深度学习概述

周晓巍

Slides adapted from Fei-Fei Li, Justin Johnson, Serena Yeung, Noah Snavely, Jia-Bin Huang

Today's class

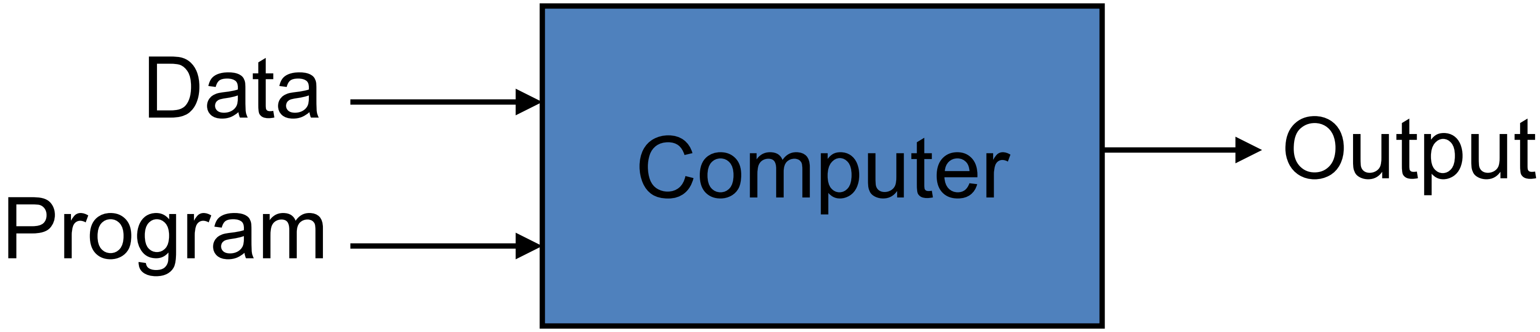
- Supervised Learning and Image Classification
- Linear Classifier
- Neural Networks
- Convolutional Neural Networks
- Training CNNs
- History and Recent Advances

Part I

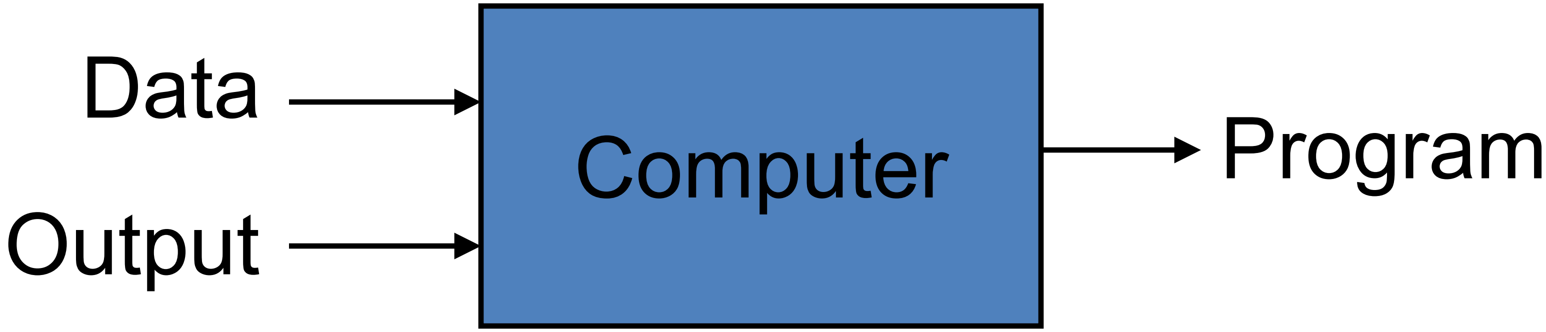
Supervised Learning
and Image Classification

Machine Learning

Traditional Programming

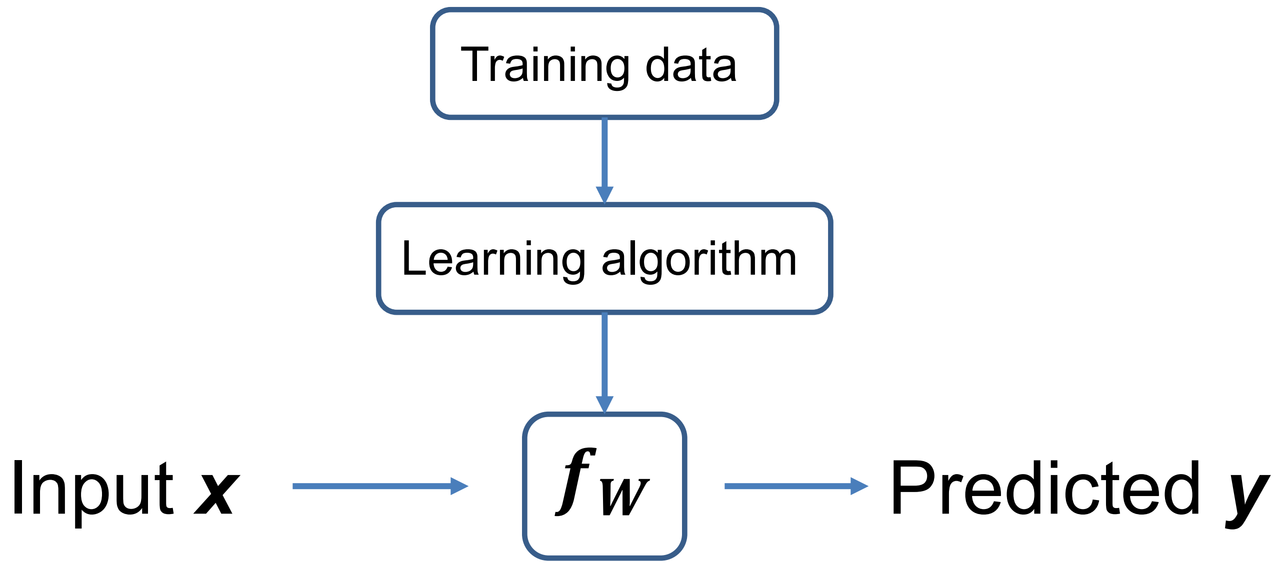


Machine Learning



Slide adapted from Pedro Domingos

Supervised Learning



y can be

- A real number (regression)
- A discrete label (classification)

Image Classification:
A core task in Computer Vision

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

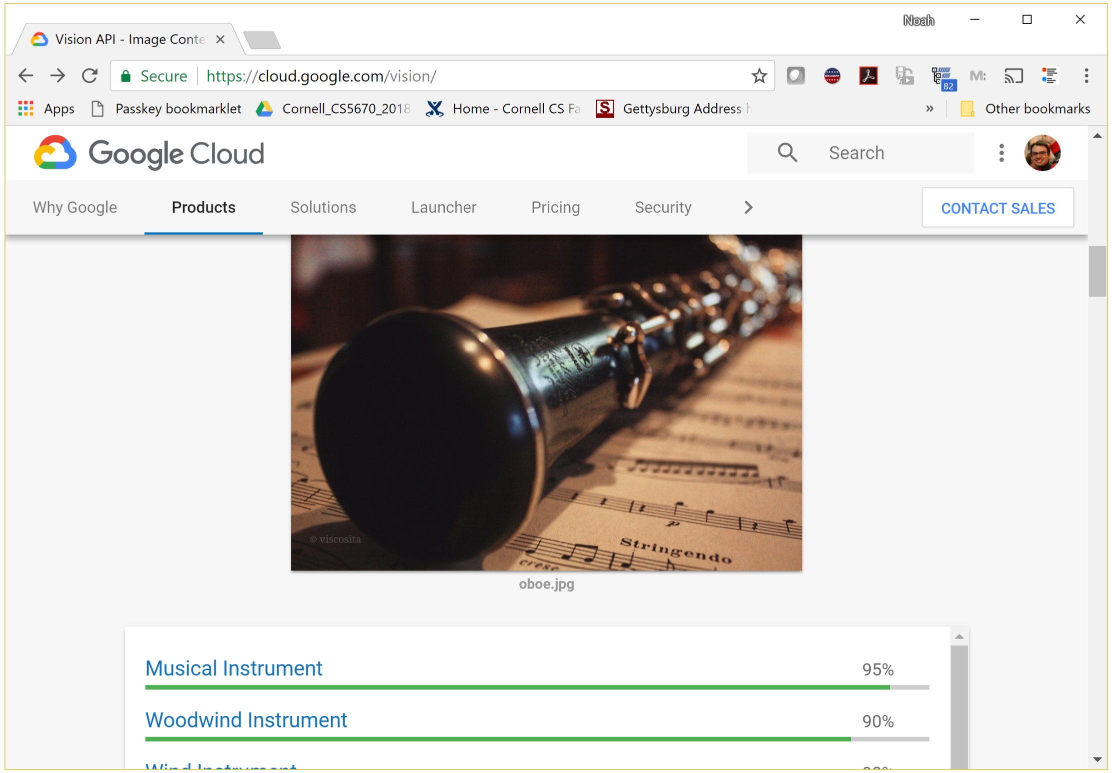
f() = “apple”

f() = “tomato”

f() = “cow”

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

Image classification demo



https://cloud.google.com/vision/

See also:

https://aws.amazon.com/rekognition/

https://www.clarifai.com/

https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/

…

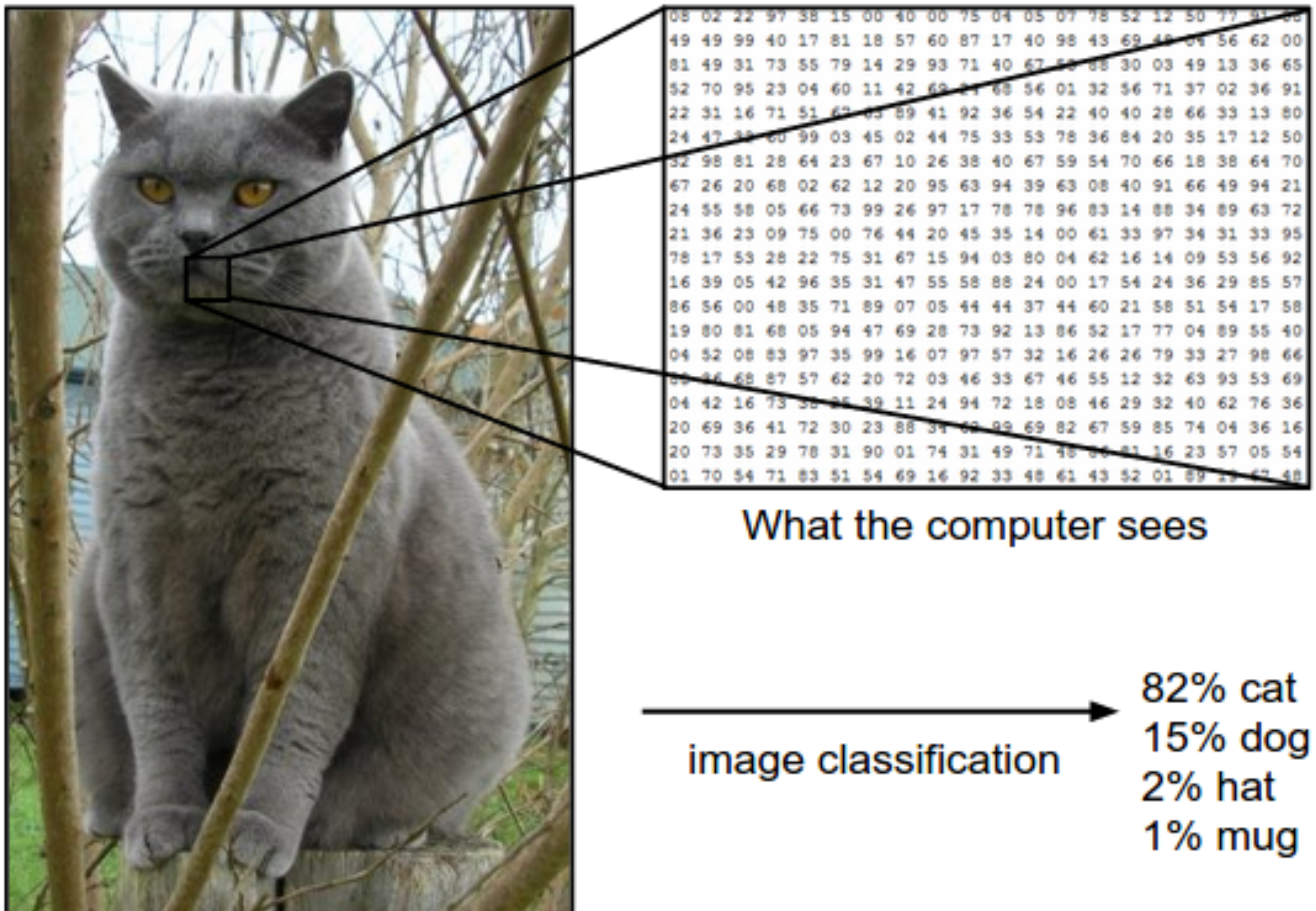
Image Classification

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

→ cat

Slides from Andrej Karpathy and Fei-Fei Li
http://vision.stanford.edu/teaching/cs231n/

Image Classification: Problem



An image classifier

```
def classify_image(image):
  # Some magic here?
  return class_label
```

Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

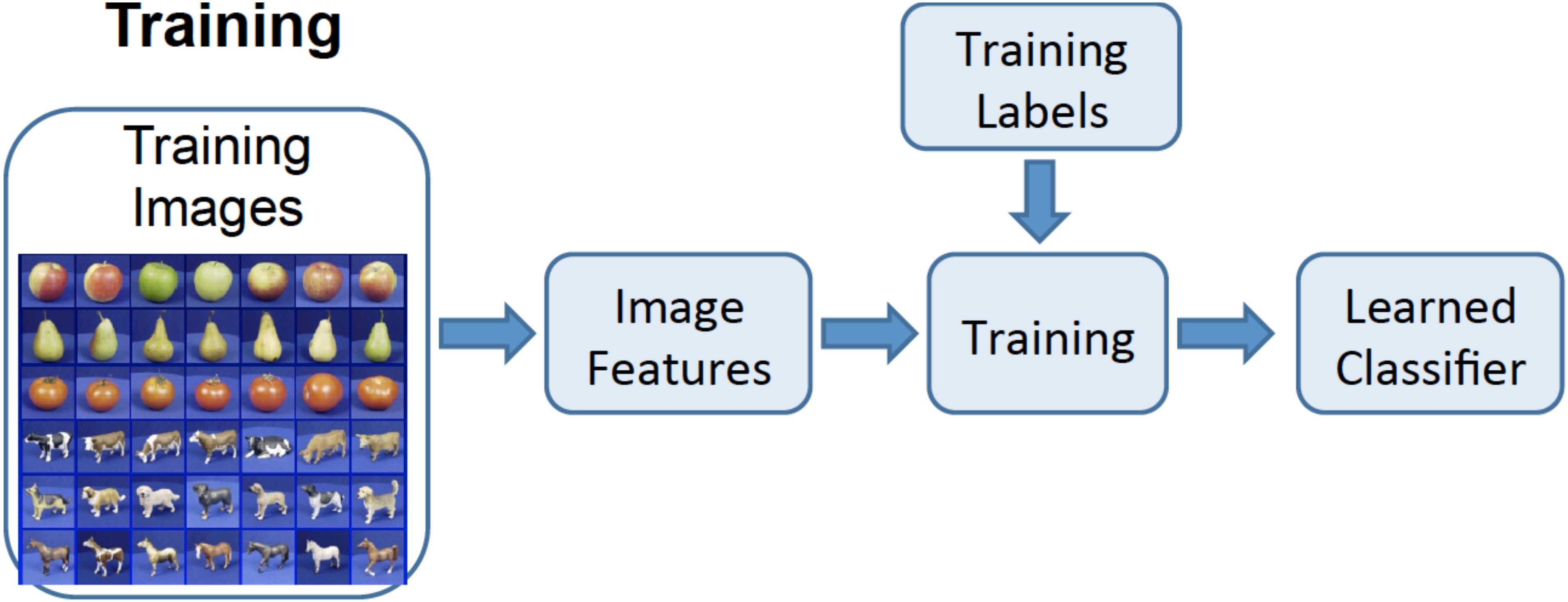
Slide credit: Fei-Fei Li & Justin Johnson & Serena Y

Data-driven approach

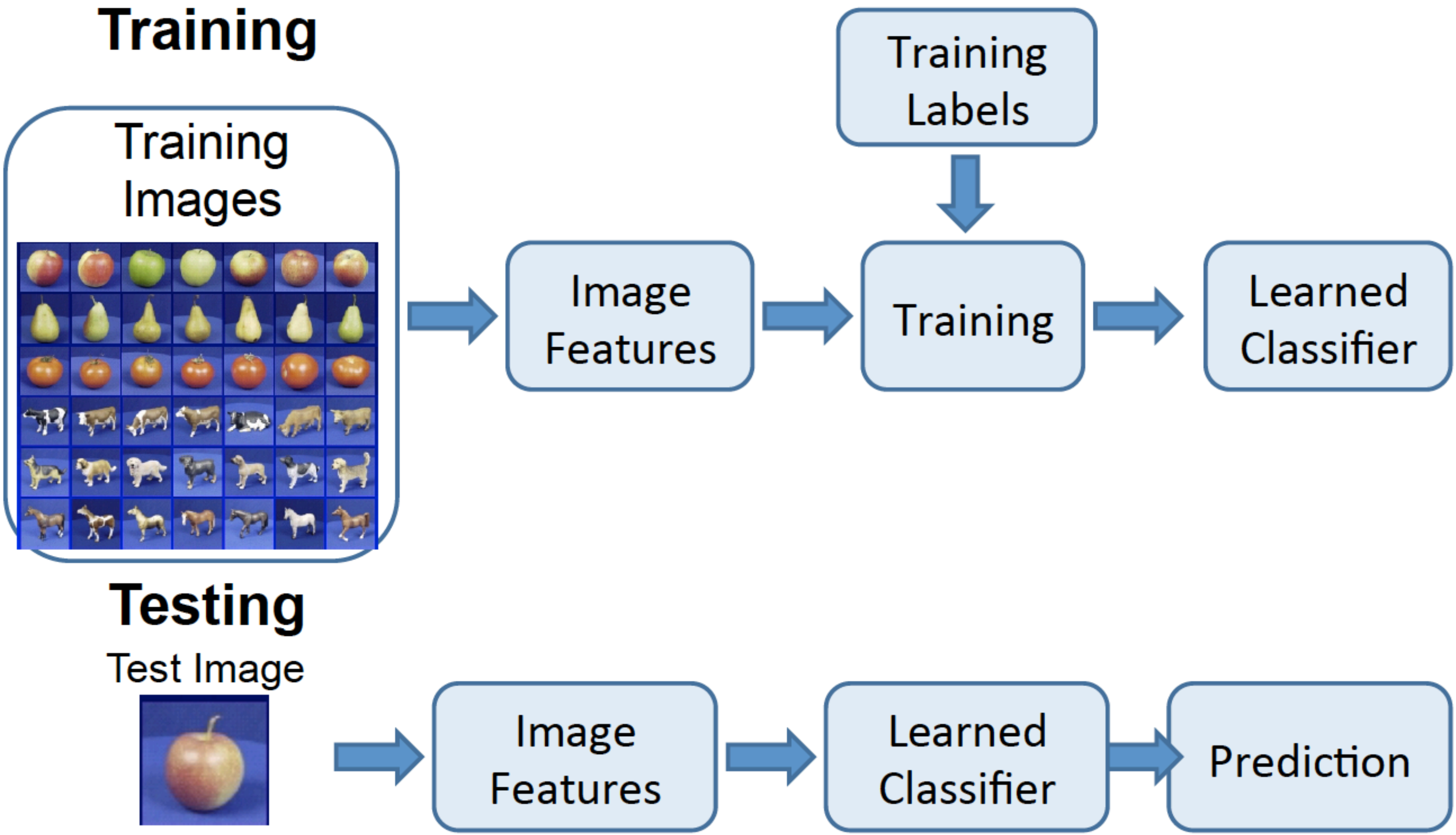
- Collect a database of images with labels
- Use ML to train an image classifier

Slides from Andrej Karpathy and Fei-Fei Li
http://vision.stanford.edu/teaching/cs231n/

Training

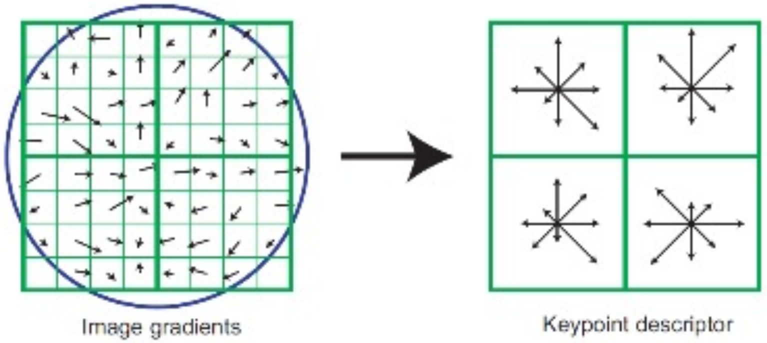


Dataset: ETH-80, by B. Leibe   Slide credit: D. Hoiem, L. Lazebnik

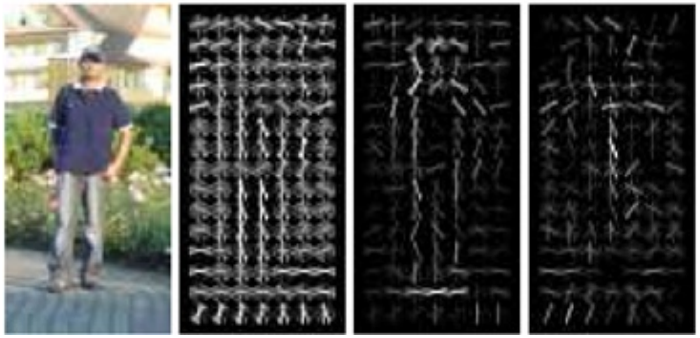


Dataset: ETH-80, by B. Leibe Slide credit: D. Hoiem, L. Lazebnik

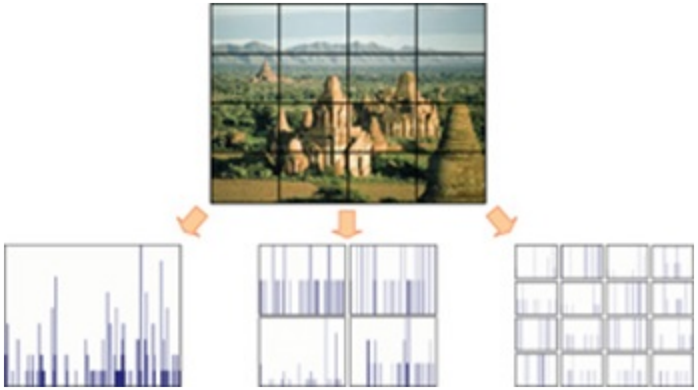
Image features



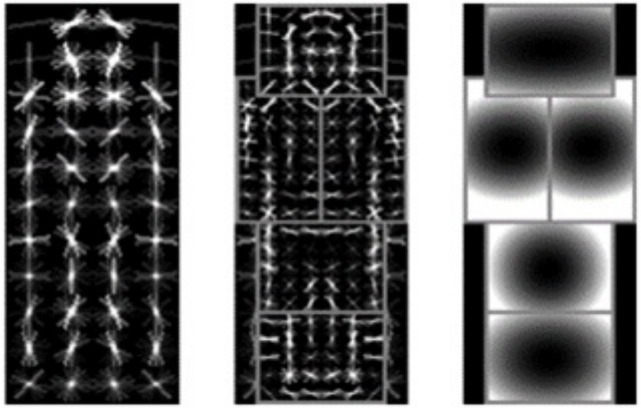
SIFT [Loewe IJCV 04]



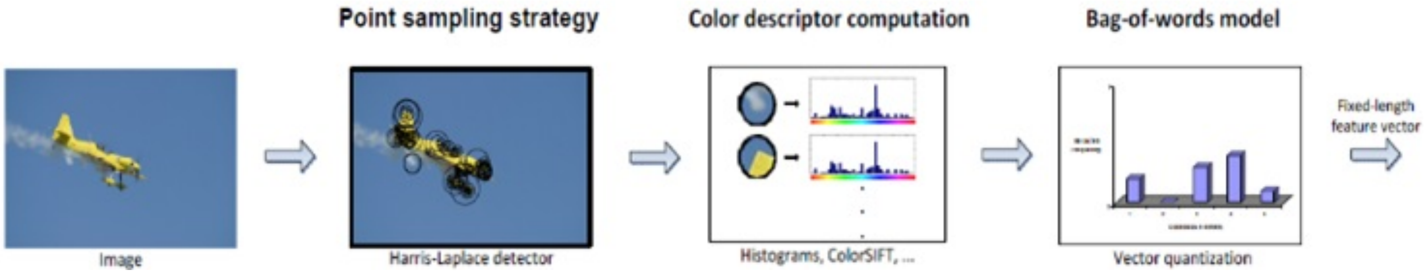
HOG [Dalal and Triggs CVPR 05]



SPM [Lazebnik et al. CVPR 06]



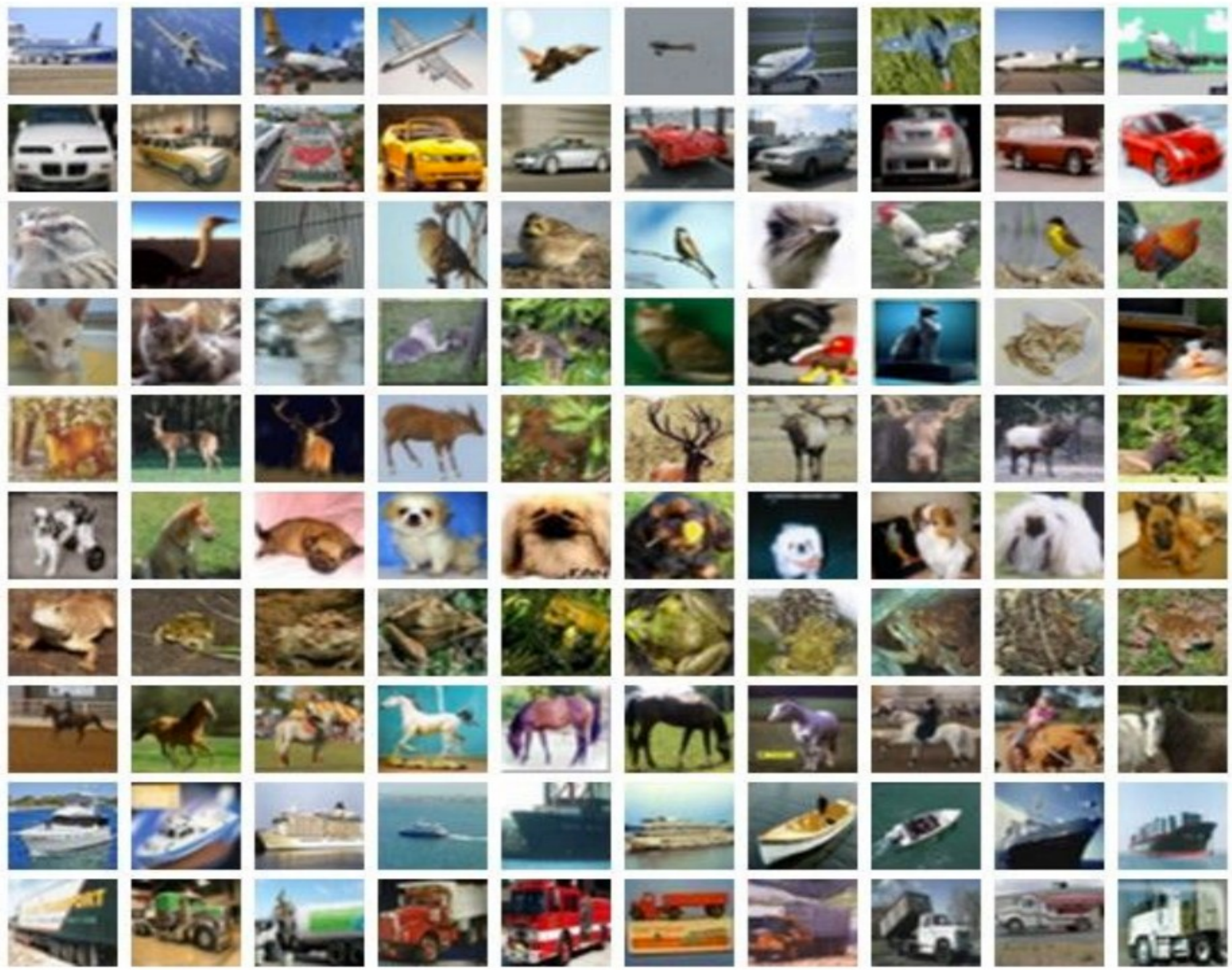
DPM [Felzenszwalb et al. PAMI 10]



Color Descriptor [Van De Sande et al. PAMI 10]

Why use features? Why not pixels?

