CS5670: Computer Vision

Noah Snavely

Convolutional neural networks

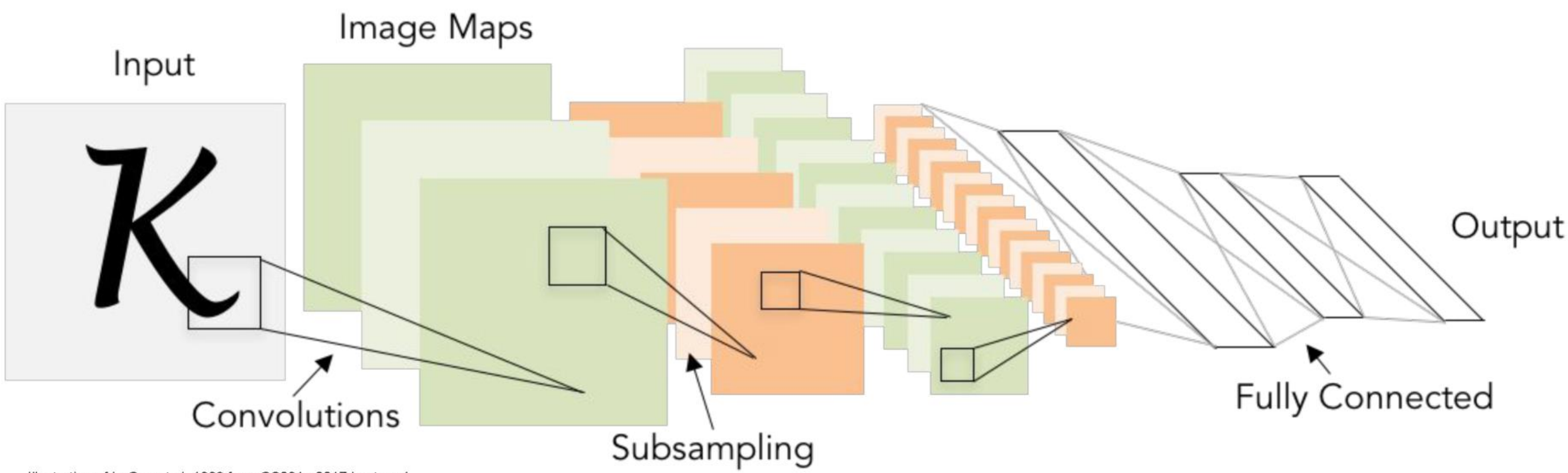


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

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

Announcements

- Project 5 released (demo today), due Friday, May 10
- Guest lecture from Dr. Jin Sun Monday on GANs
- Final exam in class, 5/6

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/

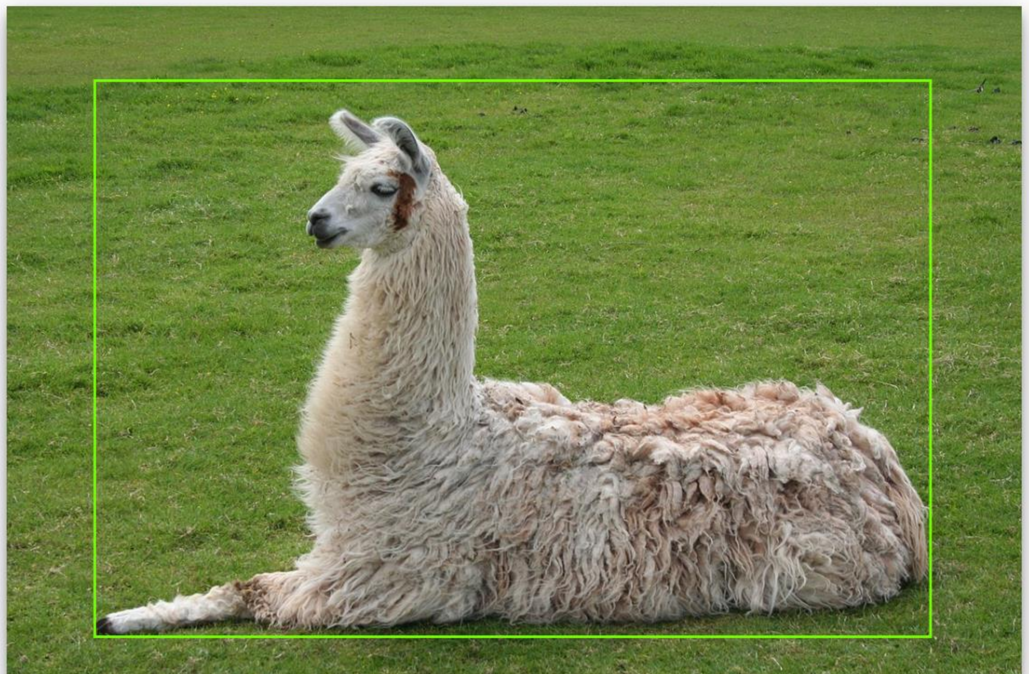
Last time

- Image classification
  - k-nearest neighbors
  - Linear classiciation
  - Score functions
  - Cross-entropy loss functions

Today

- Neural networks
- Convolutional neural networks

Alpaca or llama?



test_image_1.jpg

Alpaca 91%

Animal 53%

Recap: linear classification

- Have score function and loss function
  - Currently, score function is based on linear classifier

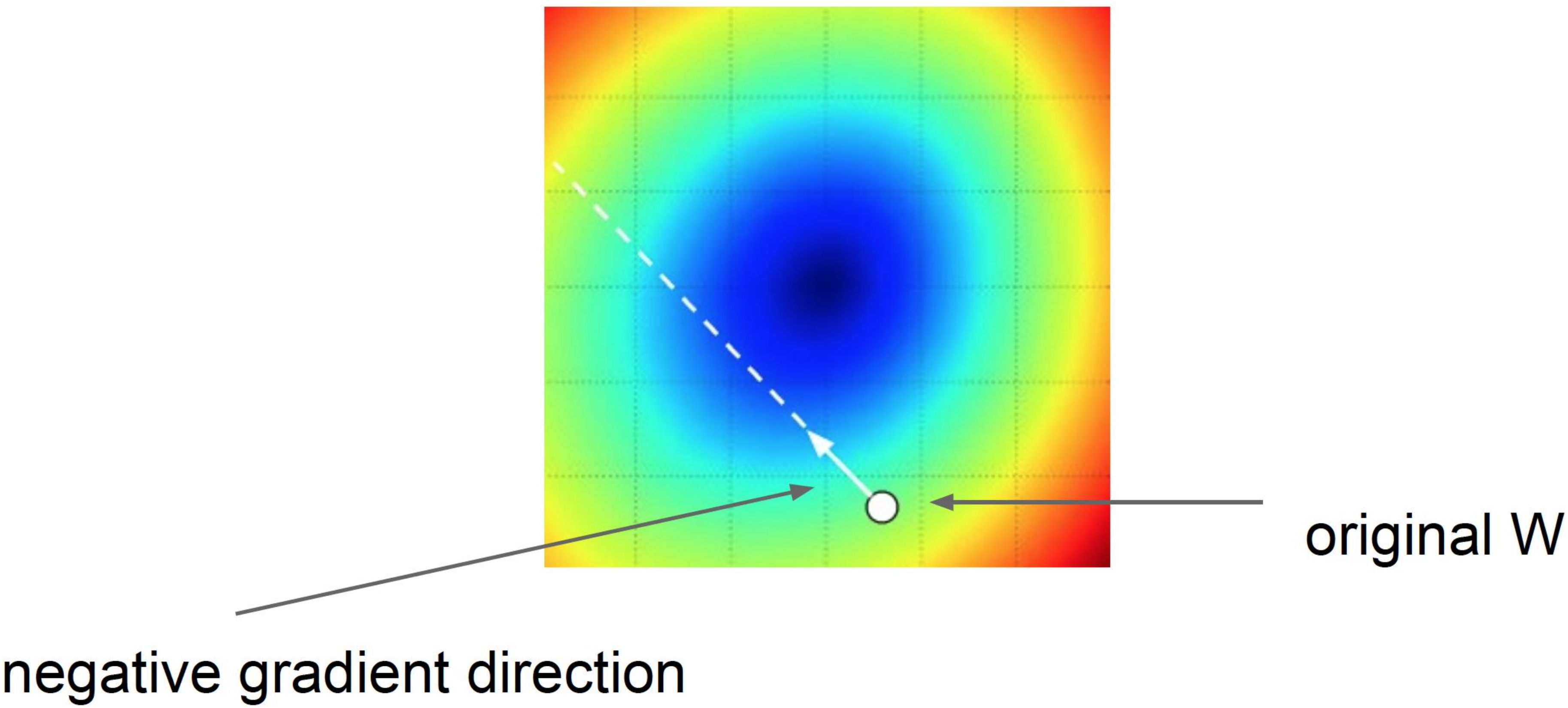
$$f(x, W) = Wx + b$$

- Find W and b to minimize loss, e.g. cross-entropy loss

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

Side note: 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

Neural networks

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

Neural networks

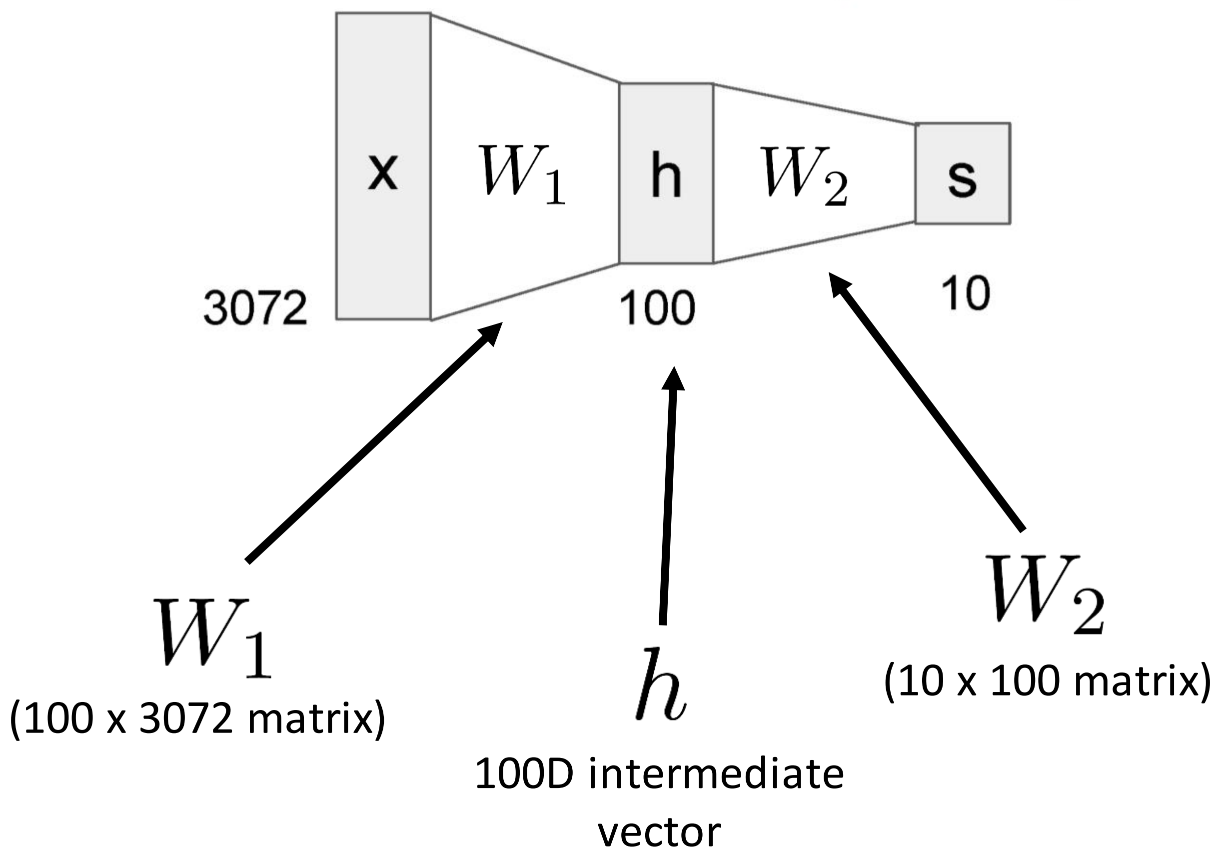
(**Before**) Linear score function: $f = Wx$

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

Neural networks

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

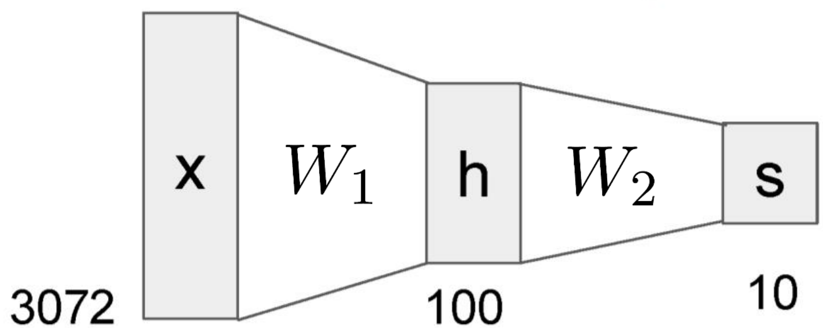
(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$



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

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

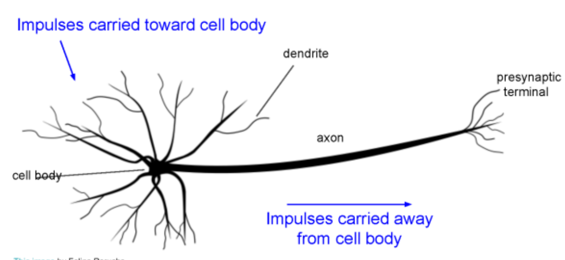
$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

Neural networks

- Very coarse generalization:
  - Linear functions chained together and separated by non-linearities (*activation functions*), e.g. “max”

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

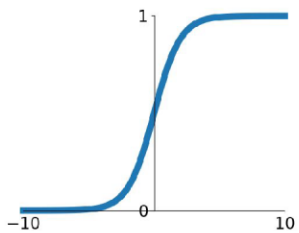
  - Why separate linear functions with non-linear functions?
  - *Very roughly* inspired by real neurons



Activation functions

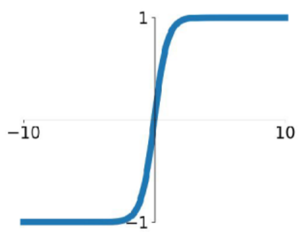
Sigmoid

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



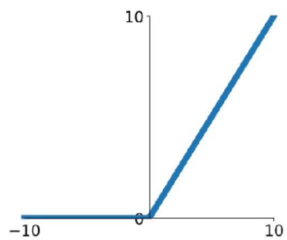
tanh

$\tanh(x)$



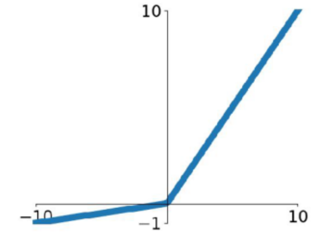
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

