

CS 4644-DL / 7643-A: LECTURE 6

DANFEI XU

Topics:

- Convolutional Neural Networks: Past and Present
- Convolution Layers

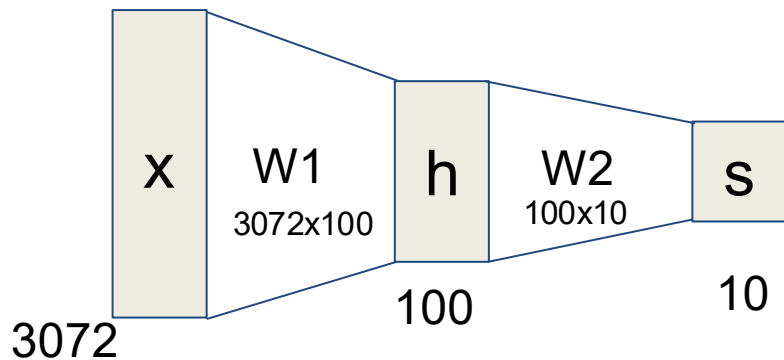
Administrative:

- Proposal template and prompt released.
- Proposal due Oct 7th 11:59pm (**No Grace Period**)
- Project lecture Wednesday
- Project helping session next Monday (Sep)
- Start finding a project team if you haven't!

Neural networks: hierarchical computation

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

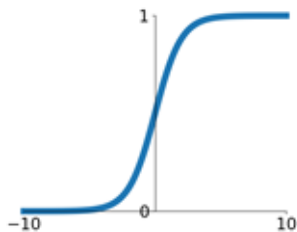


$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

Activation functions

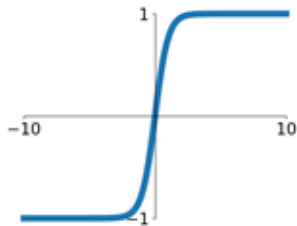
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



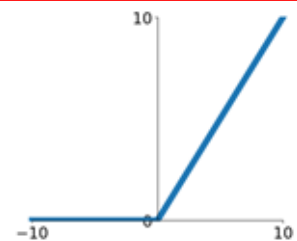
tanh

$$\tanh(x)$$



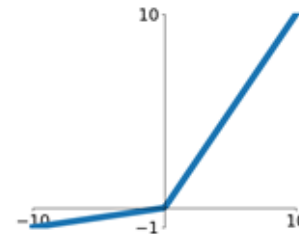
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

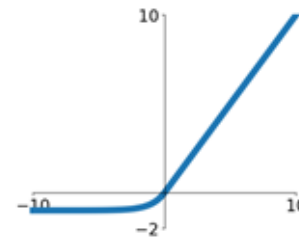


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



ReLU is a good default
choice for most problems

Neural Networks are Universal Function Approximators

Claim: Neural Networks with certain non-linear activation functions are universal function approximators.

The universal approximation theorem (Cybenko, G. 1989)

Theorem 1. *Let σ be any continuous discriminatory function. Then finite sums of the form*

$$G(x) = \sum_{j=1}^N \alpha_j \sigma(y_j^T x + \theta_j) \quad (2)$$

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum, $G(x)$, of the above form, for which

$$|G(x) - f(x)| < \varepsilon \quad \text{for all } x \in I_n.$$

Plain English: as long as the activation function is sigmoid-like and the function to be approximated is continuous, there exists a neural network with a single hidden layer that can approximate it with certain error (bounded uniform norm).

Jacobians

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

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

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?



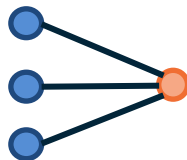
Vector to Scalar

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

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?



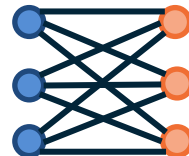
Vector to Vector

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

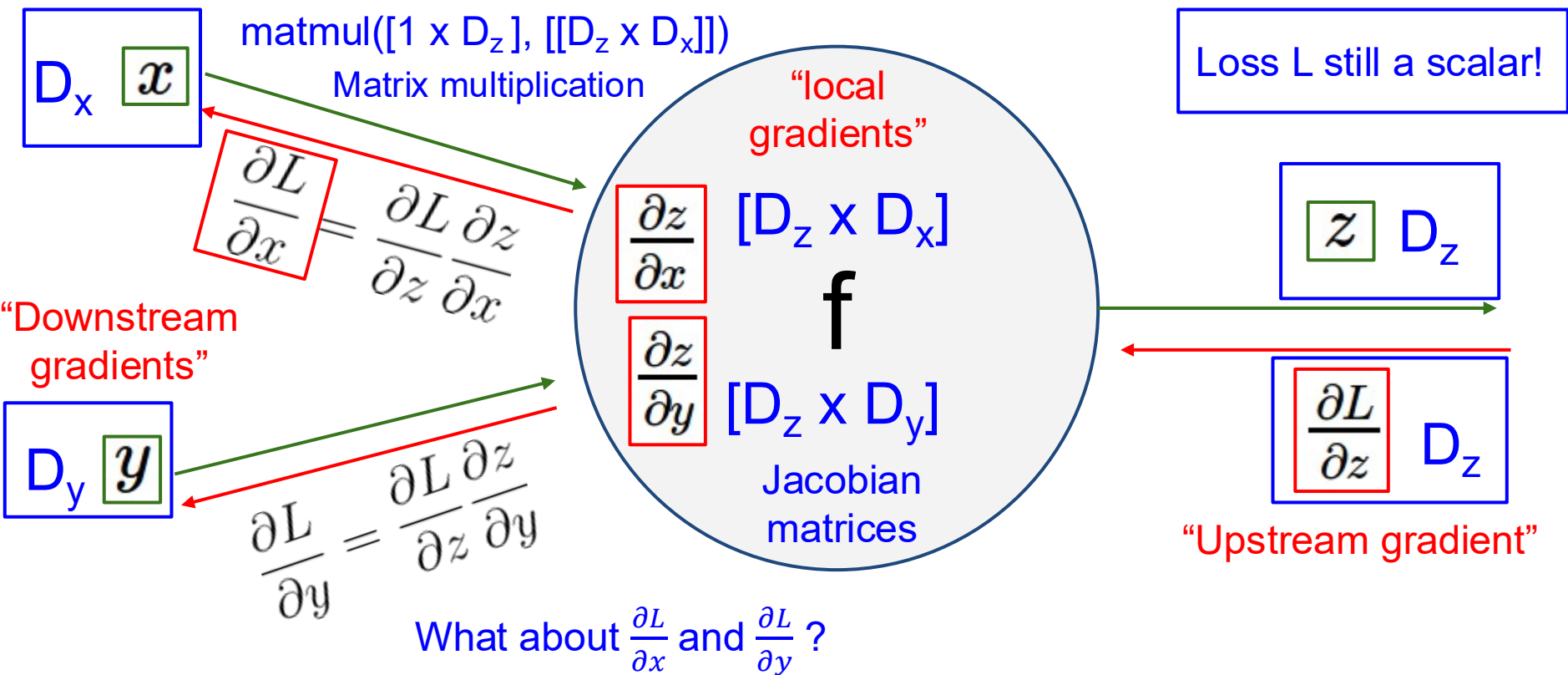
Derivative is **Jacobian**:

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

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



Backprop with Vectors



Backprop with Vectors

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

4D input x:

$\begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix}$

$$f(x) = \max(0, x) \quad (\text{elementwise})$$

4D output z:

$\begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$

4D dL/dx :

$\begin{bmatrix} 4 & 0 & 5 & 0 \end{bmatrix}$

$[dL/dz]$

$\begin{bmatrix} 4 & -1 & 5 & 9 \end{bmatrix}$

$[dz/dx]$

$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$

$[dL/dz]$

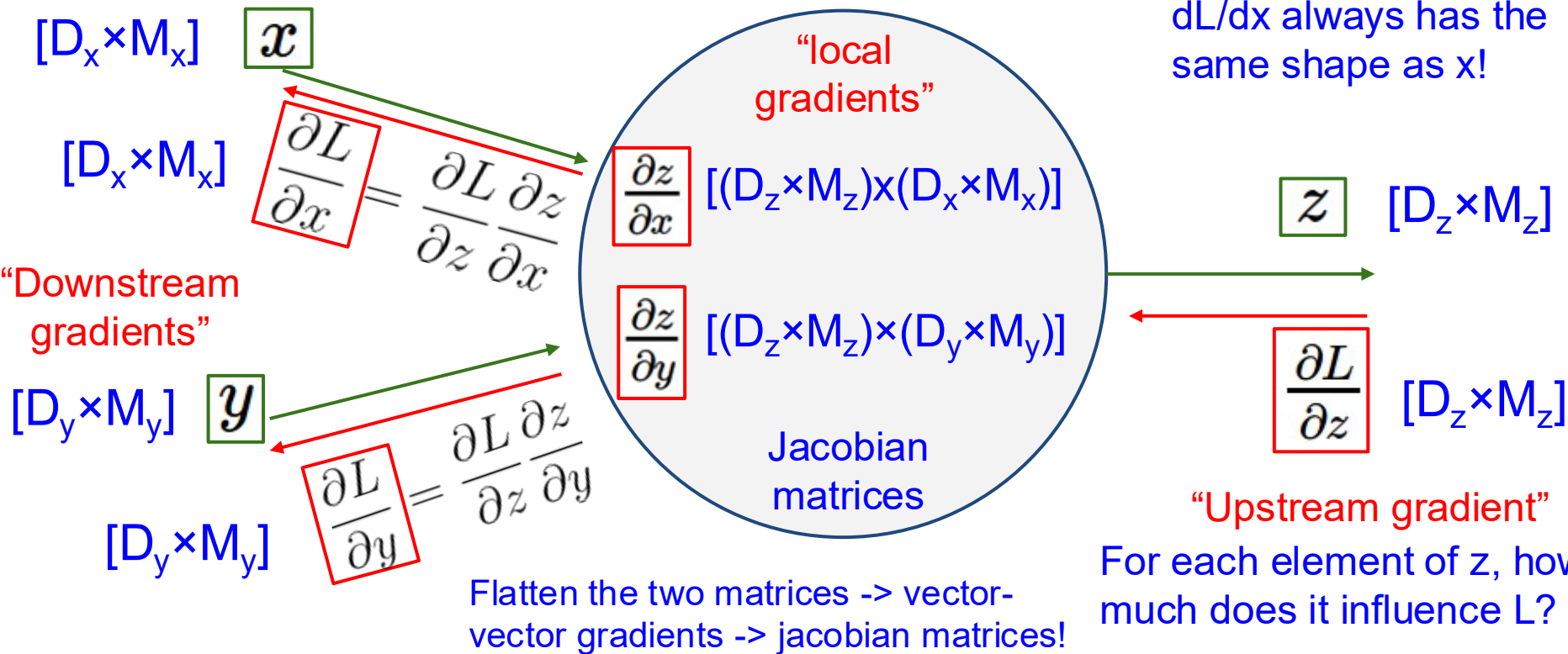
$\begin{bmatrix} 4 & -1 & 5 & 9 \end{bmatrix}$

Upstream gradient

Backprop with Matrices (or Tensors)

Loss L still a scalar!

dL/dx always has the same shape as x !



Backprop with Matrices

$$x: [N \times D]$$

$$\begin{bmatrix} 2 & \boxed{1} & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

$$w: [D \times M]$$

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & \boxed{3} & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

$$y: [N \times M]$$

$$\begin{bmatrix} -1 & -2 & \boxed{2} & 6 \\ 5 & 2 & 11 & 7 \end{bmatrix}$$

$$dL/dy: [N \times M]$$

$$\begin{bmatrix} \boxed{2} & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$$

By similar logic:

$$[N \times D] \quad [N \times M] \quad [M \times D]$$

$$\frac{\partial L}{\partial x} = \left(\frac{\partial L}{\partial y} \right) w^T$$

$$[D \times M] \quad [D \times N] \quad [N \times M]$$

$$\frac{\partial L}{\partial w} = x^T \left(\frac{\partial L}{\partial y} \right)$$

For a neural net layer with
 $N=64$, $D=M=4096$
 The large matrix (W) takes
 up to 0.13 GB memory

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

Today: Convolutional Neural Networks

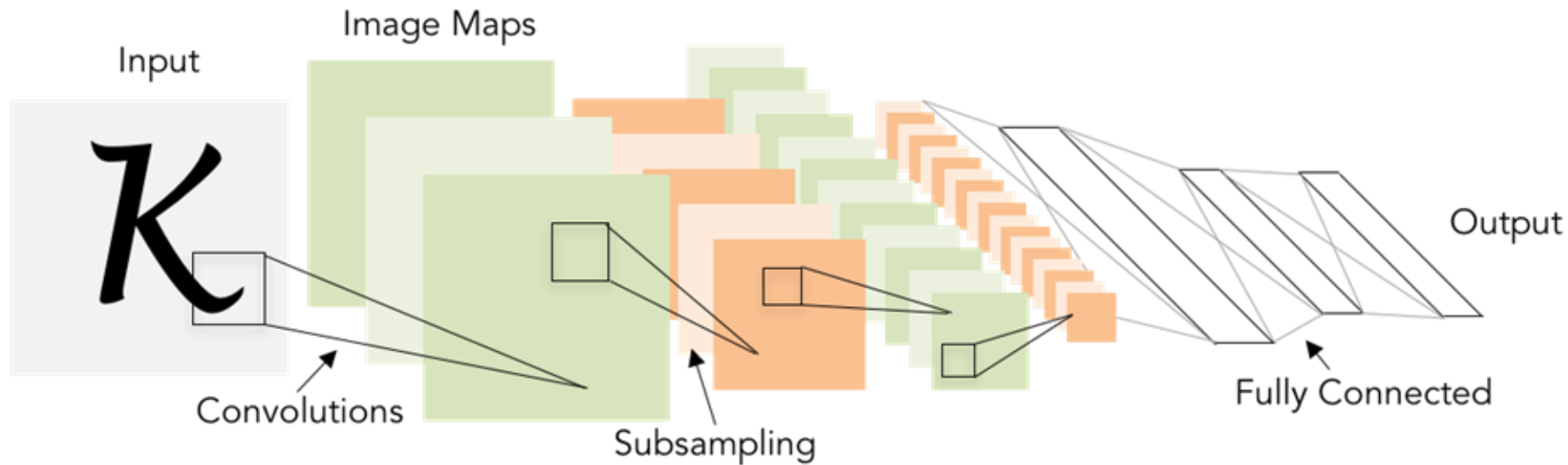


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

A bit of history...

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

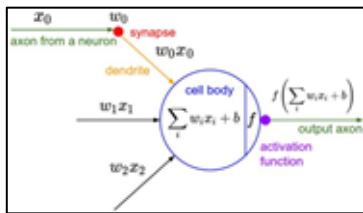
The machine was connected to a camera that used 20×20 photocells to produce a 400-pixel image.

recognized
letters of the alphabet

update rule:

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

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

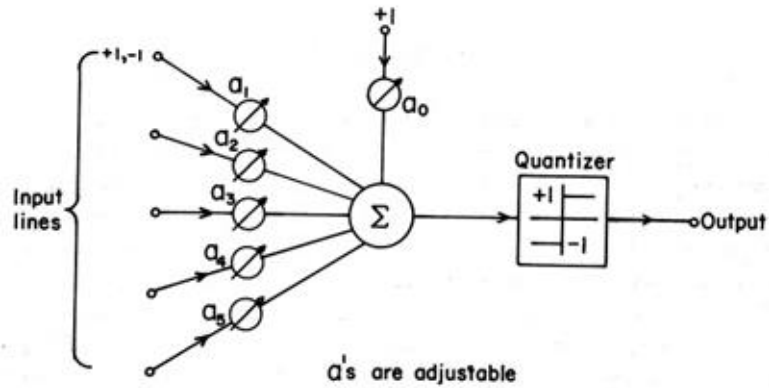


Frank Rosenblatt, ~1957: Perceptron



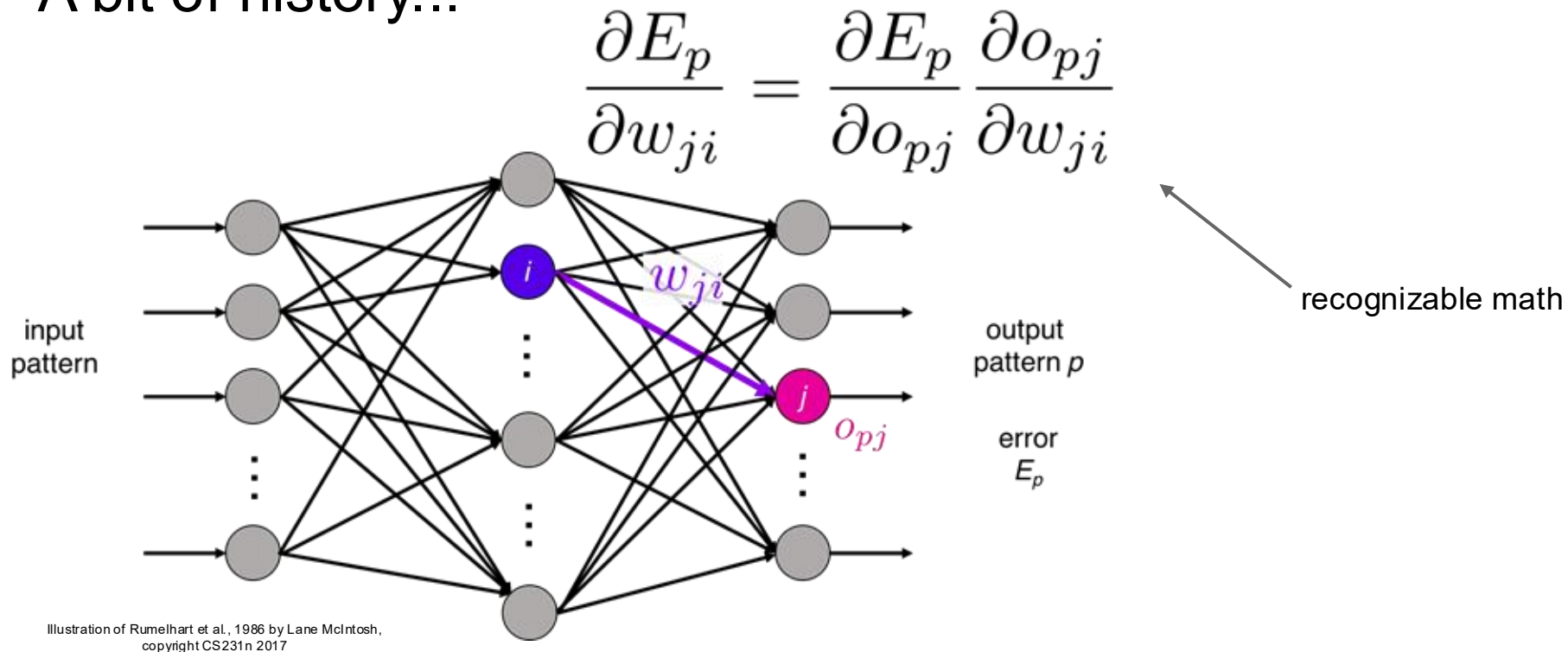
[This image](#) by Rocky Acosta is licensed under [CC-BY 3.0](#)

A bit of history...



Widrow and Hoff, ~1960: Adaline/Madaline

A bit of history...



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

A bit of history...

