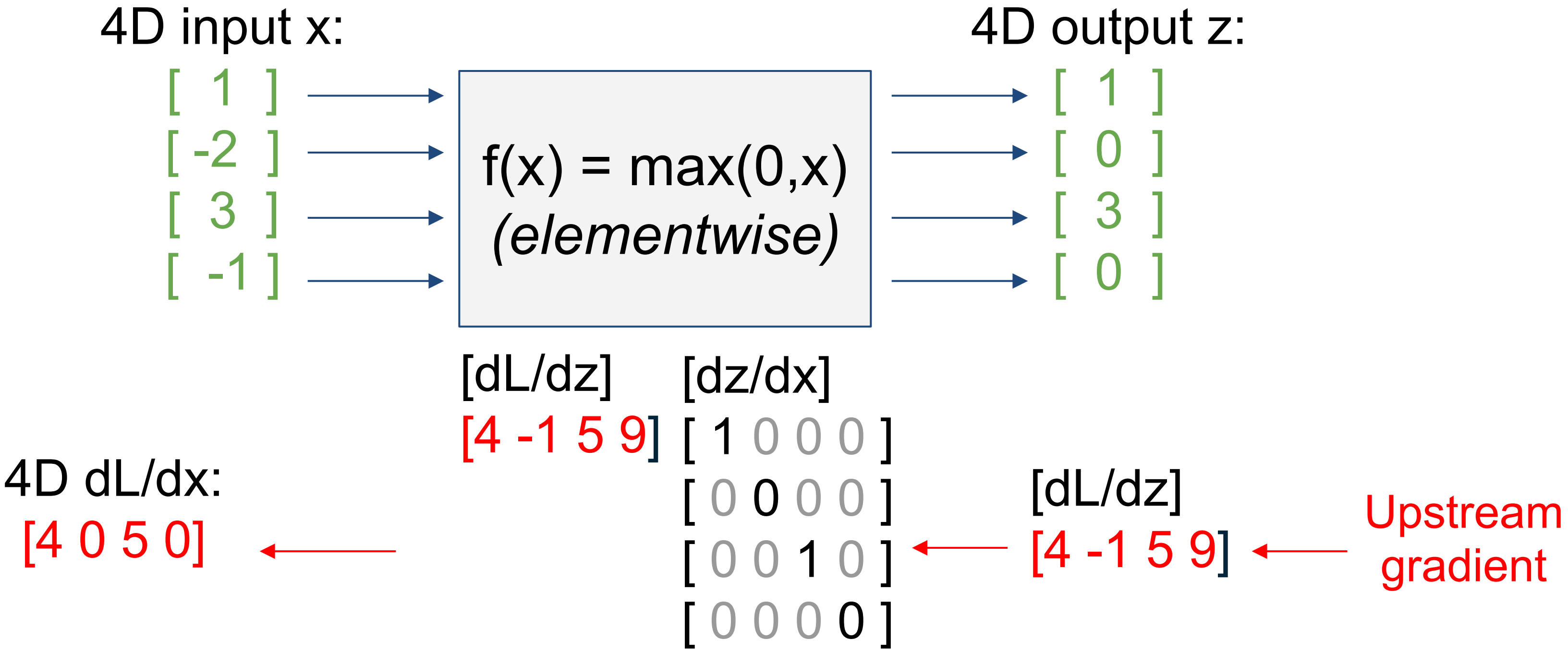
Figure source: https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant

Backprop with Vectors



Slide credit: Stanford CS231n Instructors

Backprop with Vectors



Slide credit: Stanford CS231n Instructors

Backprop with Vectors

For element-wise ops, jacobian is **sparse**: off-diagonal entries always zero! Never **explicitly** form Jacobian -- instead use Hadamard (element-wise) multiplication

4D input x:

[1]
[-2]
[3]
[-1]

f(x) = max(0,x)
(elementwise)

4D output z:

[1]
[0]
[3]
[0]

4D dL/dx:
[4 0 5 0]

$$\left(\frac{\partial L}{\partial x}\right)_i = \begin{cases} \left(\frac{\partial L}{\partial z}\right)_i & \text{if } x_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

[dL/dz]
[4 -1 5 9]

Upstream gradient

Slide credit: Stanford CS231n Instructors

Backprop with Matrices (or Tensors)

Loss L still a scalar!

dL/dx always has the same shape as x!

$[D_x \times M_x]$ $\boxed{x}$

$[D_x \times M_x]$ $\boxed{\frac{\partial L}{\partial x}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$

"Downstream gradients"

$[D_y \times M_y]$ $\boxed{y}$

$[D_y \times M_y]$ $\boxed{\frac{\partial L}{\partial y}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$

"local gradients"

$\boxed{\frac{\partial z}{\partial x}}$ $[(D_z \times M_z)x(D_x \times M_x)]$

$\boxed{\frac{\partial z}{\partial y}}$ $[(D_z \times M_z)\times(D_y \times M_y)]$

Jacobian matrices

$\boxed{z}$ $[D_z \times M_z]$

$\boxed{\frac{\partial L}{\partial z}}$ $[D_z \times M_z]$

"Upstream gradient"

For each element of z, how much does it influence L?

Flatten the two matrices -> vector-vector gradients -> jacobian matrices!

Slide credit: Stanford CS231n Instructors

Summary (Lecture 5 – here):

- Neural networks, activation functions
- NNs as Universal Function Approximators
- Neurons as biological inspirations to DNNs
- Vector Calculus
- Backpropagation through vectors / matrices

Next: Convolutional Neural Networks

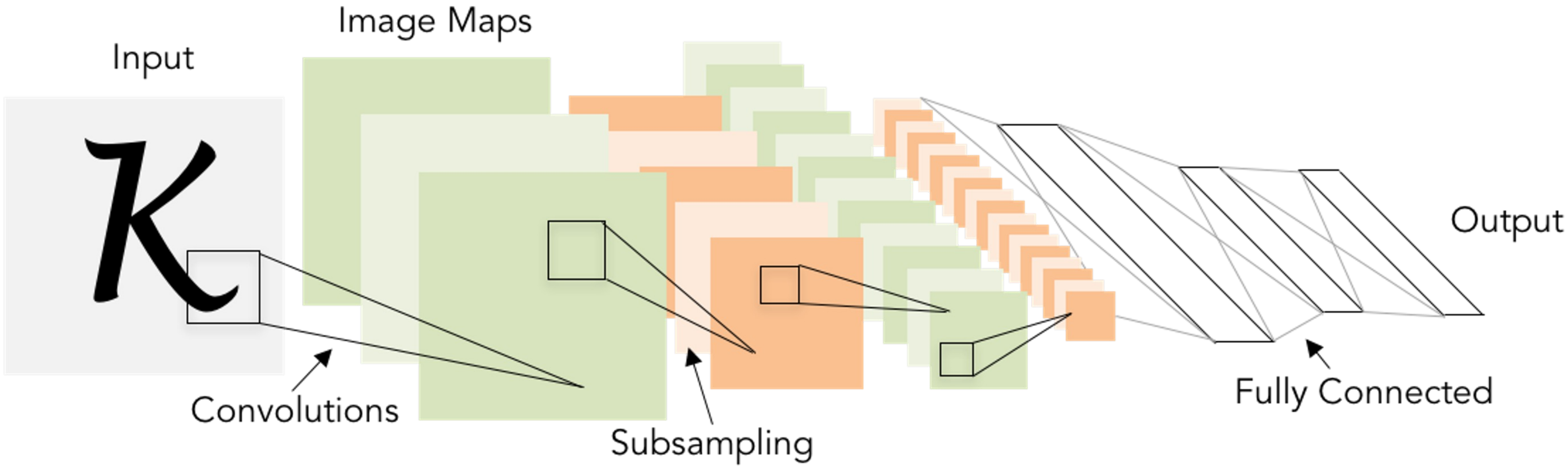


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Slide credit: Stanford CS231n Instructors

A bit of history...

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

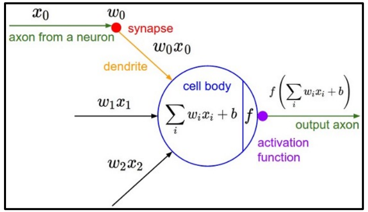
The machine was connected to a camera that used 20×20 photocells to produce a 400-pixel image.

recognized
letters of the alphabet

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

update rule:

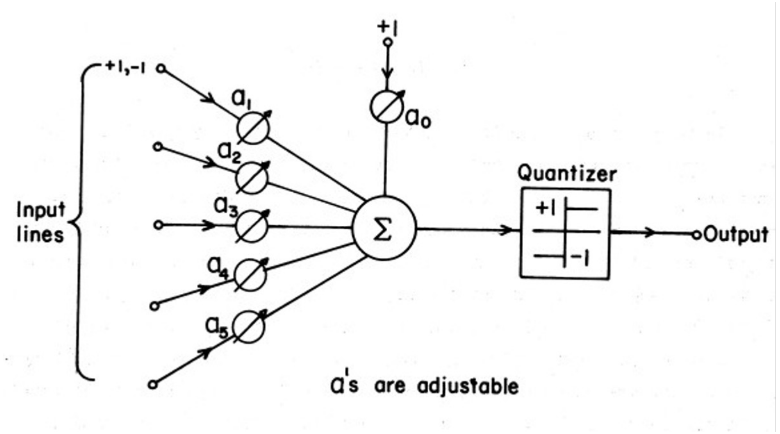
$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i},$$

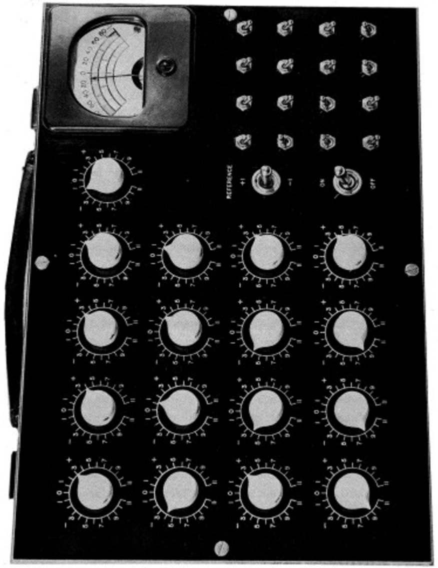


Frank Rosenblatt, ~1957: Perceptron

Slide credit: Stanford CS231n Instructors

A bit of history...





Widrow and Hoff, ~1960: Adaline/Madaline

Slide credit: Stanford CS231n Instructors

A bit of history...

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

recognizable math

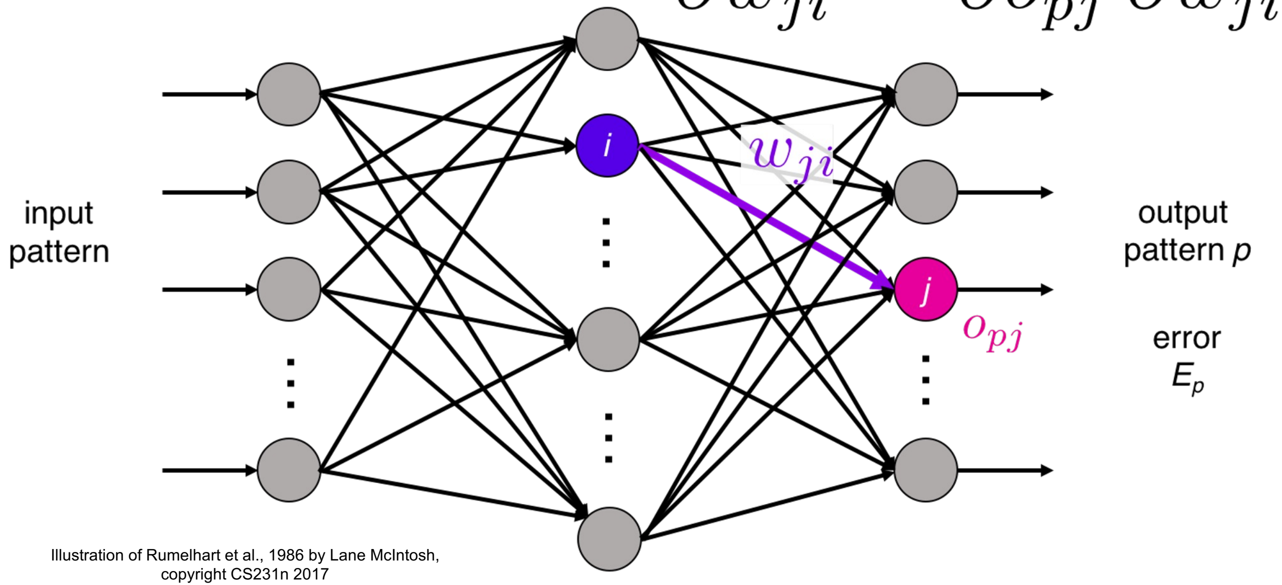


Illustration of Rumelhart et al., 1986 by Lane McIntosh, copyright CS231n 2017

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

A bit of history...

[Hinton and Salakhutdinov 2006]

Reinvigorated research in Deep Learning

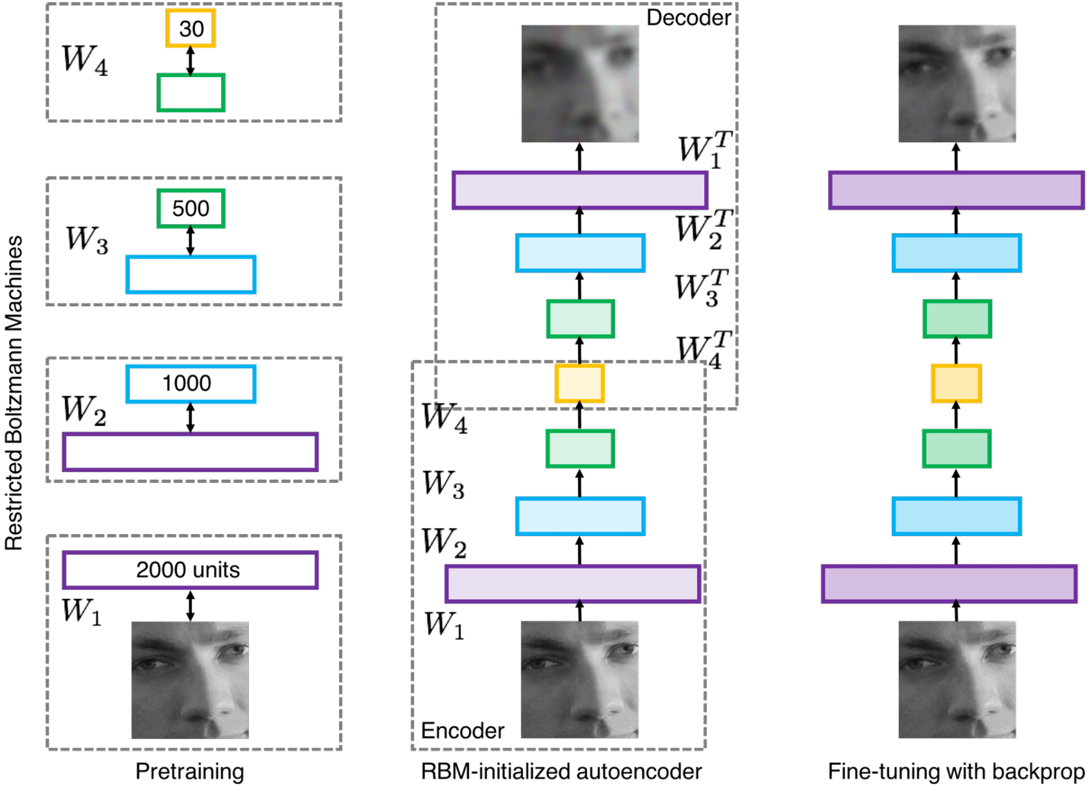


Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

Slide credit: Stanford CS231n Instructors

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

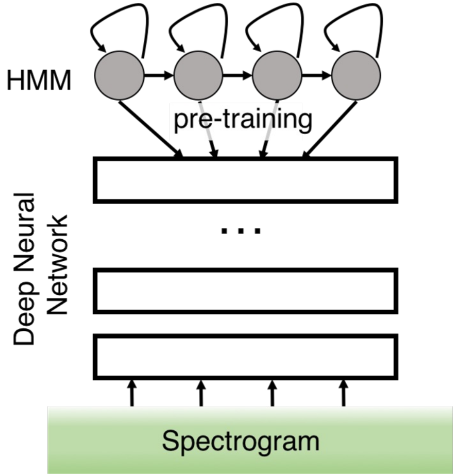
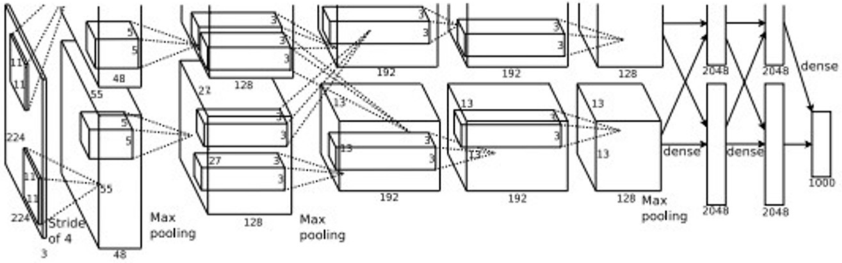
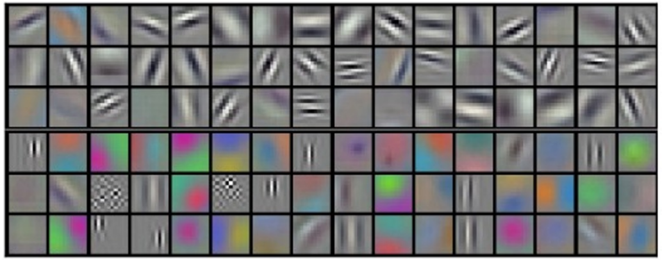


Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

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.