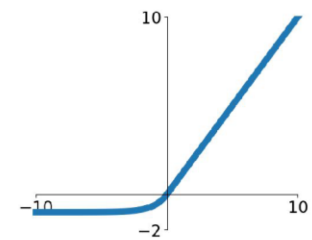


Maxout

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

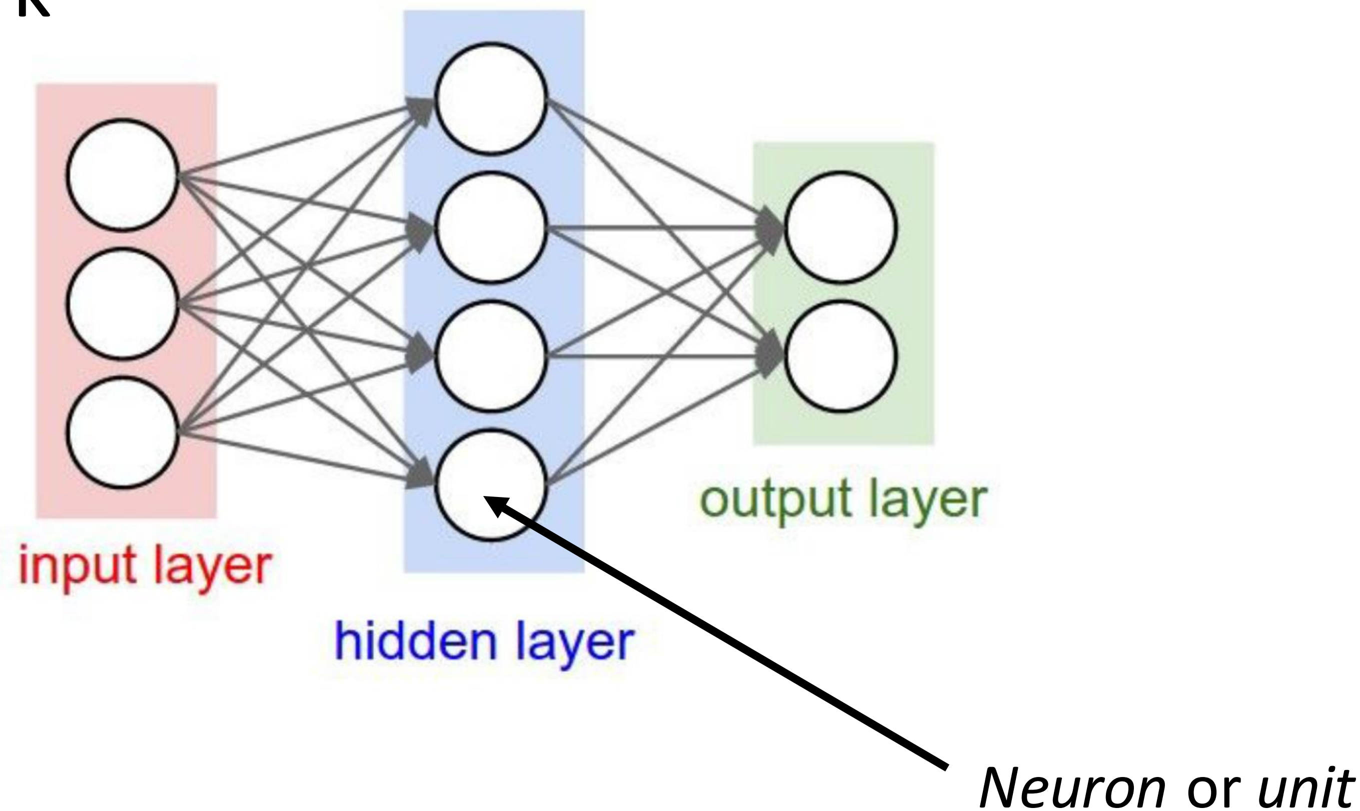
ELU

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

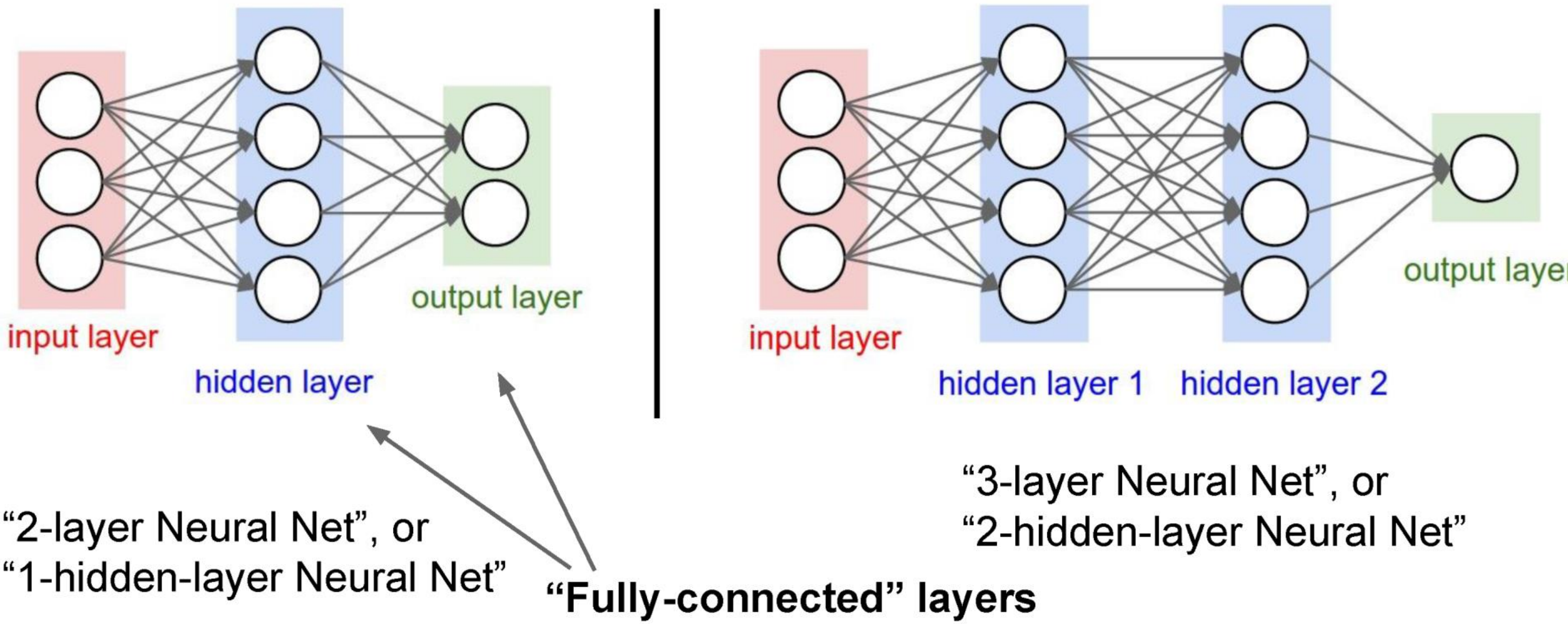


Neural network architecture

- Computation graph for a 2-layer neural network

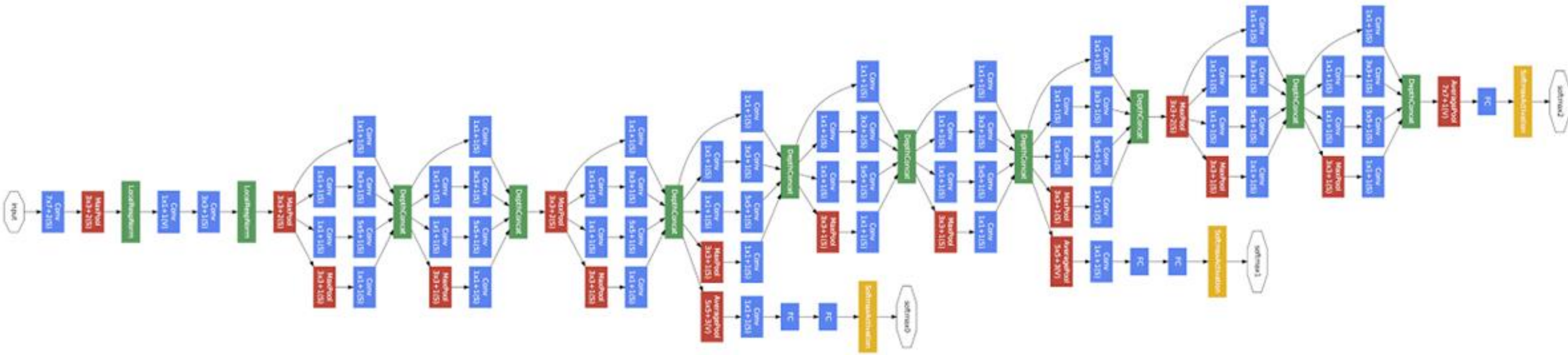


Neural networks: Architectures



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

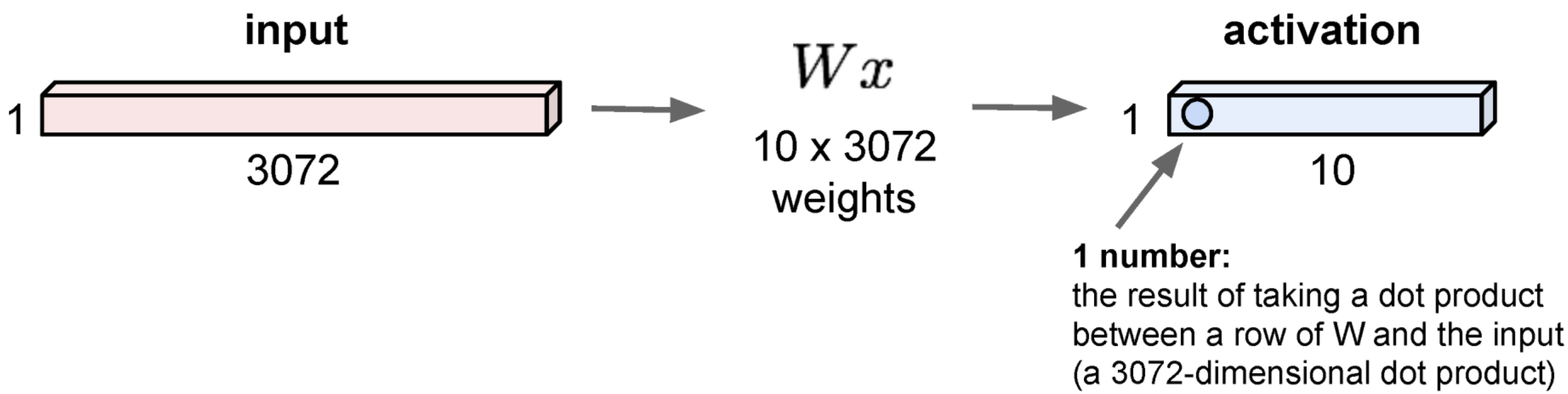
Deep neural network



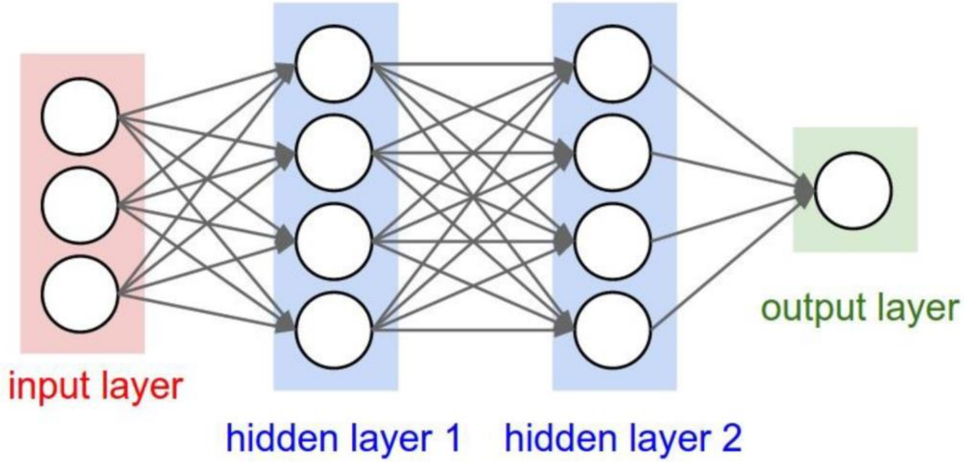
- *Inception* network (Szegedy et al, 2015)
- 22 layers

Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Example feed-forward computation of a neural network




```python
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

Summary

- We arrange neurons into fully-connected layers
- The abstraction of a **layer** has the nice property that it allows us to use efficient vectorized code (e.g. matrix multiplies)
- Neural networks are not really *neural*

Questions?

Convolutional neural networks

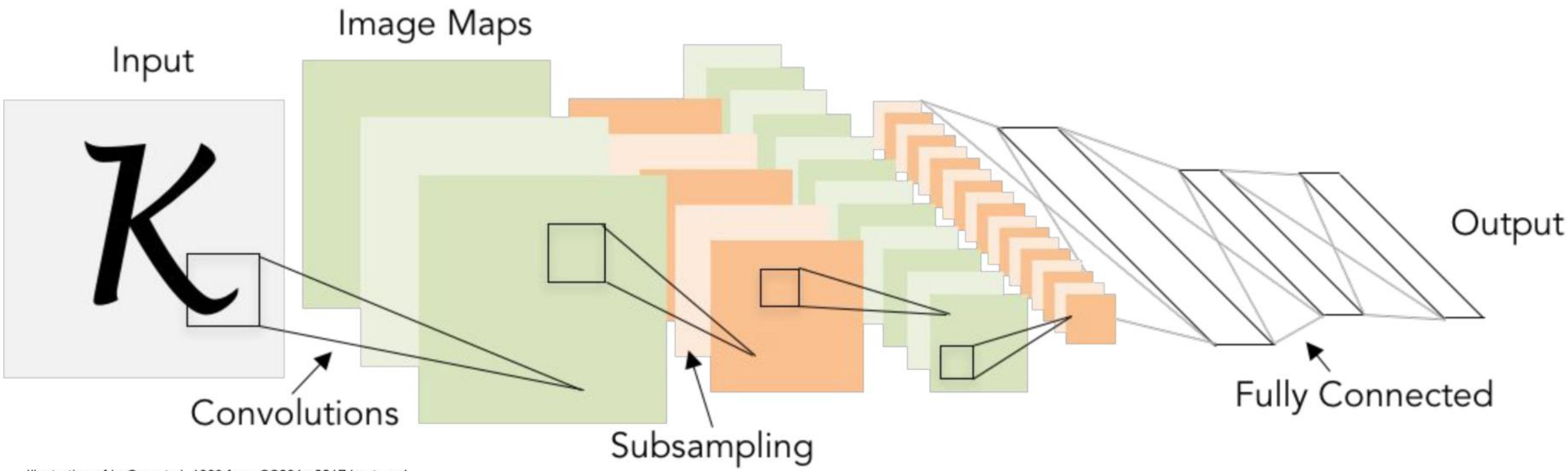


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

A bit of history...

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

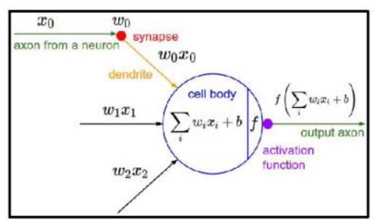
The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.

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

recognized
letters of the alphabet

update rule:

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



Frank Rosenblatt, ~1957: Perceptron

A bit of history…

[Hinton and Salakhutdinov 2006]

Reinvigorated research in Deep Learning

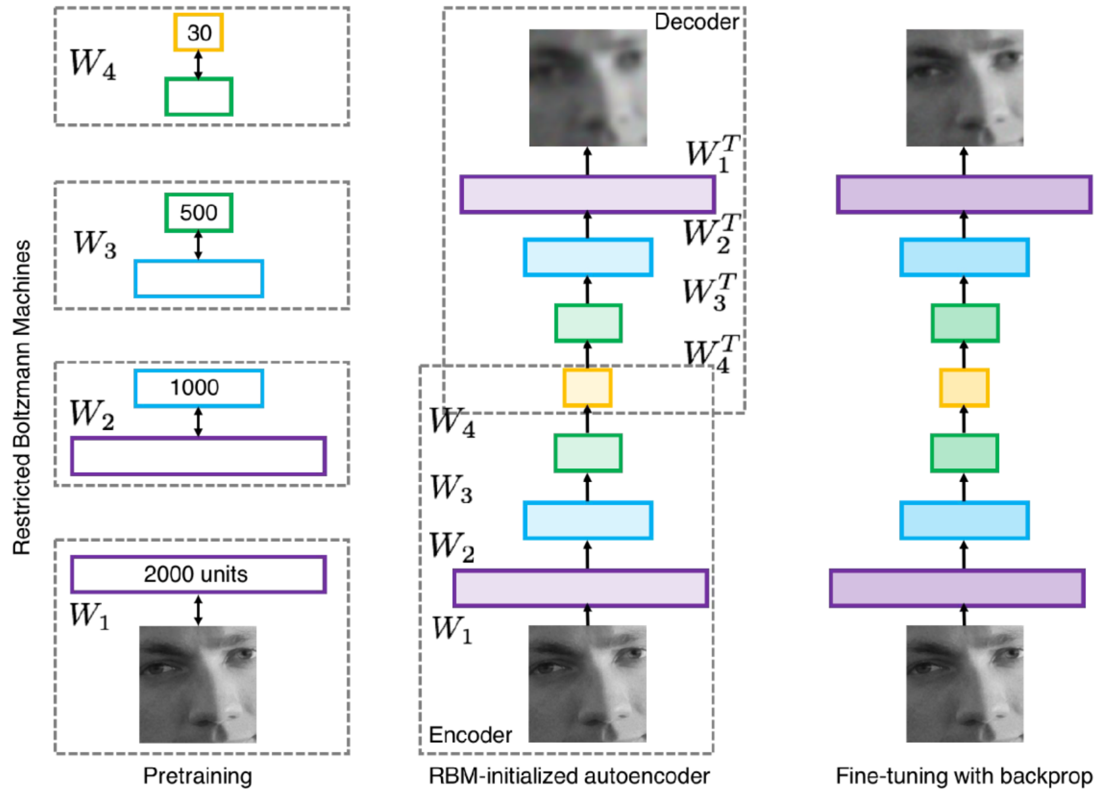


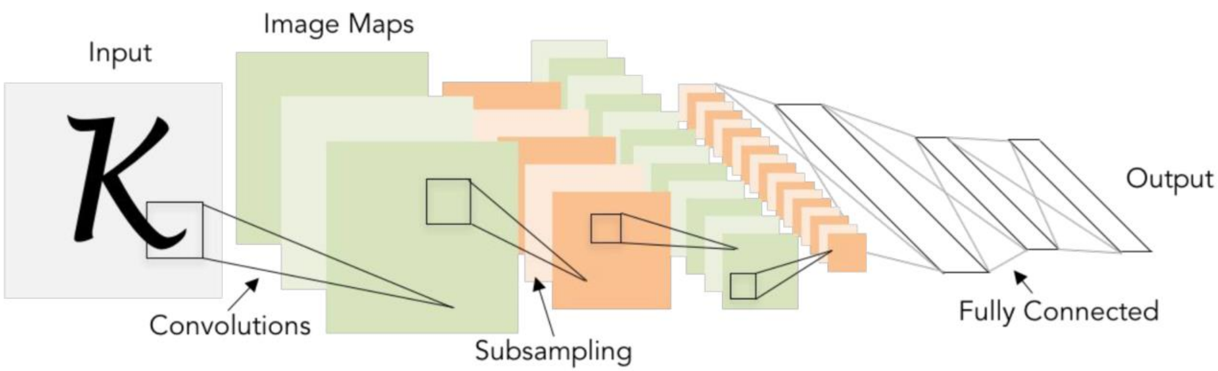
Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

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

A bit of history:

Gradient-based learning applied to document recognition

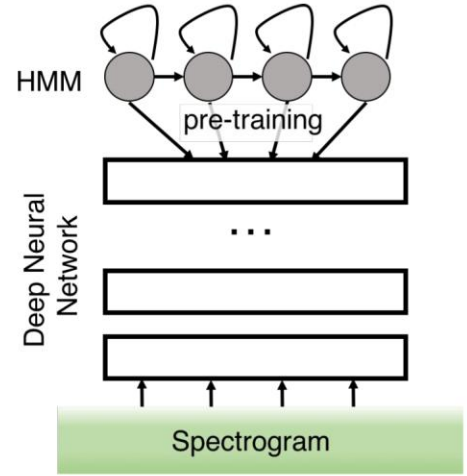
[LeCun, Bottou, Bengio, Haffner 1998]



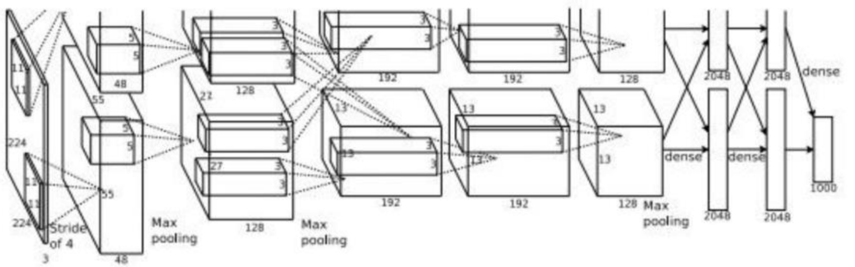
LeNet-5

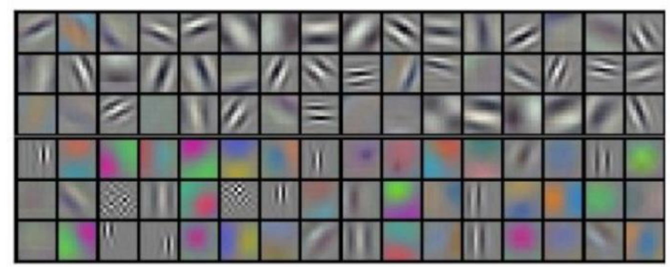
First strong results

Acoustic Modeling using Deep Belief Networks
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010
Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012



Imagenet classification with deep convolutional neural networks
Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012

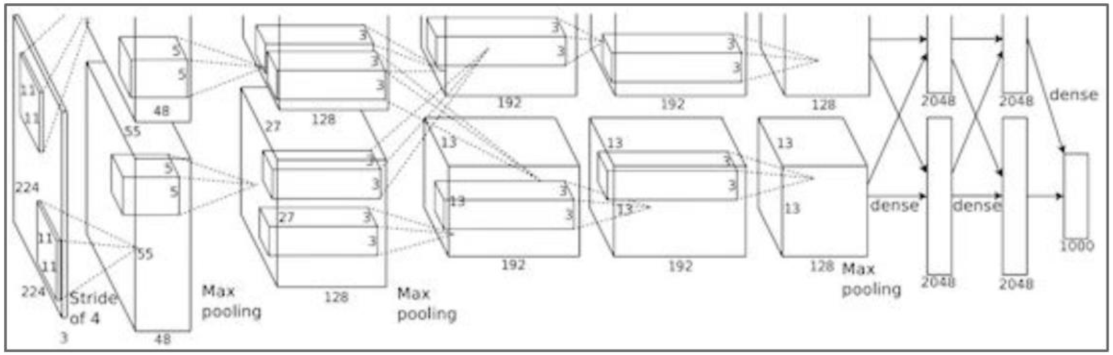




A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

