max planck institut informatik

SIC Saarland Informatics Campus

High Level Computer Vision

Backpropagation & Convolutional Neural Networks @ May 8, 2024

Bernt Schiele

https://cms.sic.saarland/hlcvss24/

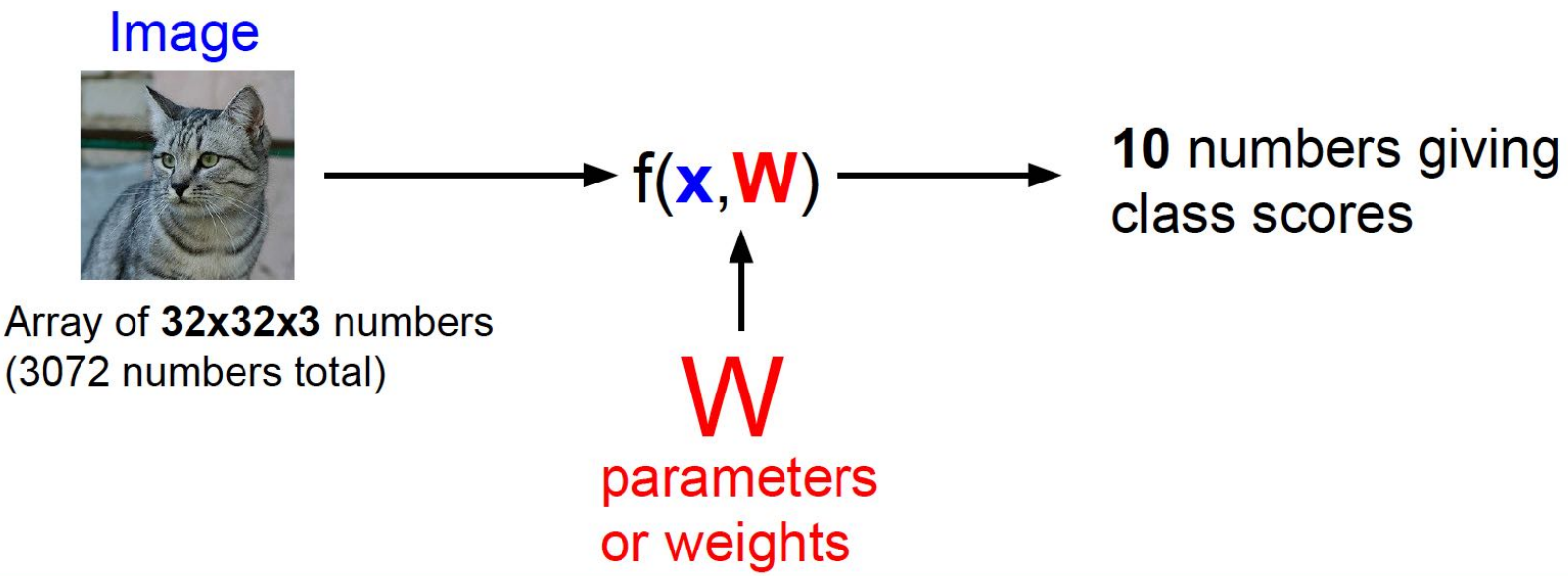
Max Planck Institute for Informatics & Saarland University,
Saarland Informatics Campus Saarbrücken

Overview Today's Lecture

- Backpropagation — Gradient Descent
  - illustrated using computational graphs
  - chain rule - upstream and local gradients
  - simple modularization

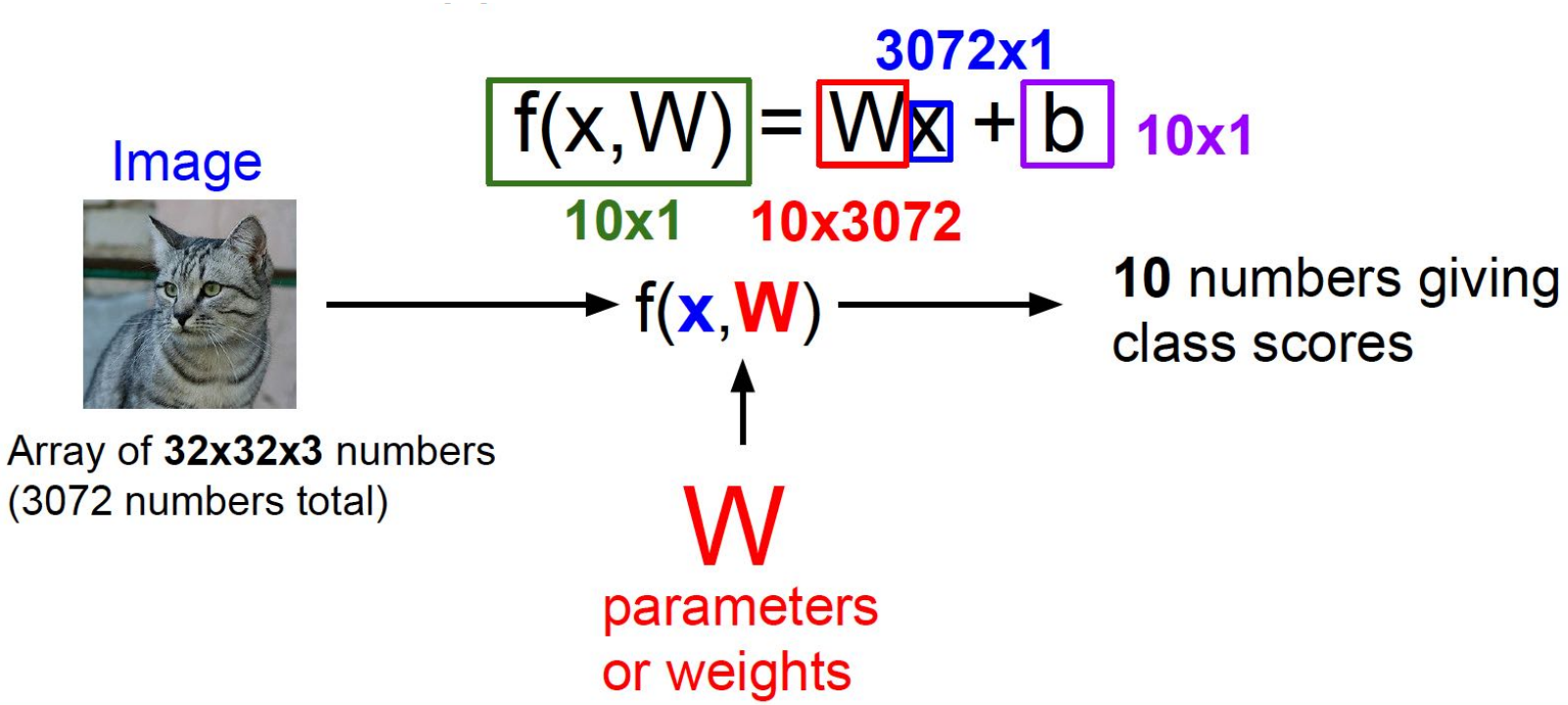
- Convolutional Neural Networks (CNNs)
  - versatile architecture & motivation
    (one of the highly successful neural networks…)
  - convolution layer + activation function + pooling + fully connected layer
  - depth matters

Where we are... : Parametric Approach



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Parametric Approach: Linear Classifier

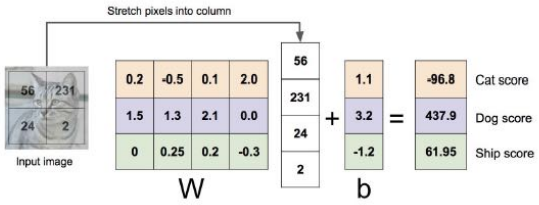


slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Linear Classifier: Three Viewpoints

Algebraic Viewpoint

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

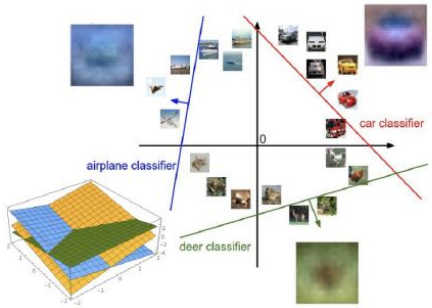


Visual Viewpoint

One template per class

Geometric Viewpoint

Hyperplanes cutting up space



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Where we are... : Parametric Approach

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

Softmax

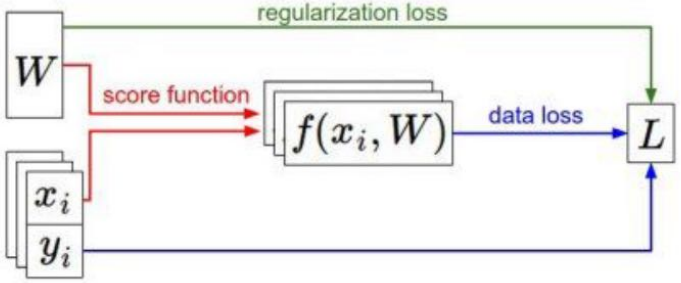
$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

SVM

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

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

Full loss = data loss + regularization



How do we find the best W?

Gradient Descent Optimization: want $\nabla_W L$

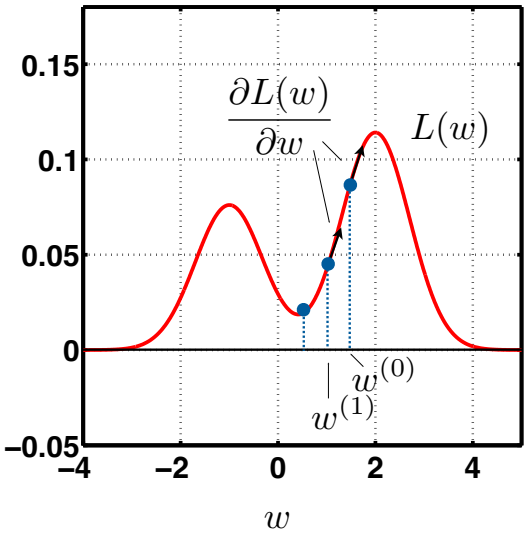
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Strategies

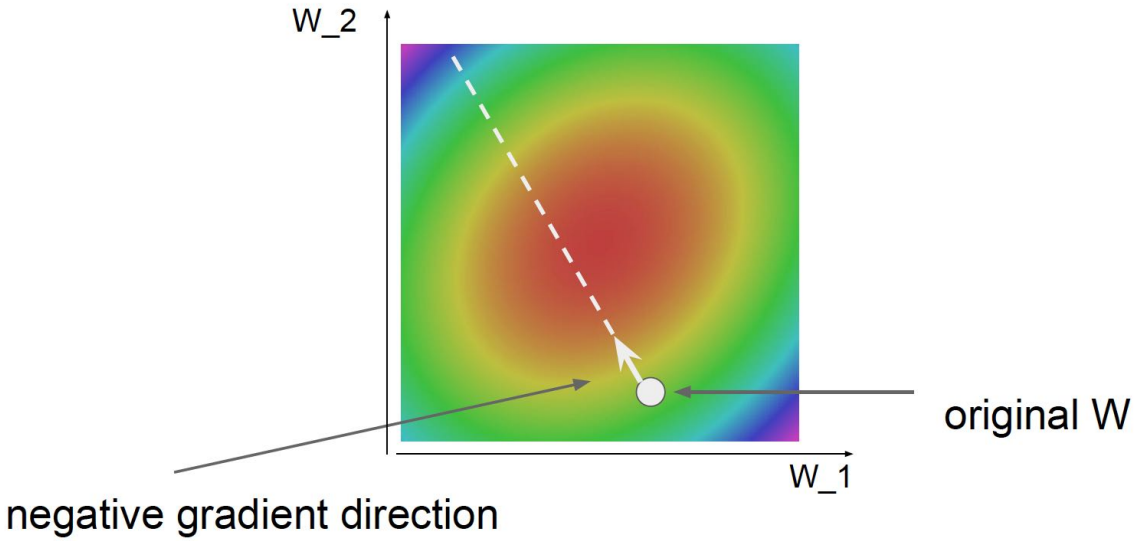
- #1 Random Search
  - can work…
  - but very expensive
- #2 “Follow the Slope”
  - aka: gradient descent…

Gradient Descent (in 1 dimension)

- Iteratively minimize the loss function: $L(w)$
  - Initialize somewhere:
    $$w^{(0)}$$
  - Compute the derivative:
    $$\frac{\partial L(w)}{\partial w} = L'(w)$$
  - Take a step in the *negative* direction of the derivative:
    $$w^{(1)} \leftarrow w^{(0)} - \alpha \cdot \frac{\partial L(w^{(0)})}{\partial w^{(0)}}$$
    step size
  - Repeat...
    $$w^{(t+1)} \leftarrow w^{(t)} - \alpha \cdot \frac{\partial L(w^{(t)})}{\partial w^{(t)}}$$



Gradient Descent (in two dimensions)



Gradient Descent (in multiple dimensions)

- in one dimension:

$$w^{(t+1)} \leftarrow w^{(t)} - \alpha \cdot \frac{\partial L(w^{(t)})}{\partial w^{(t)}}$$

- in n dimensions:
  - the loss is a function of the n-dimensional weight vector:

$$L(W^{(t)}) \quad \text{with} \quad W^{(t)} = \begin{pmatrix} w_1^{(t)} \\ \vdots \\ w_n^{(t)} \end{pmatrix}$$

  - we can define partial derivatives for each dimension of the weight vector:

$$\frac{\partial L(W^{(t)})}{\partial w_i^{(t)}}$$

Gradient Descent (in multiple dimensions)

- in one dimension:

$$w^{(t+1)} \leftarrow w^{(t)} - \alpha \cdot \frac{\partial L(w^{(t)})}{\partial w^{(t)}}$$

- in n dimensions:
  - apply the **same gradient descent rule** in **each dimension separately**:

$$w_1^{(t+1)} \leftarrow w_1^{(t)} - \alpha \cdot \frac{\partial L(W^{(t)})}{\partial w_1^{(t)}}$$

$$w_2^{(t+1)} \leftarrow w_2^{(t)} - \alpha \cdot \frac{\partial L(W^{(t)})}{\partial w_2^{(t)}}$$

$$\vdots$$

  - in vector form:

$$\begin{pmatrix} w_1^{(t+1)} \\ \vdots \\ w_n^{(t+1)} \end{pmatrix} \leftarrow \begin{pmatrix} w_1^{(t)} \\ \vdots \\ w_n^{(t)} \end{pmatrix} - \alpha \cdot \begin{pmatrix} \frac{\partial L(W^{(t)})}{\partial w_1^{(t)}} \\ \vdots \\ \frac{\partial L(W^{(t)})}{\partial w_n^{(t)}} \end{pmatrix}$$

Gradient Descent (in multiple dimensions)

- notation
  - in vector form: (previous slide)

$$\begin{pmatrix} w_1^{(t+1)} \\ \vdots \\ w_n^{(t+1)} \end{pmatrix} \leftarrow \begin{pmatrix} w_1^{(t)} \\ \vdots \\ w_n^{(t)} \end{pmatrix} - \alpha \cdot \begin{pmatrix} \frac{\partial L(W^{(t)})}{\partial w_1^{(t)}} \\ \vdots \\ \frac{\partial L(W^{(t)})}{\partial w_n^{(t)}} \end{pmatrix}$$

Vector with partial derivatives

  - compact notation:

$$W^{(t+1)} \leftarrow W^{(t)} - \alpha \cdot \nabla_{W^{(t)}} L(W^{(t)})$$

(called: Nabla operator)

Gradient Descent (“Code”)

- Mathematical notation:

$$W^{(t+1)} \leftarrow W^{(t)} - \alpha \cdot \nabla_{W^{(t)}} L(W^{(t)})$$

- translated into “code”

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

- gradient calculation expensive, when entire dataset is used (N typically large !)

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Gradient Descent - Variants…

- Assume Loss to be:
  - with N the number of training samples
  - L_i the loss for training sample x_i

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(W)$$

- **Batch training**:
  - process all training samples x_i
  - update weights based on **loss** for **all** training **samples**

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(W)$$

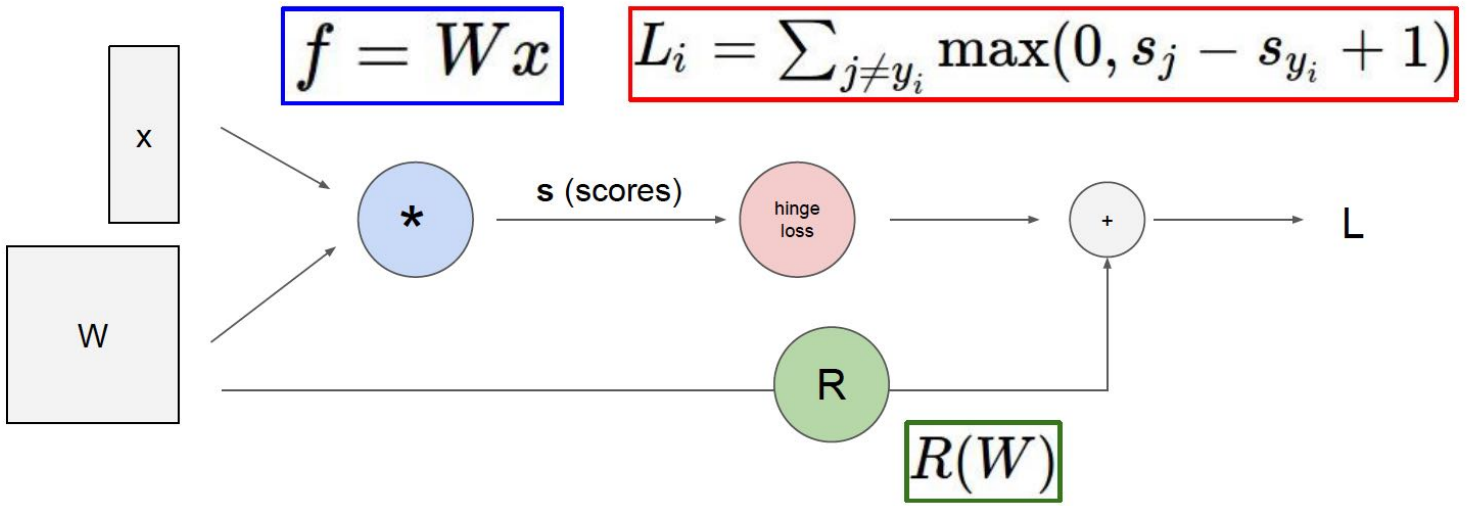
- **Stochastic Gradient Descent (SGD)**:
  - randomly choose **one** training sample x_i
  - update weights based on **loss** for **one** training sample:

$$L_i(W)$$

- **Mini-batch training**:
  - process a mini-batch = subset of training samples $M \subset \{1, \ldots, N\}$
  - update weights based on **loss** on **mini-batch**:

$$L_M(W) = \frac{1}{|M|} \sum_{i \in M} L_i(W)$$

Computational Graphs



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Neural Network Example…

Convolutional network
(AlexNet)

input image

weights

loss

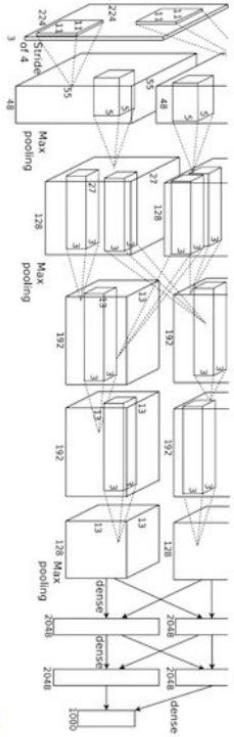


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Neural Network Example…

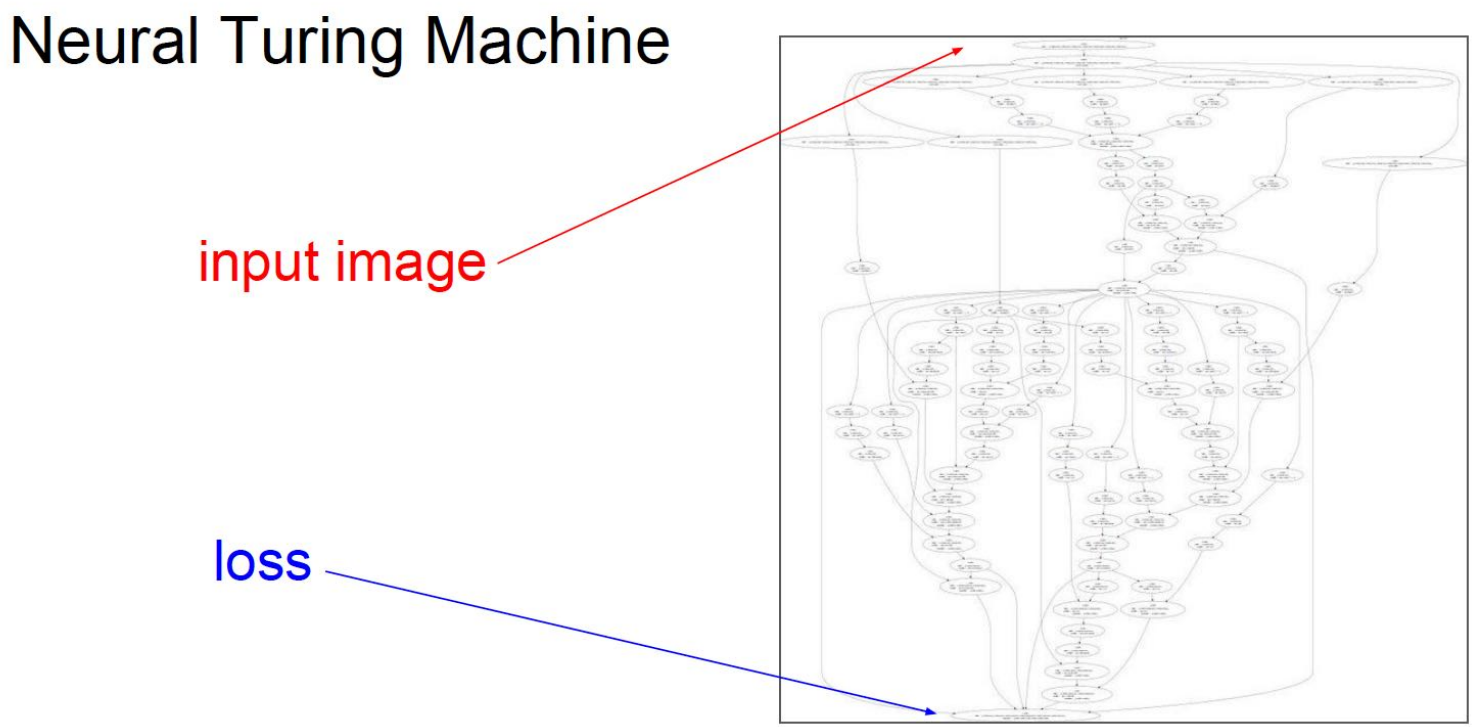


Figure reproduced with permission from a Twitter post by Andrej Karpathy.

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation - A Simple Example

Backpropagation: a simple example

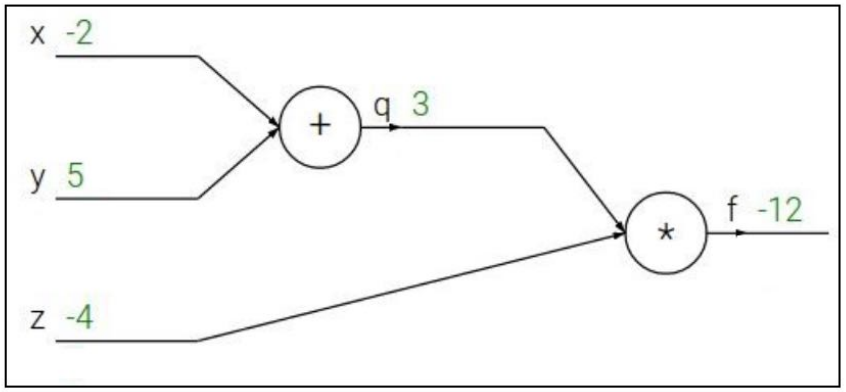
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation - A Simple Example

Backpropagation: a simple example

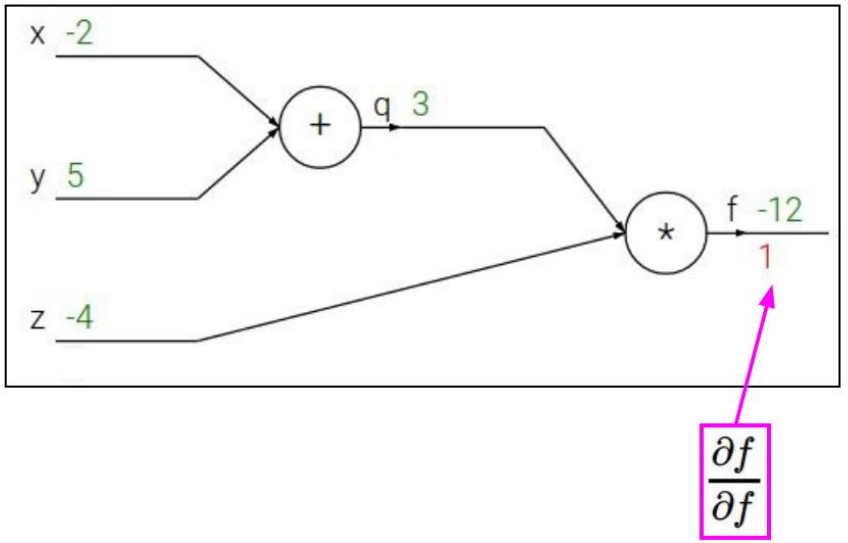
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation - A Simple Example

Backpropagation: a simple example

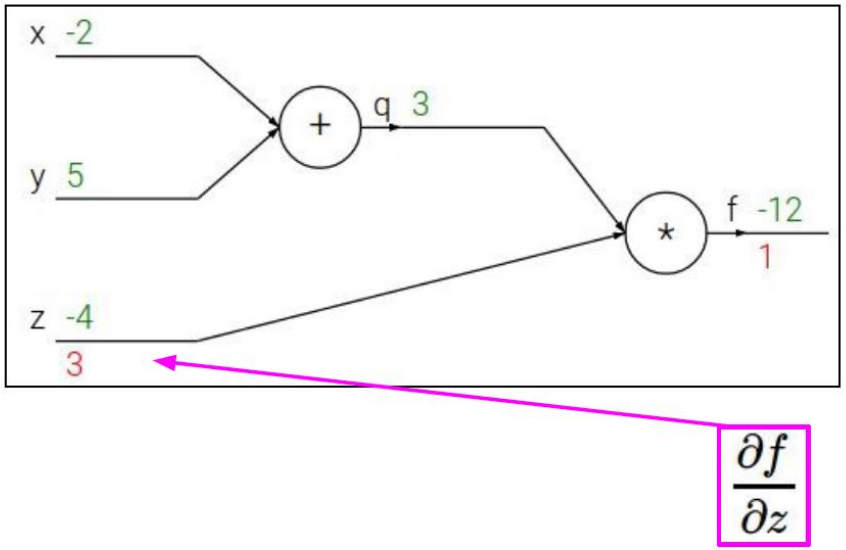
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation - A Simple Example

