CS5670: Computer Vision

Convolutional neural networks

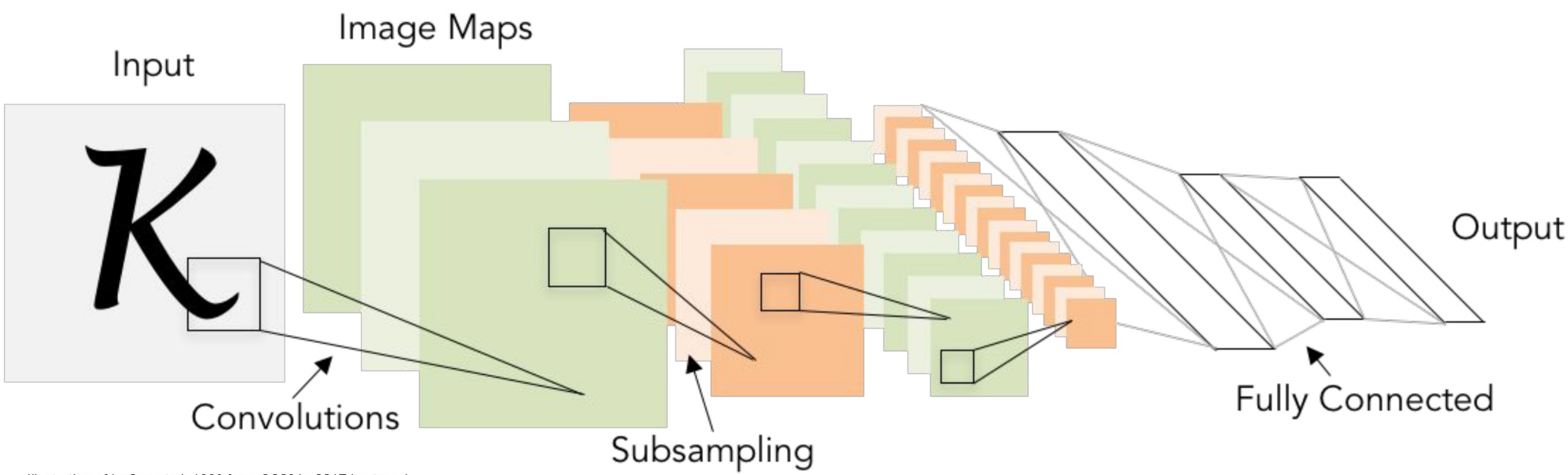


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Slides from Fei-Fei Li, Justin Johnson, Serena Yeung
http://vision.stanford.edu/teaching/cs231n/

Readings

- Neural networks
  - http://cs231n.github.io/neural-networks-1/
  - http://cs231n.github.io/neural-networks-2/
  - http://cs231n.github.io/neural-networks-3/
  - http://cs231n.github.io/neural-networks-case-study/

- Convolutional neural networks
  - http://cs231n.github.io/convolutional-networks/

Recap: Image Classification – a core task in computer vision

- Assume given set of discrete labels, e.g.
  {cat, dog, cow, apple, tomato, truck, … }

f() = “apple”

f() = “tomato”

f() = “cow”

Dataset: ETH-80, by B. Leibe Slide credit: L. Lazebnik

Recap: linear classification

- What we have: a score function and loss function
  - Score function maps an input data instance (e.g., an image) to a vector of scores, one for each category
  - Last time, our score function is based on linear classifier

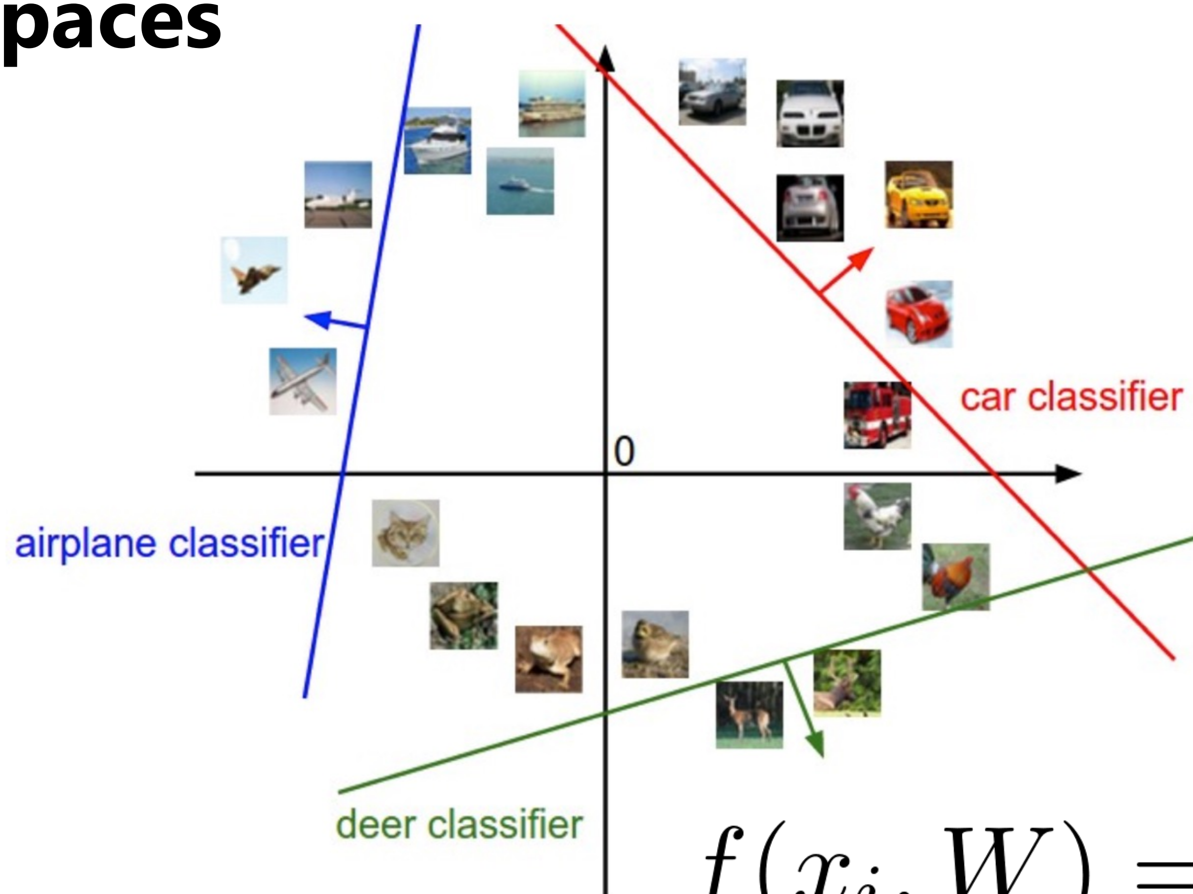
$$f(x, W) = Wx + b$$

f: score function
x: input instance
W, b: parameters of a linear (actually affine) function

- Find **W** and **b** that minimize a *loss* over labeled training data, e.g. cross-entropy loss

$$L = \frac{1}{N} \sum_i -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

Linear classifiers separate features space into half-spaces



$$f(x_i, W) = Wx_i + b$$

Figure credit: Fei-Fei Li and Andrej Karpathy

Neural networks

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

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

Neural networks

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

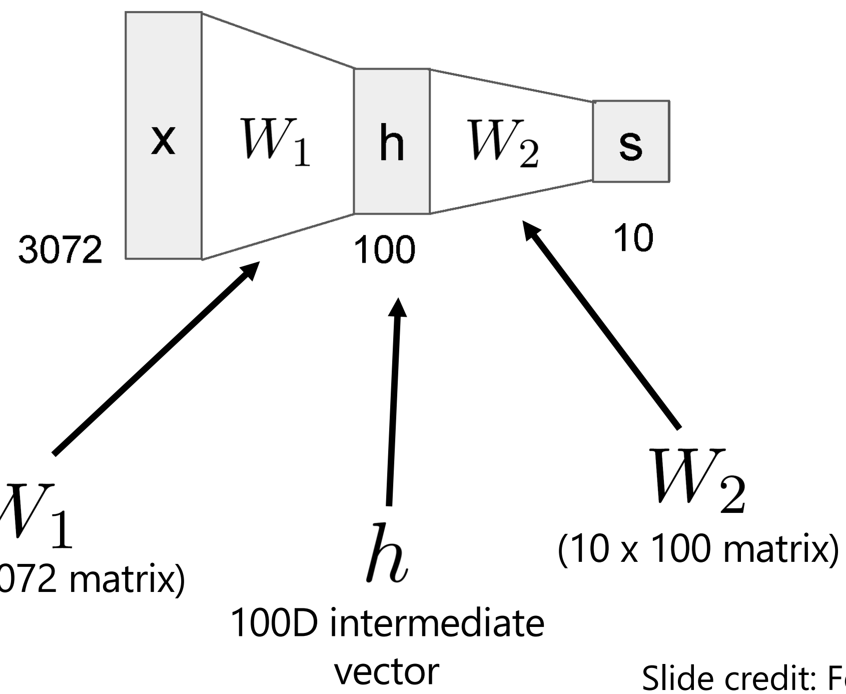
(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

Neural networks

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

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

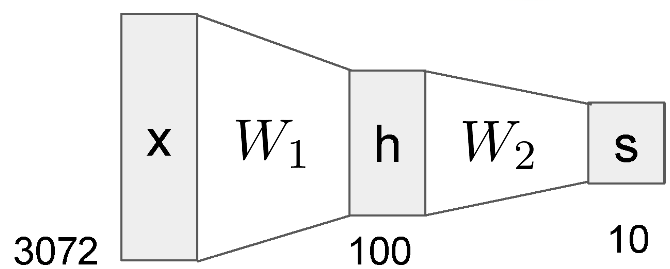


Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

Neural networks

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

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



- Total number of weights to learn:

  3,072 x 100 + 100 x 10 = 308,200

Adapted from Fei-Fei Li & Andrej Karpathy & Serena Leu

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

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

also called "Multilayer Perceptrons" (MLPs)

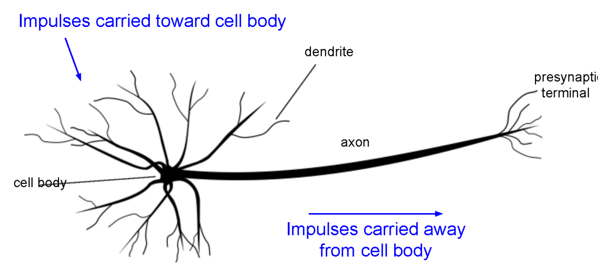
Adapted from Fei-Fei Li & Andrej Karpathy & Serena Leu

Neural networks

- Very coarse generalization of neural networks:
  - Linear functions chained together and separated by non-linearities (*activation functions*), e.g. "max"

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

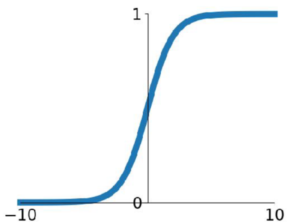
  - Why separate linear functions with non-linear functions?
  - *Very roughly* inspired by real neurons



Activation functions

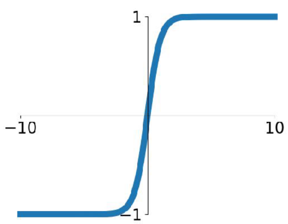
Sigmoid

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



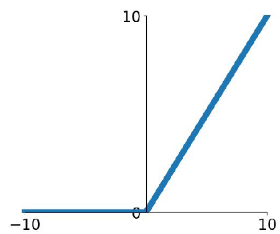
tanh

$\tanh(x)$



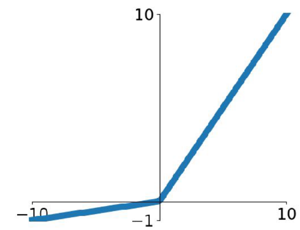
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

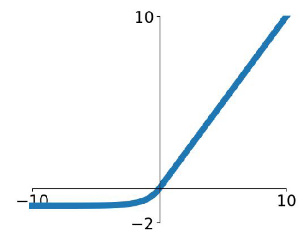


Maxout

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

ELU

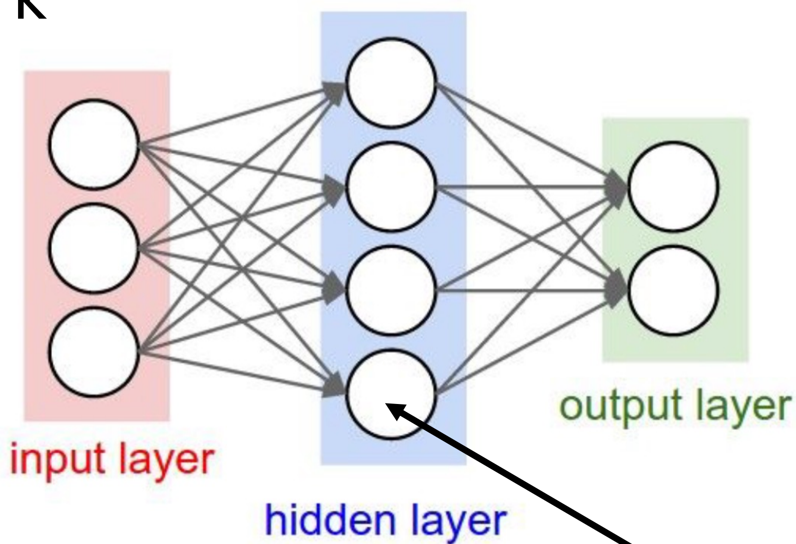
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

Neural network architecture

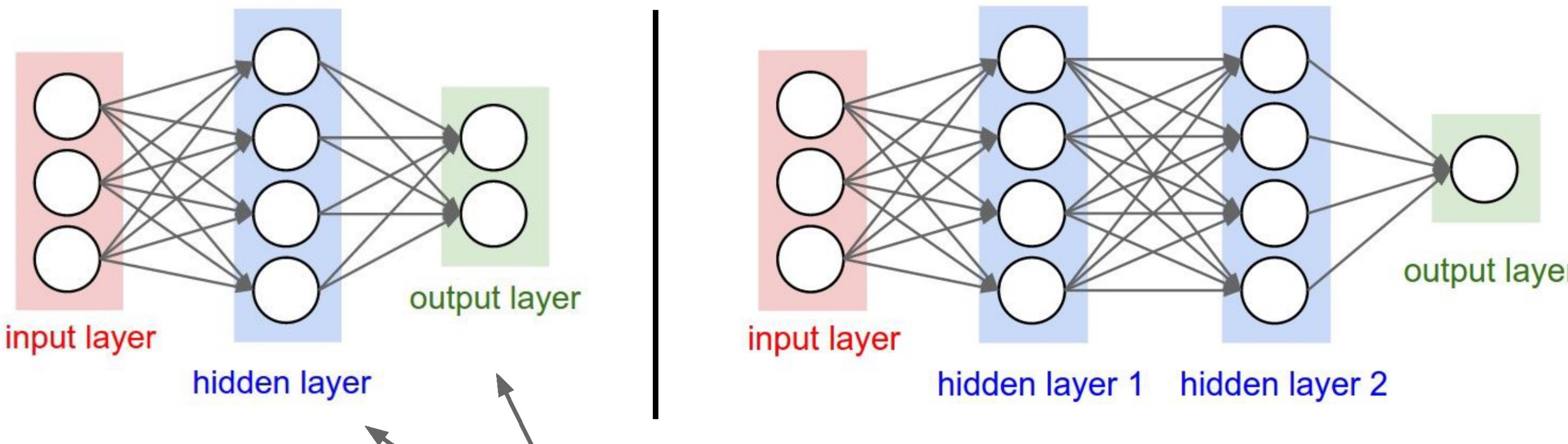
- Computation graph for a 2-layer neural network



Neuron or *unit*

Adapted from Fei-Fei Li & Andrej Karpathy & Serena Leu

Neural networks: Architectures



"2-layer Neural Net", or
"1-hidden-layer Neural Net"