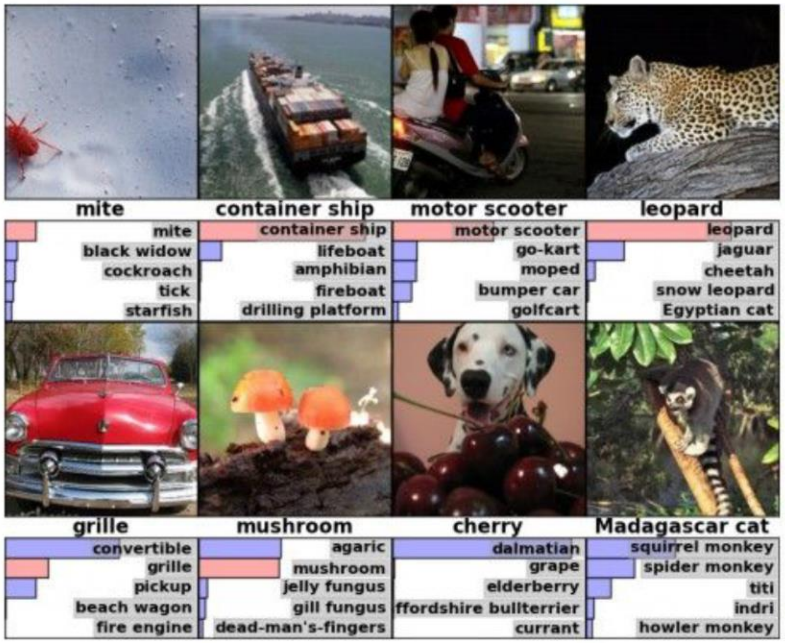
[Krizhevsky, Sutskever, Hinton, 2012]



“AlexNet”

Fast-forward to today: ConvNets are everywhere

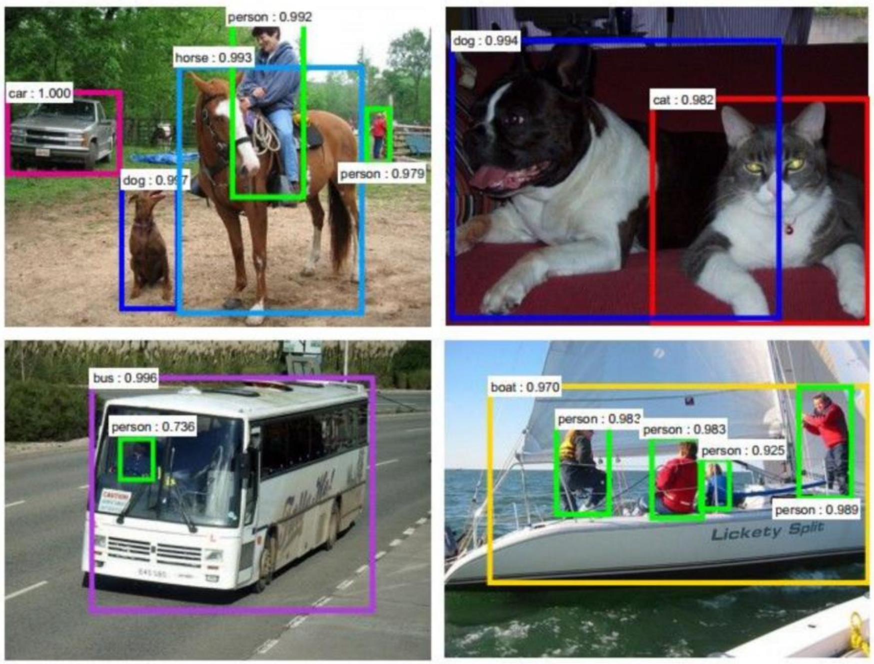
Classification



Retrieval

Fast-forward to today: ConvNets are everywhere

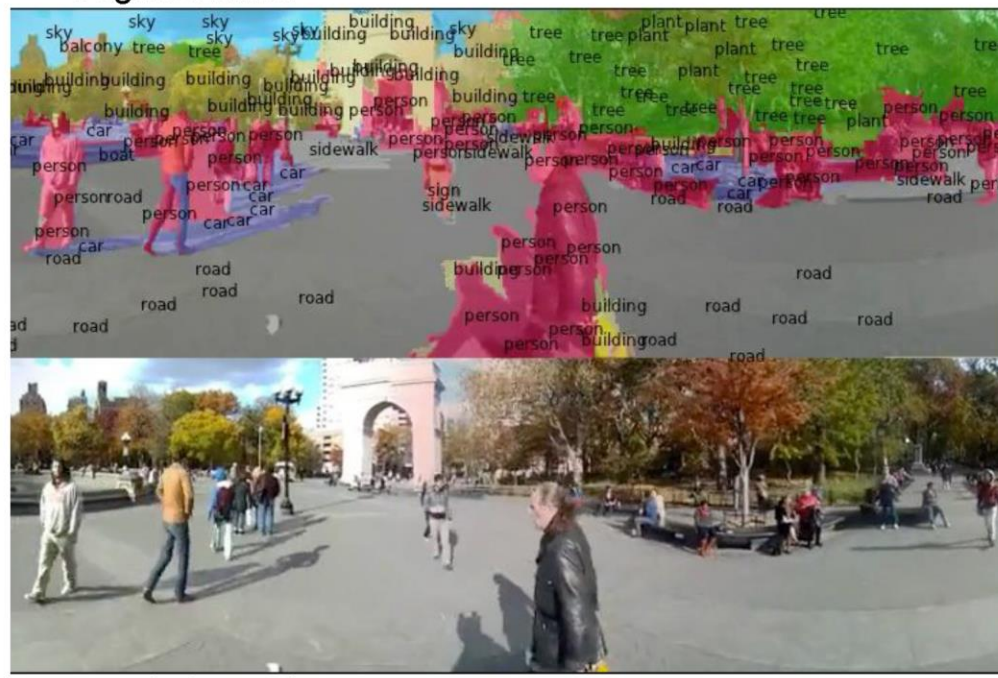
Detection



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

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

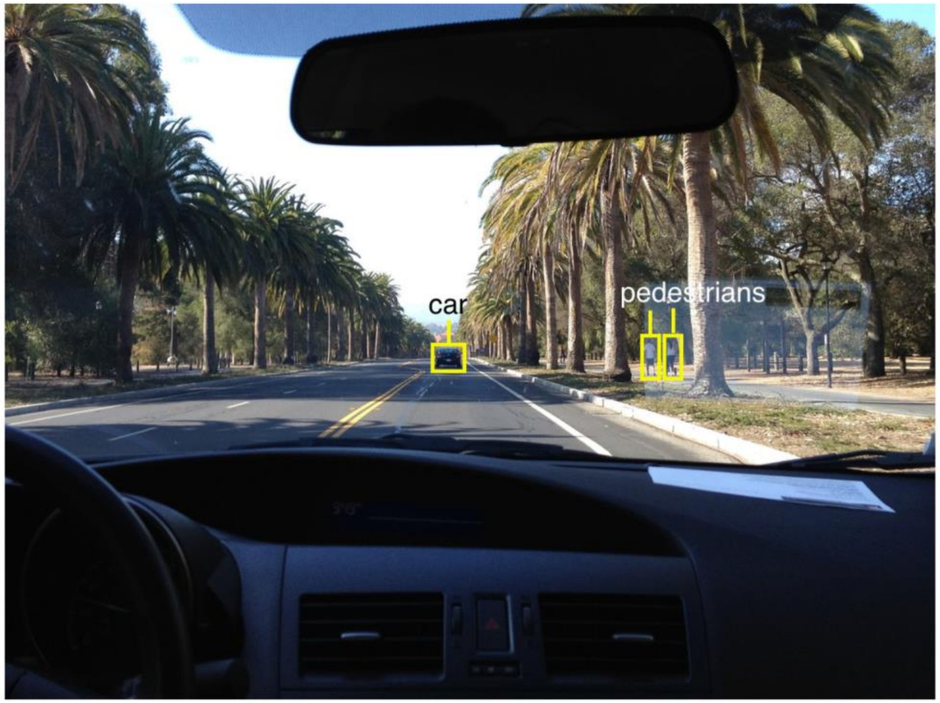
Segmentation



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere



self-driving cars

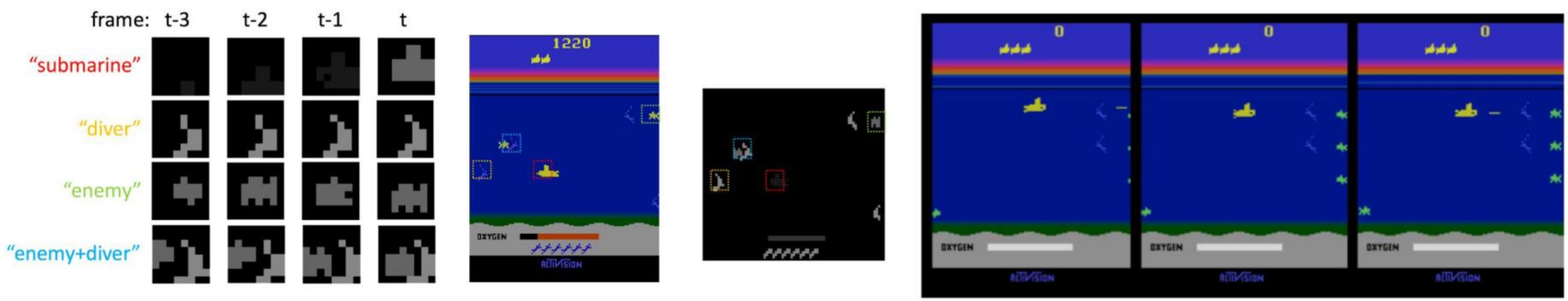
NVIDIA Tesla line
(these are the GPUs on rye01.stanford.edu)

Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

Fast-forward to today: ConvNets are everywhere

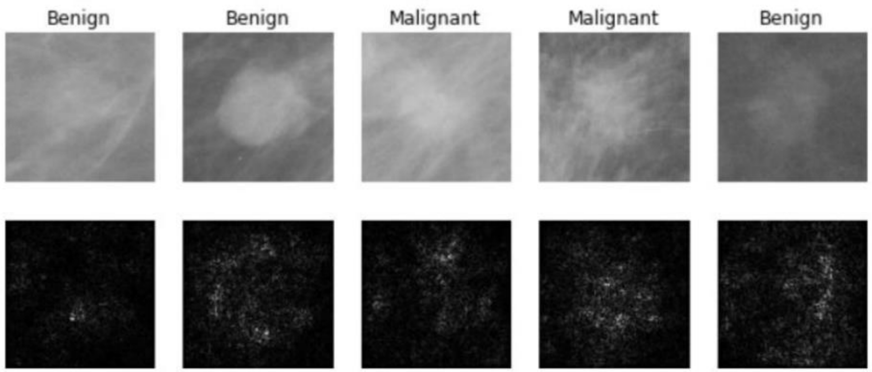
Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]



[Guo et al. 2014]

Fast-forward to today: ConvNets are everywhere



[Levy et al. 2016]

[Dieleman et al. 2014]

[Sermanet et al. 2011]
[Ciresan et al.]

Image Captioning

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

No errors

A white teddy bear sitting in the grass

A man riding a wave on top of a surfboard

Minor errors

A man in a baseball uniform throwing a ball

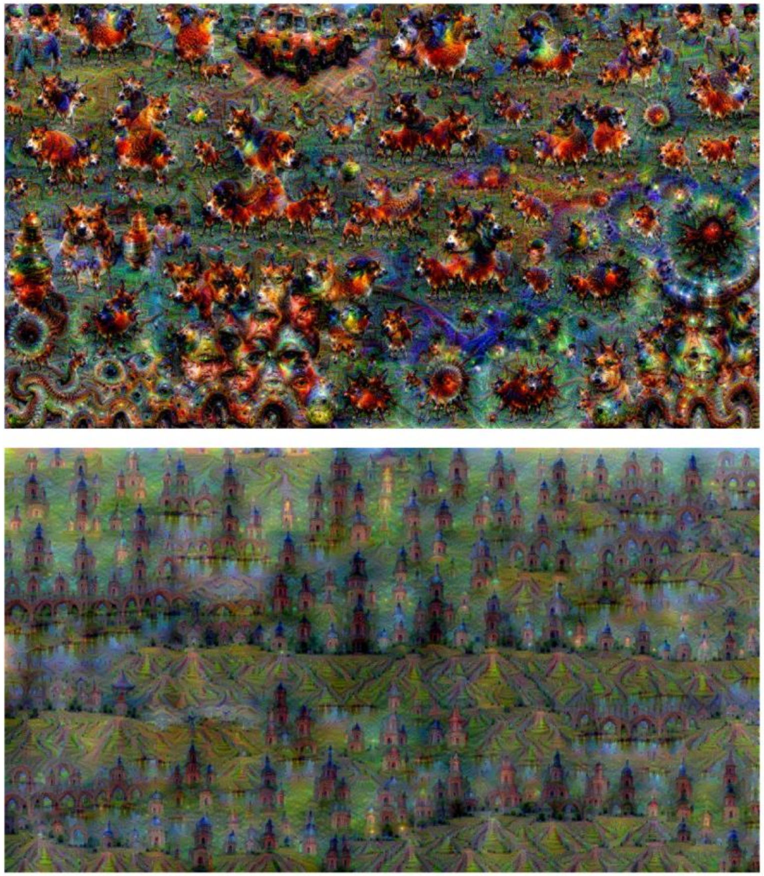
A cat sitting on a suitcase on the floor

Somewhat related

A woman is holding a cat in her hand

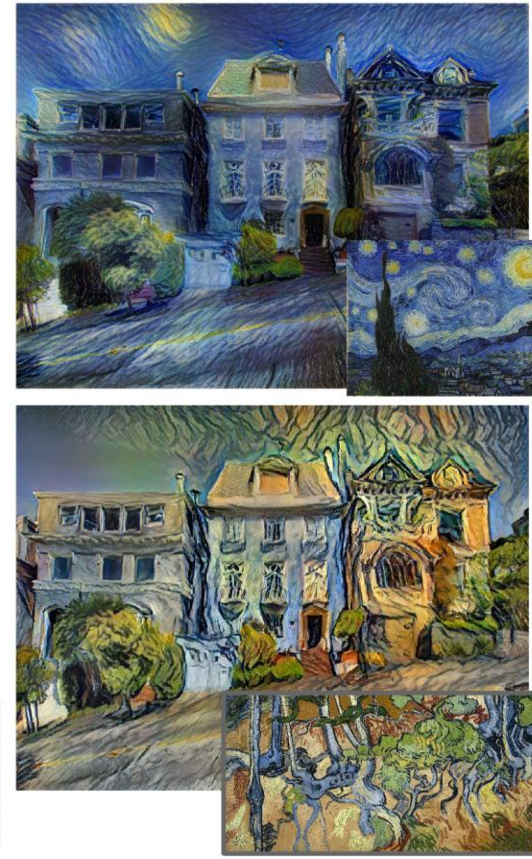
A woman standing on a beach holding a surfboard

Captions generated by Justin Johnson using Neuraltalk2



Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a blog post by Google Research.

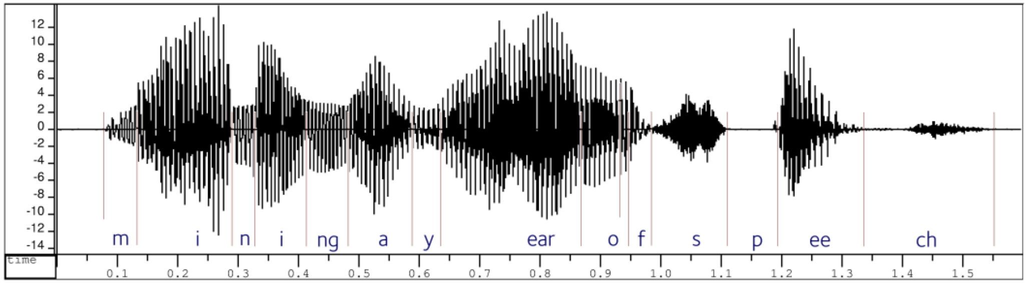
Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017;
reproduced with permission



Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

Convolutional neural networks