Backpropagation: a simple example

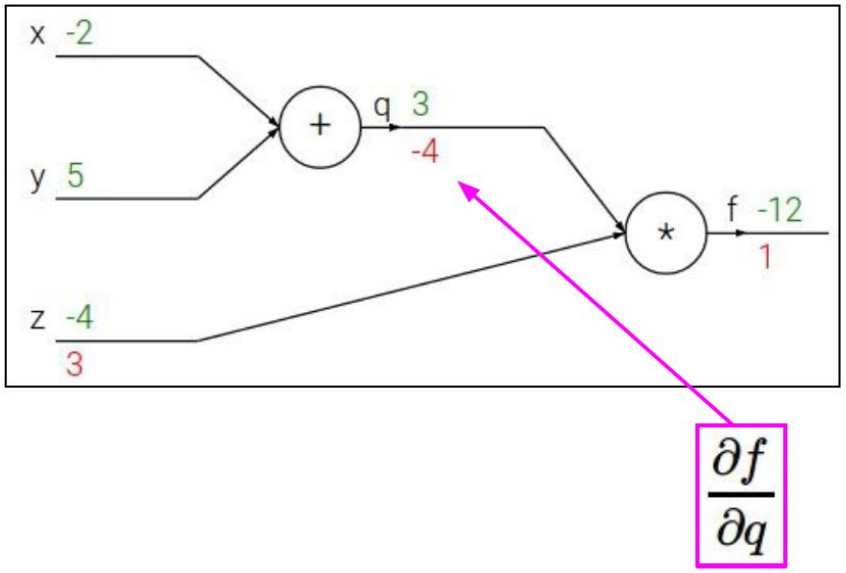
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation - A Simple Example

Backpropagation: a simple example

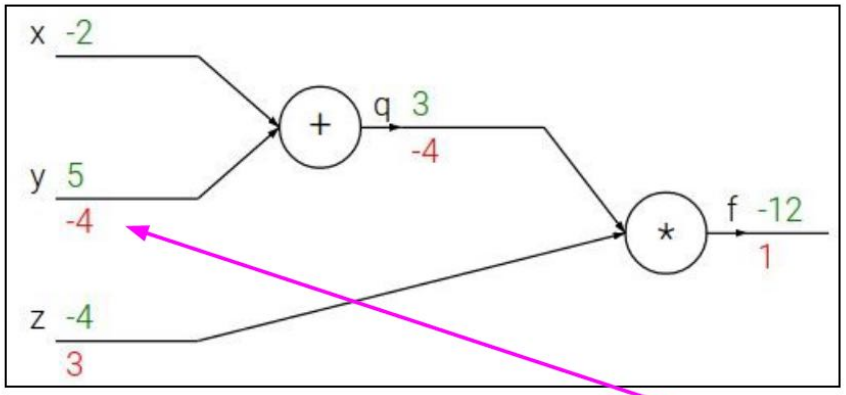
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Upstream gradient

Local gradient

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation - A Simple Example

Backpropagation: a simple example

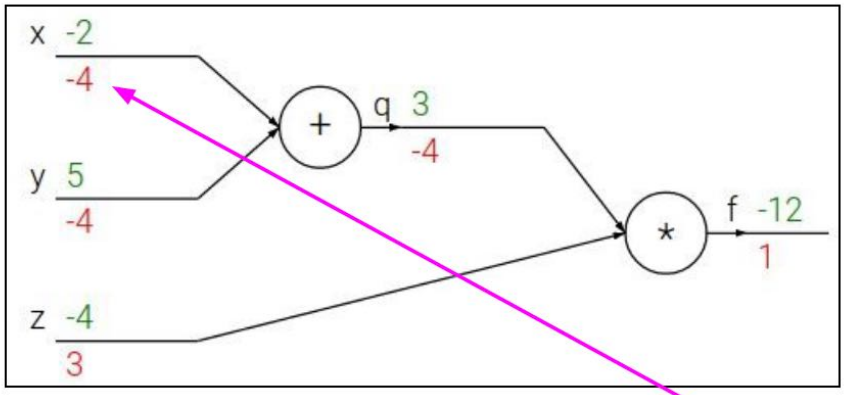
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

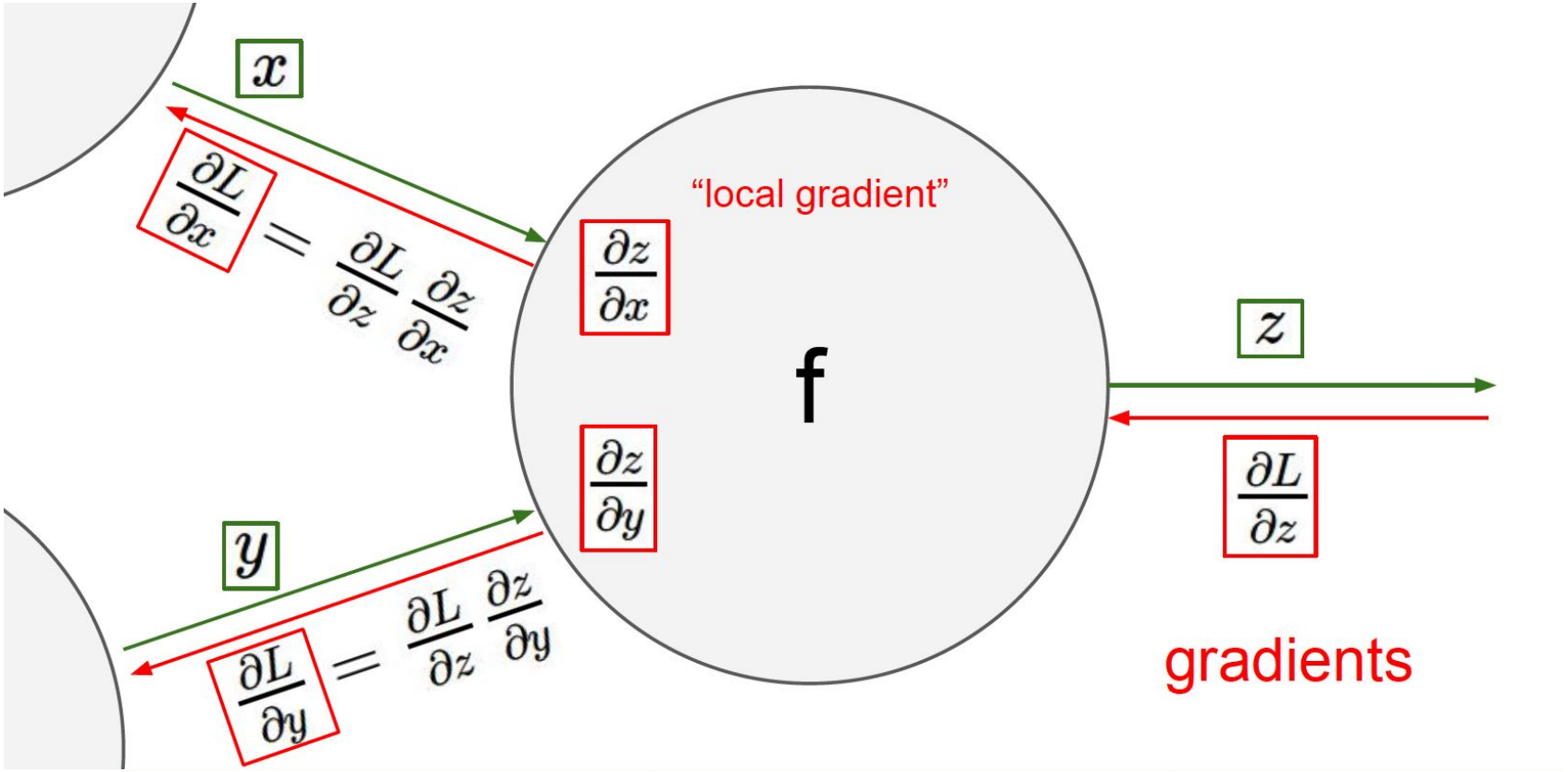
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Upstream gradient

Local gradient

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

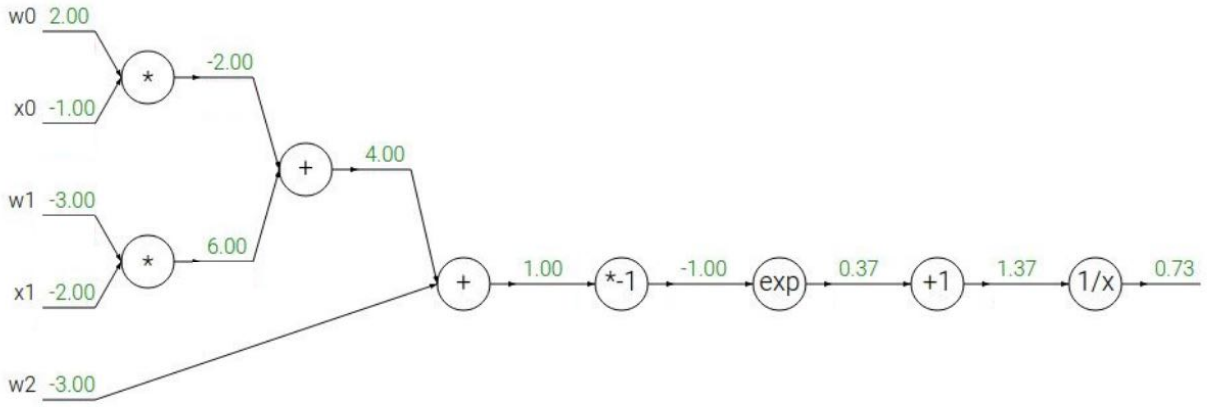
Backpropagation - Local View



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

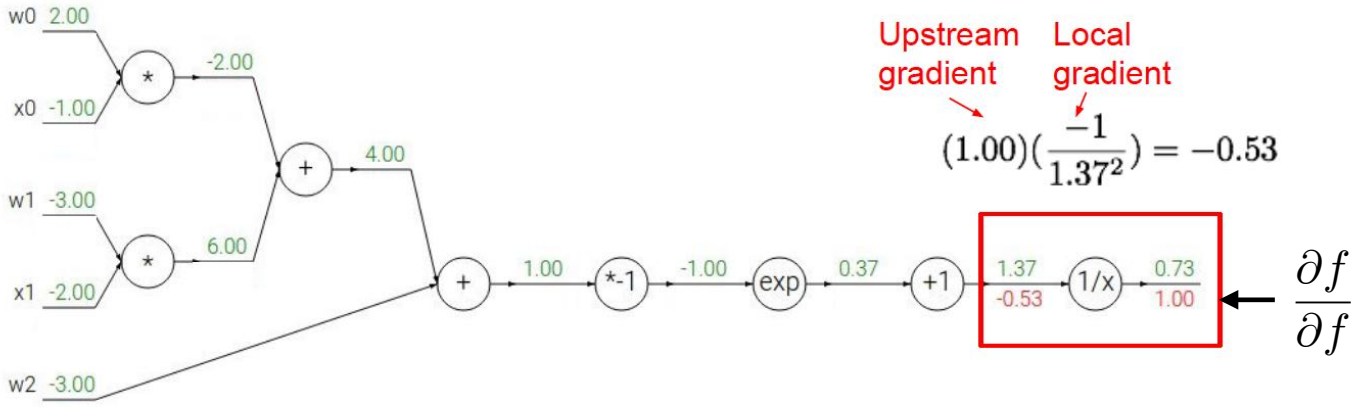
$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

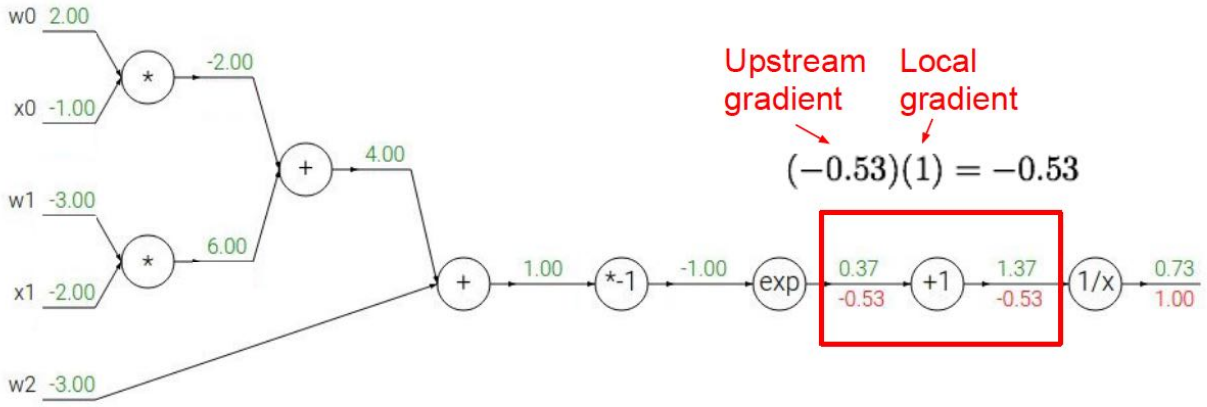
$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

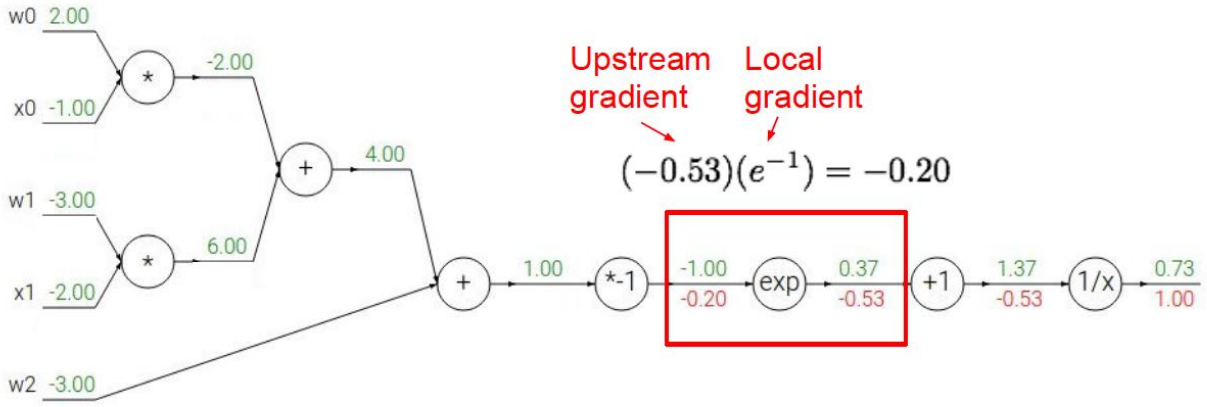
$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

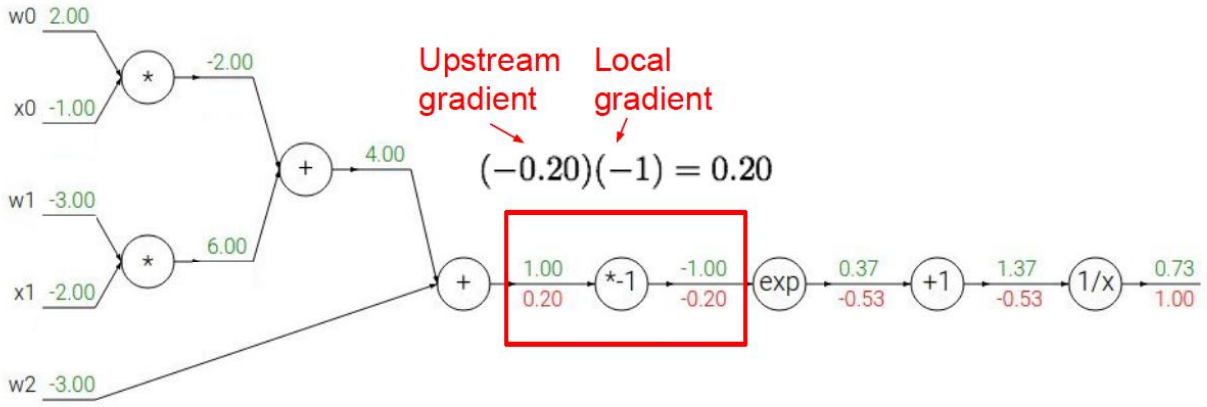


$f(x) = e^x$	$\rightarrow$	$\frac{df}{dx} = e^x$	$f(x) = \frac{1}{x}$	$\rightarrow$	$\frac{df}{dx} = -1/x^2$
$f_a(x) = ax$	$\rightarrow$	$\frac{df}{dx} = a$	$f_c(x) = c + x$	$\rightarrow$	$\frac{df}{dx} = 1$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

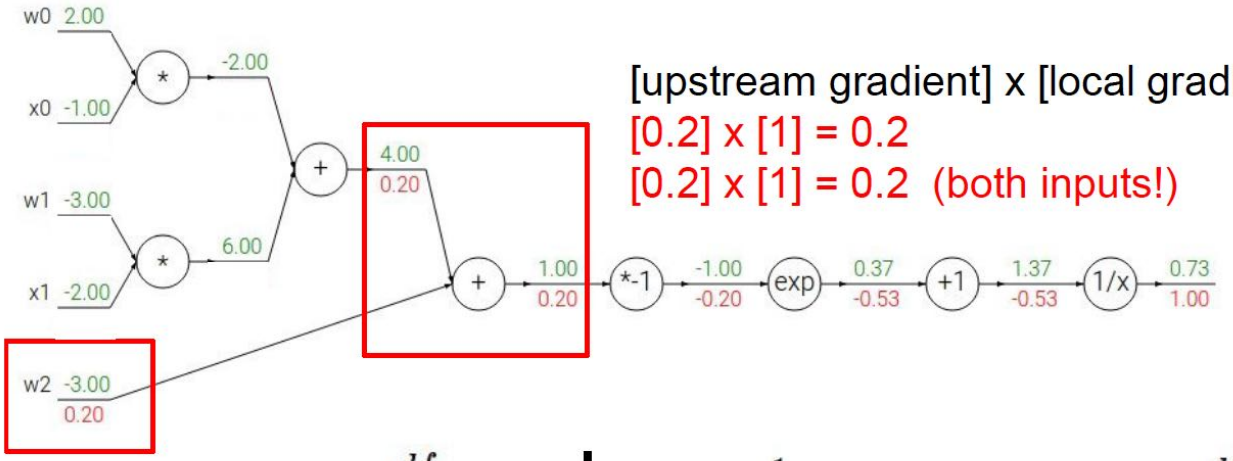
$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



[upstream gradient] x [local gradient]
[0.2] x [1] = 0.2
[0.2] x [1] = 0.2 (both inputs!)

$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

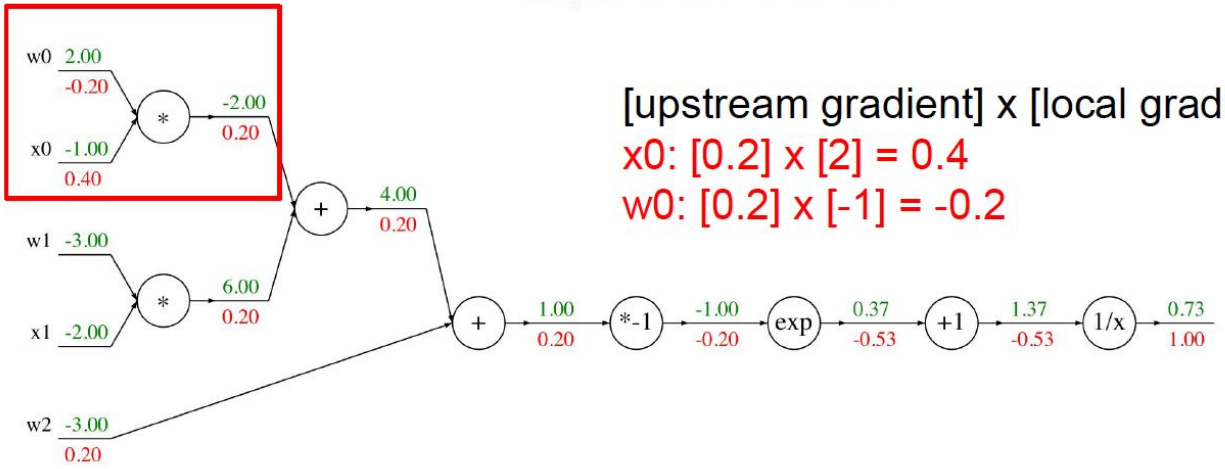
$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



[upstream gradient] x [local gradient]

x0: [0.2] x [2] = 0.4

w0: [0.2] x [-1] = -0.2

$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation

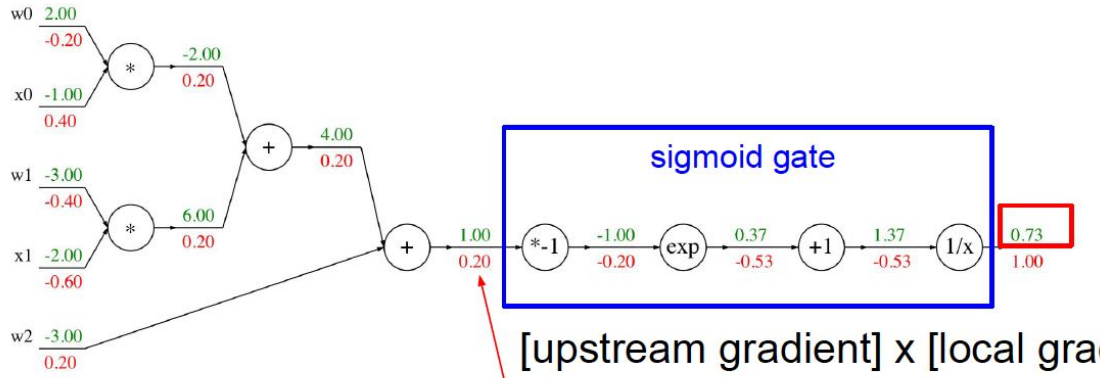
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

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

sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



[upstream gradient] x [local gradient]

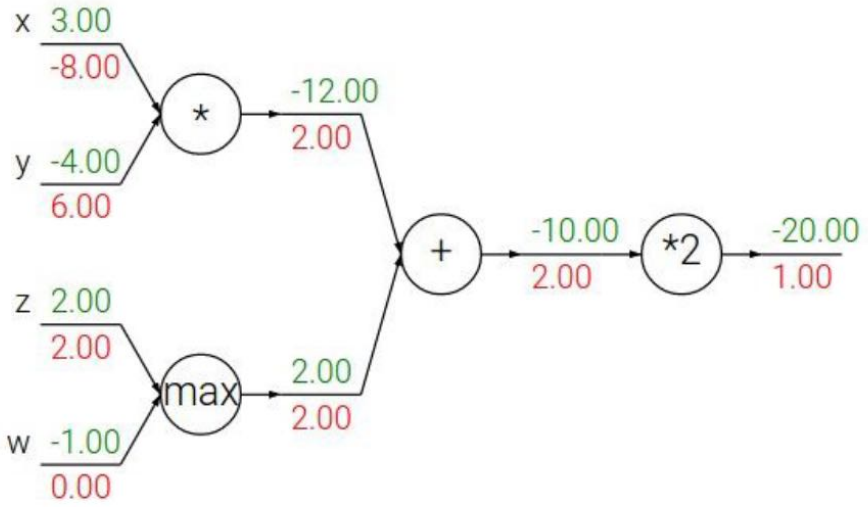
[1.00] x [(1 - 0.73) (0.73)]= 0.2

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation

Patterns in backward flow

add gate: gradient distributor



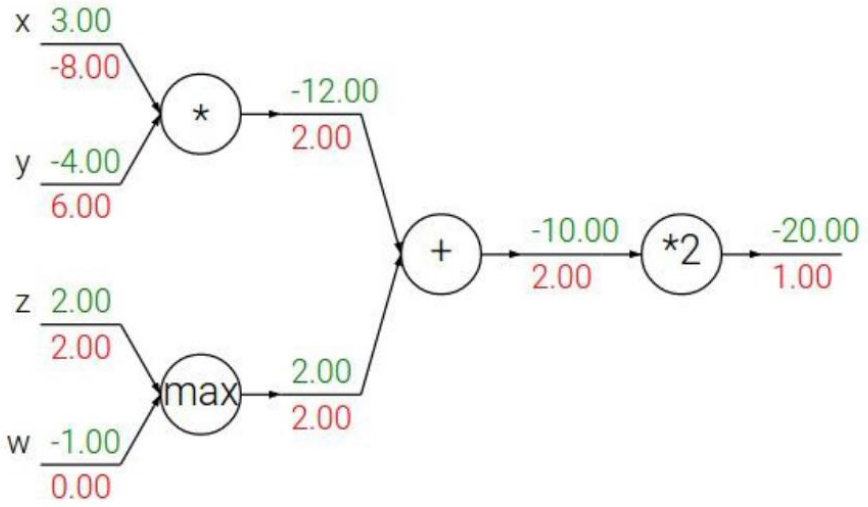
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation

Patterns in backward flow

add gate: gradient distributor

Q: What is a **max** gate?



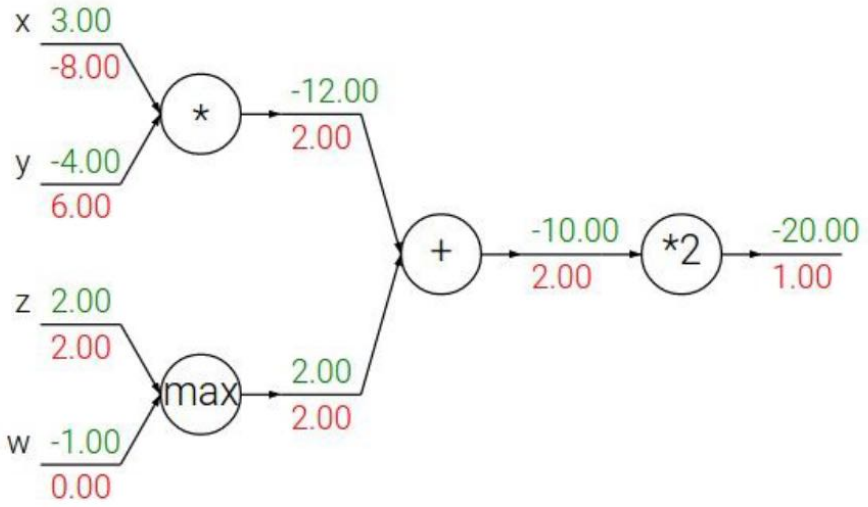
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation

Patterns in backward flow

add gate: gradient distributor

max gate: gradient router



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

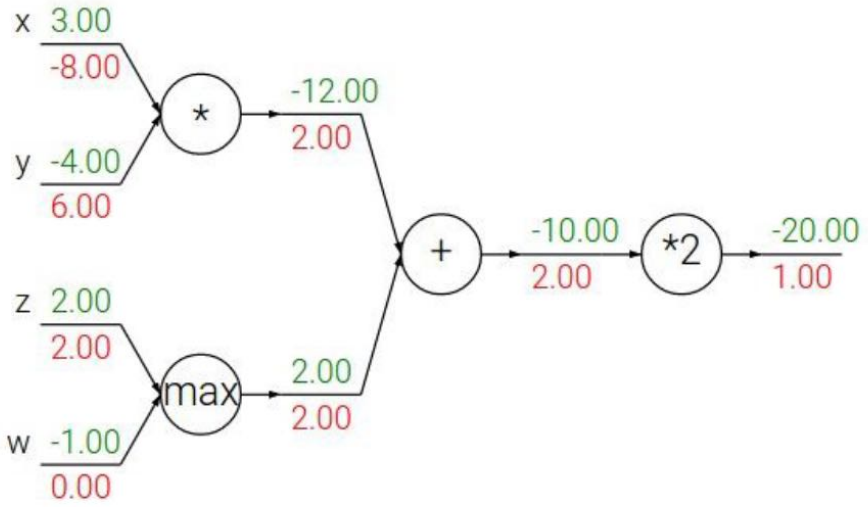
Backpropagation

Patterns in backward flow

add gate: gradient distributor

max gate: gradient router

Q: What is a **mul** gate?