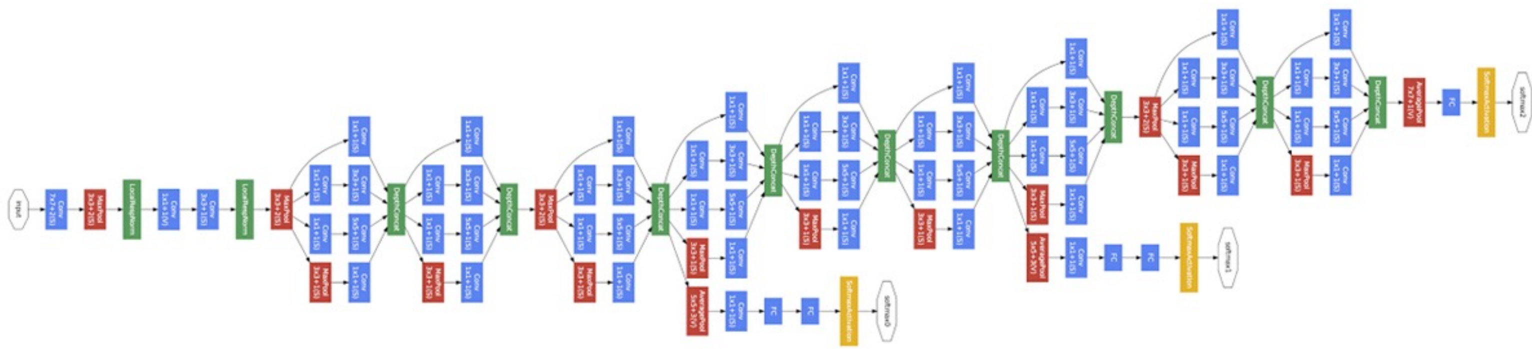
"Fully-connected" layers

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

- **Deep** networks typically have many layers and potentially millions of parameters

Adapted from Fei-Fei Li & Andrej Karpathy & Serena Leu

Deep neural network

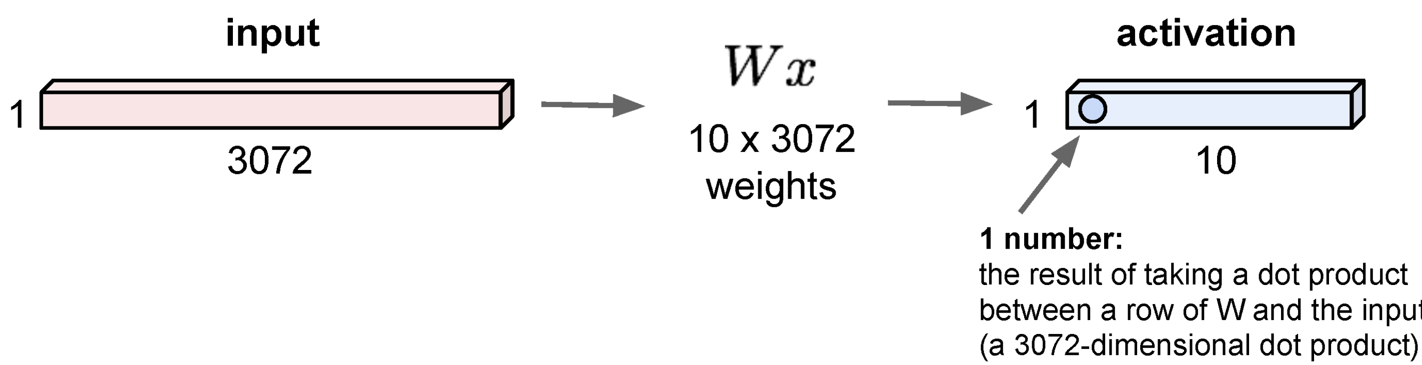


- *Inception* network (Szegedy et al, 2015)
- 22 layers

Adapted from Fei-Fei Li & Andrej Karpathy & Serena Leu

Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



- Just like a linear classifer – but in this case, just one layer of a larger *network*

Adapted from Fei-Fei Li & Andrej Karpathy & Serena Leu

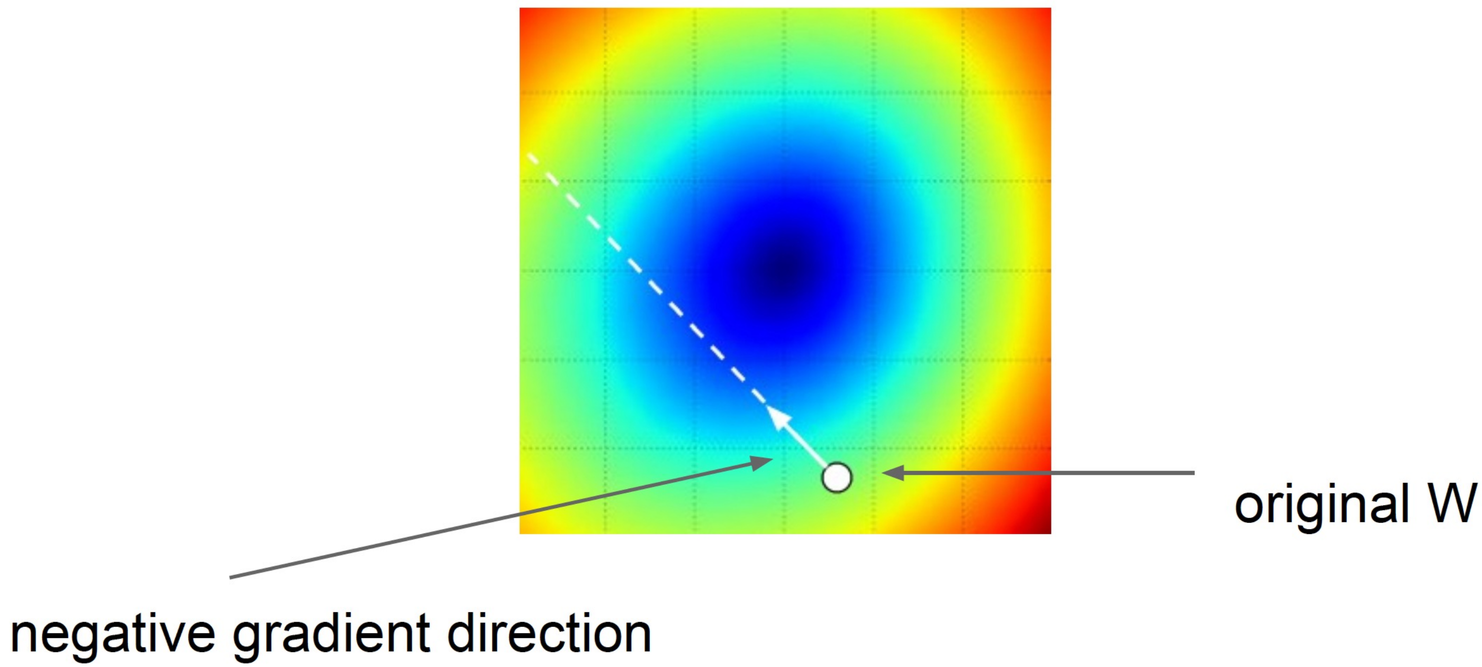
Summary so far

- A classic neural network arranges neurons into fully-connected layers
- The **layer** abstraction enables efficient implementations of neural networks using vectorized operations like matrix multiplication

Adapted from Fei-Fei Li & Andrej Karpathy & Serena Leu

Optimizing parameters with gradient descent

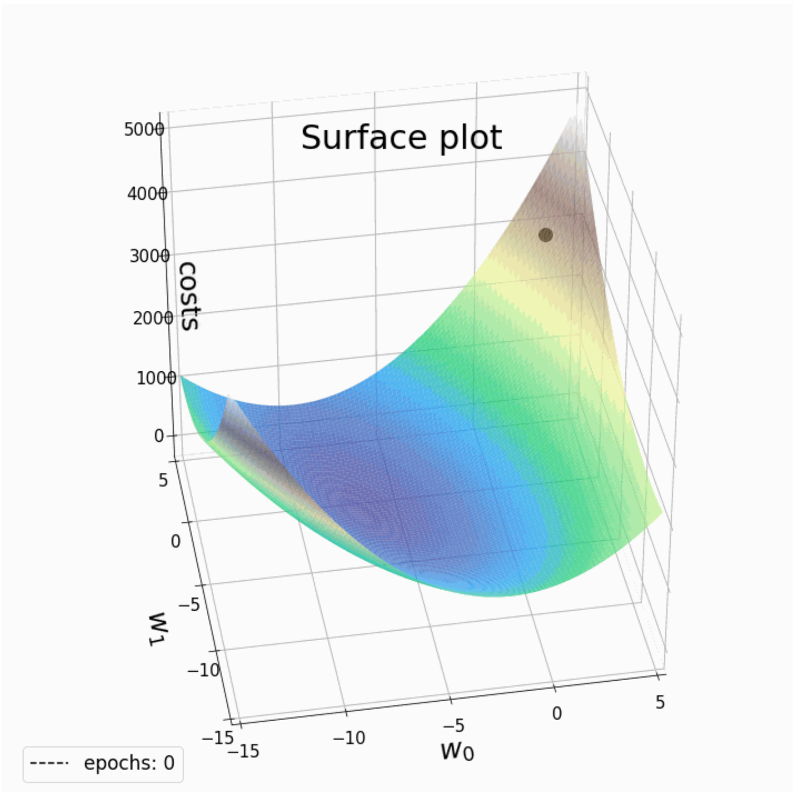
- How do we find the best **W** and **b** parameters?
- In general: *gradient descent*
  1. Start with a guess of a good **W** and **b** (or randomly initialize them)
  2. Compute the loss function for this initial guess and the *gradient* of the loss function
  3. Step some distance in the negative gradient direction (direction of steepest descent)
  4. Repeat steps 2 & 3
- Note: efficiently performing step 2 for deep networks is called *backpropagation*



Gradient descent: walk in the direction opposite gradient

- **Q**: How far?
- **A**: Step size: *learning rate*
- Too big: will miss the minimum
- Too small: slow convergence

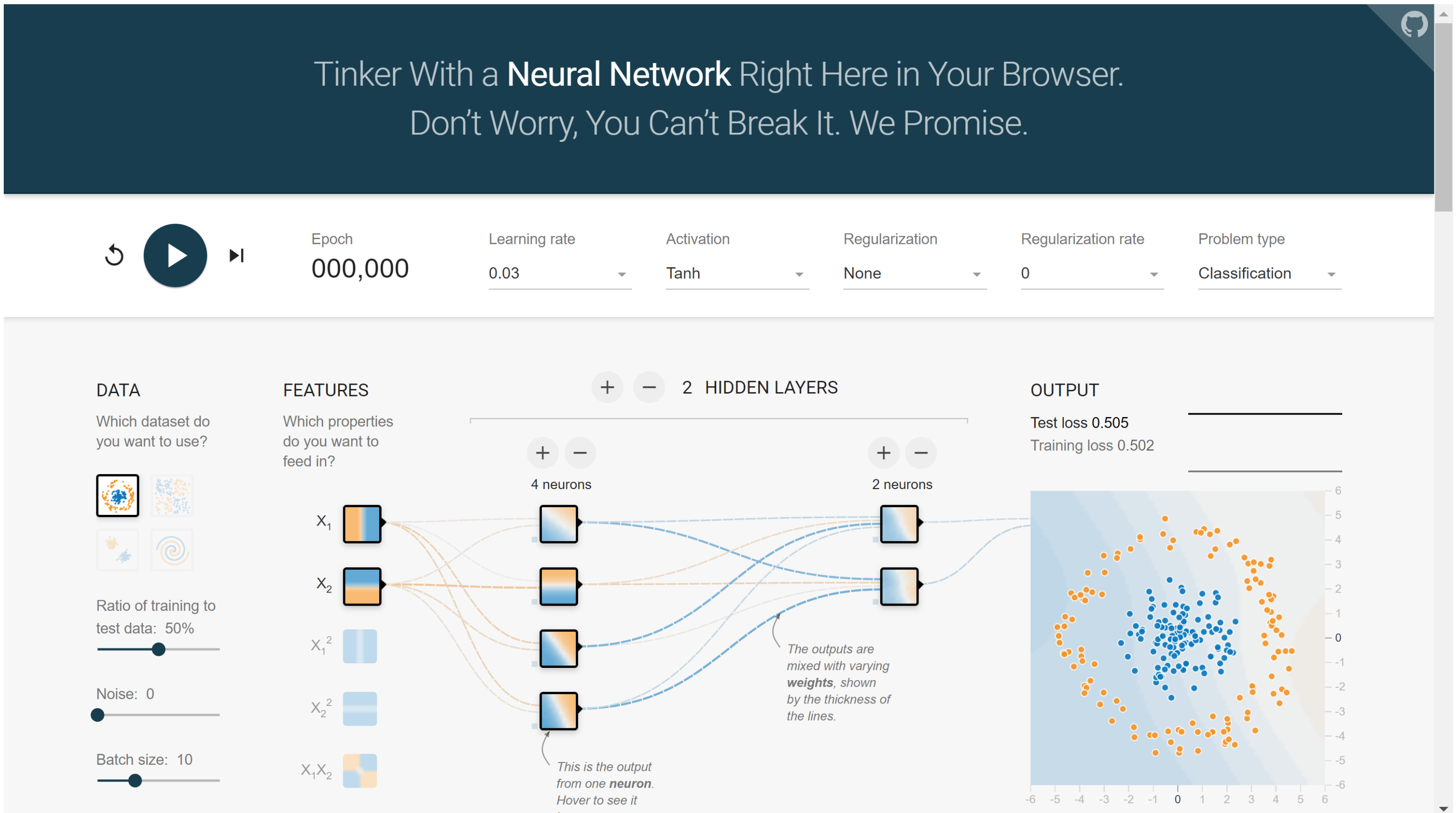
2D example of gradient descent



- In reality, in deep learning we are optimizing a highly complex loss function with millions of variables (or more)
- More on this later…

https://laptrinhx.com/gradient-descent-animation-2-multiple-linear-regression-3070246823/

2D example: TensorFlow Playground



https://playground.tensorflow.org

Questions?

Convolutional neural networks (or CNNs, or ConvNets)

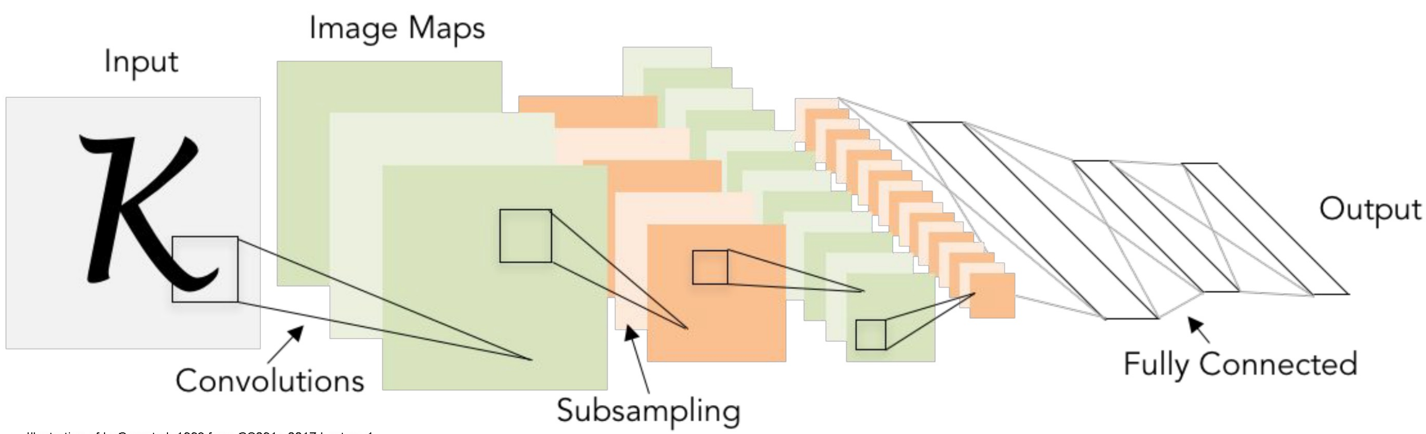


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

A bit of history...

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

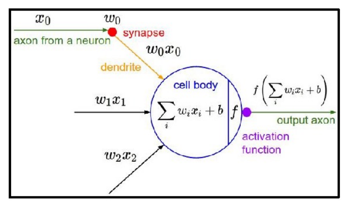
The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.

recognized
letters of the alphabet

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

update rule:

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



Frank Rosenblatt, ~1957: Perceptron

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leur

A bit of history...