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck



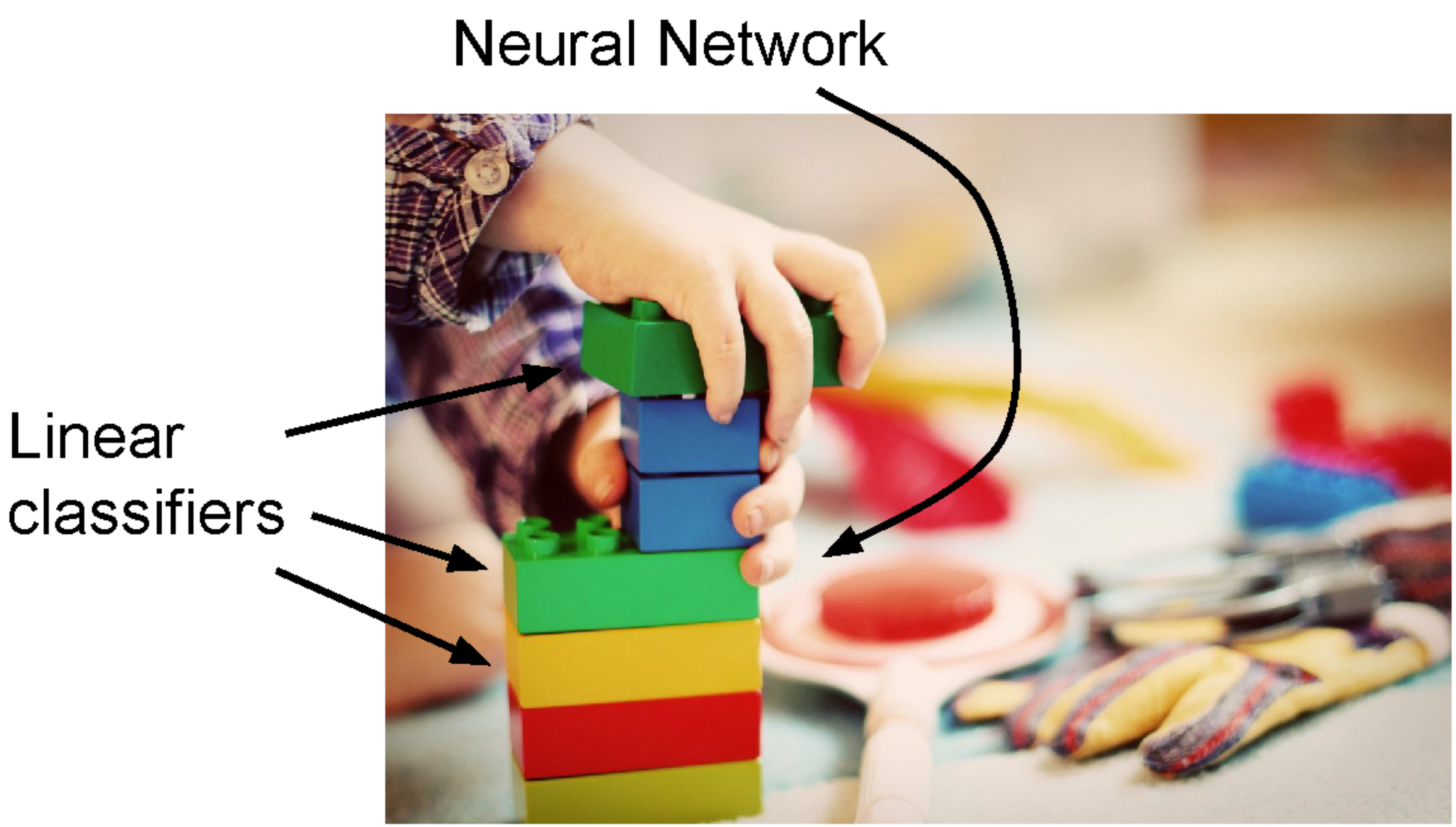
Classifiers

- Nearest Neighbor
- kNN (“k-Nearest Neighbors”)
- Linear Classifier
- Decision Tree…

Part II

Linear Classifier

Linear classifiers



Score function

class scores

Slides from Andrej Karpathy and Fei-Fei Li
http://vision.stanford.edu/teaching/cs231n/

Score function: f

Parametric approach

image parameters

f(**x**,**W**)

[32x32x3]
array of numbers 0...1
(3072 numbers total)

10 numbers,
indicating class
scores

Score function: f

define a **score function**

data (image)

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

class scores

"weights"

"bias vector"

"parameters"

Parametric approach: Linear classifier

$$\boxed{f(x,W)} = \boxed{W}\boxed{x}$$

3072x1

10x1

10x3072

[32x32x3]
array of numbers 0...1

10 numbers, indicating class scores

parameters, or "weights"

Parametric approach: Linear classifier

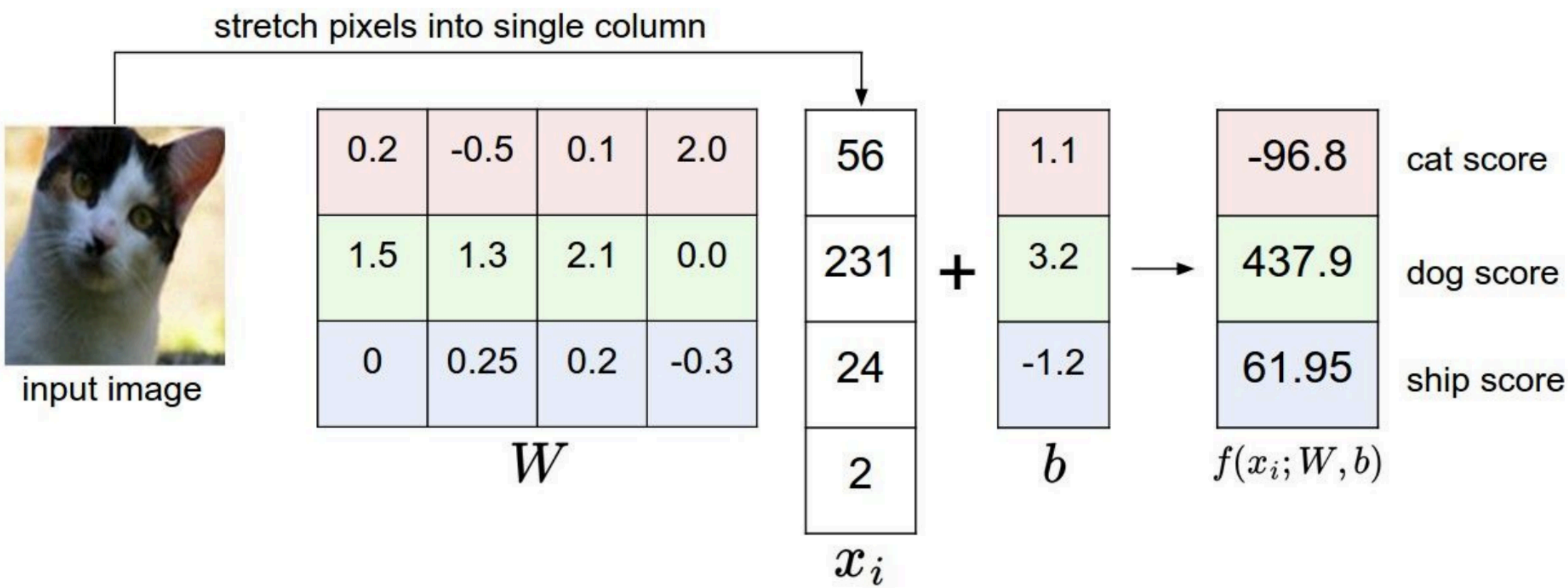
$$\underbrace{\boxed{f(x,W)}}_{10\text{x}1} = \underbrace{\boxed{W}}_{10\text{x}3072}\overbrace{\boxed{x}}^{3072\text{x}1} \quad \boxed{(+b)}\ 10\text{x}1$$

[32x32x3]
array of numbers 0...1

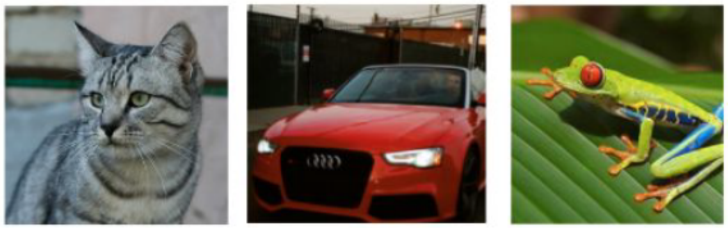
10 numbers, indicating class scores

parameters, or "weights"

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



Training: how to find good W based on training data?



airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

Output scores

TODO:

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.

2. Come up with a way of efficiently finding the parameters that minimize the loss function. **(optimization)**

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

A **loss function** tells how good our current classifier is

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and
y_i is (integer) label

Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

How to define a loss function for predicted scores?

1. Convert scores to probabilities
2. Compute cross entropy between predicted and true probabilities

$$H(p, q) = -\sum_{x} p(x) \log q(x)$$

Covert scores to probabilities

Softmax function: $\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ for $j = 1, \ldots, K$.