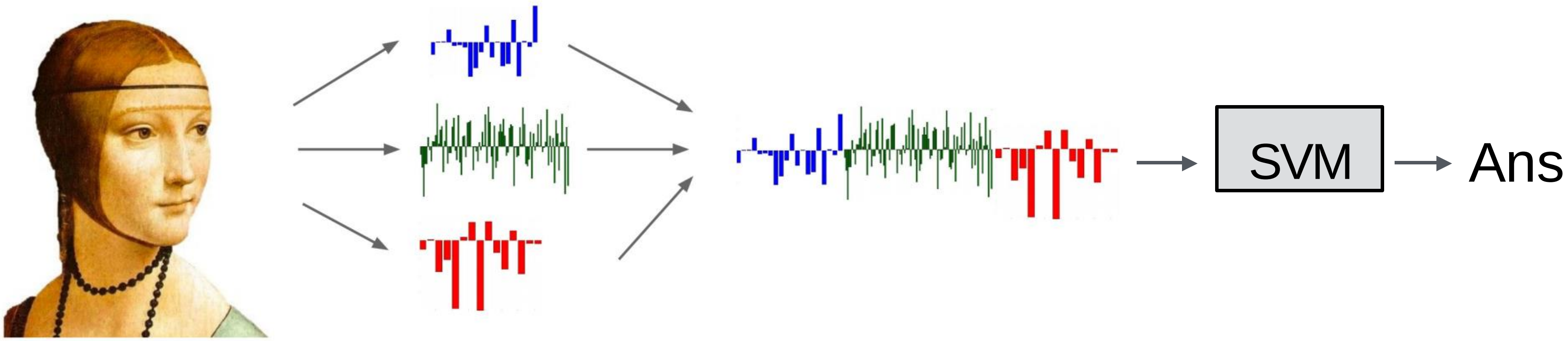
- Version of deep neural networks designed for signals
  - 1D signals (e.g., speech waveform)



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

Motivation – Feature Learning

Life Before Deep Learning



Input Pixels

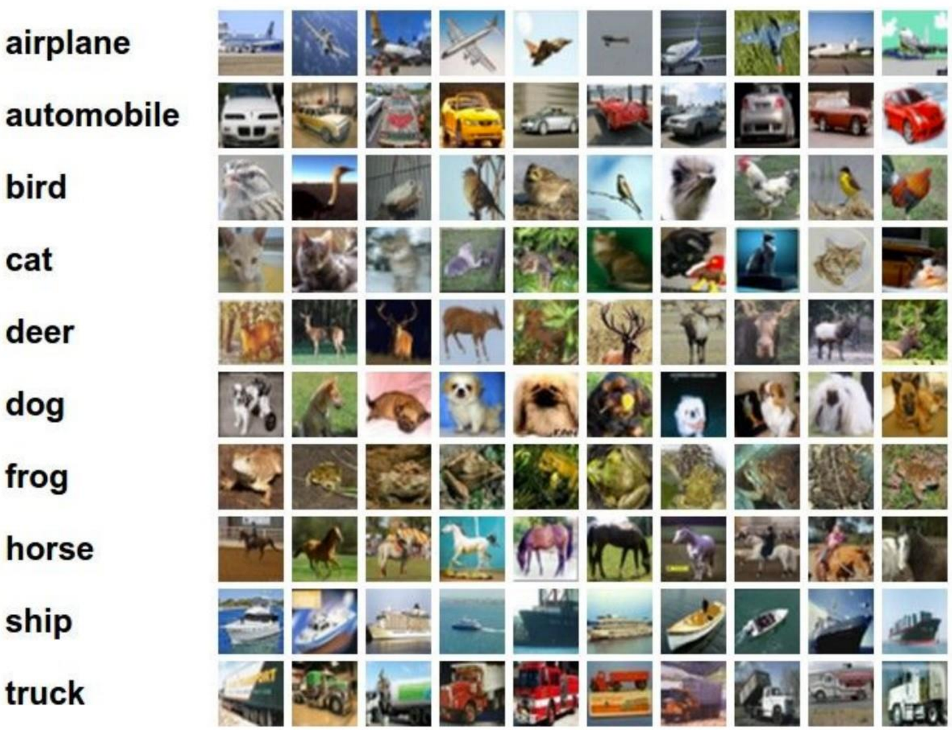
Extract Hand-Crafted Features

Concatenate into a vector ***x***

Linear Classifier

Figure: Karpathy 2016

Why use features?
Why not pixels?



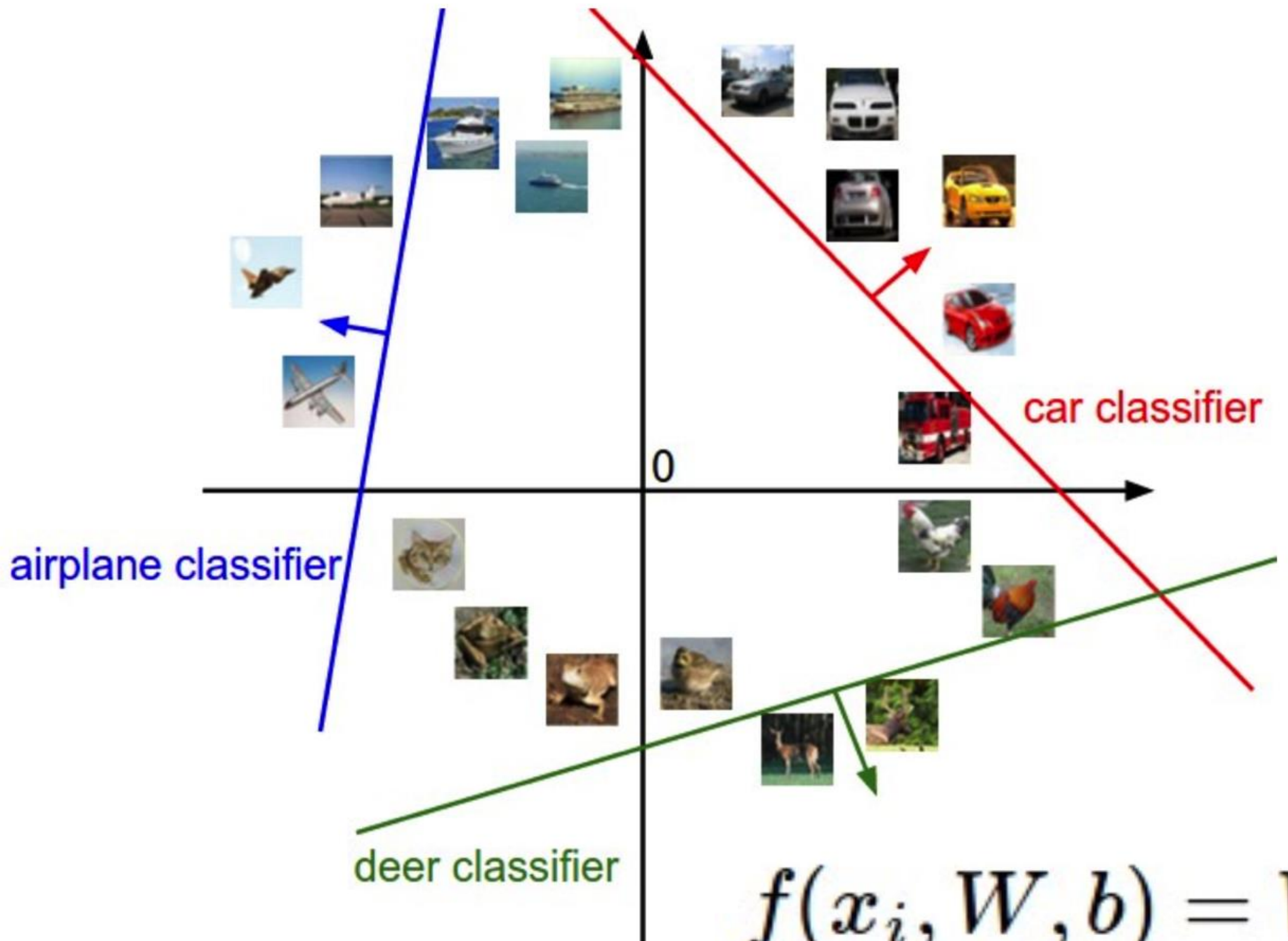
$$f(x_i, W, b) = Wx_i + b$$

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

(assuming x = pixels)

Slide from Karpathy 2016

Linearly separable classes



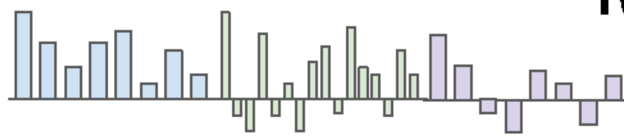
$$f(x_i, W, b) = Wx_i + b$$

Aside: Image Features

Class
scores

$$f(x) = Wx$$

Aside: Image Features

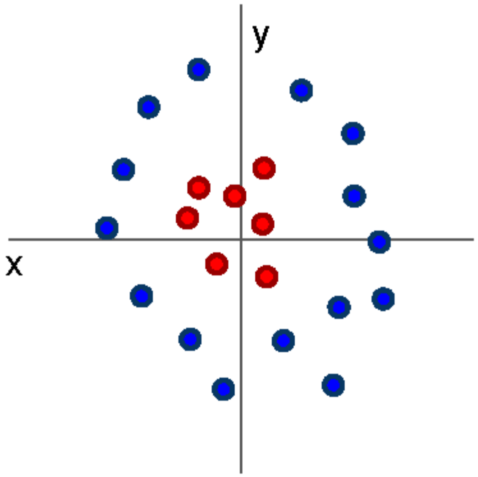


Feature Representation

$$f(x) = Wx$$

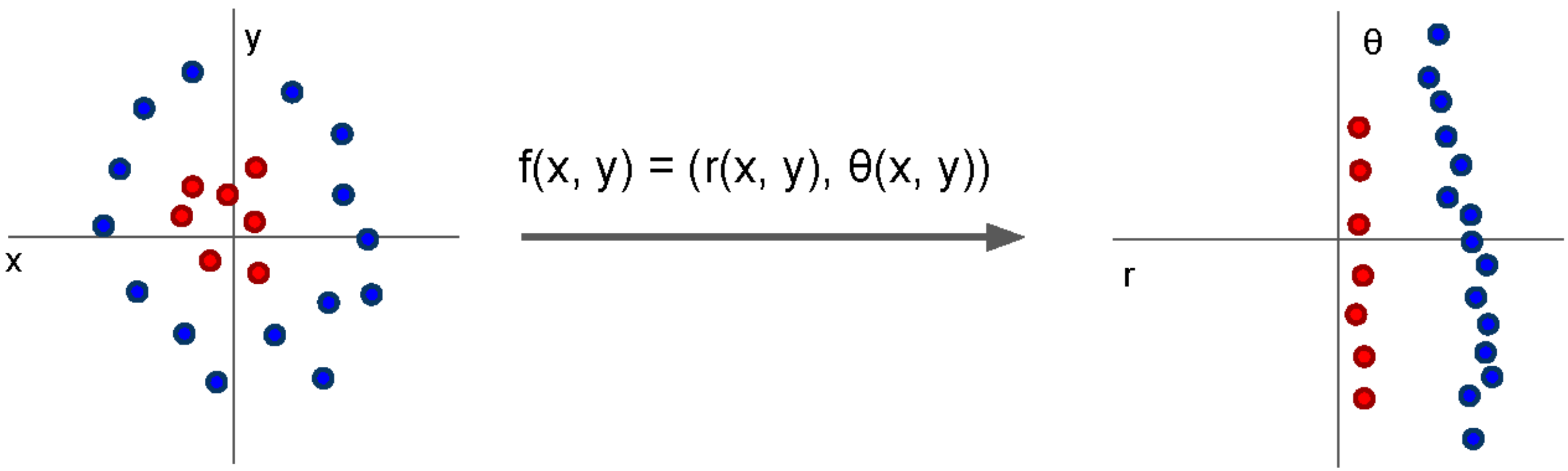
Class scores

Image Features: Motivation



Cannot separate red
and blue points with
linear classifier

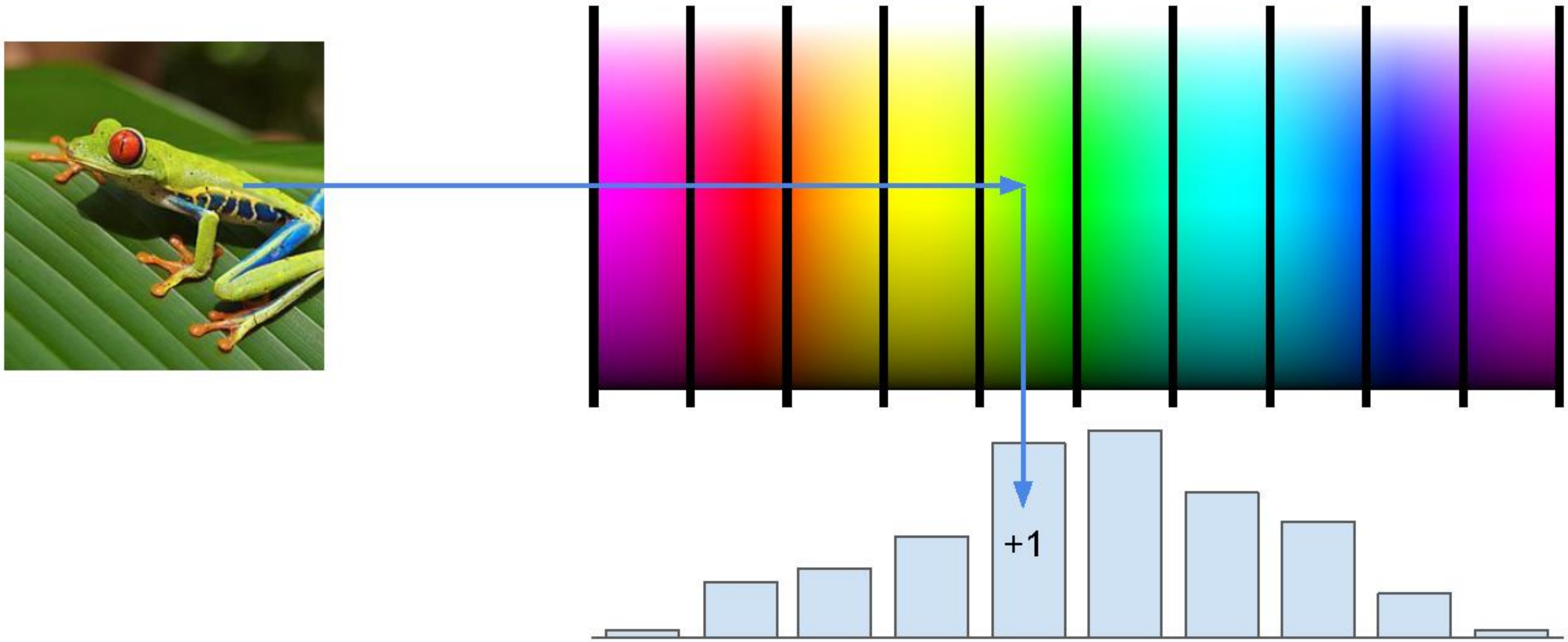
Image Features: Motivation



Cannot separate red and blue points with linear classifier

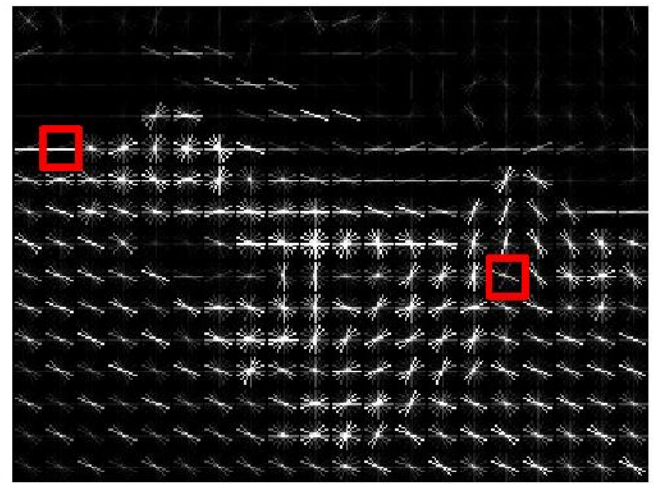
After applying feature transform, points can be separated by linear classifier

Example: Color Histogram



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

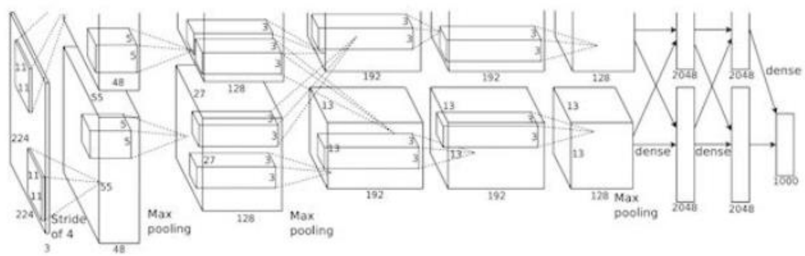
Image features vs ConvNets

Feature Extraction

f

10 numbers giving scores for classes

training



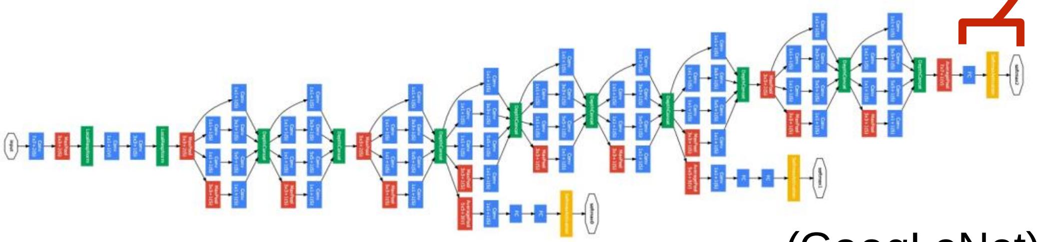
Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012.

10 numbers giving scores for classes

training

The last layer of (most) CNNs are linear classifiers

This piece is just a linear classifier



Ans

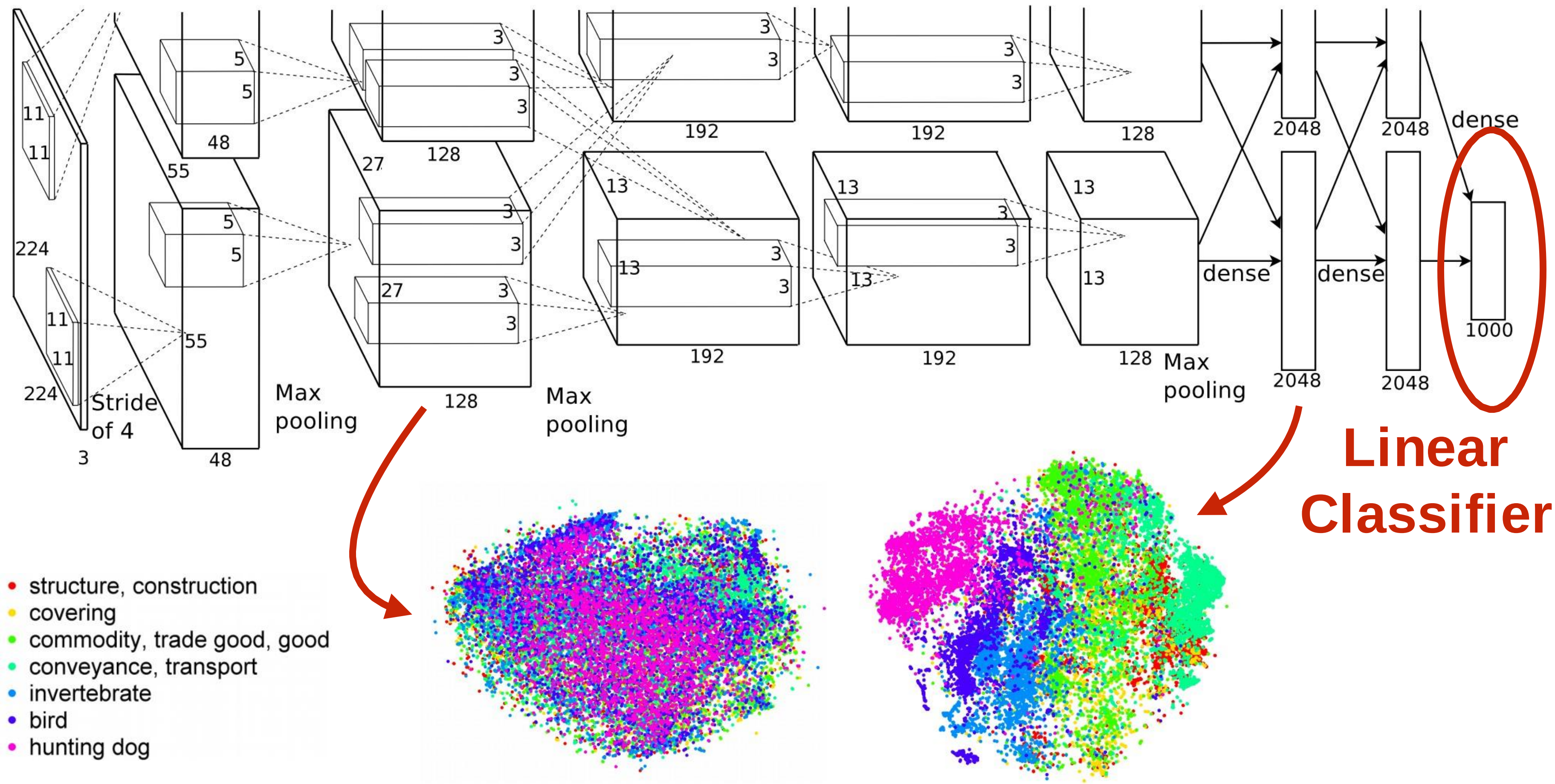
(GoogLeNet)

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