[Hinton and Salakhutdinov 2006]

Reinvigorated research in Deep Learning

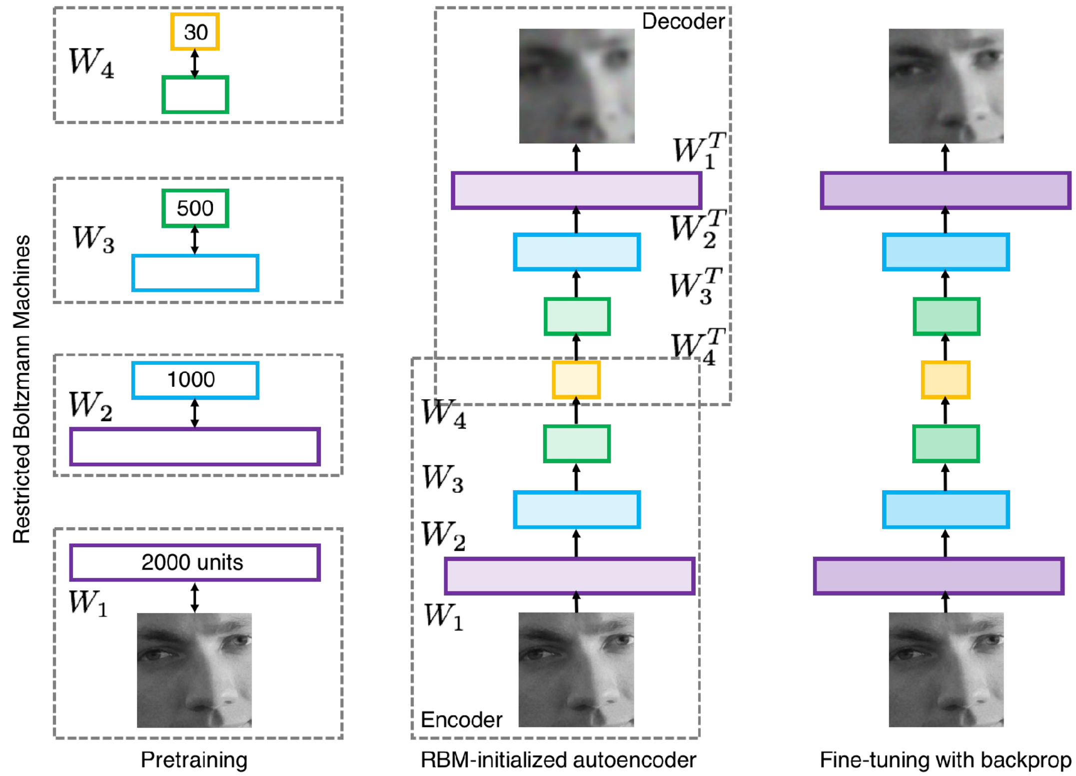


Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

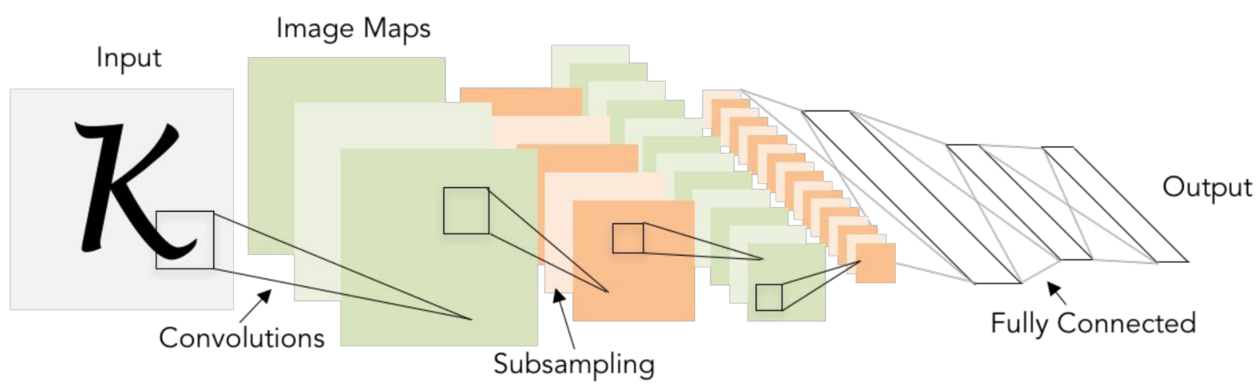
Hinton and Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 2016.

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]



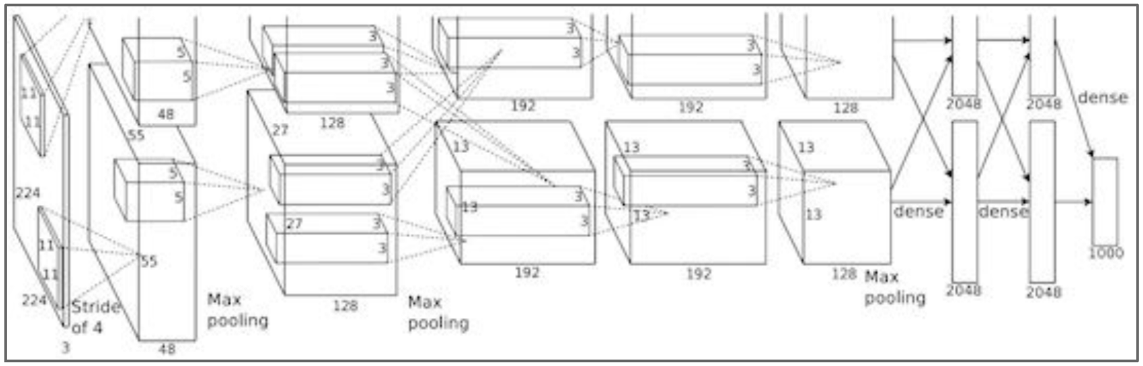
LeNet-5

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

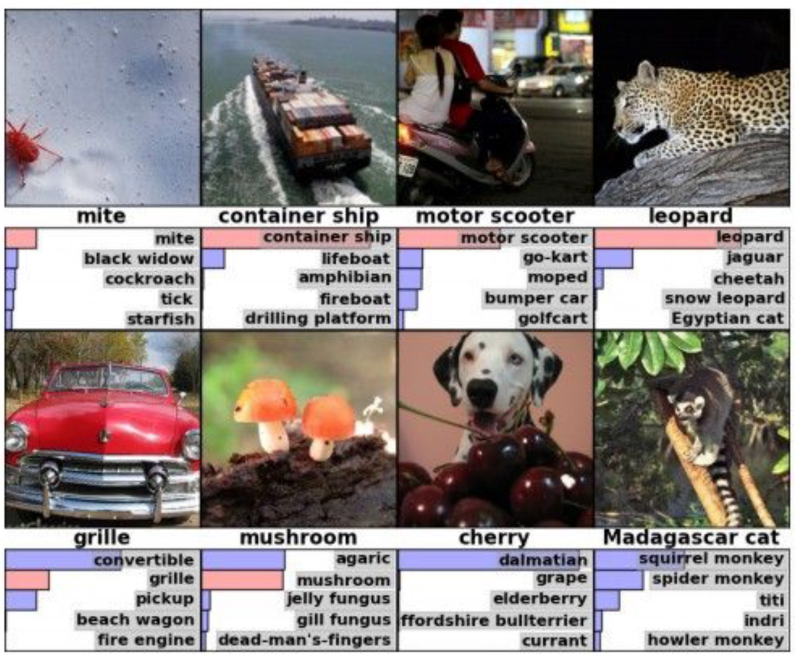


"AlexNet"

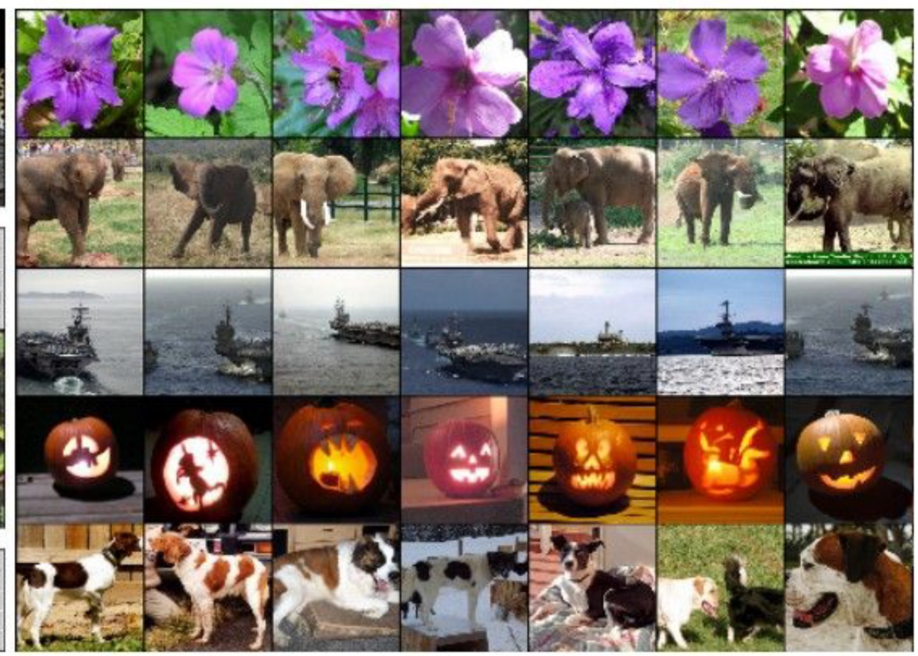
Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

Fast-forward to today: ConvNets* are everywhere

Classification



Retrieval

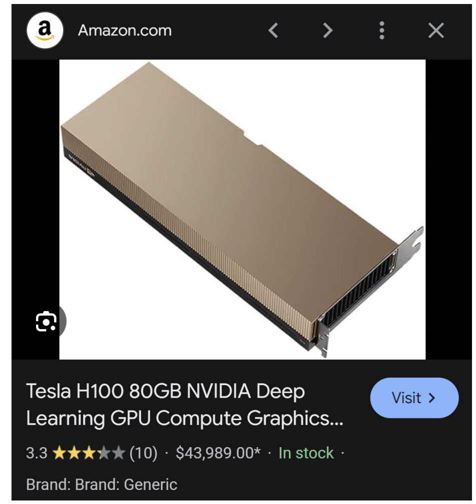


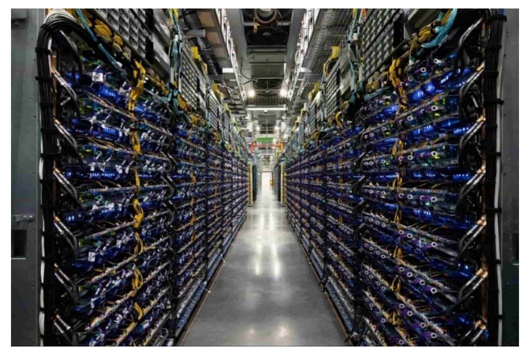
* and other recent architectures, like Transformers

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

Fast-forward to today: ConvNets are everywhere

Self-driving cars (video courtesy Tesla)
https://www.tesla.com/AI





Cloud TPU v4 Pods
https://cloud.google.com/tpu/

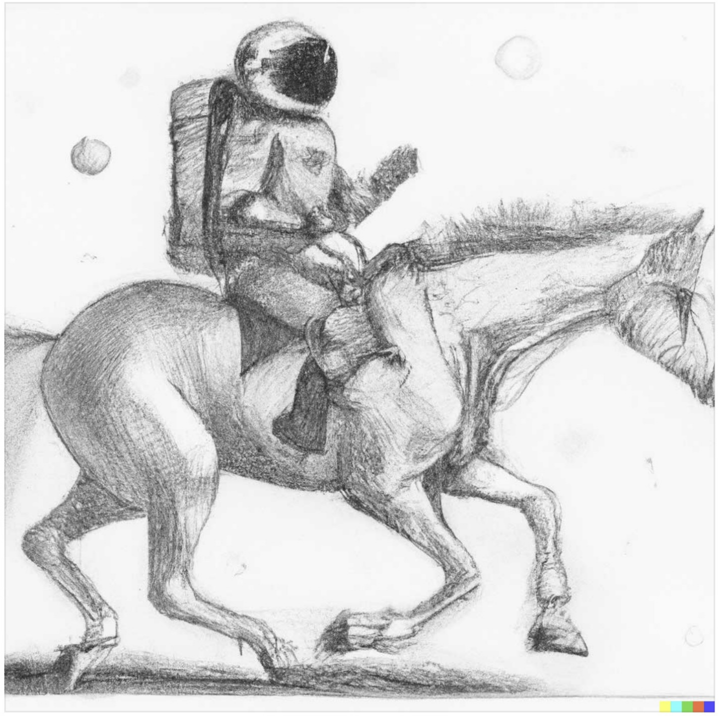
Text-to-image

An astronaut Teddy bears A bowl of soup

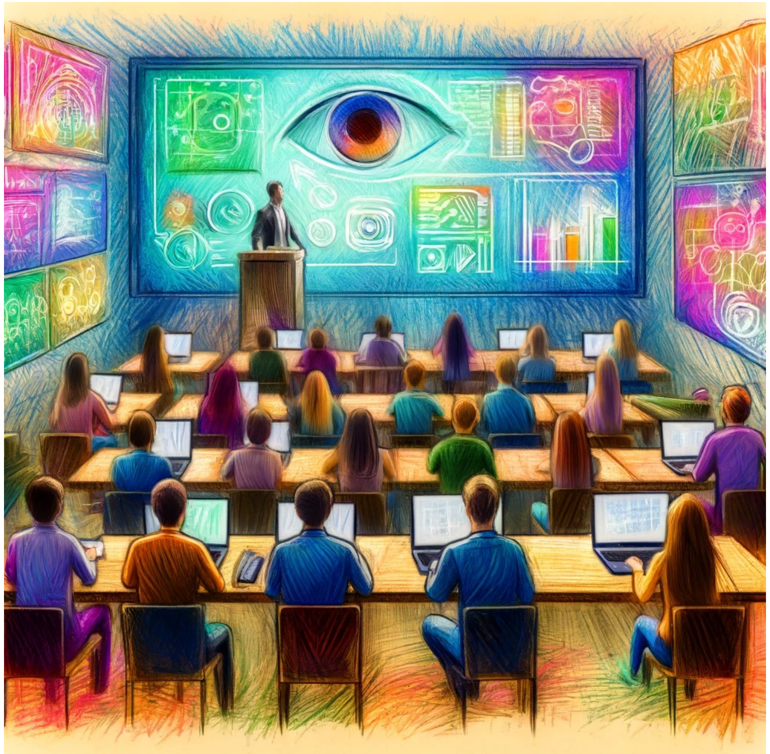
riding a horse lounging in a tropical resort in space playing basketball with cats in space

in a photorealistic style in the style of Andy Warhol as a pencil drawing

→



https://openai.com/dall-e-2/

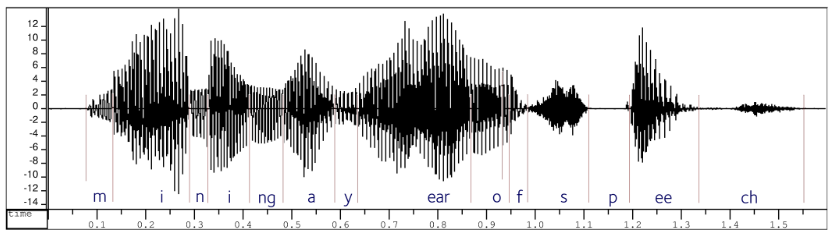


“A computer vision class watching a cool lecture, crayon drawing”

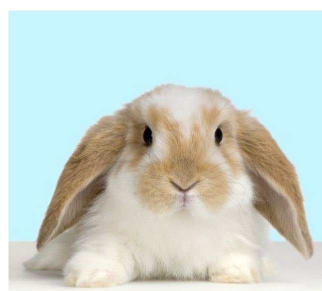
“A computer vision class watching a cool lecture, album cover”

What is a ConvNet?