Slide credit: Stanford CS231n Instructors

A bit of history:

Hubel & Wiesel,

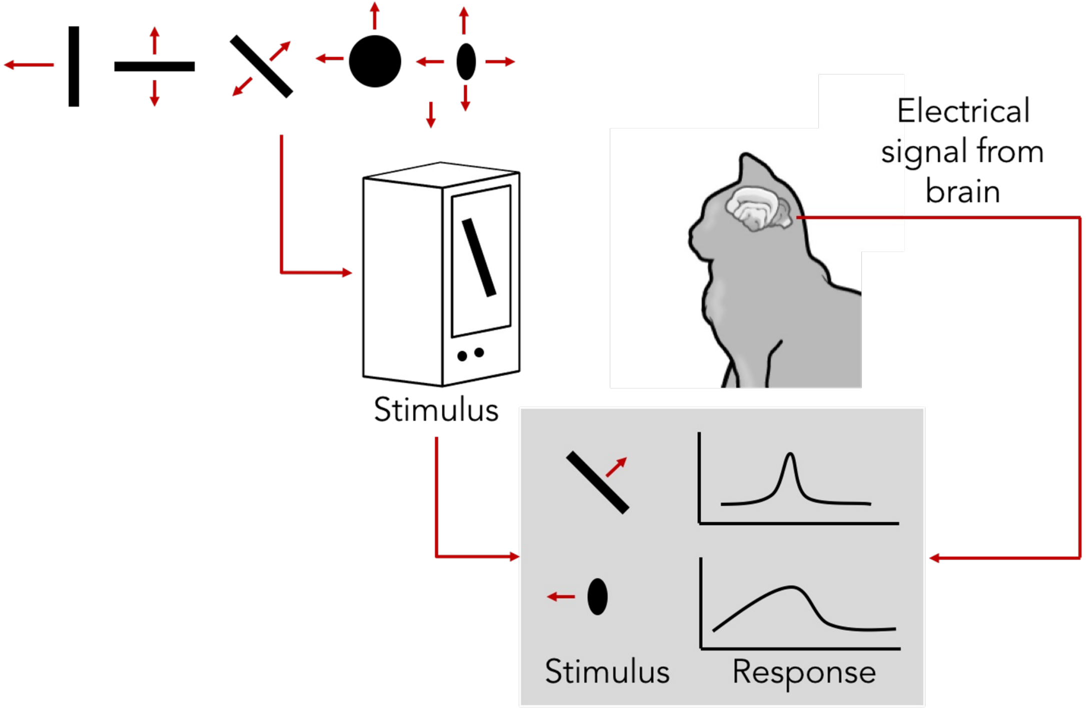
1959

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...



Slide credit: Stanford CS231n Instructors

Hierarchical organization

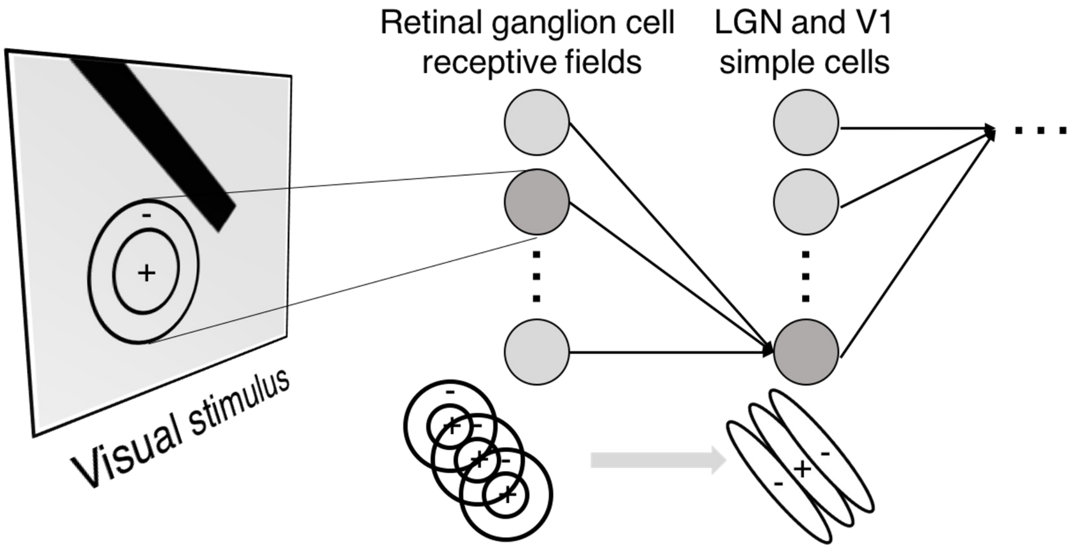
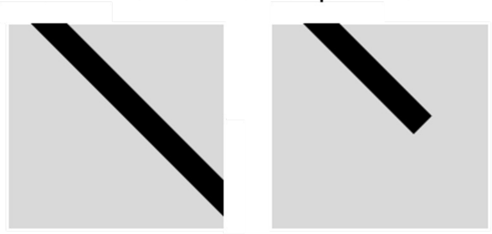


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

Simple cells:
Response to light orientation

Complex cells:
Response to light orientation and movement

Hypercomplex cells:
response to movement with an end point



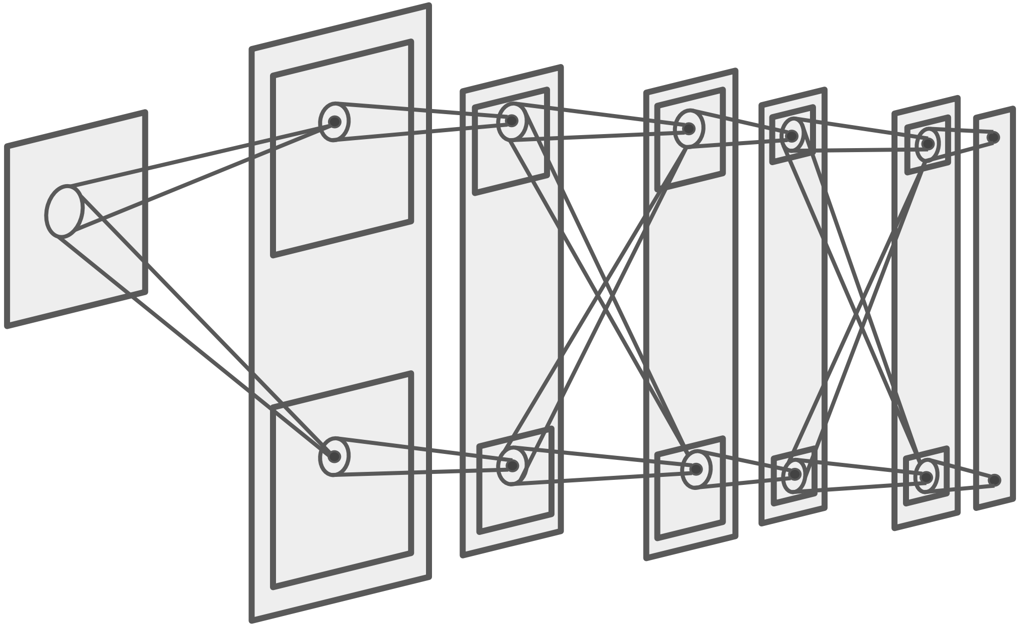
No response

Response (end point)

Slide credit: Stanford CS231n Instructors

A bit of history:

Neocognitron
[Fukushima 1980]



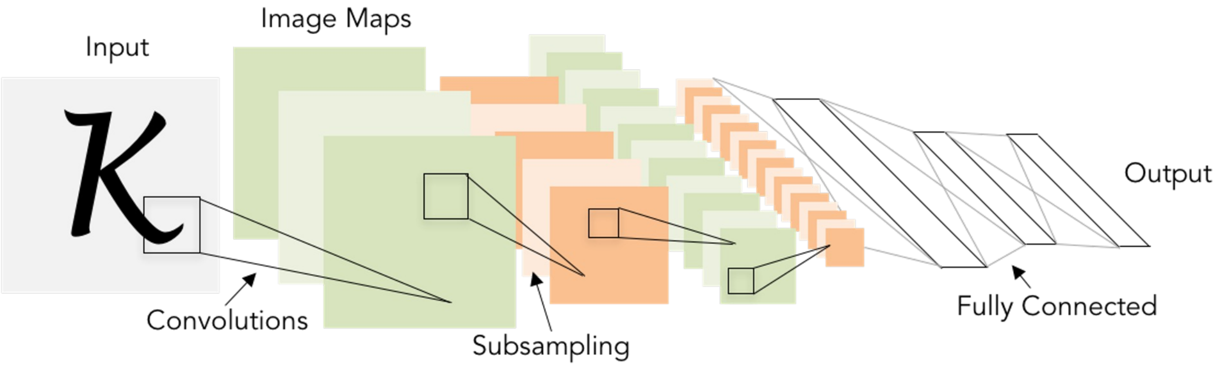
“sandwich” architecture (SCSCSC…)
simple cells: modifiable parameters
complex cells: perform pooling

Slide credit: Stanford CS231n Instructors

A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]



LeNet-5

Slide credit: Stanford CS231n Instructors

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

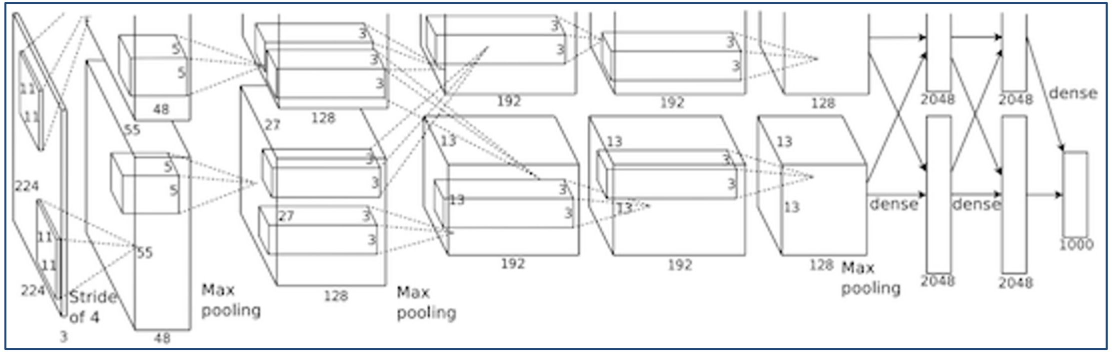


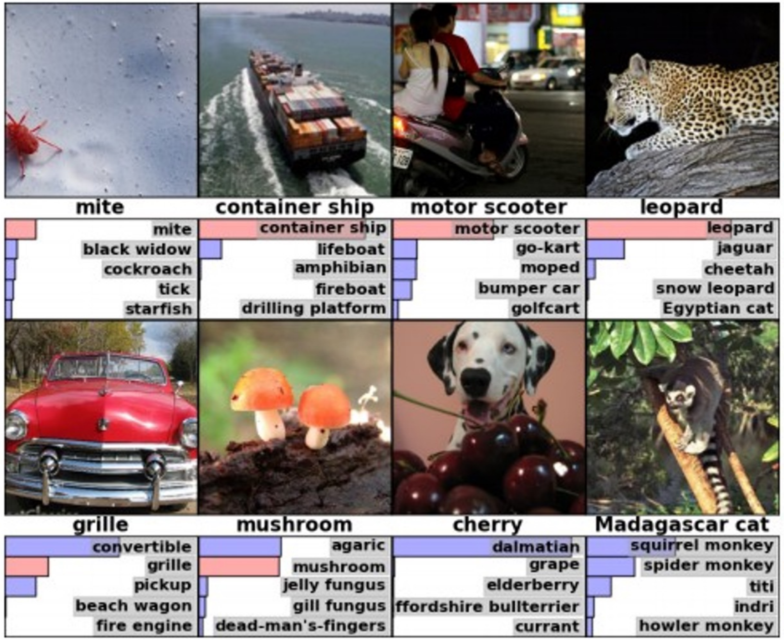
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

“AlexNet”

Slide credit: Stanford CS231n Instructors

Fast-forward to today: ConvNets are everywhere

Classification

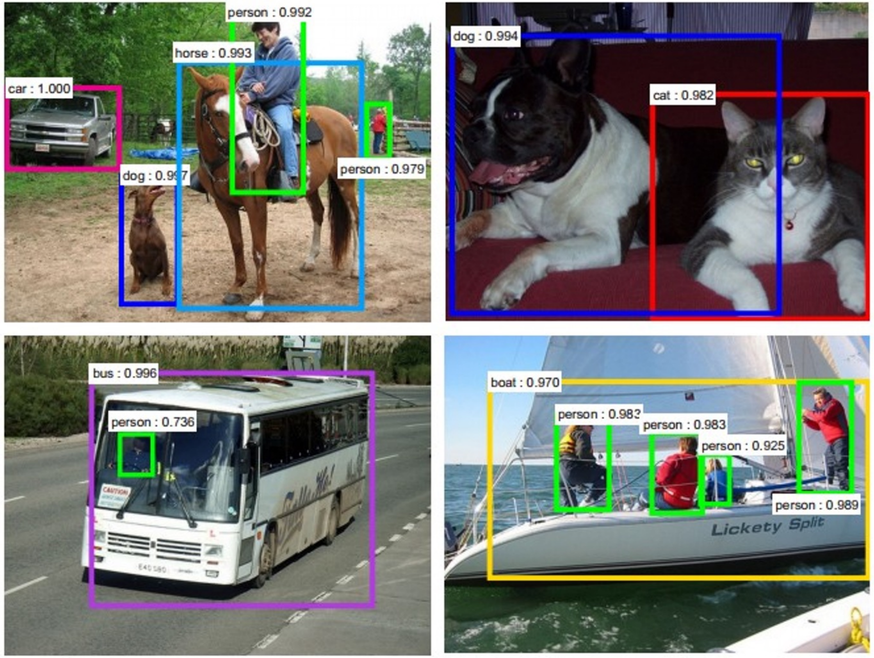


Retrieval

Slide credit: Stanford CS231n Instructors

Fast-forward to today: ConvNets are everywhere

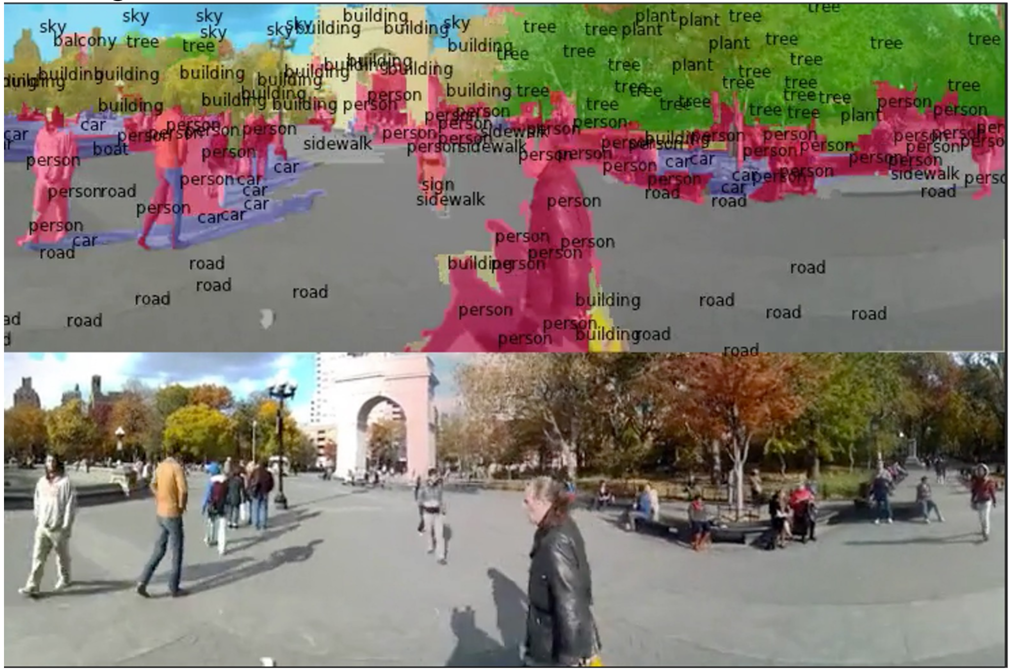
Detection



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

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

Segmentation



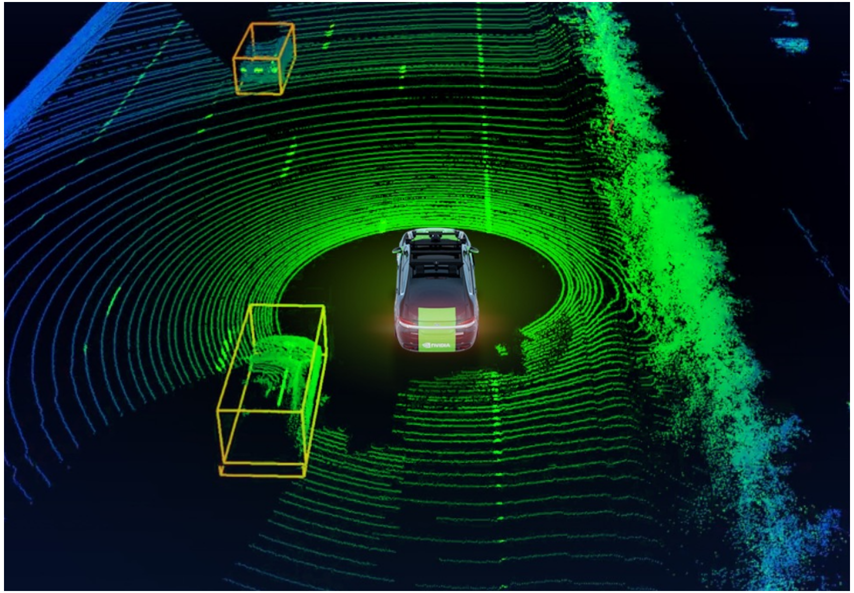
Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