[Hinton and Salakhutdinov 2006]

Reinvigorated research in Deep Learning

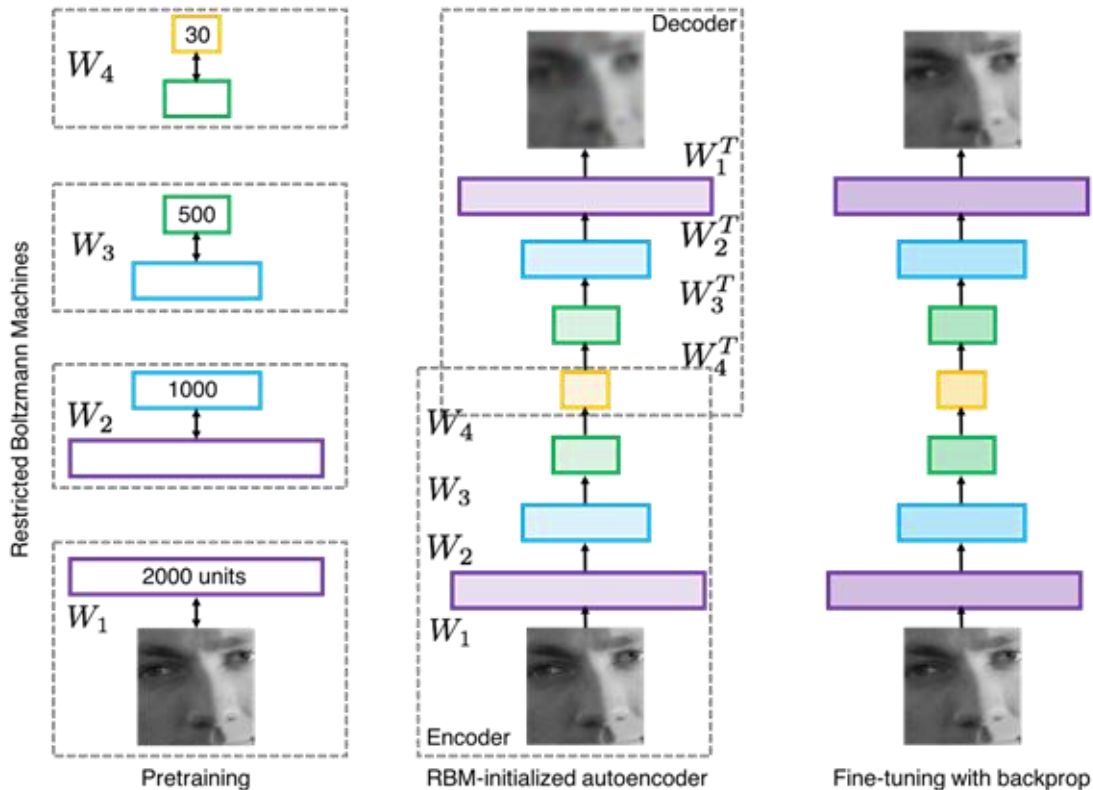


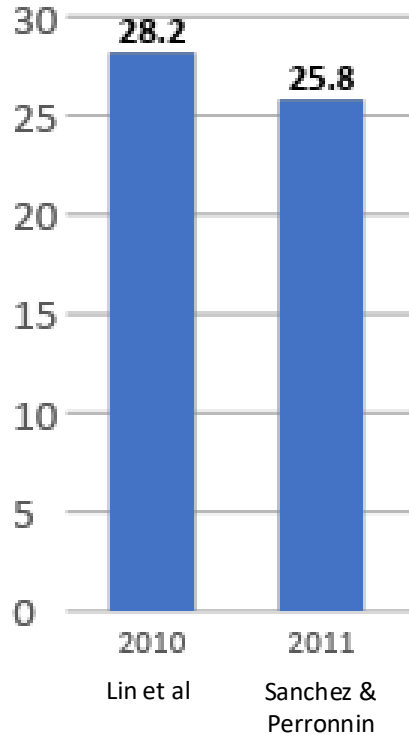
Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

A bit of history... ImageNet (Deng et al., 2009)

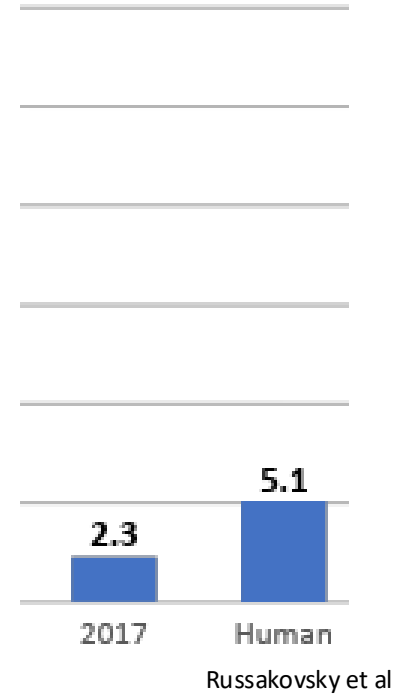


The **ImageNet** dataset contains 14,197,122 annotated images according to the WordNet hierarchy. ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a benchmark for image classification and object detection based on the dataset.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ConvNet Happened



Elements of Deep Learning (circa 2012)

Algorithms

(Stochastic) Gradient Descent, Backpropagation

Architectures / Models

(Deep) Convolutional Neural Networks

Data

CIFAR10, CIFAR100, Pascal VOC, ImageNet

Computation Hardware

GPU (NVIDIA GeForce 600 series)

A bit of history:

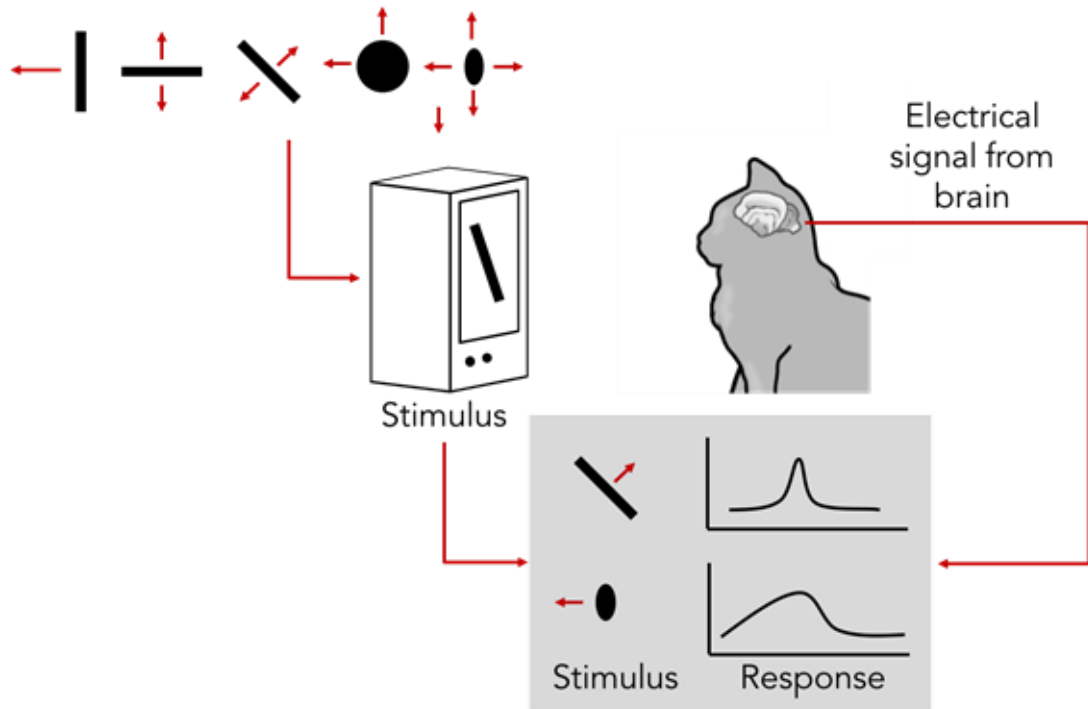
Hubel & Wiesel, 1959

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

1962

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

1968...



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

Hierarchical organization

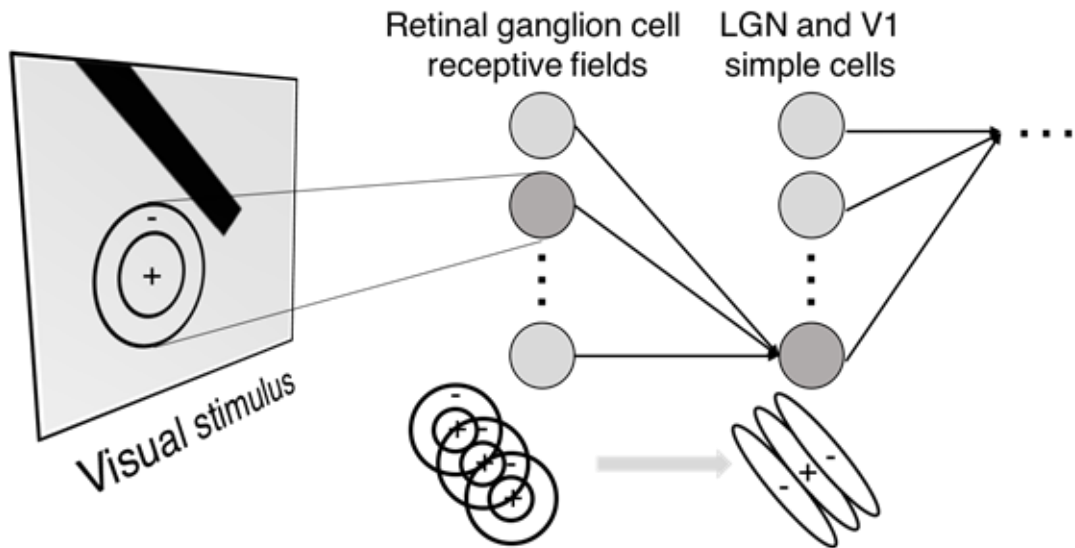


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

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



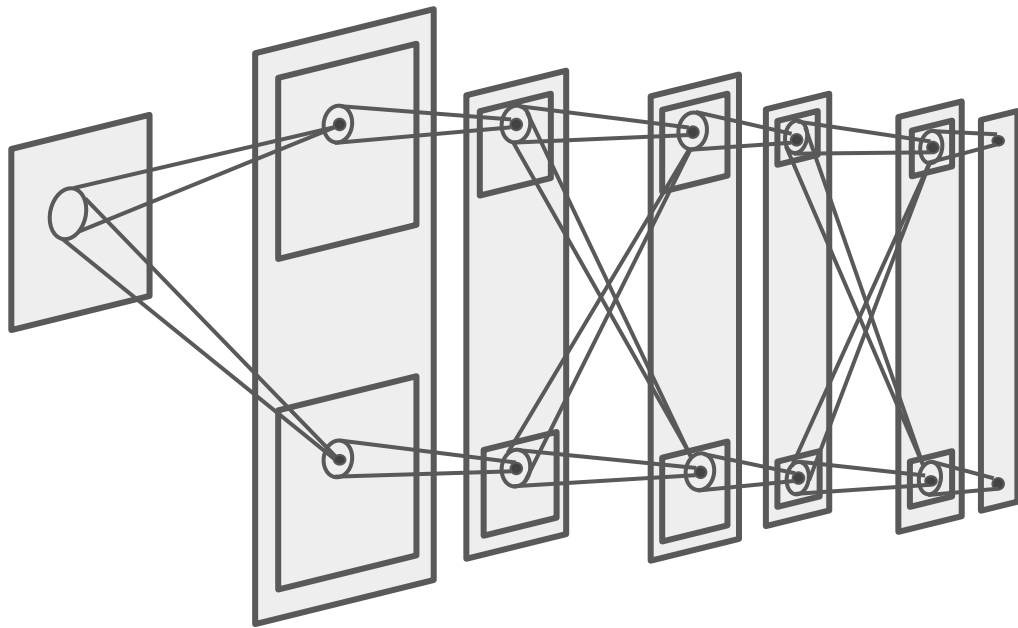
Response
(end point)

A bit of history:

Neocognitron

[Fukushima 1980]

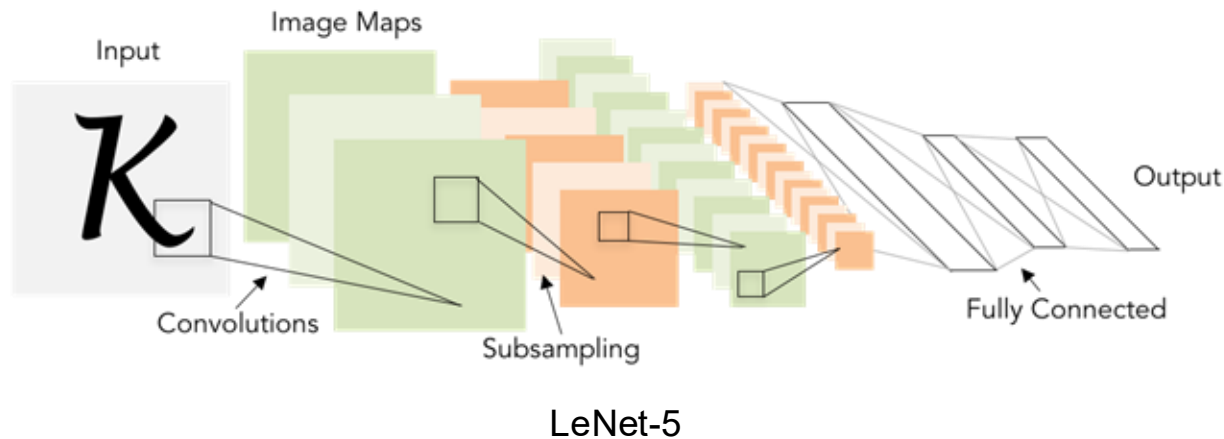
“sandwich” architecture (SCSCSC...)
simple cells: modifiable parameters
complex cells: perform pooling



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]

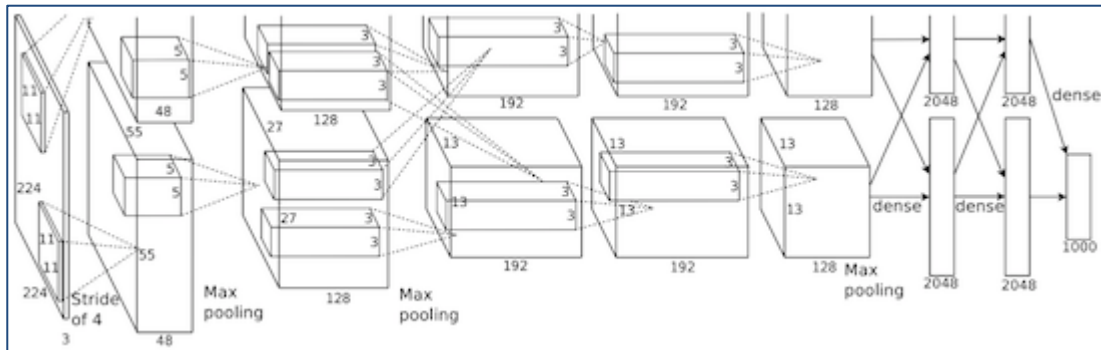
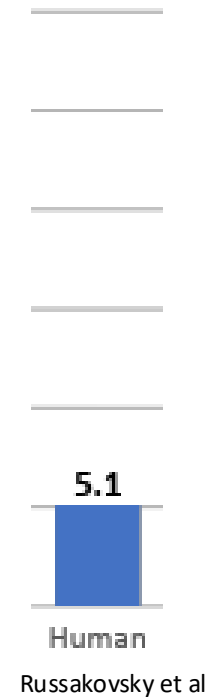
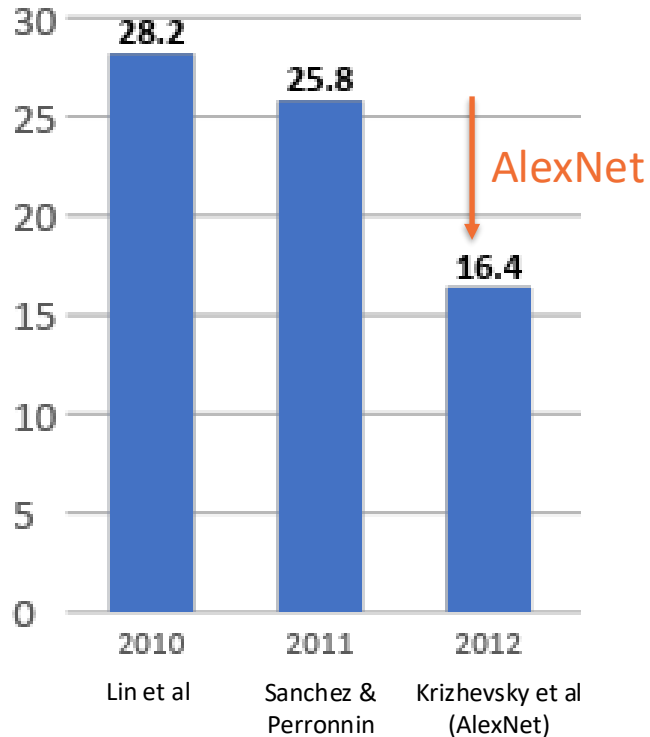


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

“AlexNet”

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

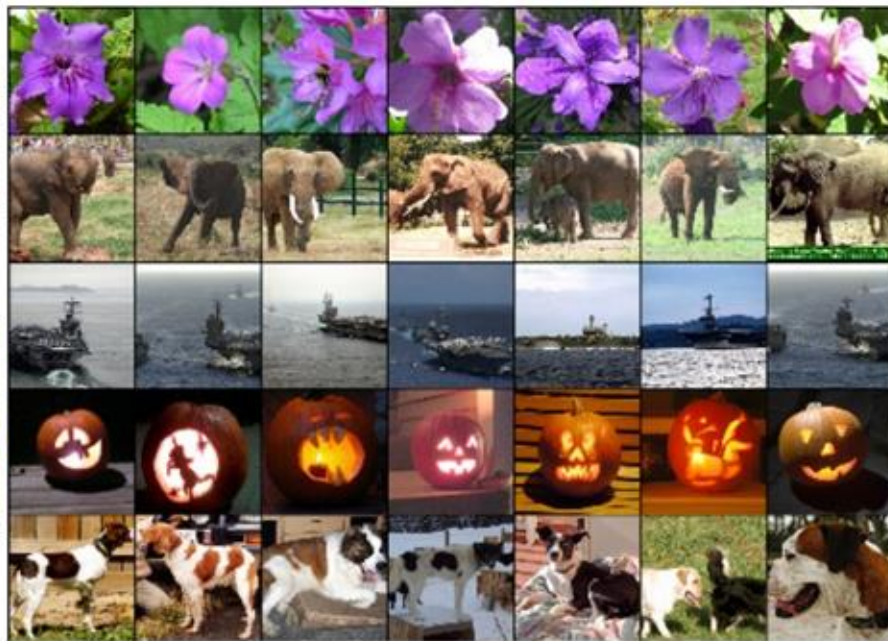


Fast-forward to today: ConvNets are everywhere

Classification



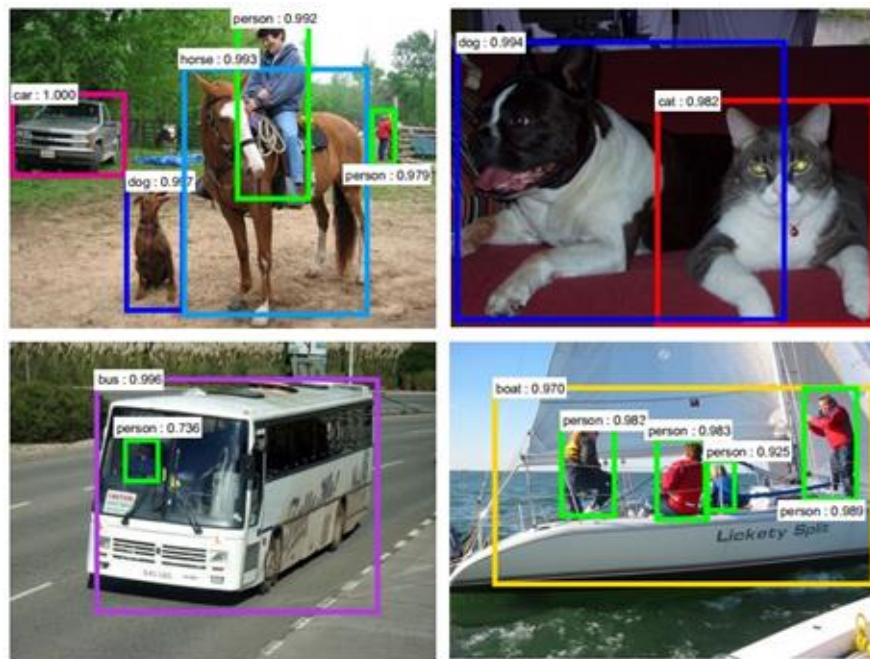
Retrieval



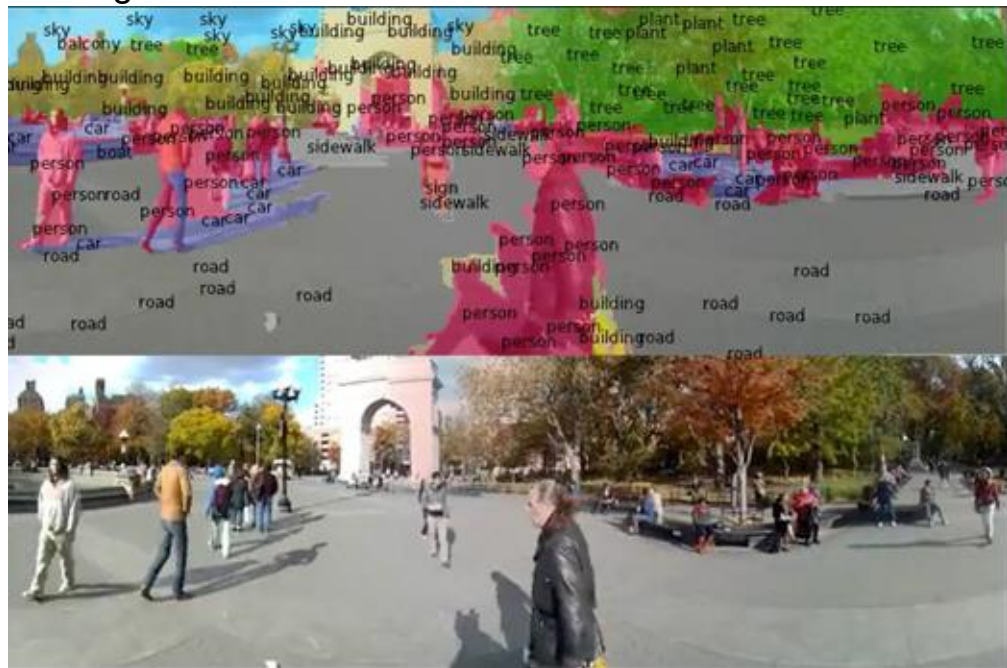
Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Fast-forward to today: ConvNets are everywhere

Detection



Segmentation



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

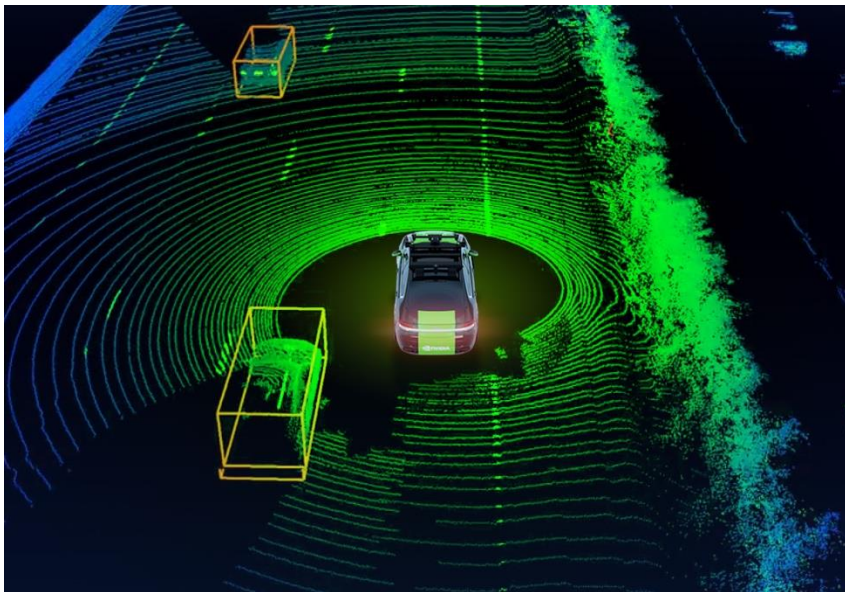
[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Figures copyright Clement Farabet, 2012. Reproduced with permission.