- Version of deep neural networks designed for signals
  - 1D signals (e.g., speech waveforms)



  - 2D signals (e.g., images)



Motivation – Feature Learning

Life Before Deep Learning

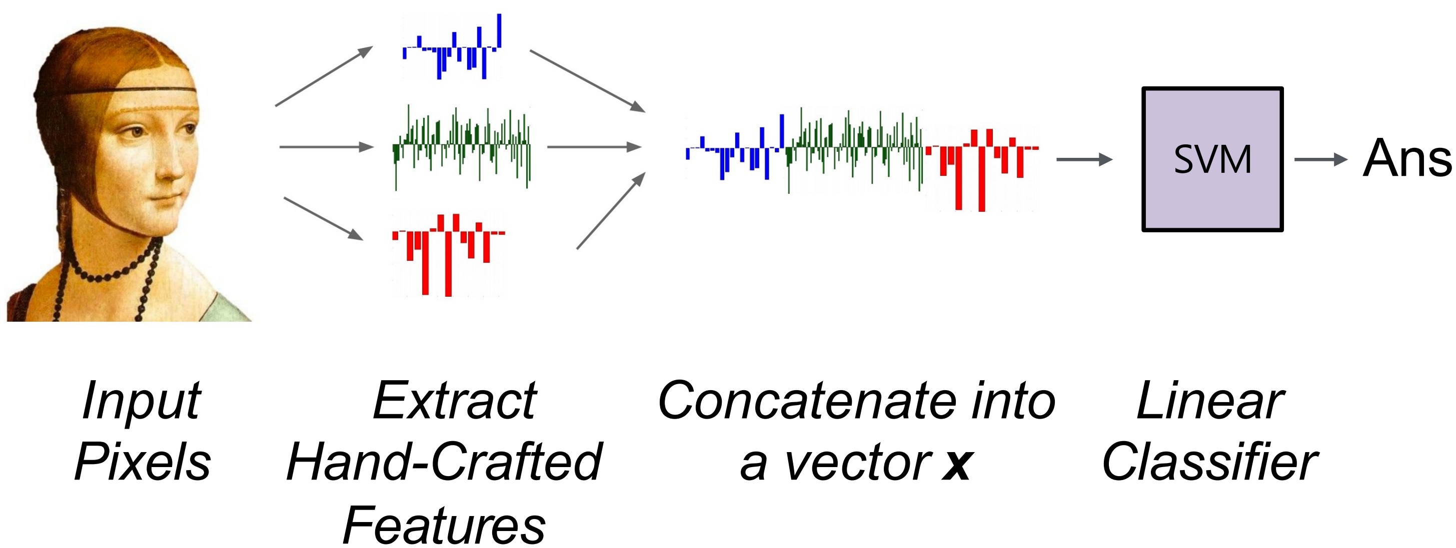
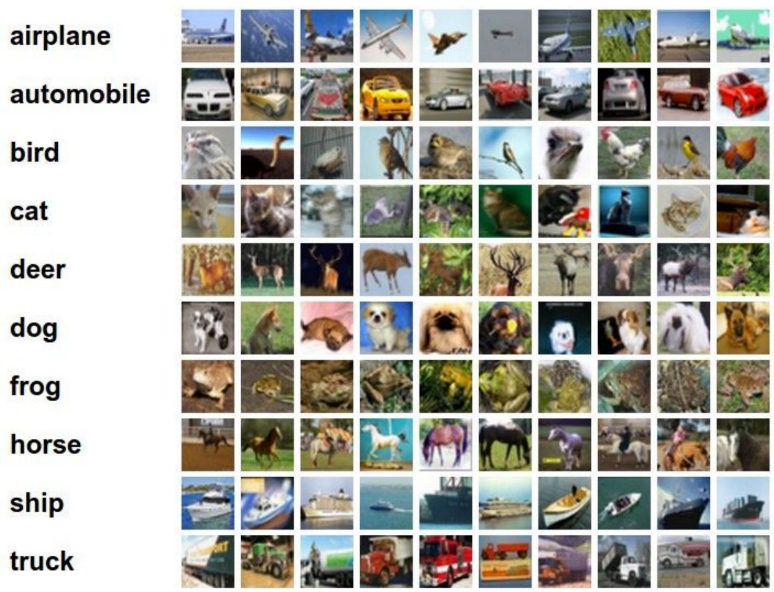


Figure: Karpathy 2016

Why use features? Why not pixels?



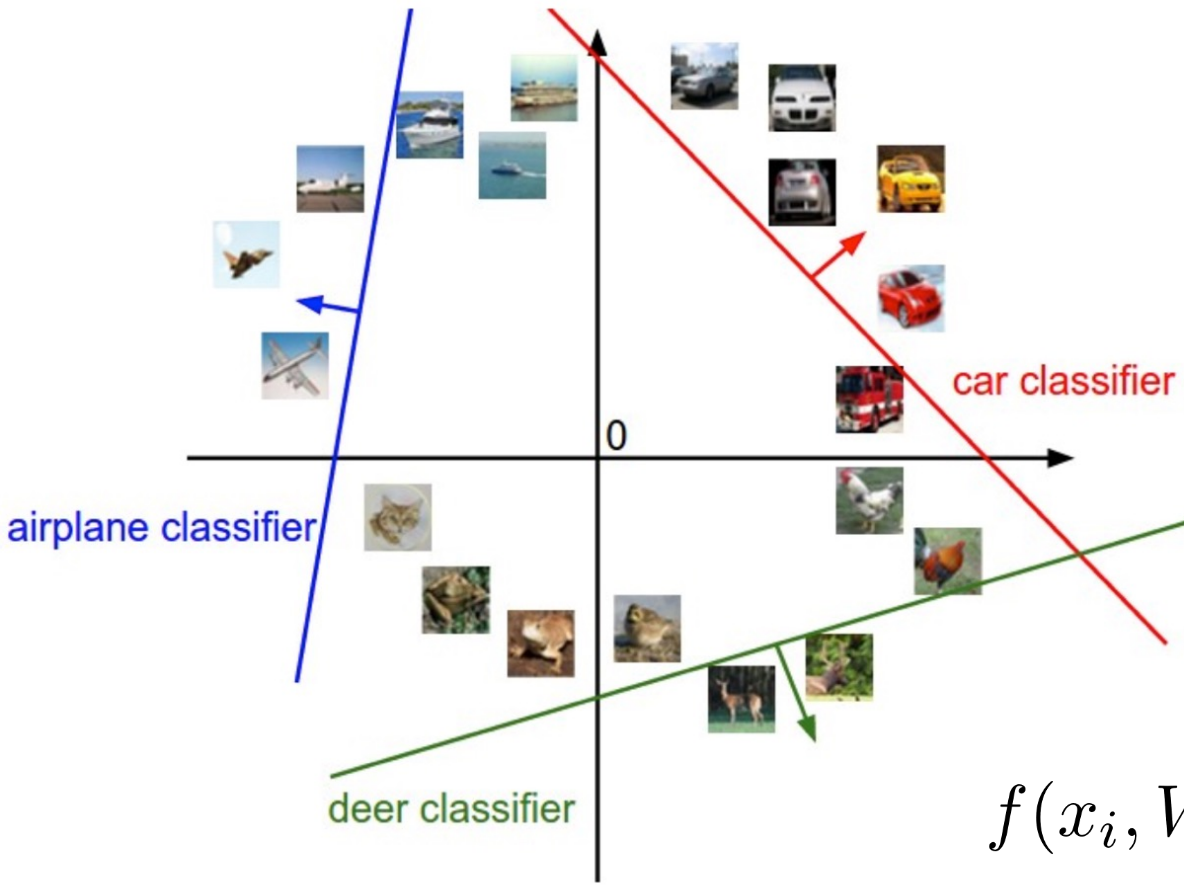
$$f(x_i, W) = Wx_i + b$$

Q: What would be a very hard set of classes for a linear classifier to distinguish?

(assuming x = pixels)

Slide from Karpathy 2016

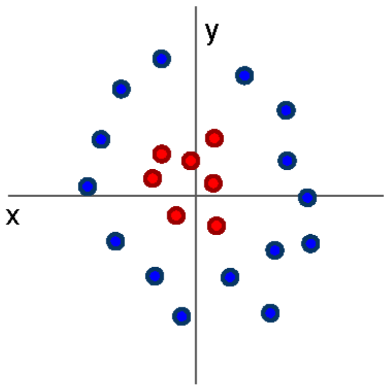
Goal: linearly separable classes



$$f(x_i, W) = Wx_i + b$$

Aside: Image Features

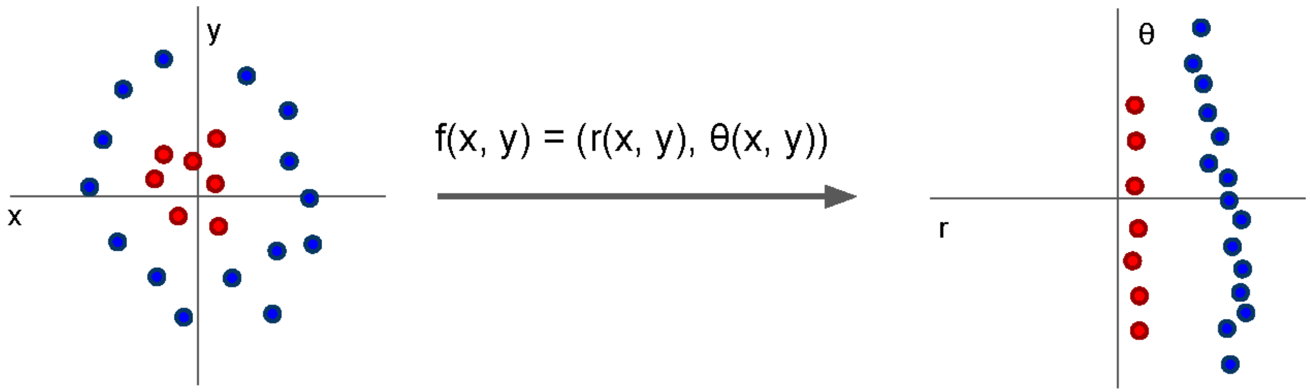
Image Features: Motivation



Cannot separate red and blue points with linear classifier

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

Image Features: Motivation

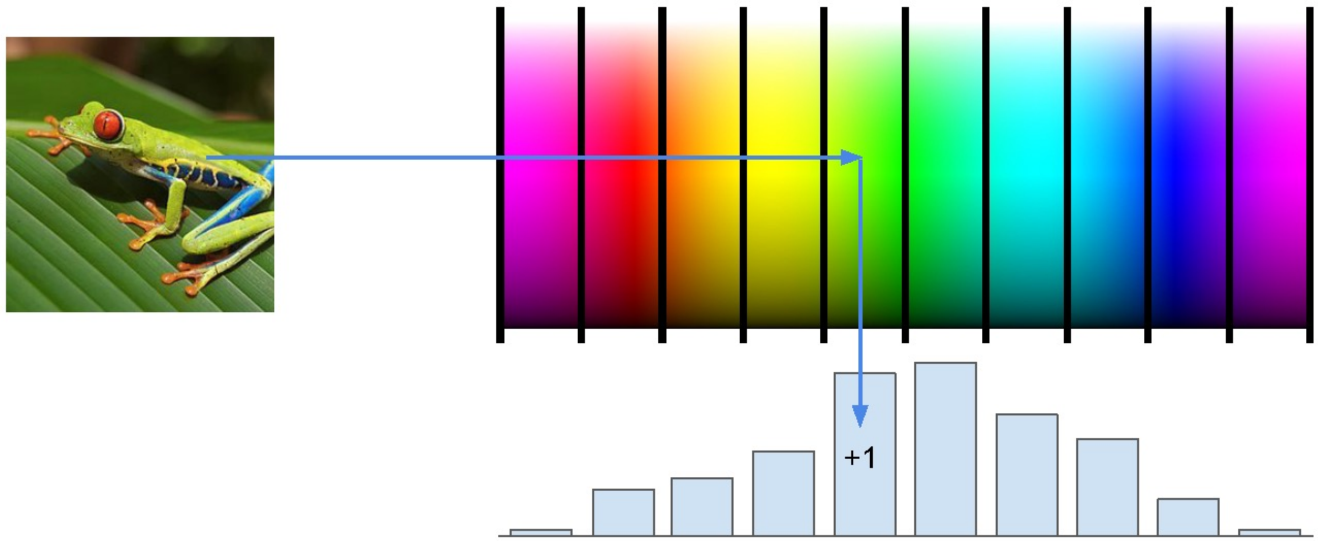


Cannot separate red and blue points with linear classifier

After applying feature transform, points can be separated by linear classifier

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

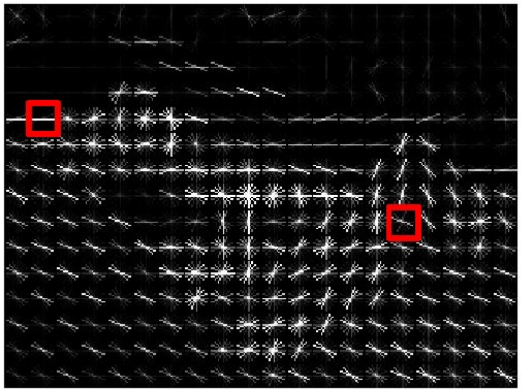
Example: Color Histogram



Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

Example: Histogram of Oriented Gradients (HoG)

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

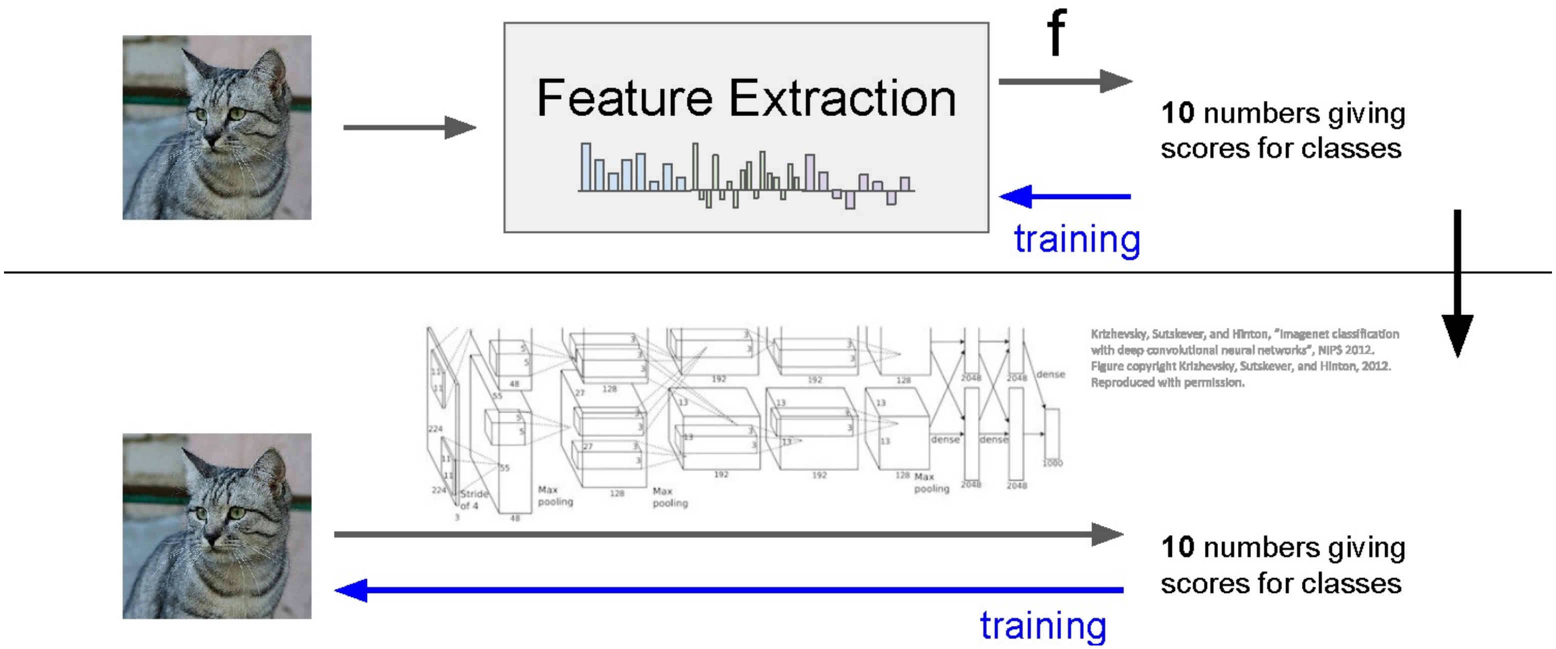


Example: 320x240 image gets divided into 40x30 bins; in each bin there are 9 numbers so feature vector has 30*40*9 = 10,800 numbers

Lowe, "Object recognition from local scale-invariant features", ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

Image features vs ConvNets

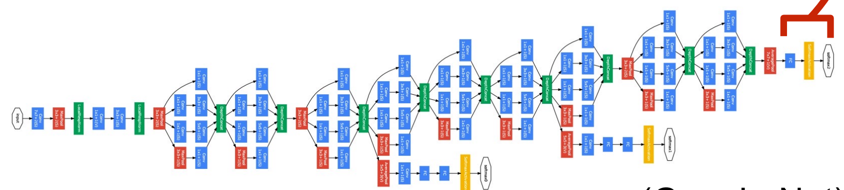


Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012.