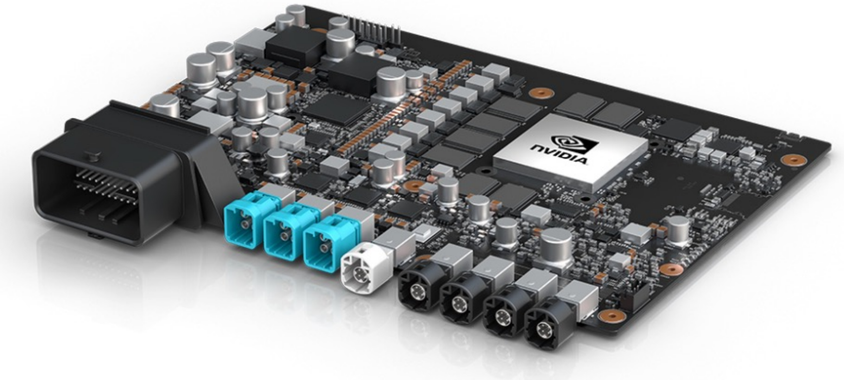
Slide credit: Stanford CS231n Instructors

Fast-forward to today: ConvNets are everywhere

Autonomous Driving: GPUs & specialized chips are fast and compact enough for on-board compute!



https://blogs.nvidia.com/blog/2021/01/27/lidar-sensor-nvidia-drive/



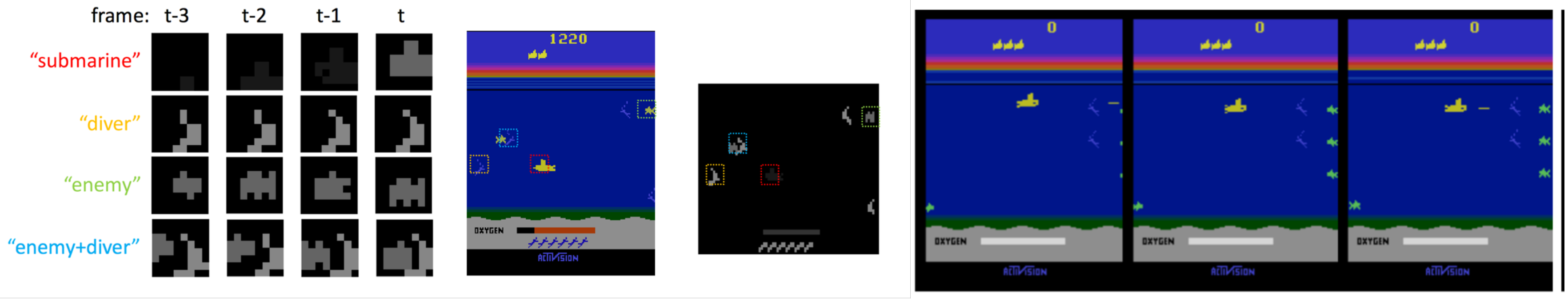
https://www.nvidia.com/en-us/self-driving-cars/

Slide credit: Stanford CS231n Instructors

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]

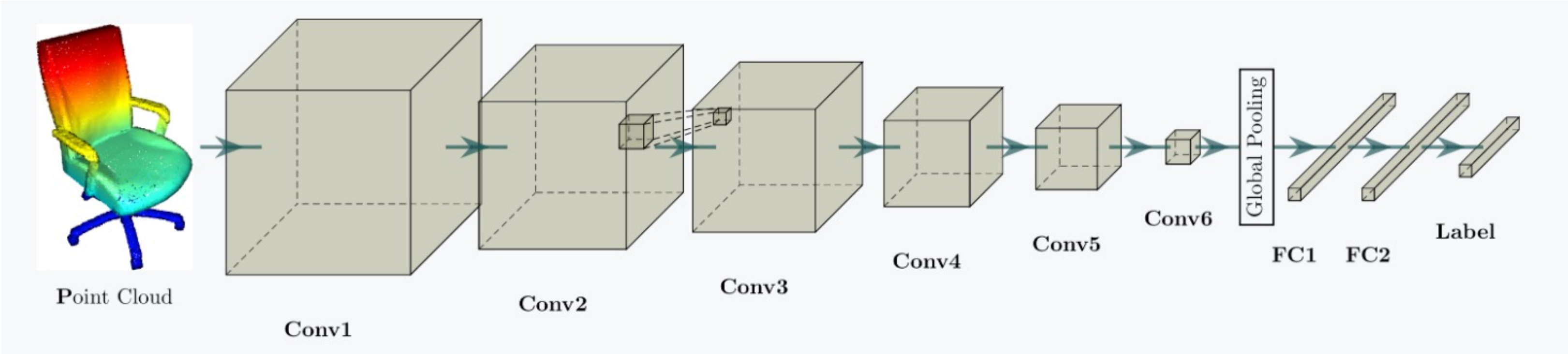


[Guo et al. 2014]

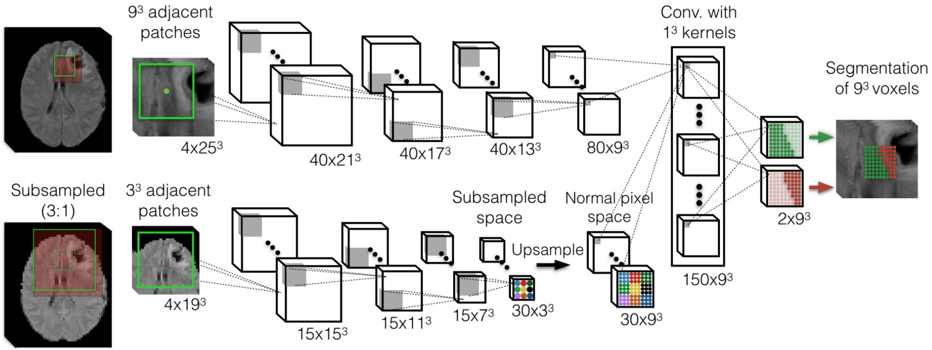
Slide credit: Stanford CS231n Instructors

Fast-forward to today: ConvNets are everywhere

Generalized convolution: **spatial convolution**



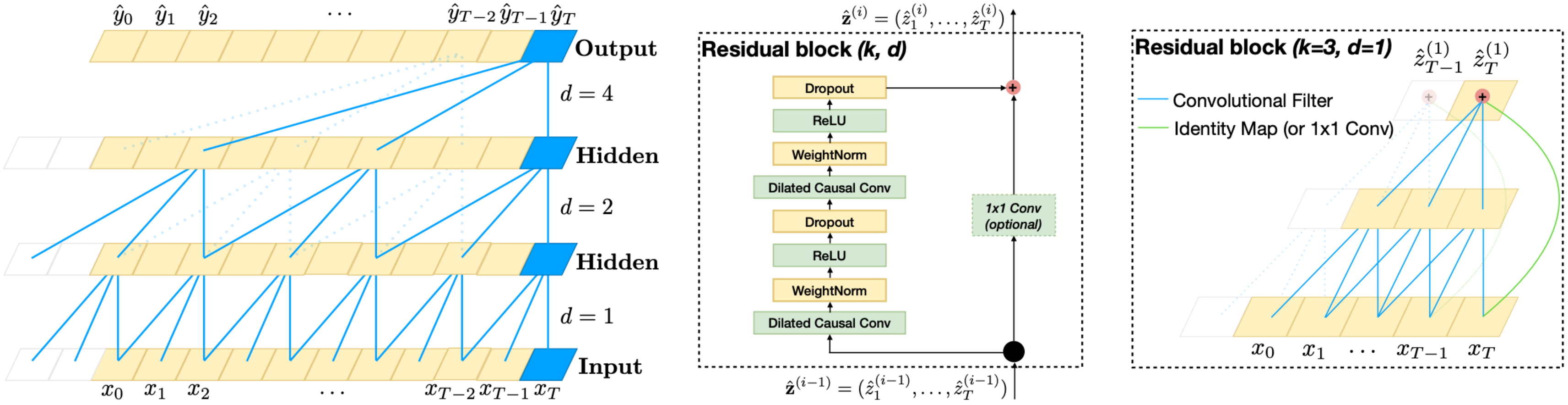
Choi et al., 2019



Kamnitsas et al., 2015

Fast-forward to today: ConvNets are everywhere

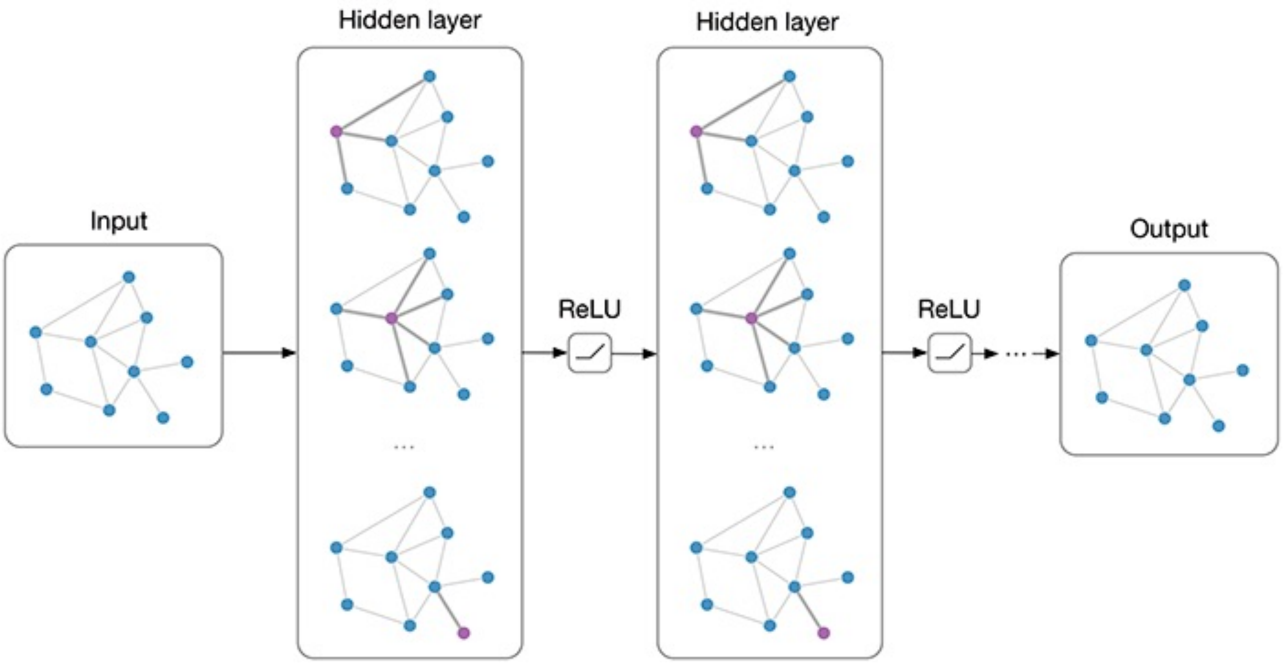
Generalized convolution: **temporal convolution**



Bai et al., 2018

Fast-forward to today: ConvNets are everywhere

Generalized convolution**: graph convolution**



Kipf et al., 2017

Image-to-text

No errors

Minor errors

Somewhat related

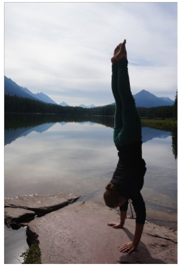
A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

A woman is holding a cat in her hand

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor



A woman standing on a beach holding a surfboard

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]
[Radford, 2021]

Captions generated by Justin Johnson using Neuraltalk2

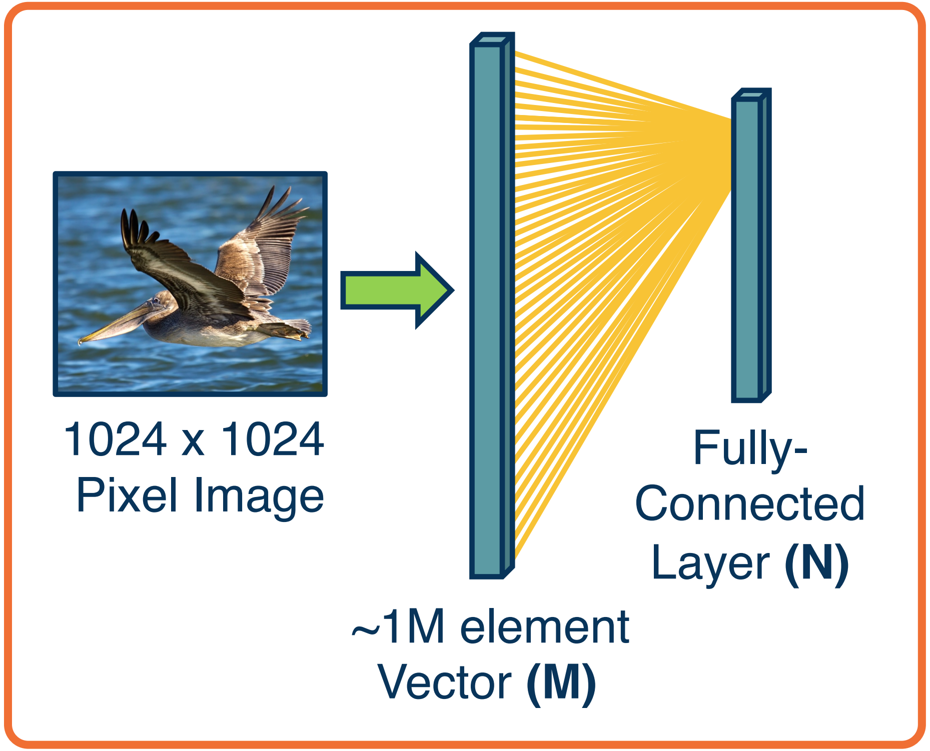
Text-to-Image

"An avocado armchair"

[Reed, 2016]
[Zhang, 2017]
[Johnson, 2018]
[Ramesh, 2021]
[Frans, 2021]
[Saharia, 2022]
[Ramesh, 2022]

Convolutional Neural Networks

The connectivity in linear layers **doesn't always make sense**



How many parameters?