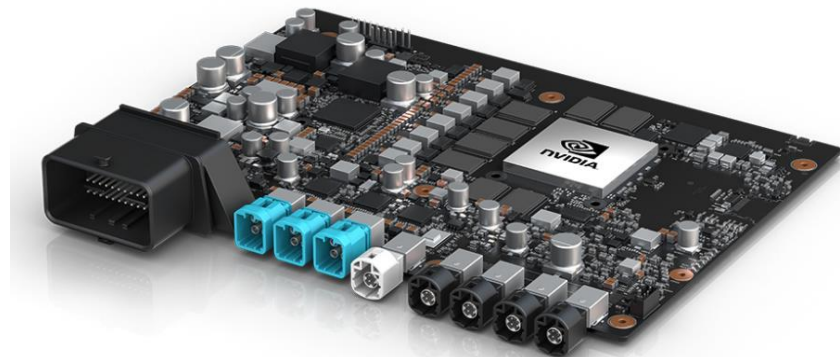
[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere

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/>



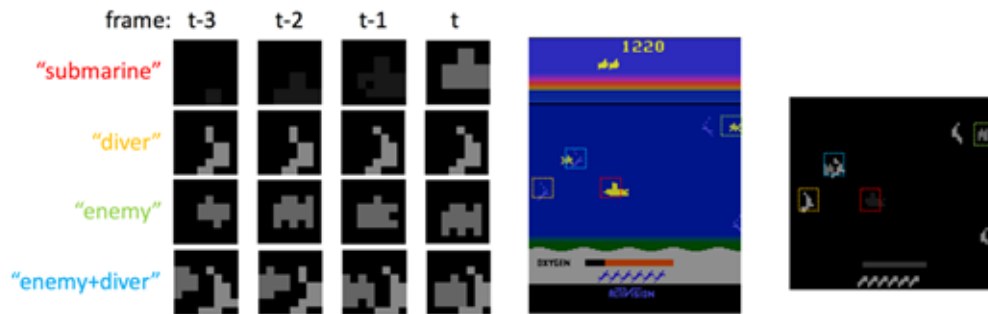
<https://www.nvidia.com/en-us/self-driving-cars/>

Fast-forward to today: ConvNets are everywhere

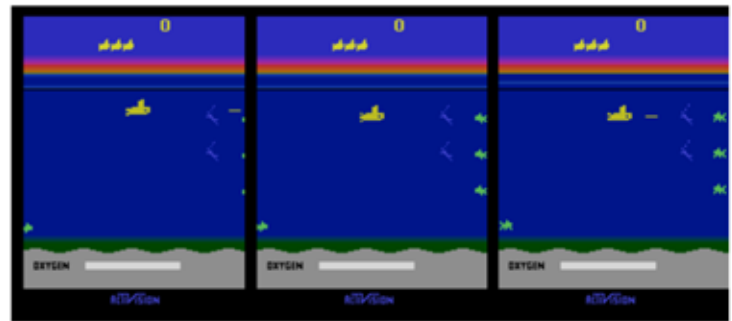


Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]



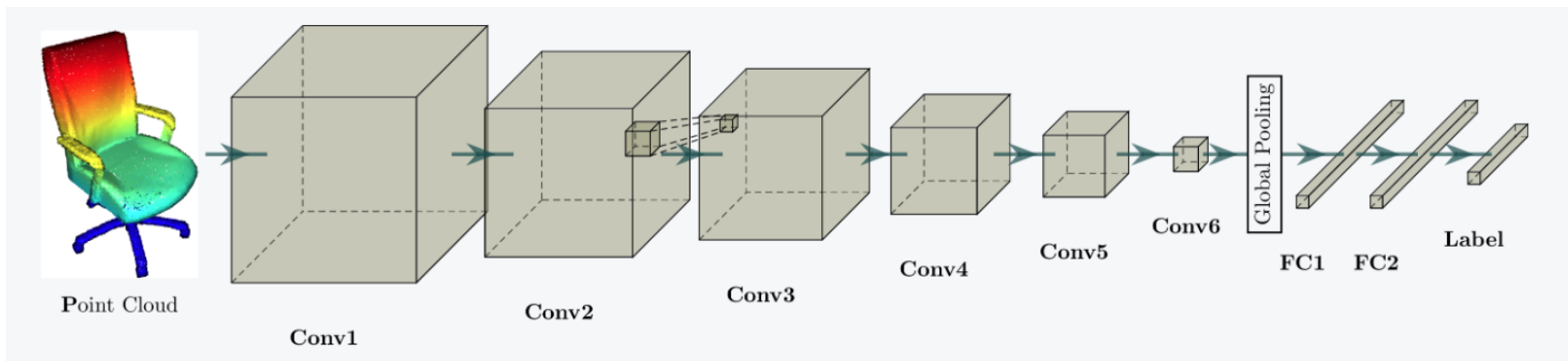
[Guo et al. 2014]



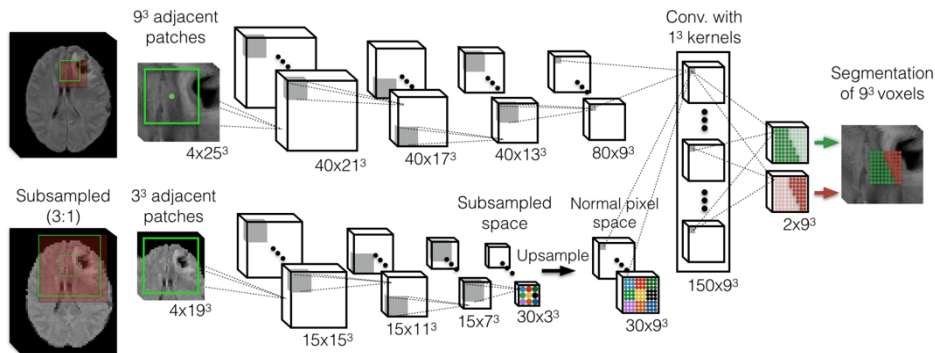
Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

Fast-forward to today: ConvNets are everywhere

Generalized convolution: **spatial convolution**



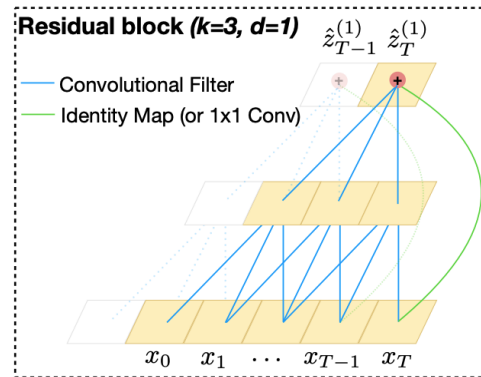
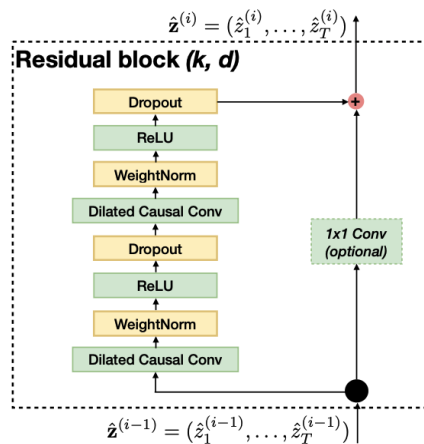
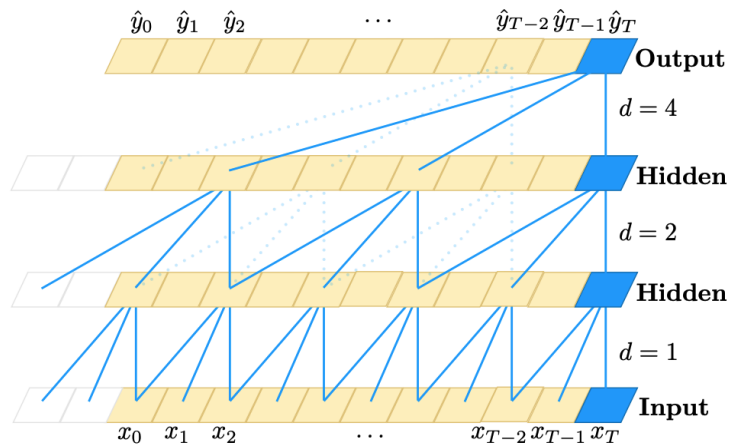
Choi et al., 2019



Kamnitsas et al., 2015

Fast-forward to today: ConvNets are everywhere

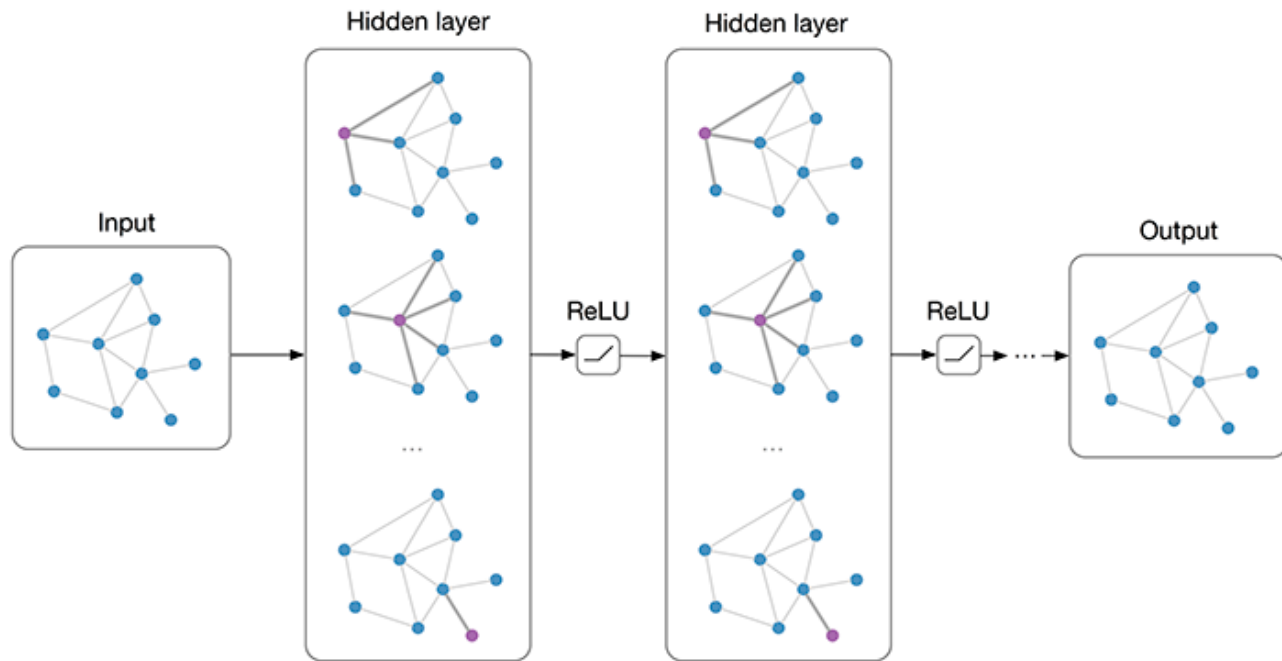
Generalized convolution: **temporal convolution**



Bai et al., 2018

Fast-forward to today: ConvNets are everywhere

Generalized convolution: **graph convolution**



Kipf et al., 2017

No errors



A white teddy bear sitting in the grass

Minor errors



A man in a baseball uniform throwing a ball

Somewhat related



A woman is holding a cat in her hand

Image-to-text

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



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

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](#)

Text-to-Image



“An avocado armchair”

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

Convolutional Neural Networks

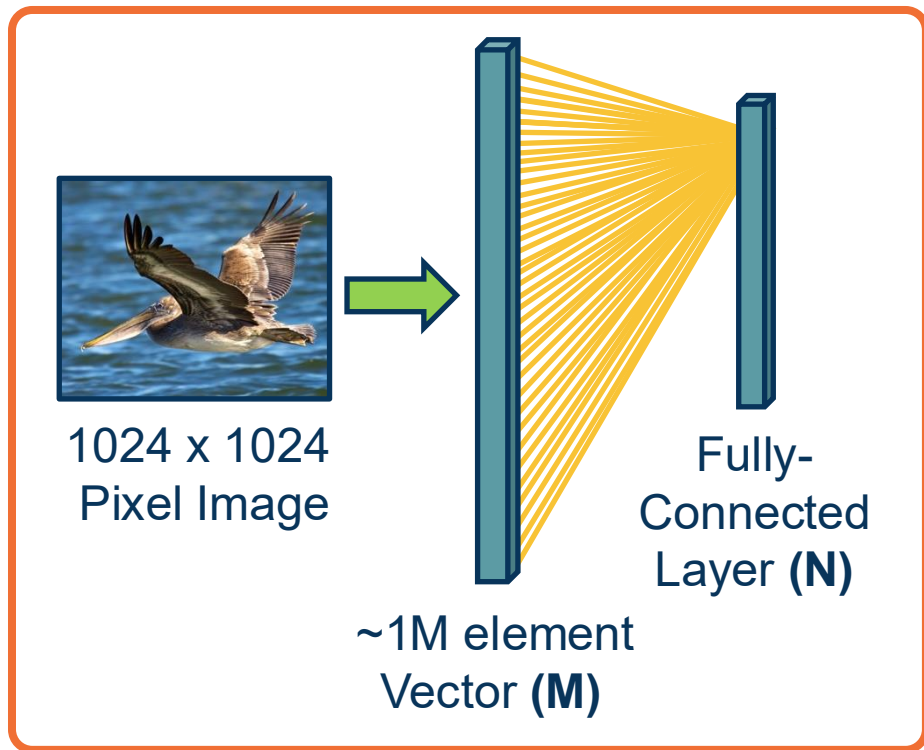
Convolution Operations

Convolution as a Neural Network Operator

Parameter Sharing

Convolution Layer

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



Q: 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?

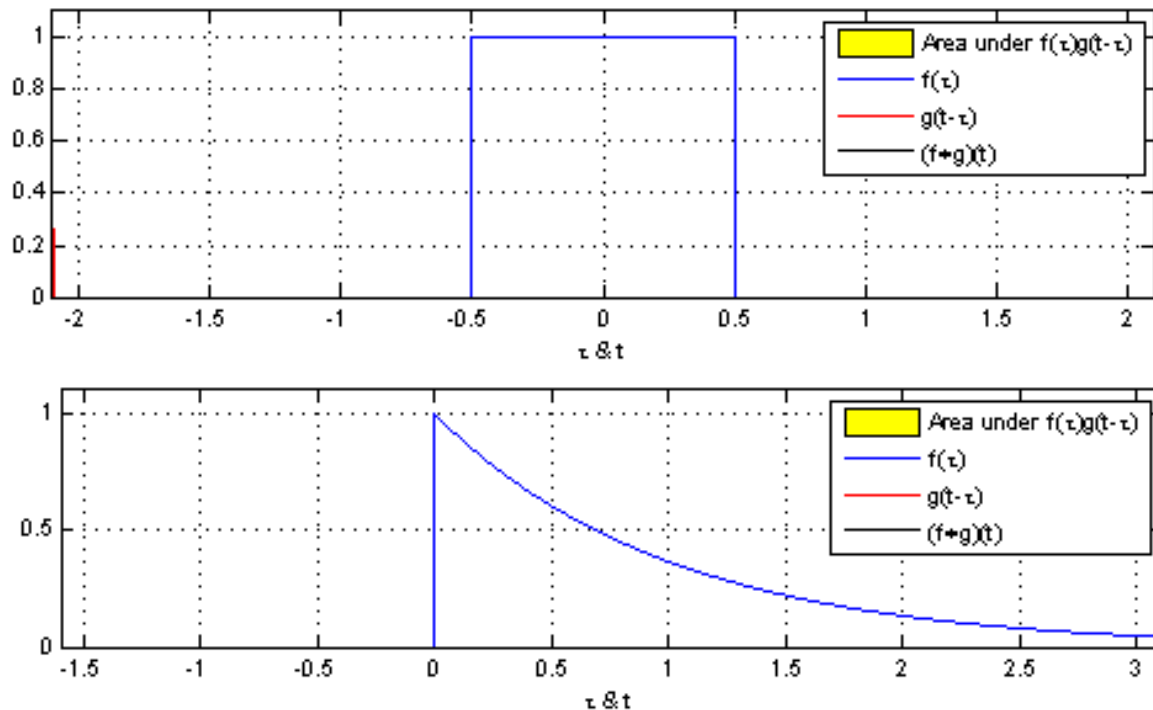
Convolution: A 1D Visual Example

Input function: $f()$

Kernel/filter function: $g()$

Convolution (response): $f * g()$

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau.$$



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

Convolution

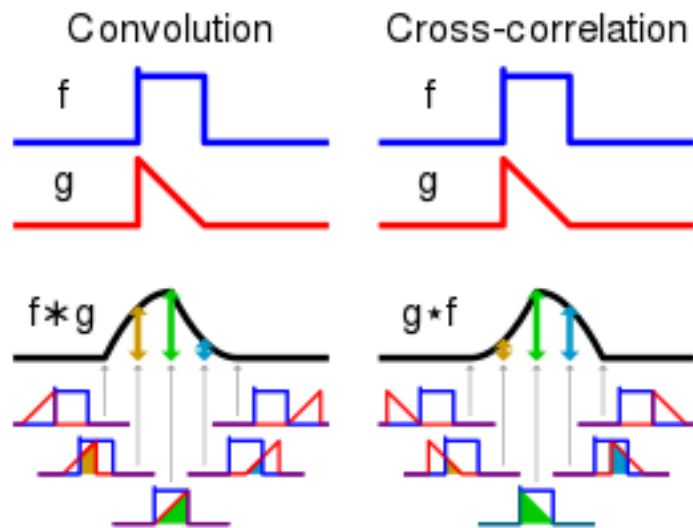
$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau.$$