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

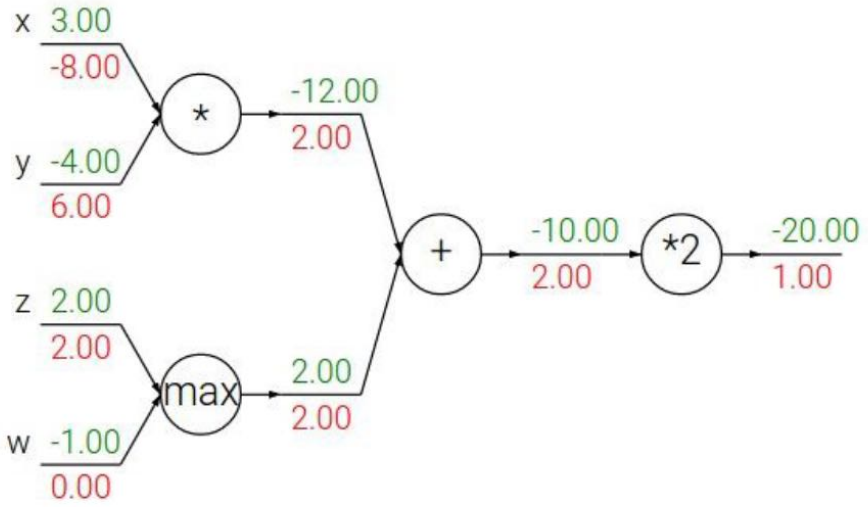
Backpropagation

Patterns in backward flow

add gate: gradient distributor

max gate: gradient router

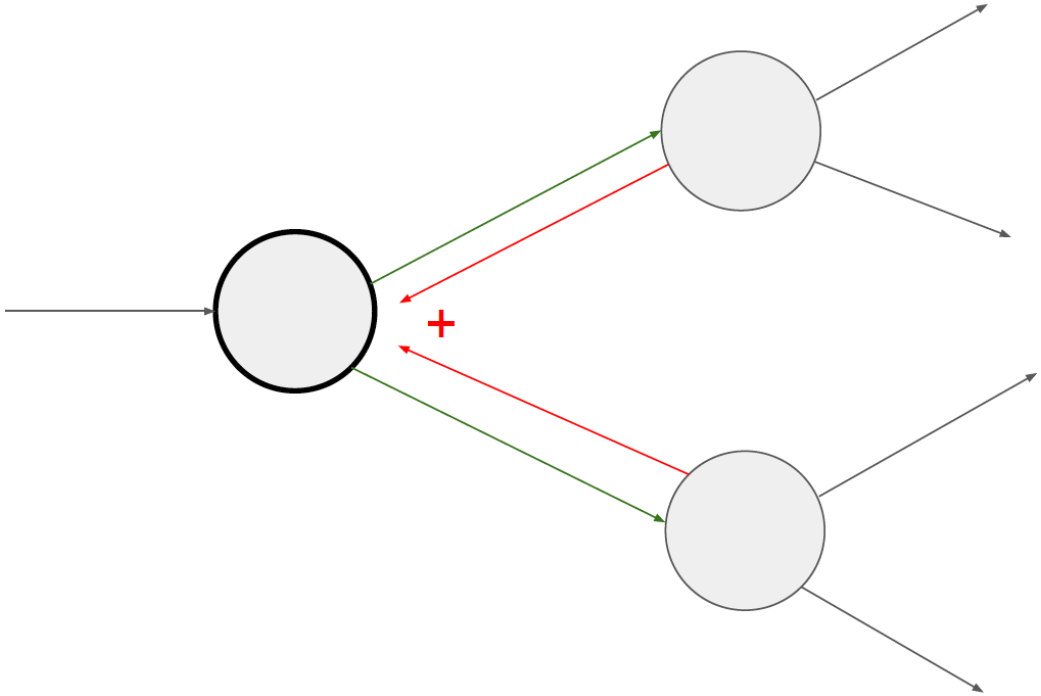
mul gate: gradient switcher



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

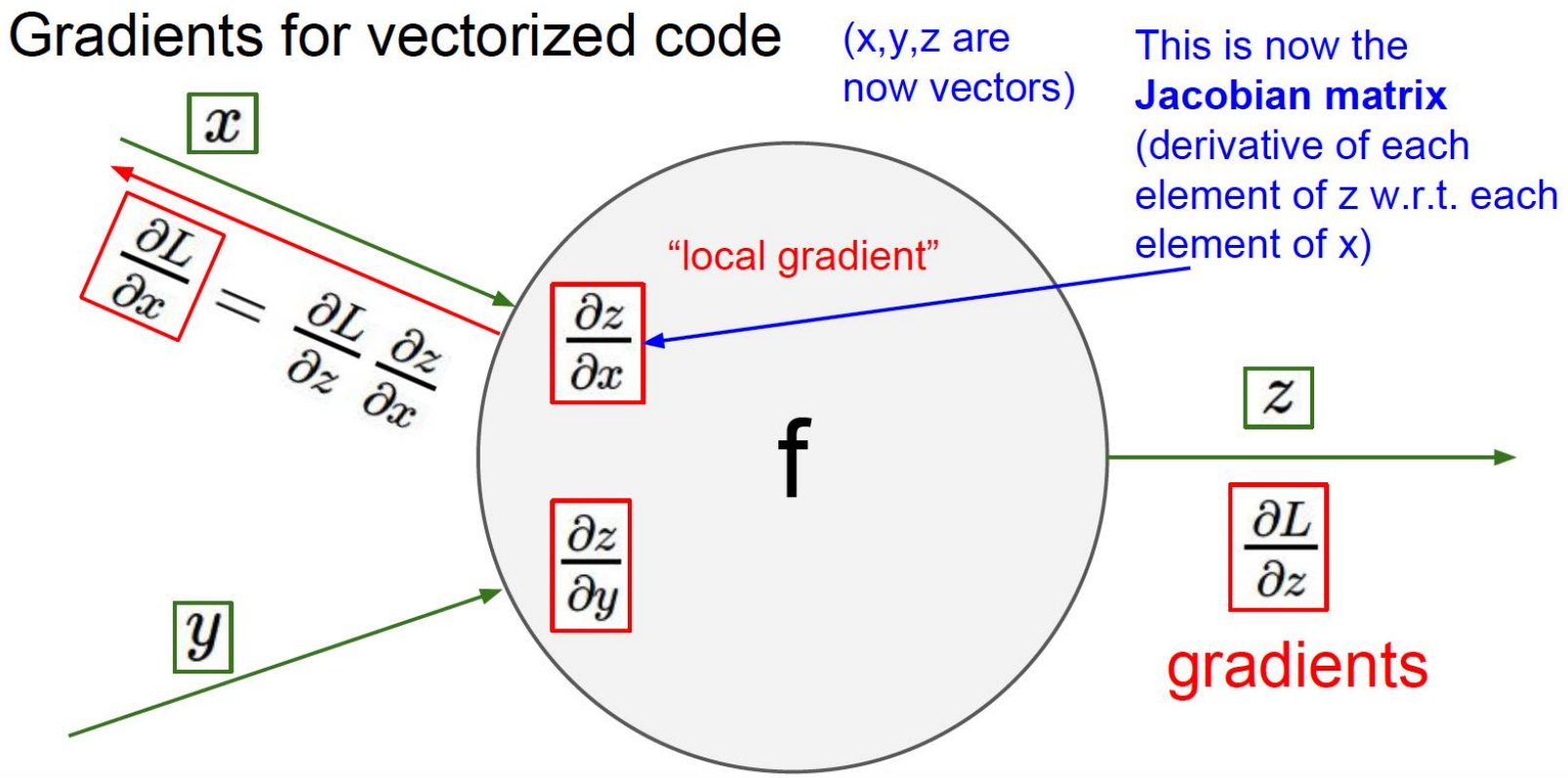
Backpropagation

Gradients add at branches



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

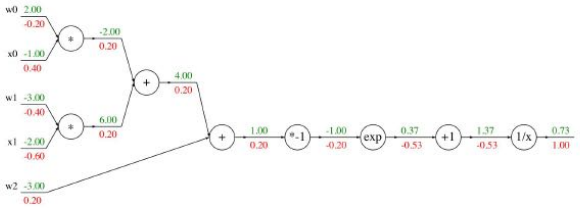
Backpropagation



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation

Modularized implementation: forward / backward API



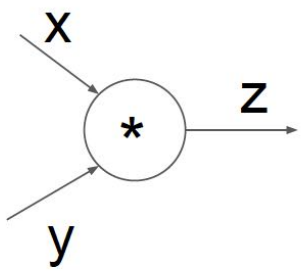
Graph (or Net) object *(rough pseudo code)*

```
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Backpropagation

Modularized implementation: forward / backward API



(x,y,z are scalars)

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

Local gradient

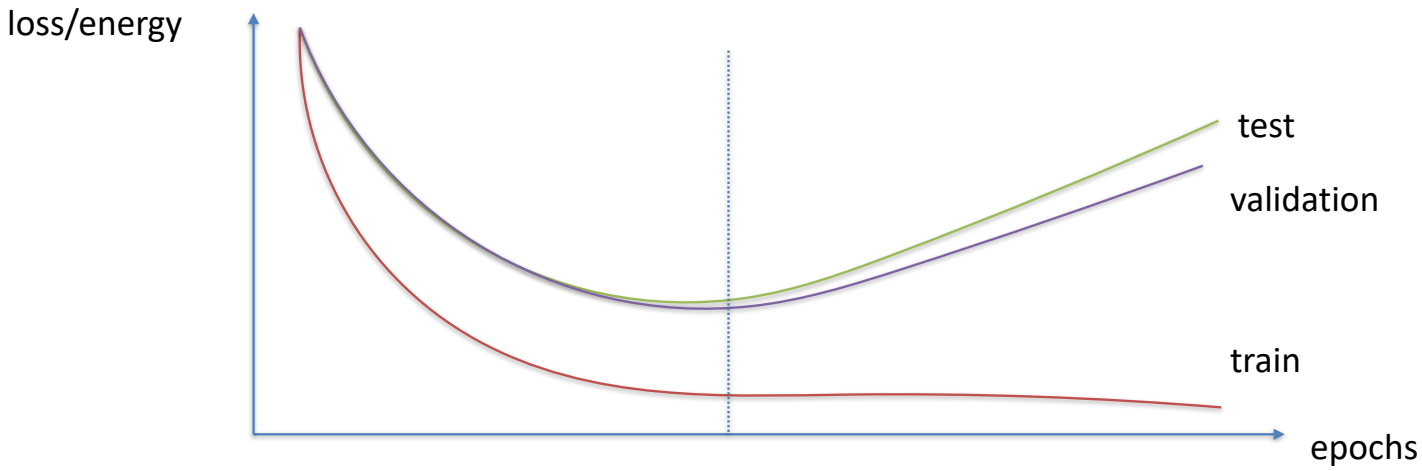
Upstream gradient variable

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Gradient & Training

- Only two things need to be implemented to define new layer:
  - forward pass
  - error back propagation

- Watch overfitting



Summary so far...

- neural nets will be very large: impractical to write down gradient formula by hand for all parameters
- **backpropagation** = recursive application of the chain rule along a computational graph to compute the gradients of all inputs/parameters/intermediates
- implementations maintain a graph structure, where the nodes implement the **forward**() / **backward**() API
- **forward**: compute result of an operation and save any intermediates needed for gradient computation in memory
- **backward**: apply the chain rule to compute the gradient of the loss function with respect to the inputs

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Overview Today's Lecture

- Backpropagation — Gradient Descent
  - illustrated using computational graphs
  - chain rule - upstream and local gradients
  - modularization simple

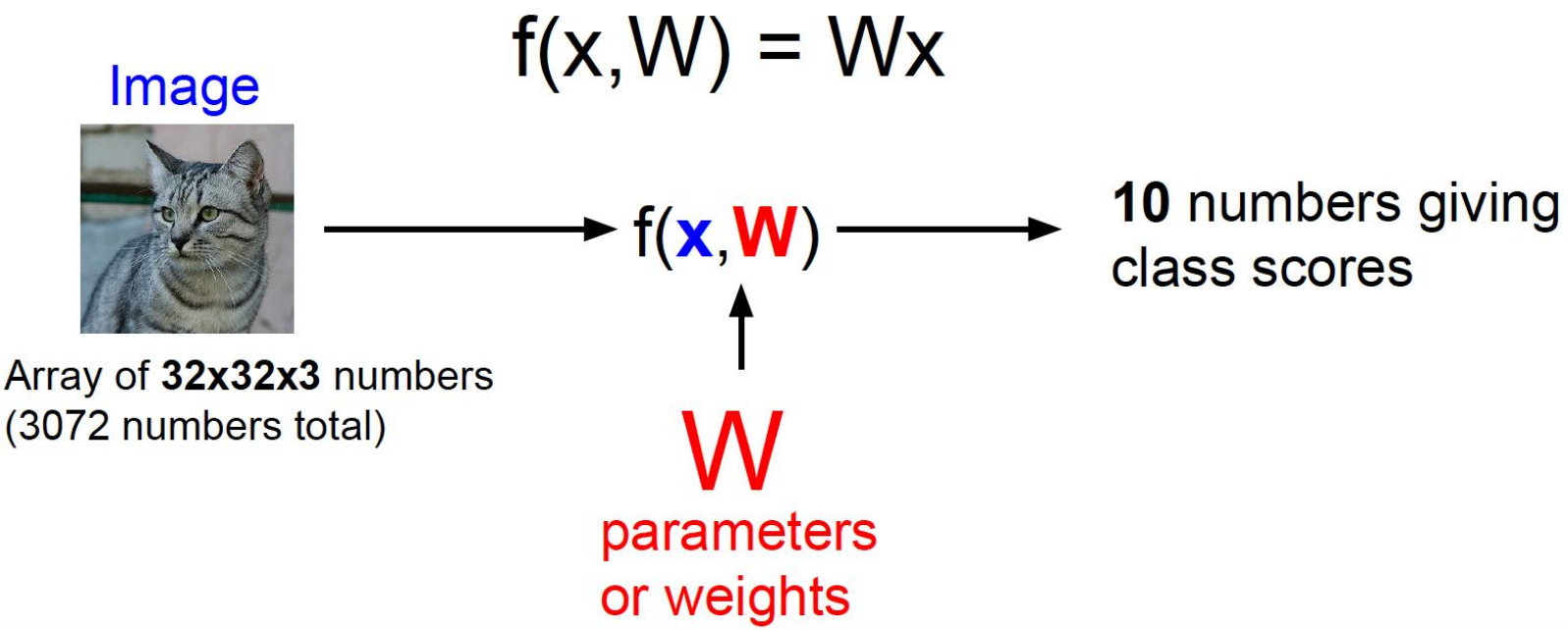
- **Convolutional Neural Networks (CNNs)**
  - versatile architecture & motivation
    (one of the highly successful neural networks…)
  - convolution layer + activation function + pooling + fully connected layer
  - depth matters