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

Last layer of many CNNs is a linear classifier

This piece is just a linear classifier



(GoogLeNet)

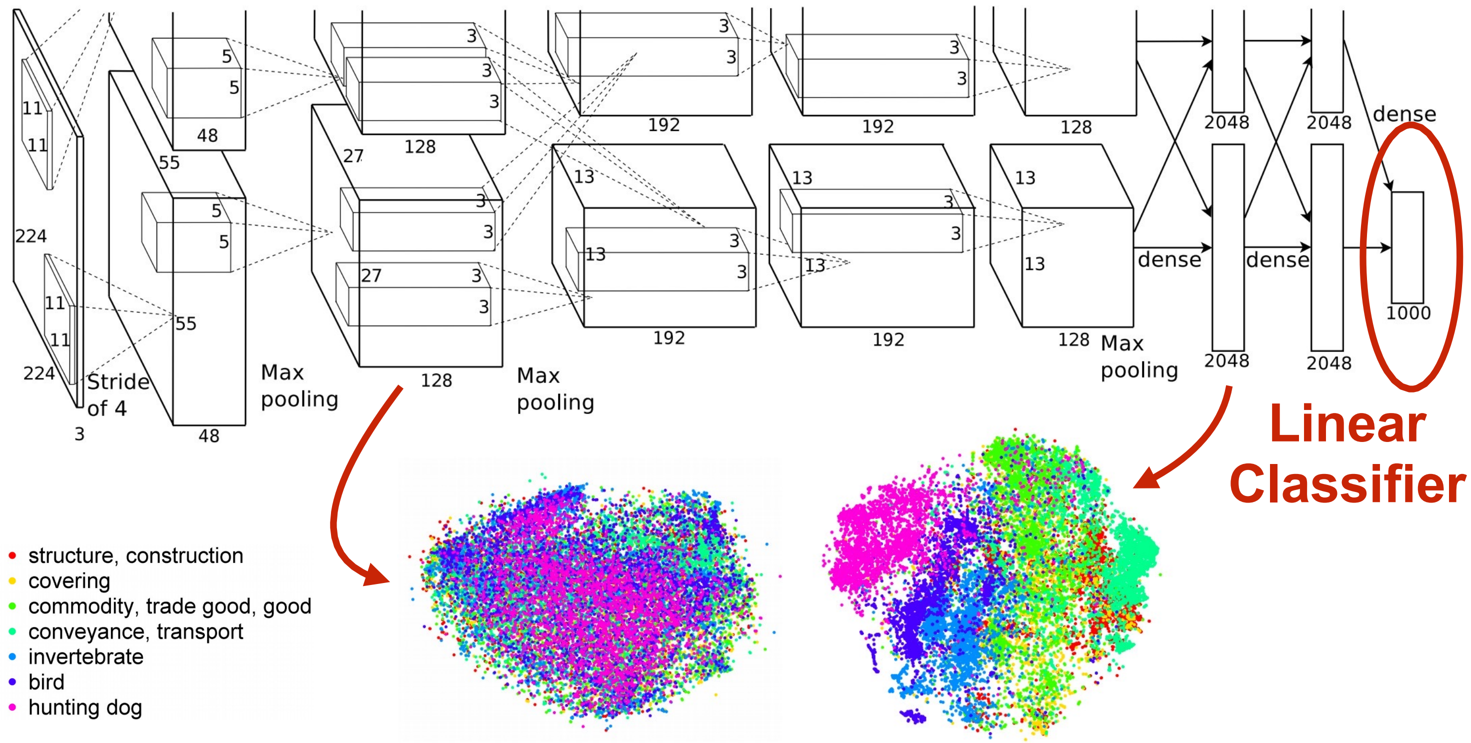
→ Ans

Input Pixels

Perform everything with a big neural network, trained end-to-end

Key: perform enough processing so that by the time you get to the end of the network, the classes are linearly separable

Visualizing AlexNet in 2D with t-SNE



(2D visualization using t-SNE)

(c) DeCAF_1

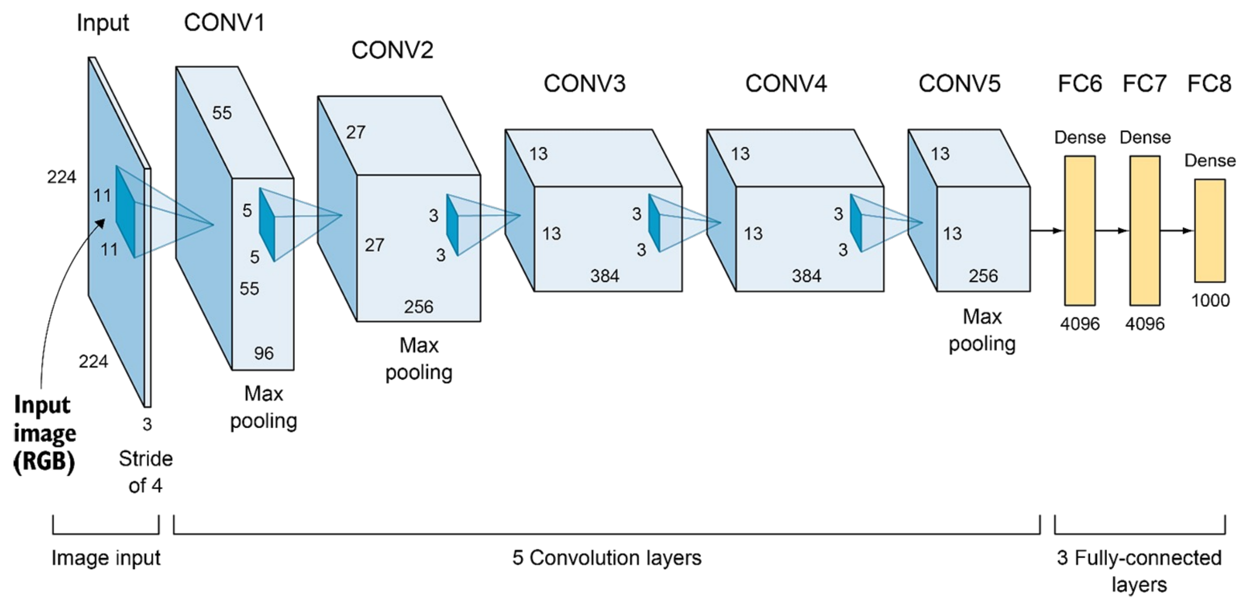
(d) DeCAF_6

[Donahue, “DeCAF: DeCAF: A Deep Convolutional …”, arXiv 2013]

Convolutional neural networks

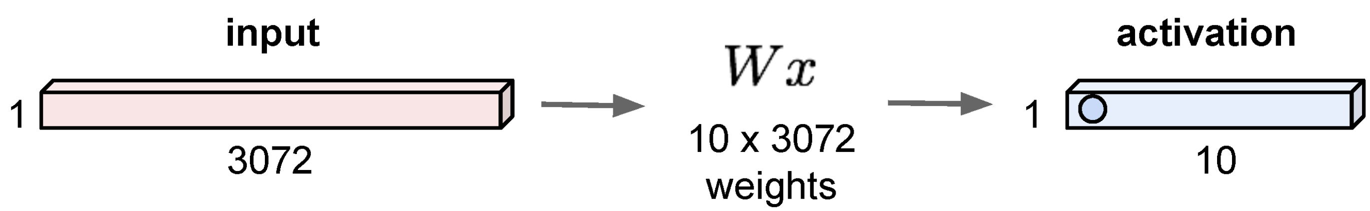
Layer types:

- *Convolutional layer*
- Pooling layer
- Fully-connected layer



Fully Connected Layer

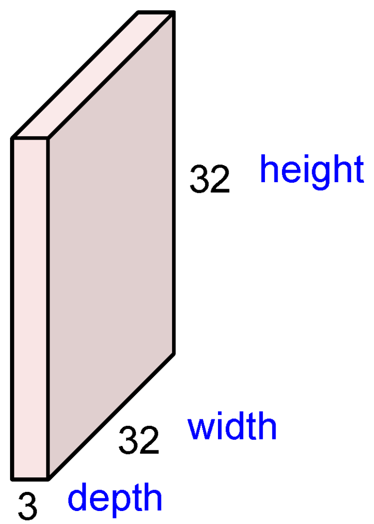
32x32x3 image -> stretch to 3072 x 1



Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

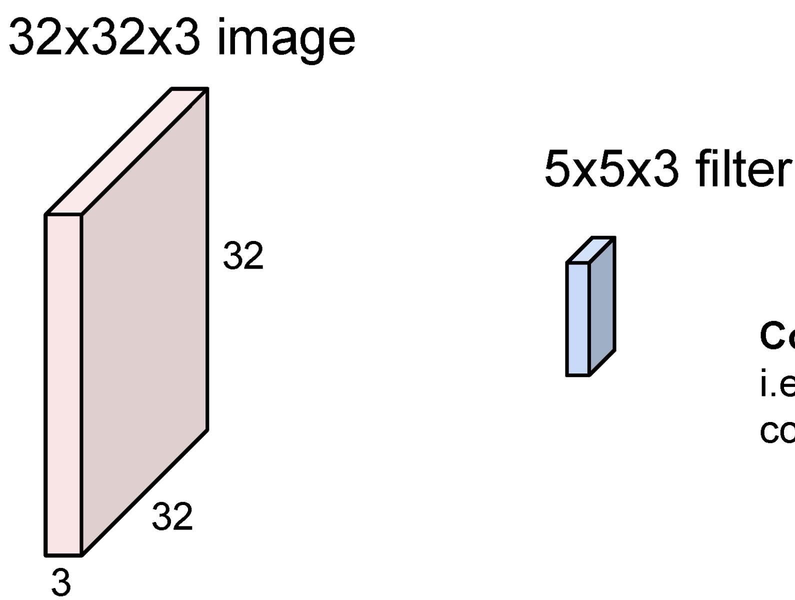
Convolution Layer

32x32x3 image -> preserve spatial structure



Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

Convolution Layer



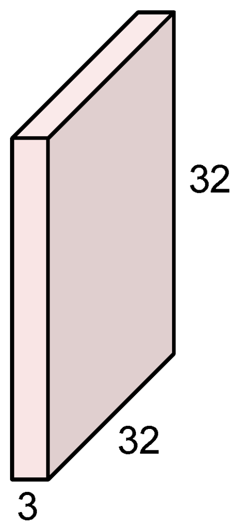
Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

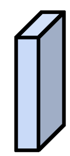
Convolution Layer

Filters always extend the full
depth of the input volume

32x32x3 image



5x5x3 filter

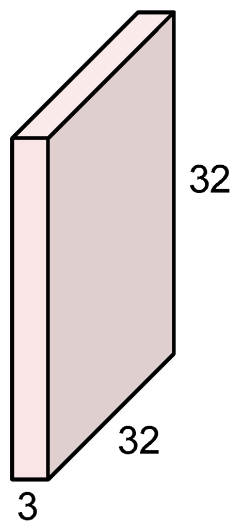


Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"

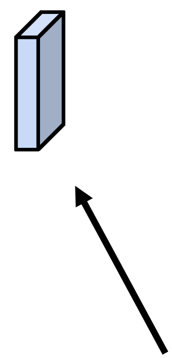
Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

Convolution Layer

32x32x3 image



5x5x3 filter

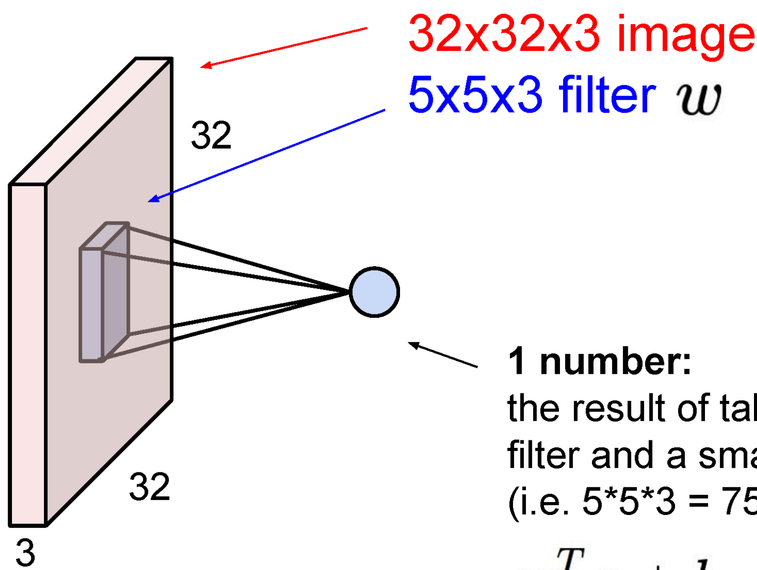


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Number of weights: 5 x 5 x 3 + 1 = **76**
(vs. 3072 for a fully-connected layer)
(+1 for bias term)