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

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

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

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



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

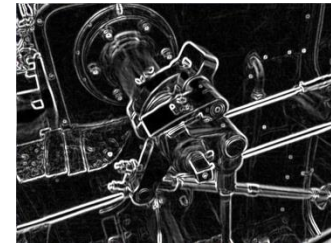
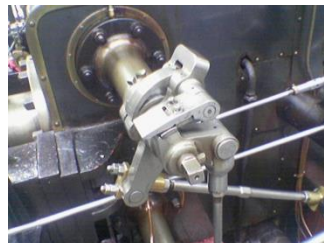
Convolution in Computer Vision (non-Deep)



Convolution with Gaussian Filter (Gaussian Blur)

$$\frac{1}{273}$$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1



Convolution with Sobel Filter (Edge Detection)

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A}$$
$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

Intuition for pattern recognition and learning using convolution

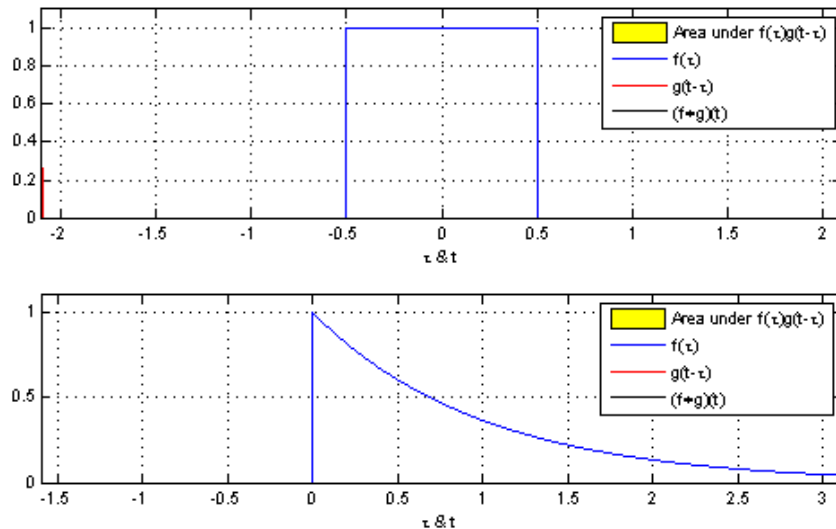
$g()$: filter / pattern template

$f()$: signal / observed data

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

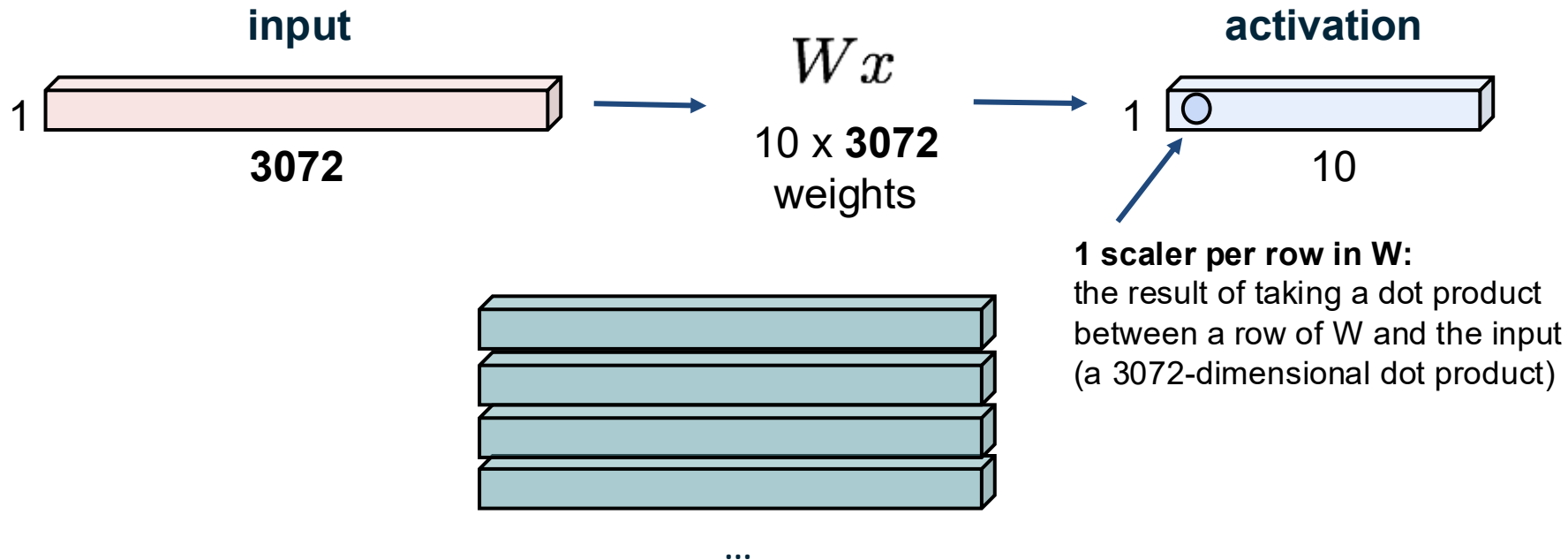
For Convolution Layers in NN:

- $g()$ as the **weights** to learn
- $f()$ as the **input** to the layer (e.g., images / features) *From <https://en.wikipedia.org/wiki/Convolution>*
- $f*g()$ as the **output** of the layer (result of convolution)
- Discrete instead of continuous convolution (sum instead of integral)
- $g()$ and $f()$ may be N -dimensional, where $N \geq 1$



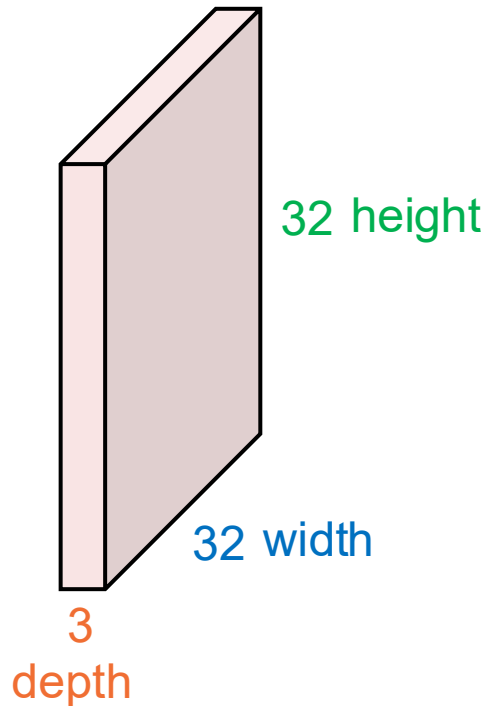
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



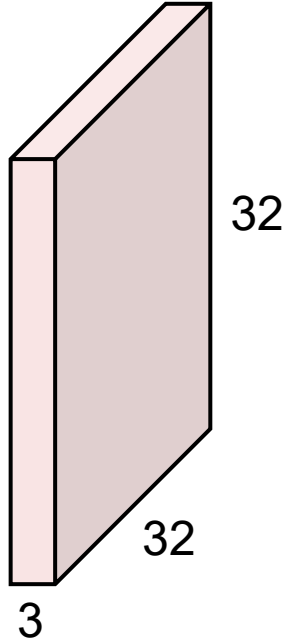
2-D Convolution Layer

32x32x3 image -> preserve spatial structure



2-D Convolution Layer

32x32x3 image



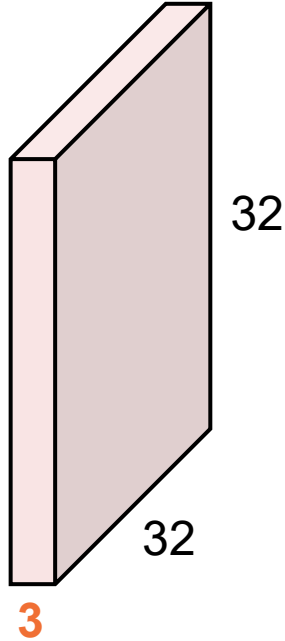
5x5x3 filter



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

2-D Convolution Layer

32x32x3 image



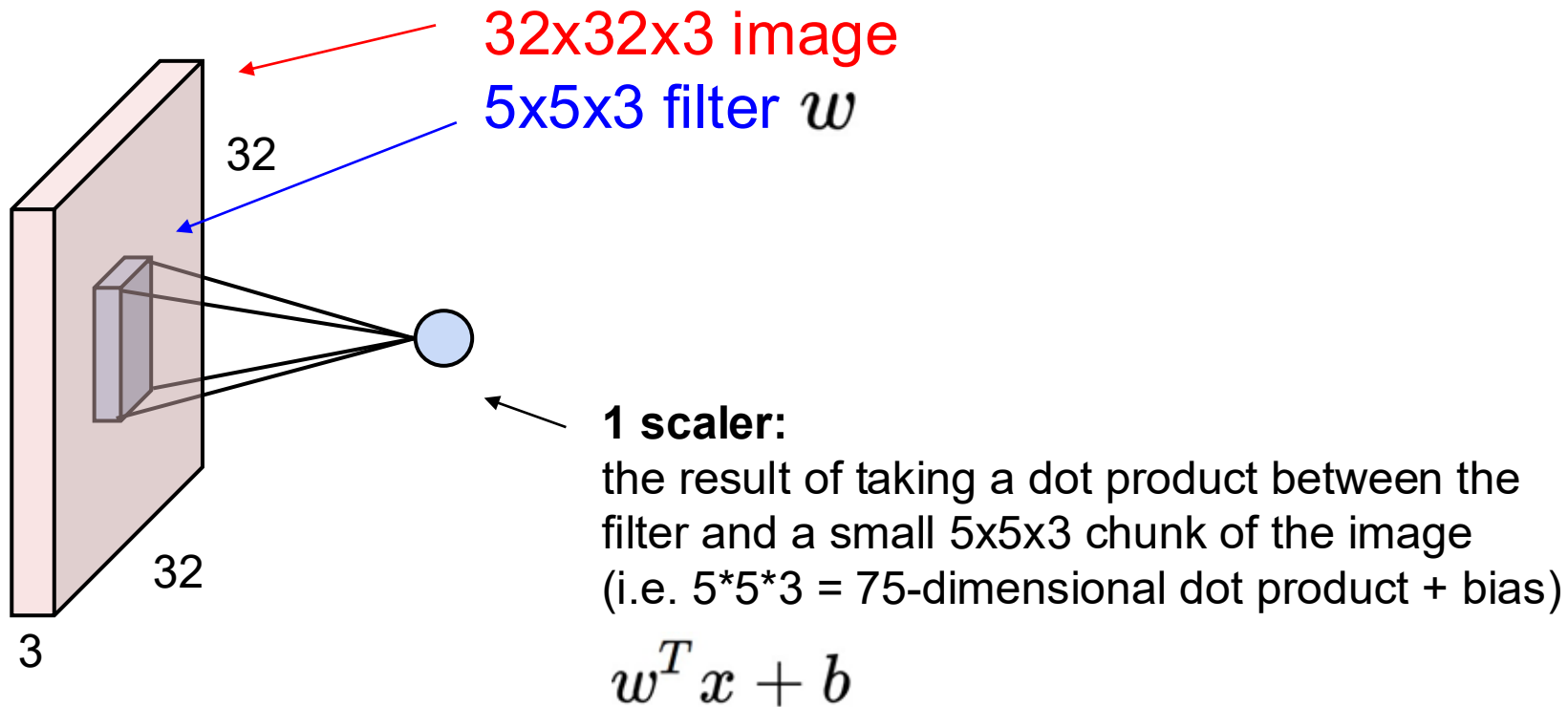
Filters always extend the full depth of the input volume

5x5x3 filter

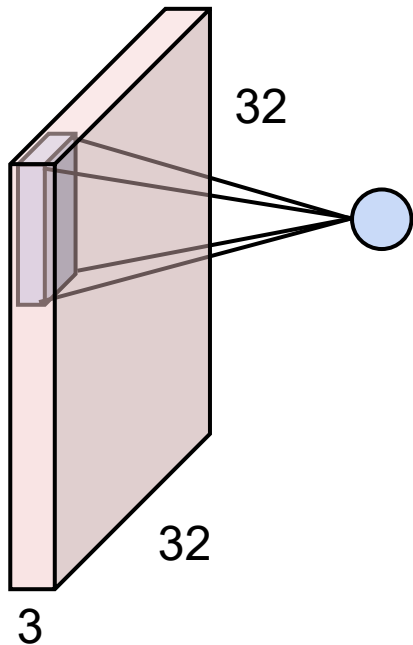


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

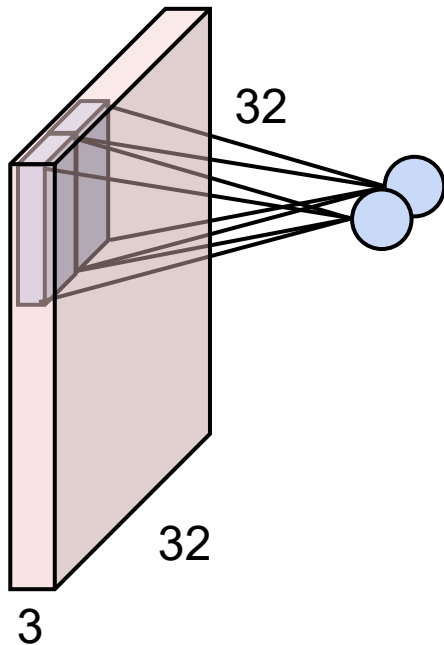
2-D Convolution Layer



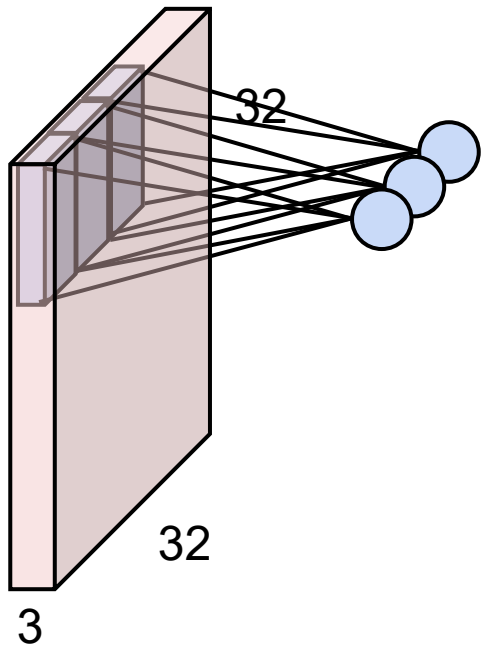
2-D Convolution Layer



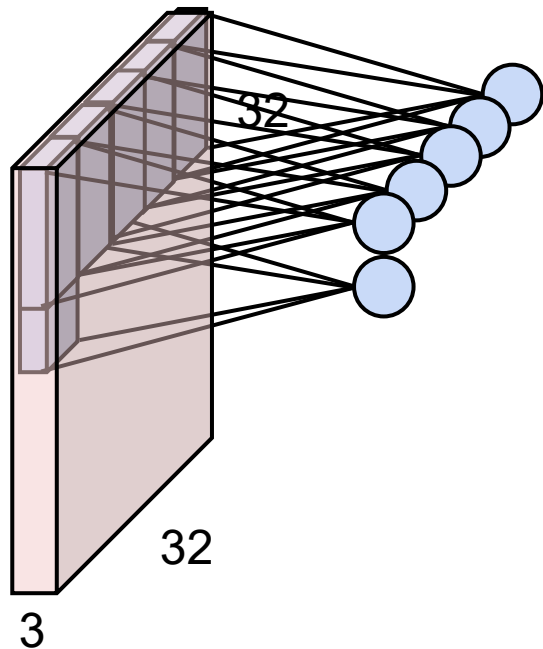
2-D Convolution Layer



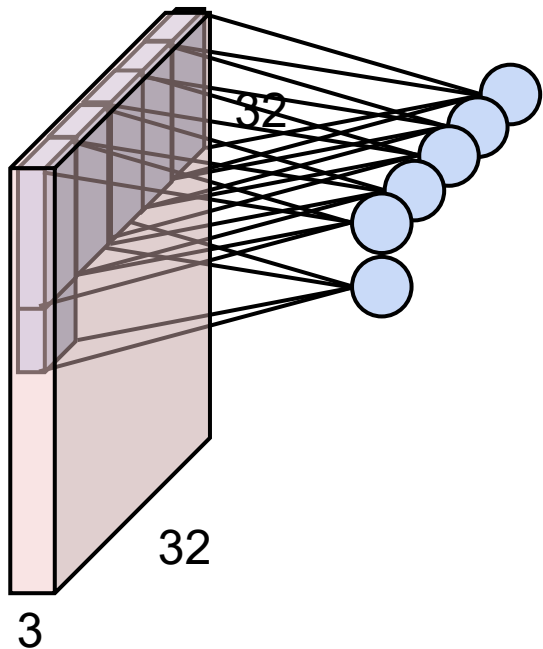
2-D Convolution Layer



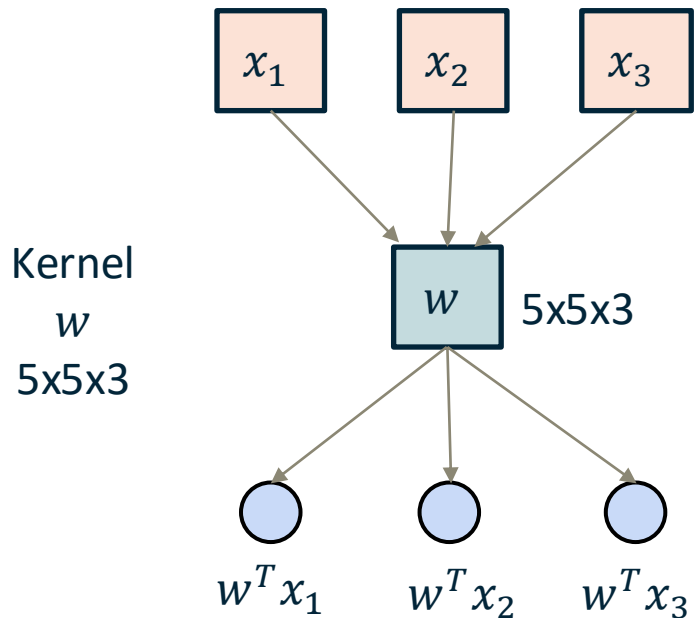
2-D Convolution Layer



Parameter Sharing

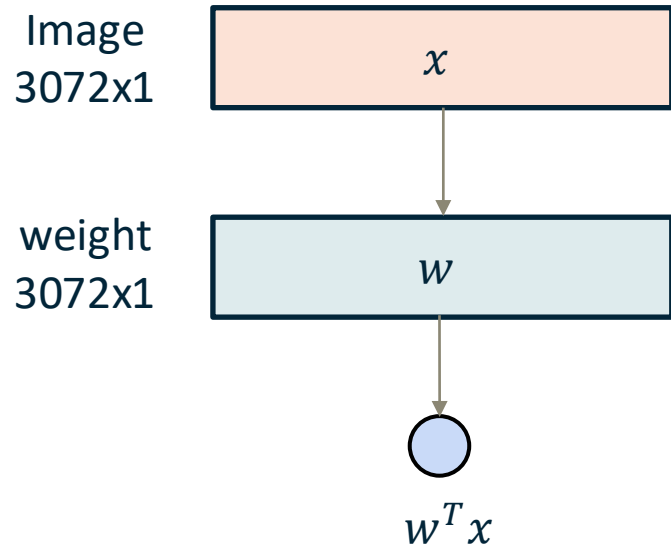


The same conv kernel applied across spatial locations!