Example: Visualizing AlexNet in 2D with t-SNE

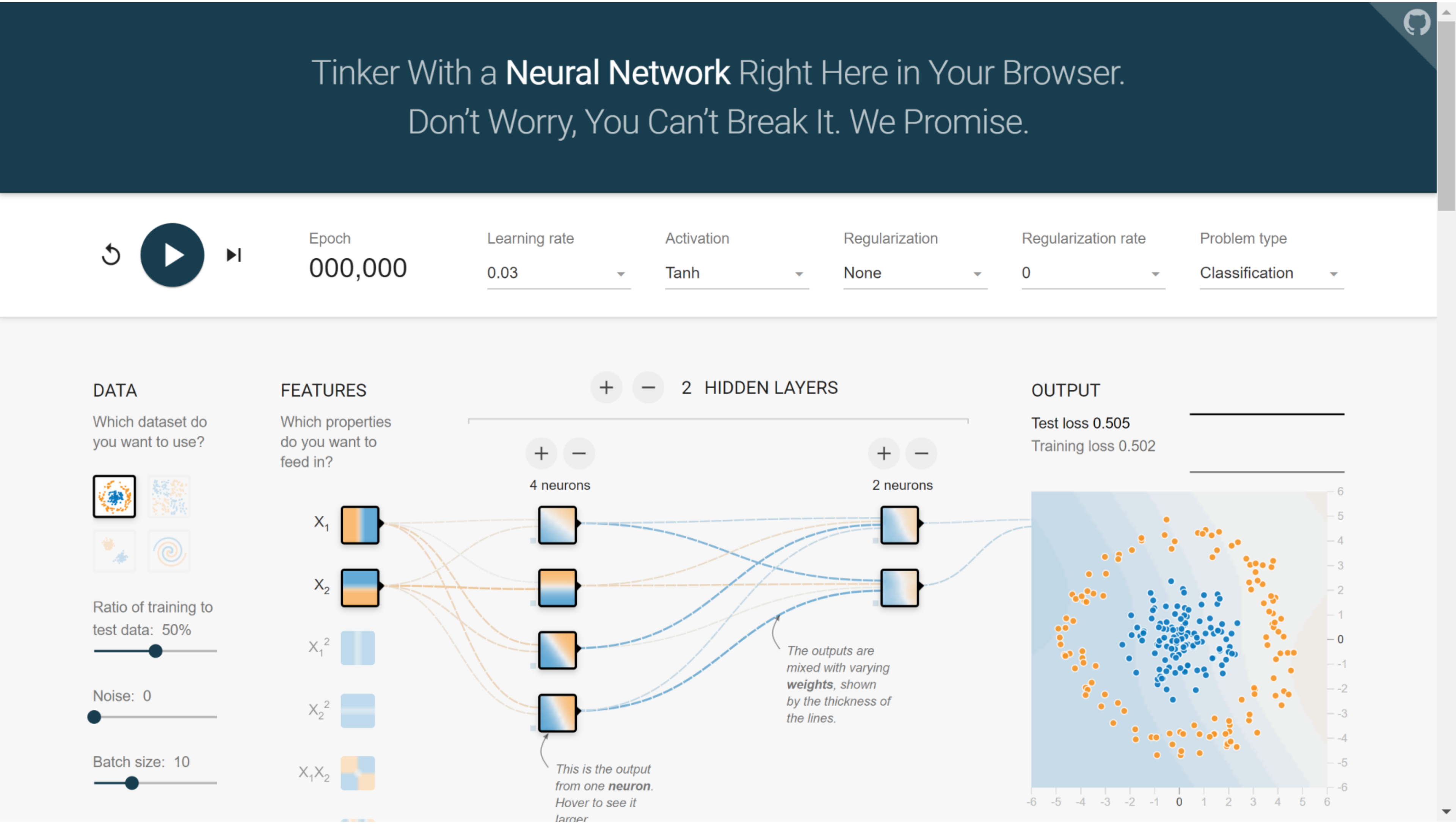


(c) DeCAF$_1$ (d) DeCAF$_6$

(2D visualization using t-SNE)

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

2D example: TensorFlow Playground



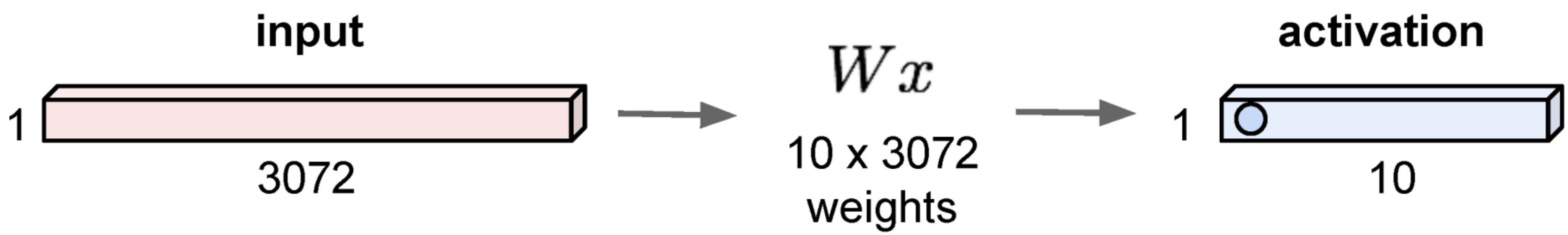
https://playground.tensorflow.org

Convolutional neural networks

- Layer types:
  - Fully-connected layer
  - *Convolutional layer*
  - Pooling layer

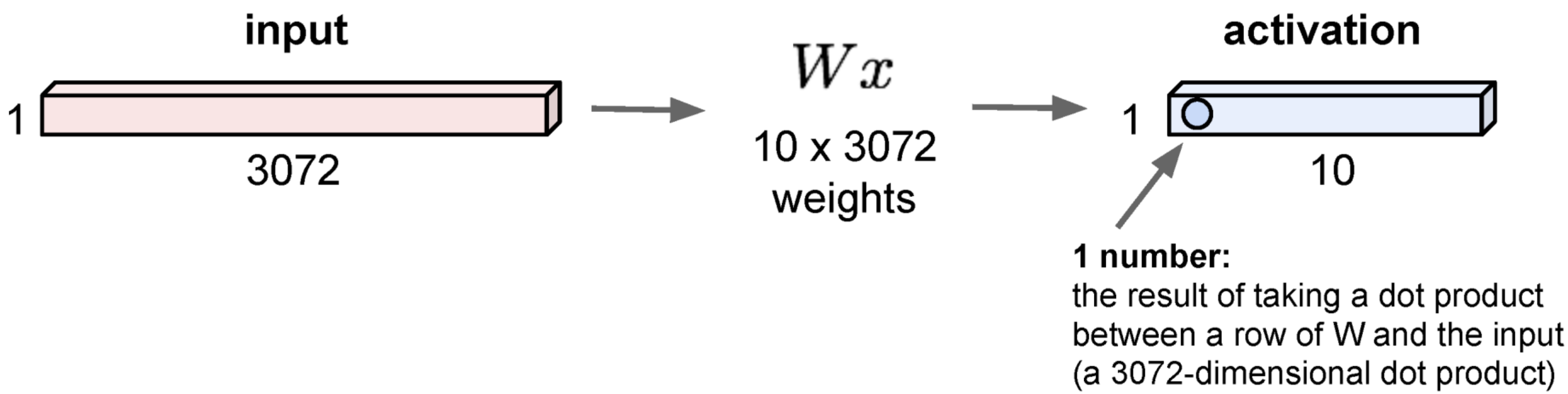
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Fully Connected Layer

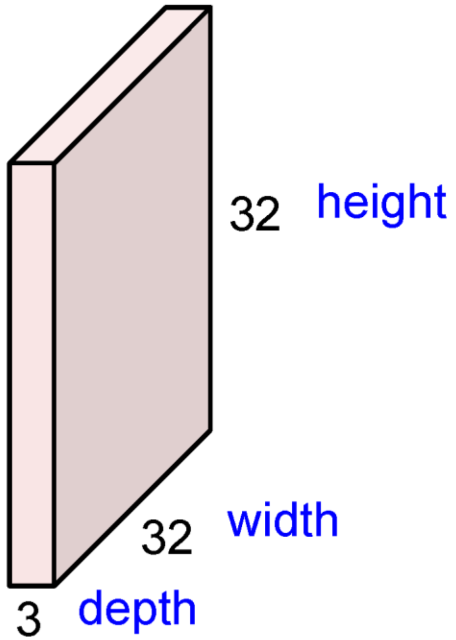
32x32x3 image -> stretch to 3072 x 1



Same as a linear classifer!

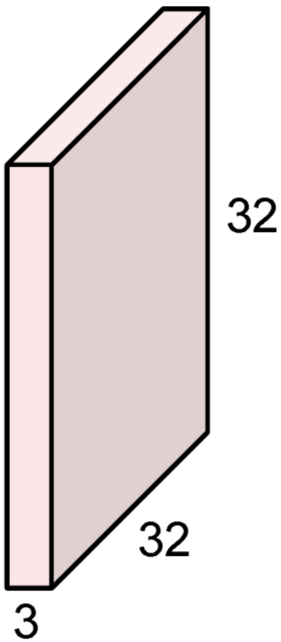
Convolution Layer

32x32x3 image -> preserve spatial structure

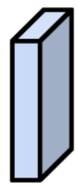


Convolution Layer

32x32x3 image



5x5x3 filter

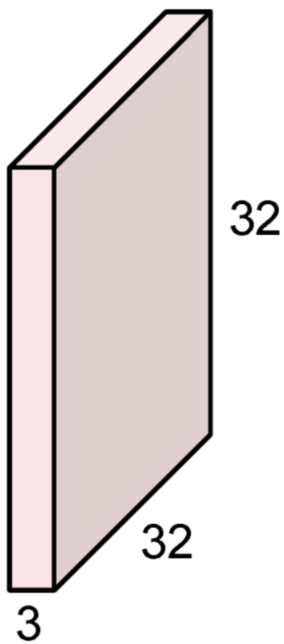


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

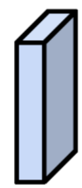
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



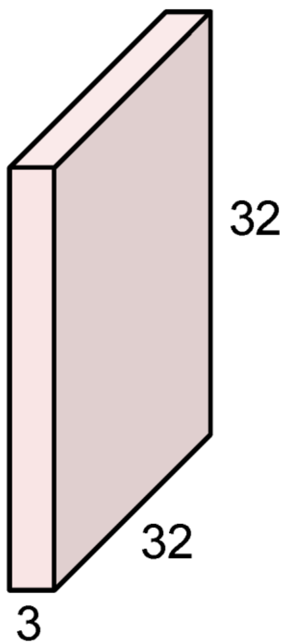
5x5x3 filter



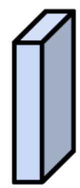
Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

32x32x3 image



5x5x3 filter



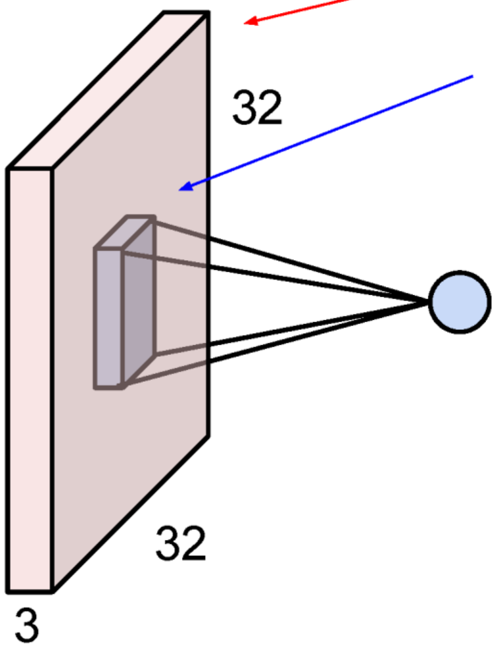
Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

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

Convolution Layer

32x32x3 image

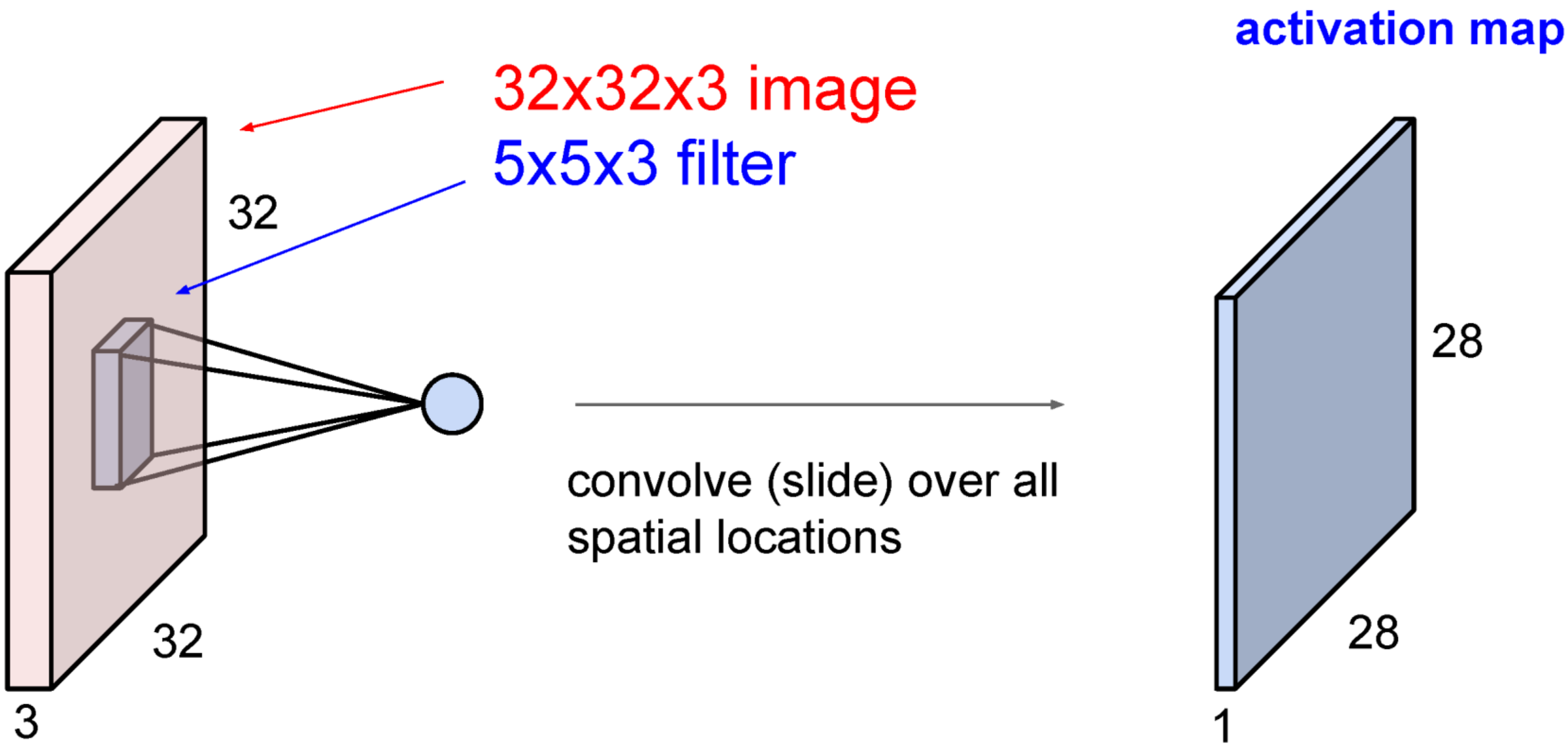
5x5x3 filter w



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

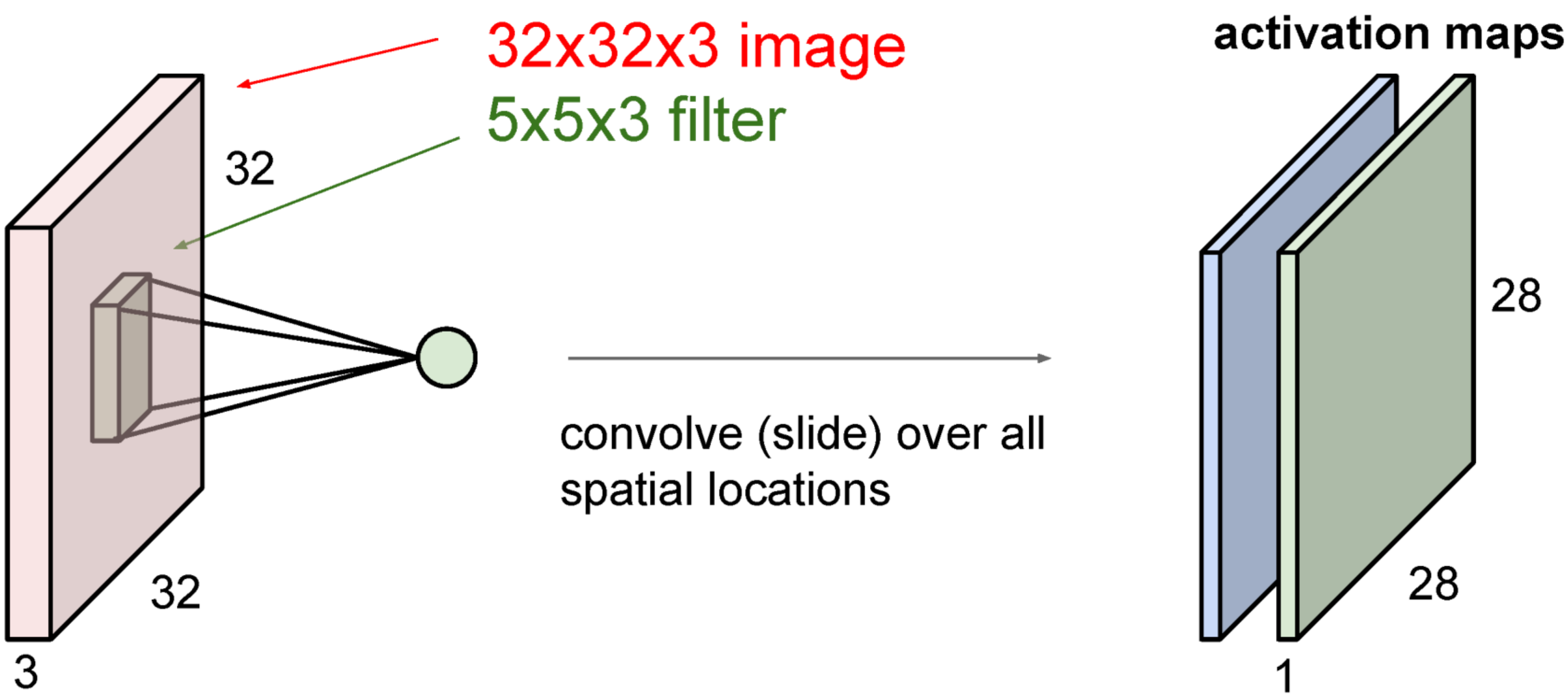
$$w^T x + b$$

Convolution Layer

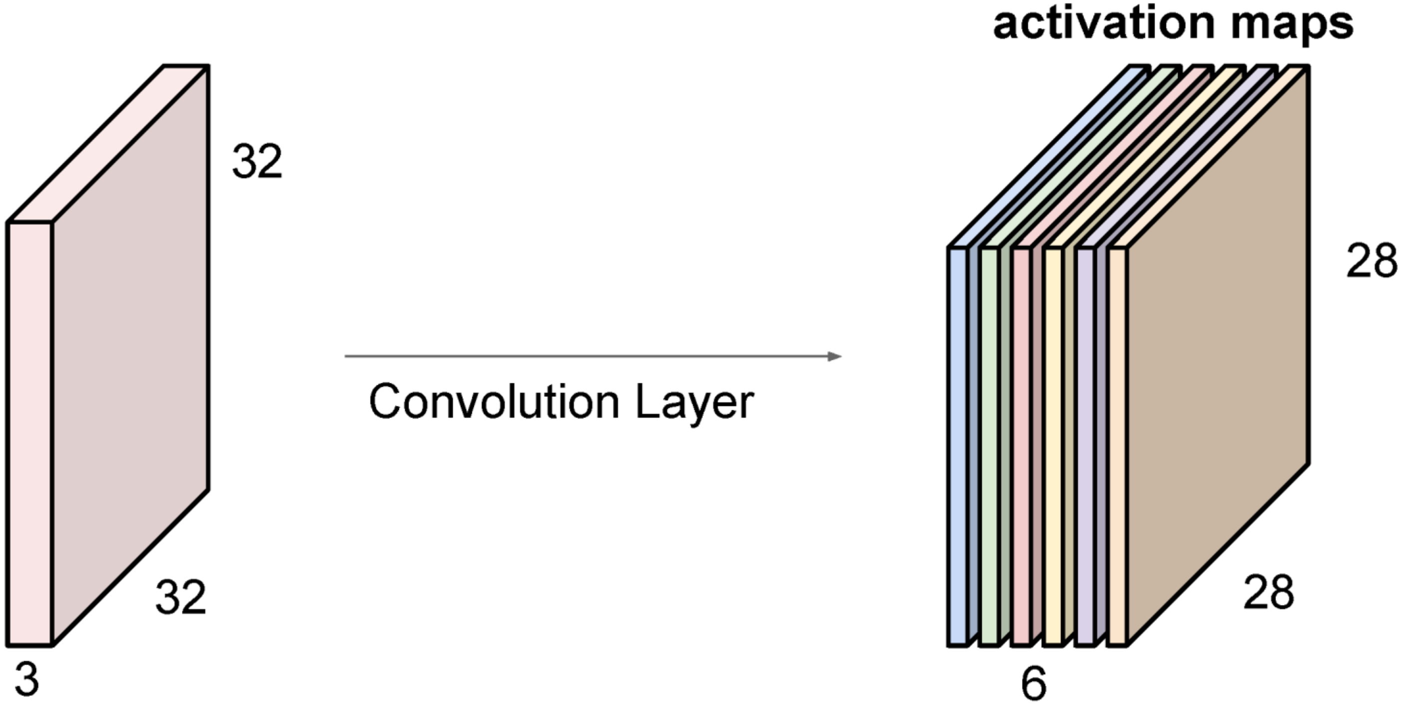


Convolution Layer

consider a second, green filter



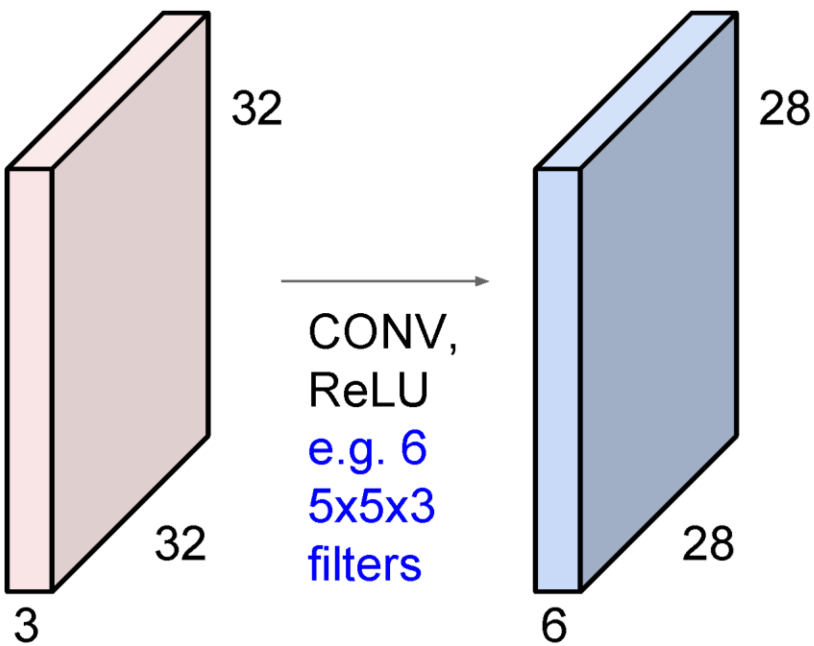
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



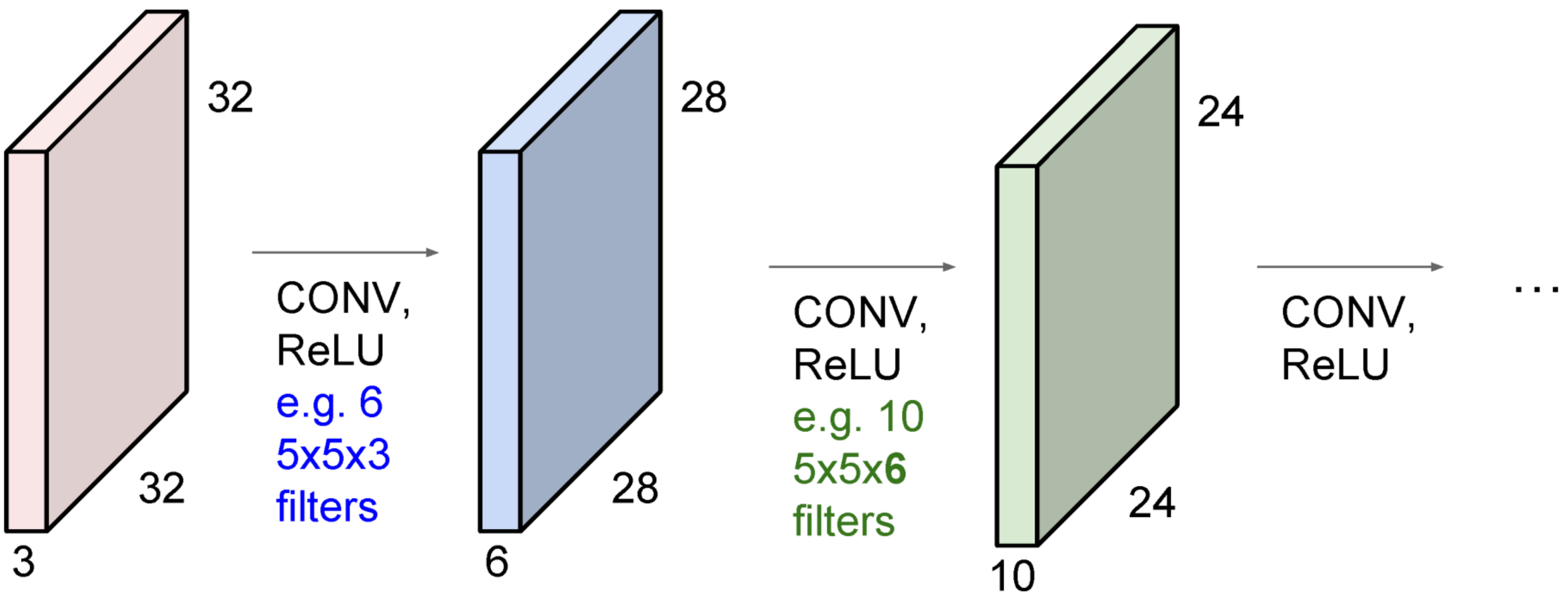