max planck institut
informatik

SIC Saarland Informatics Campus

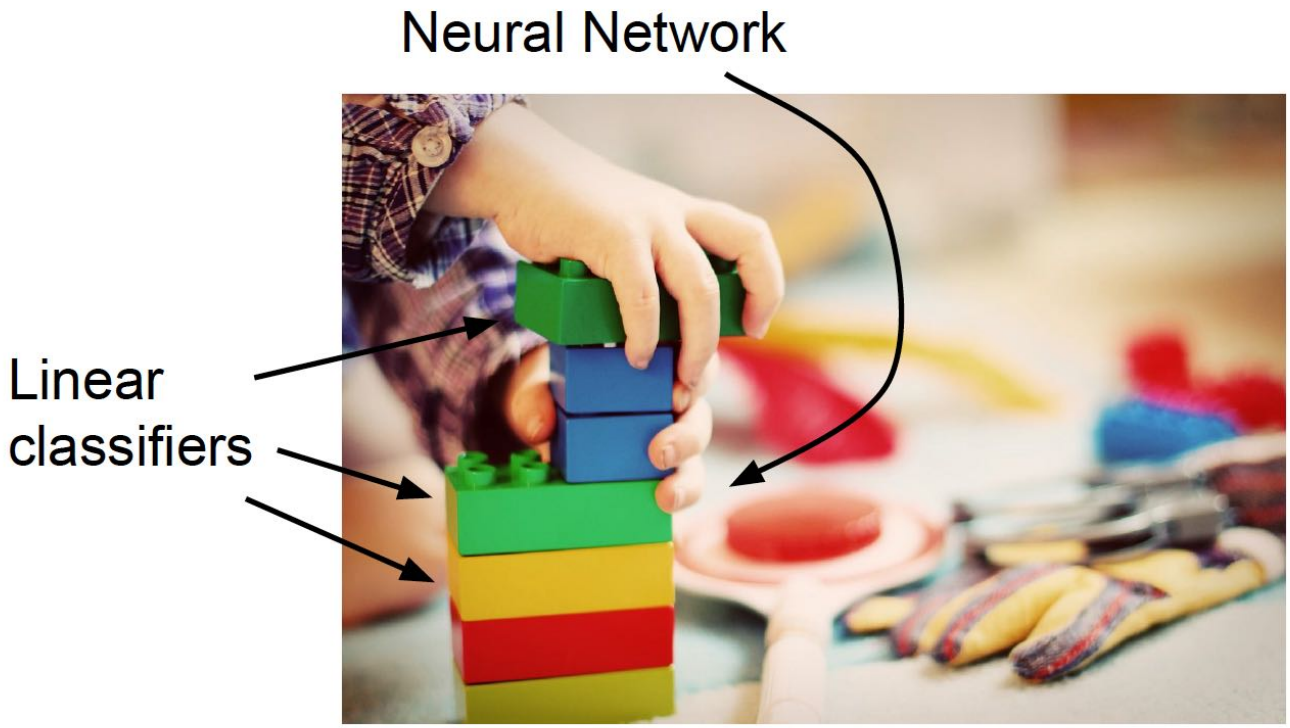
“Standard” Neural Networks

Parametric Approach: Linear Classifier



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Linear Classification



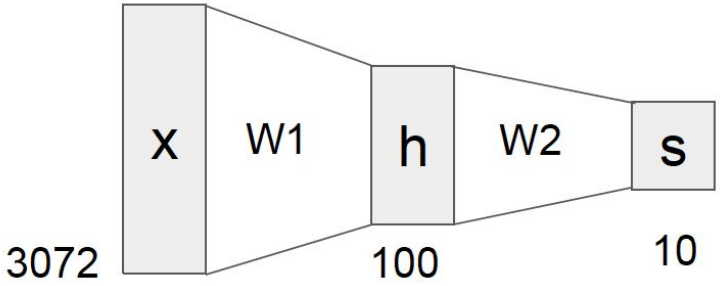
This image is CC0 1.0 public domain

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Standard Neural Networks

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

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



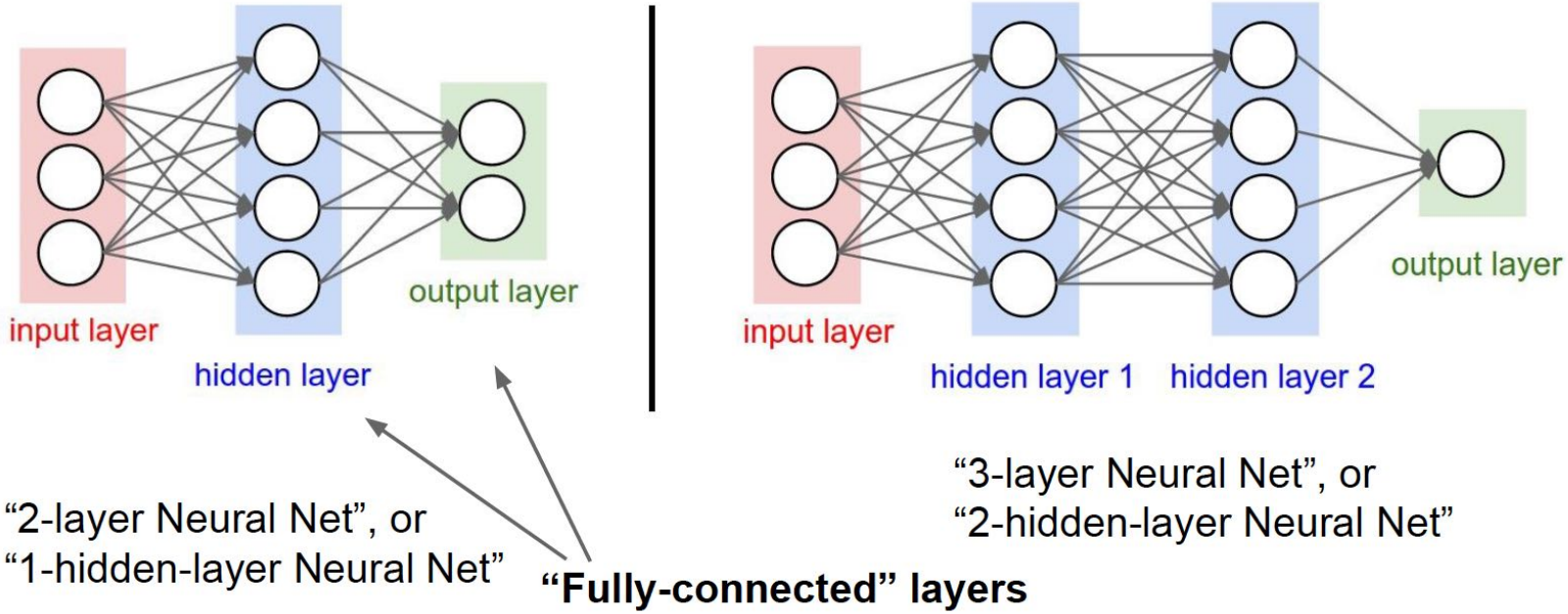
or 3-layer Neural Network

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Standard Neural Networks - Classic Architectures

- (Multi-Layer) Perceptron (also called Fully Connected Network/Layers)



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

max planck institut
informatik

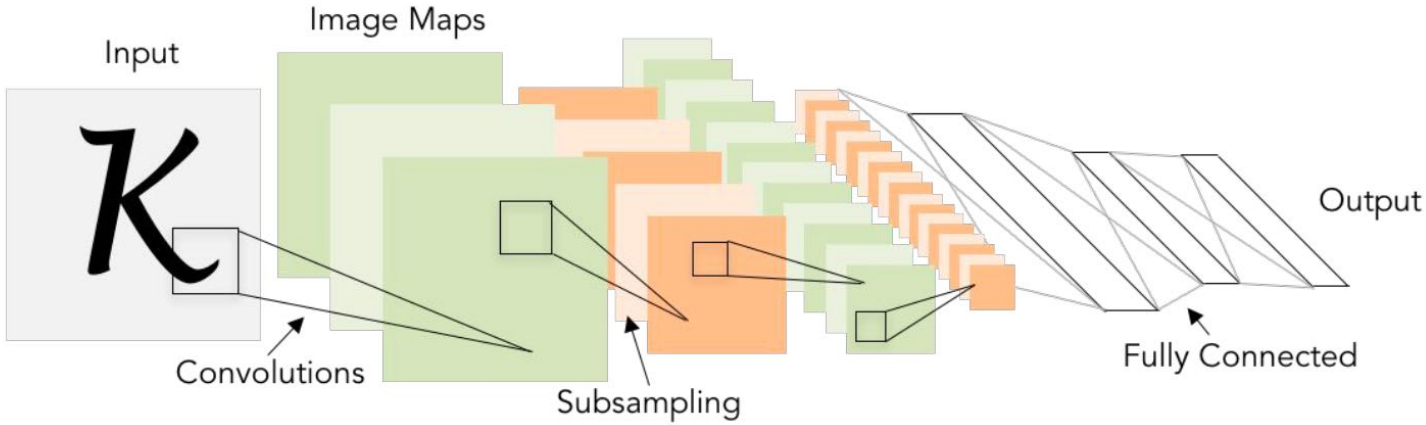
SIC Saarland Informatics Campus

Convolutional Neural Networks (CNNs)

A bit of history:

Gradient-based learning applied to document recognition

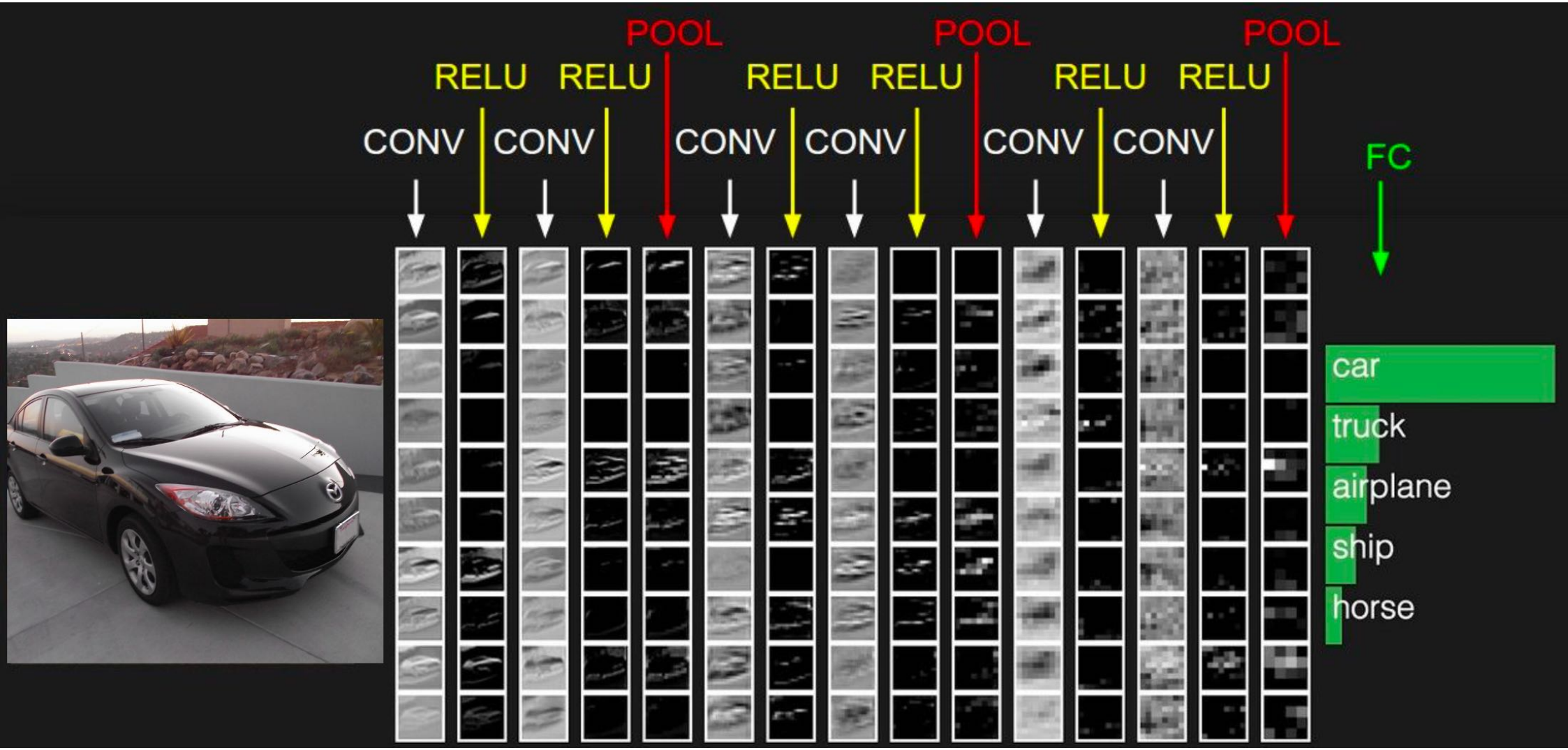
[LeCun, Bottou, Bengio, Haffner 1998]



LeNet-5

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

preview:



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

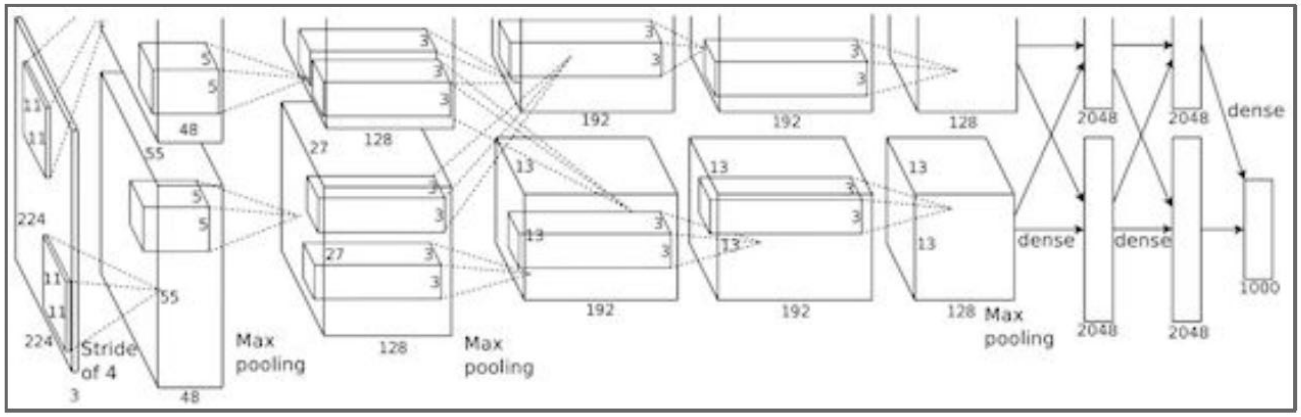


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

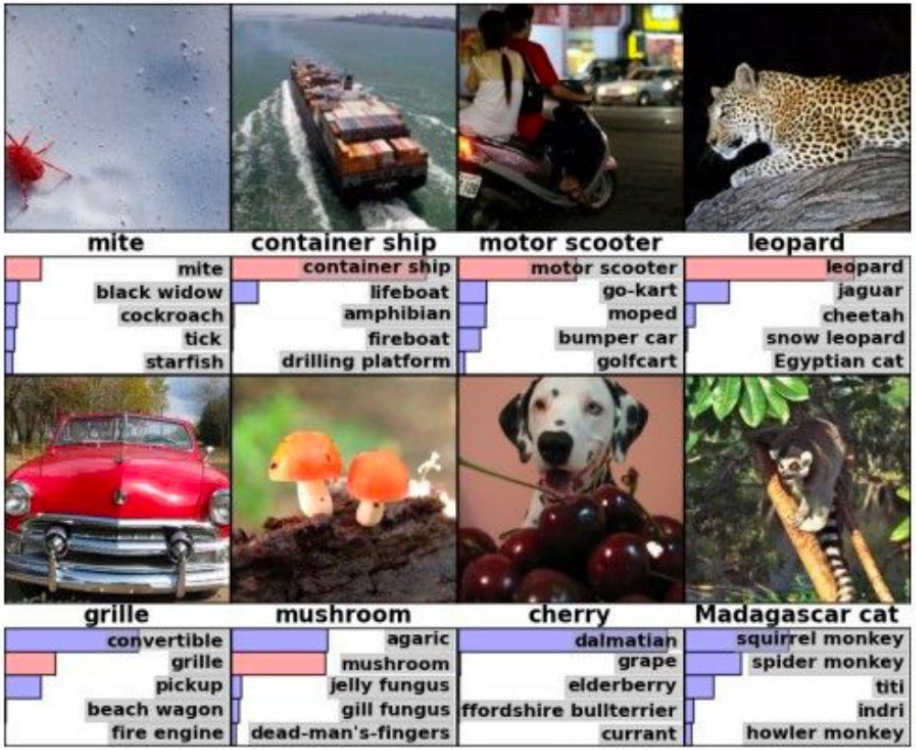
"AlexNet"

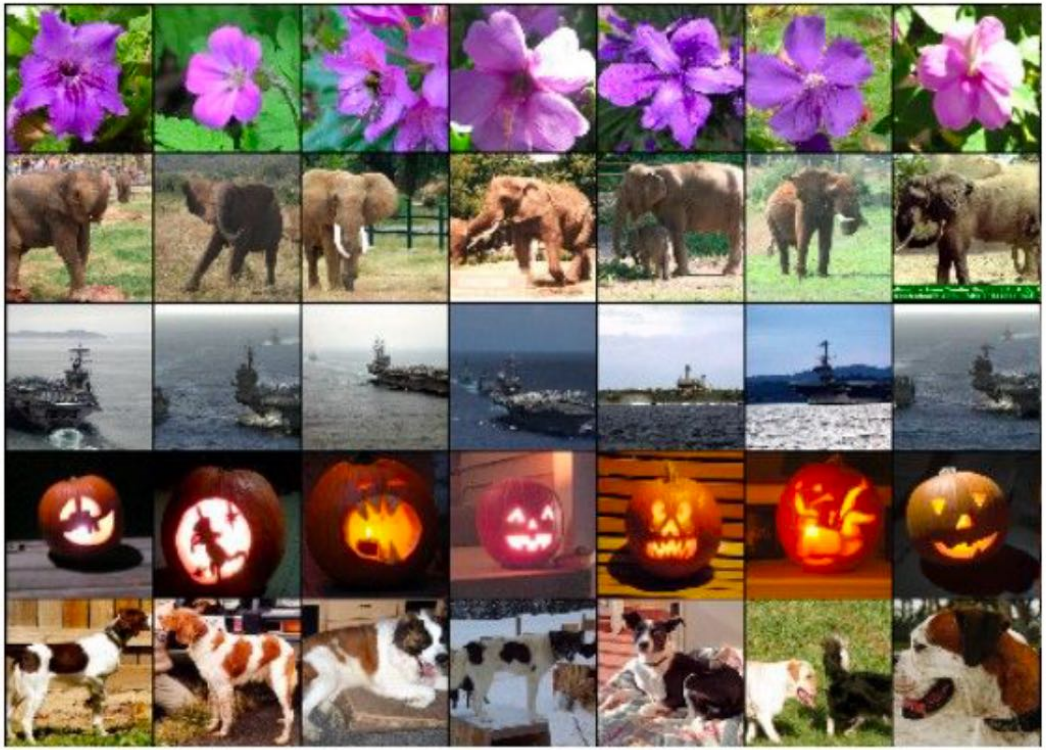
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Fast-forward to today: ConvNets are everywhere

Classification

Retrieval

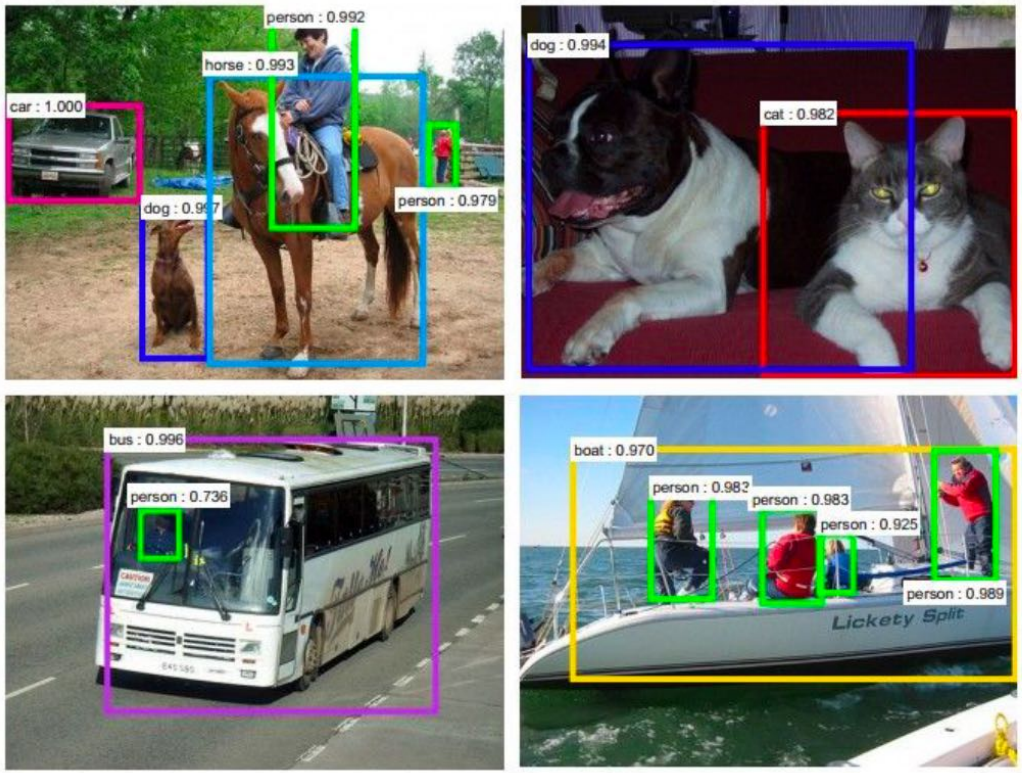




slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Fast-forward to today: ConvNets are everywhere

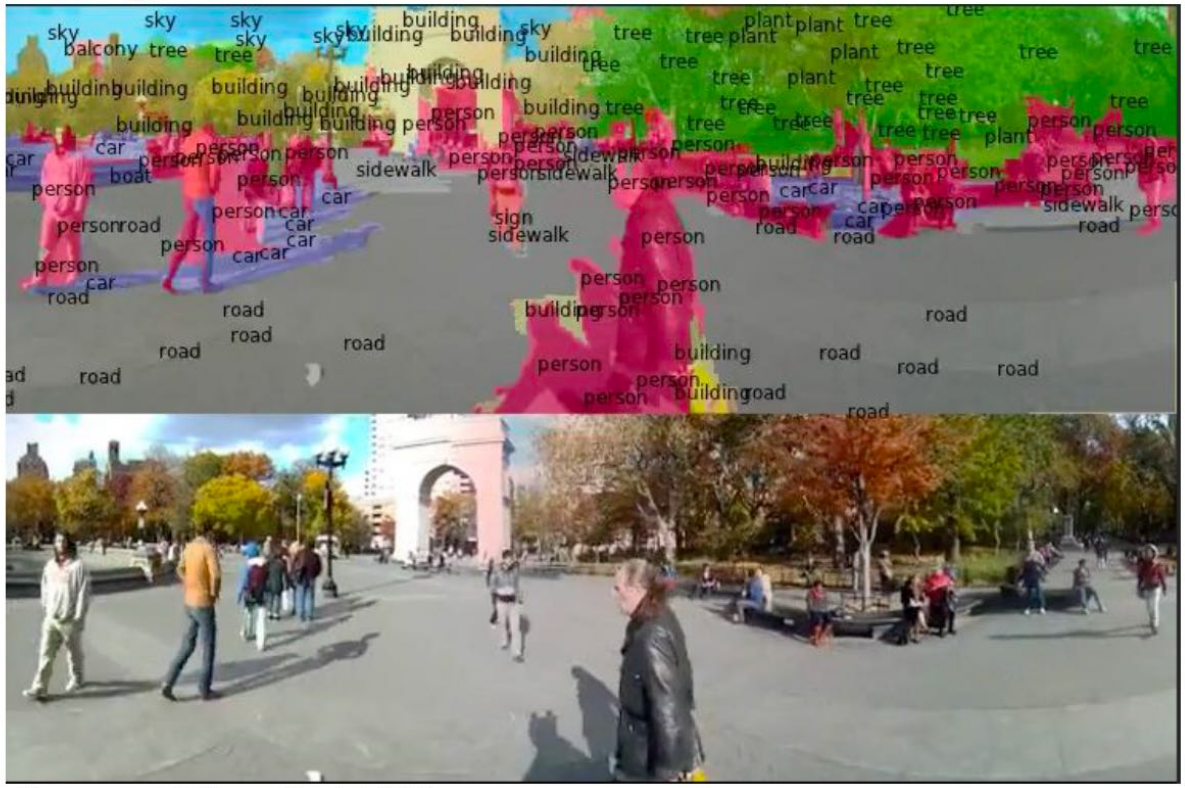
Detection



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

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

Segmentation

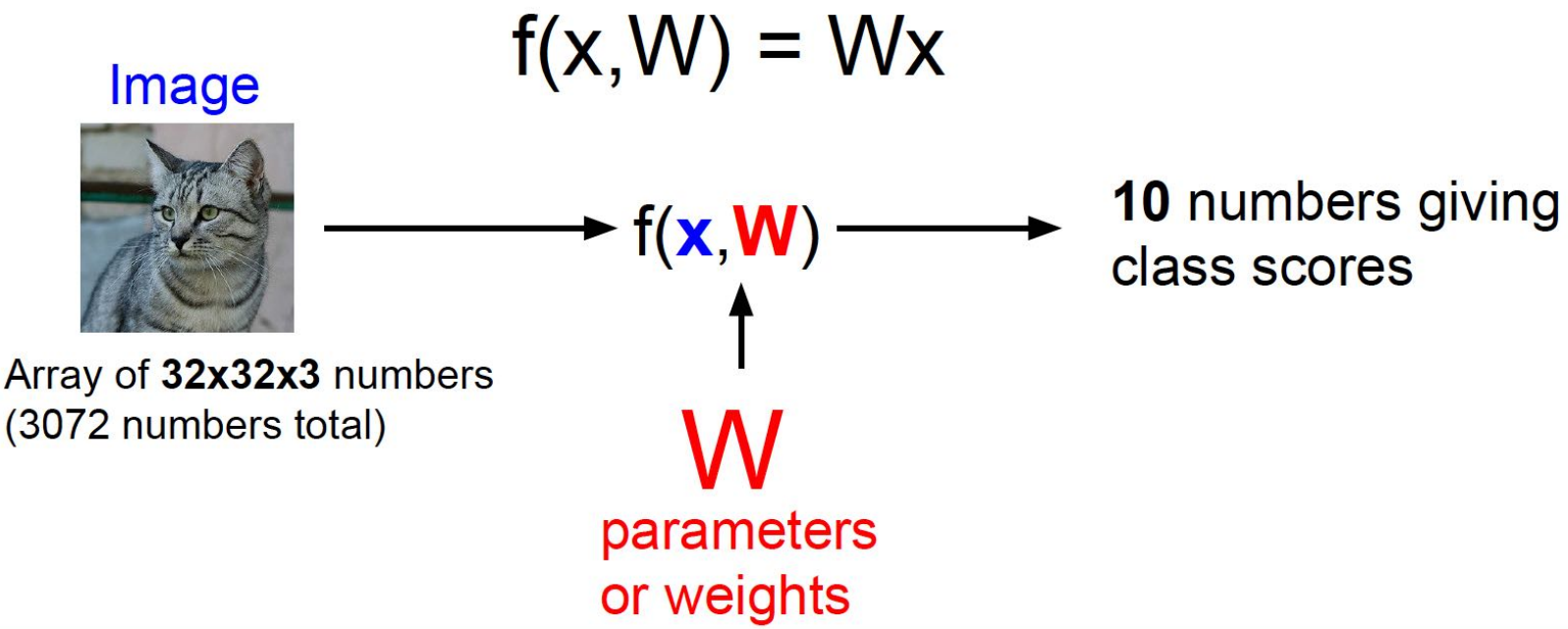


Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

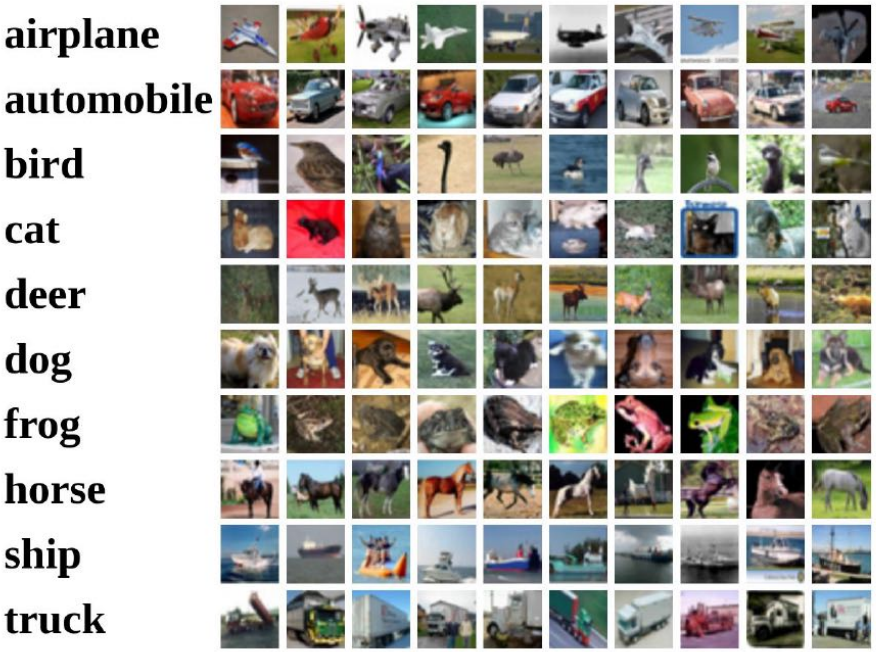
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Parametric Approach: Linear Classifier



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Recall CIFAR10

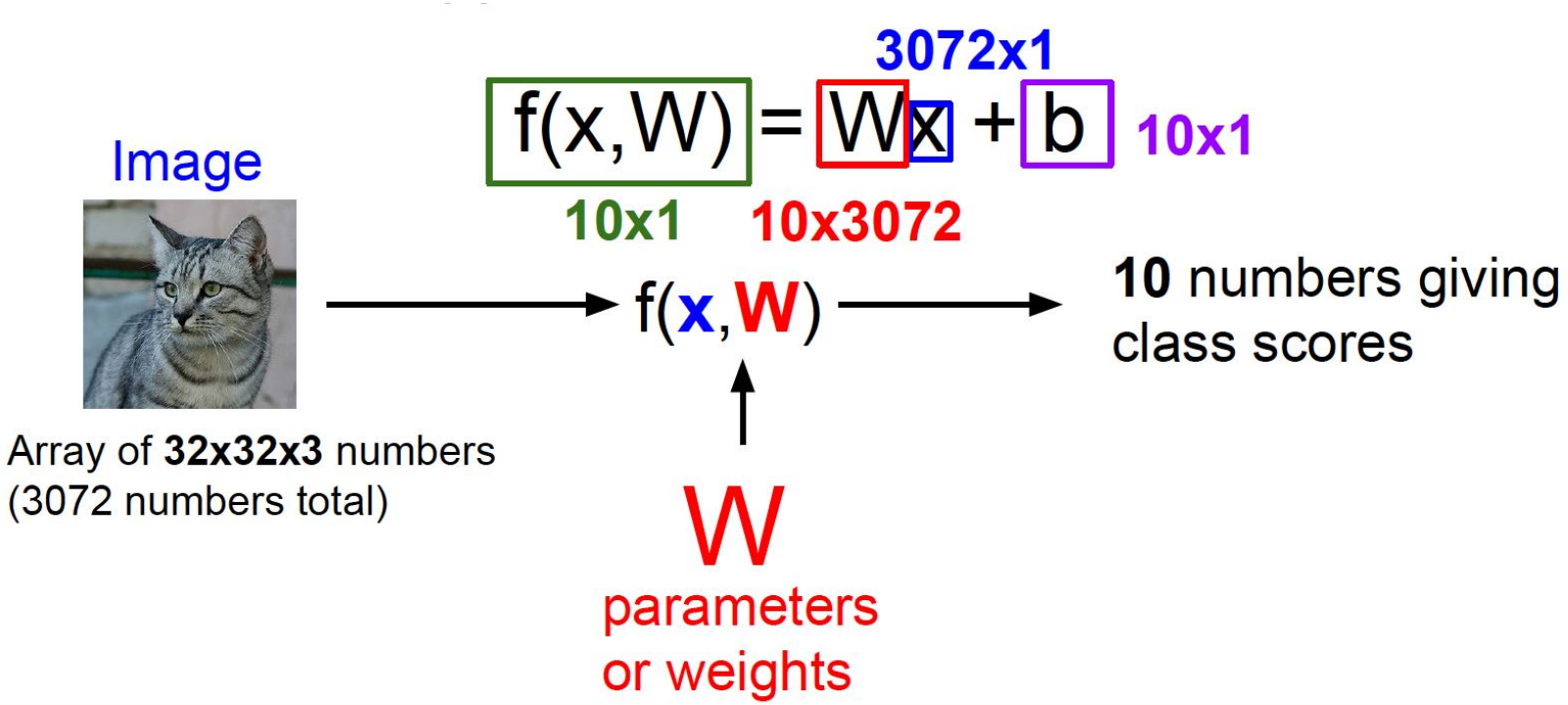


50,000 training images
each image is **32x32x3**

10,000 test images.