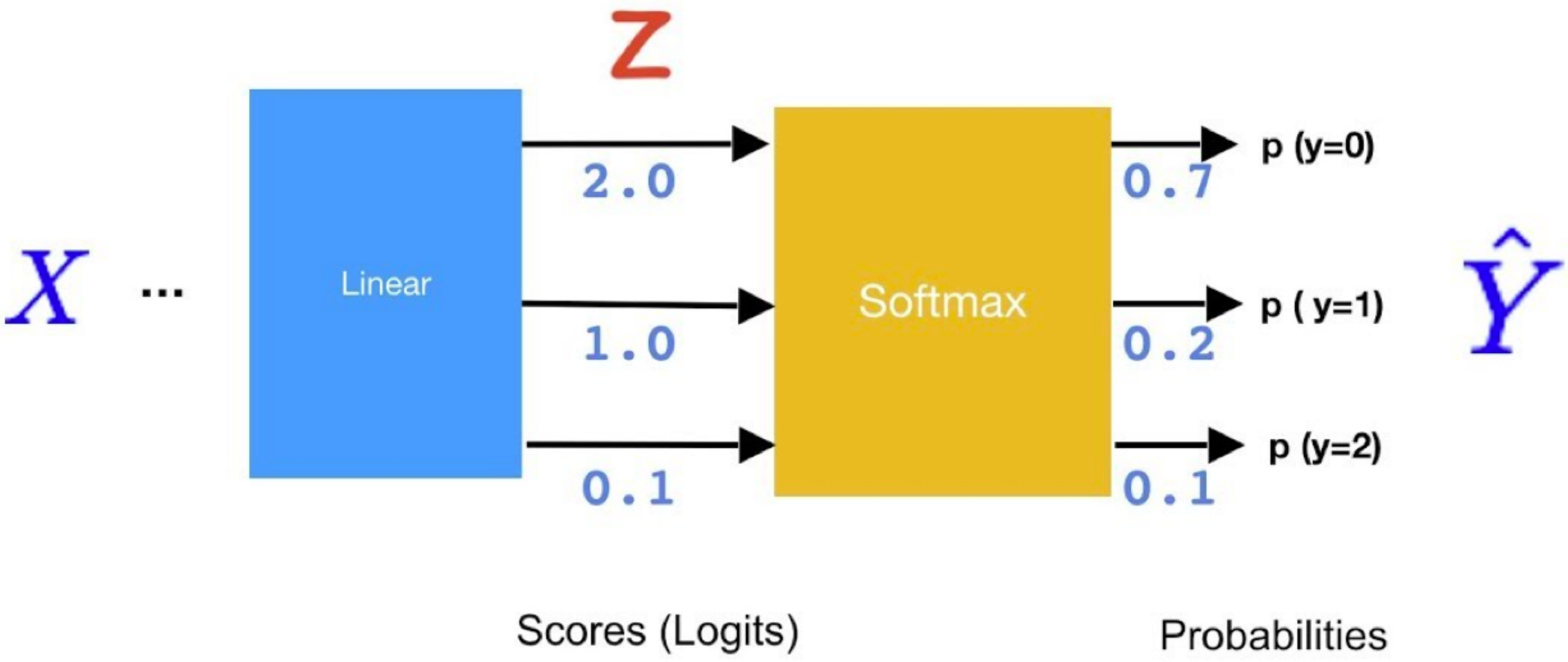


Image credit: Sung Kim

Cross entropy as loss function

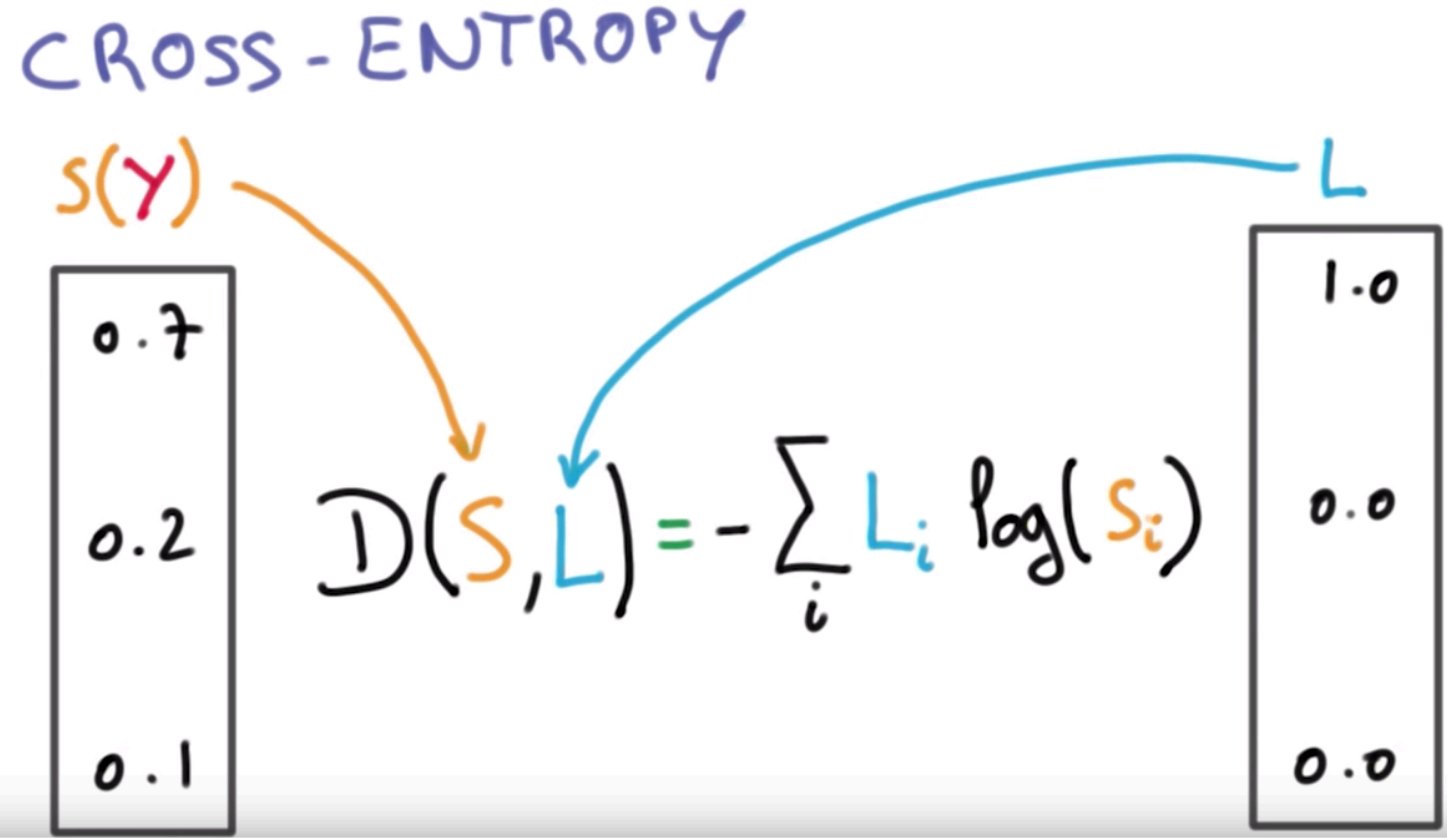


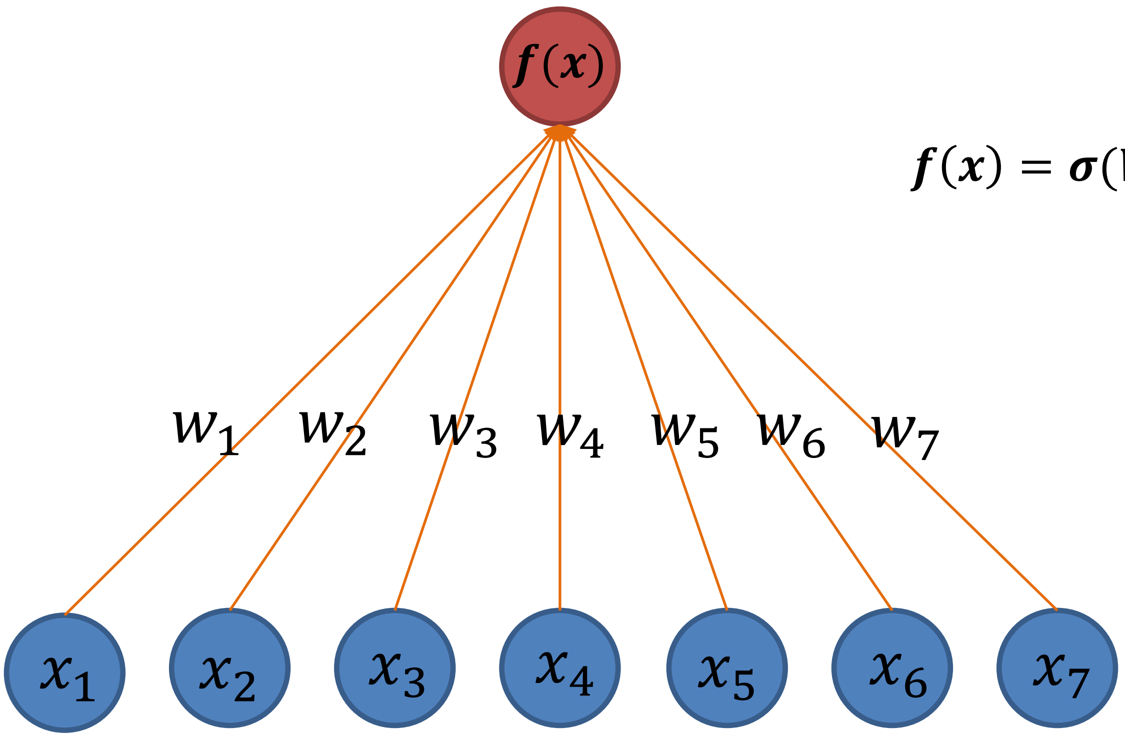
Image credit: Ritchie Ng

Part III

Neural Networks

Neural networks

- Perceptron

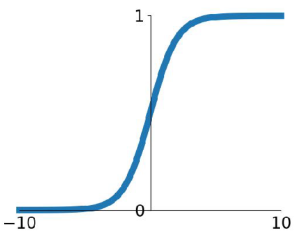


$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{\sigma}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

Activation functions

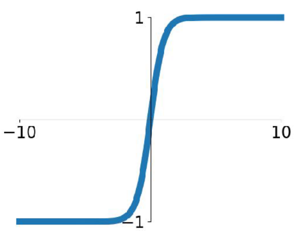
Sigmoid

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



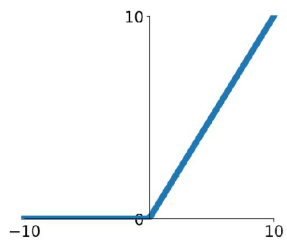
tanh

$\tanh(x)$



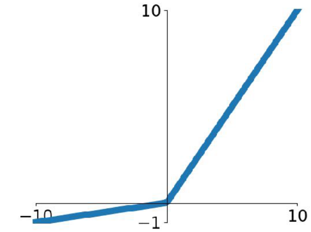
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

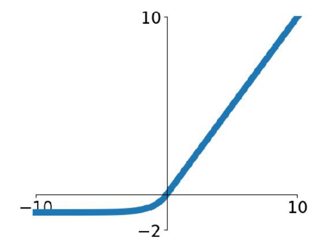


Maxout

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

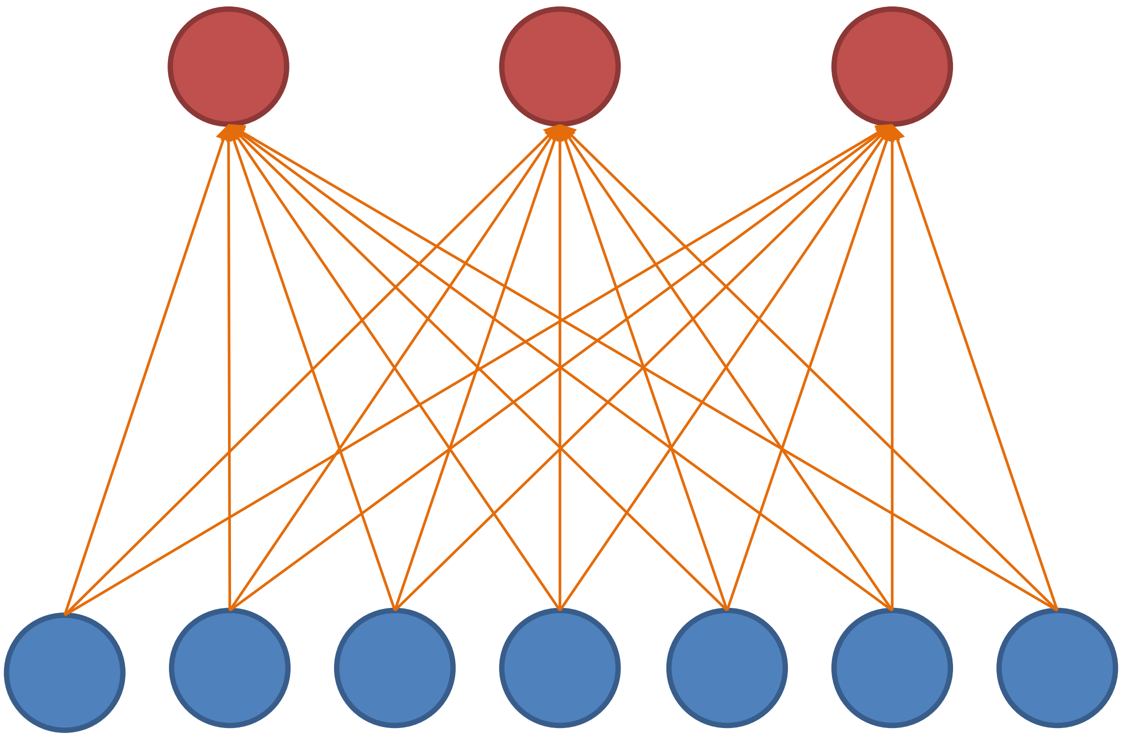
ELU

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

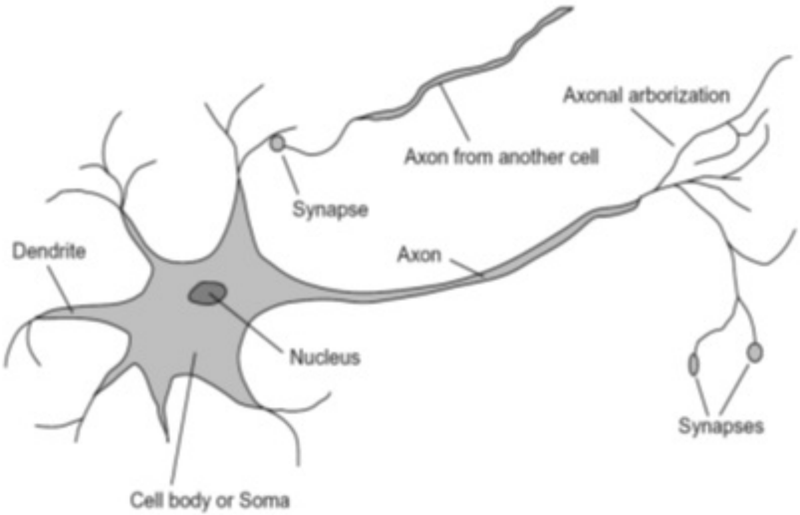


Neural networks

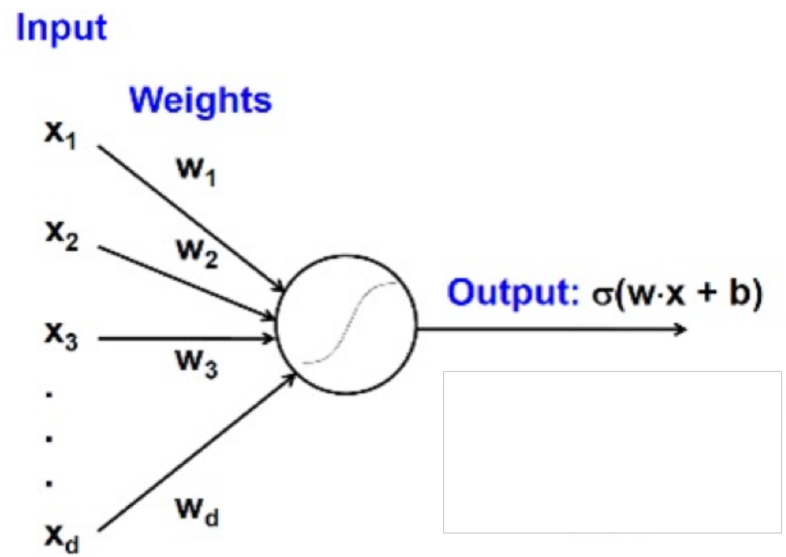
- Extend to multiple outputs



Biological neuron and Perceptrons

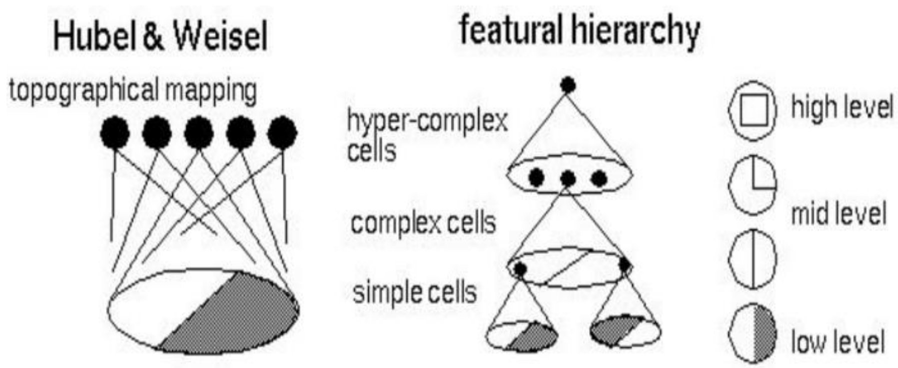


A biological neuron

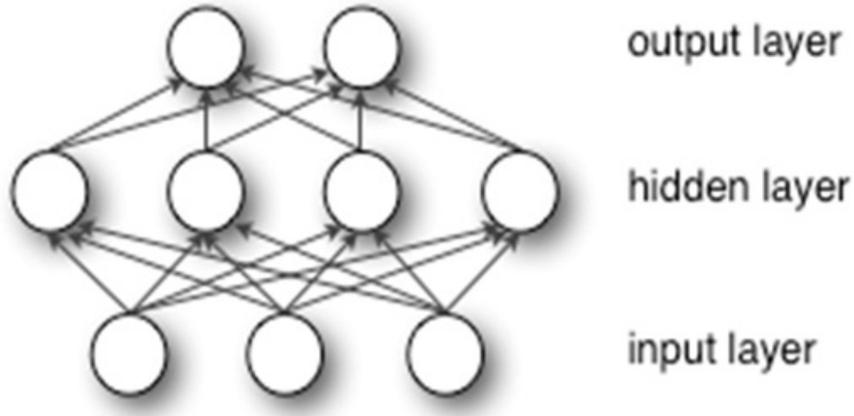


An artificial neuron (Perceptron)
- a linear classifier

Hubel/Wiesel Architecture and Multi-layer Neural Network



Hubel and Weisel's architecture



Multi-layer Neural Network
- A *non-linear* classifier

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 $f = W_2 \max(0, W_1 x)$
or 3-layer Neural Network