- M*N (weights) + N (bias)

Hundreds of millions of parameters **for just one layer**

More parameters => More data needed & slower to train / inference

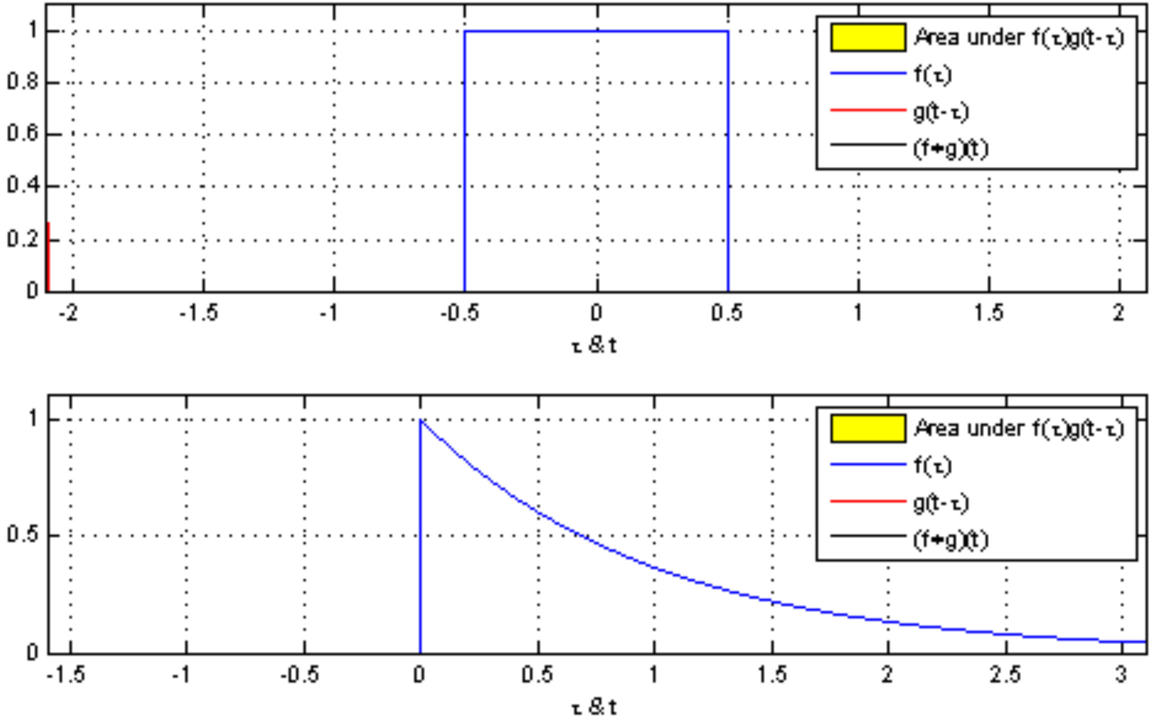
Is this necessary?

Image features are spatially localized!

- Smaller features repeated across the image
  - Edges
  - Color
  - Motifs (corners, etc.)
- No reason to believe one feature tends to appear in a fixed location. Need to search in entire image.

Can we induce a *bias* in the design of a neural network layer to reflect this?

Convolution: A 1D Visual Example



From https://en.wikipedia.org/wiki/Convolution

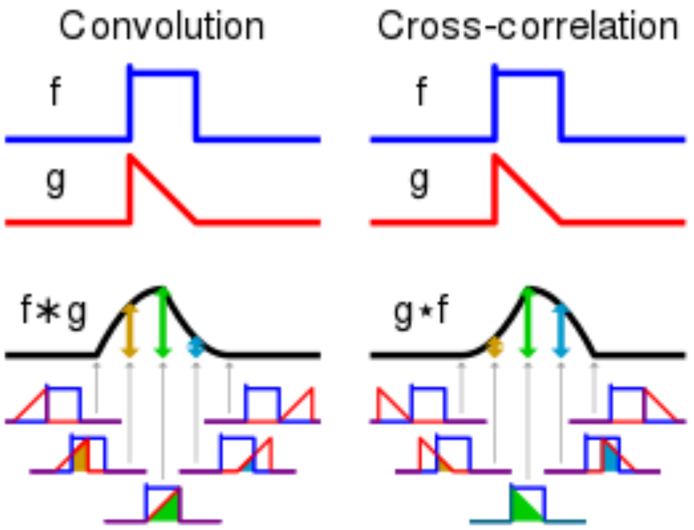
Convolution

1-D Convolution is defined as the **integral** of the **product** of two functions after one is reflected about the y-axis and shifted.

Cross-correlation is convolution without the y-axis reflection.

Intuitively: given function f and filter g. How similar is $g(-x)$ with the part of $f(x)$ that it's operating on.

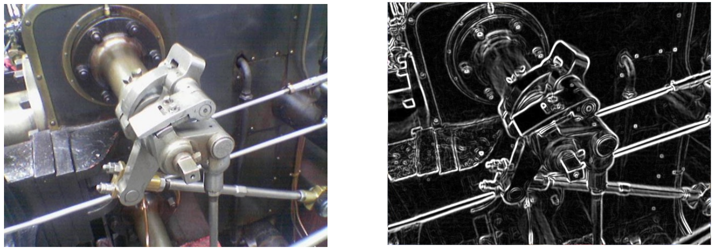
For ConvNets, we don't flip filters, so we are really using Cross-Correlation Nets!



From https://en.wikipedia.org/wiki/Convolution

Convolution in Computer Vision (non-Deep)

Convolution with Gaussian Filter (Gaussian Blur)



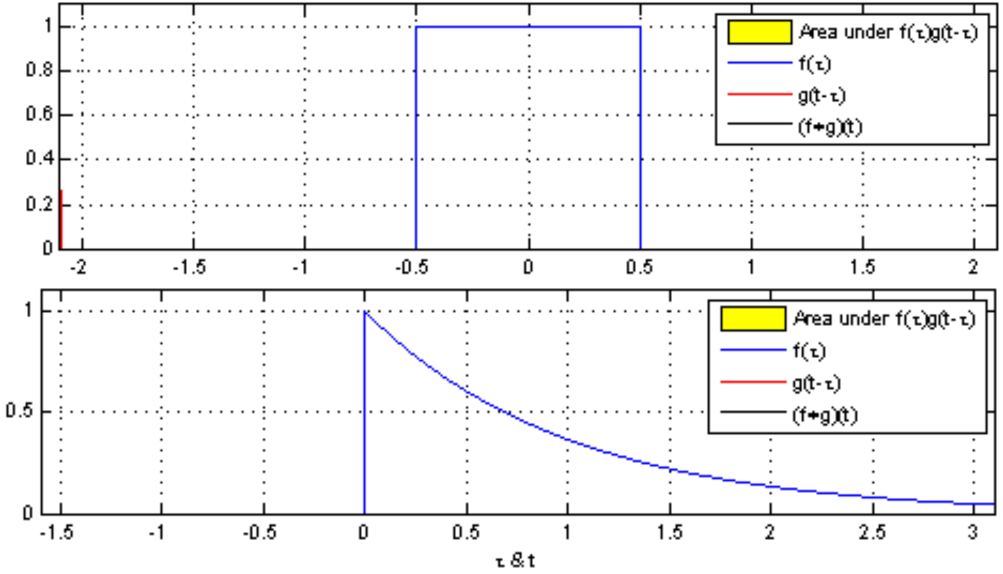
Convolution with Sobel Filter (Edge Detection)

Convolution: A 1D Visual Example

g(): filter / pattern template

f(): signal / observed data

f*g(): how well data
matches with the template



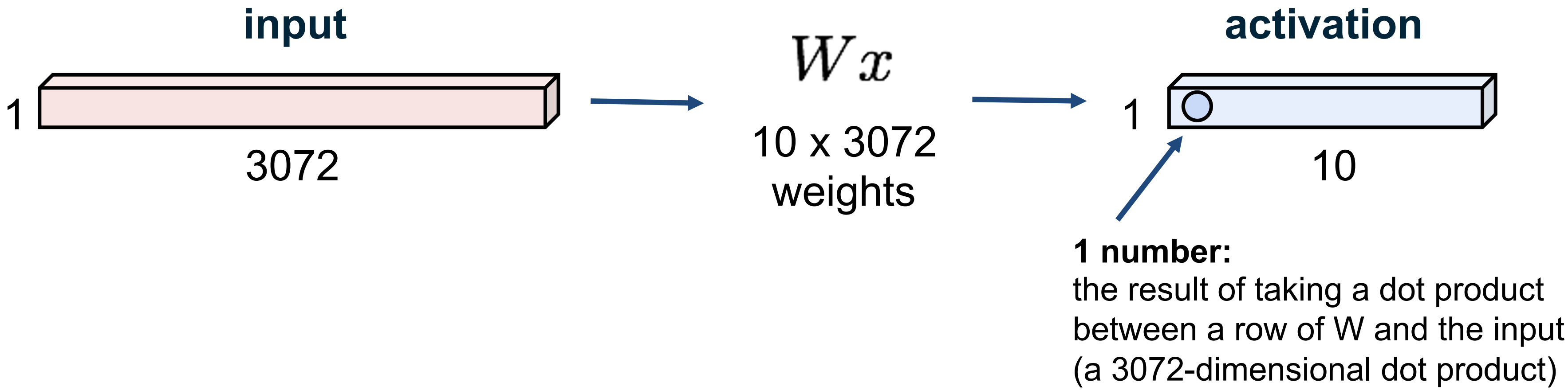
From https://en.wikipedia.org/wiki/Convolution

For Convolution Layers in NN, think of:

- g() as the weights to learn
- f() as the input to the layer
- f*g() as the output of the layer (result of convolution)

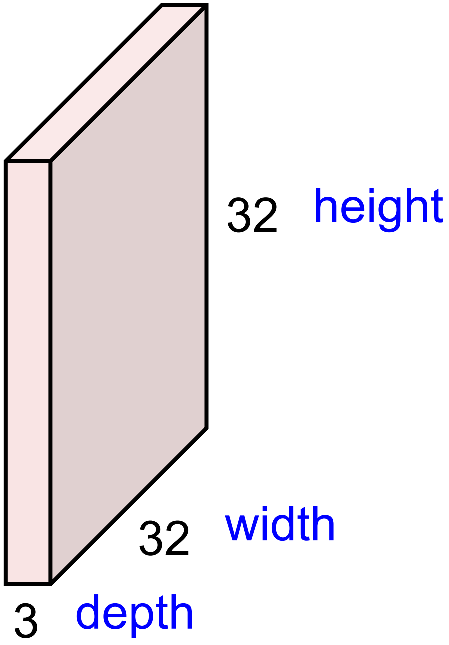
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



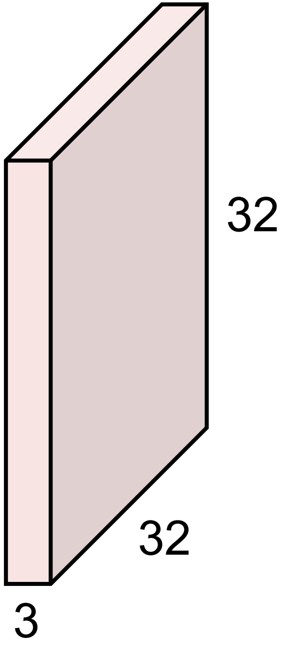
Convolution Layer

32x32x3 image -> preserve spatial structure

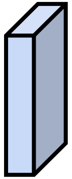


Convolution Layer

32x32x3 image



5x5x3 filter

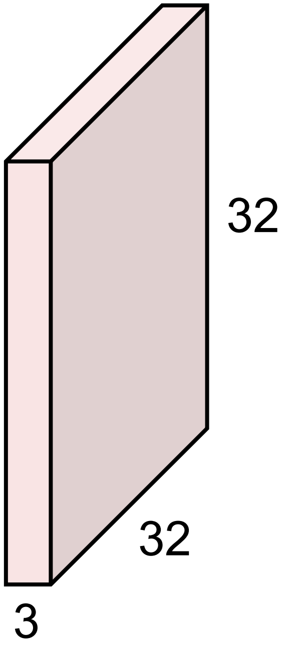


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

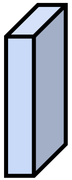
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter

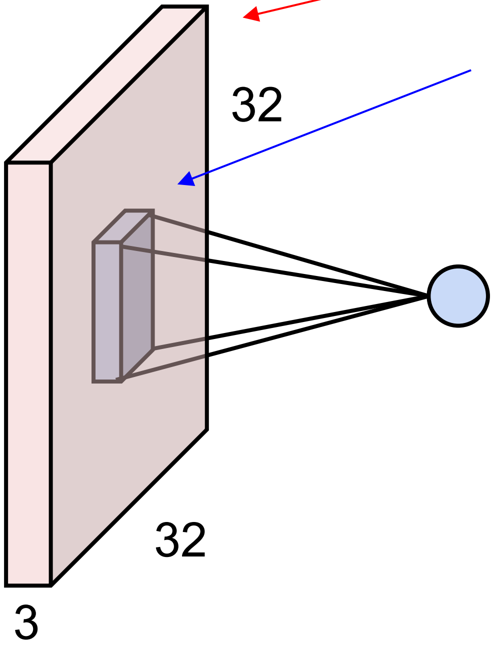


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

32x32x3 image

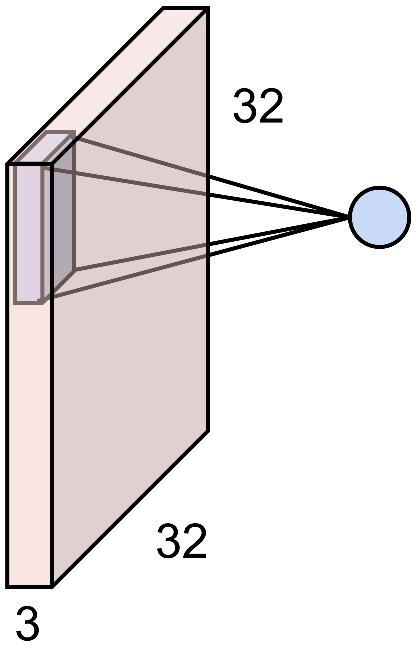
5x5x3 filter w



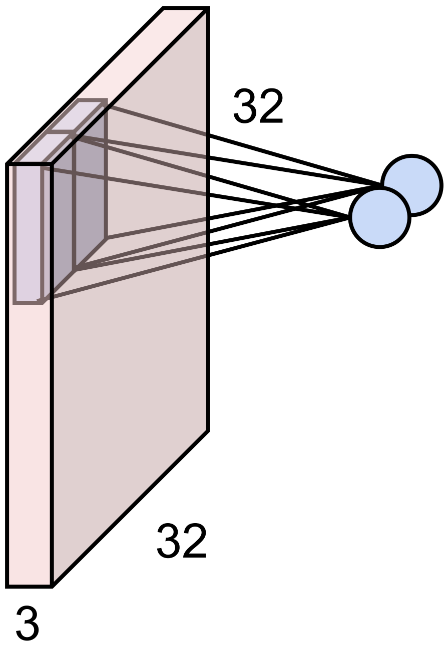
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