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

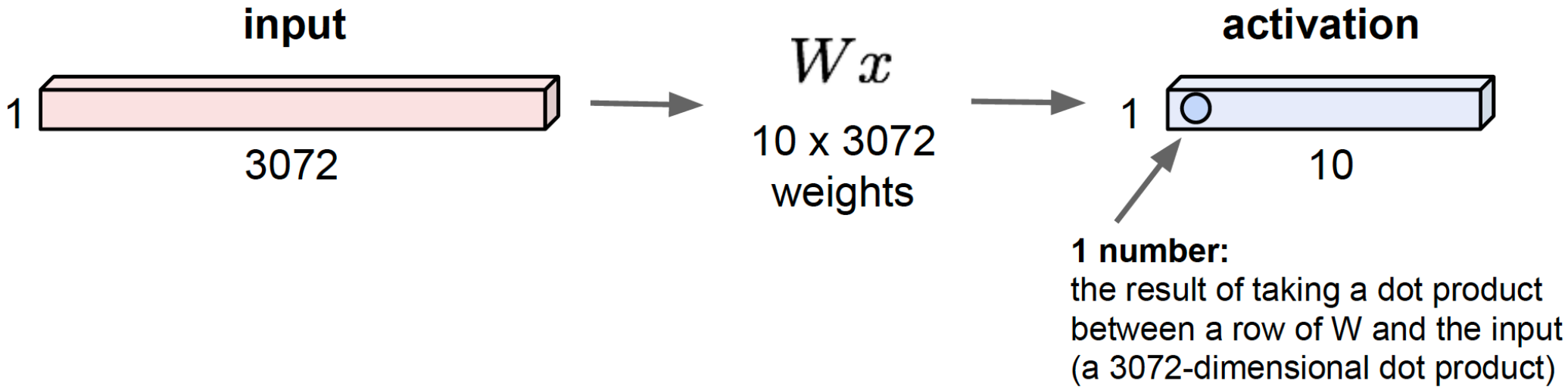
Parametric Approach: Linear Classifier



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Fully Connected Layer

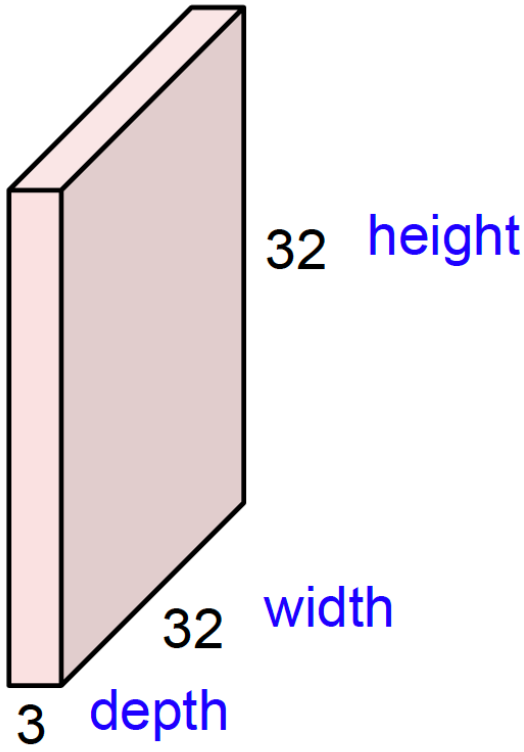
32x32x3 image -> stretch to 3072 x 1



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

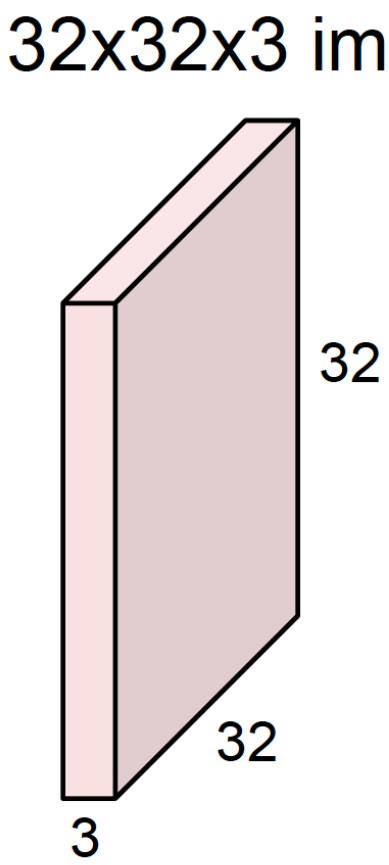
Convolution Layer

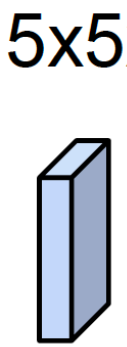
32x32x3 image -> preserve spatial structure



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Convolution Layer





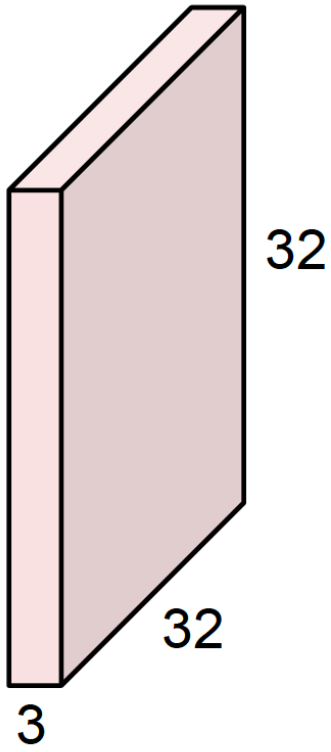
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

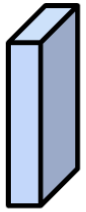
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



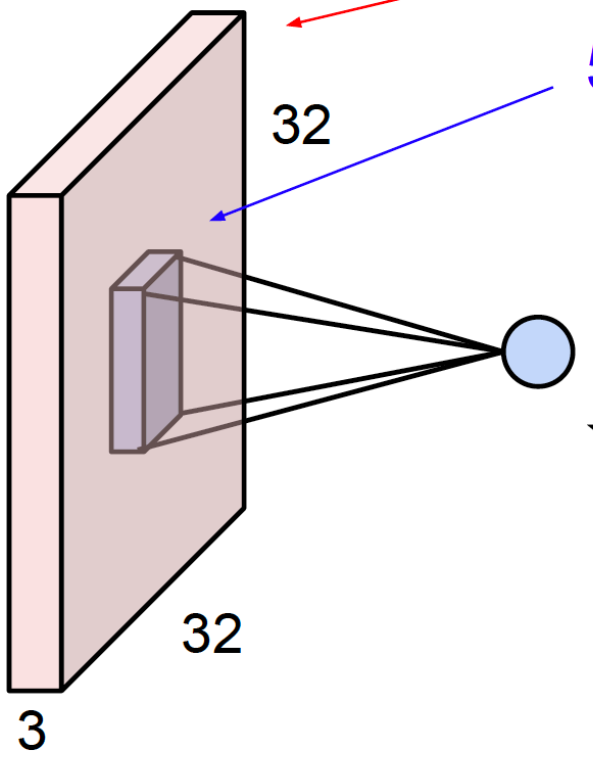
5x5x3 filter



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

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Convolution Layer



32x32x3 image

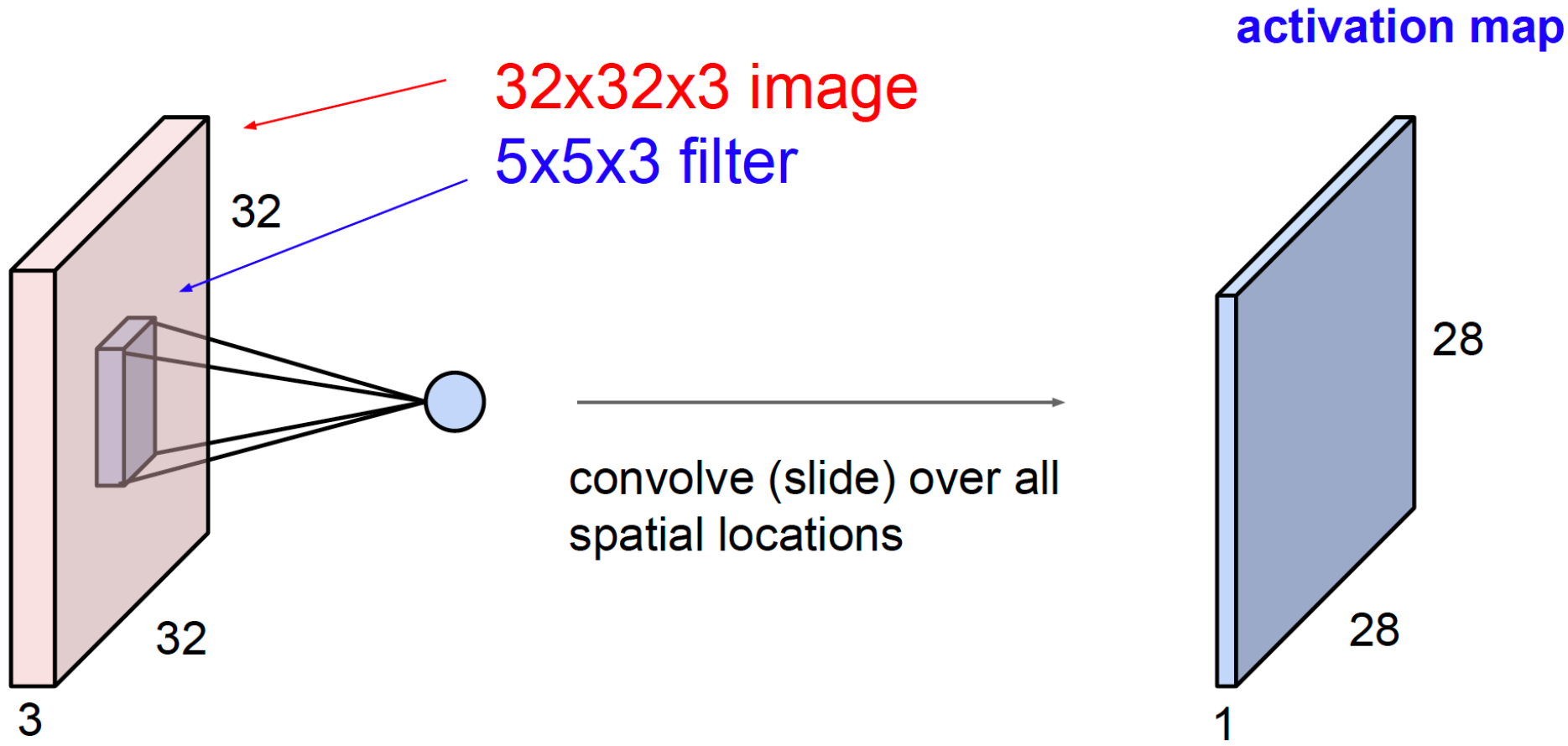
5x5x3 filter w

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)

$$w^T x + b$$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

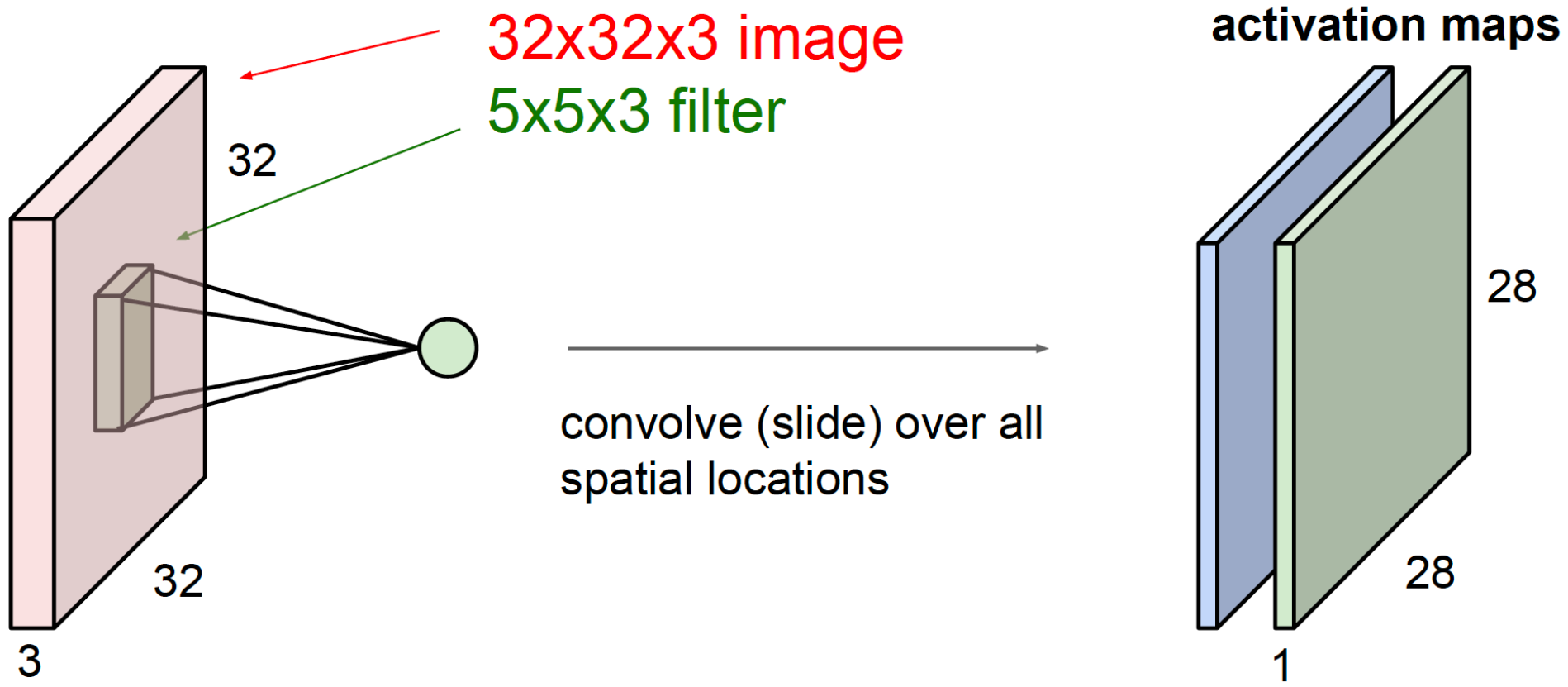
Convolution Layer



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

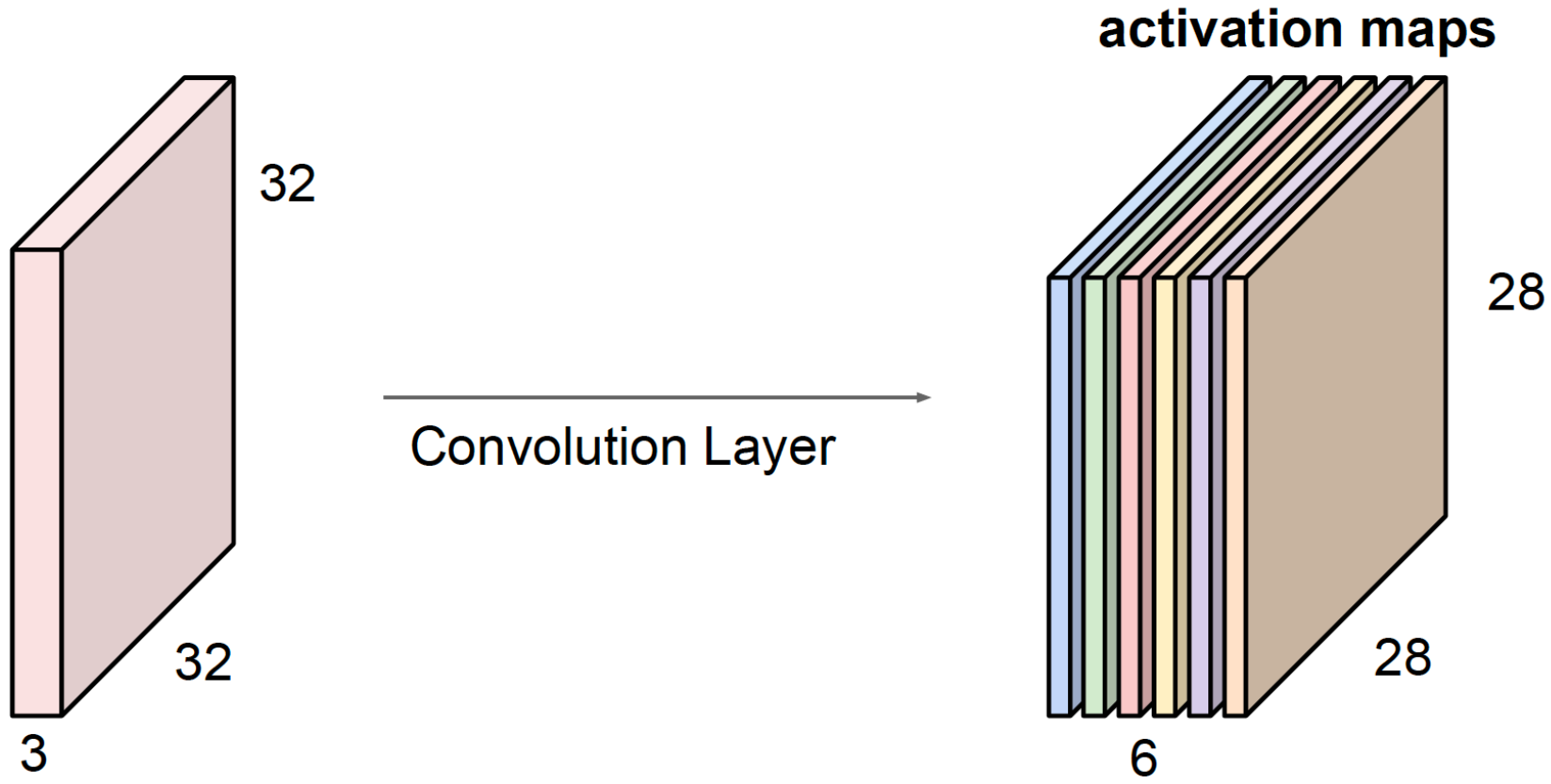
Convolution Layer

consider a second, green filter



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

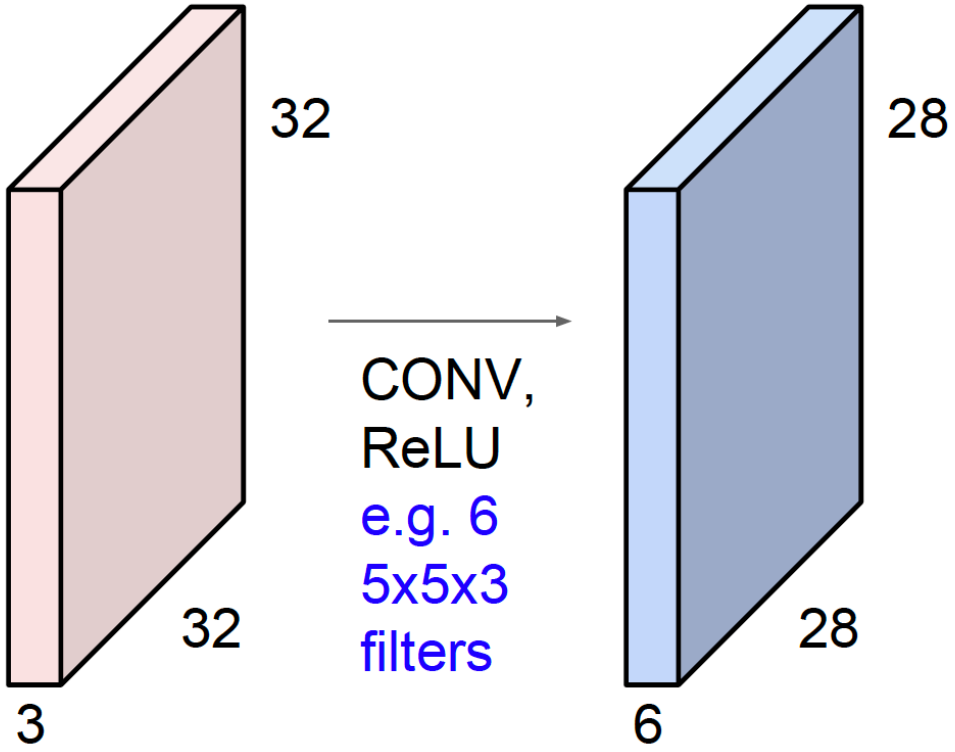
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

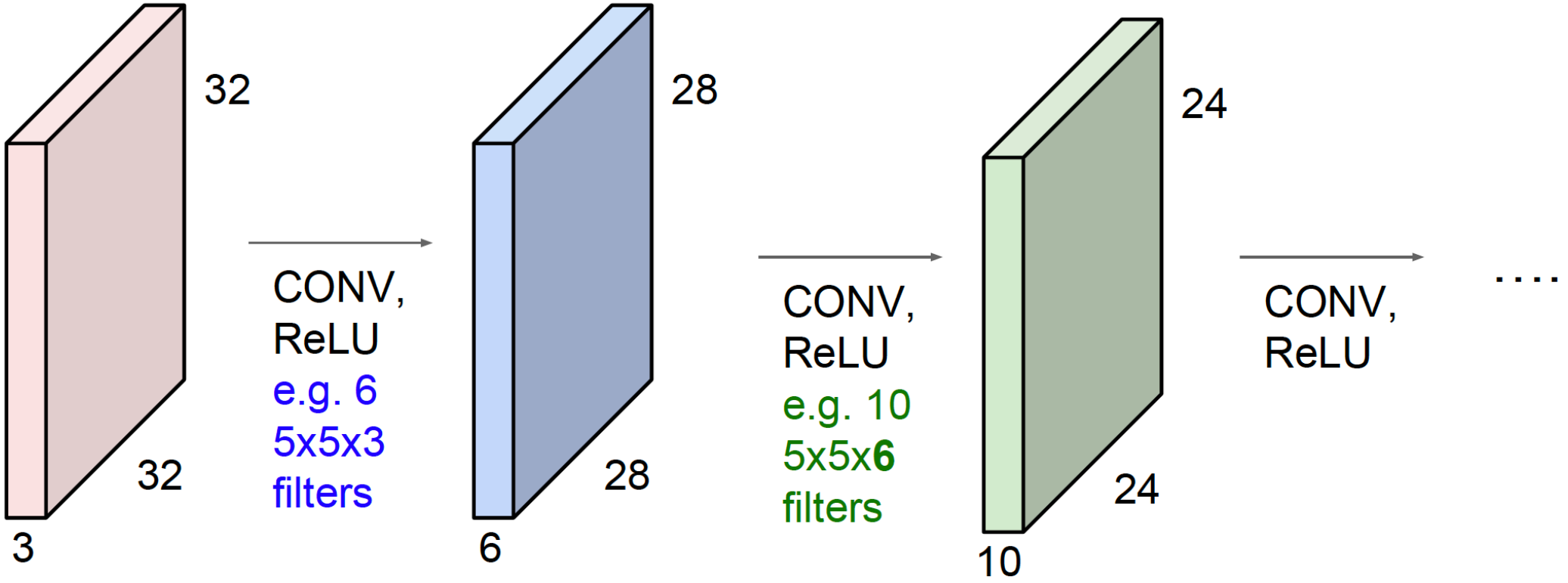
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

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



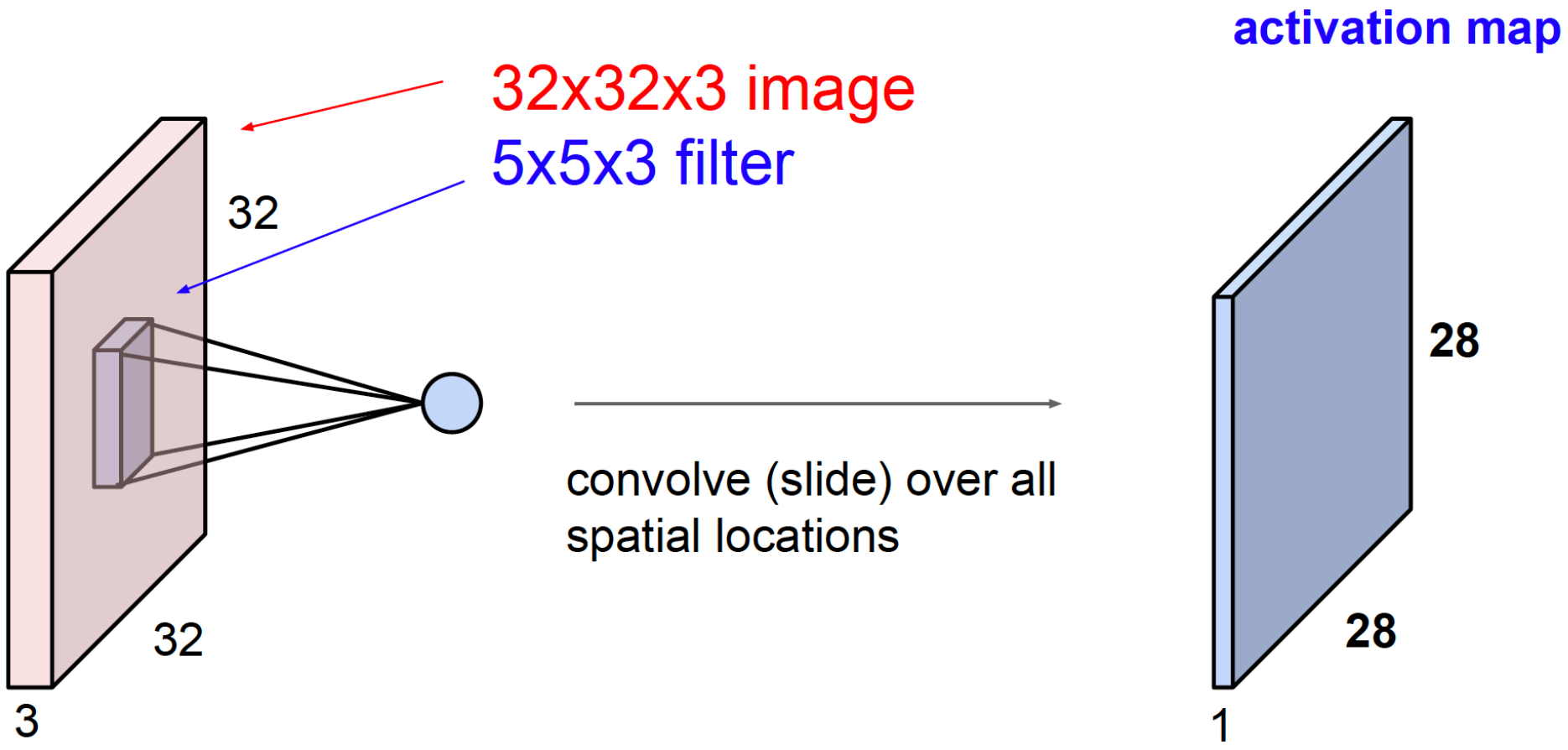
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

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



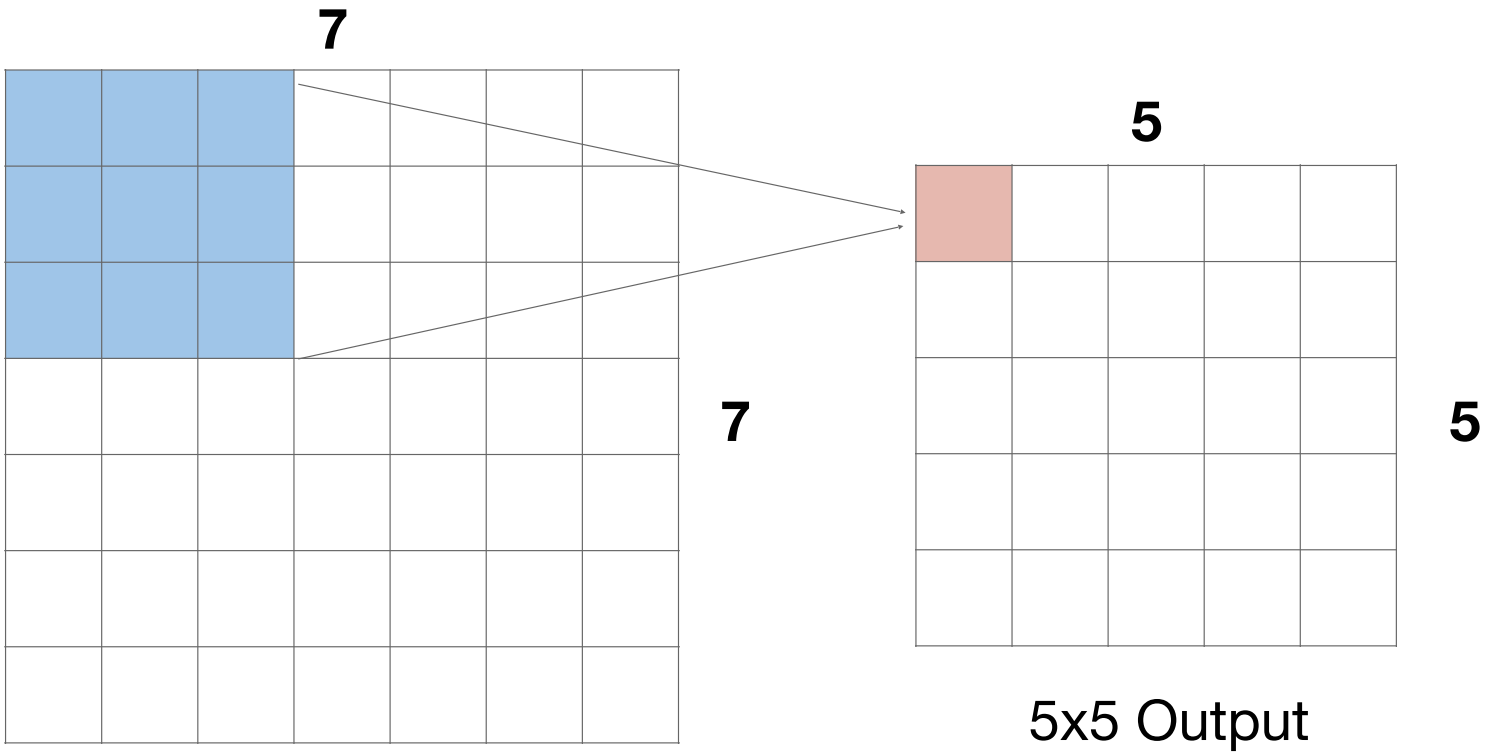
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

A closer look at spatial dimensions:

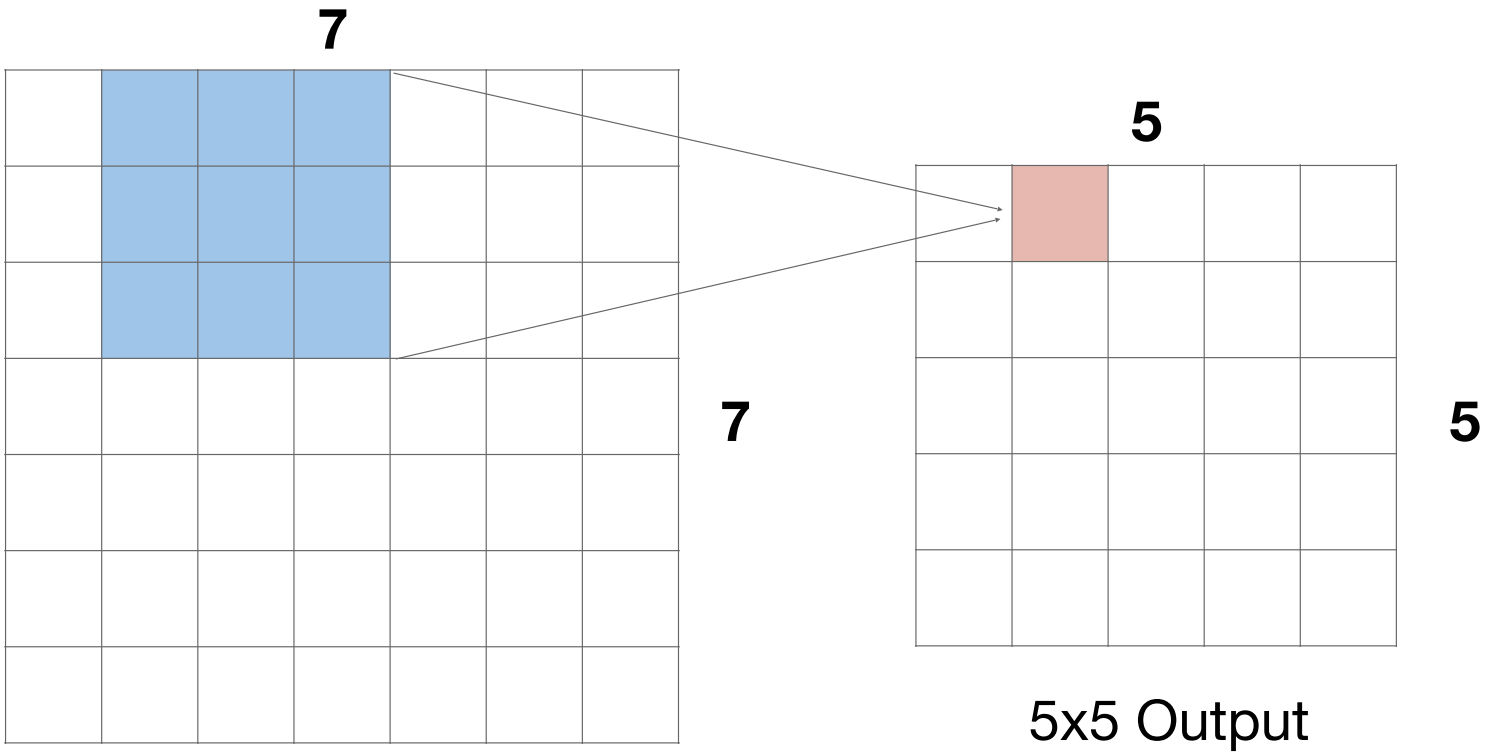


slide credit: Fei-Fei, Justin Johnson, Serena Yeung

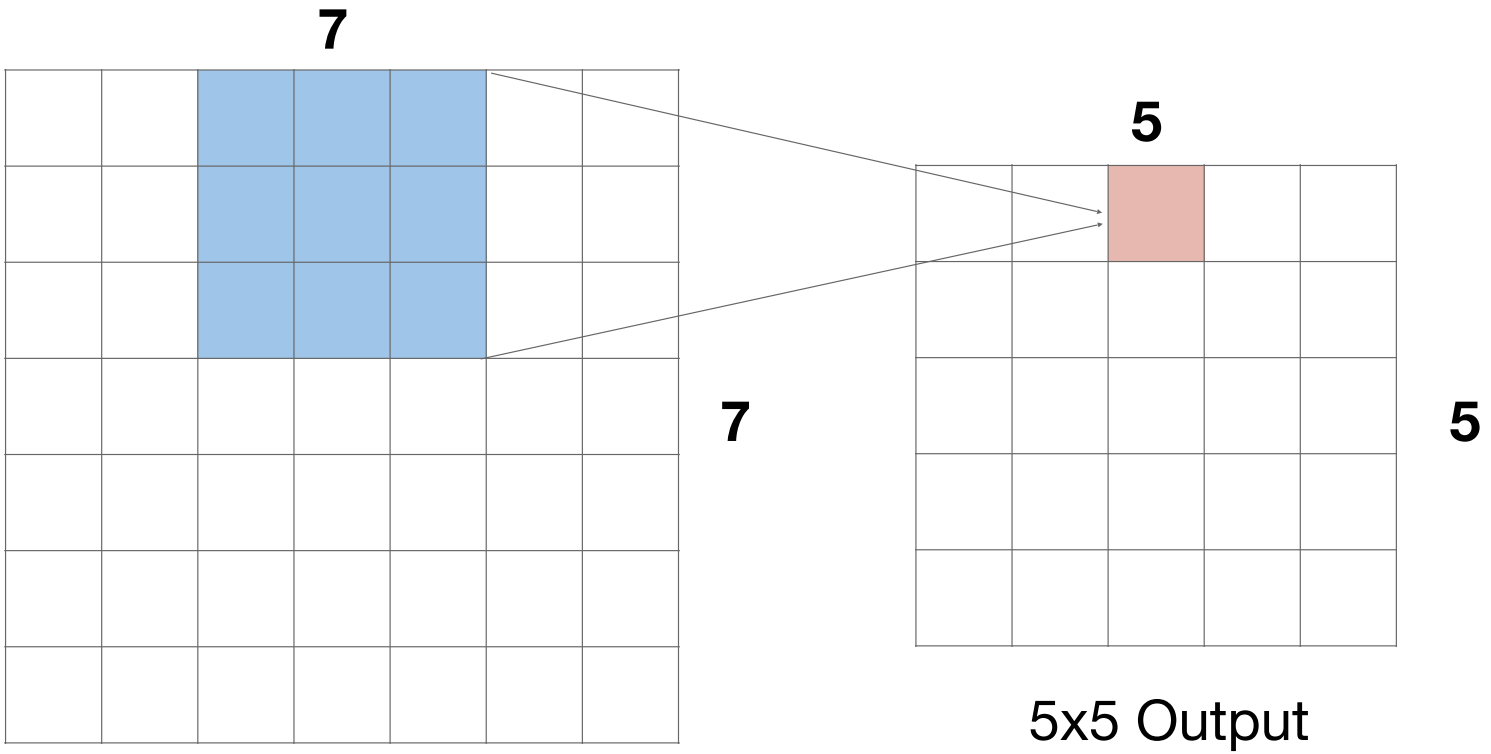
Closer look at spatial dimensions



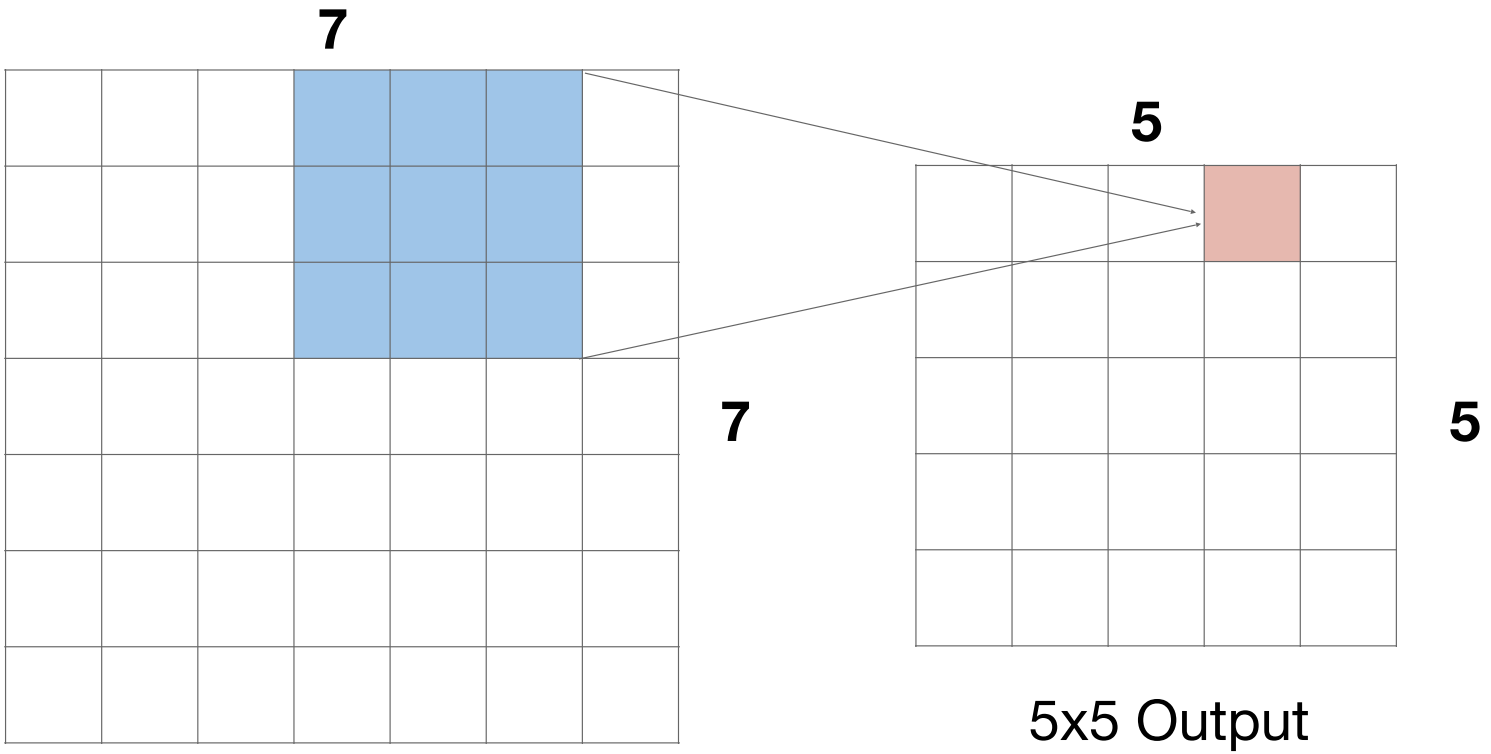
Closer look at spatial dimensions



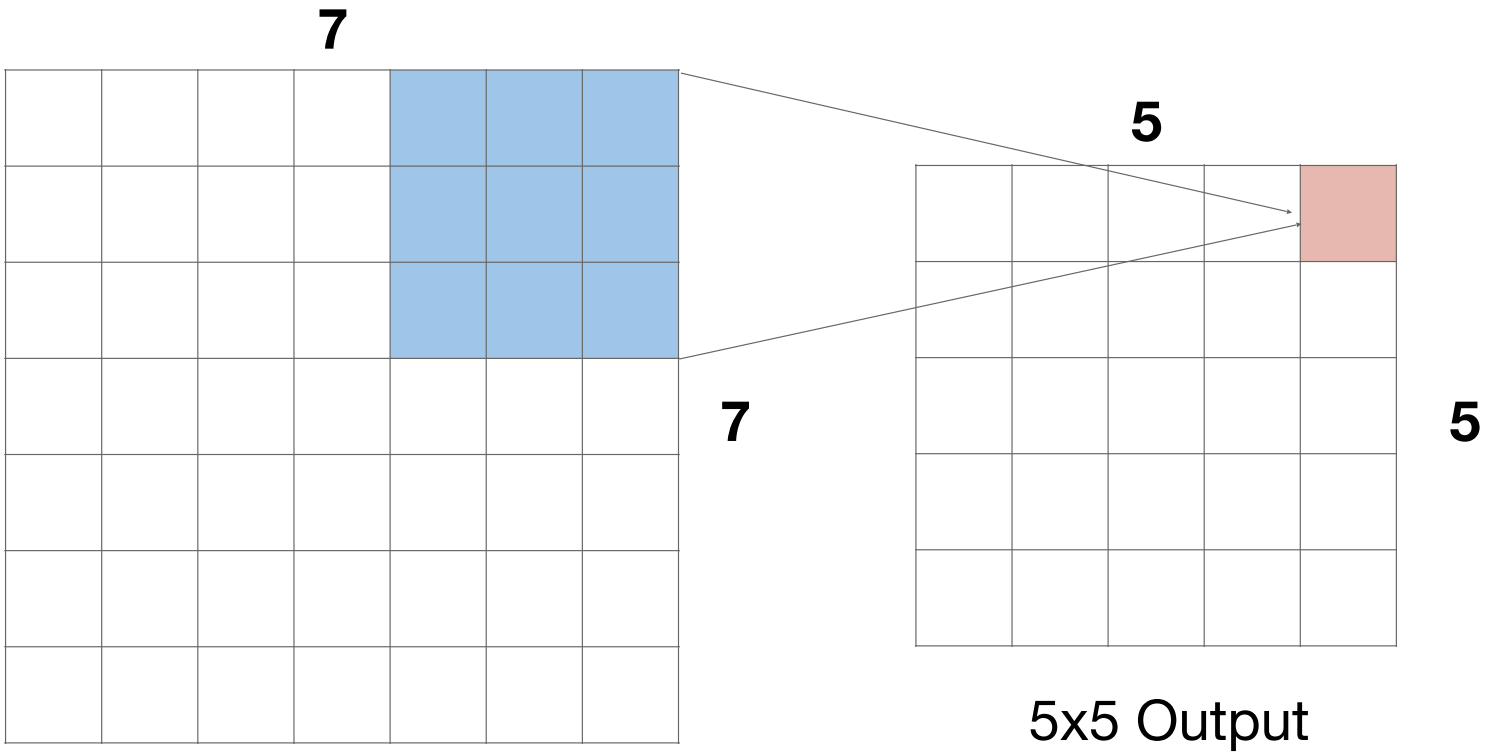
Closer look at spatial dimensions



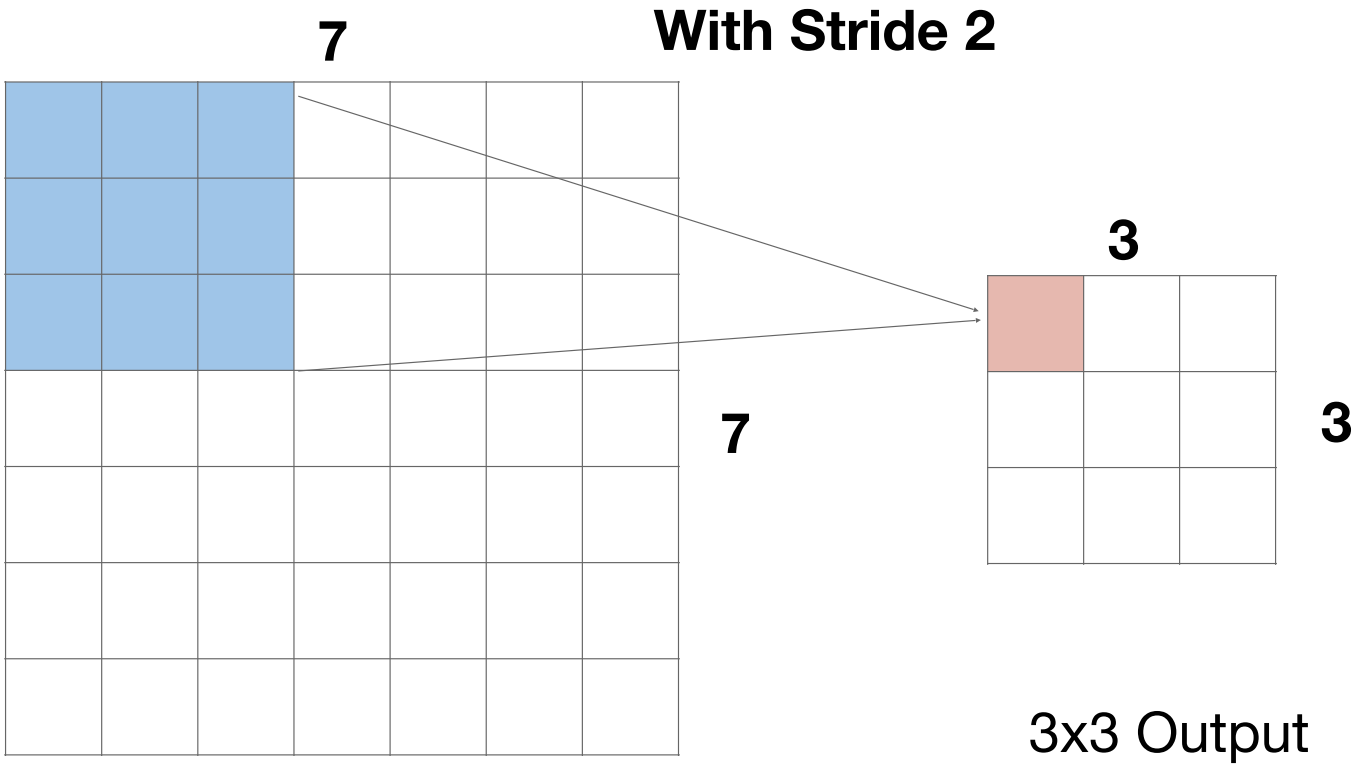
Closer look at spatial dimensions