We stack these up to get a "new image" of size 28x28x6!

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

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



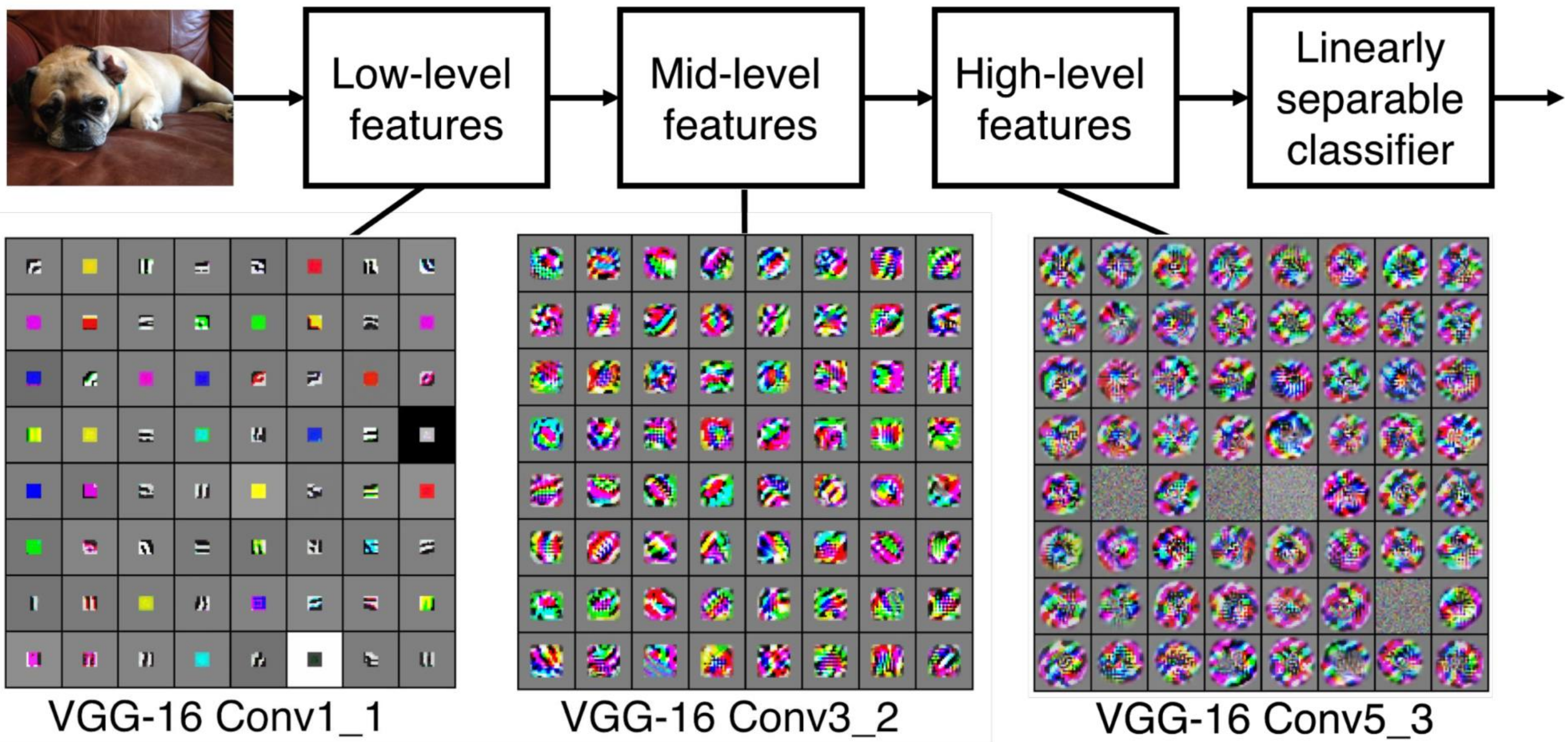
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



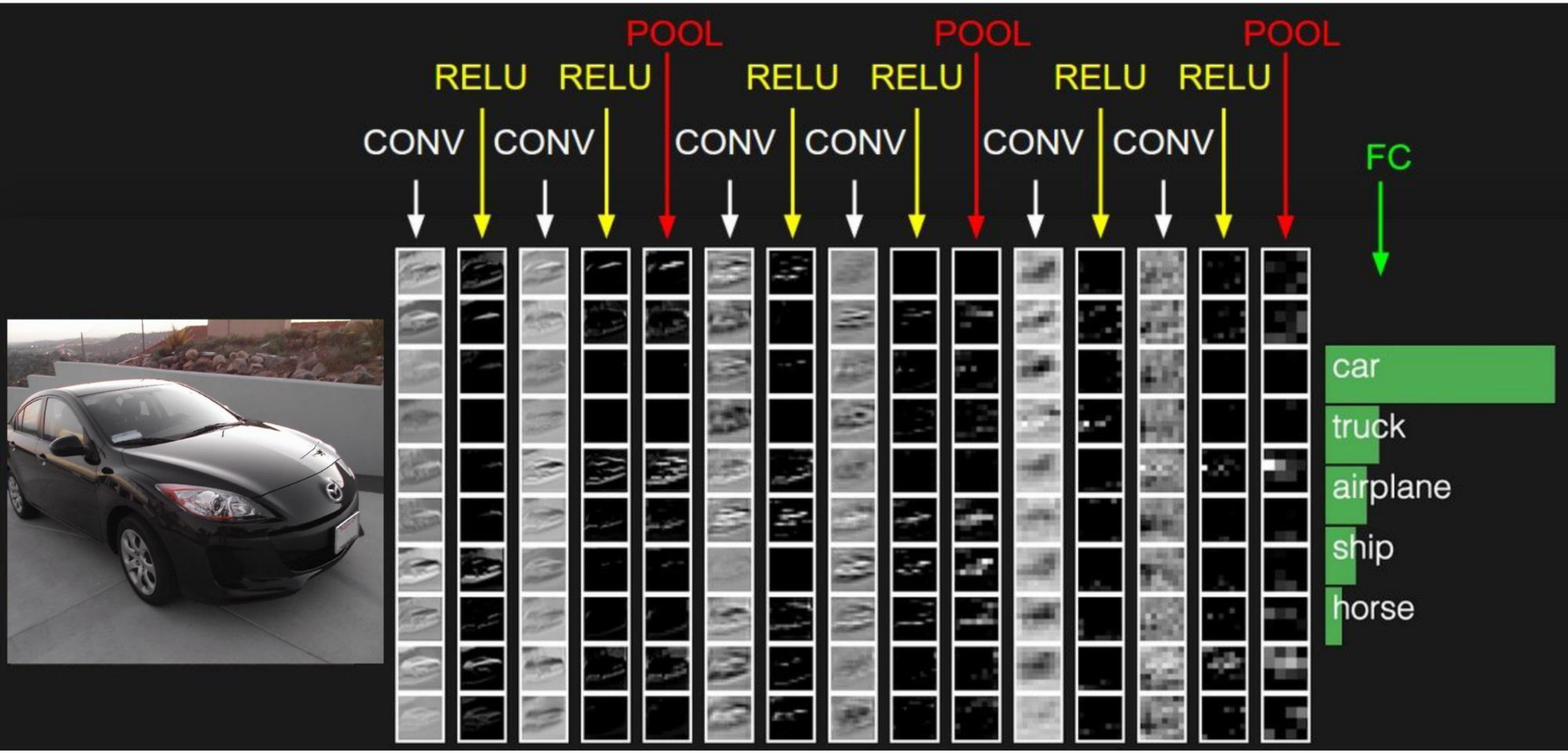
Preview

[Zeiler and Fergus 2013]

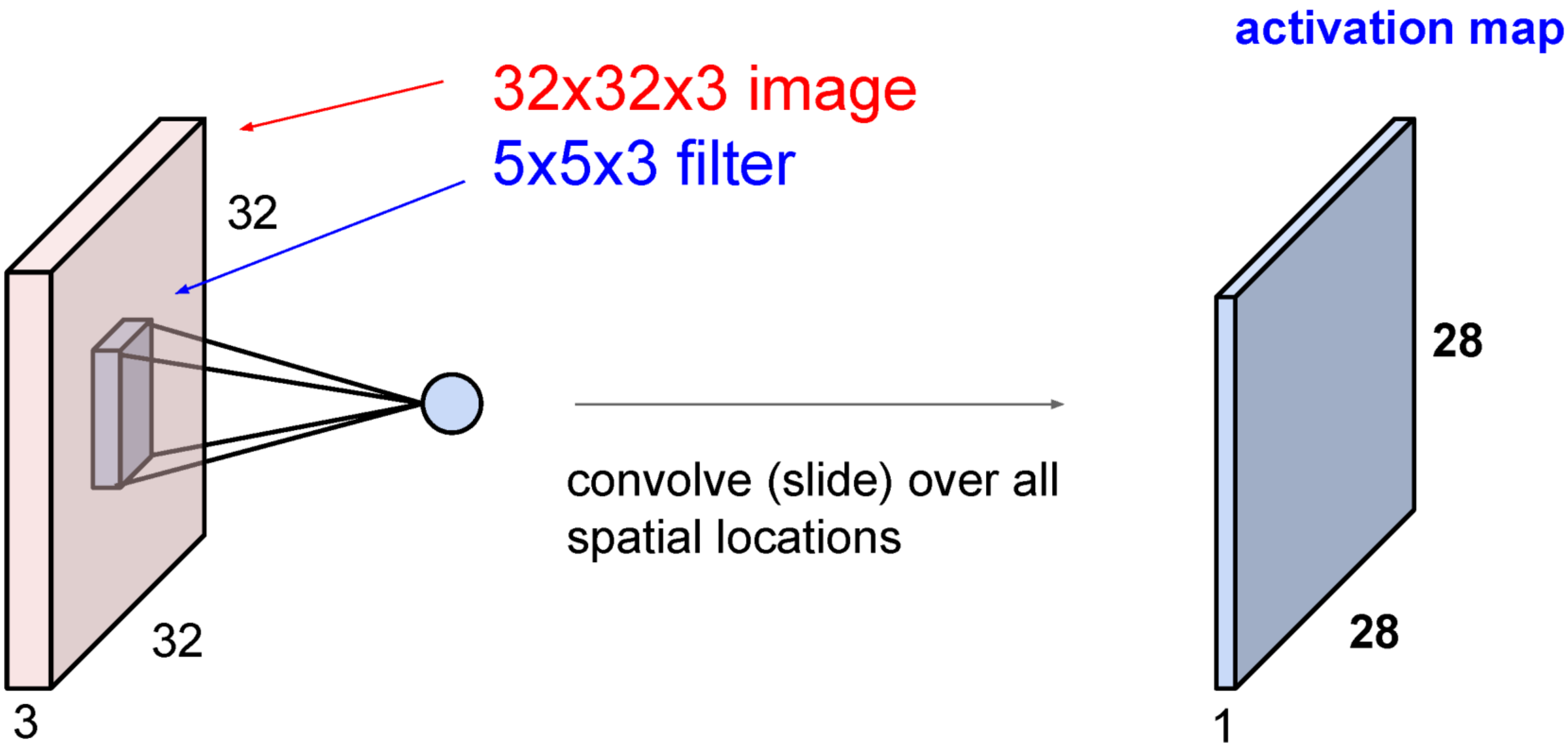
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



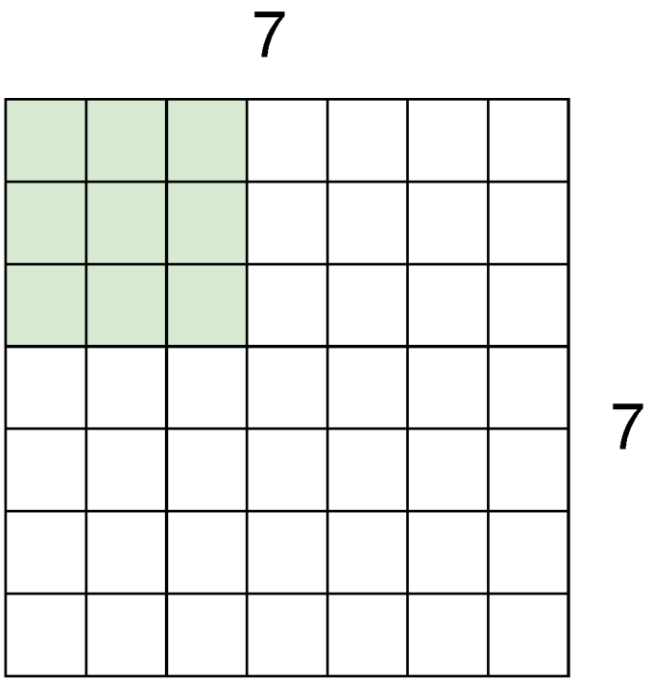
preview:


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

A closer look at spatial dimensions:

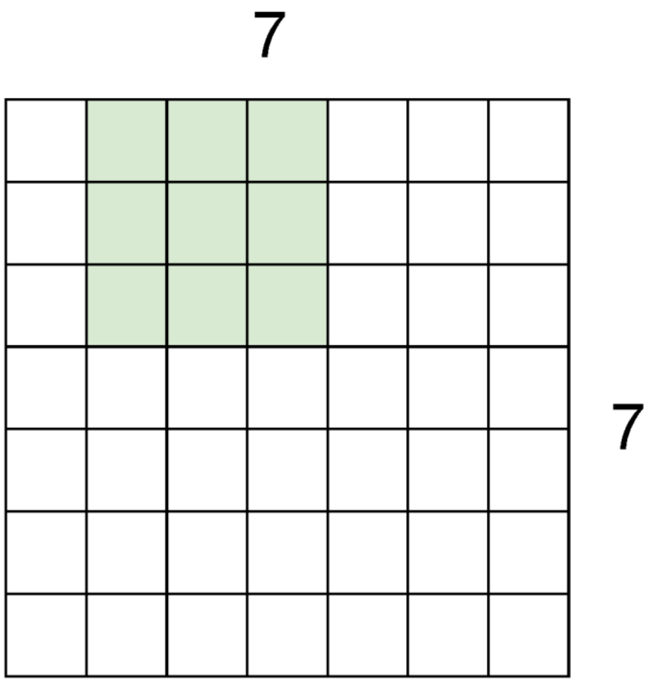


A closer look at spatial dimensions:



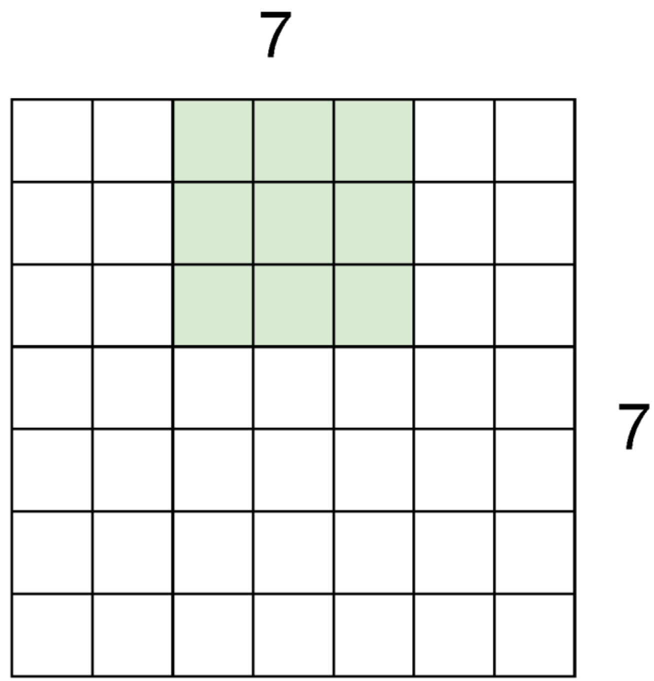
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



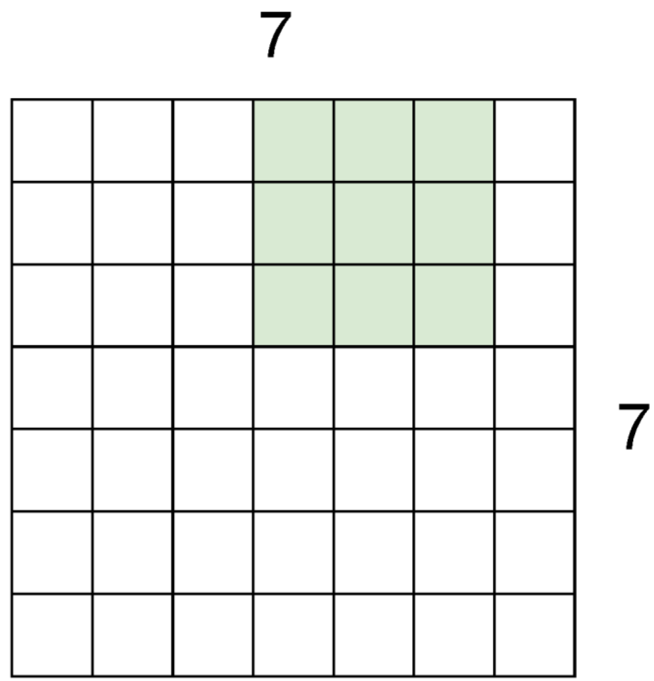
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



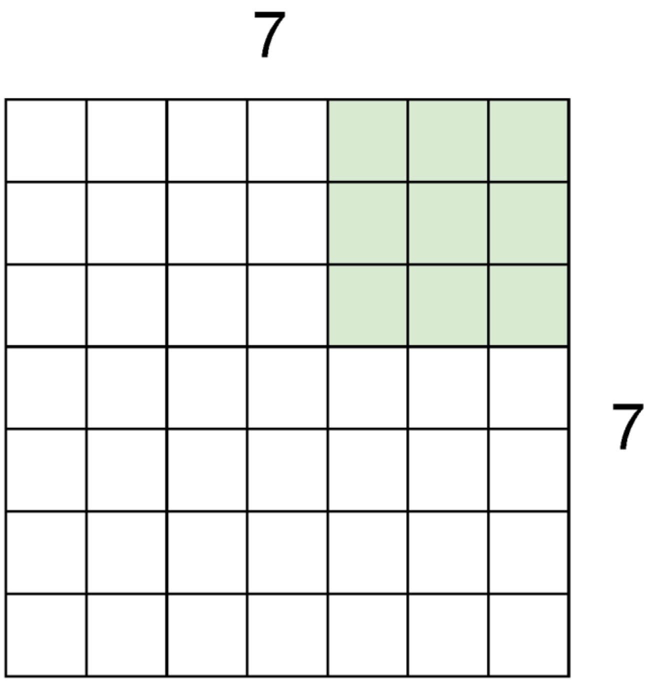
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

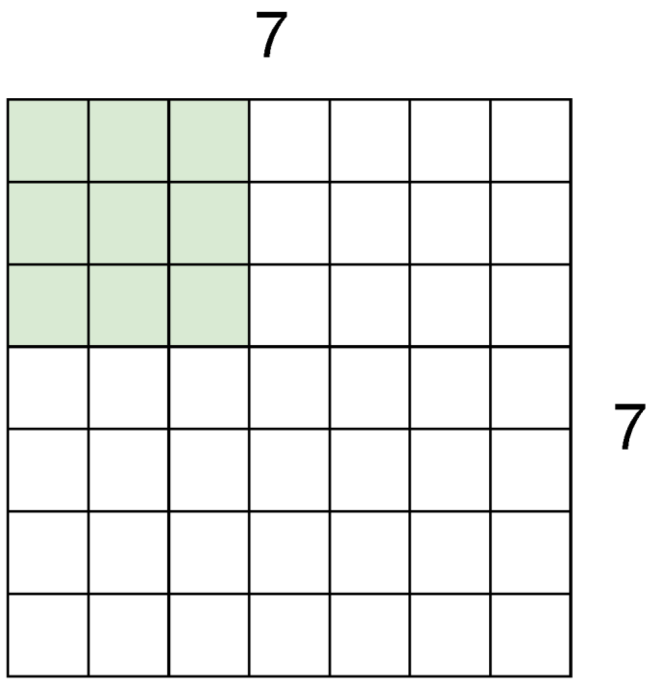
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

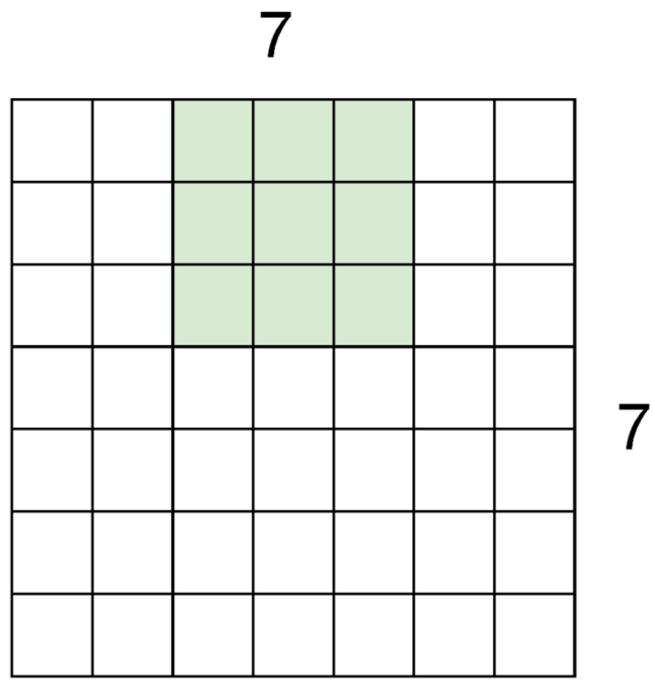
=> 5x5 output

A closer look at spatial dimensions:



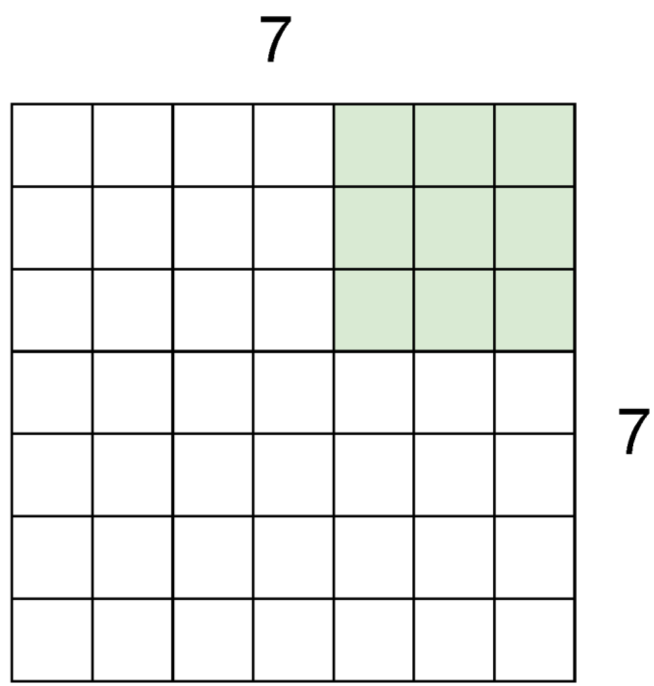
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



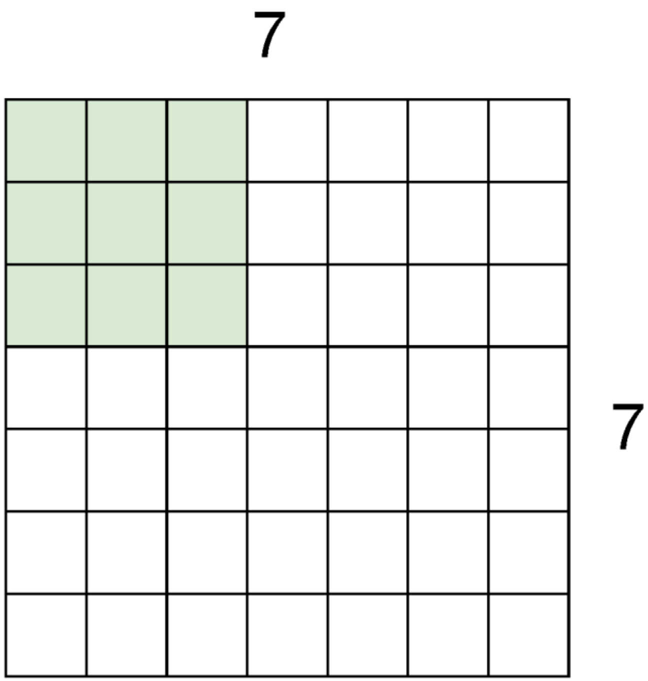
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



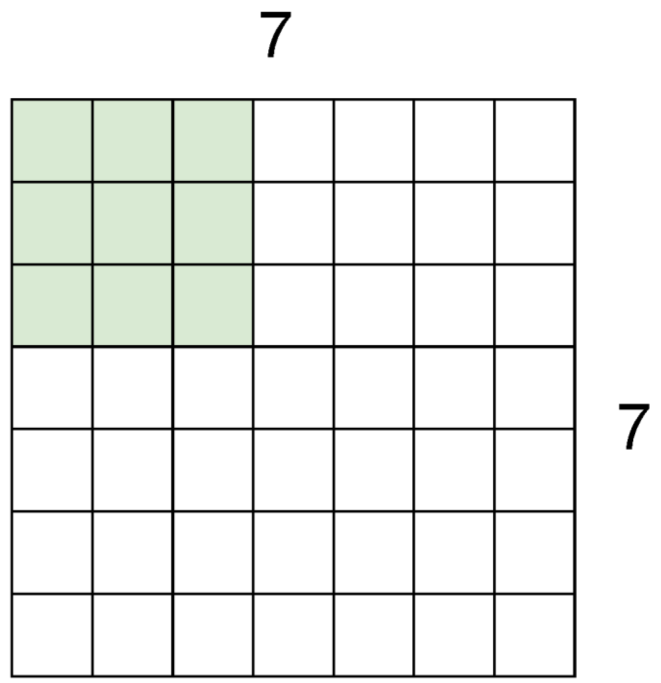
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



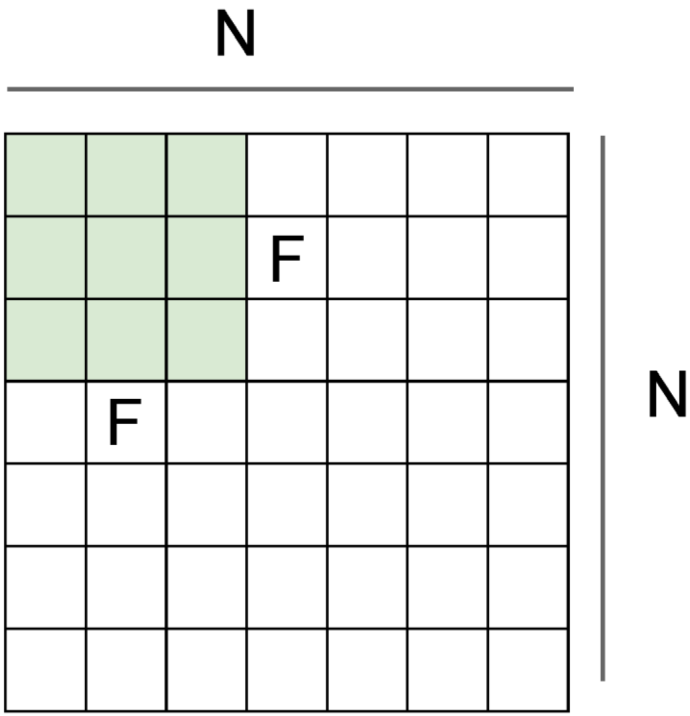
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



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

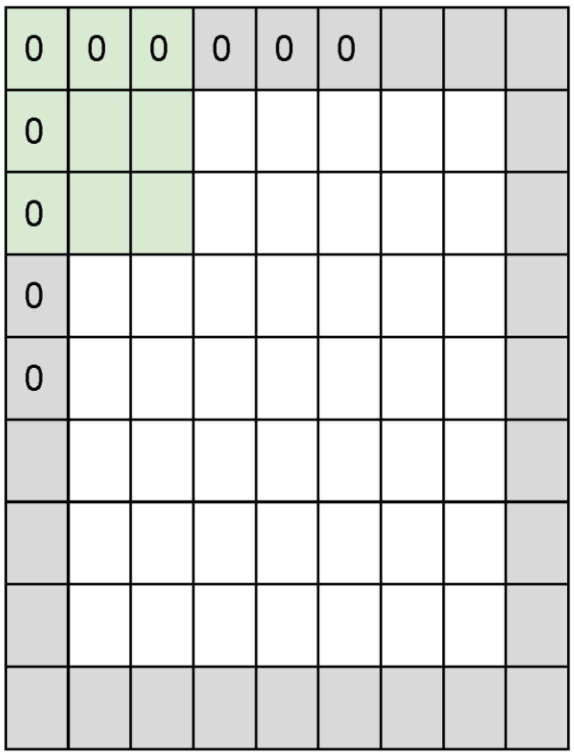
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



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

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

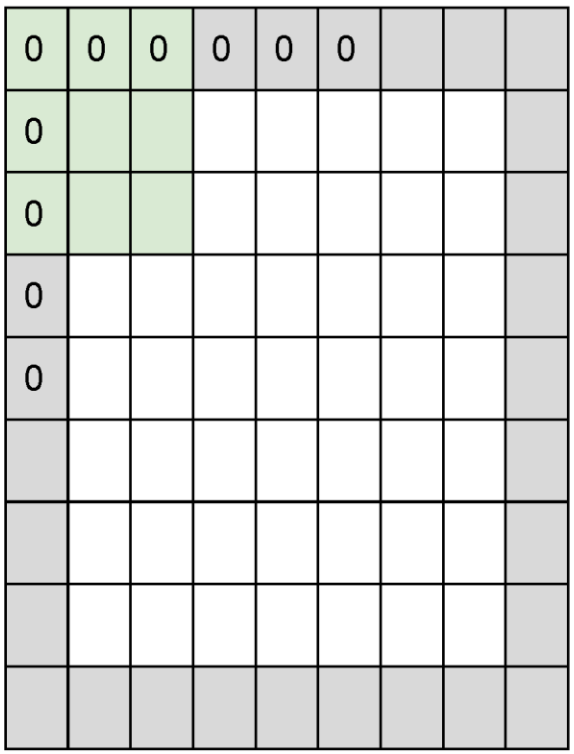
In practice: Common to zero pad the border