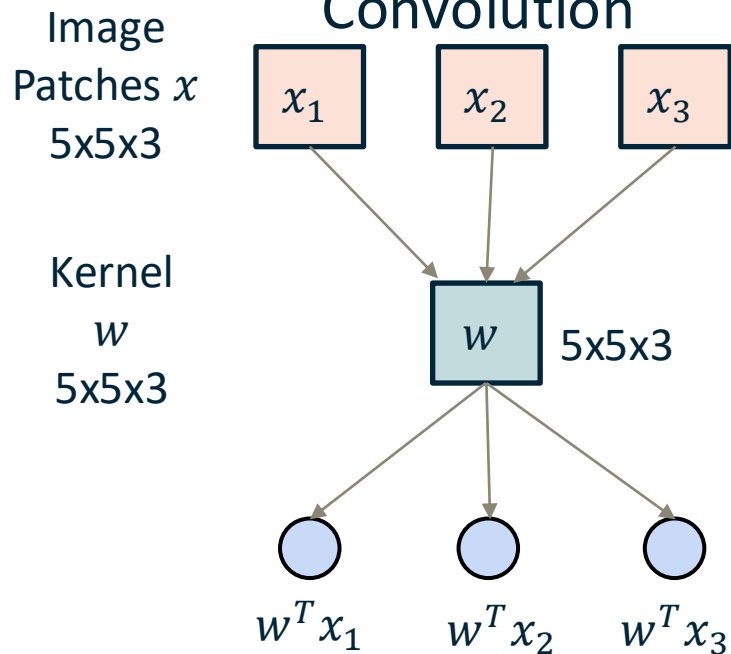


Parameter Sharing

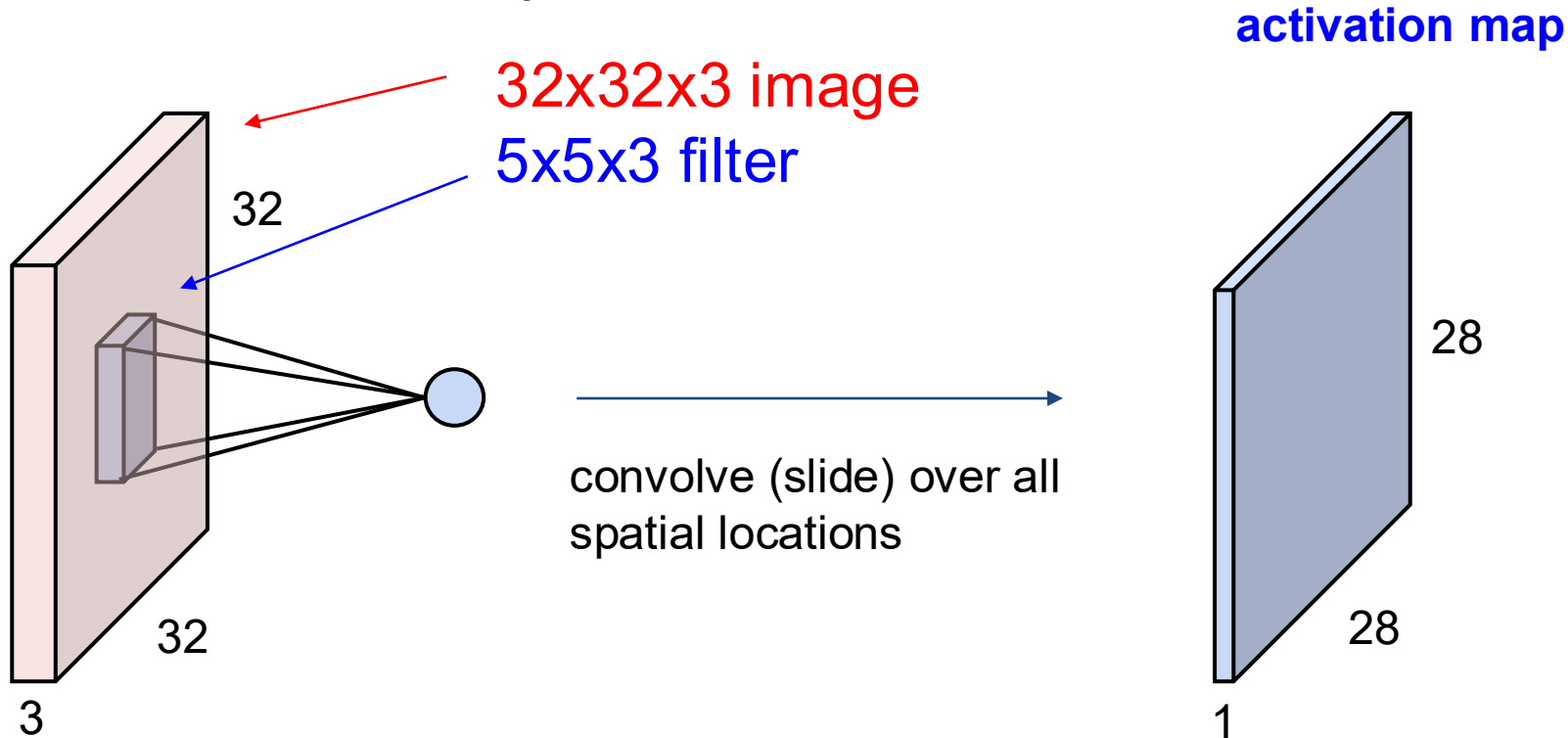
Fully connected layer



Convolution

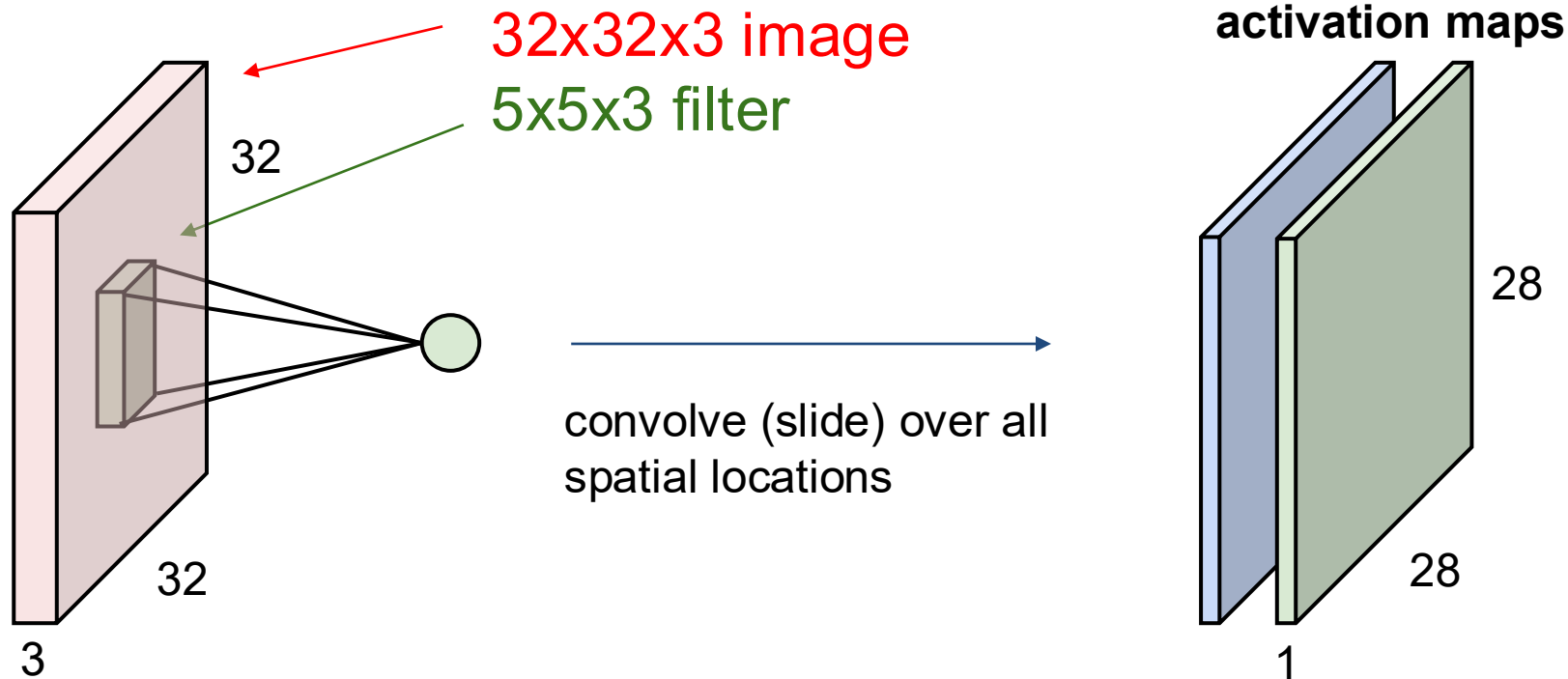


2-D Convolution Layer

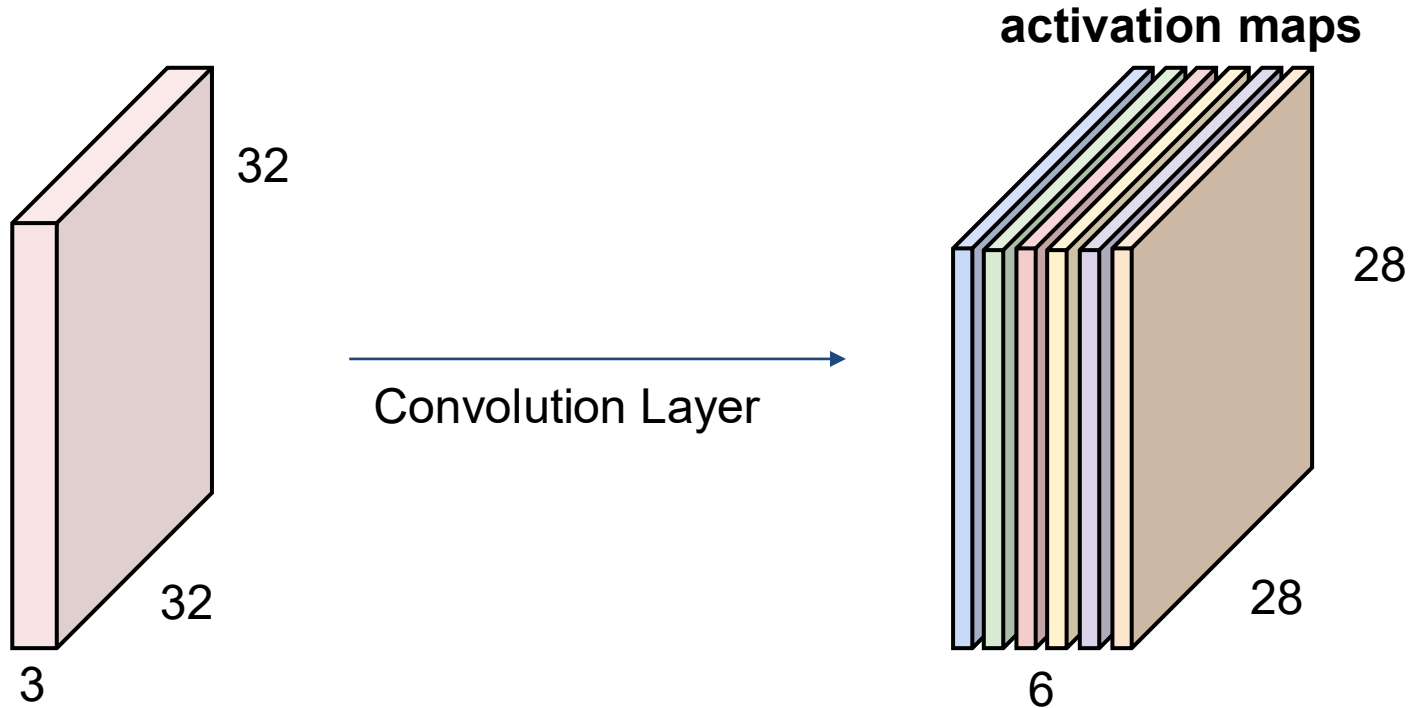


2-D Convolution Layer

consider a second, **green** filter

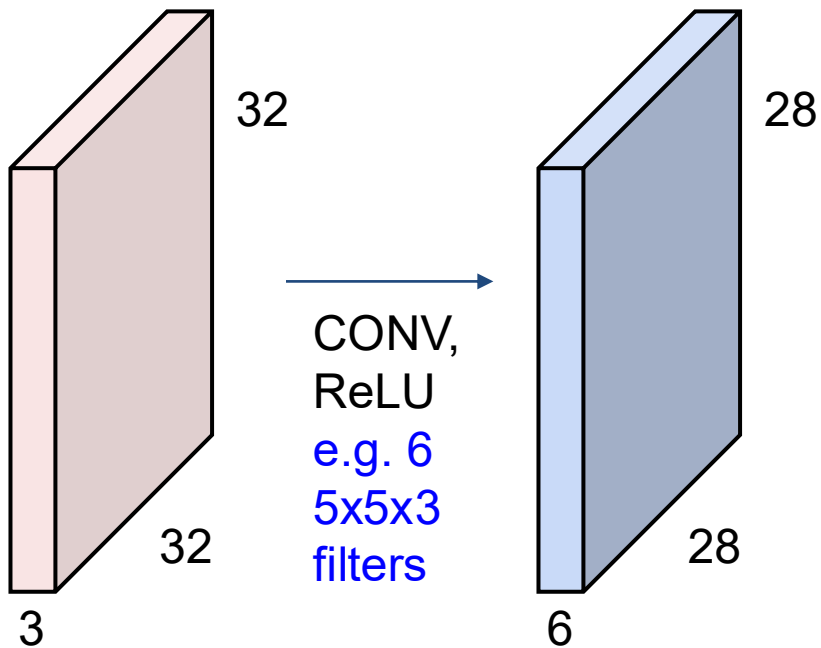


For example, if we had **6** 5x5 filters, we'll get 6 separate activation maps:

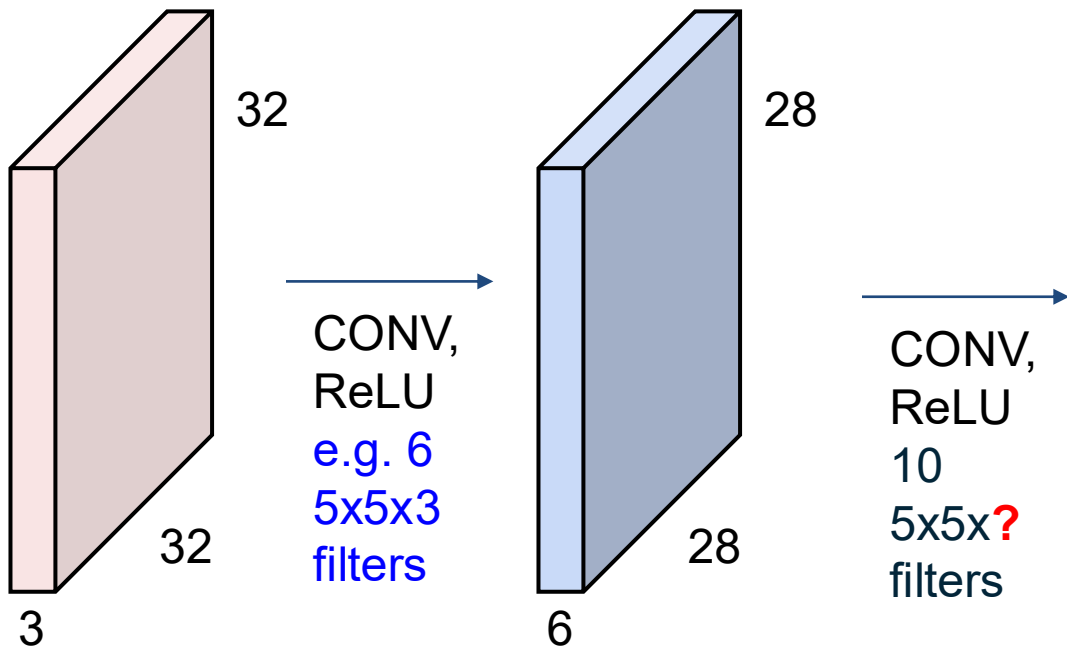


We stack these up to get a “new image” of size 28x28x6!

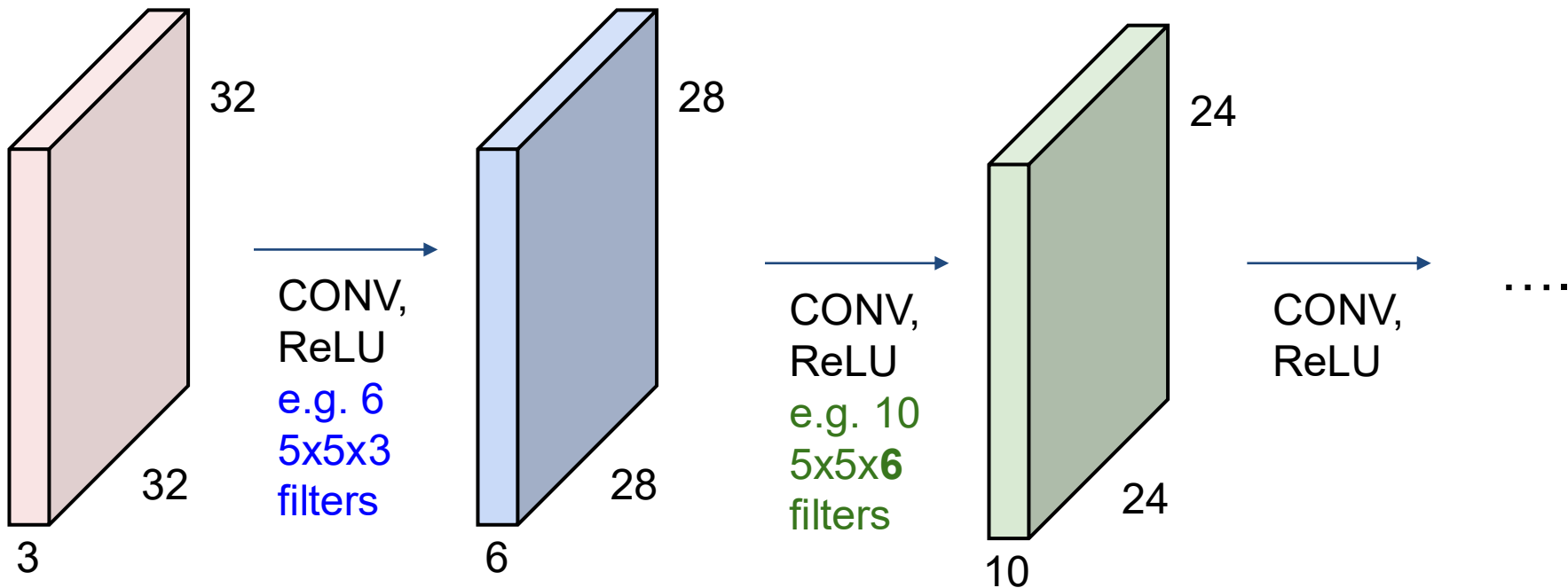
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



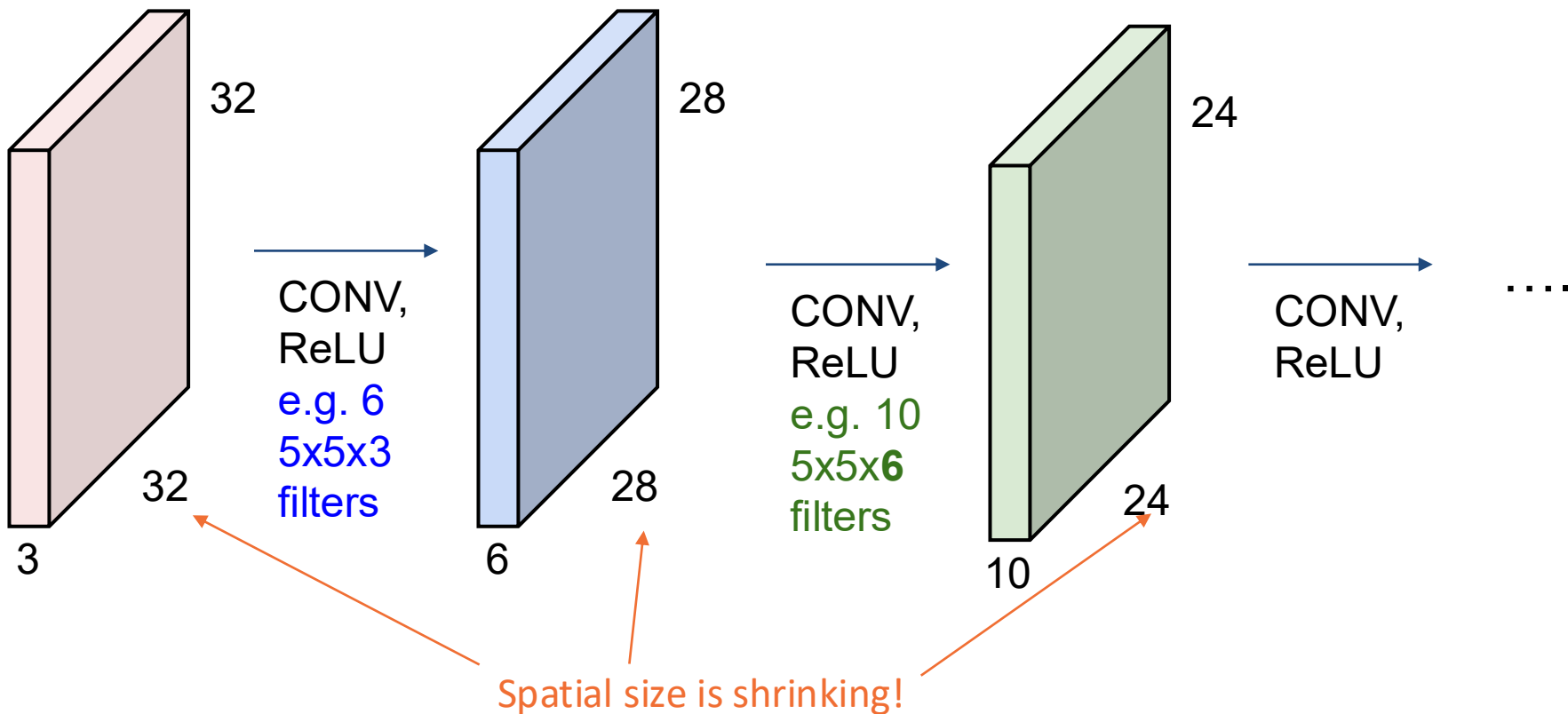
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