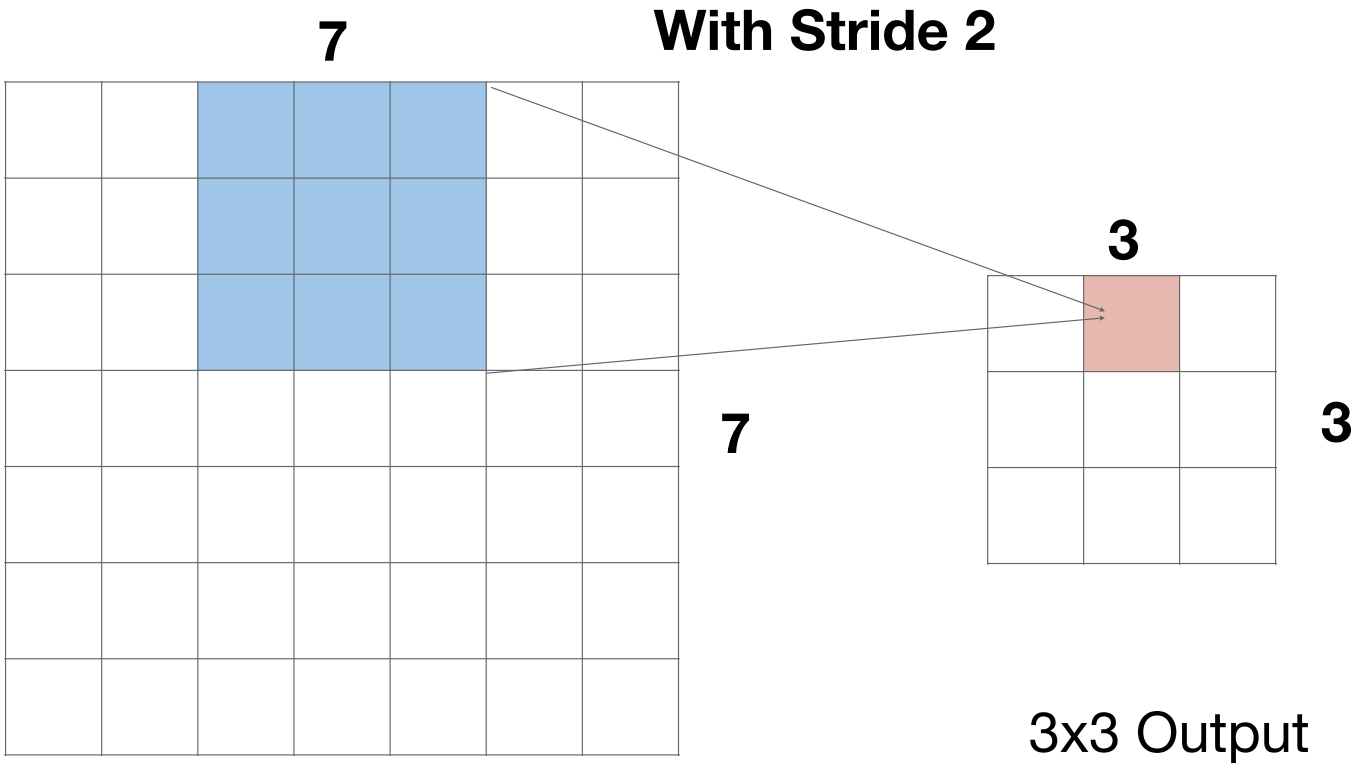
Closer look at spatial dimensions



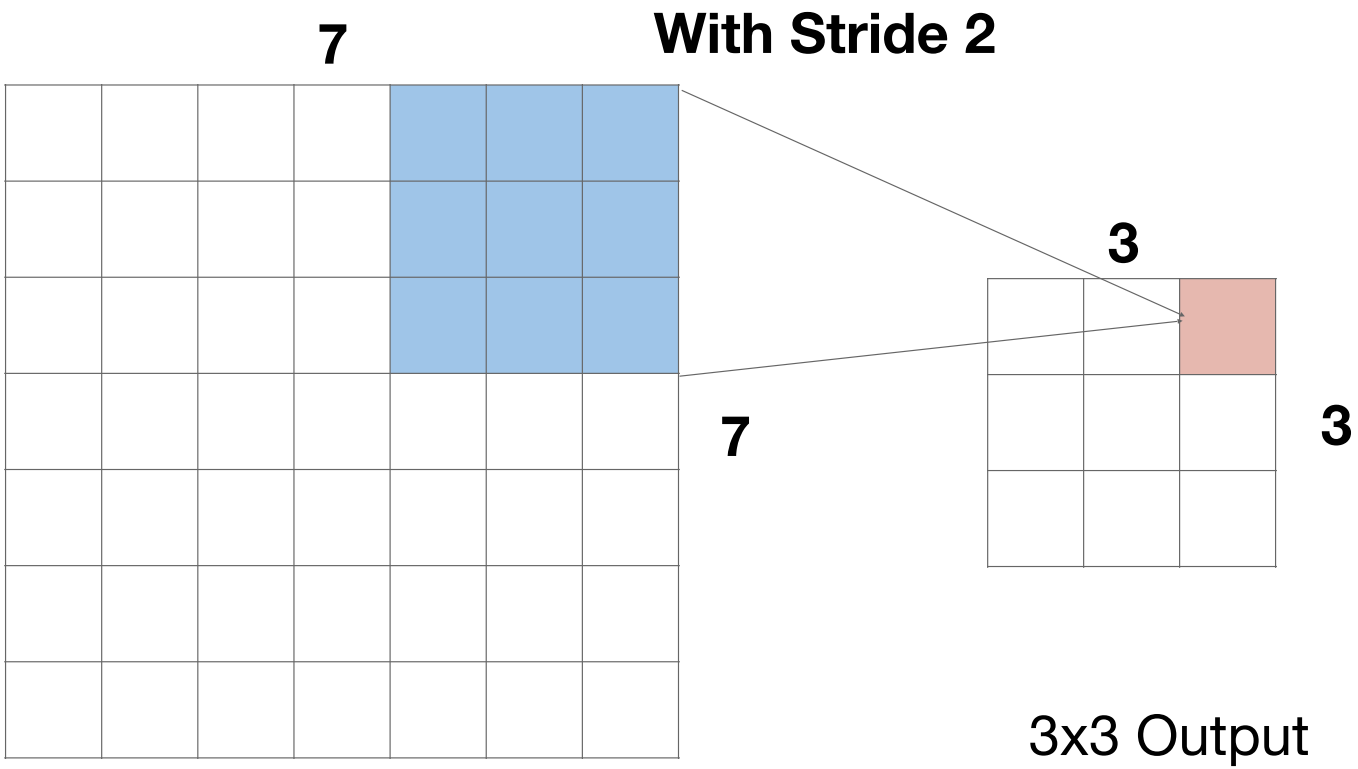
Closer look at spatial dimensions



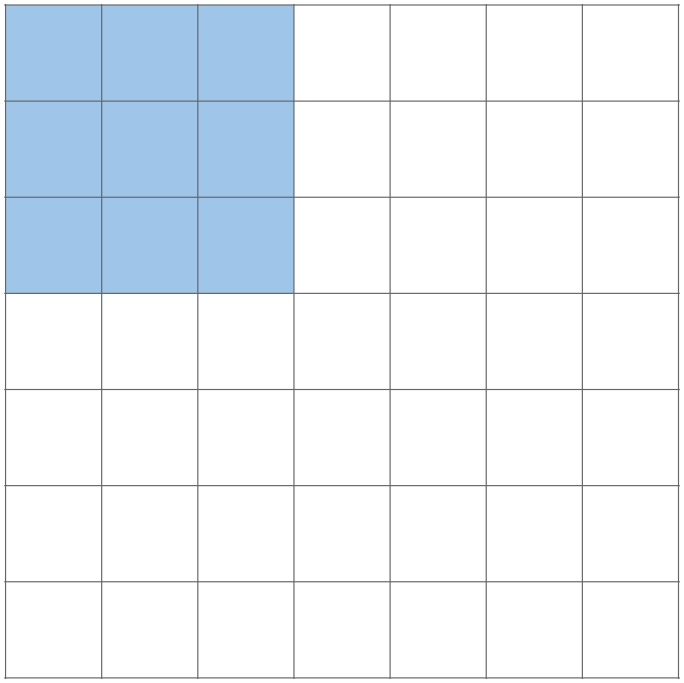
Closer look at spatial dimensions



Closer look at spatial dimensions



Closer look at spatial dimensions

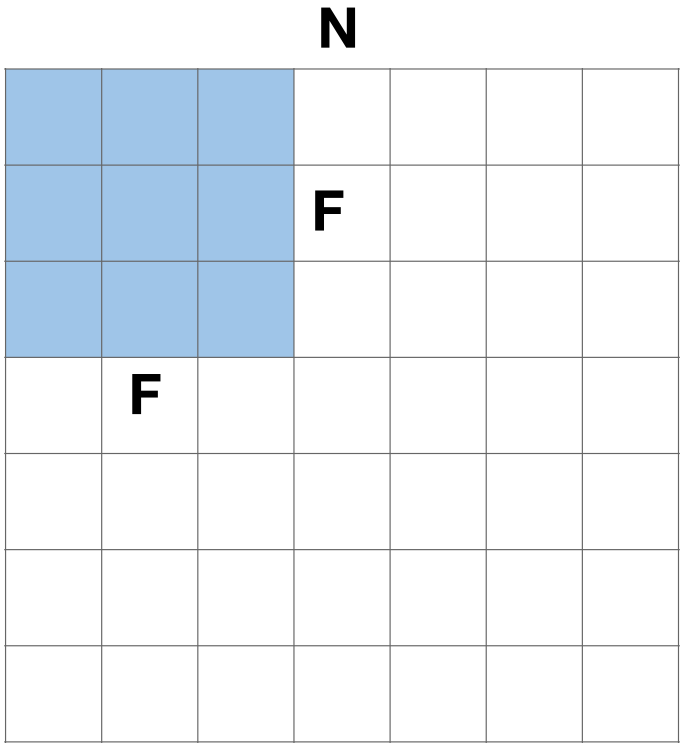


7x7 input (spatially)
assume 3x3 filter

With Stride 3 ?

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

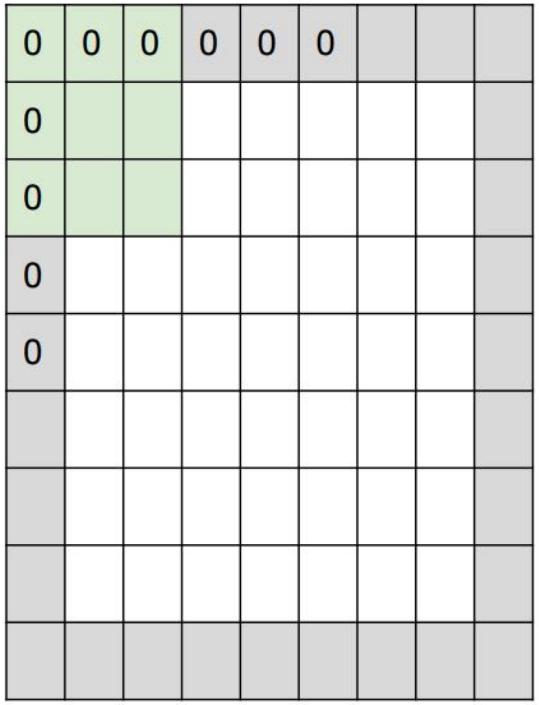
Output dimensions



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 :\

In Practice: Zero pad to preserve size

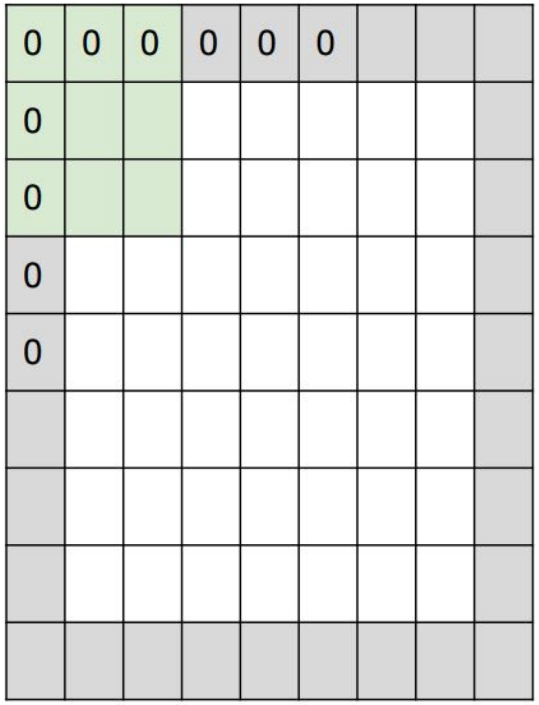


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

In Practice: Zero pad to preserve size

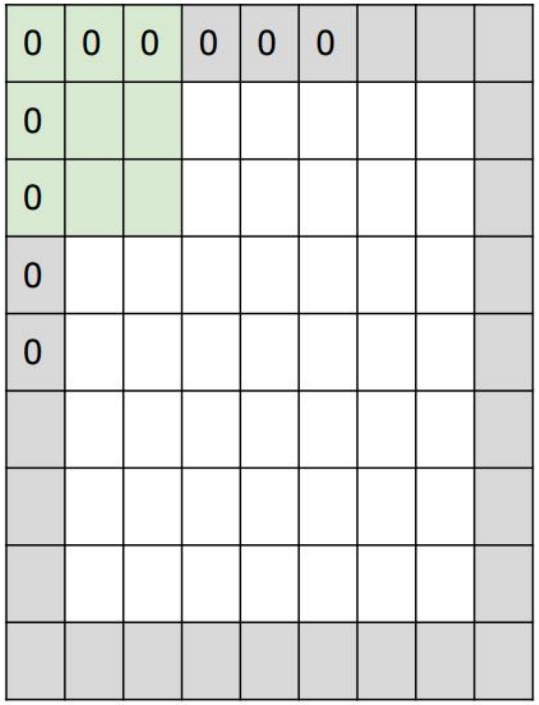


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

In Practice: Zero pad to preserve size



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

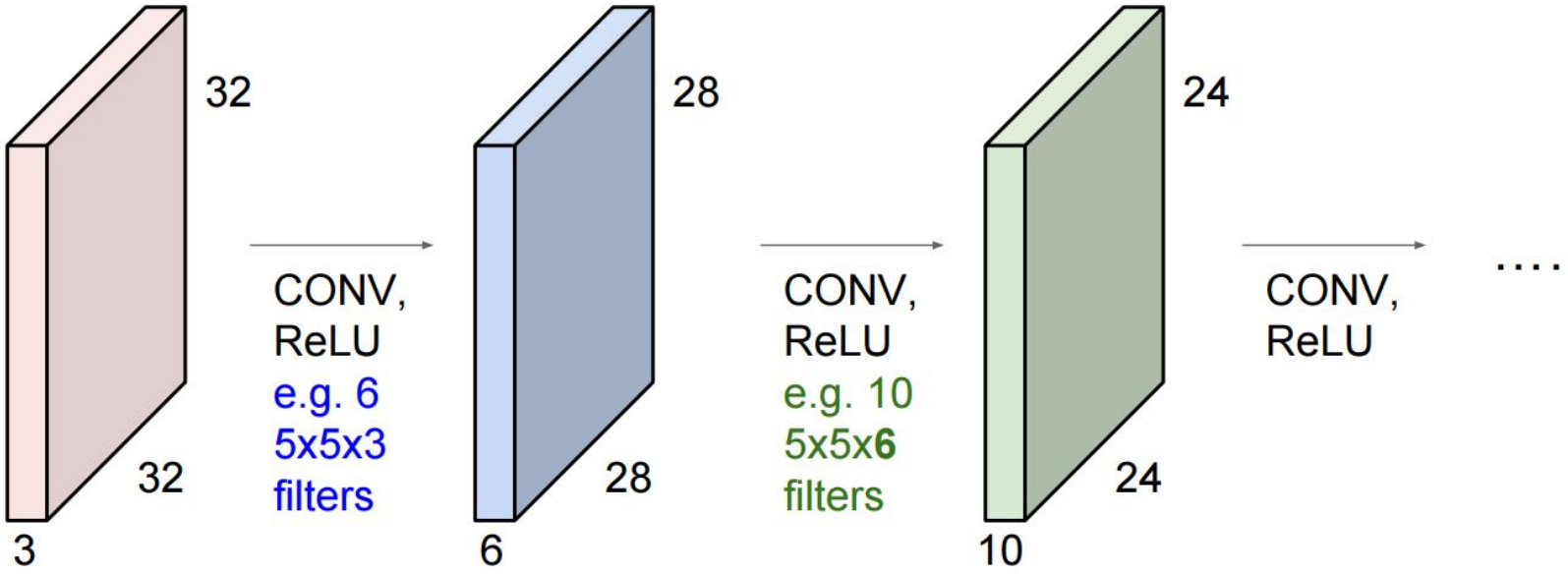
7x7 output!

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

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

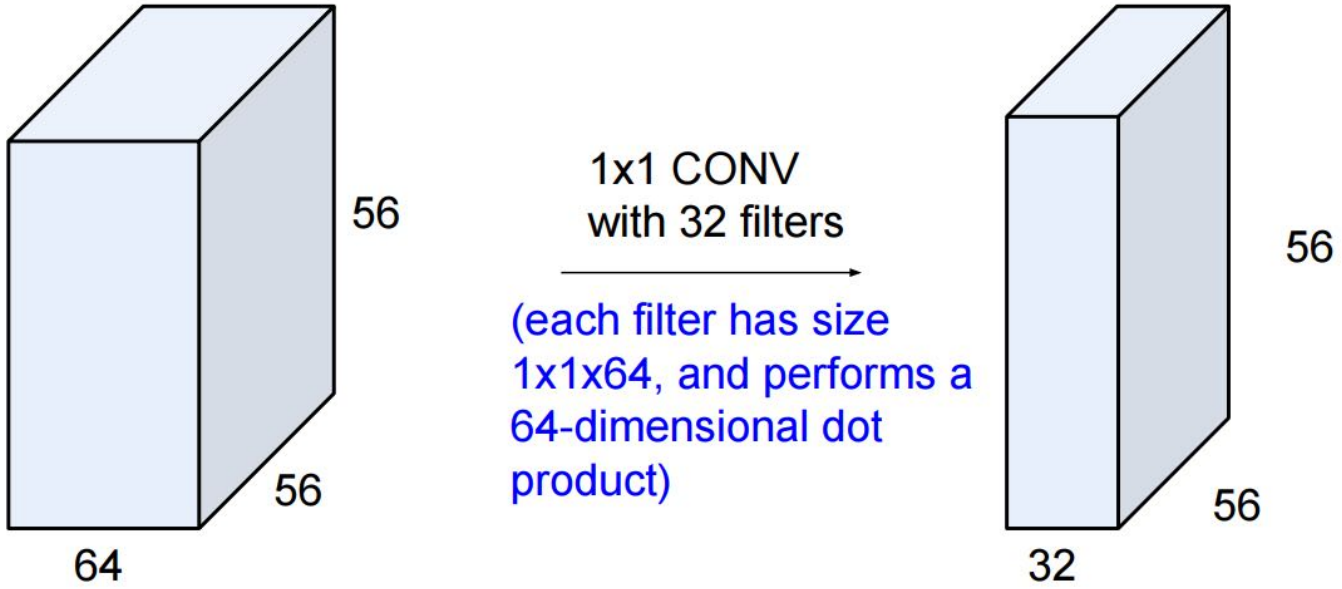
Zero pad to preserve size

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.



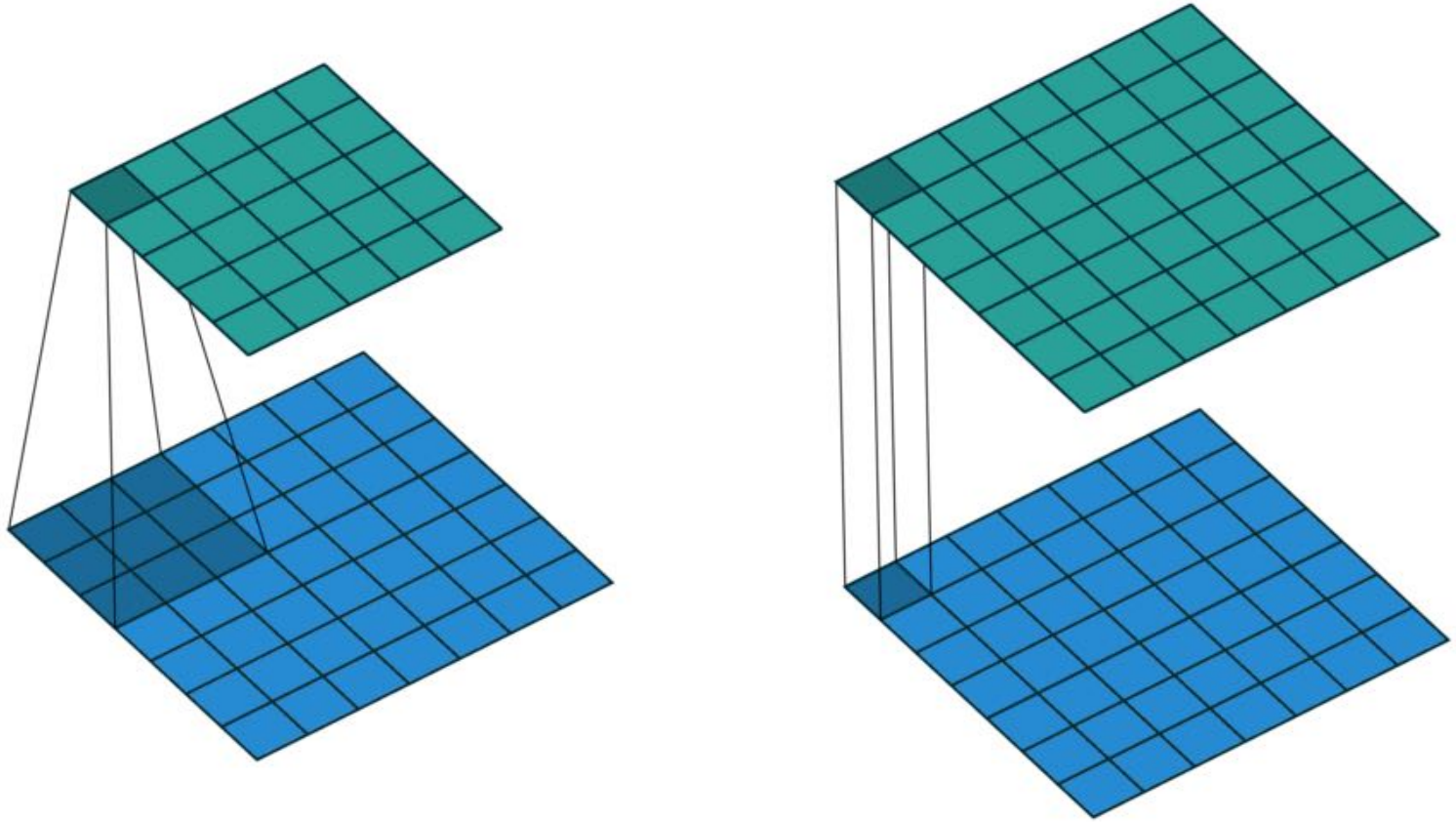
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

(btw, 1x1 convolution layers make perfect sense)



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

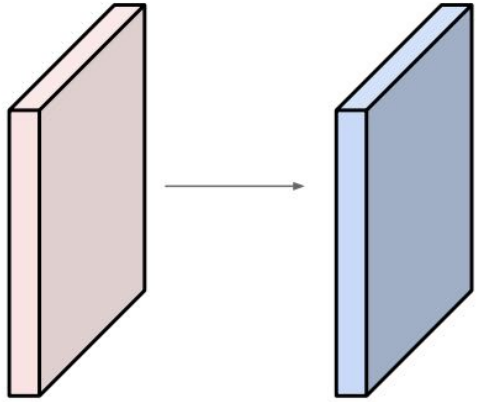
3x3 vs 1x1 convolutions



[1] Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Examples time

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



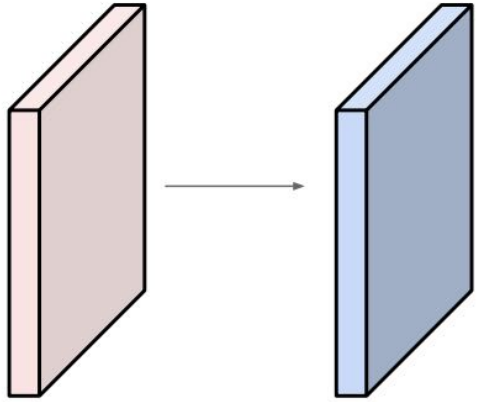
Output volume size: ?

(recall:)
(N - F) / stride + 1

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Examples time

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

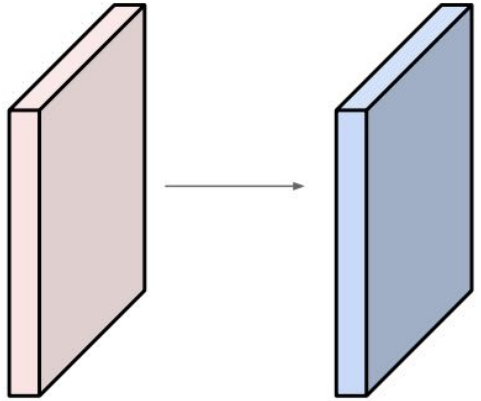


Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

(recall:)
(N - F) / stride + 1

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Examples time

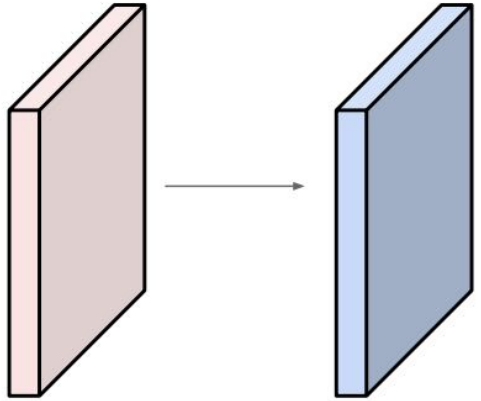


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

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**

FC layer of the same size = 32*32*3*10
= 30720 params

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

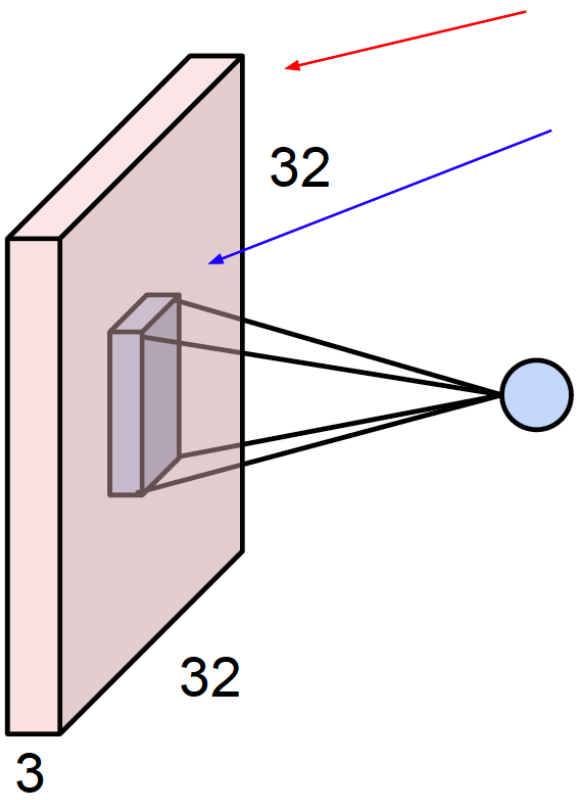