Adapted from Fei-Fei Li & Andrej Karpathy & Serena Le

Convolution Layer

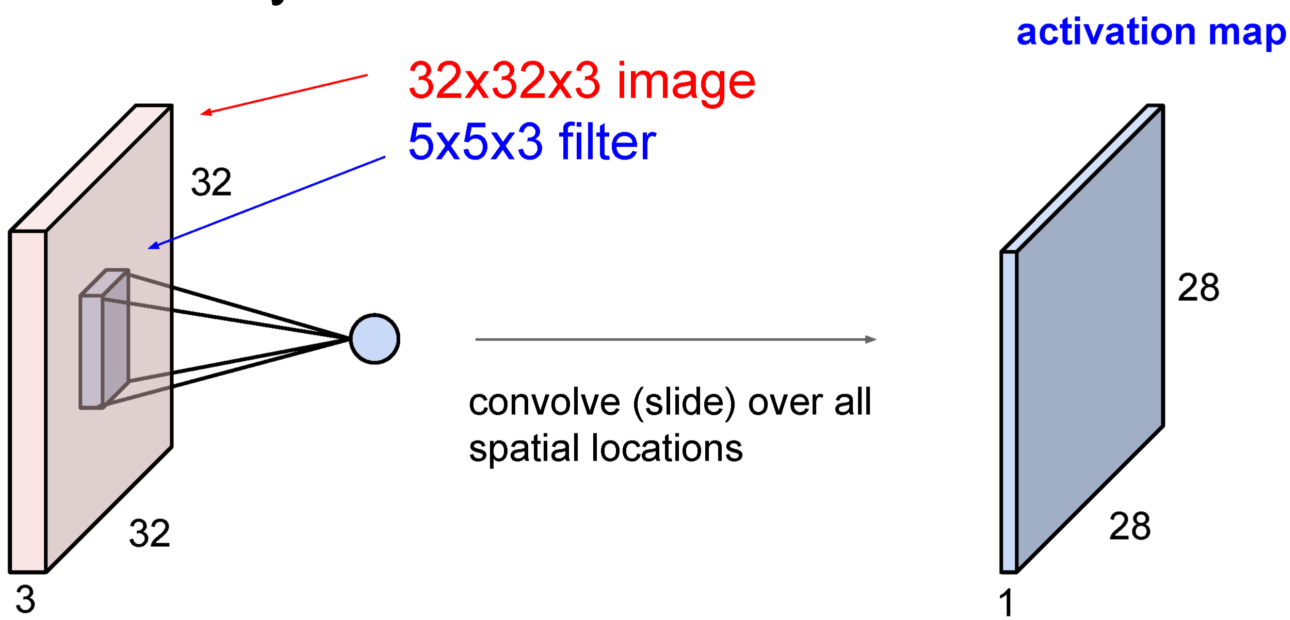


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

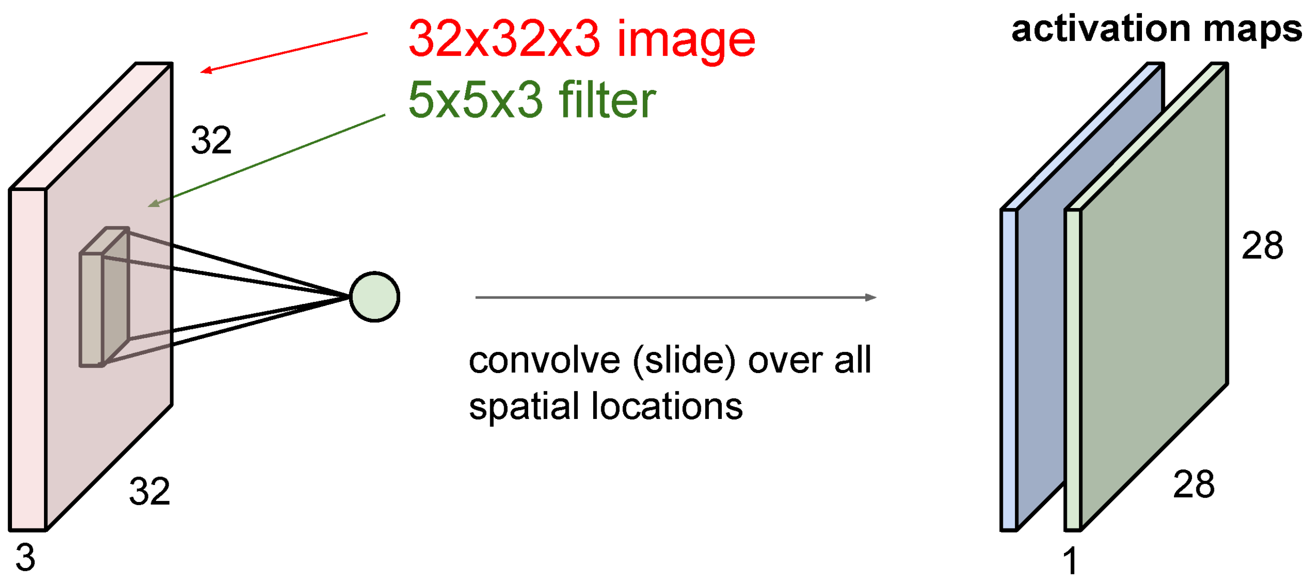
Convolution Layer



Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

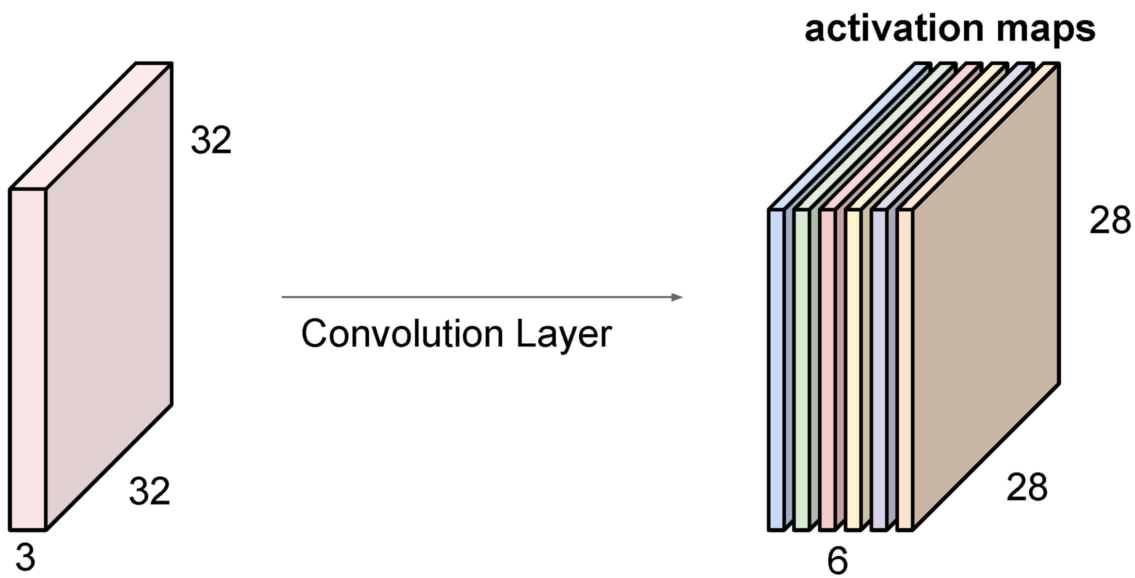
Convolution Layer

consider a second, green filter



Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

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



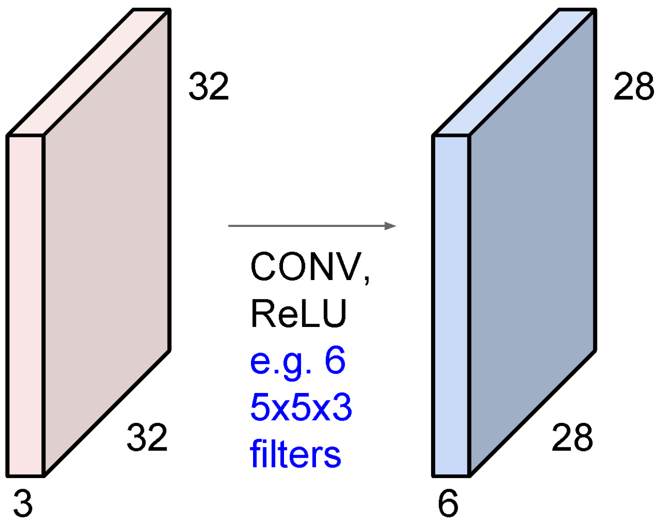
We stack these up to get a "new image" of size 28x28x6!

(total number of parameters to learn: 6 x (75 + 1) = **456**) slido

How many parameters are in a convolution layer consisting of 3 3x3x1 filters (each with bias term)?

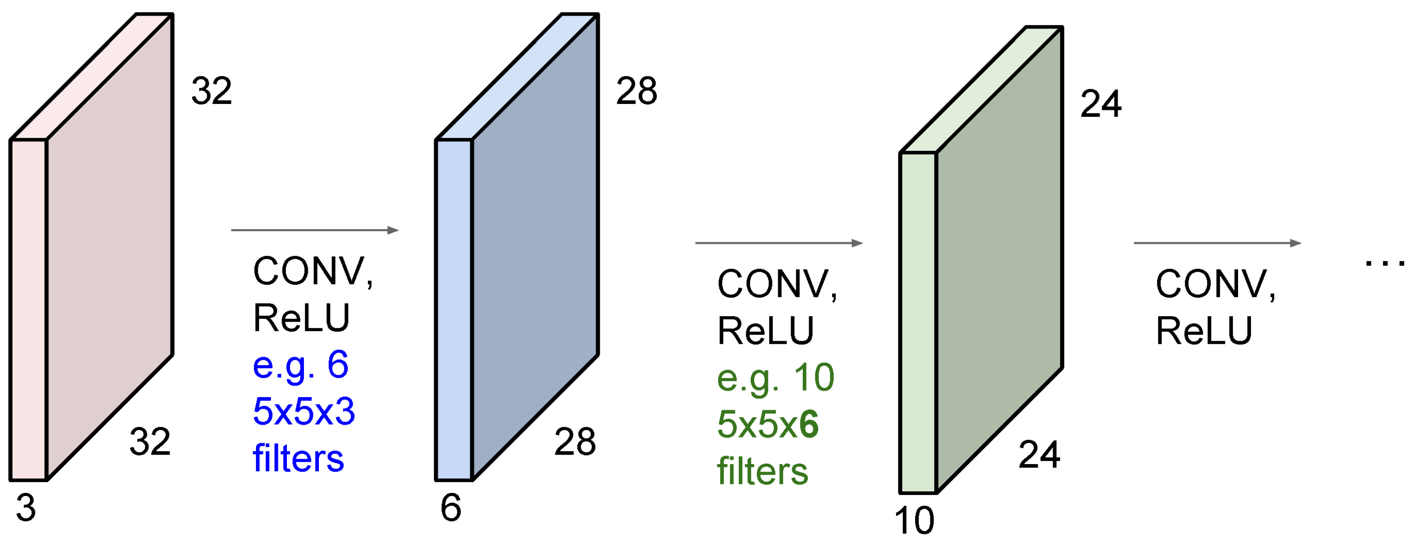
ⓘ Start presenting to display the poll results on this slide.

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

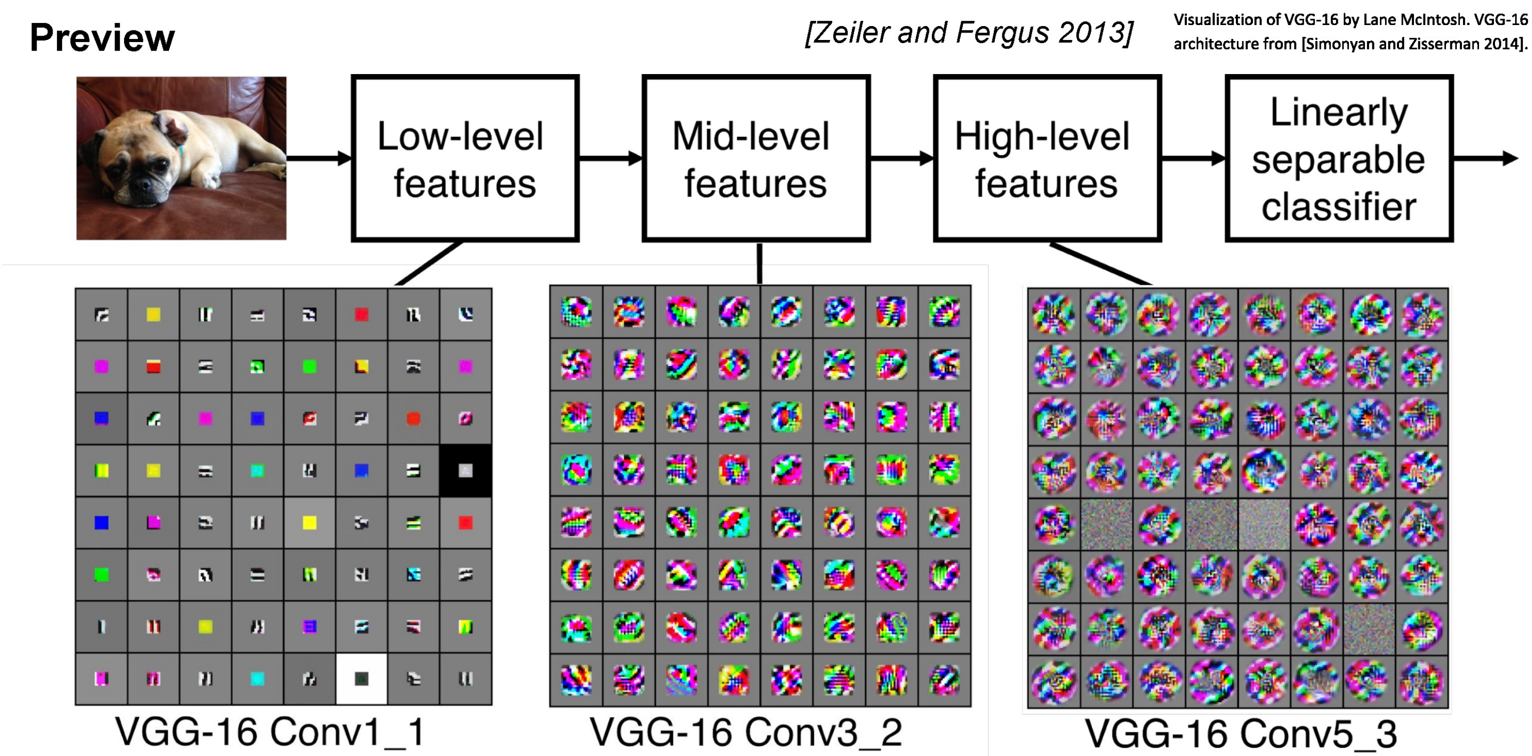


Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

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

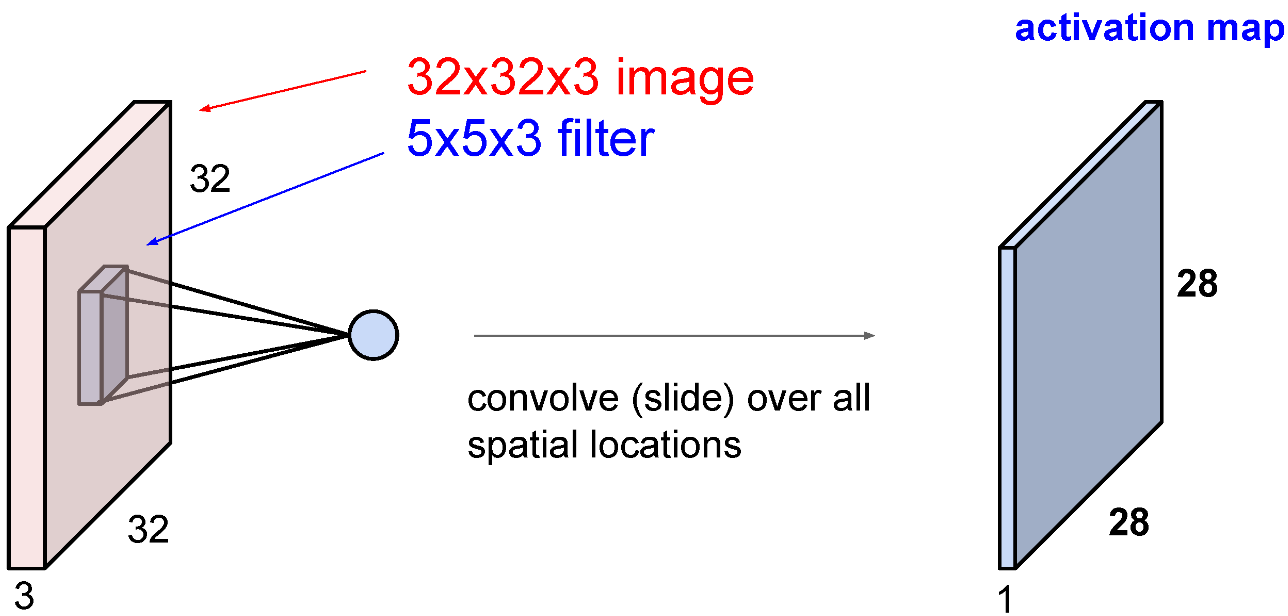


Preview



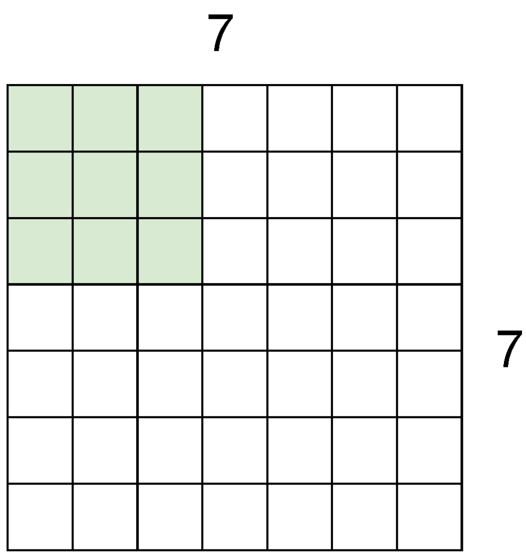
Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

A closer look at spatial dimensions:



Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

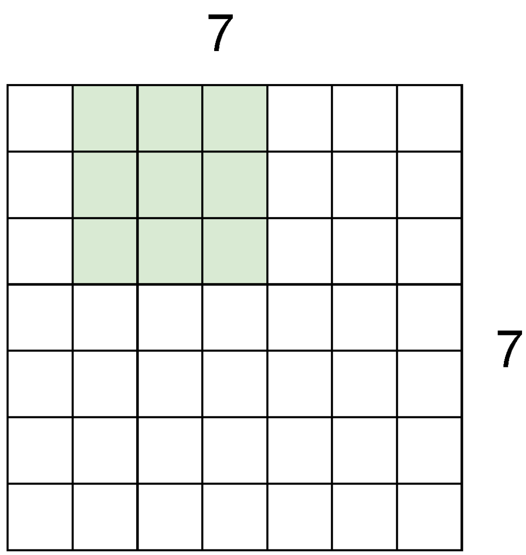
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

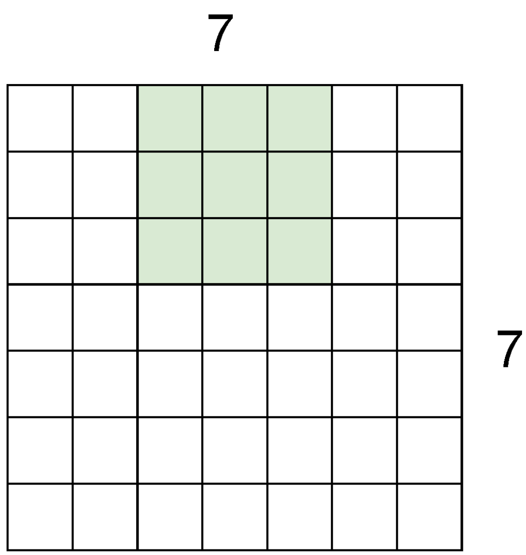
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