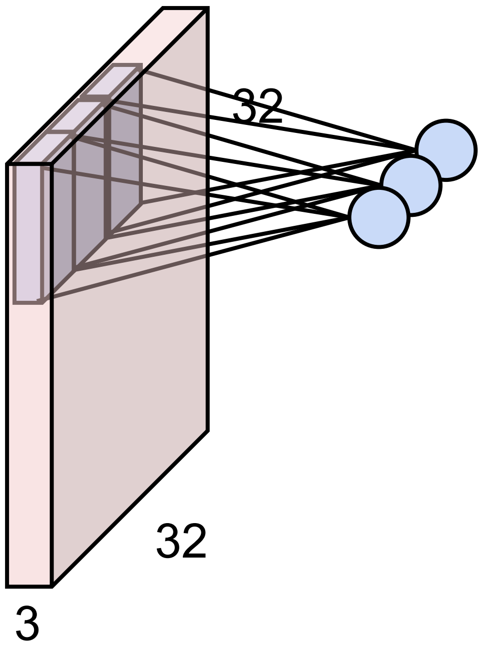
Convolution Layer



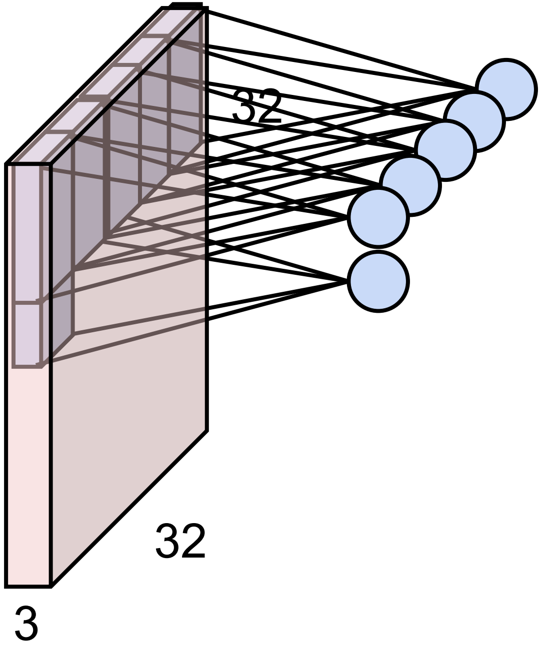
Convolution Layer



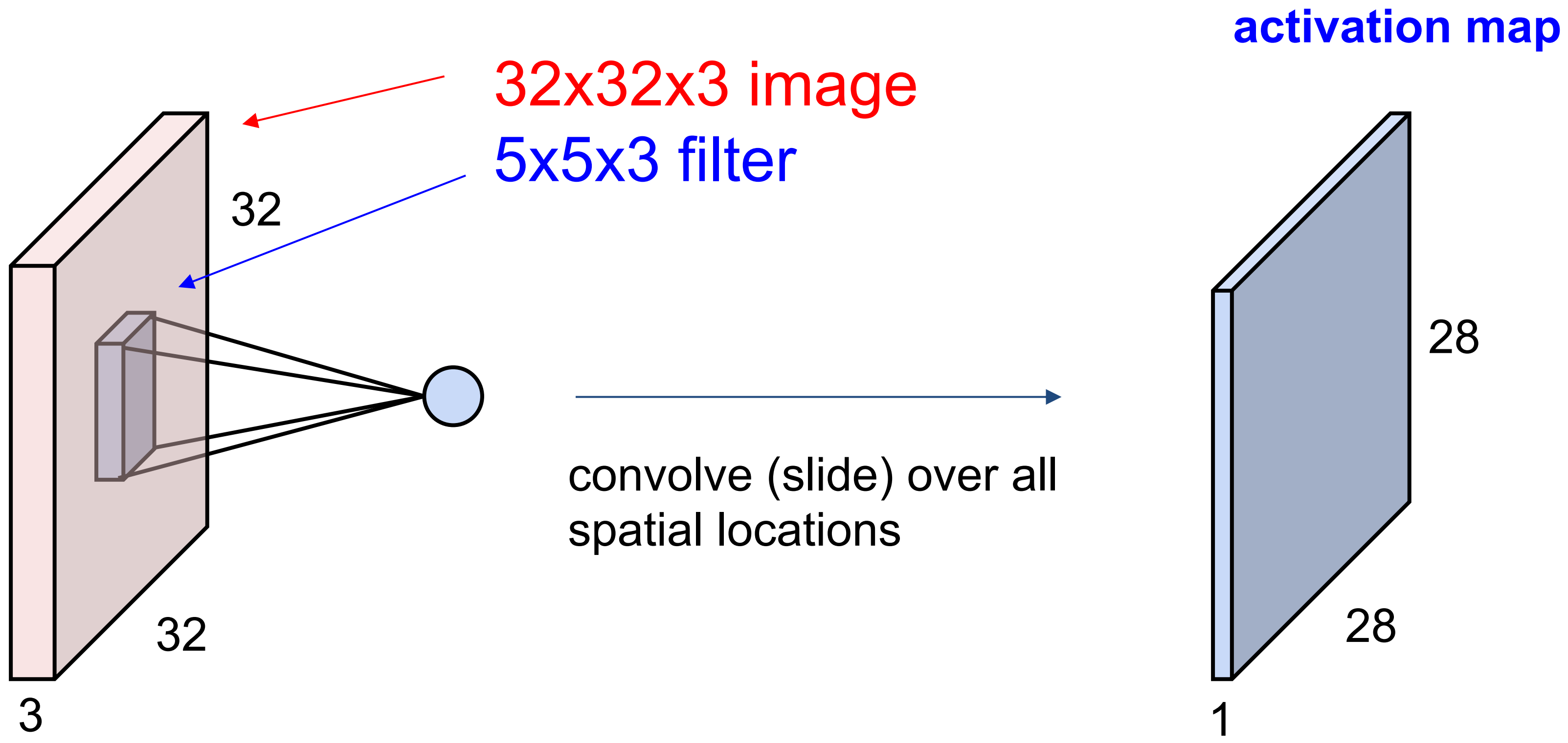
Convolution Layer



Convolution Layer



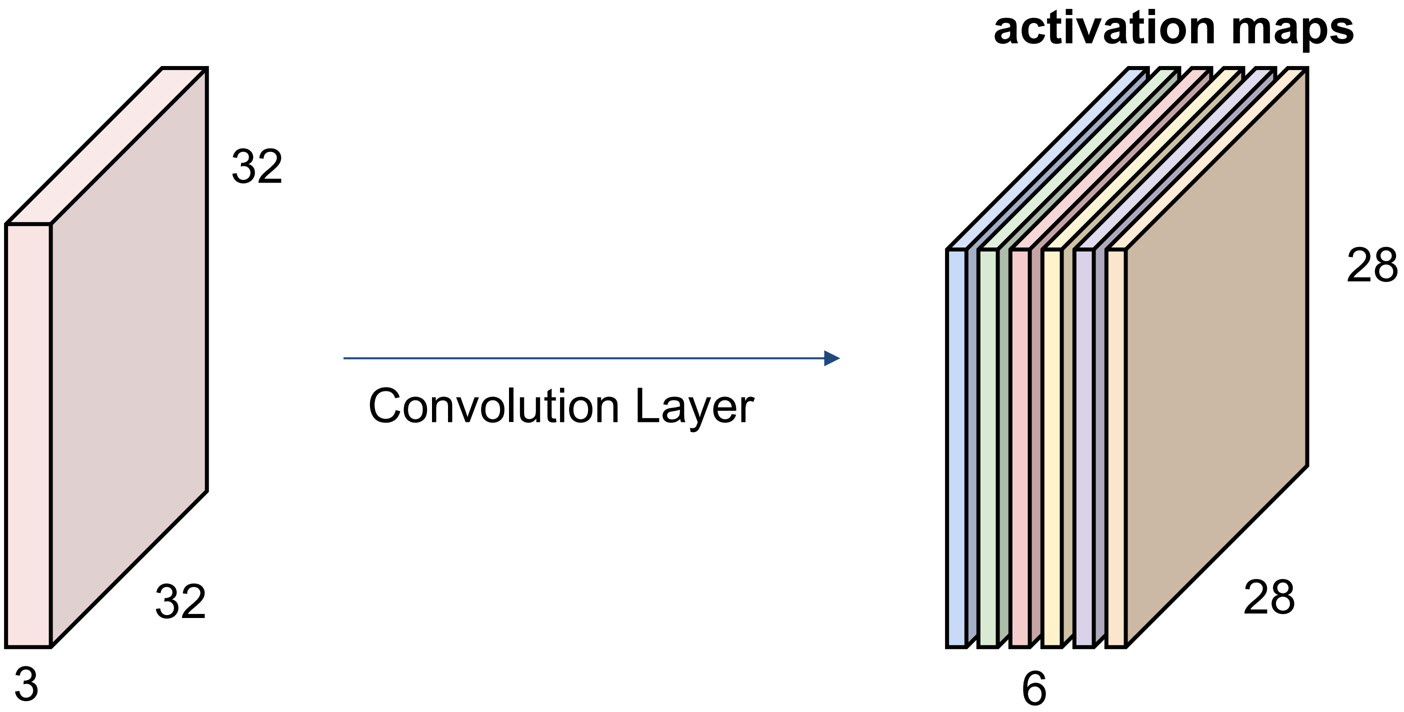
Convolution Layer



Convolution Layer

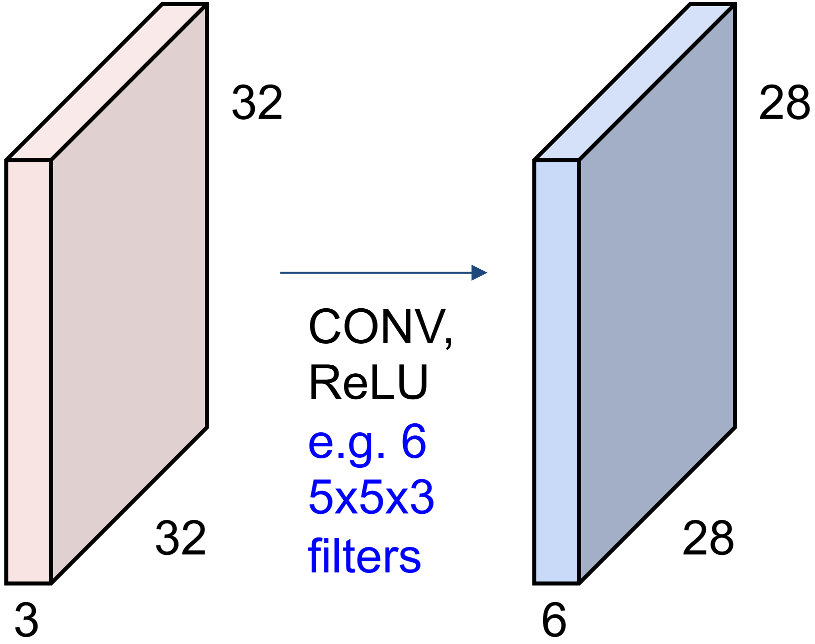
consider a second, green filter

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

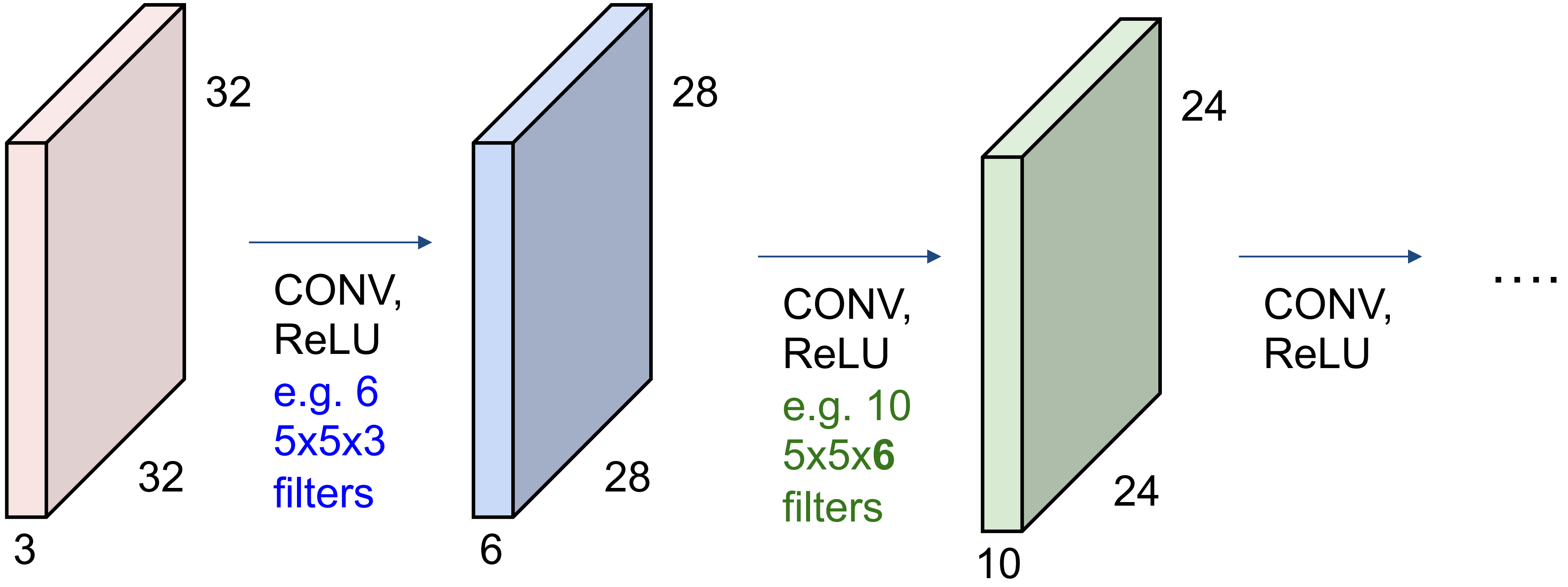


We stack these up to get a "new image" of size 28x28x6!

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



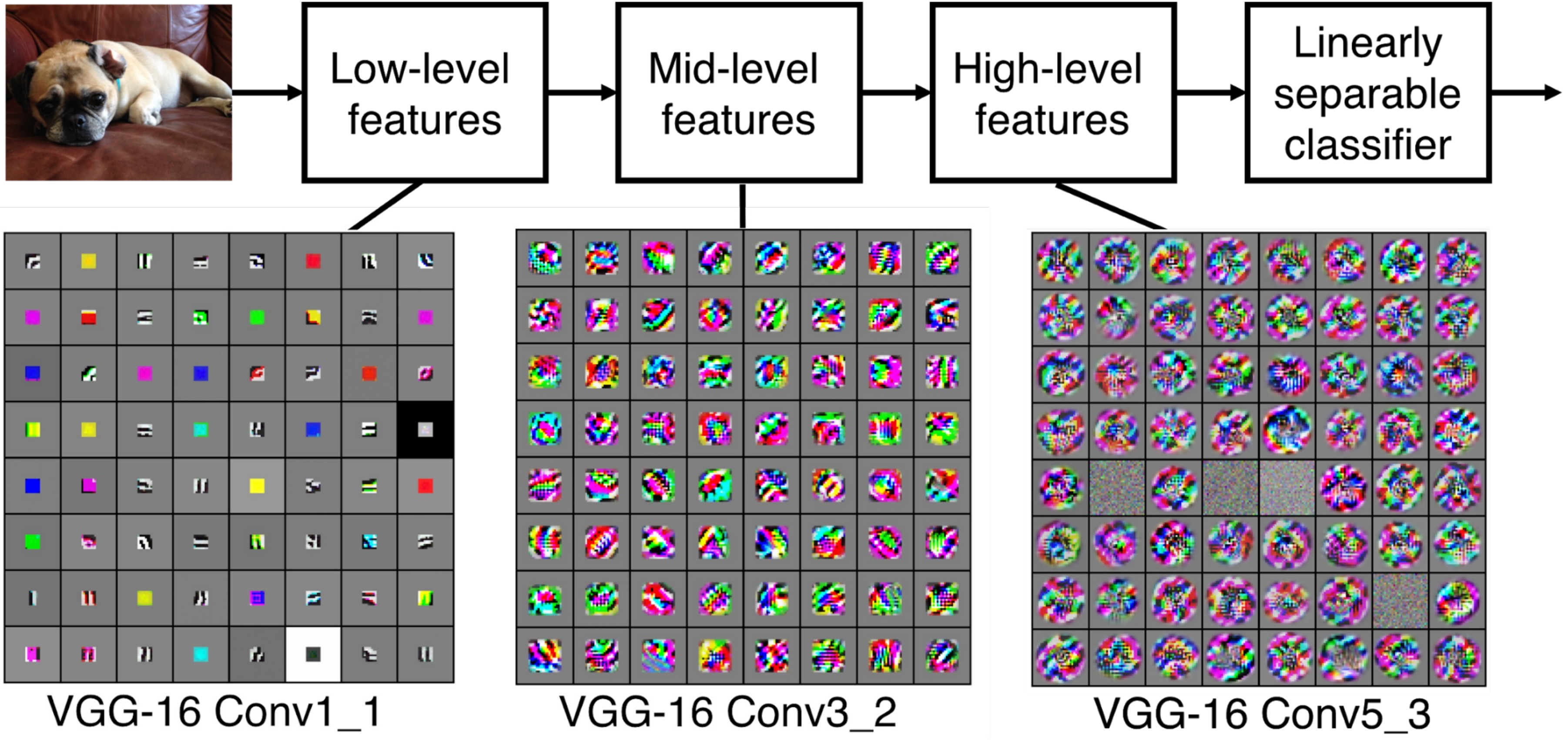
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



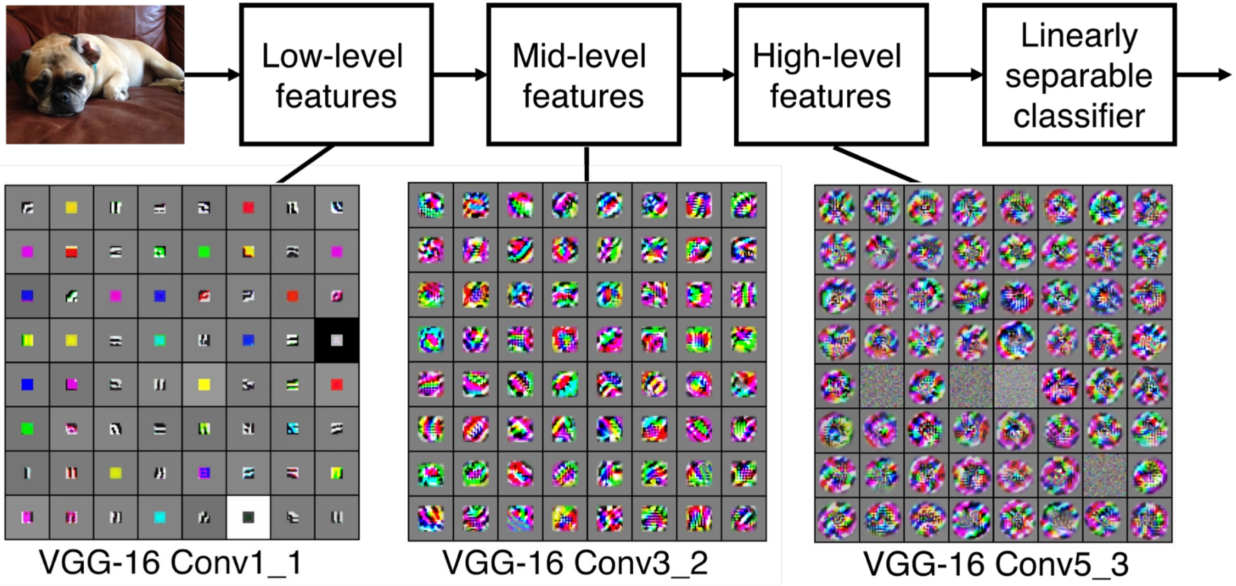
Preview

[Zeiler and Fergus 2013]