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

Neural networks

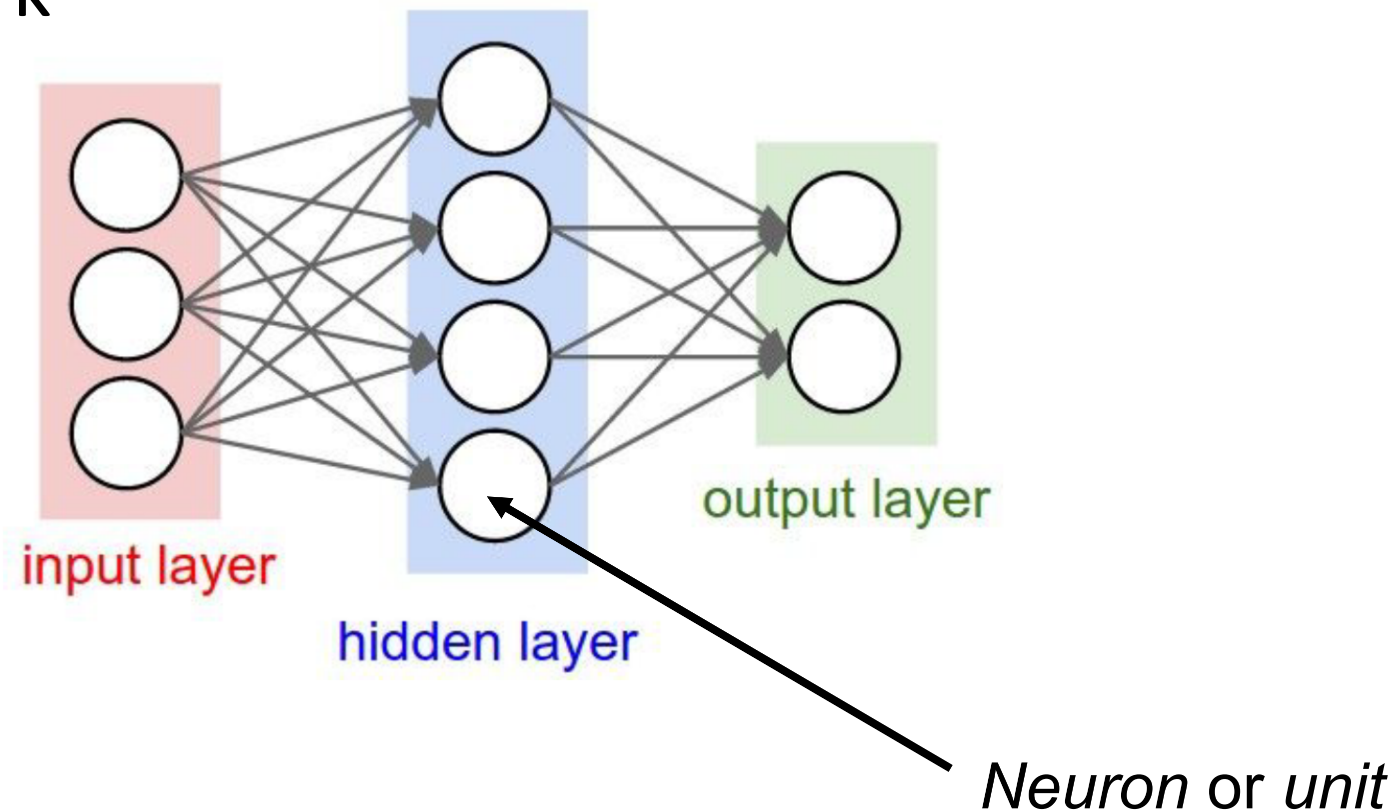
- Very coarse generalization:
  - Linear functions chained together and separated by non-linearities (*activation functions*), e.g. “max”

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

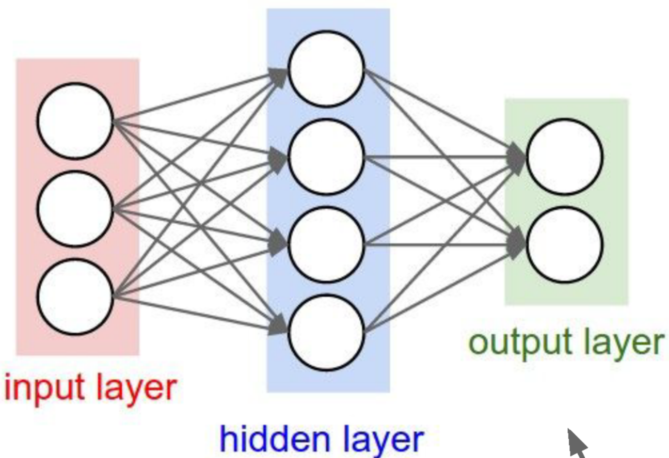
  - Why separate linear functions with non-linear activation functions?

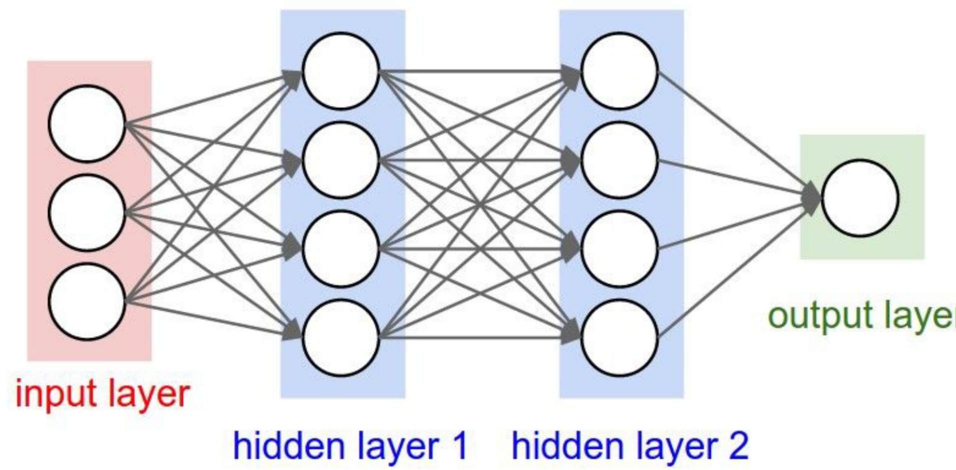
Neural network architecture

- Computation graph for a 2-layer neural network



Neural networks: Architectures





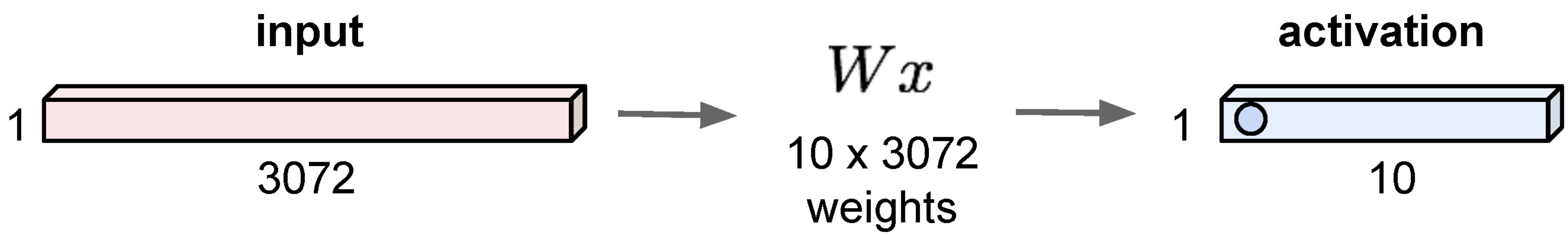
“2-layer Neural Net”, or
“1-hidden-layer Neural Net”

“Fully-connected” layers

“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

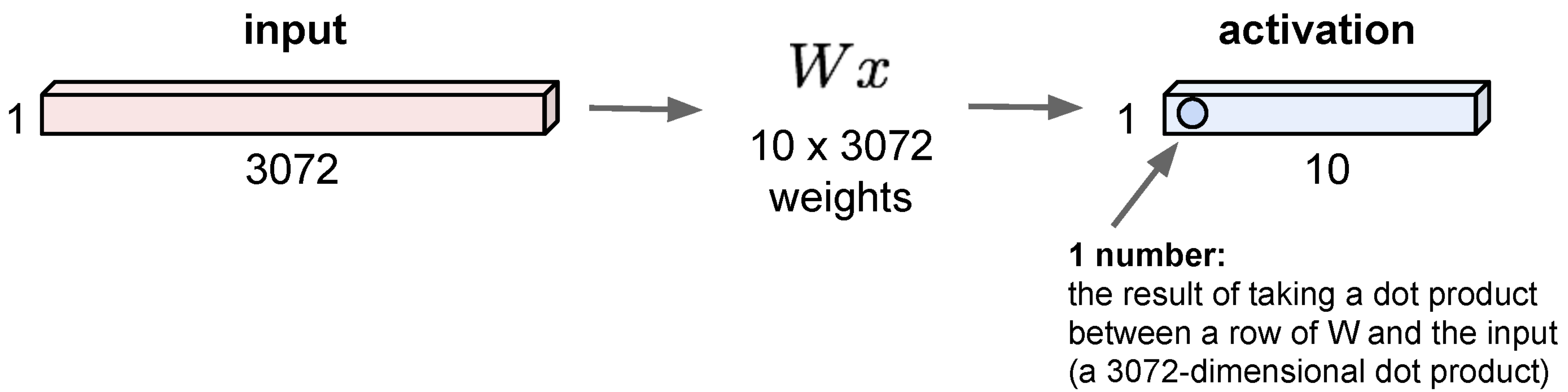
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



- **Deep** networks typically have many layers and potentially millions of parameters
- How to reduce number of parameters?

Part IV

Convolutional Neural Networks

Convolutional neural networks

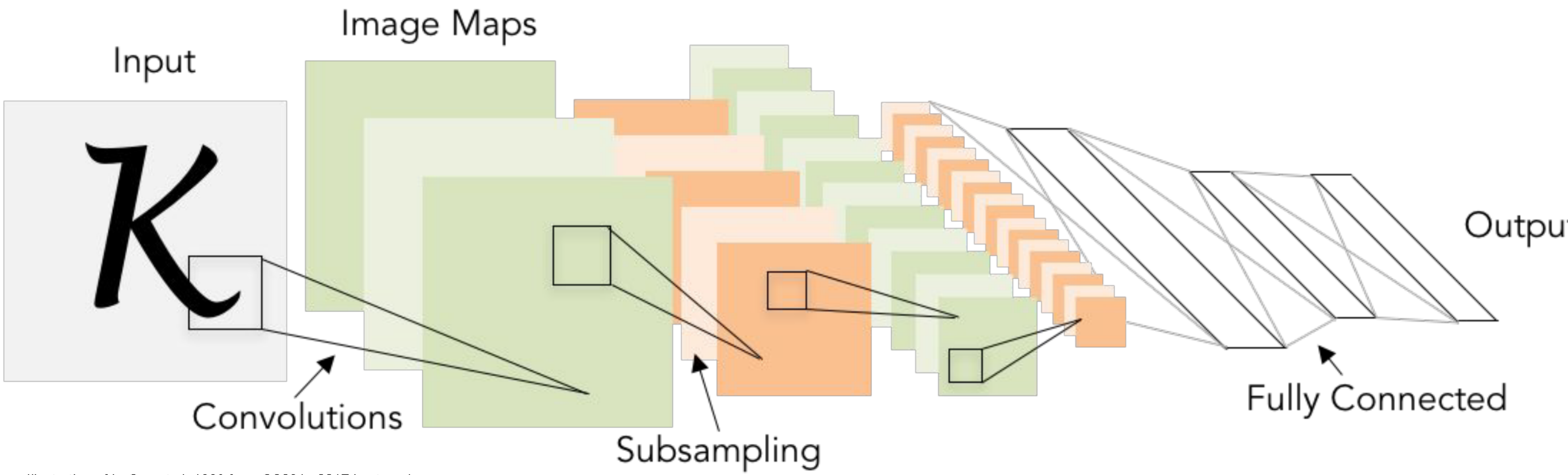
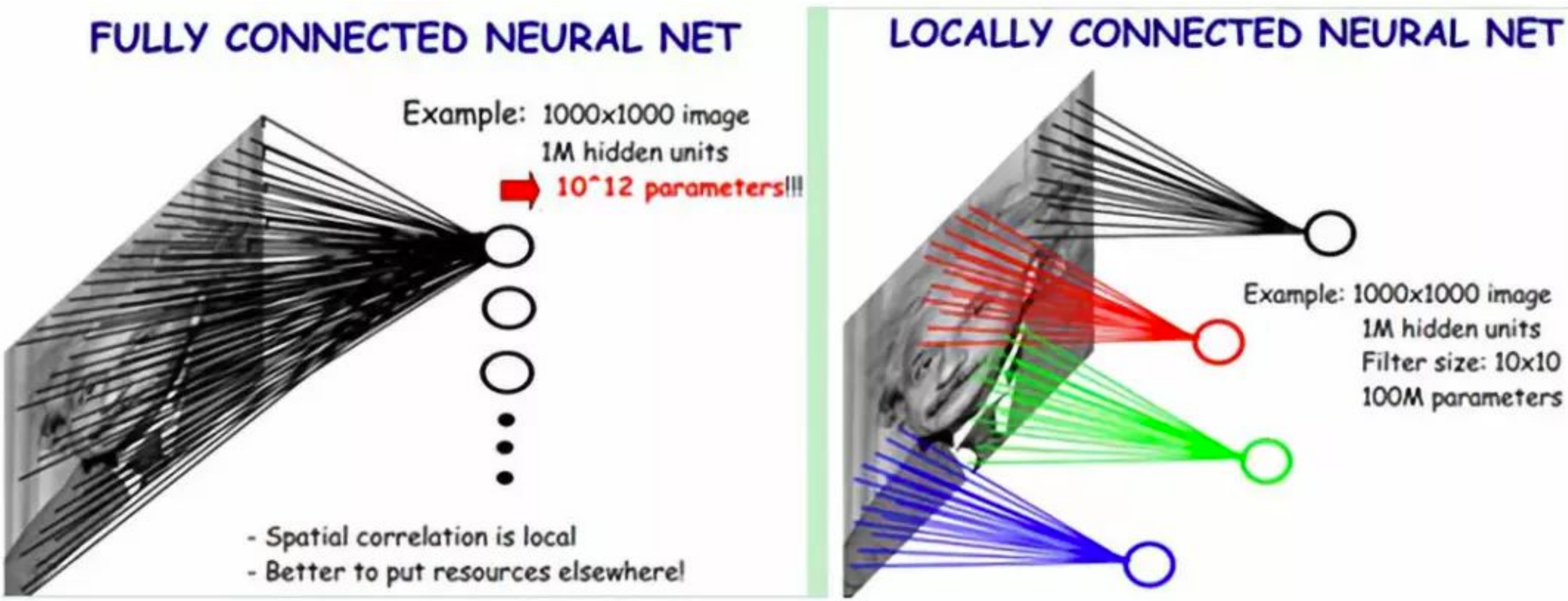
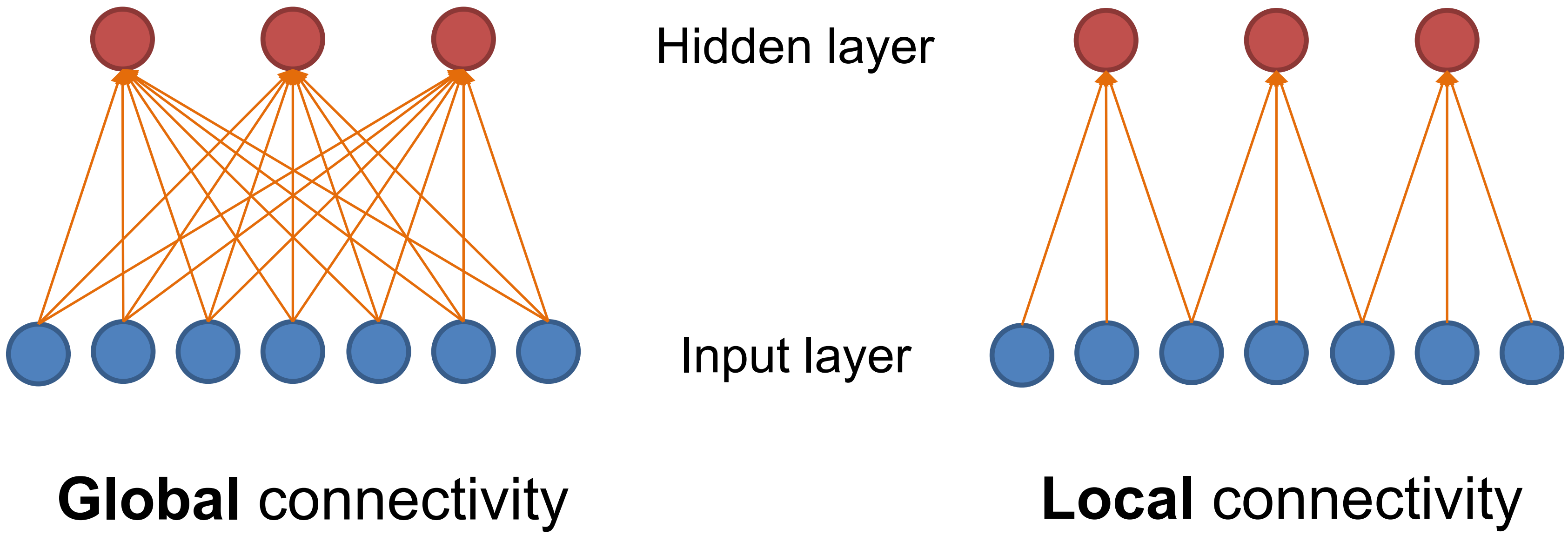