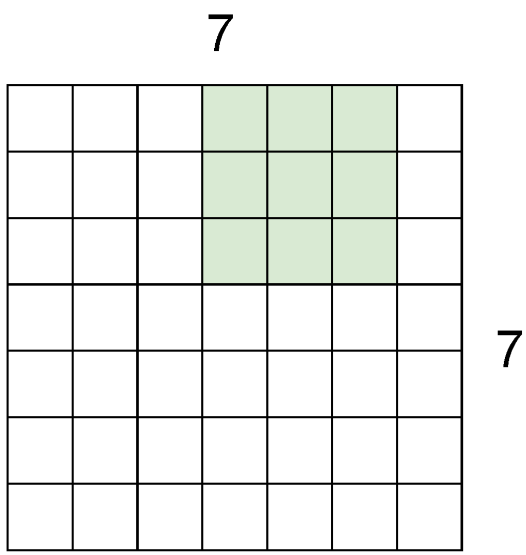
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

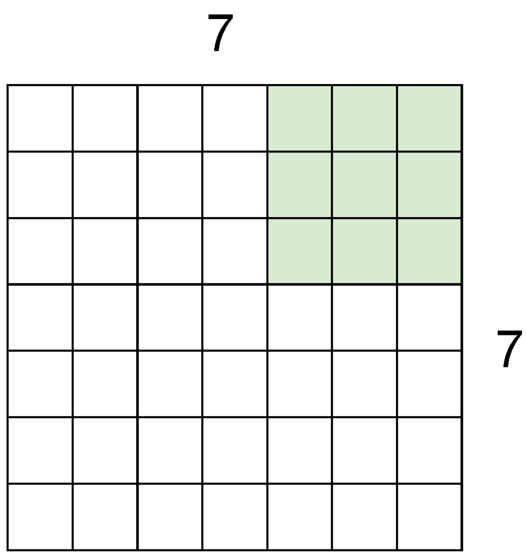
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

A closer look at spatial dimensions:

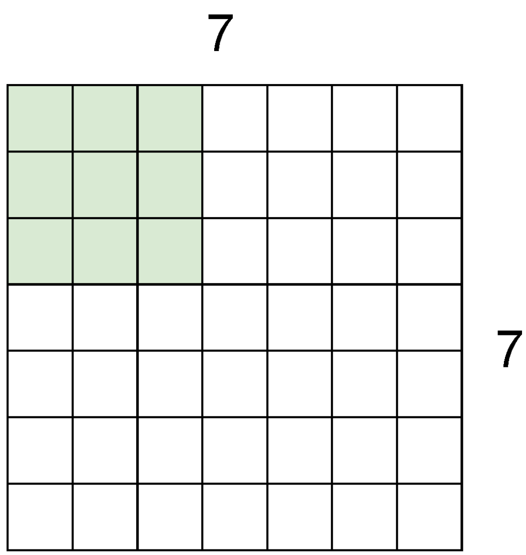


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

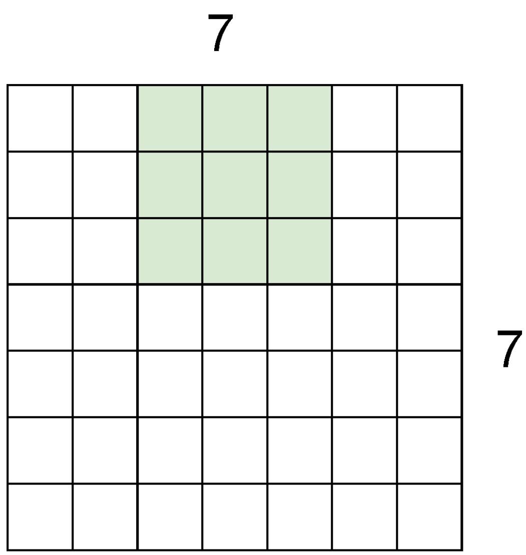
A closer look at spatial dimensions:



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

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

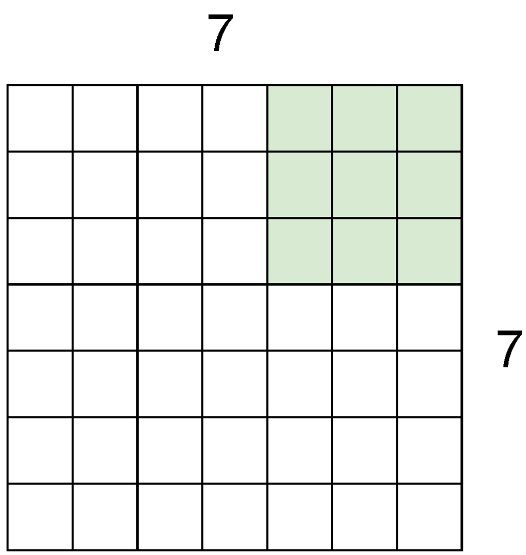
A closer look at spatial dimensions:



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

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

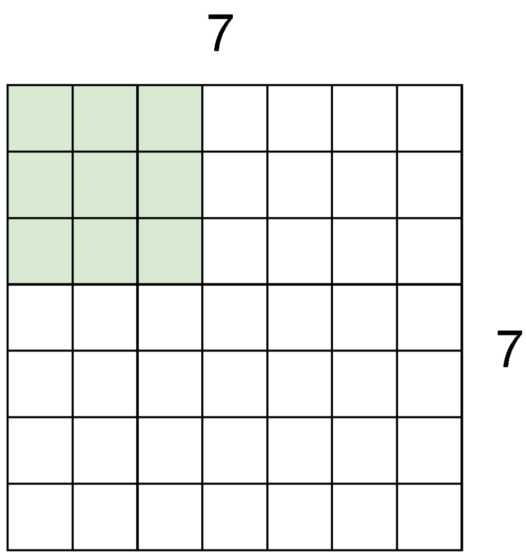
A closer look at spatial dimensions:



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

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

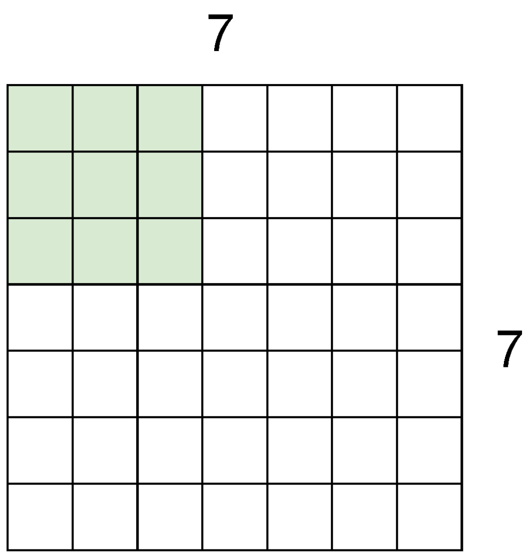
A closer look at spatial dimensions:



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

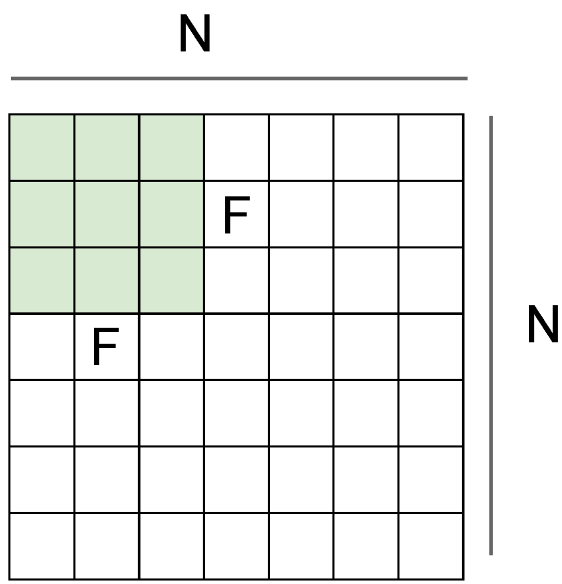
Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

A closer look at spatial dimensions:



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

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

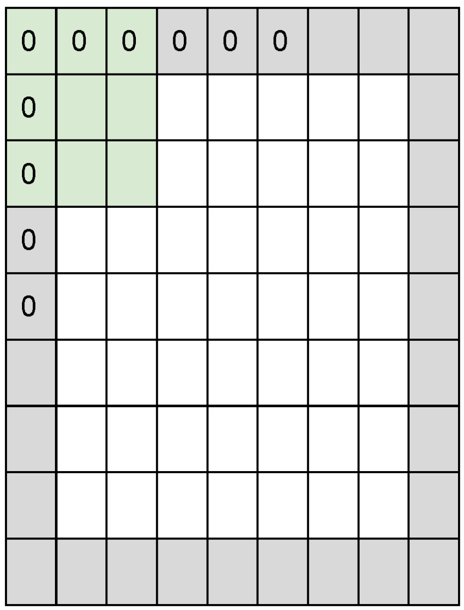


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

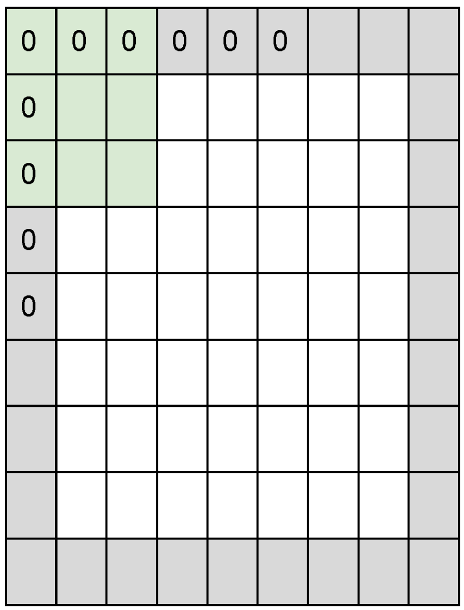
In practice: Common to zero pad the border



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

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

In practice: Common to zero pad the border

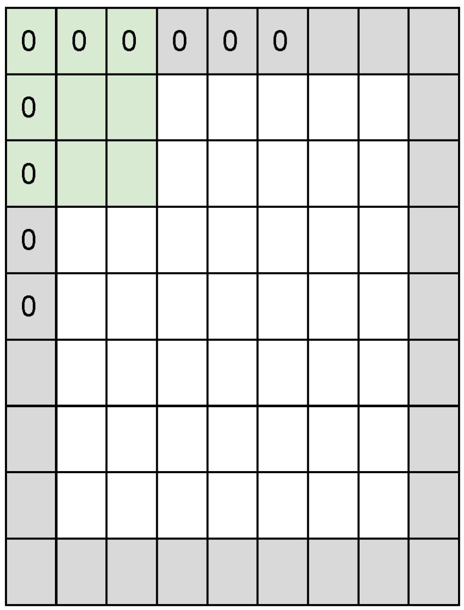


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

In practice: Common to zero pad the border



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

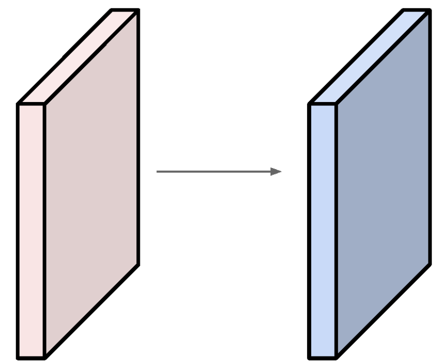
7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

Examples time:

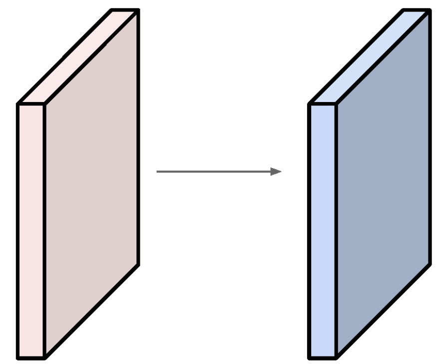
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?



Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

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

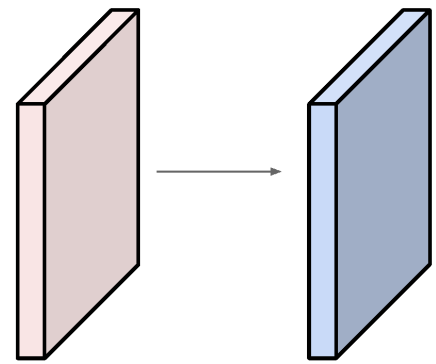
32x32x10

Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

Examples time:

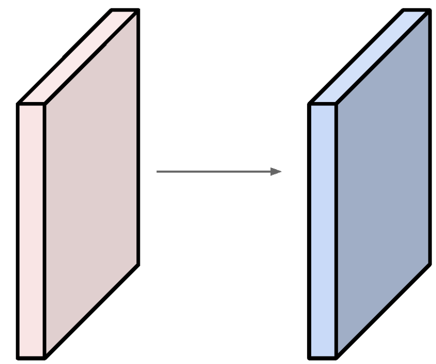
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?



Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

Examples time:



Input volume: **32x32x3**

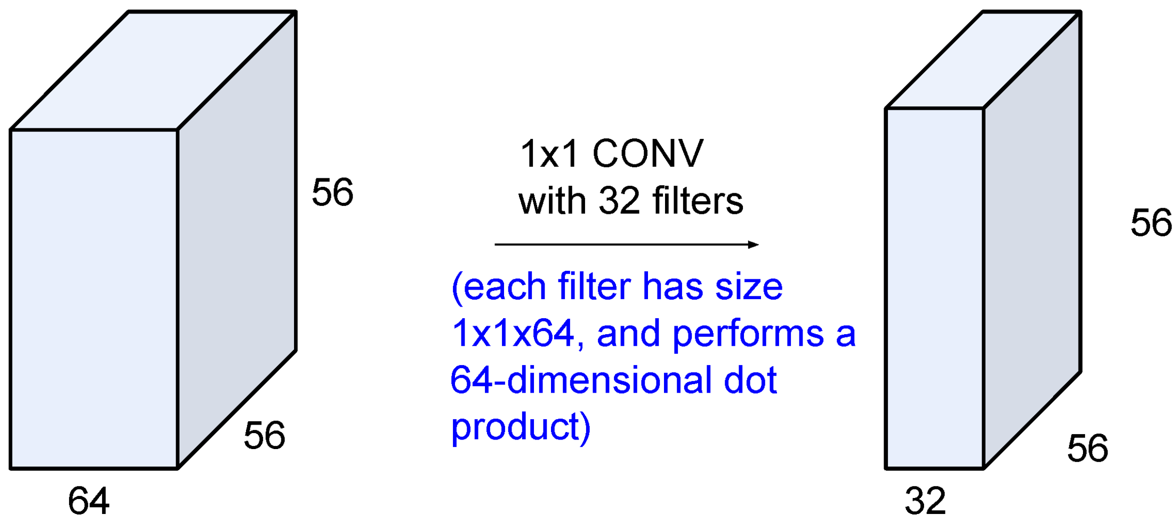
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

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

=> 76*10 = **760**

“1x1 convolutions”

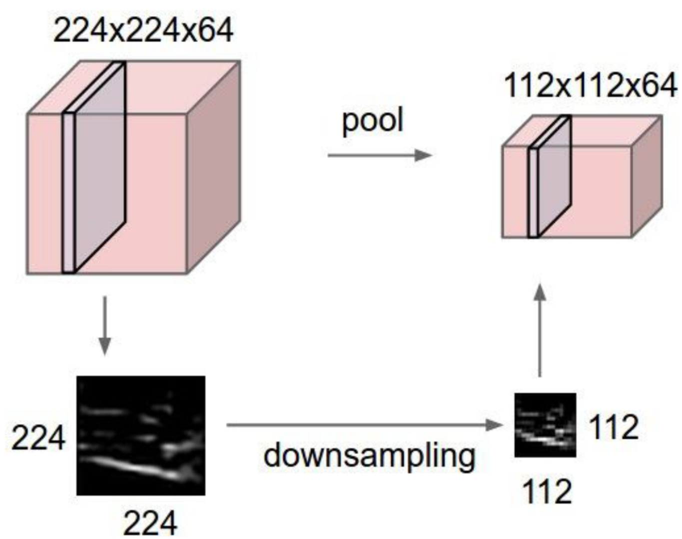


Convolutional layer—properties

- Small number of parameters to learn compared to a fully connected layer
- Preserves spatial structure—output of a convolutional layer is shaped like an image
- **Translation equivariant**: passing a translated image through a convolutional layer is (almost) equivalent to translating the convolution output (but be careful of image boundaries)