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



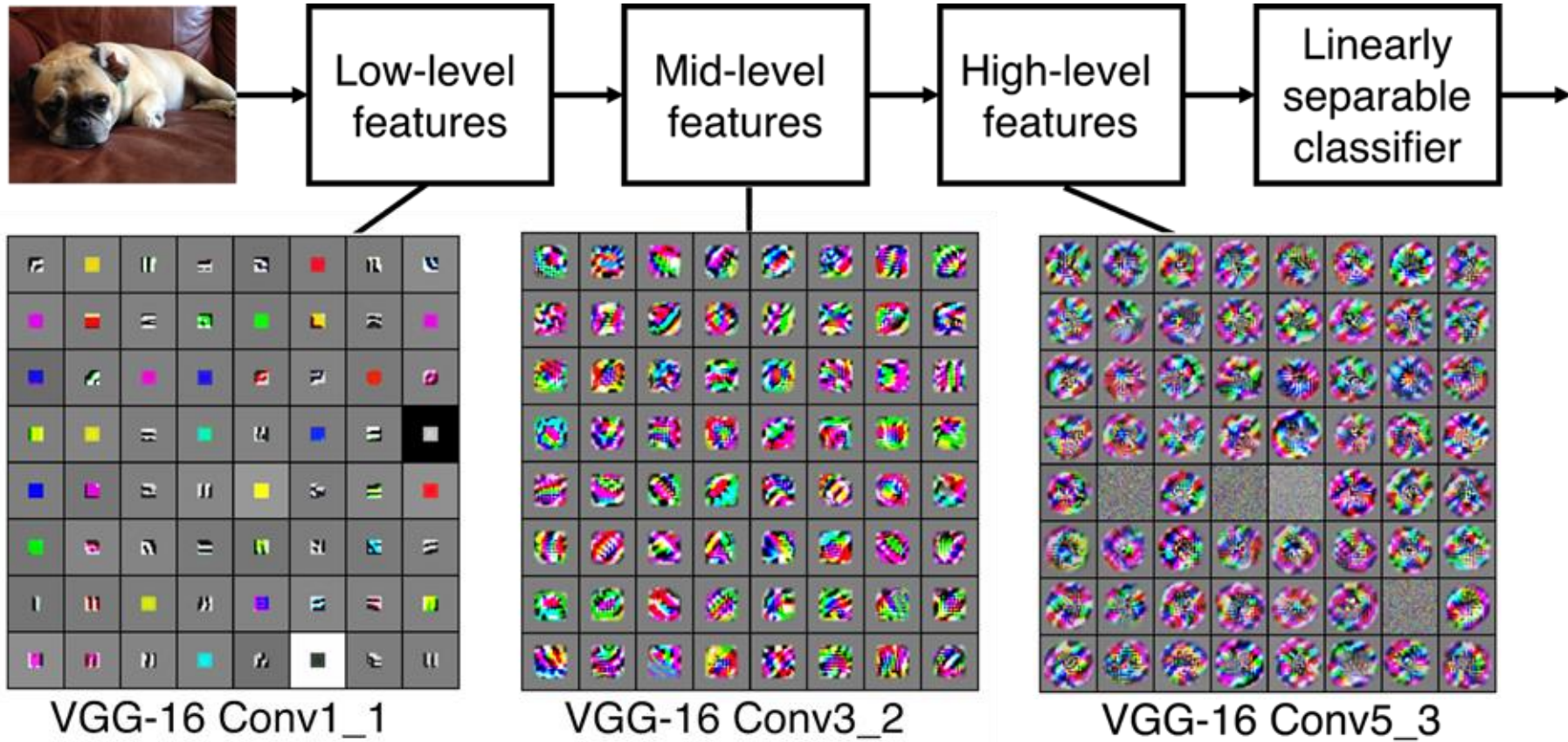
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



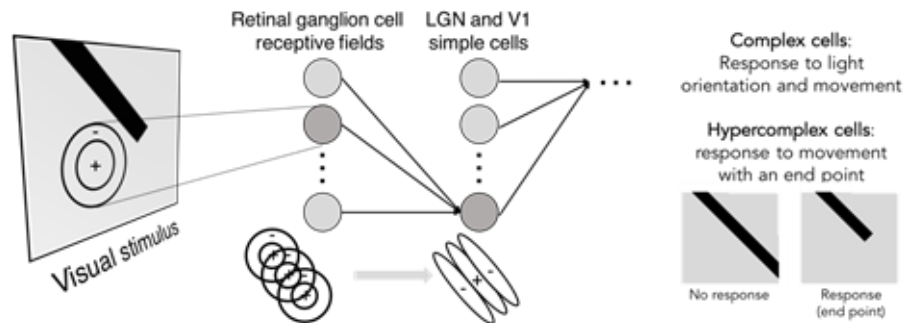
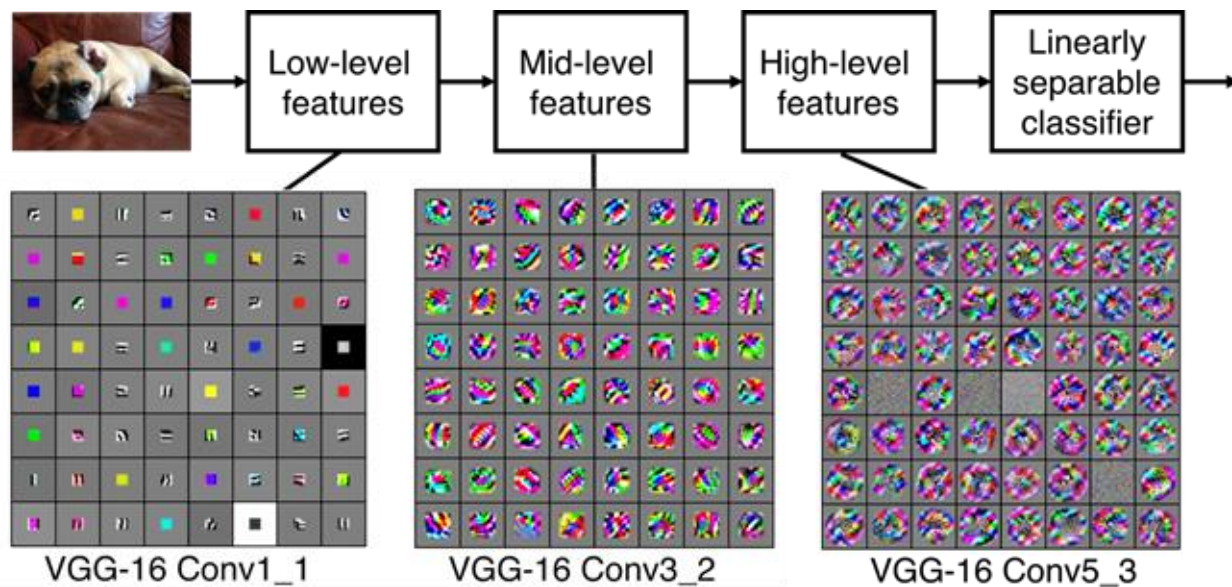
Preview

[Zeiler and Fergus 2013]

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



Preview





one filter =>
one activation map



Activations:

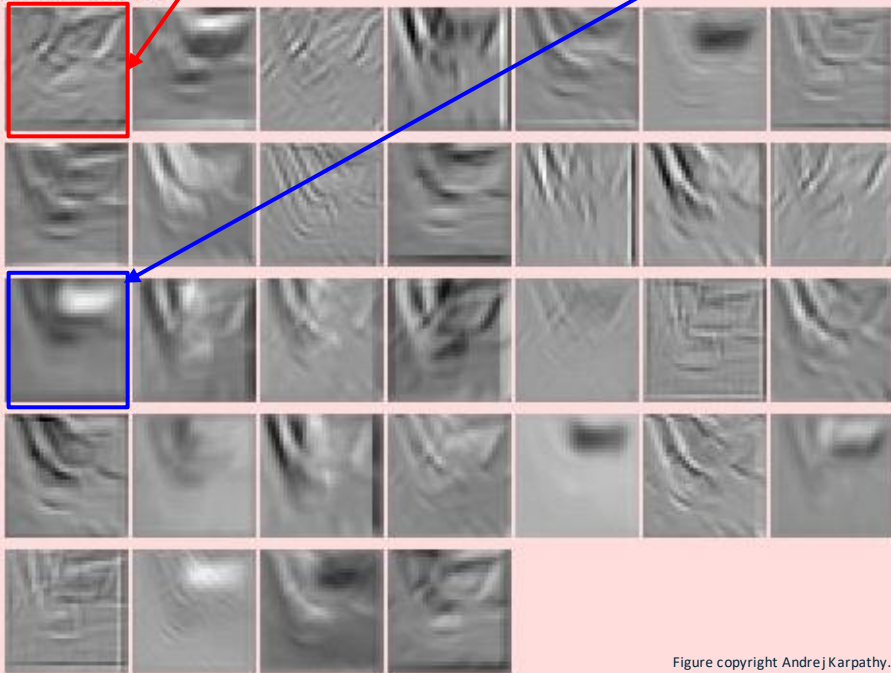
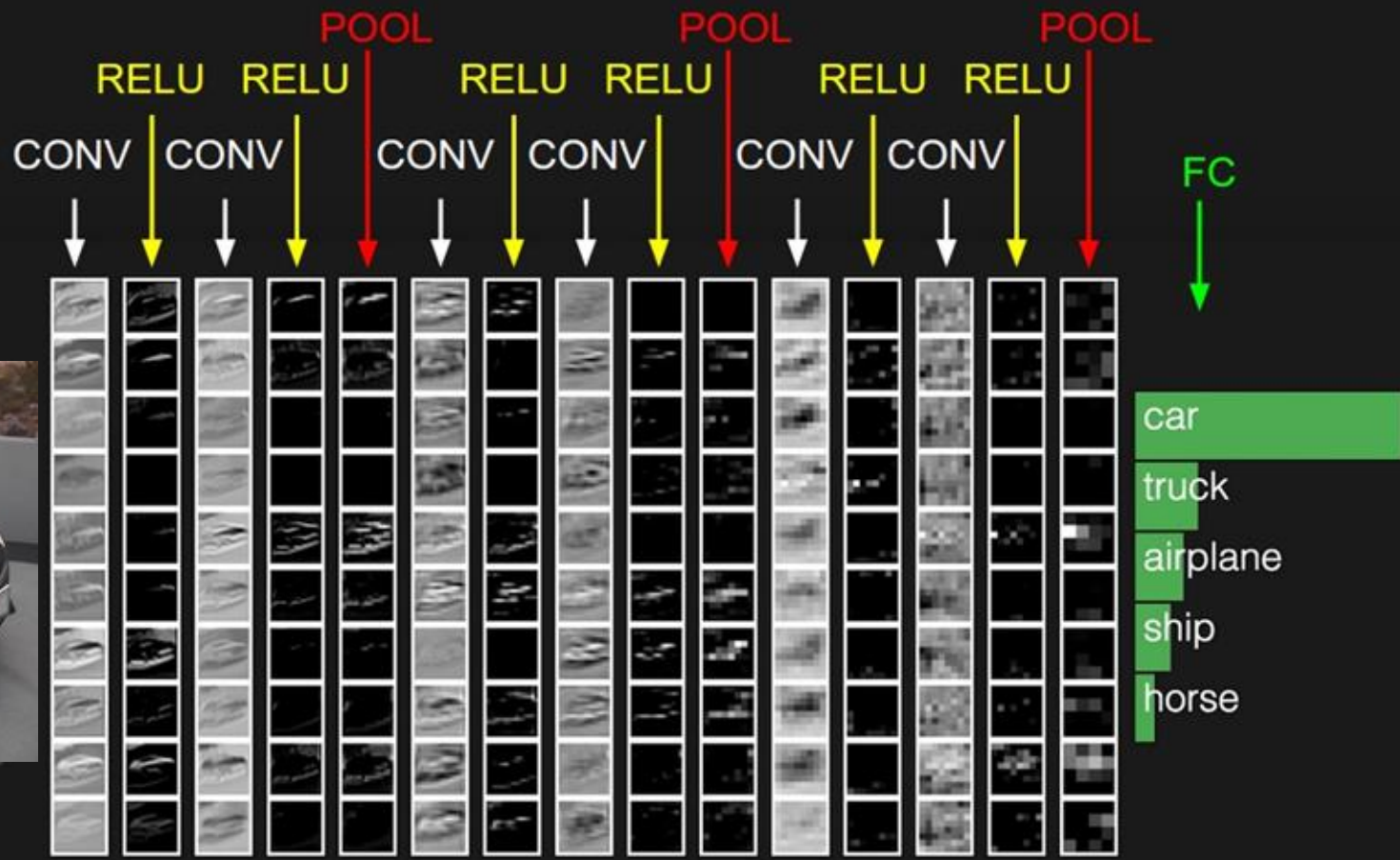


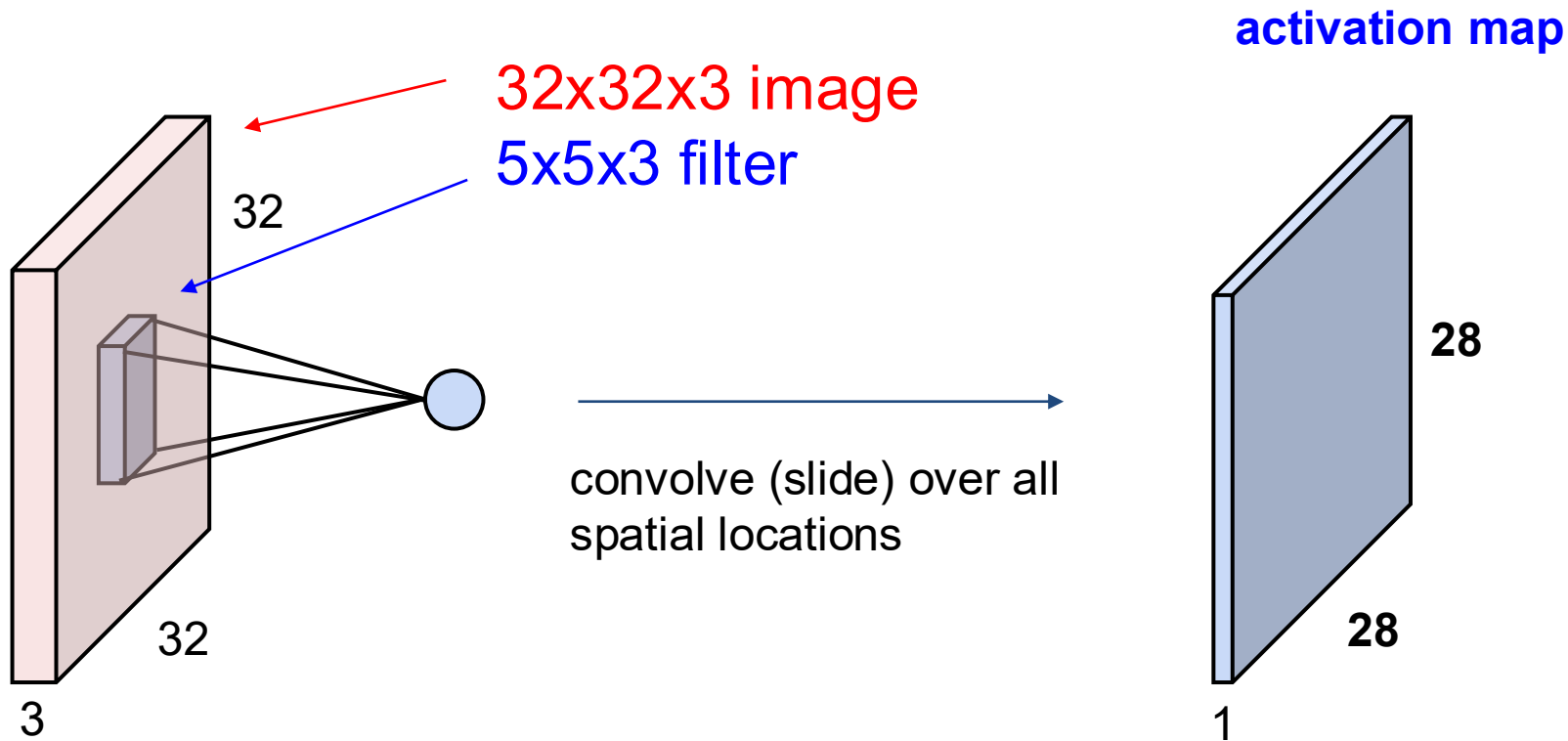
Figure copyright Andrej Karpathy.

example 5x5 filters
(32 total)

preview:



A closer look at spatial dimensions:



Conv2D in PyTorch

Conv2d

What are these?



```
CLASS torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1,  
                    groups=1, bias=True, padding_mode= 'zeros ', device=None, dtype=None) \[SOURCE\]
```

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

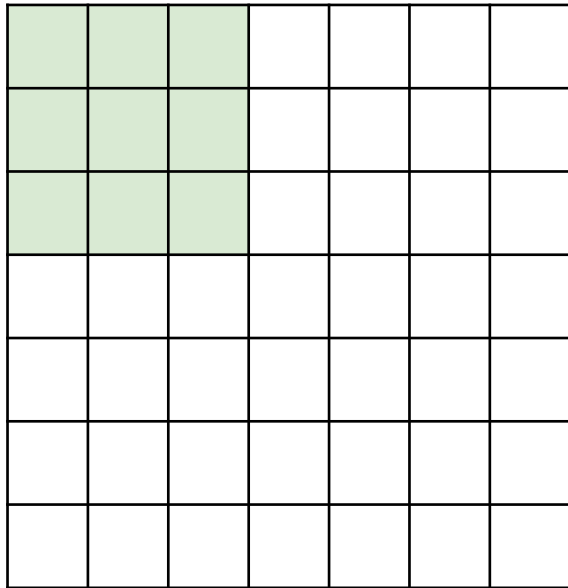
In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

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

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.