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Local features are important

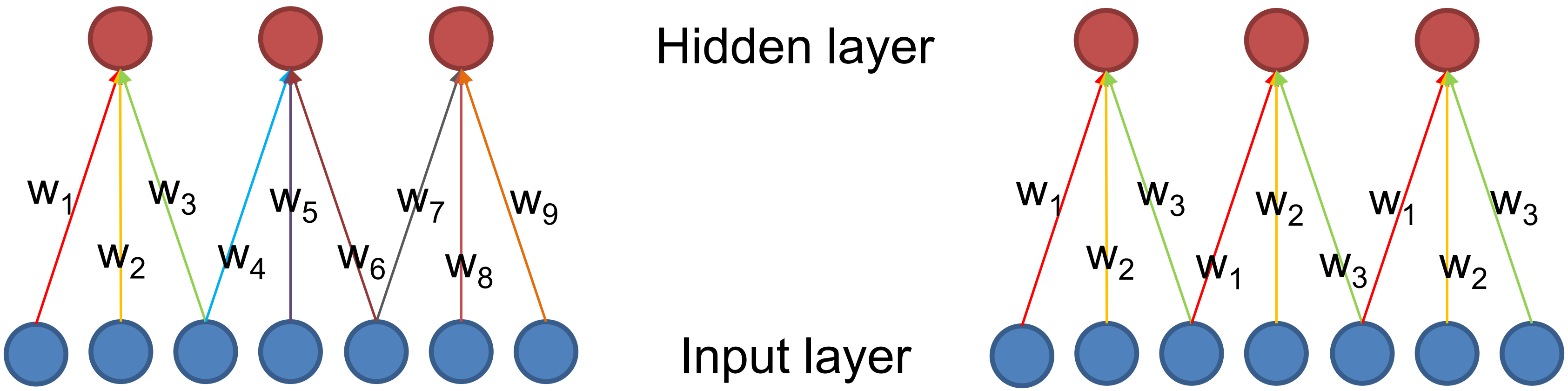

FULLY CONNECTED NEURAL NET
Example: 1000x1000 image
1M hidden units
10^12 parameters!!!
- Spatial correlation is local
- Better to put resources elsewhere!
LOCALLY CONNECTED NEURAL NET
Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

Local Connectivity



- # input units (neurons): 7
- # hidden units: 3
- Number of parameters
  - Global connectivity: 3 x 7 = 21
  - Local connectivity: 3 x 3 = 9

Weight Sharing



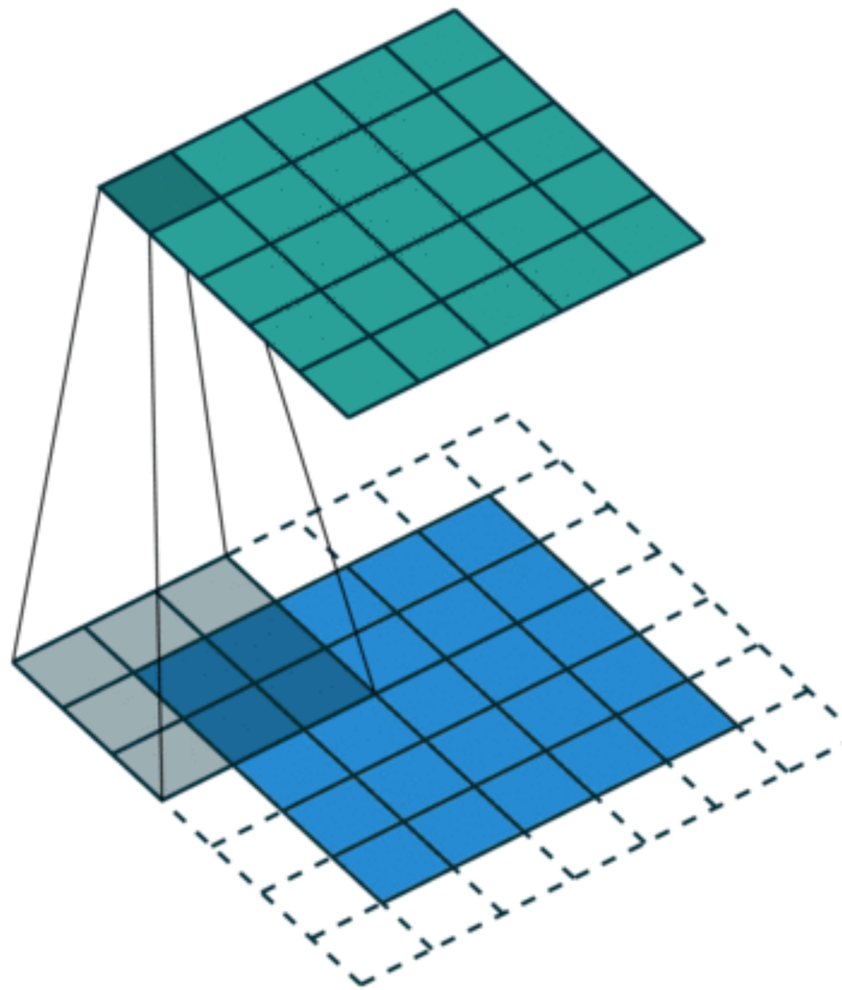
Without weight sharing

With weight sharing

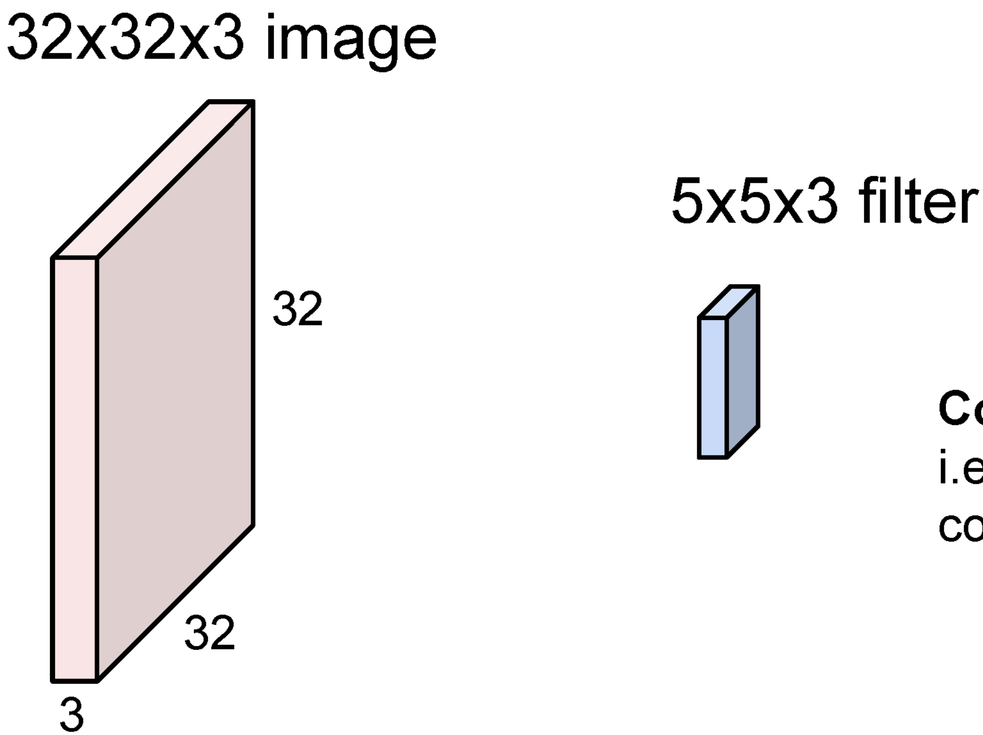
- # input units (neurons): 7
- # hidden units: 3
- Number of parameters
  - Without weight sharing: 3 x 3 = 9
  - With weight sharing : 3 x 1 = 3

Convolution layer

Local connectivity + weight sharing
= convolution!



Convolution Layer

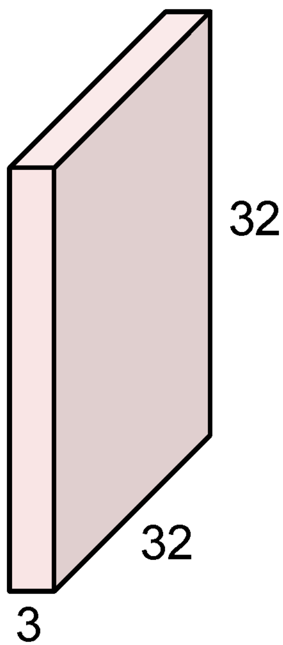


Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"

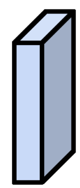
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



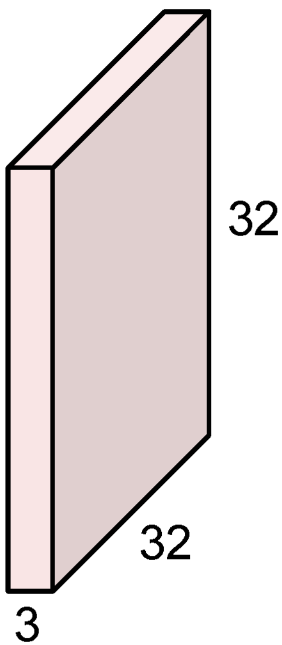
5x5x3 filter



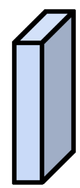
Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