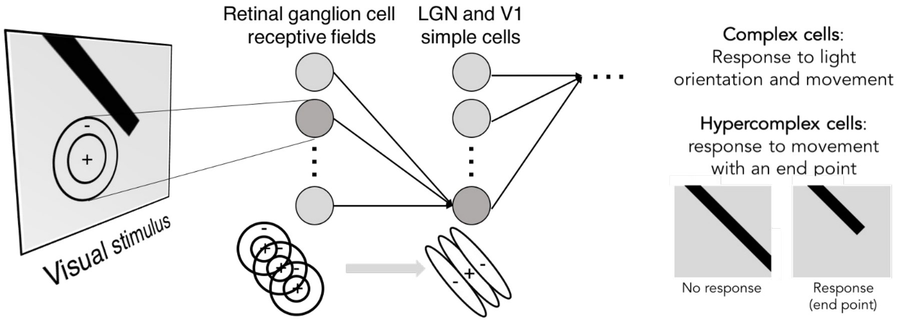
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



Preview



Low-level features
Mid-level features
High-level features
Linearly separable classifier
VGG-16 Conv1_1
VGG-16 Conv3_2
VGG-16 Conv5_3



Retinal ganglion cell receptive fields
LGN and V1 simple cells
Visual stimulus
Complex cells: Response to light orientation and movement
Hypercomplex cells: response to movement with an end point
No response
Response (end point)

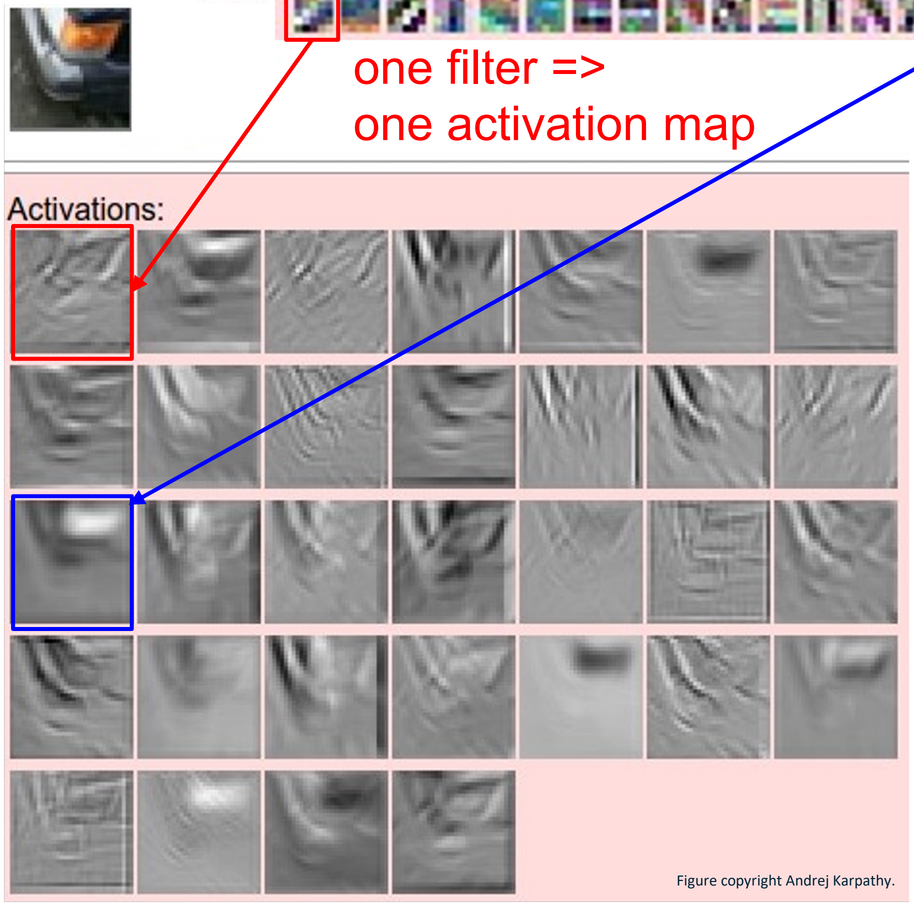


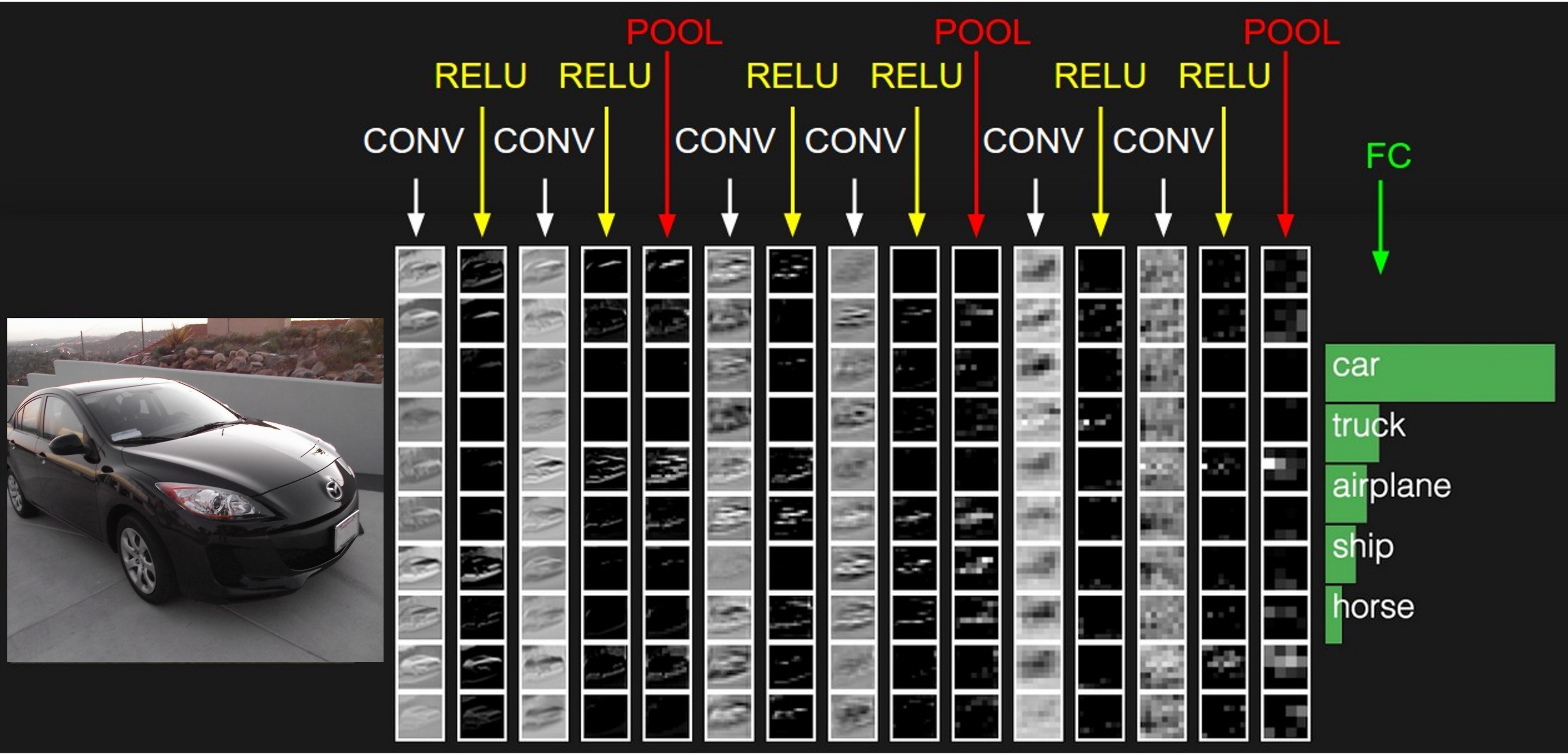
Figure copyright Andrej Karpathy.

example 5x5 filters
(32 total)

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

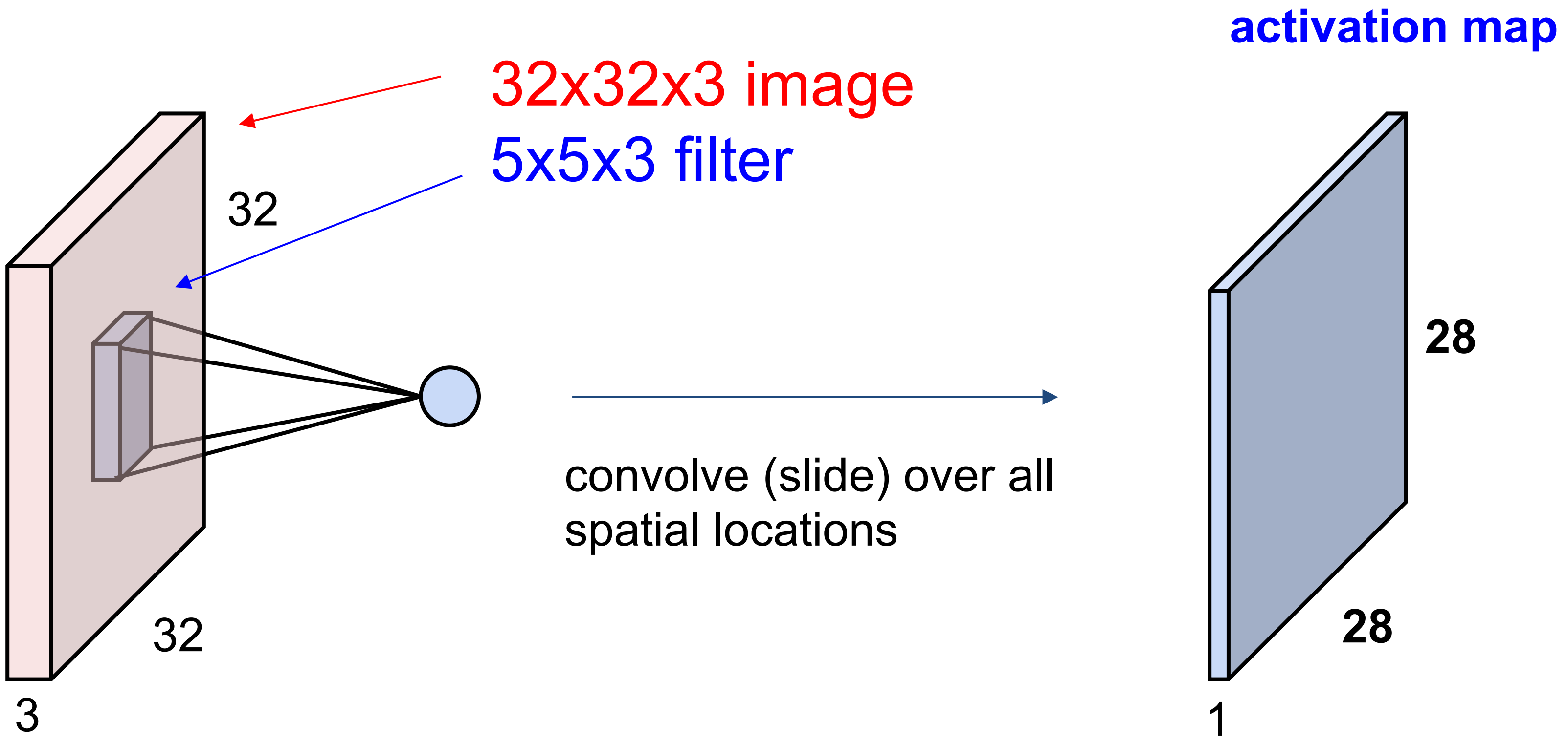
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

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

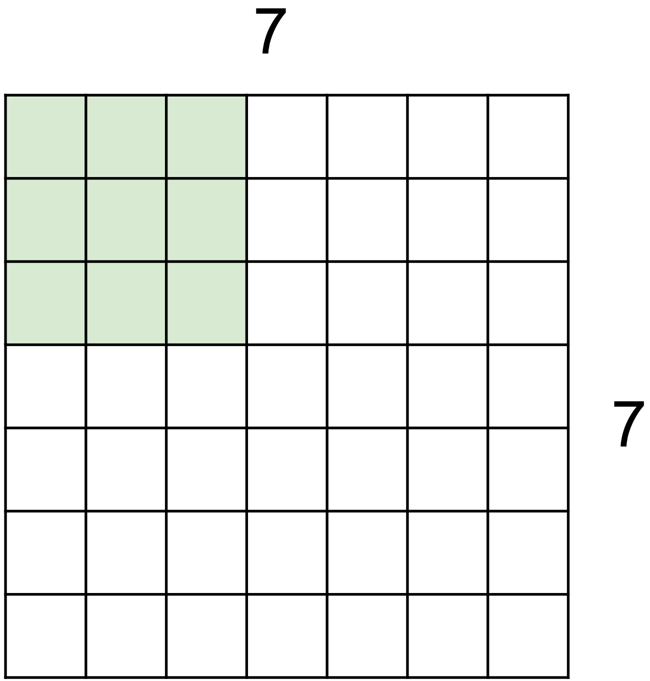


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

A closer look at spatial dimensions:

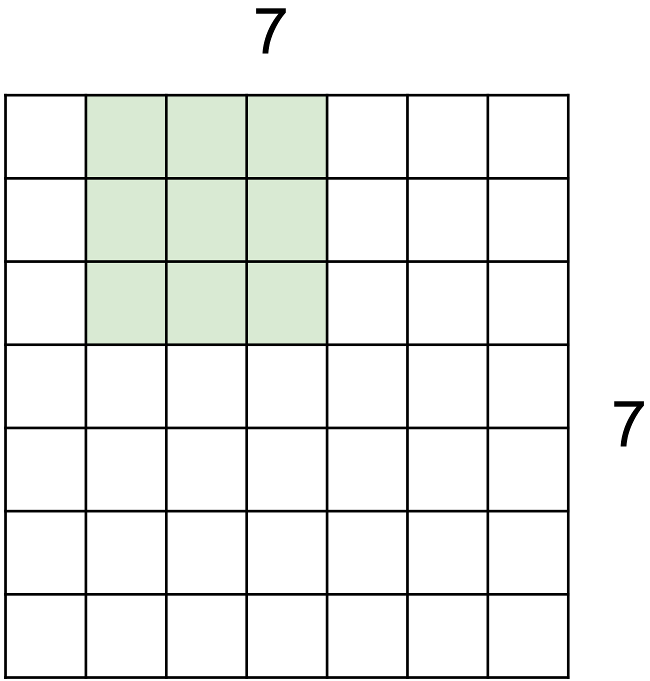


A closer look at spatial dimensions:



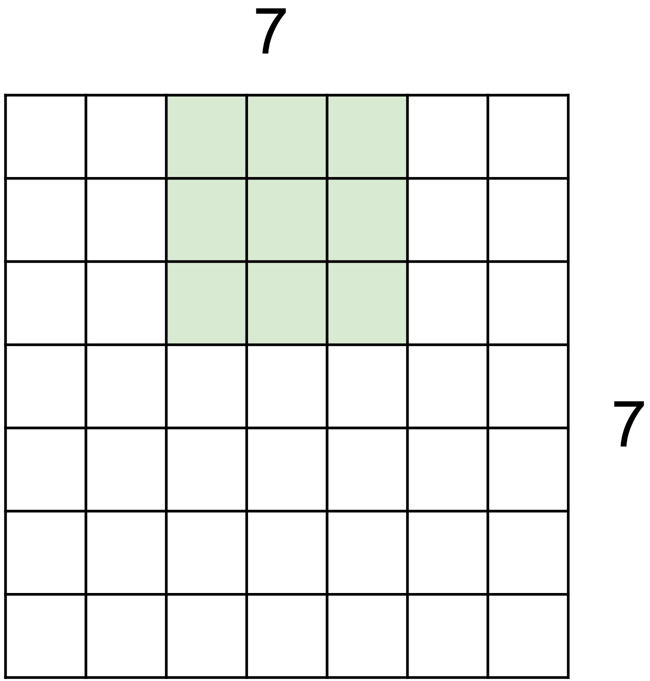
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