A closer look at spatial dimensions:

7

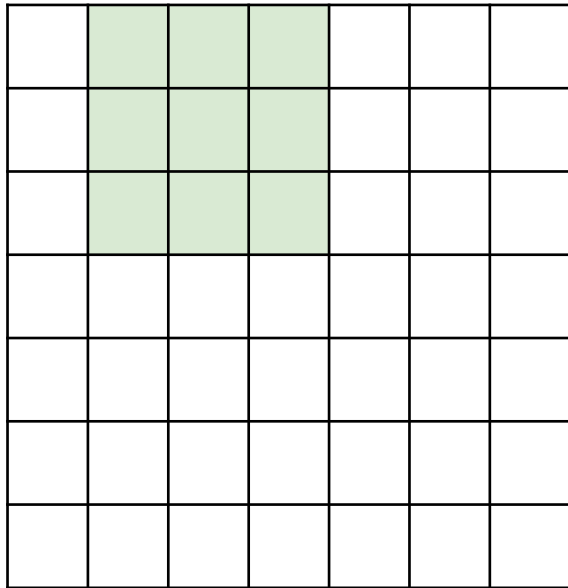


7

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

7

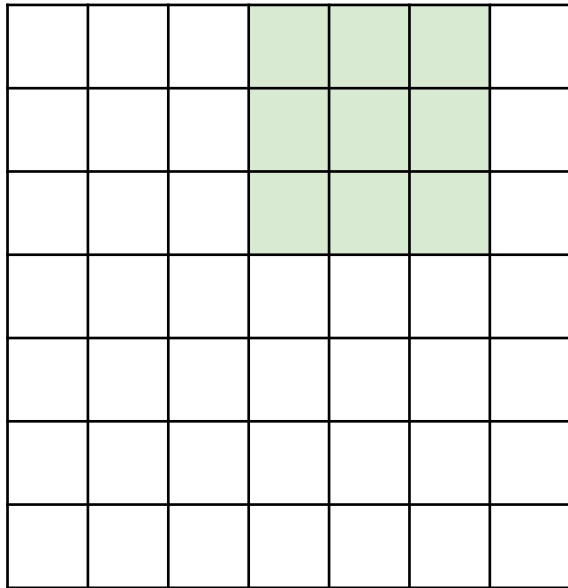
7x7 input (spatially)
assume 3x3 filter

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

E.g., here we have stride = 1

A closer look at spatial dimensions:

7



7

7x7 input (spatially)

assume 3x3 filter **with stride = 1**

A closer look at spatial dimensions:

7

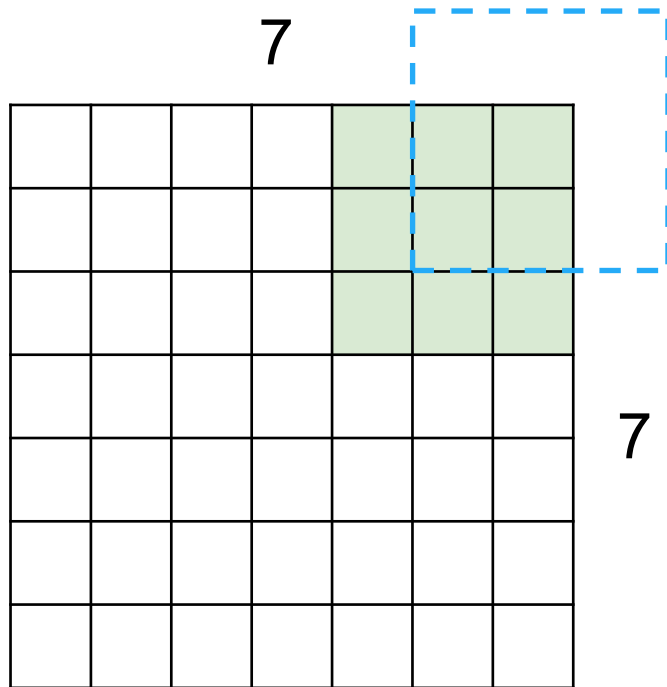
7

7x7 input (spatially)

assume 3x3 filter with stride = 1

=> 5x5 output

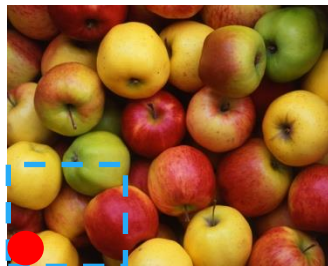
A closer look at spatial dimensions:



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

=> 5x5 output

But what about the features at the border?



Ideally the filter should be
centered wrt the input signal!

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?

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

7x7 output!

in general, common to see CONV layers with stride 1, filters of size $F \times F$, and zero-padding with $(F-1)/2$. (will preserve size spatially)

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

$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

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!

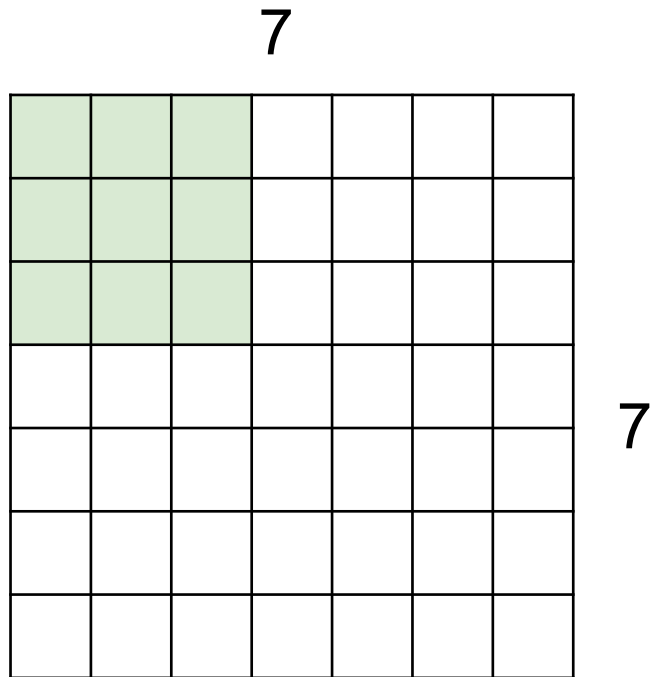
N = input dimension

P = padding size

F = filter size

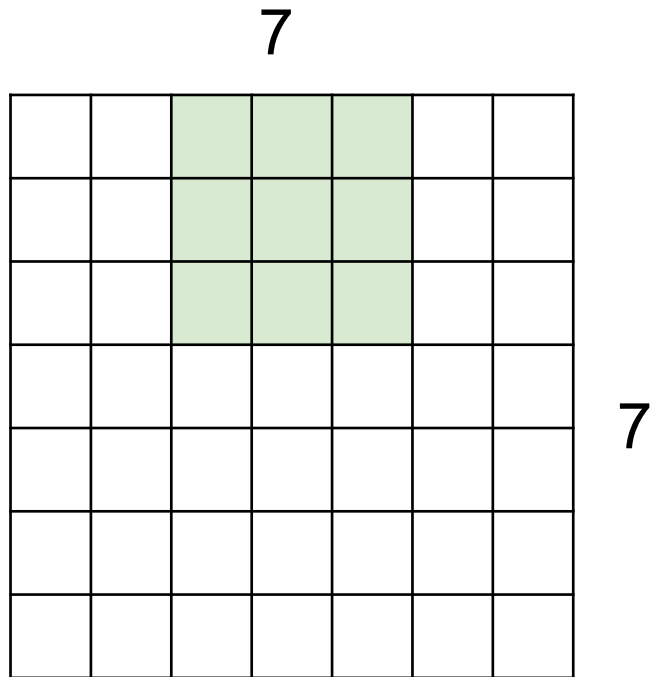
$$\begin{aligned}\text{Output size} &= (N - F + 2P) / \text{stride} + 1 \\ &= (7 - 3 + 2 * 1) / 1 + 1 = 7\end{aligned}$$

A closer look at spatial dimensions:



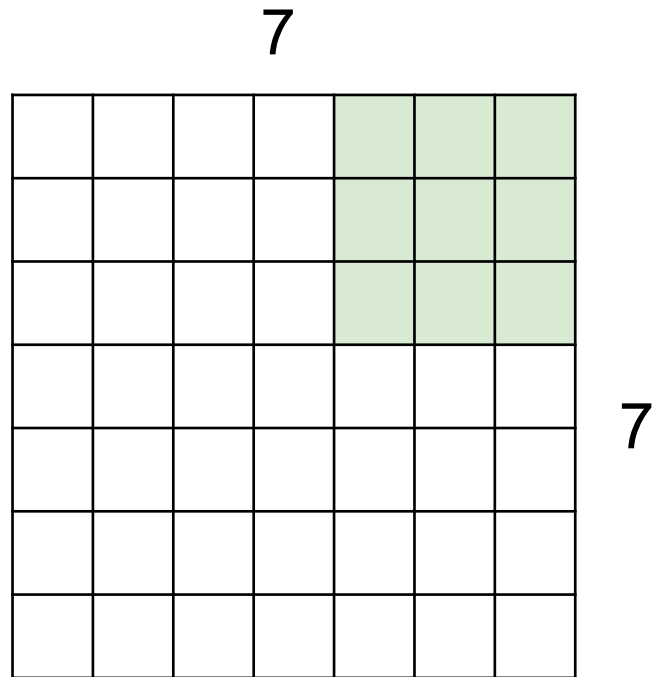
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



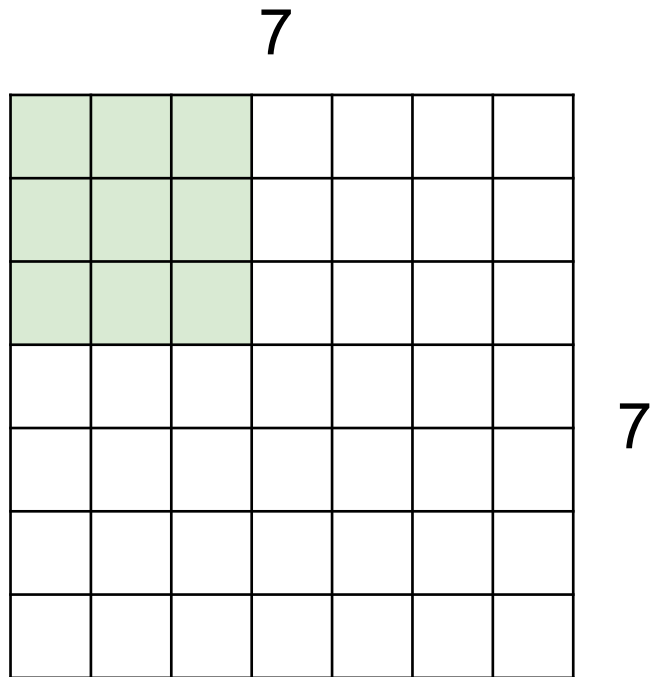
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:

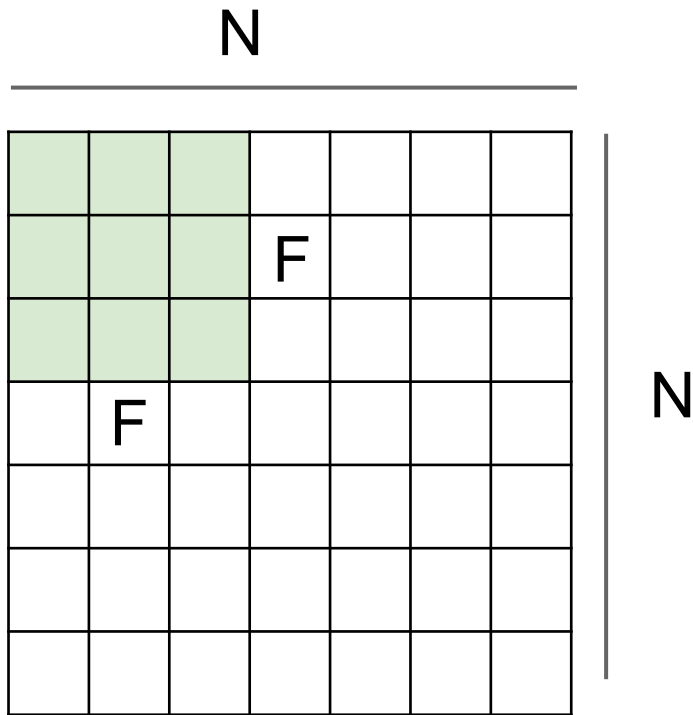


7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



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



Output size:
 $(N - F) / \text{stride} + 1$

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

stride 1 $\Rightarrow (7 - 3) / 1 + 1 = 5$

stride 2 $\Rightarrow (7 - 3) / 2 + 1 = 3$

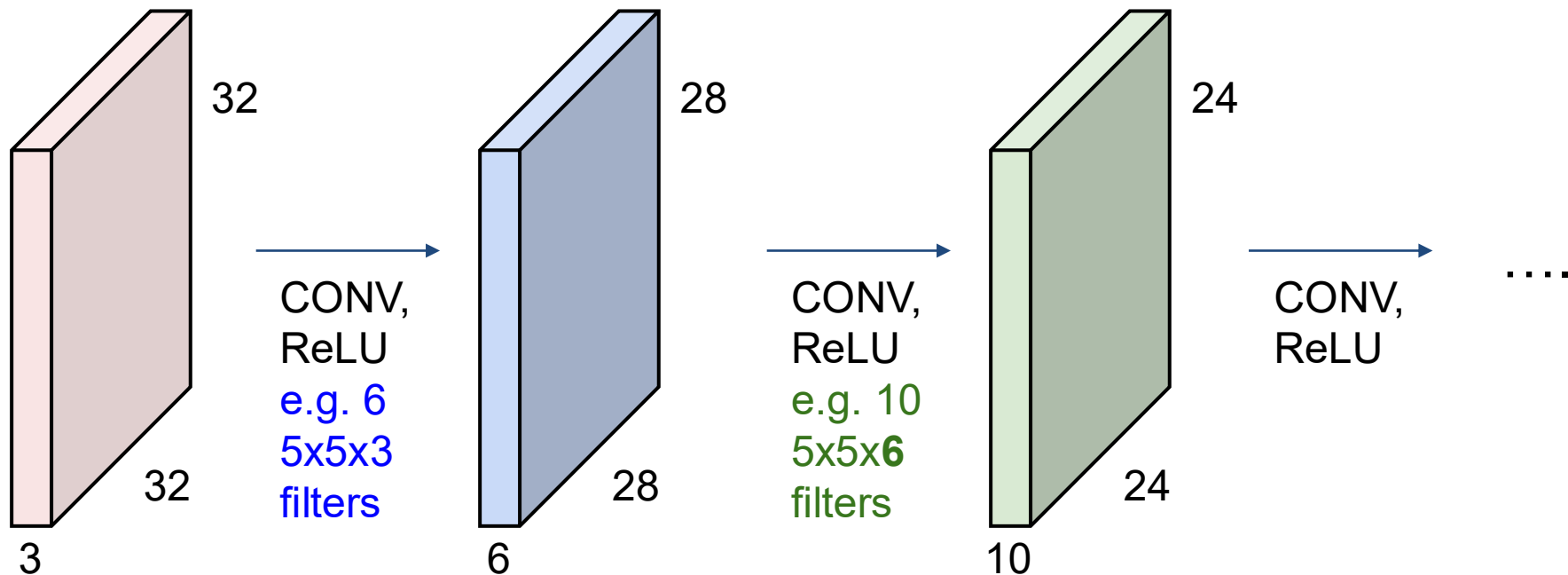
stride 3 $\Rightarrow (7 - 3) / 3 + 1 = 2.33 \therefore \backslash$

We use floor division to
 calculate output size:

$$(7 - 3) // 3 + 1 = 2$$

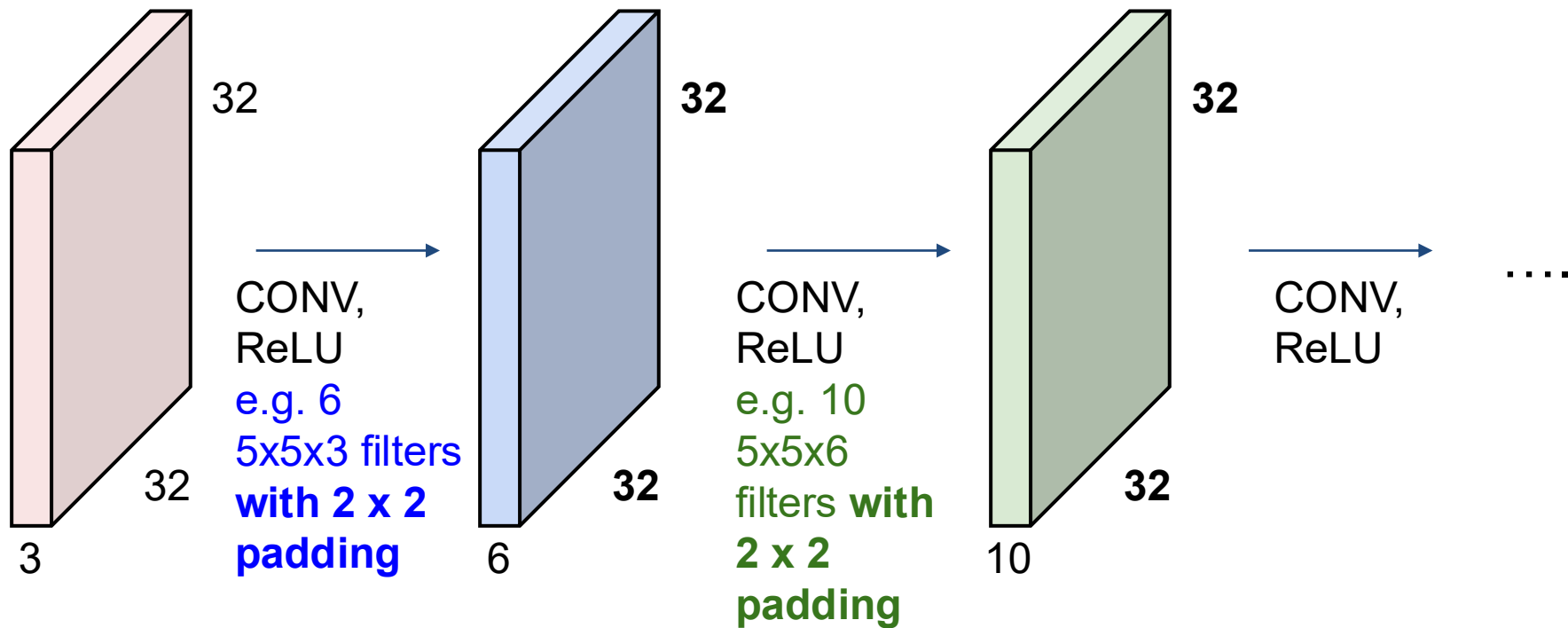
Remember back to...

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.



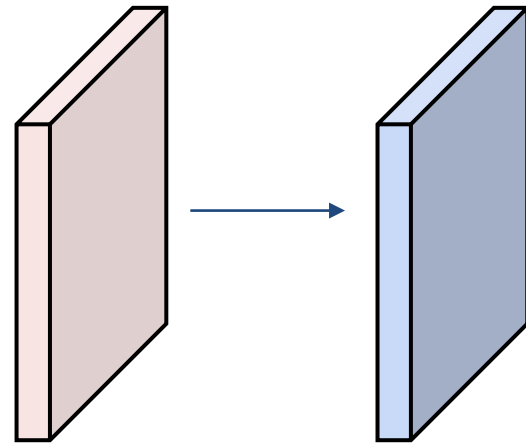
Examples time:

Input volume: **32x32x3**

Conv layer: 10 5x5 filters with
stride 1, pad 2

Output volume size: ?

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



Examples time:

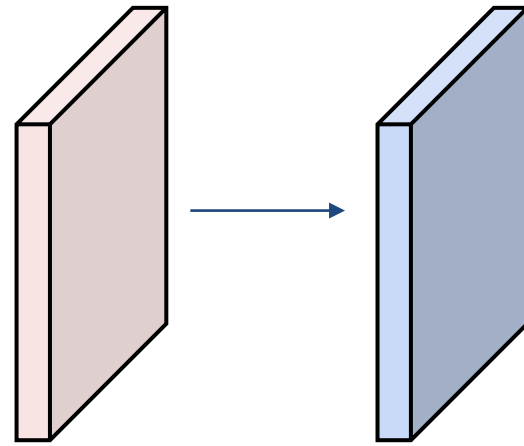
Input volume: **32x32x3**

Conv layer: **10** **5x5** filters with
stride **1**, pad **2**

Output volume size:

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

32x32x10

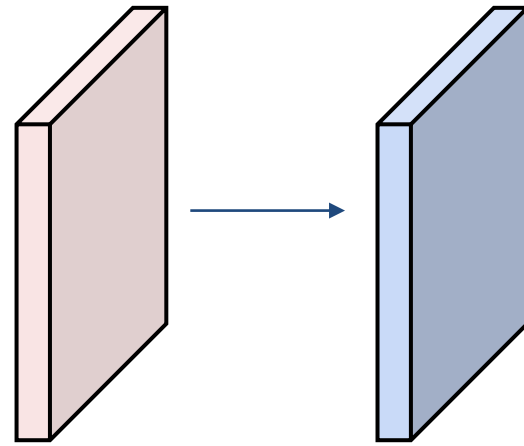


Examples time:

Input volume: **32x32x3**

Conv layer: 10 5x5 filters with
stride 1, pad 2

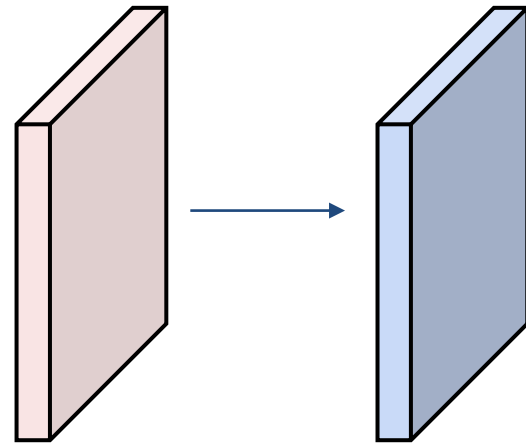
Number of parameters in this
layer?



Examples time:

Input volume: **32x32x3**

Conv layer: **10** **5x5** filters with
stride 1, pad 2



Number of parameters in this layer?