Convolution Layer

32x32x3 image



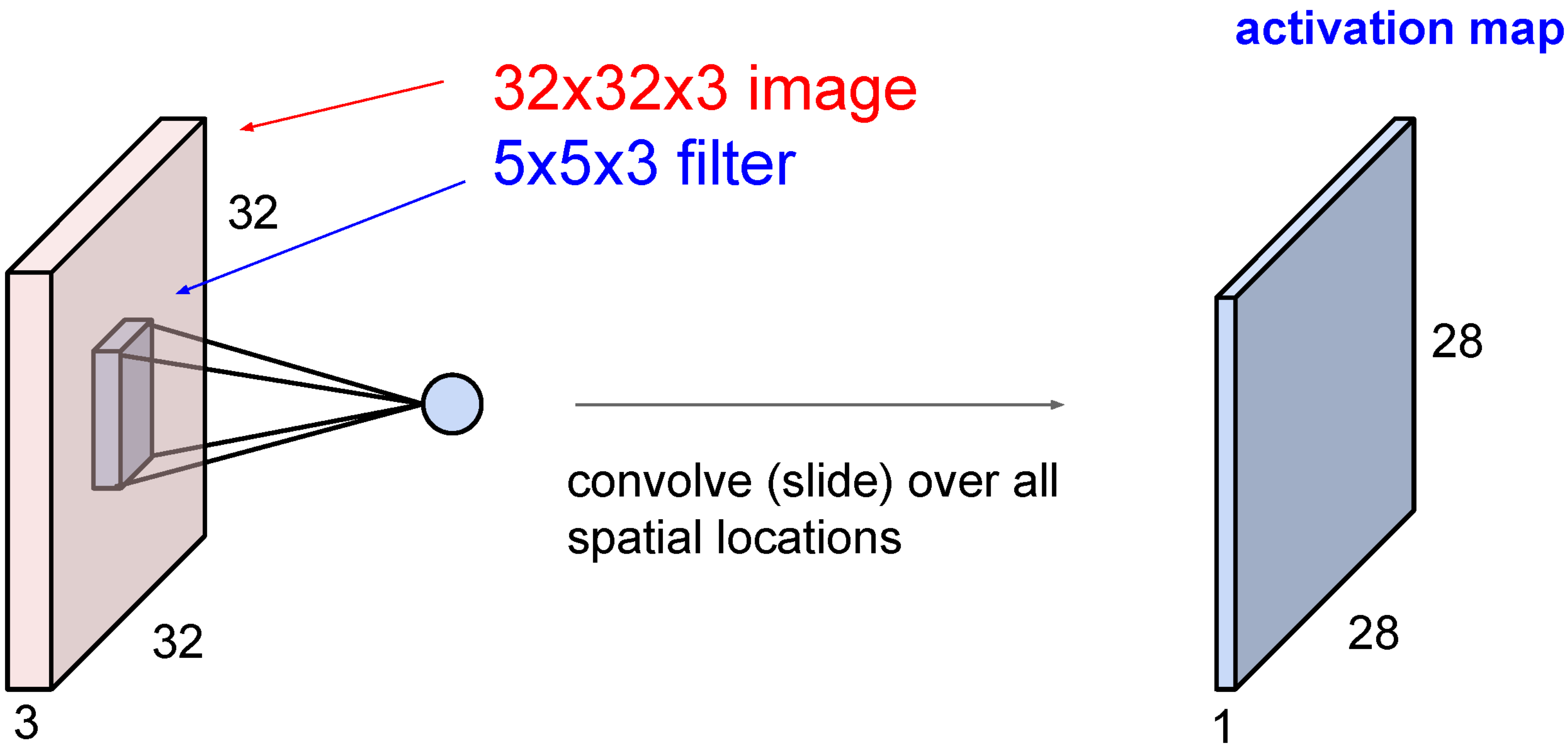
5x5x3 filter



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

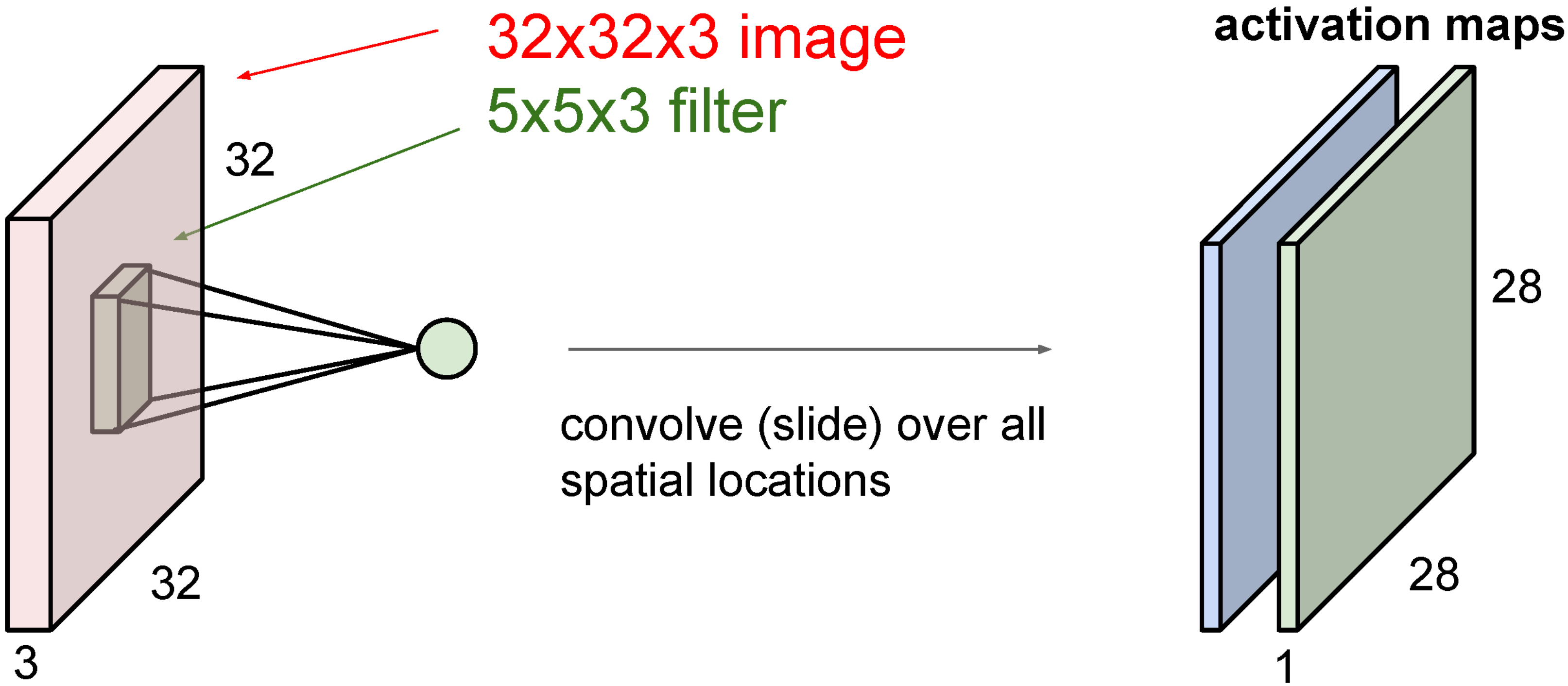
Number of weights: 5 x 5 x 3 + 1 = **76**
(vs. 3072 for a fully-connected layer)

Convolution Layer

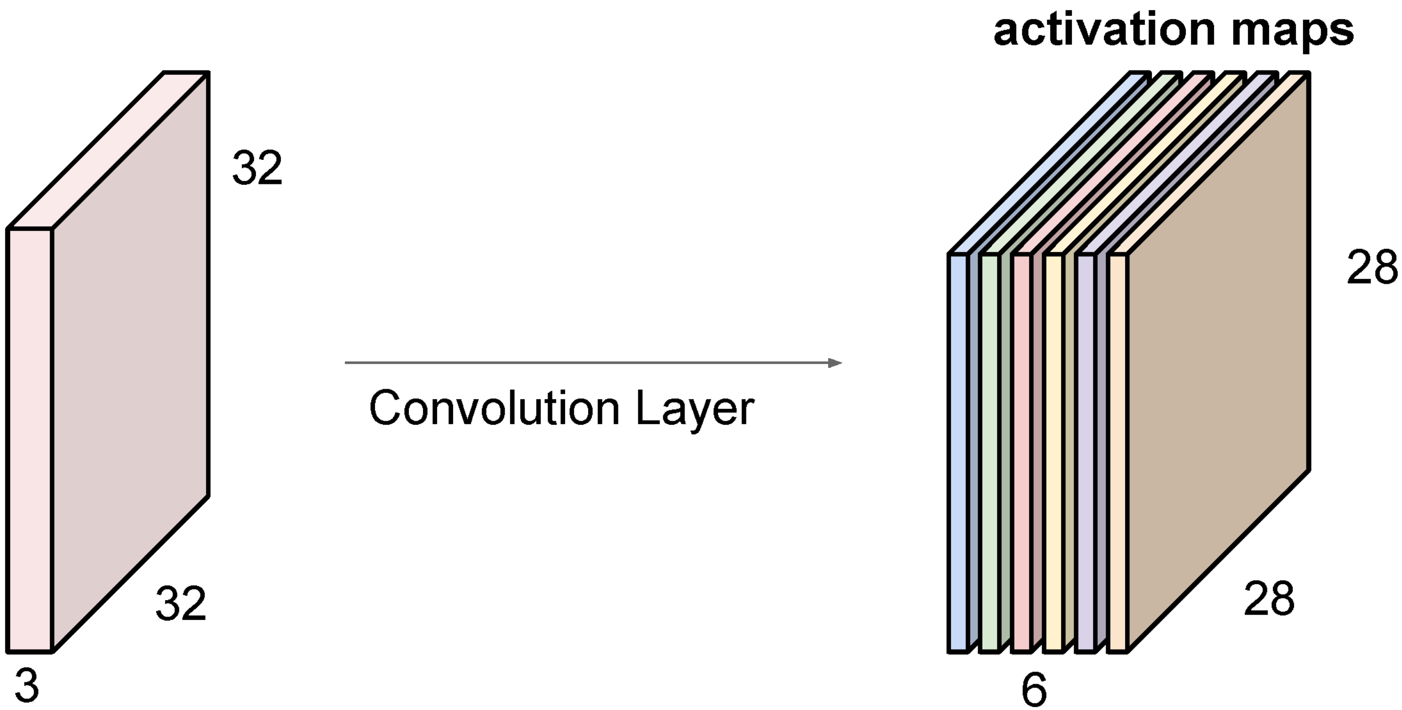


Convolution Layer

consider a second, green filter



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



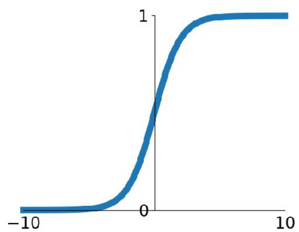
We stack these up to get a "new image" of size 28x28x6!

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

Activation functions

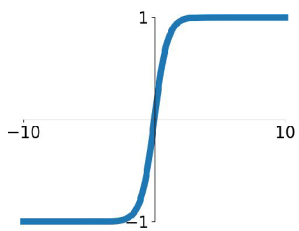
Sigmoid

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



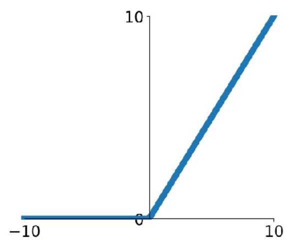
tanh

$\tanh(x)$



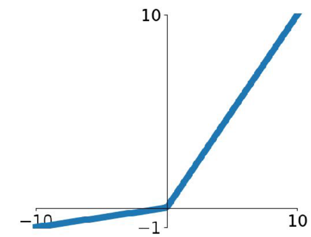
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

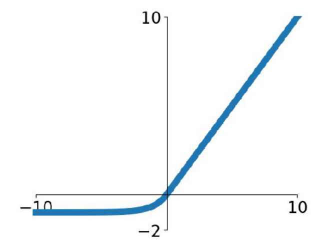


Maxout

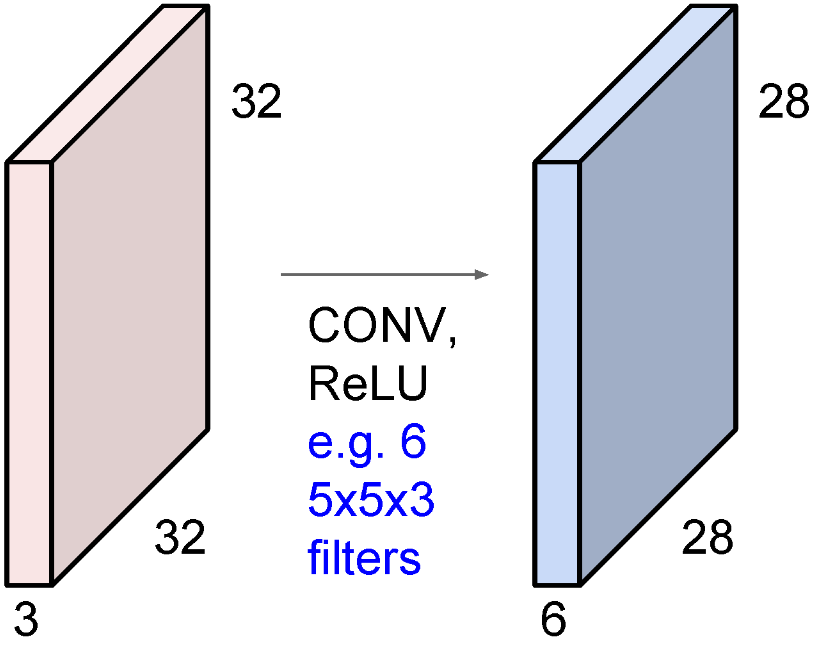
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

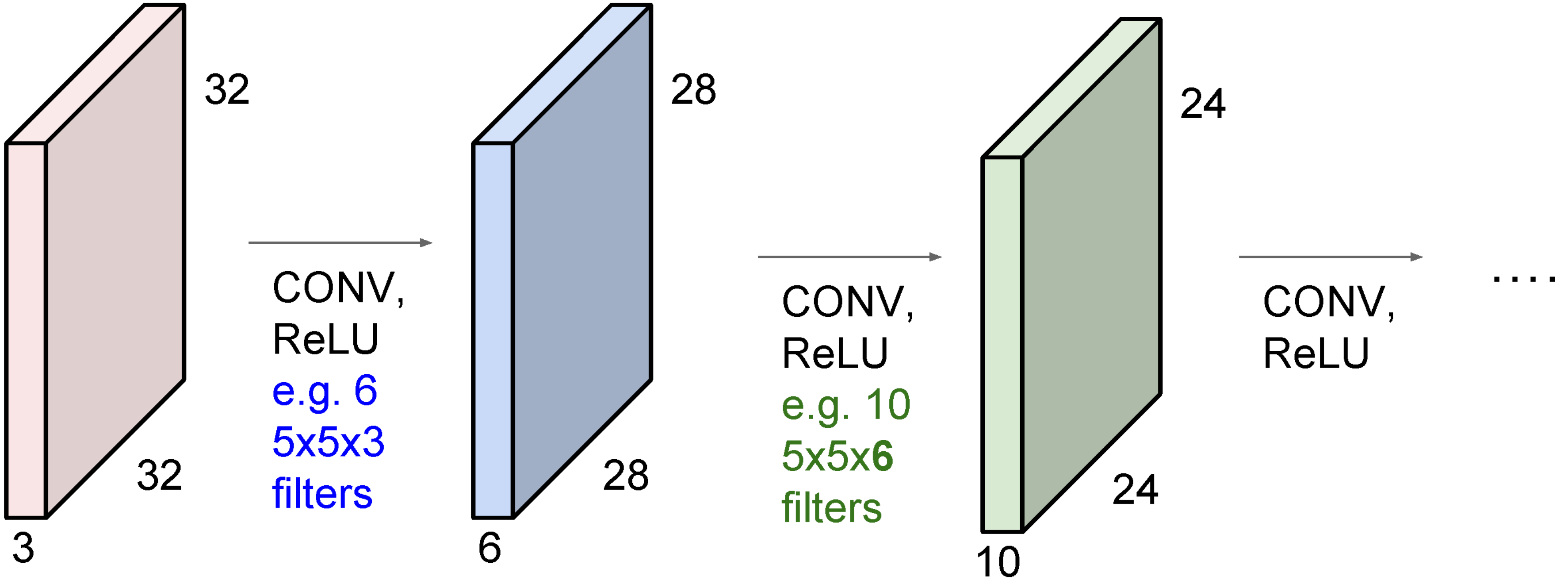
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