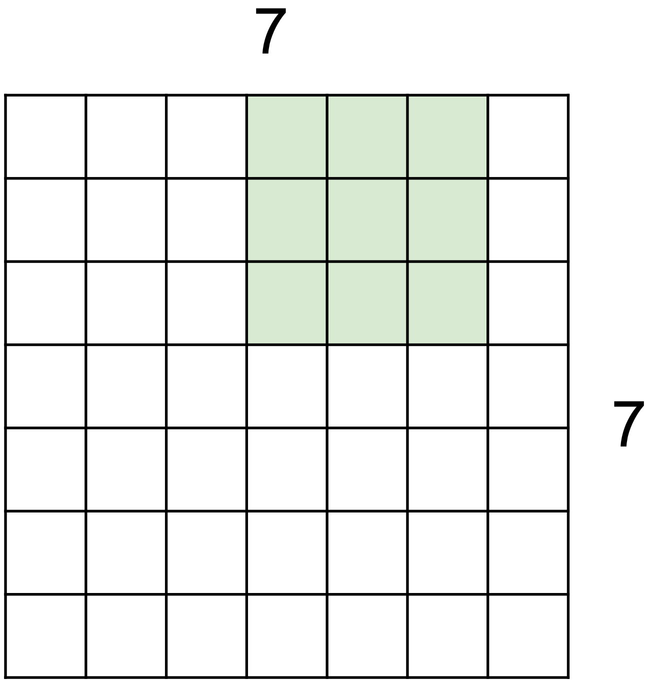


7x7 input (spatially)
assume 3x3 filter

The # of grid that the filter shifts
is called **stride.**

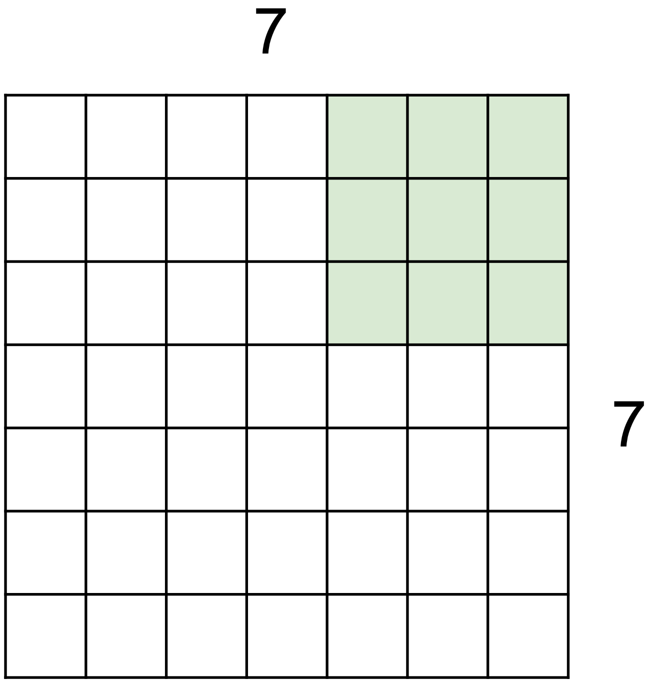
E.g., here we have stride = 1

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter **with stride = 1**

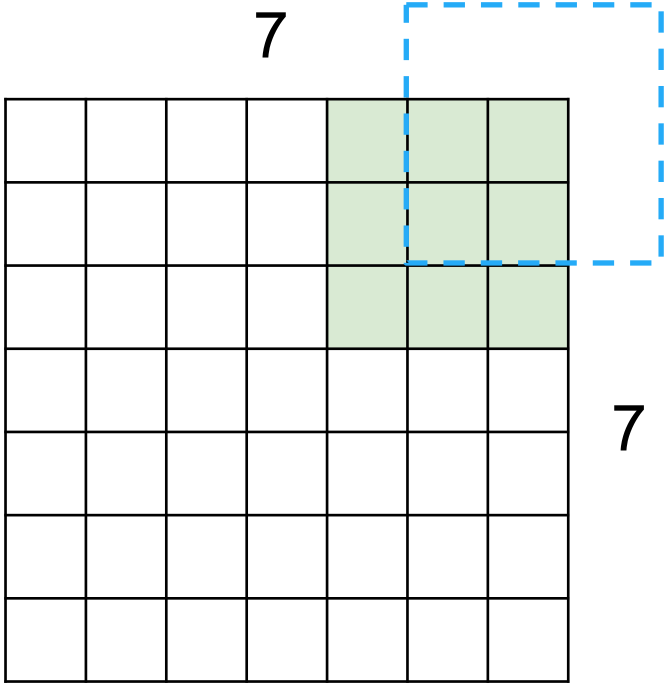
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter with stride = 1

=> 5x5 output

A closer look at spatial dimensions:

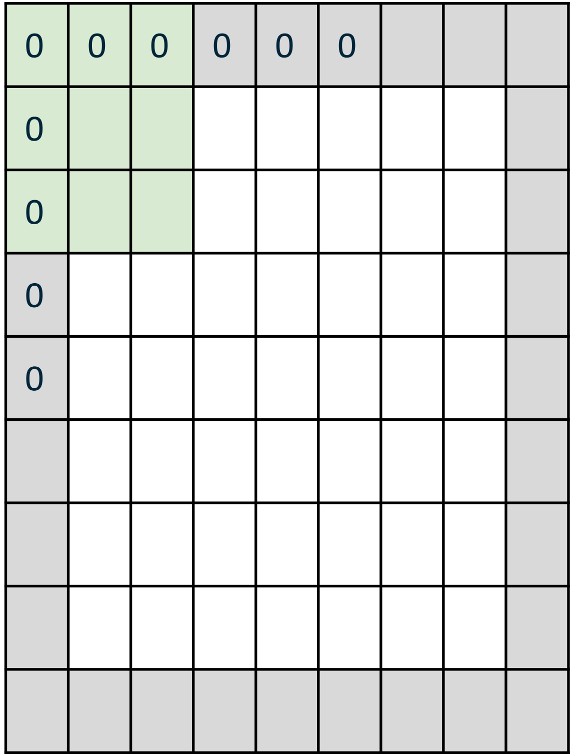


7x7 input (spatially)
assume 3x3 filter with stride = 1

=> 5x5 output

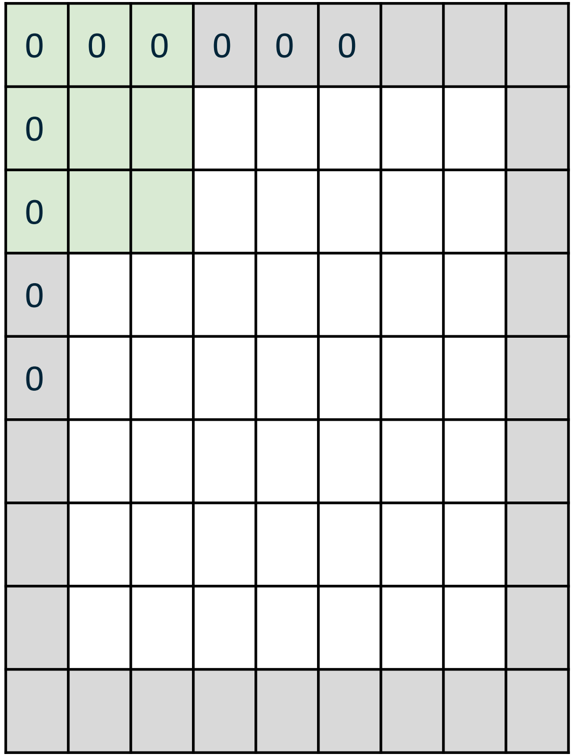
But what about the features at the border?

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

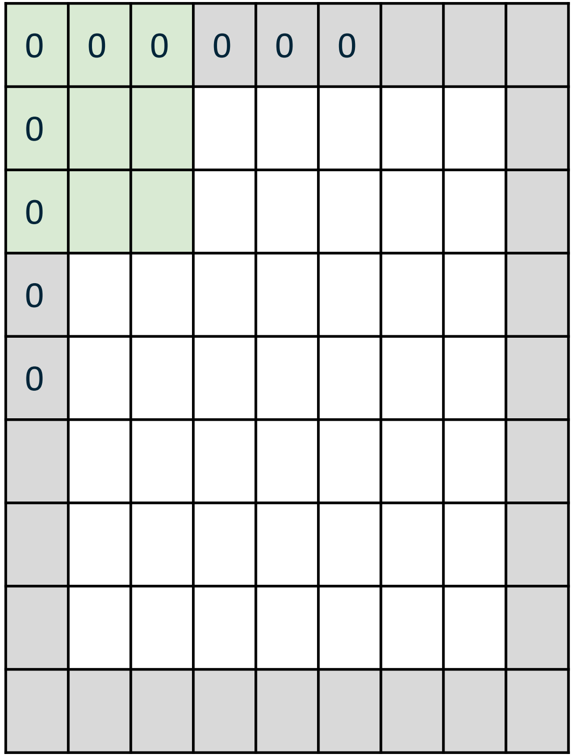
In practice: Common to zero pad the border



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 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

In practice: Common to zero pad the border

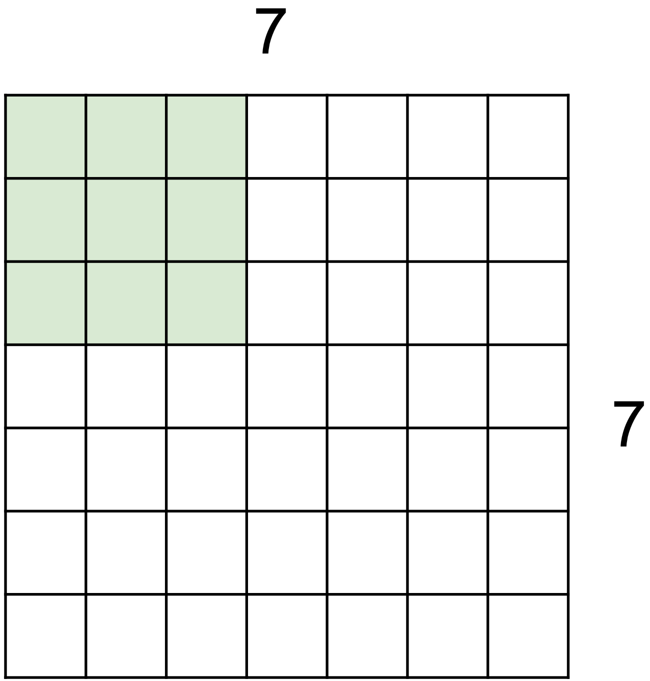


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

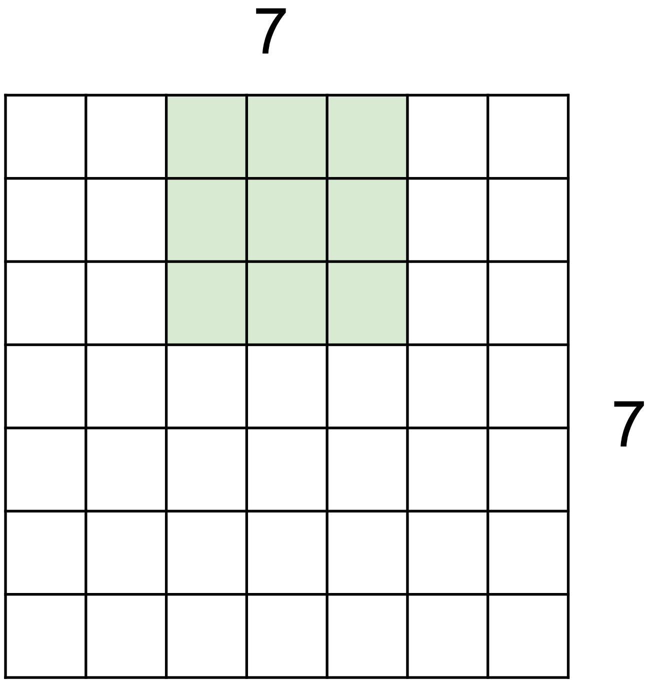
N = input dimension
P = padding size
F = filter size
Output size = (N – F + 2P) / stride + 1
= (7 – 3 + 2 * 1) / 1 + 1 = 7

A closer look at spatial dimensions:



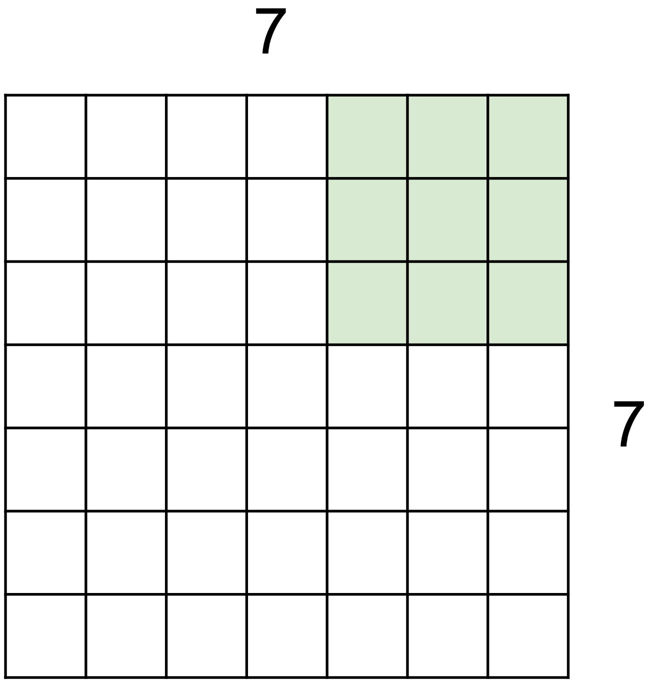
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



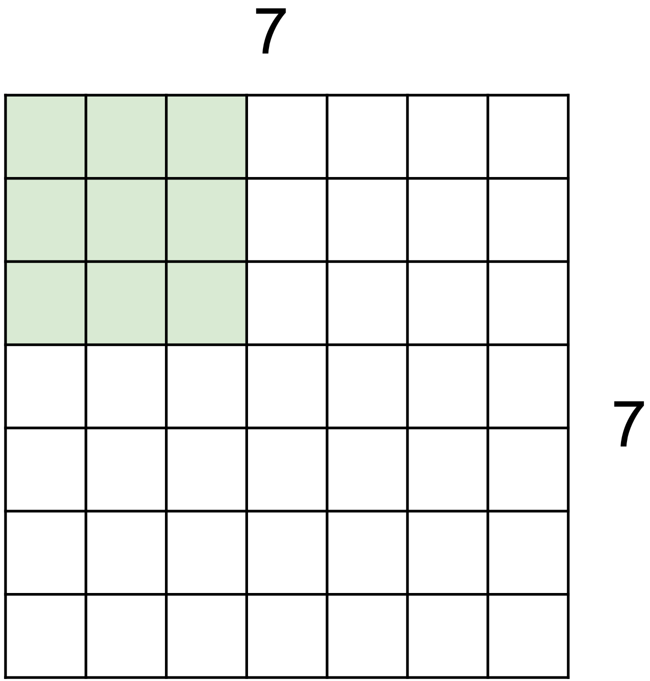
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



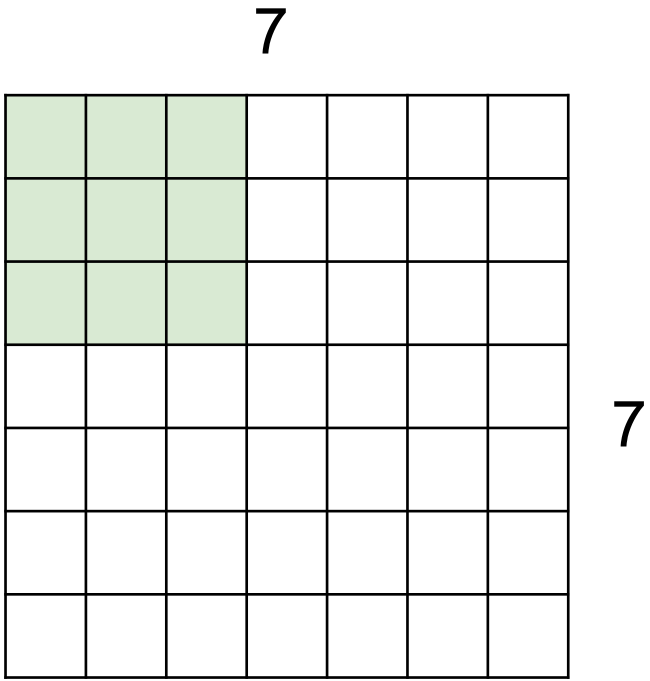
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



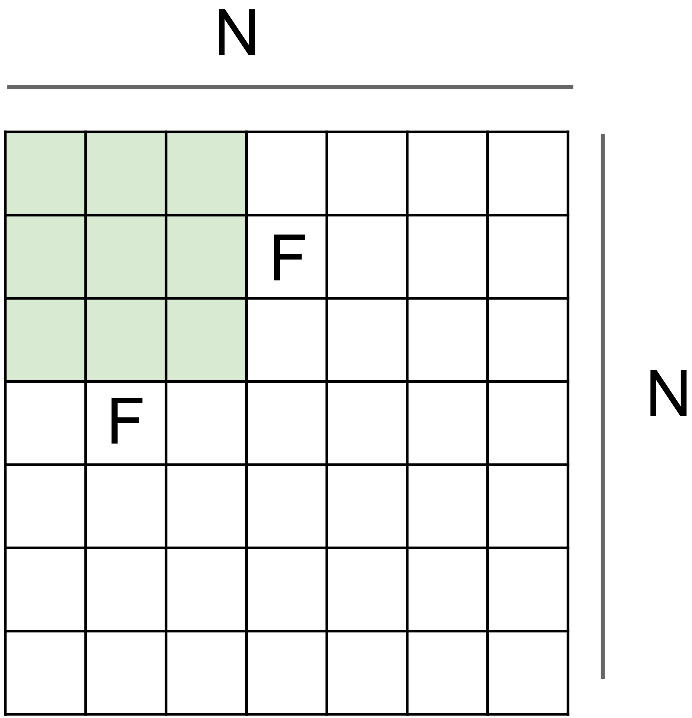
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