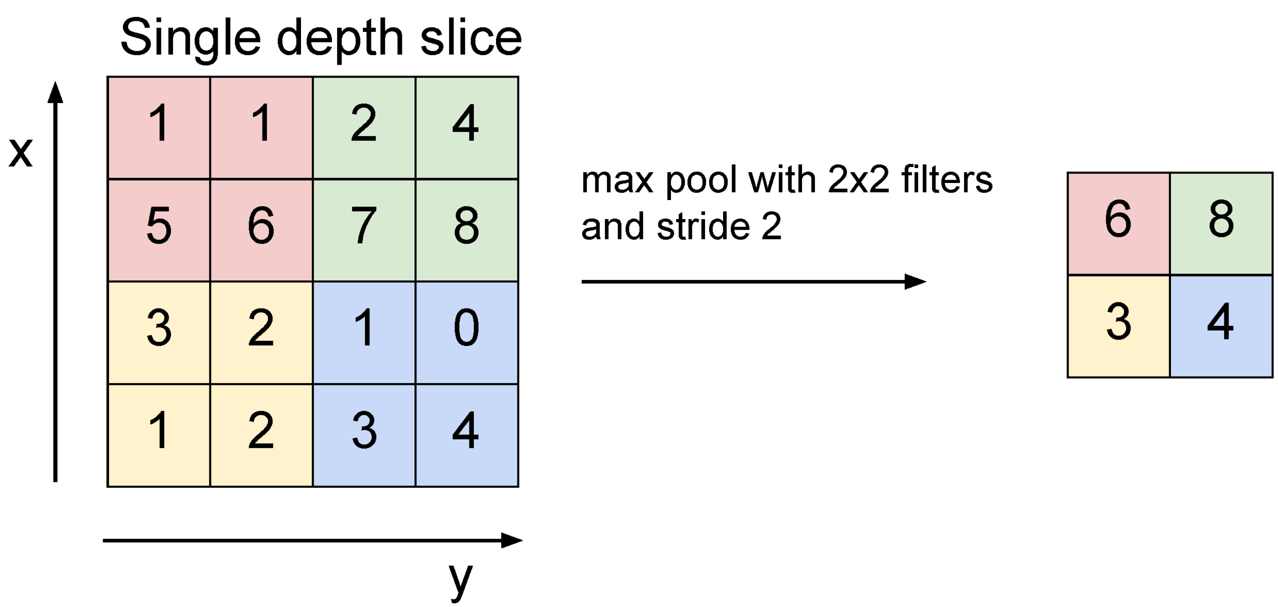
Pooling layer

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



Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

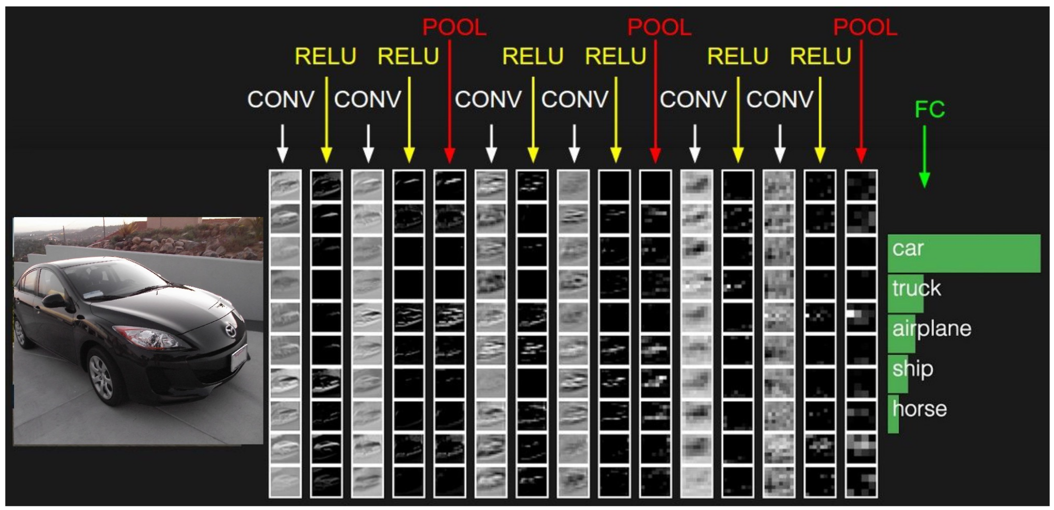
MAX POOLING



Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

Fully Connected Layer (FC layer)

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



Slide credit: Fei-Fei Li & Andrej Karpathy & Serena Leu

[ConvNetJS demo: training on CIFAR-10]

ConvNetJS CIFAR-10 demo

Description

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and verically.

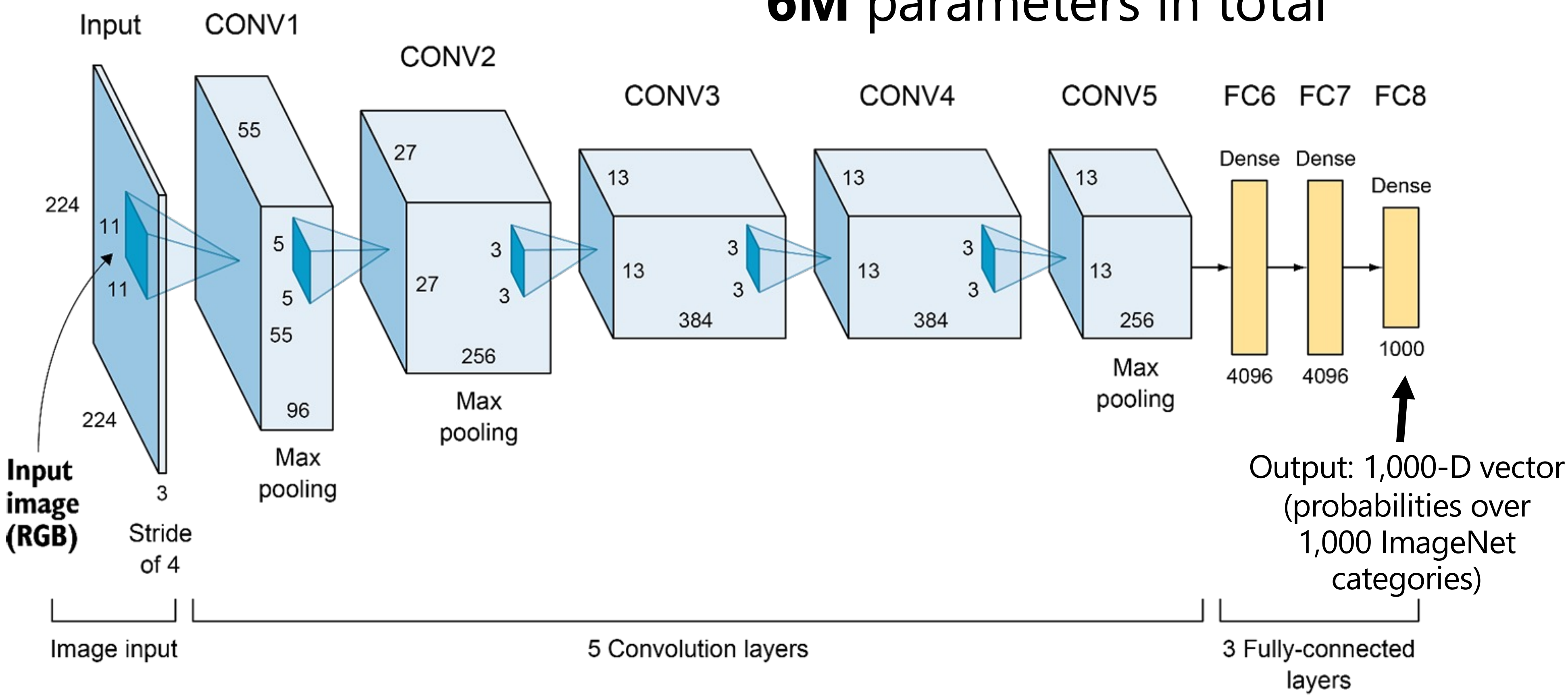
By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.

https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

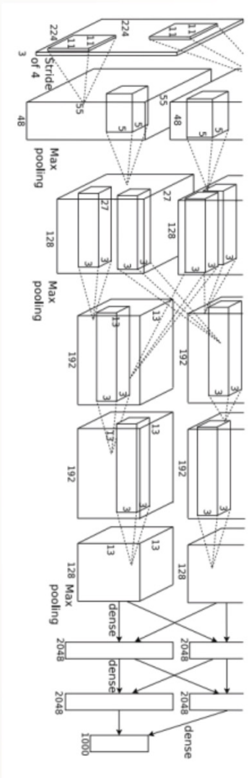
AlexNet (2012)

6M parameters in total



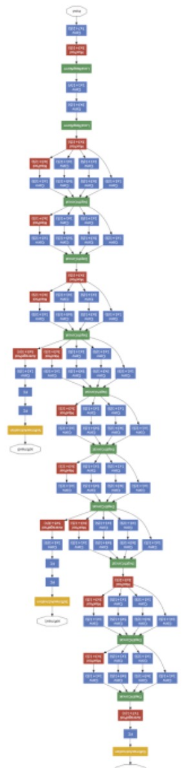
Elgendy, *Deep Learning for Vision Systems*, https://livebook.manning.com/book/grokking-deep-learning-for-computer-vision/chapter-5/v-3/

"AlexNet"



[Krizhevsky et al. NIPS 2012]

"GoogLeNet"

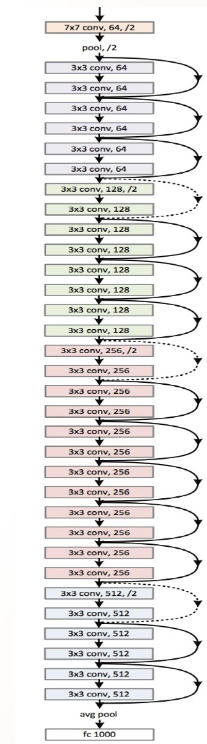


[Szegedy et al. CVPR 2015]

"VGG Net"

[Simonyan & Zisserman, ICLR 2015]

"ResNet"



[He et al. CVPR 2016]

Big picture

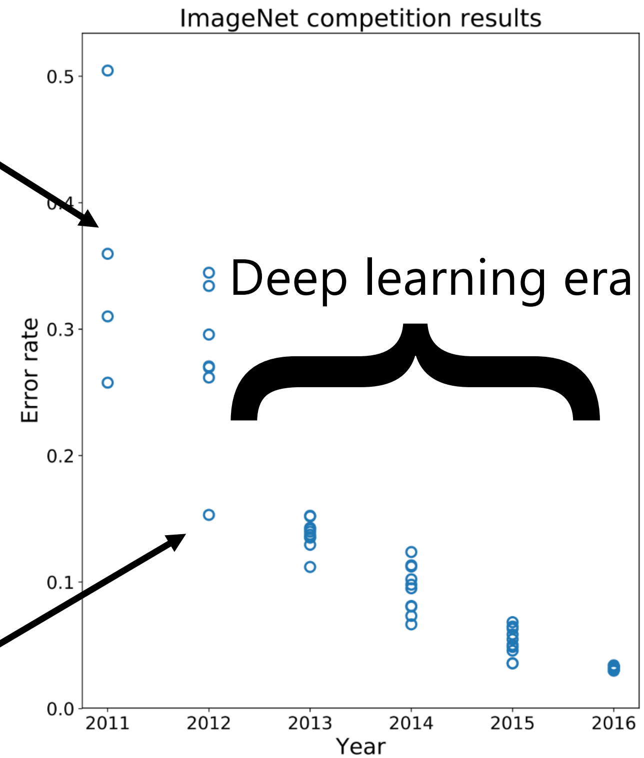
- A convolutional neural network can be thought of as a function from images to class scores
  - With millions of adjustable weights…
  - … leading to a very non-linear mapping from images to features / class scores.
  - We will set these weights based on classification accuracy on training data…
  - … and hopefully our network will generalize to new images at test time

Data is key—enter ImageNet

- ImageNet (and the ImageNet Large-Scale Visual Recognition Challege, aka **ILSVRC**) has been key to training deep learning methods
  - J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, **ImageNet: A Large-Scale Hierarchical Image Database.** *CVPR*, 2009.
- **ILSVRC**: 1,000 object categories, each with ~700-1300 training images. Test set has 100 images per categories (100,000 total).
- Standard ILSVRC error metric: top-5 error
  - if the correct answer for a given test image is in the top 5 categories, your answer is judged to be correct

Performance improvements on ILSVRC

- ImageNet Large-Scale Visual Recognition Challenge
- Held from 2011-2017
- 1000 categories, 1000 training images per category
- Test performance on held-out test set of images



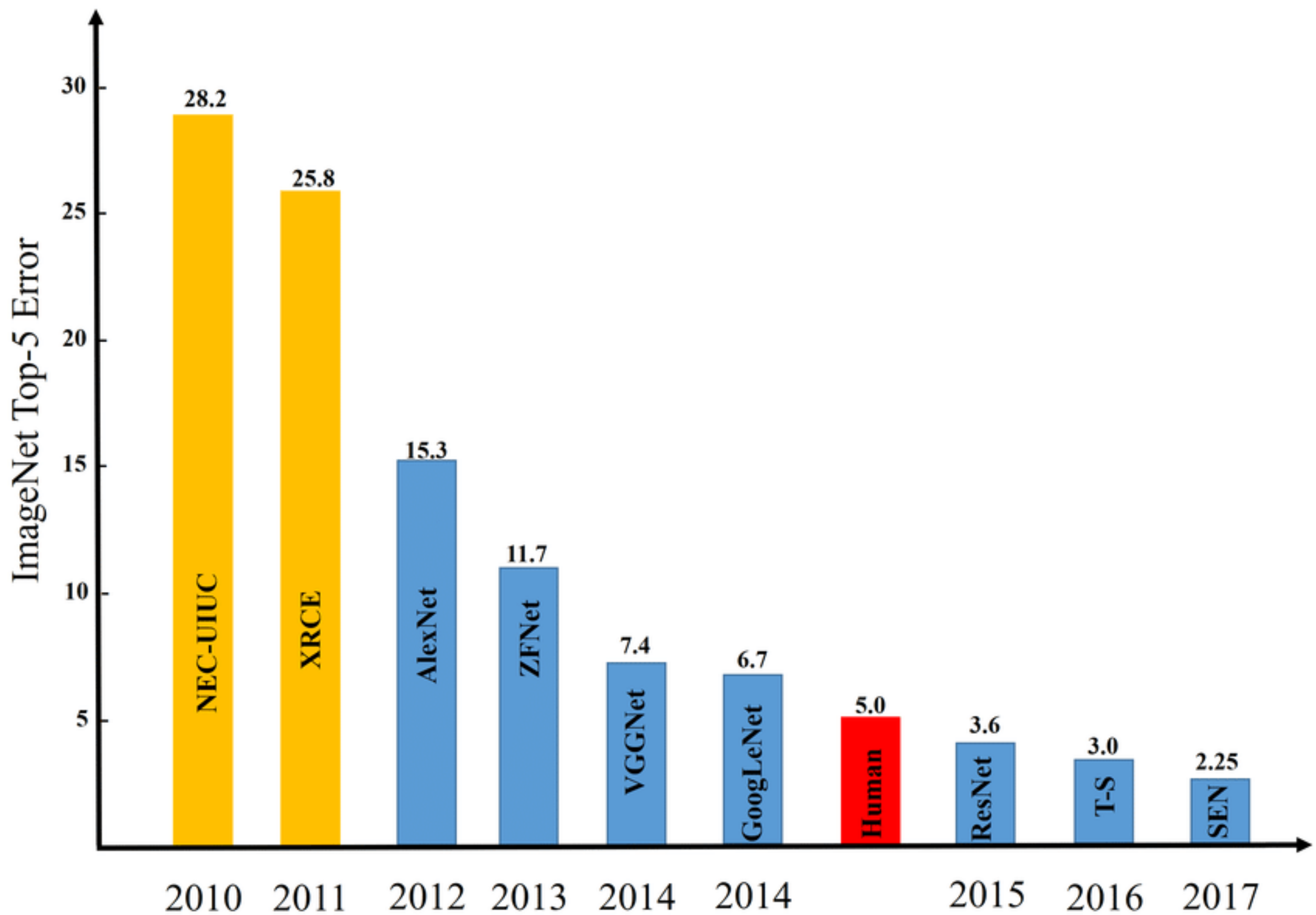


Image credit: Zaid Alyafeai, Lahouari Ghouti

Questions?