Common settings:

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

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

The brain/neuron view of CONV Layer



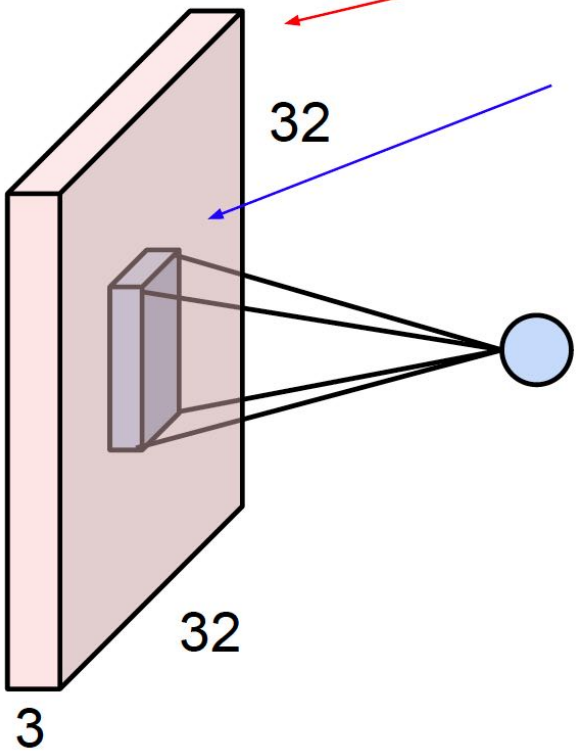
32x32x3 image

5x5x3 filter

1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

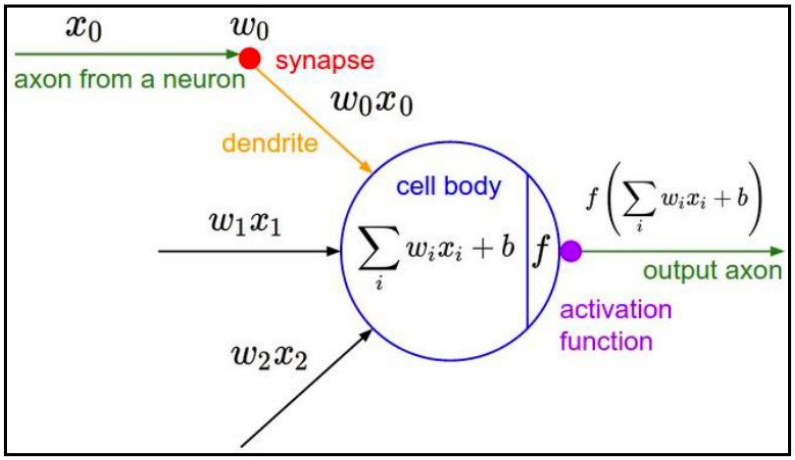
slide credit: Fei-Fei, Justin Johnson, Serena Yeung

The brain/neuron view of CONV Layer



32x32x3 image

5x5x3 filter

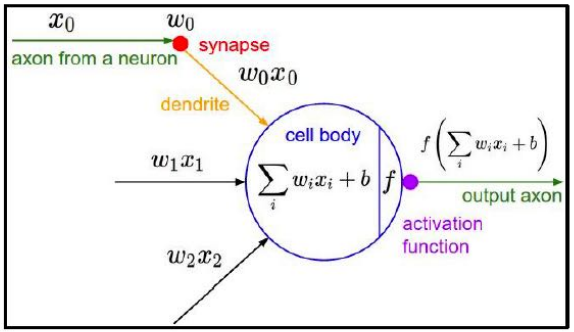


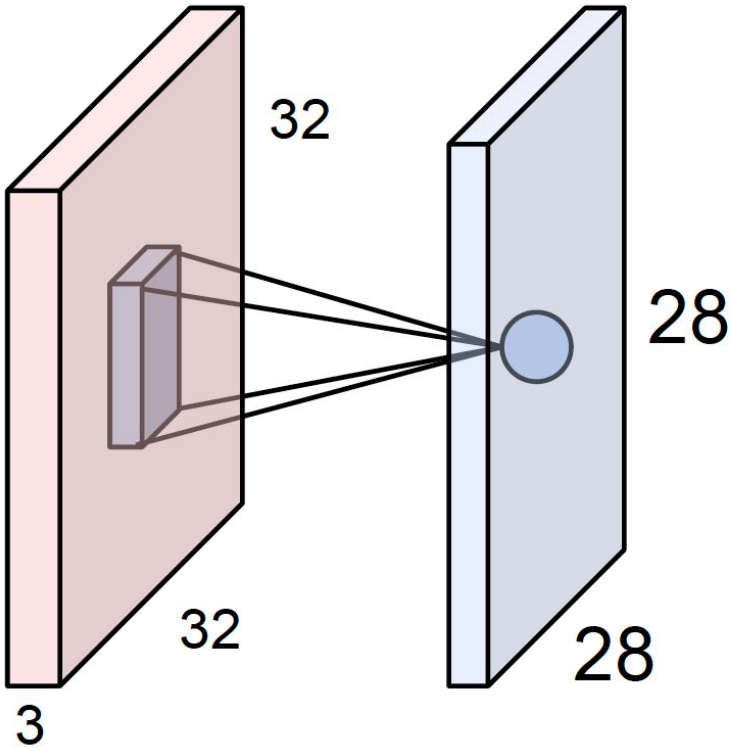
It's just a neuron with local connectivity...

1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

The brain/neuron view of CONV Layer





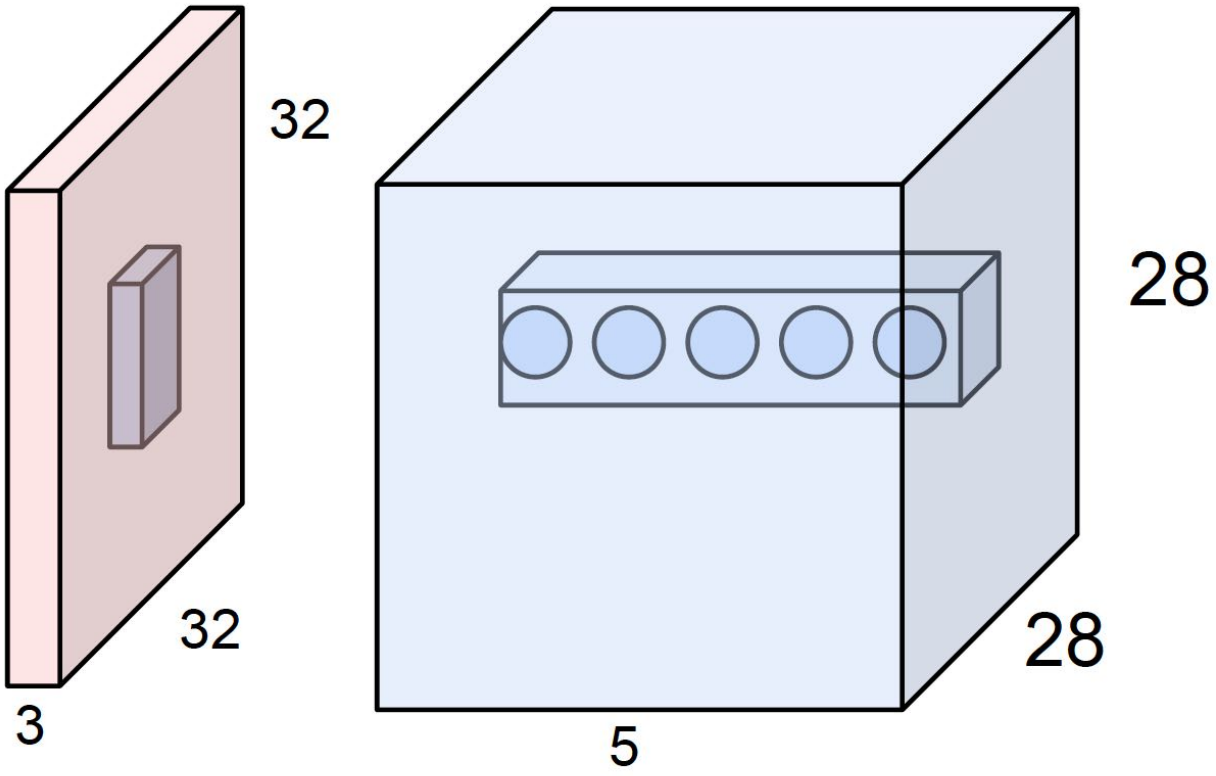
An activation map is a 28x28 sheet of neuron outputs:

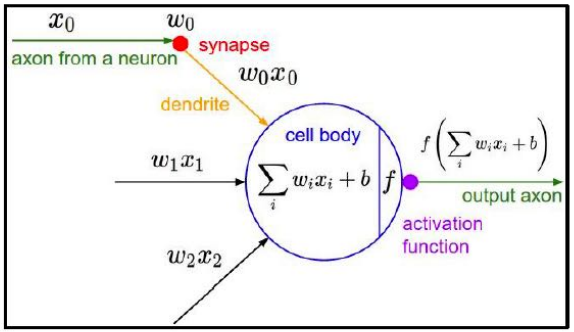
1. Each is connected to a small region in the input
2. All of them share parameters

"5x5 filter" -> "5x5 receptive field for each neuron"

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

The brain/neuron view of CONV Layer



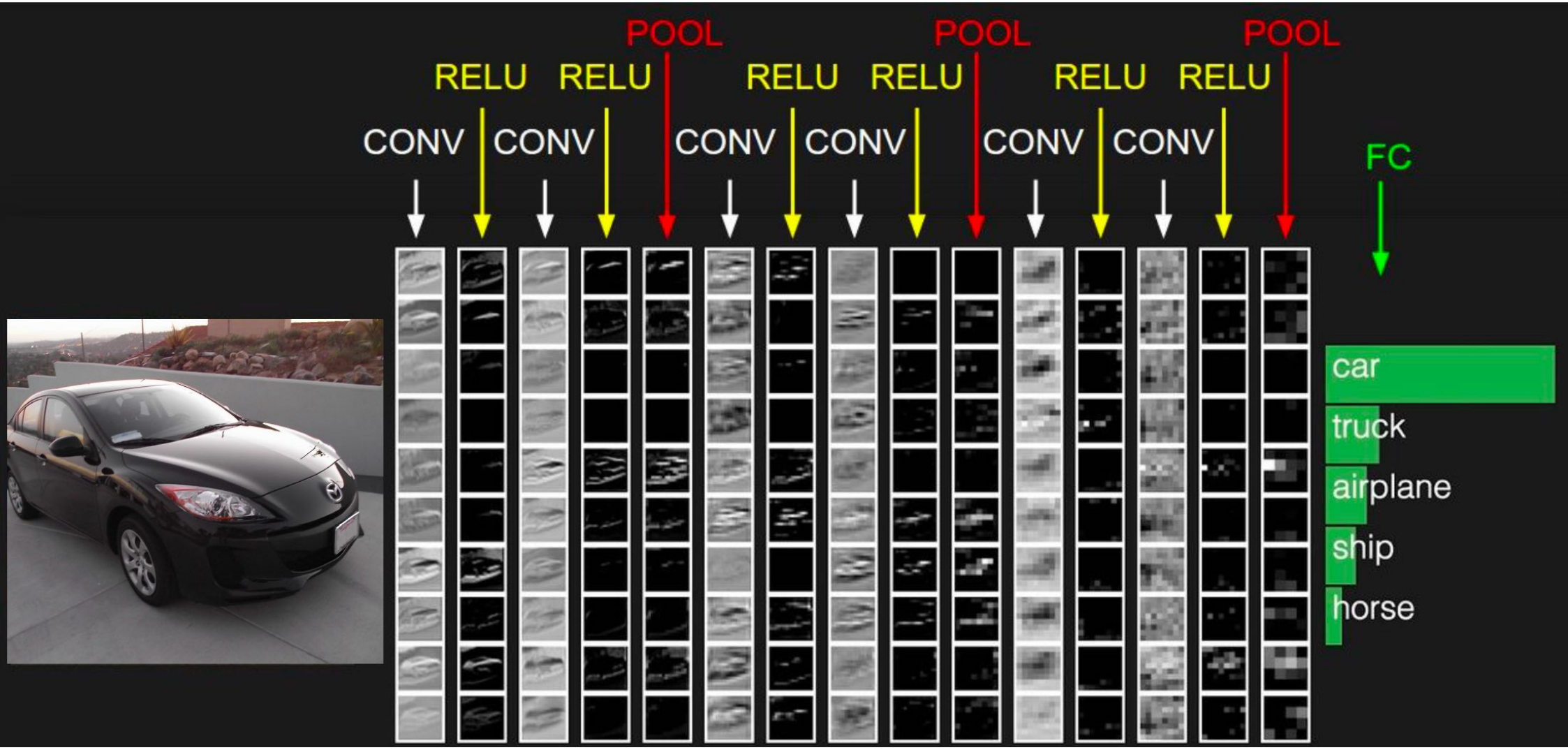


E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

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

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

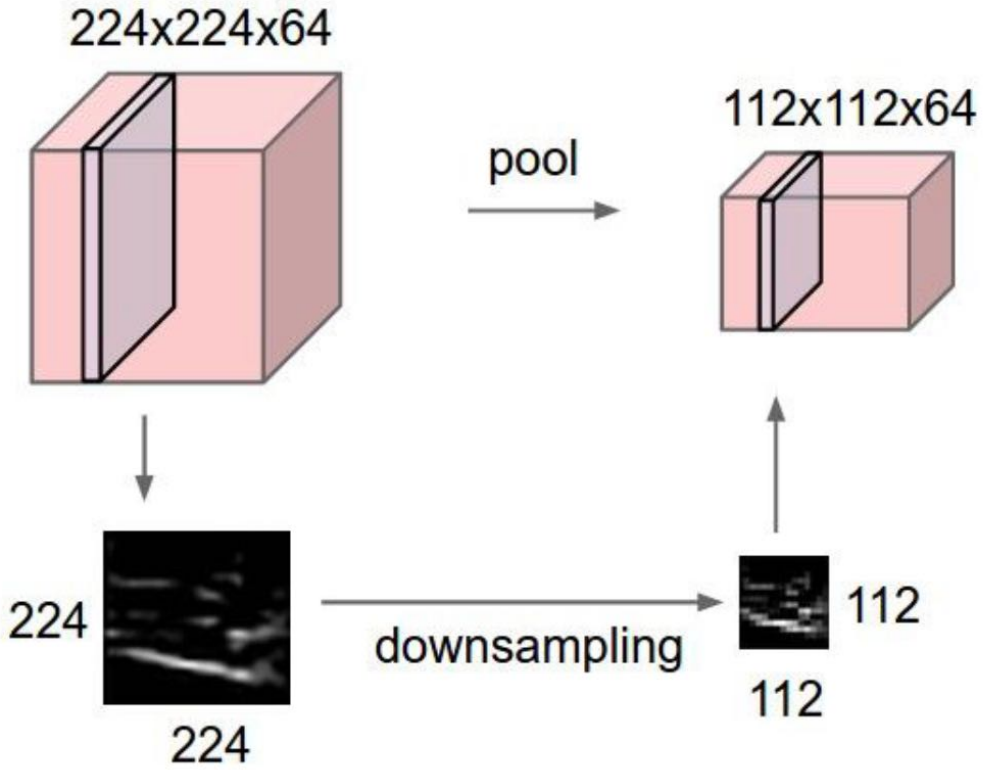
two more layers to go: POOL/FC



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

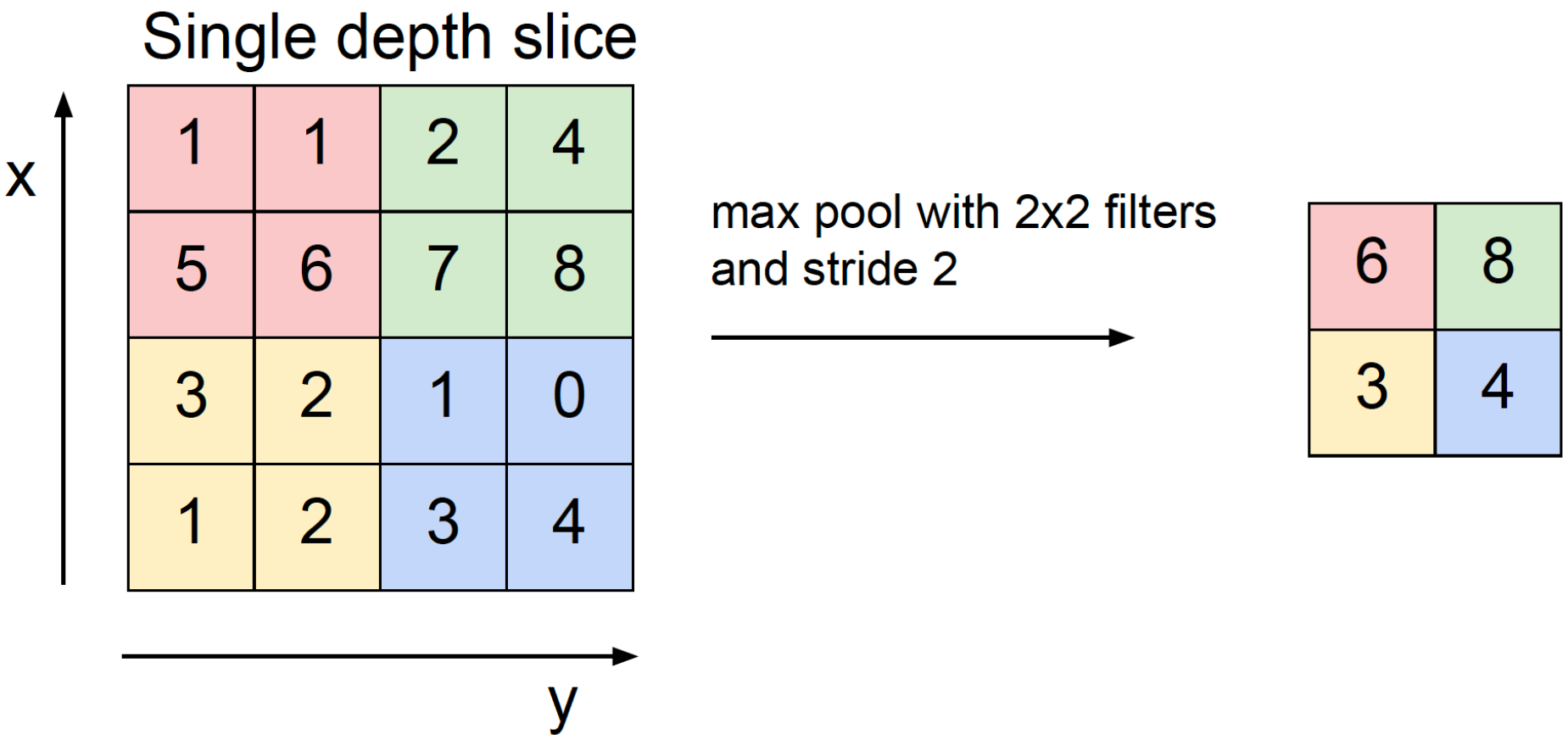
Pooling layer

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



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

MAX POOLING



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

(Max) Pooling Layer

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

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

(Max) Pooling Layer

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

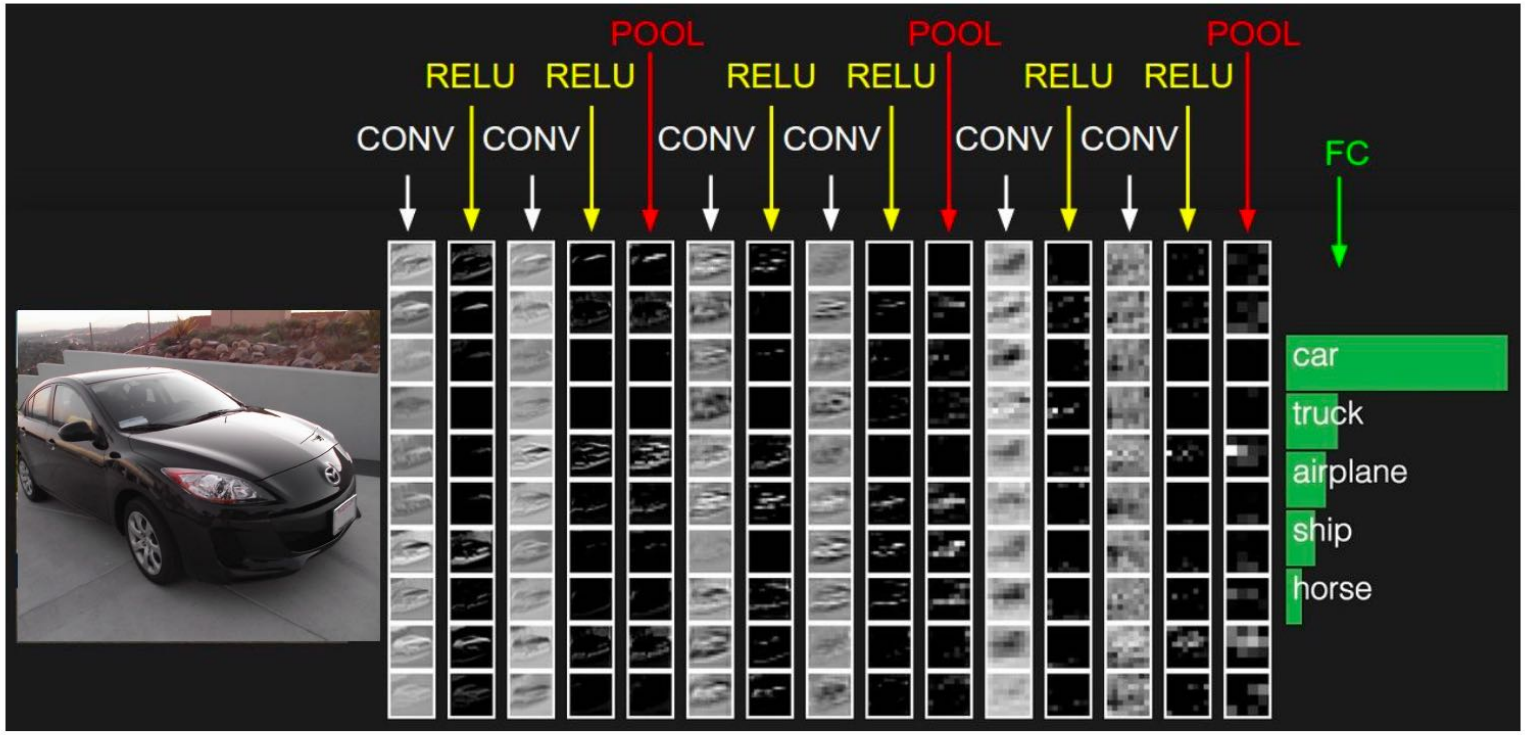
Common settings:

F = 2, S = 2
F = 3, S = 2

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



slide credit: Fei-Fei, Justin Johnson, Serena Yeung

Convolutional Neural Networks

- CNNs stack CONV, POOL, FC layers
  - Trend towards smaller filters and deeper architectures
  - Trend towards getting rid of POOL / FC layers (just CONV)

- Typical Architecture:
  - { (CONV+RELU) * N + POOL } * M + {FC+RELU} * K + SOFTMAX

  - softmax function:

$$\sigma(z_i) = \frac{e^{(-z_i)}}{\sum_{j=1}^{C} e^{(-z_j)}} = \frac{\exp(-z_i)}{\sum_{j=1}^{C} \exp(-z_j)}$$

slide credit: Fei-Fei, Justin Johnson, Serena Yeung

max planck institut
informatik

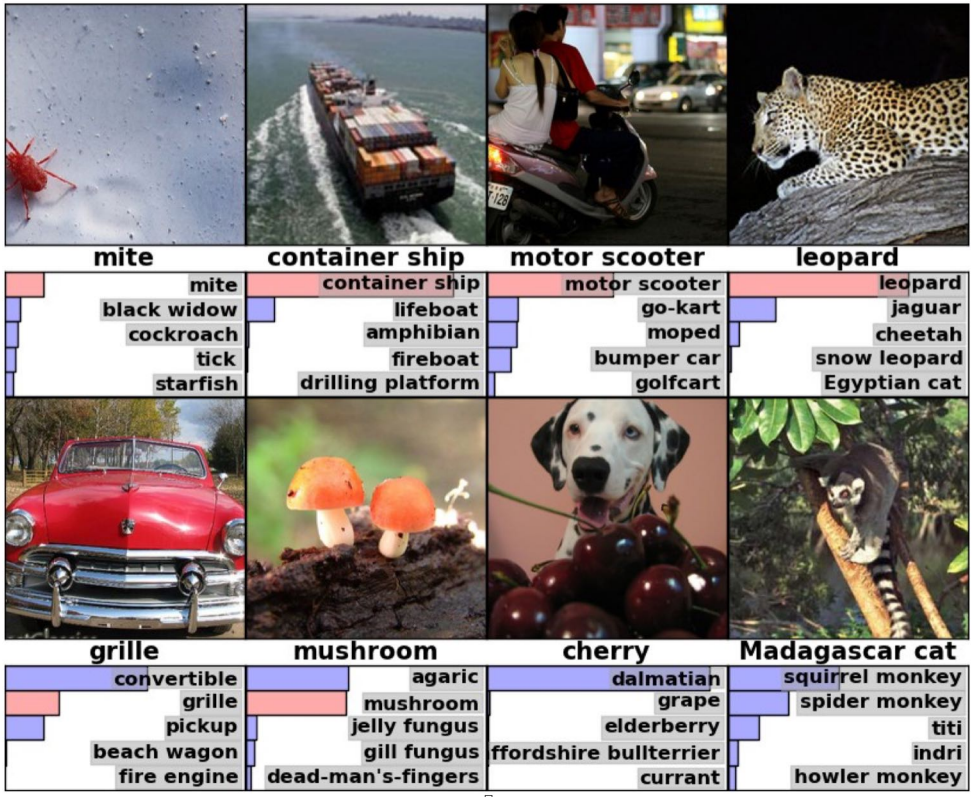
SIC Saarland Informatics Campus

Convolutional Neural Networks

Depth Matters !

Computer Vision: ImageNet Image Classification

- 1000 classes — around 1000 images for each class



slide credit: Fergus

Architecture of Krizhevsky et al. (Alexnet)

- 8 layers total (~61 million parameters)

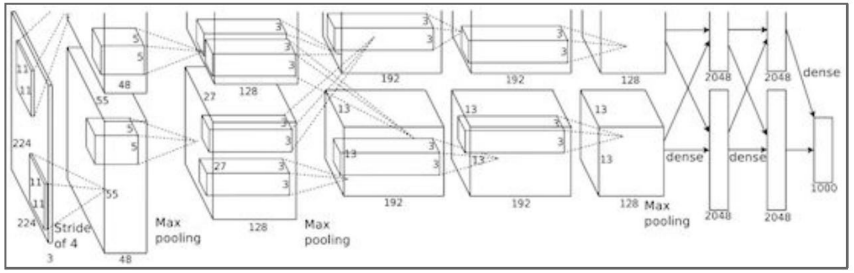
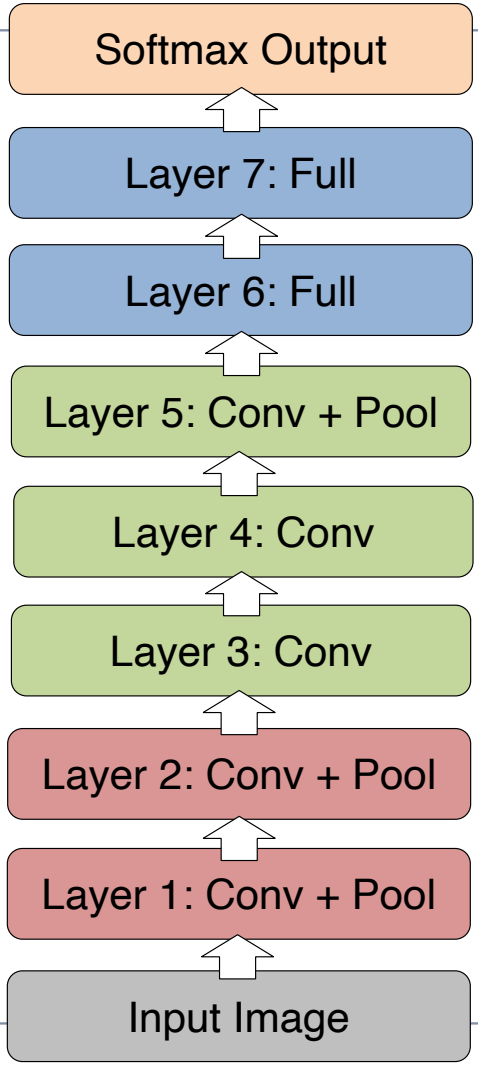


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"

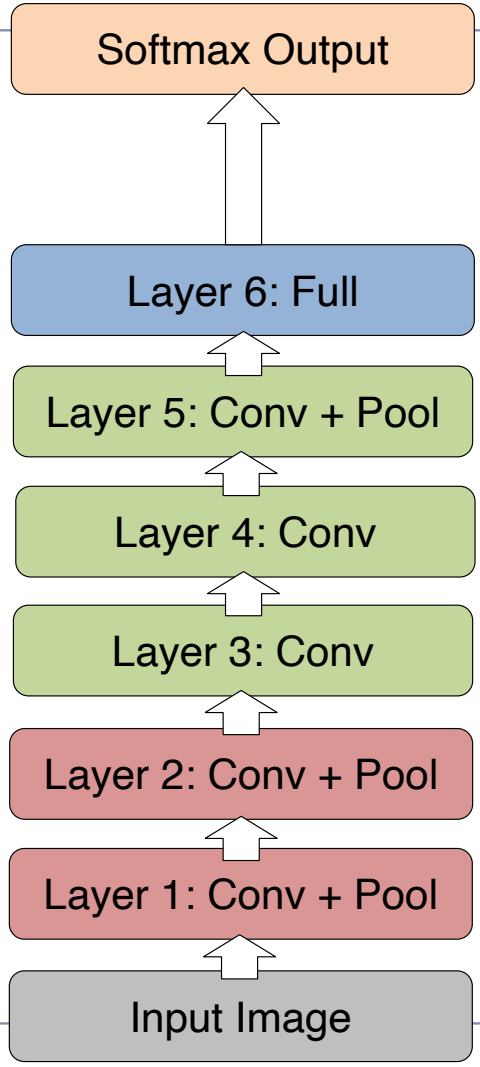
- Trained on Imagenet dataset [Deng et al. CVPR'09]
- 18.1% top-5 error



slide credit: Fergus

Architecture of Krizhevsky et al. (Alexnet)

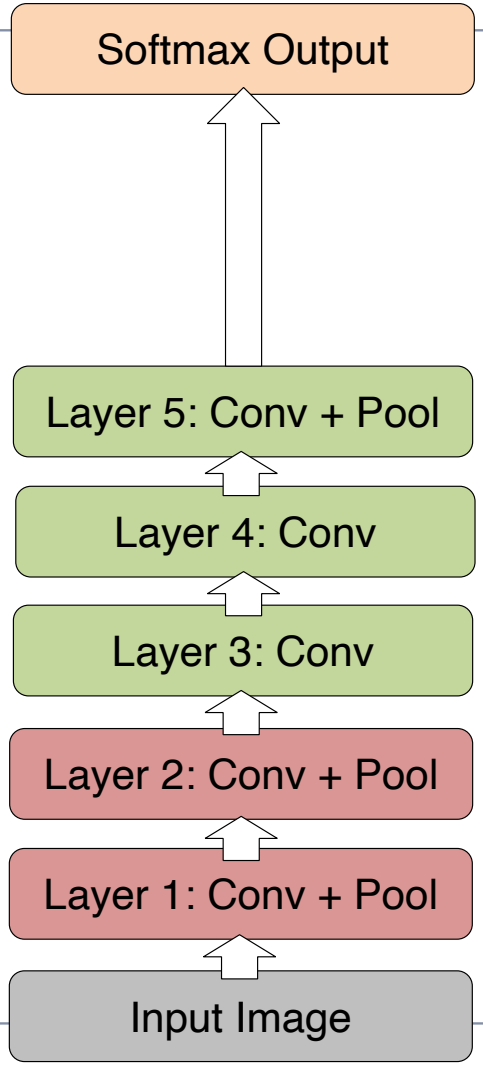
- Remove top fully connected layer
  - Layer 7
- Drop 16 million parameters (26%)
- Only 1.1% drop in performance!



slide credit: Fergus

Architecture of Krizhevsky et al. (Alexnet)

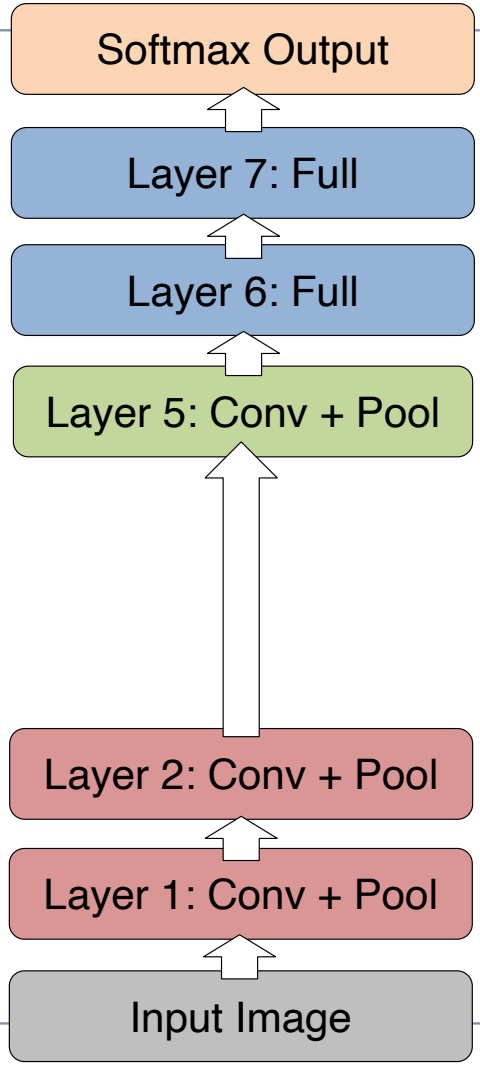
- Remove both fully connected layers
  - Layer 6 & 7
- Drop ~50 million parameters (82%)
- 5.7% drop in performance



slide credit: Fergus

Architecture of Krizhevsky et al. (Alexnet)

- Now try removing upper feature extractor layers:
  - Layers 3 & 4
- Drop ~1 million parameters (<2%)
- 3.0% drop in performance

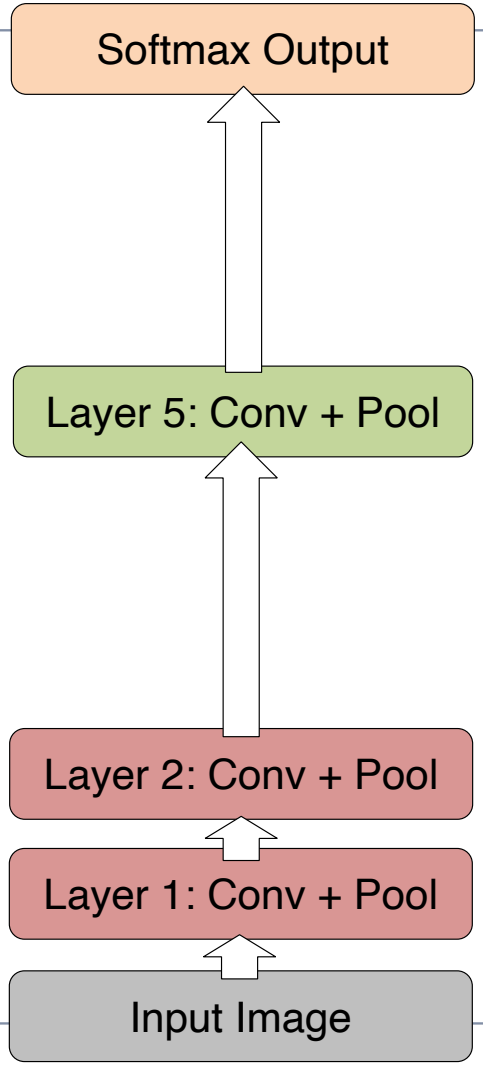


slide credit: Fergus

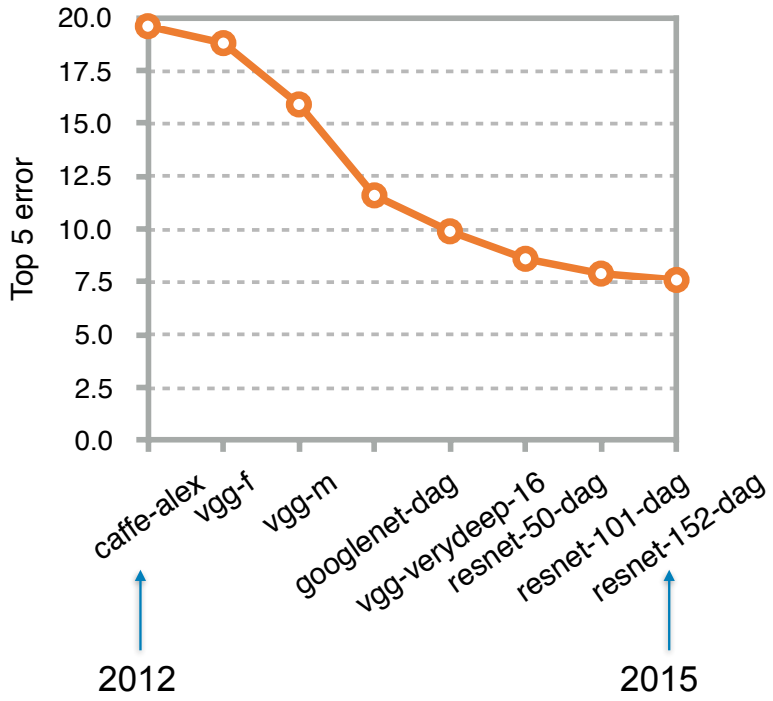
Architecture of Krizhevsky et al. (Alexnet)

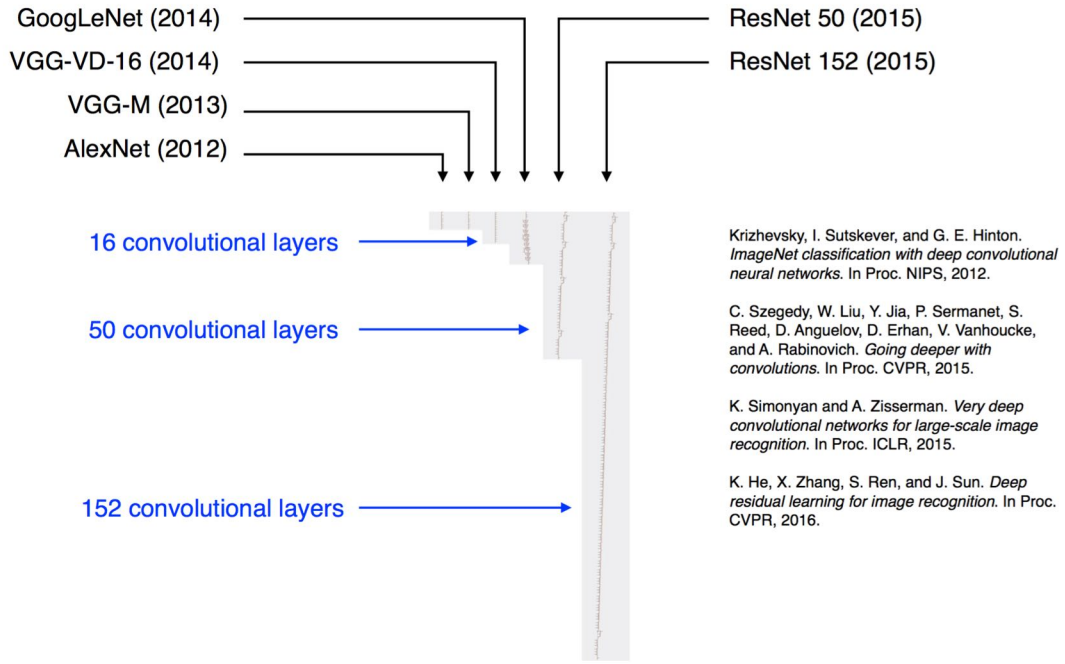
- Now try removing upper feature extractor layers & fully connected:
  - Layers 3, 4, 6 ,7
- Now only 4 layers
- 33.5% drop in performance

→Depth of network is key



Architectures get deeper





Krizhevsky, I. Sutskever, and G. E. Hinton. *ImageNet classification with deep convolutional neural networks.* In Proc. NIPS, 2012.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. *Going deeper with convolutions.* In Proc. CVPR, 2015.

K. Simonyan and A. Zisserman. *Very deep convolutional networks for large-scale image recognition.* In Proc. ICLR, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. *Deep residual learning for image recognition.* In Proc. CVPR, 2016.

~ 2.6x improvement in 3 years