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

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



Output size:

(N - F) / stride + 1

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

stride 1 => (7 - 3)/1 + 1 = 5

stride 2 => (7 - 3)/2 + 1 = 3

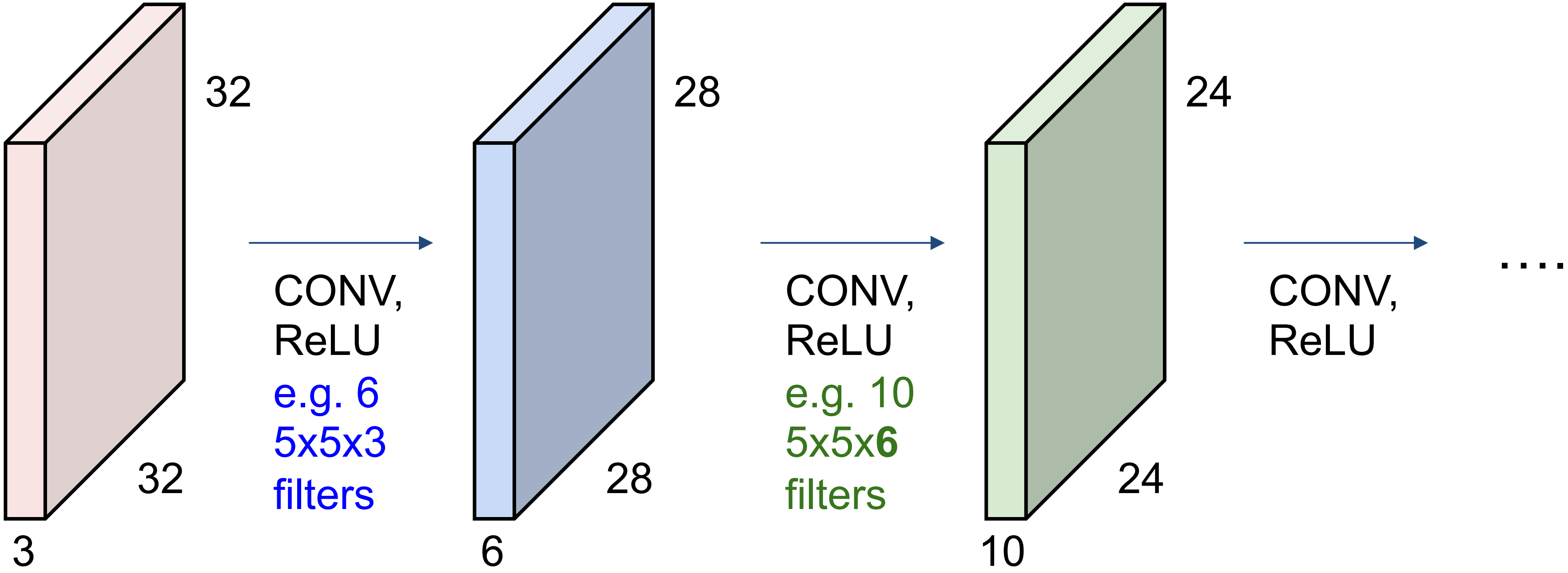
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

With padding of 1 x 1:

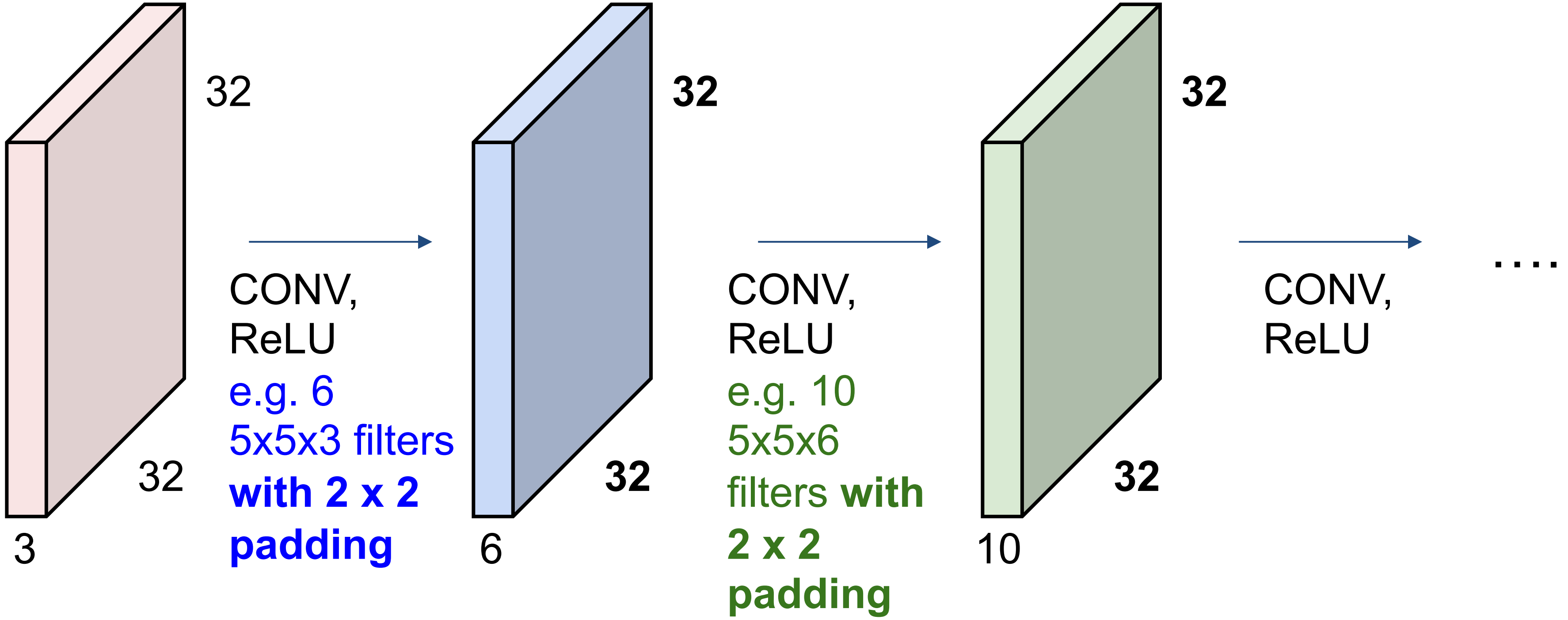
stride 3 => (7 – 3 **+ 2**)/3 + 1 = 3

Remember back to…

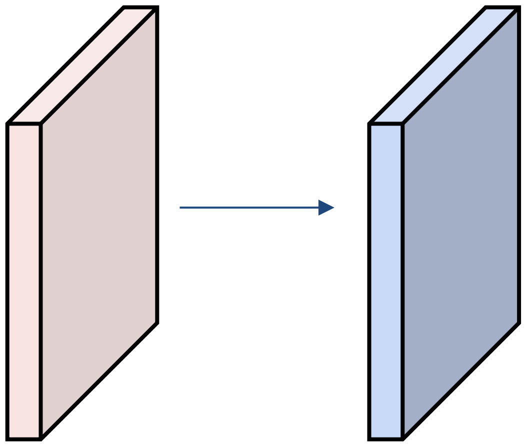
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



Remember back to…
With padding, we can keep the same spatial feature dimension throughout the convolution layers.



Examples time:

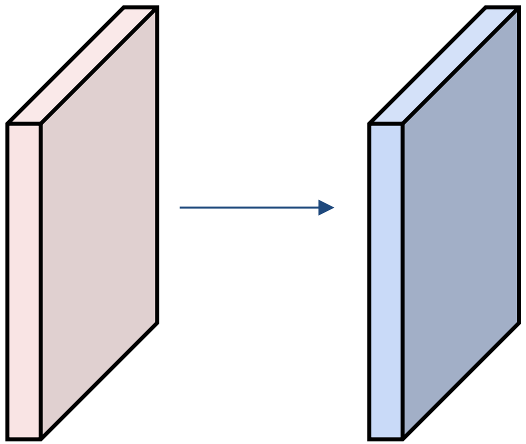


Input volume: **32x32x3**
Conv layer: 10 5x5 filters with stride 1, pad 2

Output volume size: ?

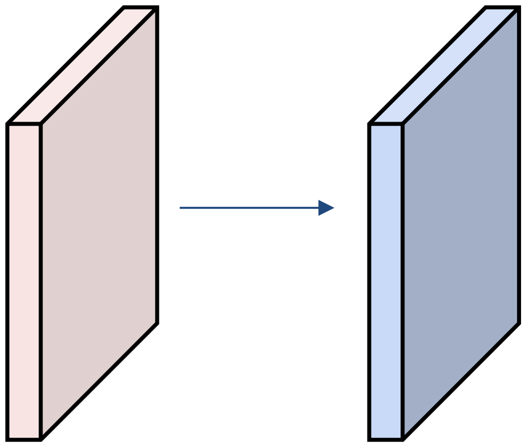
Examples time:

Input volume: **32x32x3**
Conv layer: 10 5x5 filters with stride 1, pad 2



Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

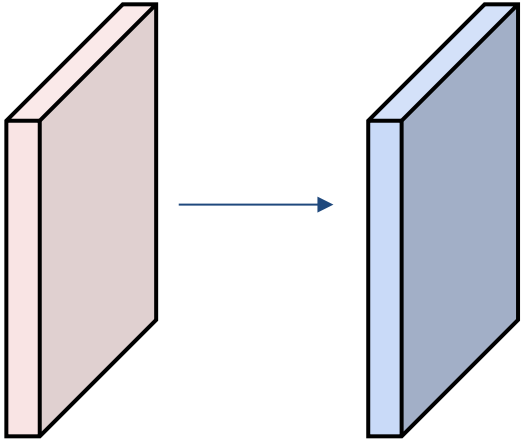
Examples time:



Input volume: **32x32x3**
Conv layer: 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 x H_1 x 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 x H_2 x 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 x H_1 x 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 x H_2 x K
where:

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

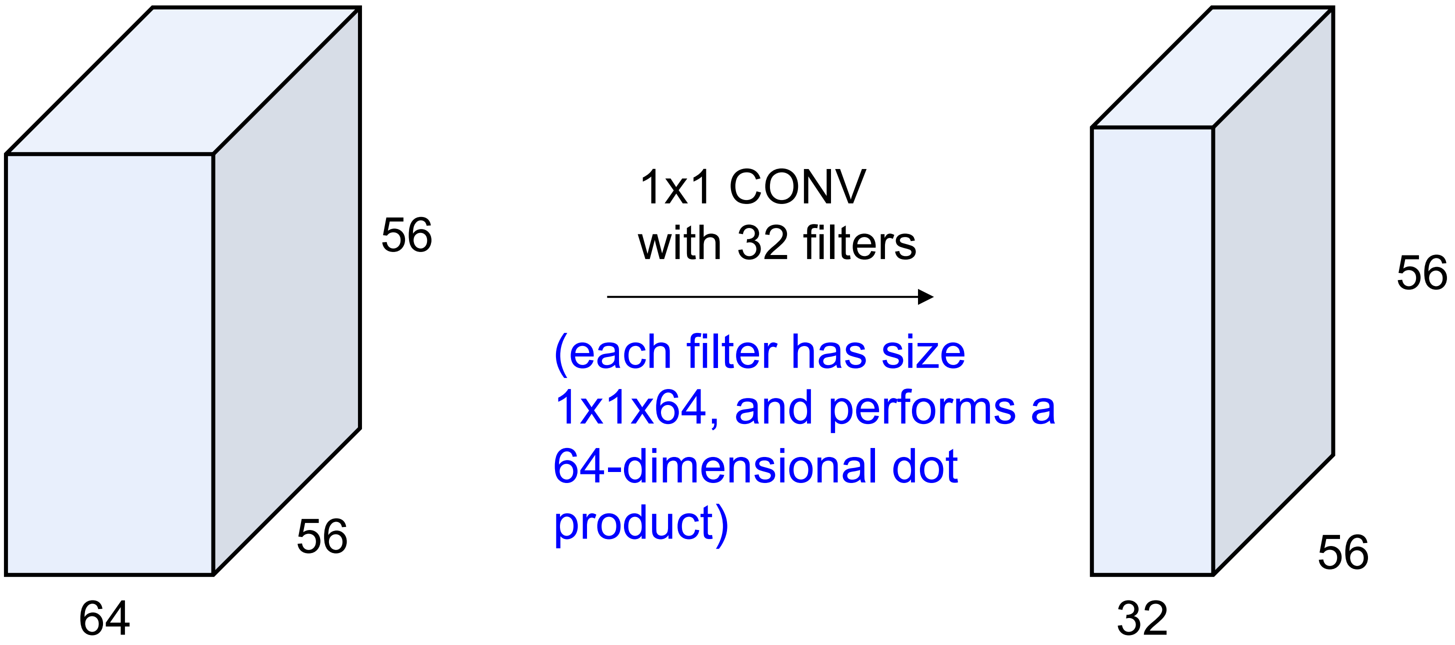
Number of parameters: F^2CK and K biases

Common settings:

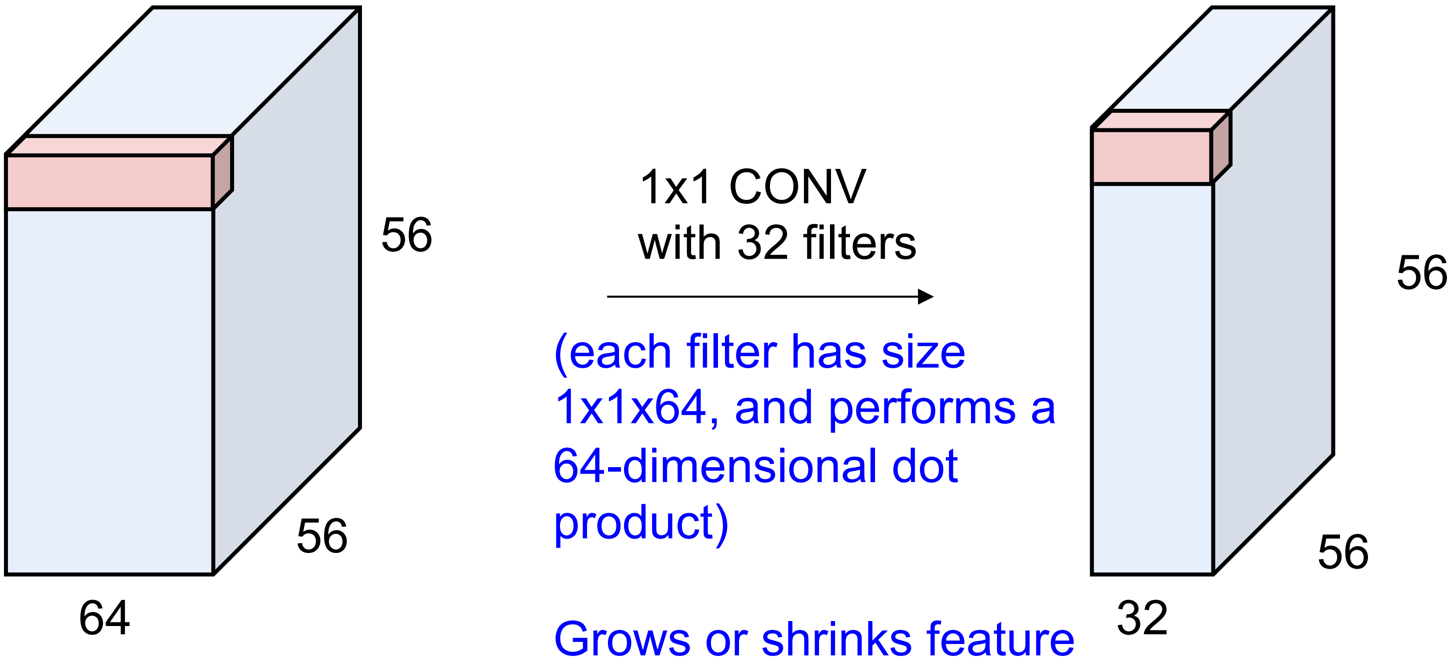
K = (powers of 2, e.g. 32, 64, 128, 512)

- 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)



(btw, 1x1 convolution layers make perfect sense)



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

Grows or shrinks feature channel dimension

Example: CONV layer in PyTorch

Conv layer needs 4 hyperparameters:
- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

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_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

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

where $\star$ 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

Next Time:

- Pooling
- Convolutional Neural Nets!