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

=> $76*10 = 760$

Convolution layer: summary

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

Conv layer needs 4 hyperparameters:

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

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

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

Number of parameters: F^2CK and K biases

Convolution layer: summary

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

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

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

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: $\begin{bmatrix} C_{out} \\ C_{in} \end{bmatrix}$.

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

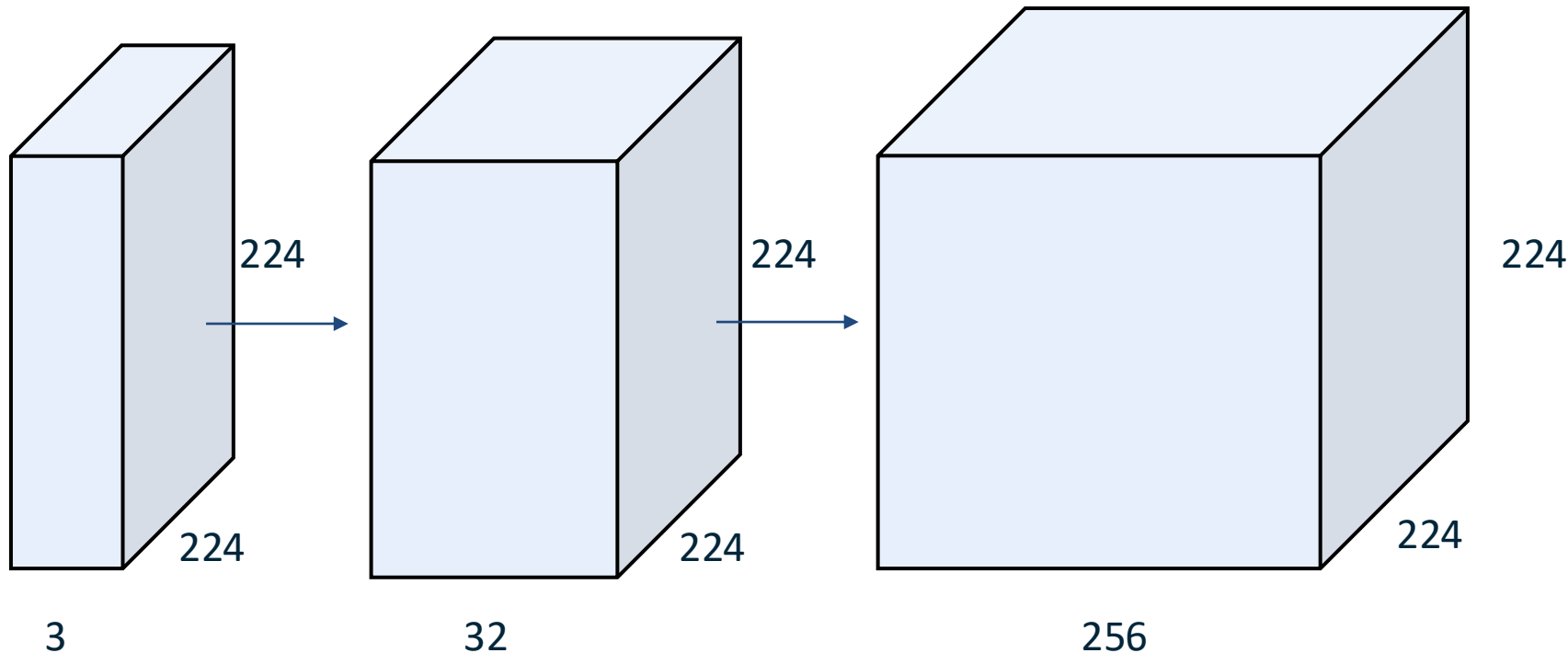
- a single `int` – in which case the same value is used for the height and width dimension
- a `tuple` of two `ints` – in which case, the first `int` is used for the height dimension, and the second `int` for the width dimension

Conv layer needs 4 hyperparameters:

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

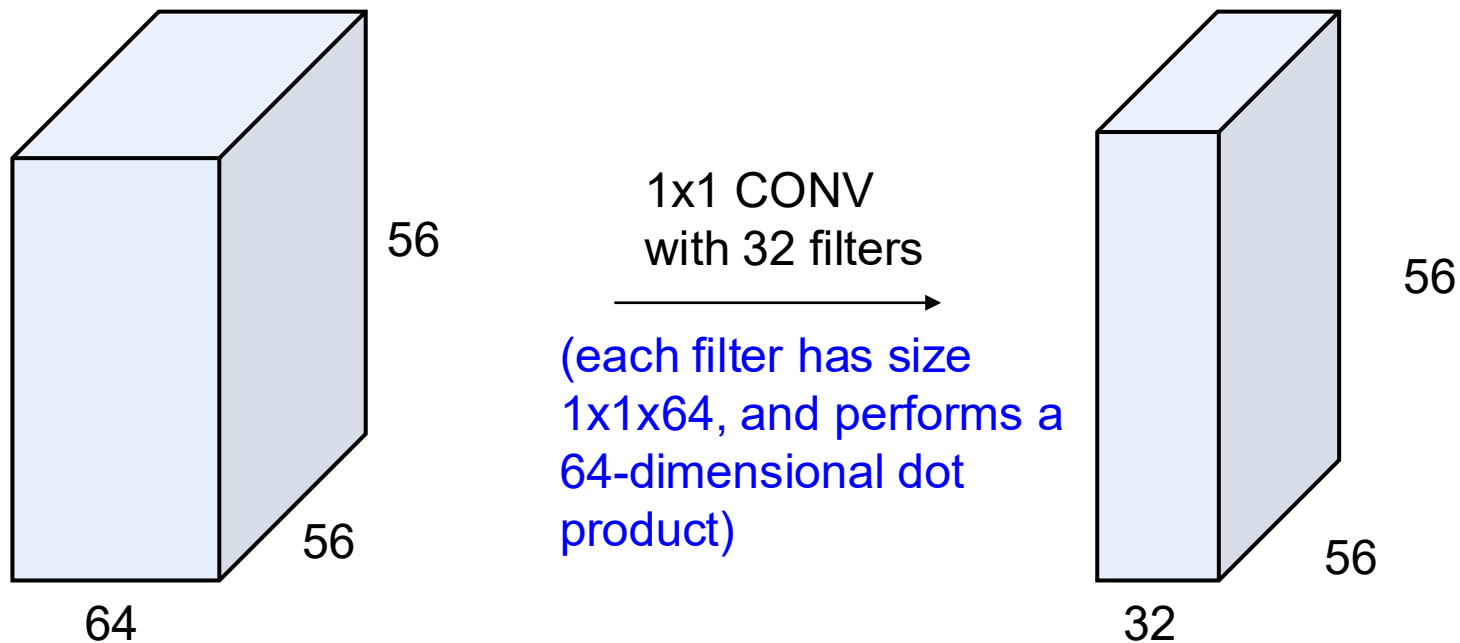
[PyTorch](#) is licensed under [BSD 3-clause](#).

Conv features can grow really big really quickly...

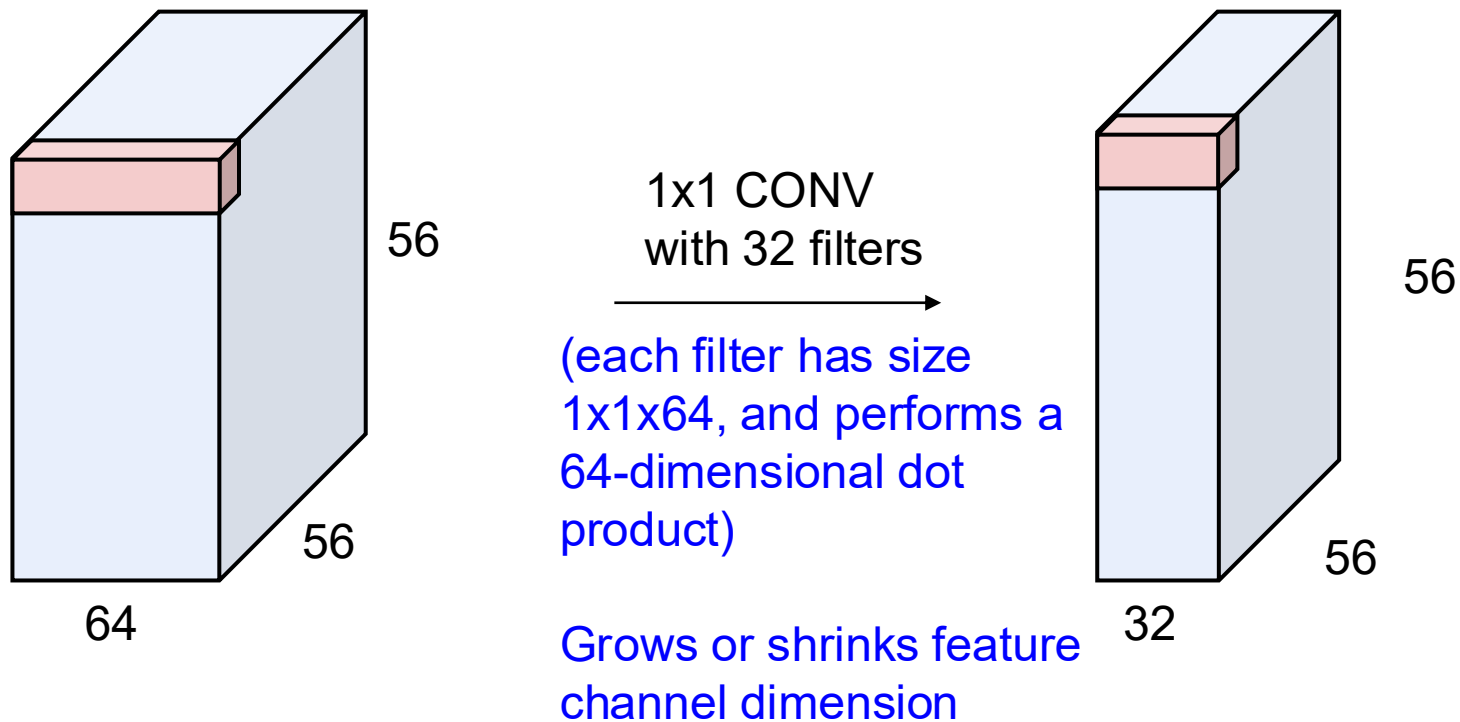


~12million fp!

Solution 1: 1x1 Convolution

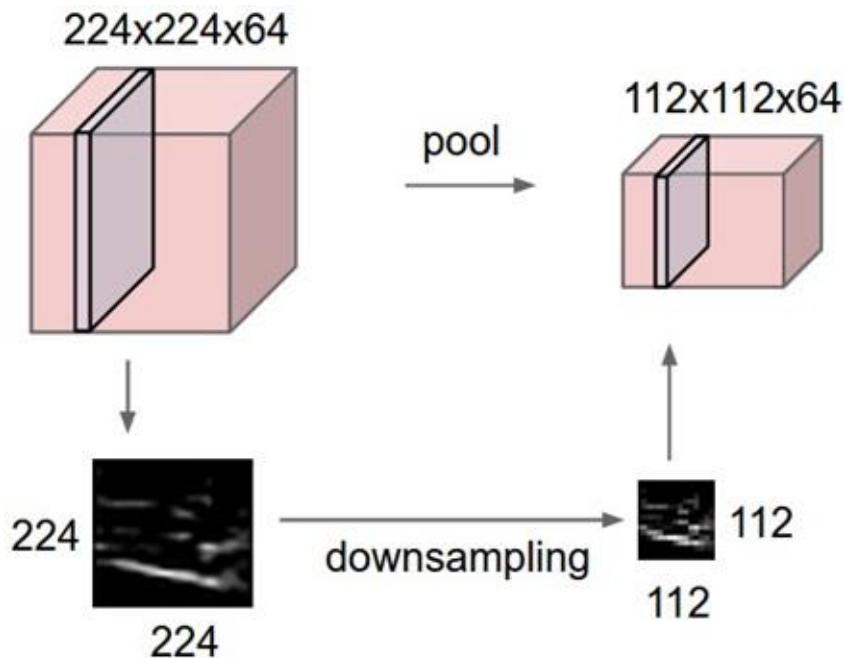


Solution 1: 1x1 Convolution

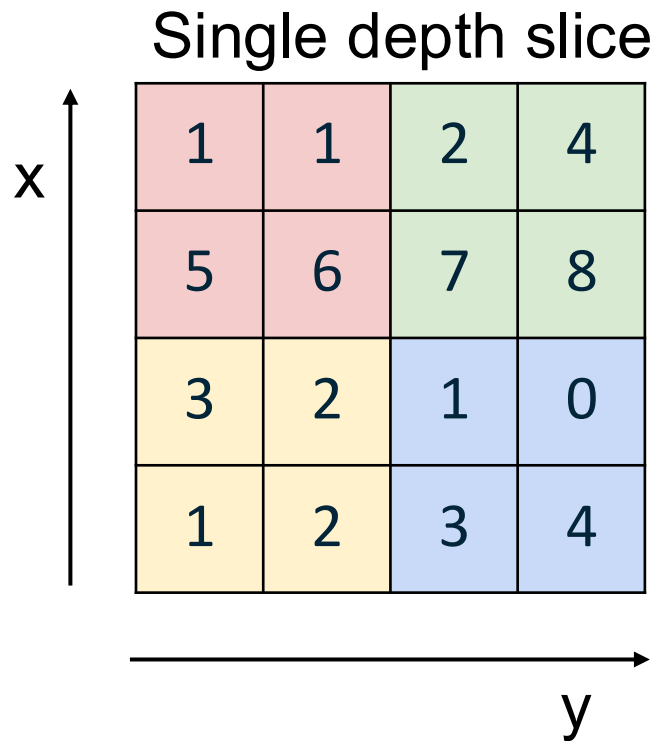


Solution 2: Pooling (downsampling)

- makes the representations **spatially smaller**
- saves computation (GPU mem & speed), allows go deeper
- operates over each activation map independently:



MAX POOLING



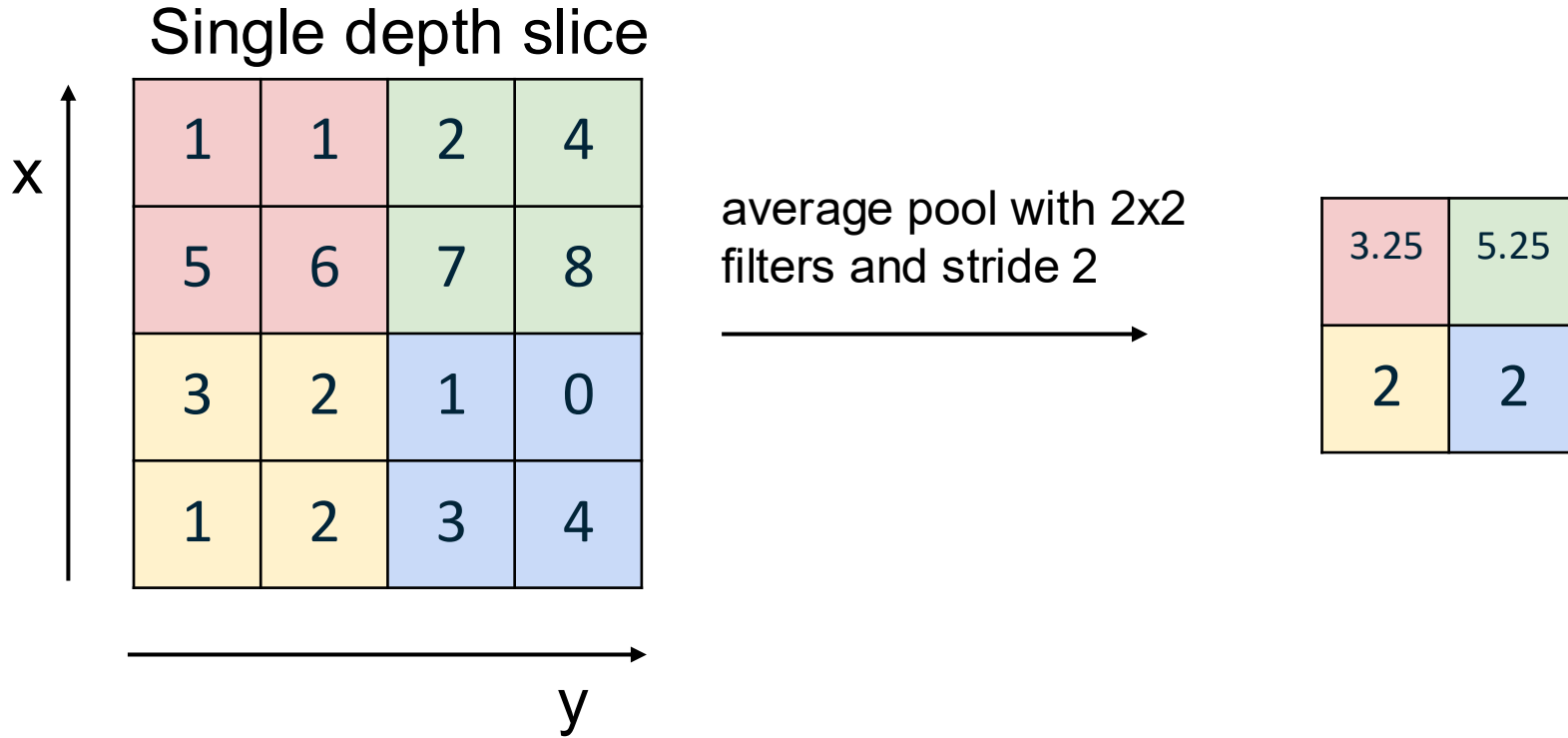
max pool with 2x2 filters
and stride 2



6	8
3	4

- Intuitively, only forward the most important features in the region.
- Also improve spatial invariance (output is agnostic to where the max value comes from)

AVG POOLING



Pooling layer: summary

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

Pooling layer needs 2 hyperparameters:

- The spatial extent **F** (e.g., 2)
- The stride **S** (e.g., 2)

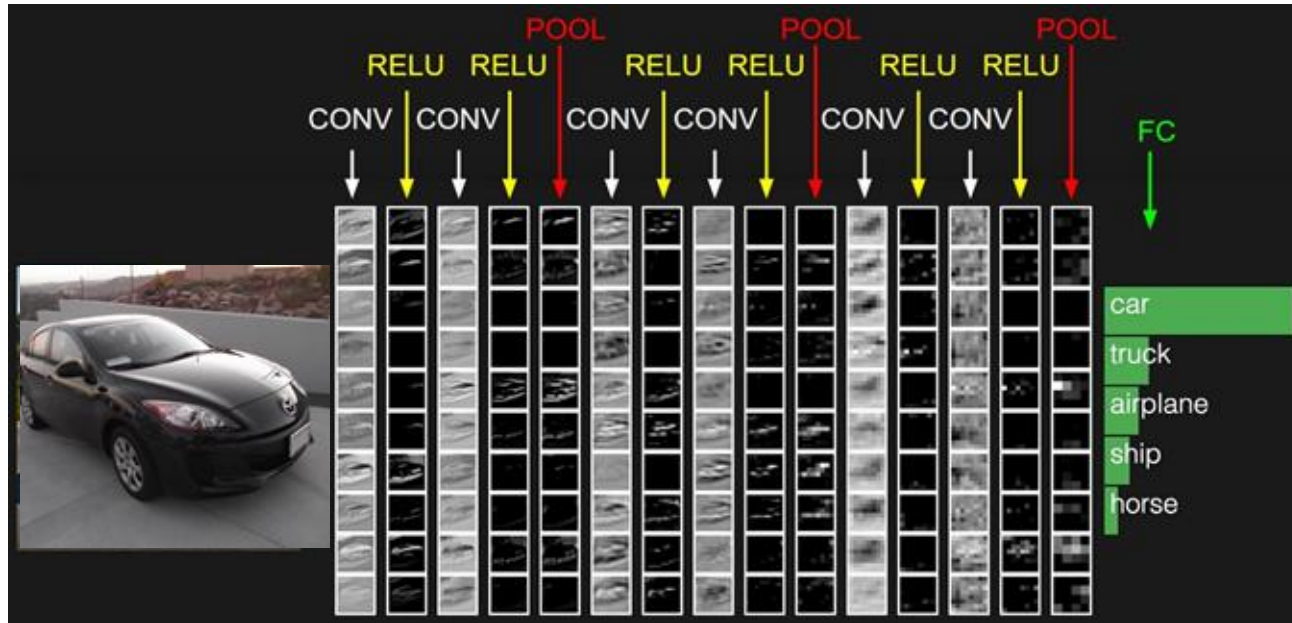
This will produce an output of $W_2 \times H_2 \times C$ where:

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

Number of parameters?

0

A canonical (shallow) convolutional neural net



Next Time: How to Pick a Project
(and how to do research)