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

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

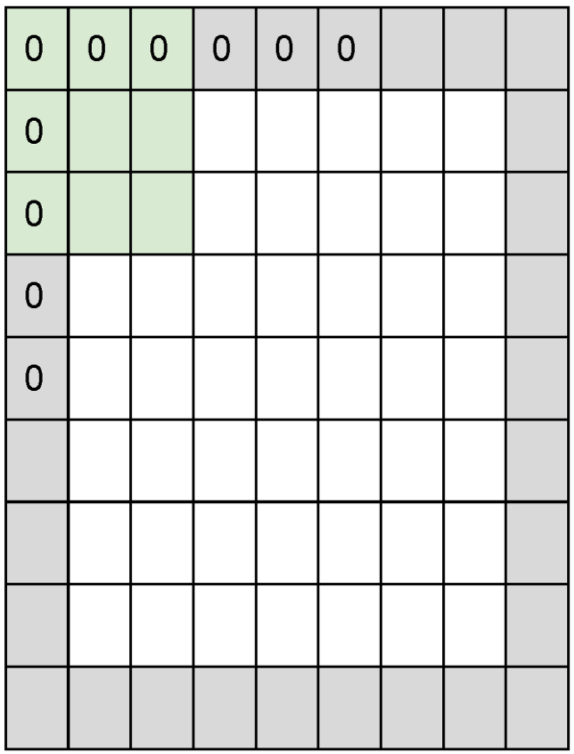
In practice: Common to zero pad the border



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

7x7 output!

In practice: Common to zero pad the border

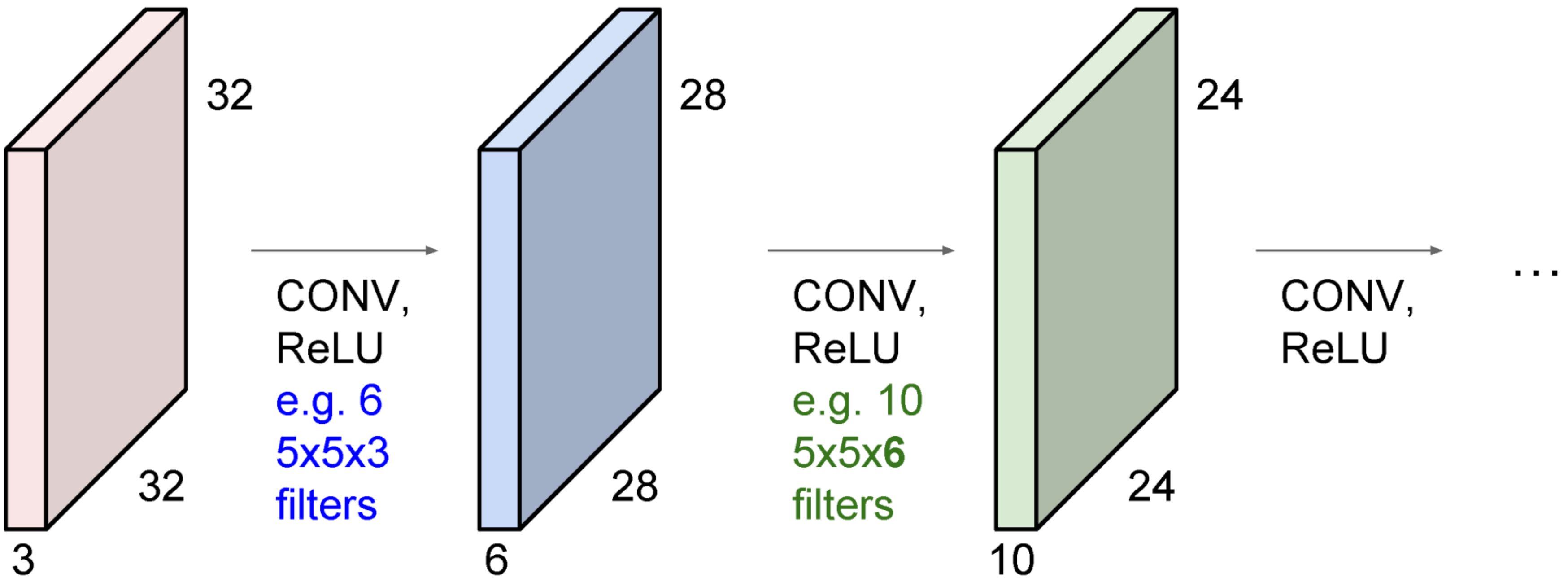


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

Remember back to…

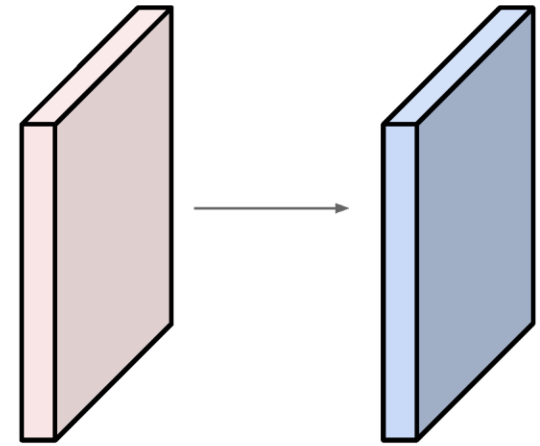
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



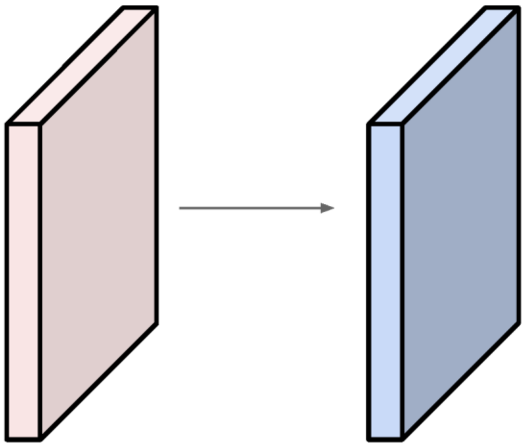
Examples time:

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

Output volume size: ?



Examples time:

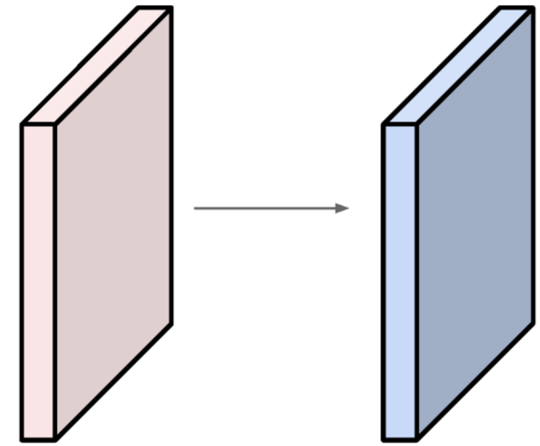


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

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

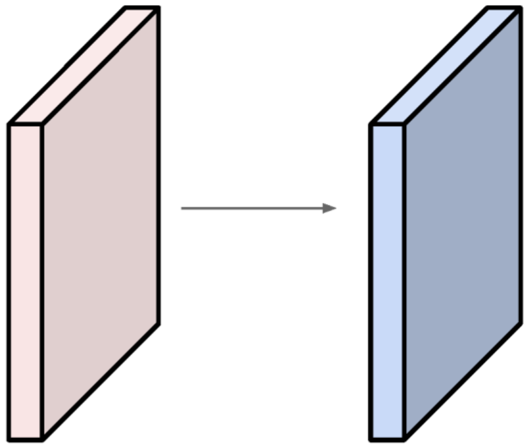
Examples time:

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



Number of parameters in this layer?

Examples time:



Input volume: **32x32x3**

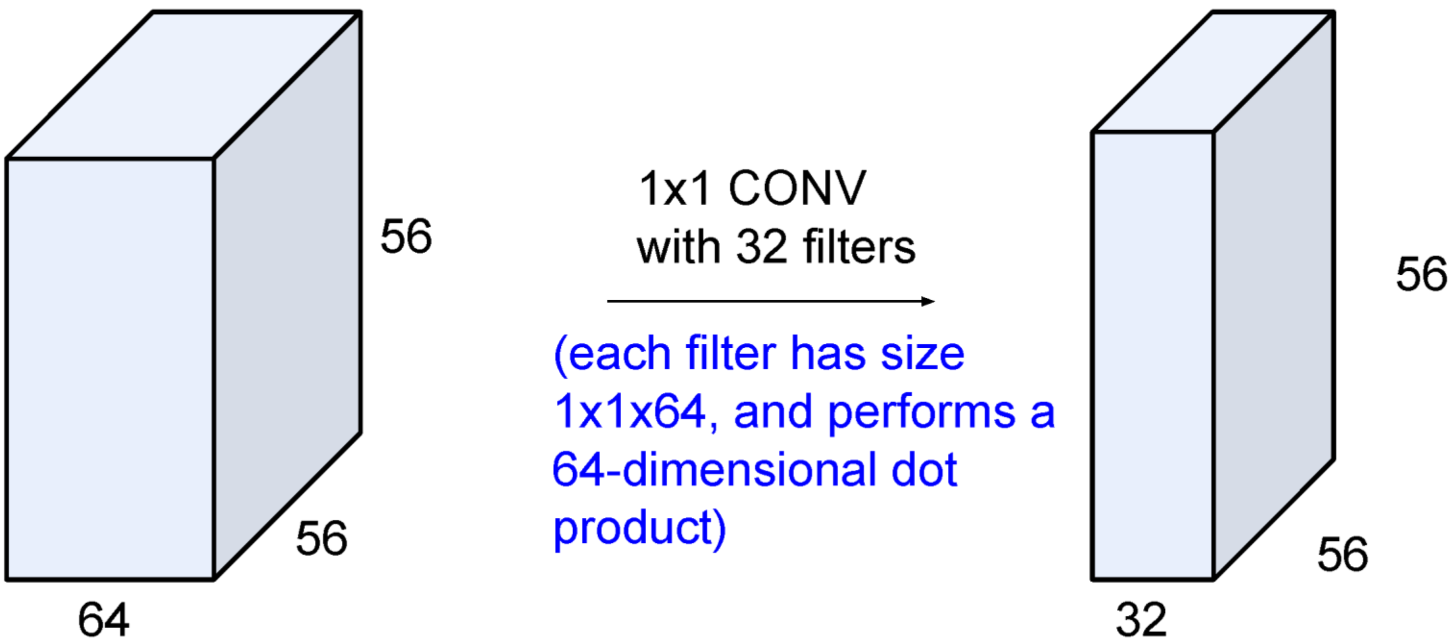
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

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

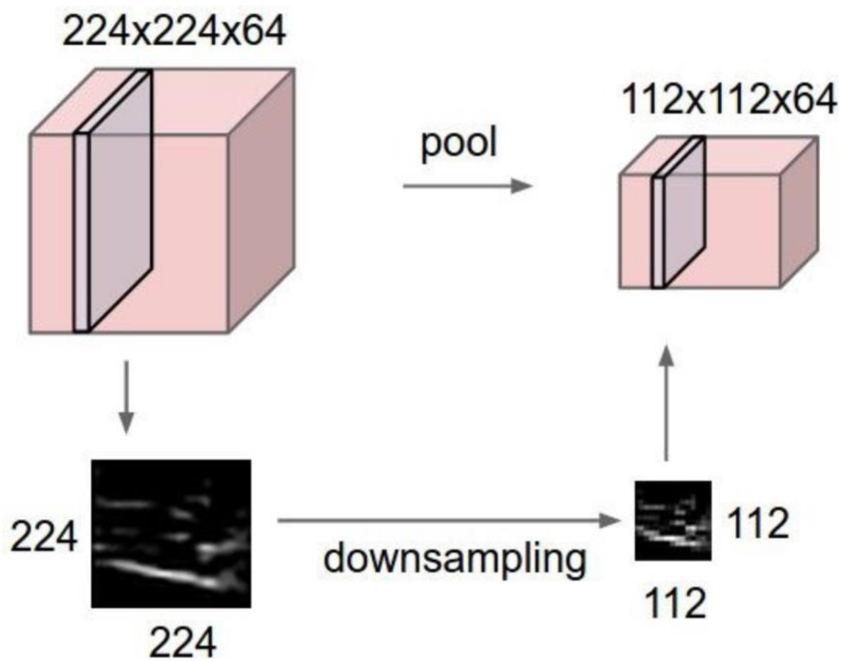
=> 76*10 = **760**

(btw, 1x1 convolution layers make perfect sense)

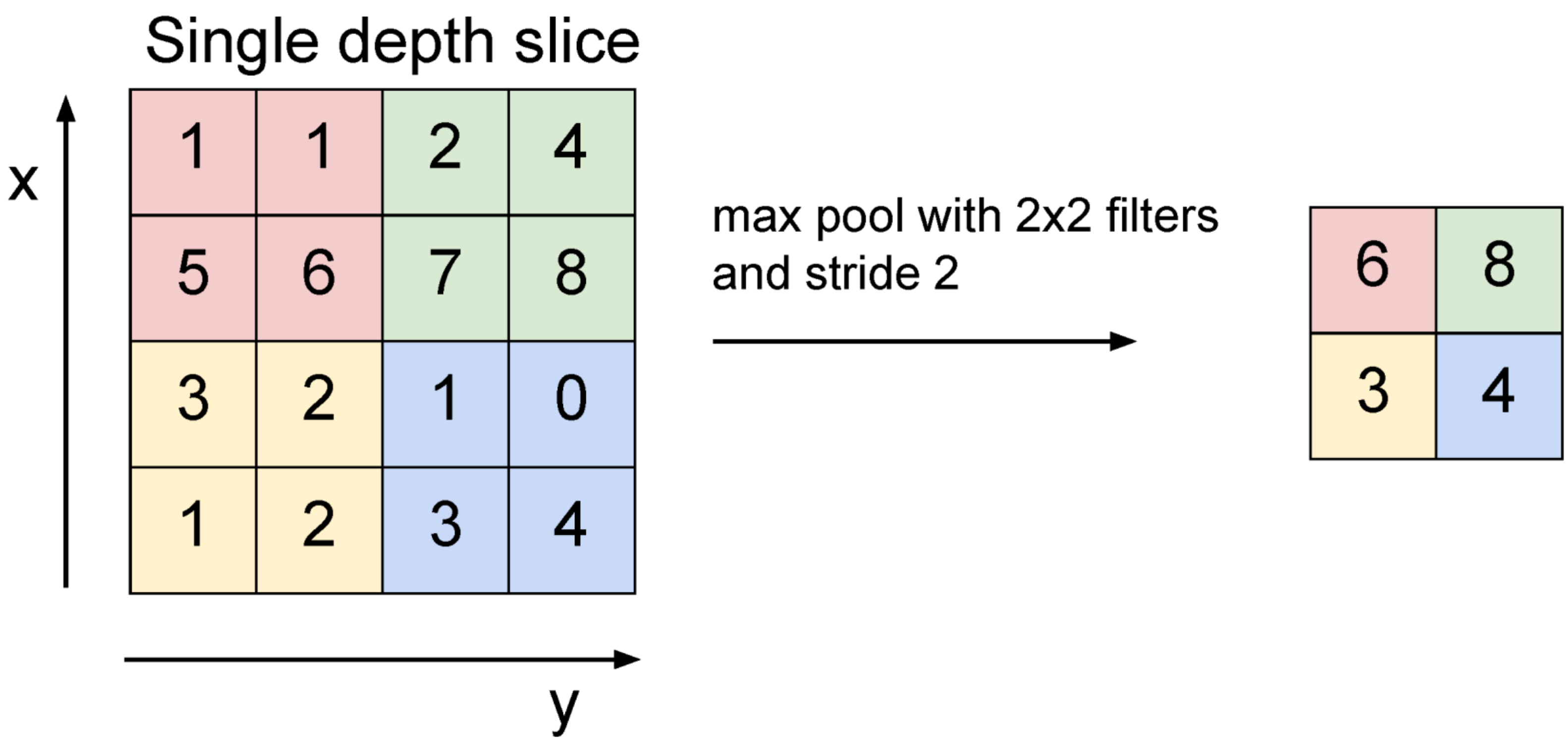


Pooling layer

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

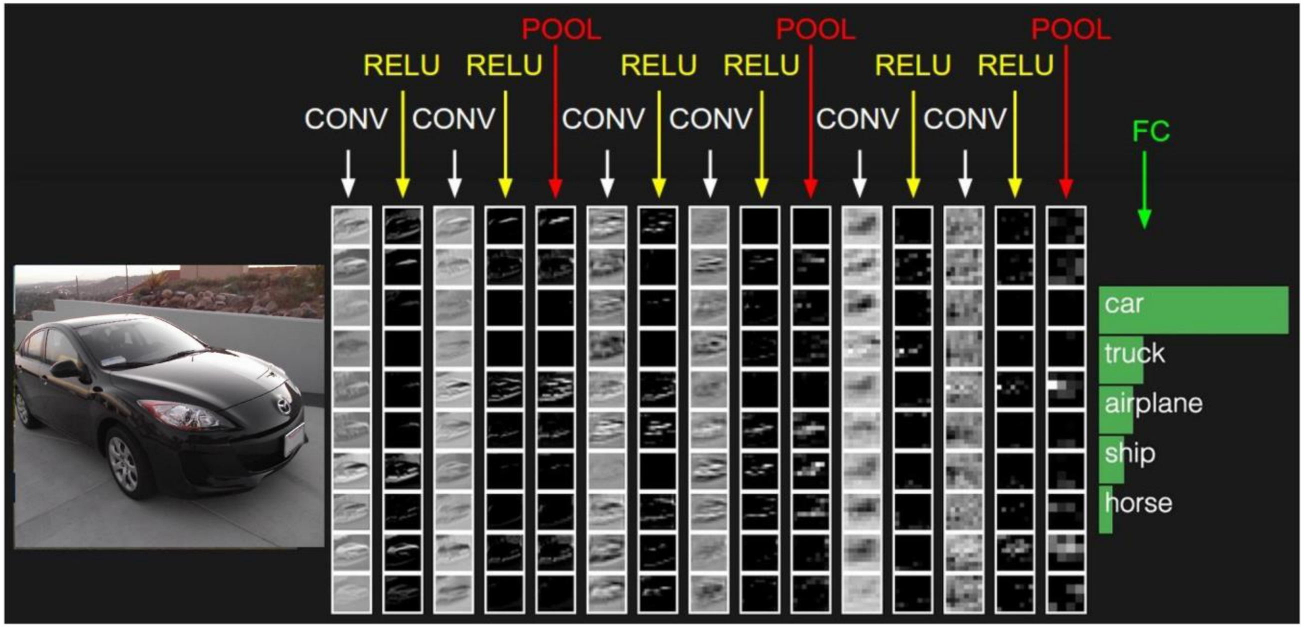


MAX POOLING


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Fully Connected Layer (FC layer)

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



[ConvNetJS demo: training on CIFAR-10]

ConvNetJS CIFAR-10 demo

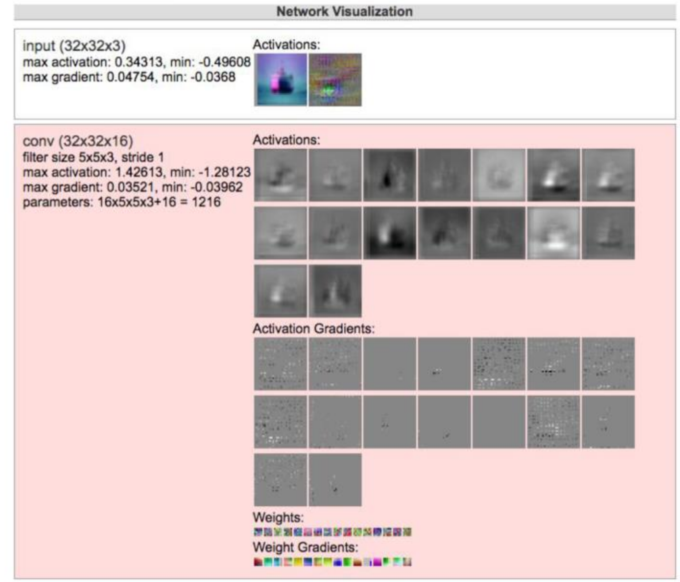
Description

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

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

Questions?