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

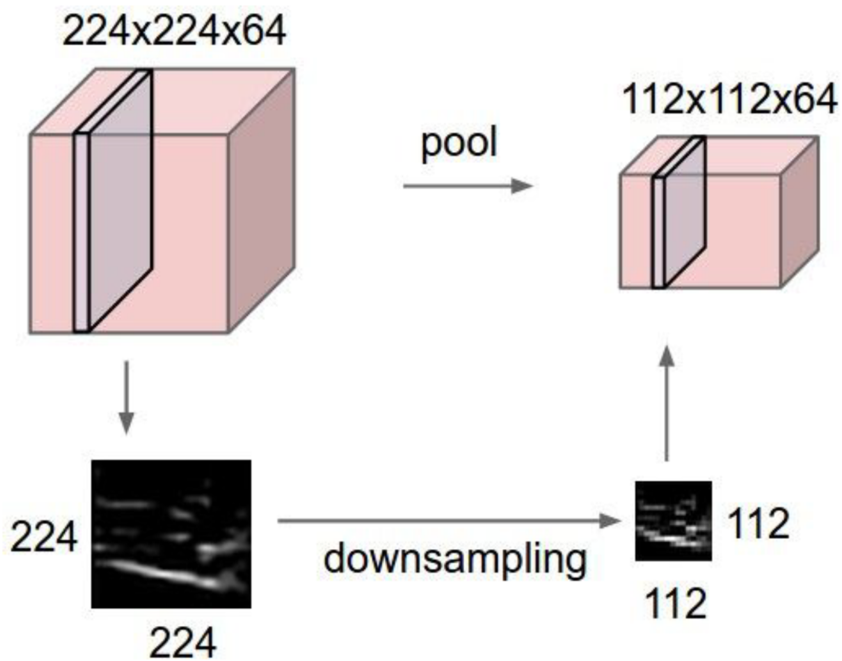


Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

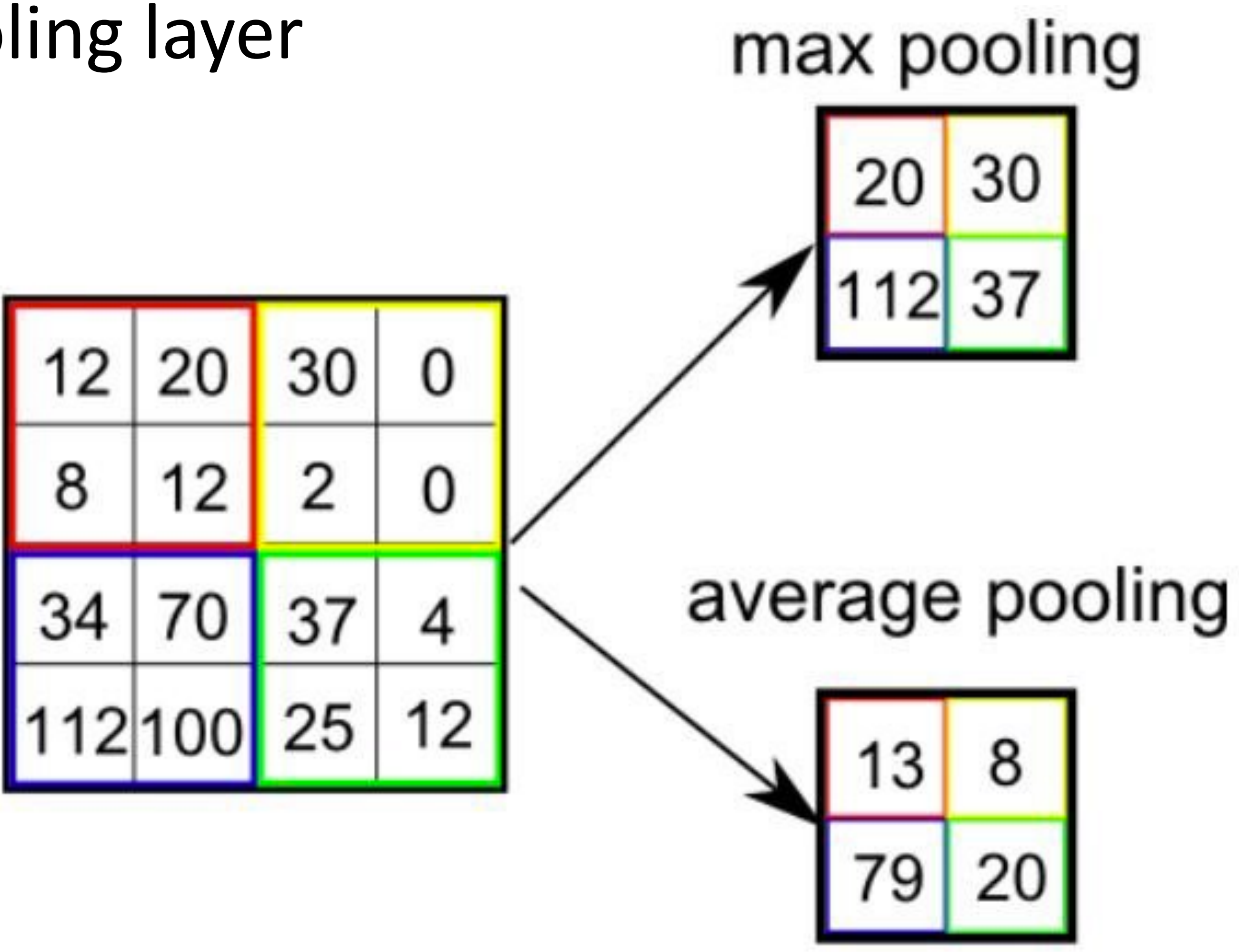


Pooling layer

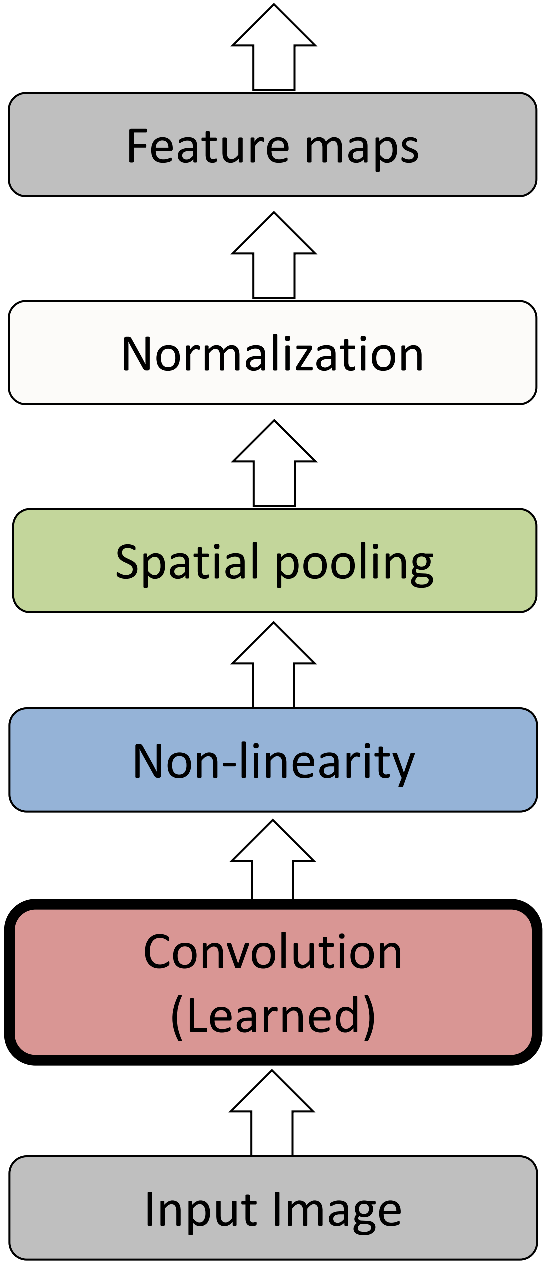
- makes the representations smaller and more manageable
- operates over each activation map independently:

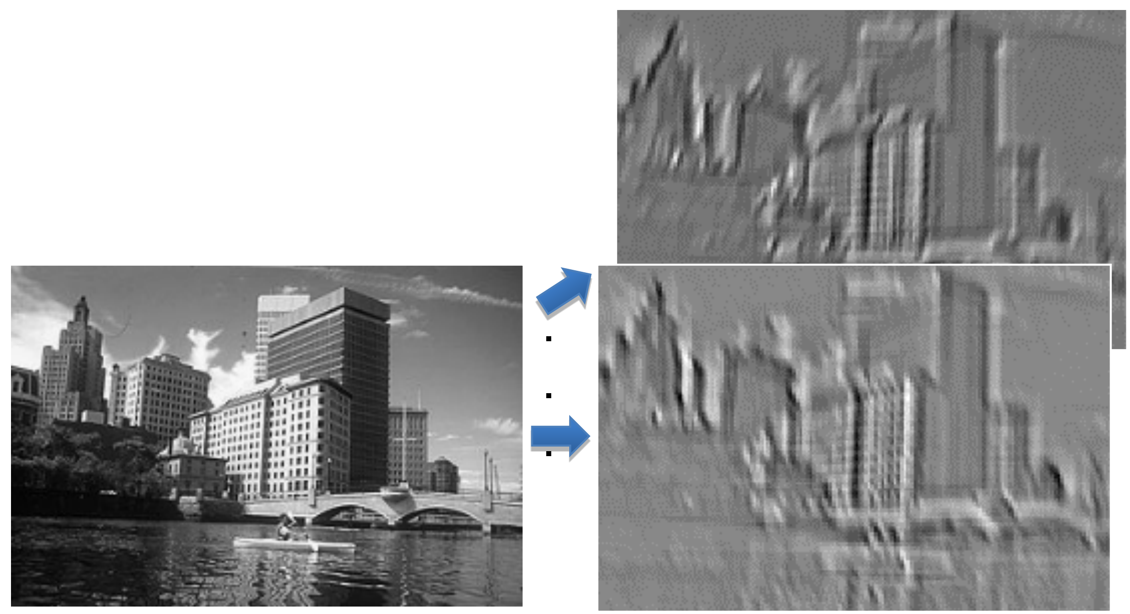


- Pooling layer



Convolutional Neural Networks



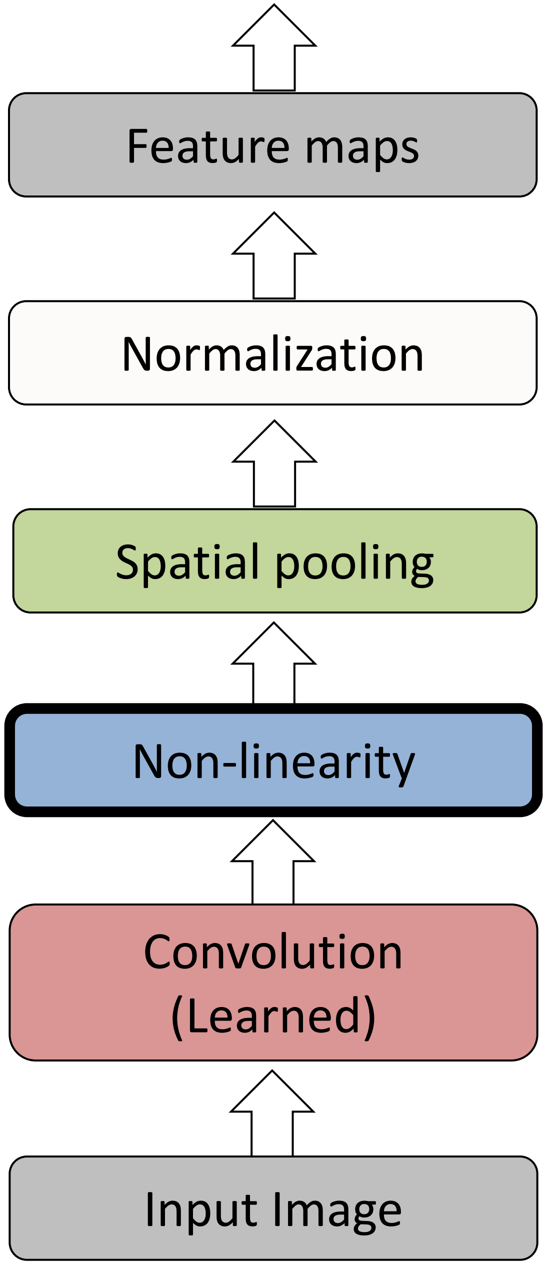


Input

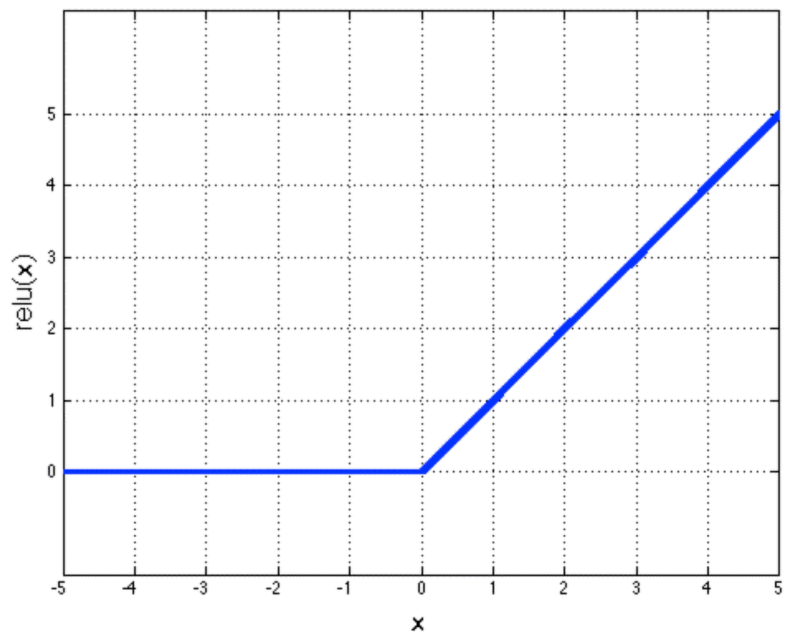
Feature Map

slide credit: S. Lazebnik

Convolutional Neural Networks

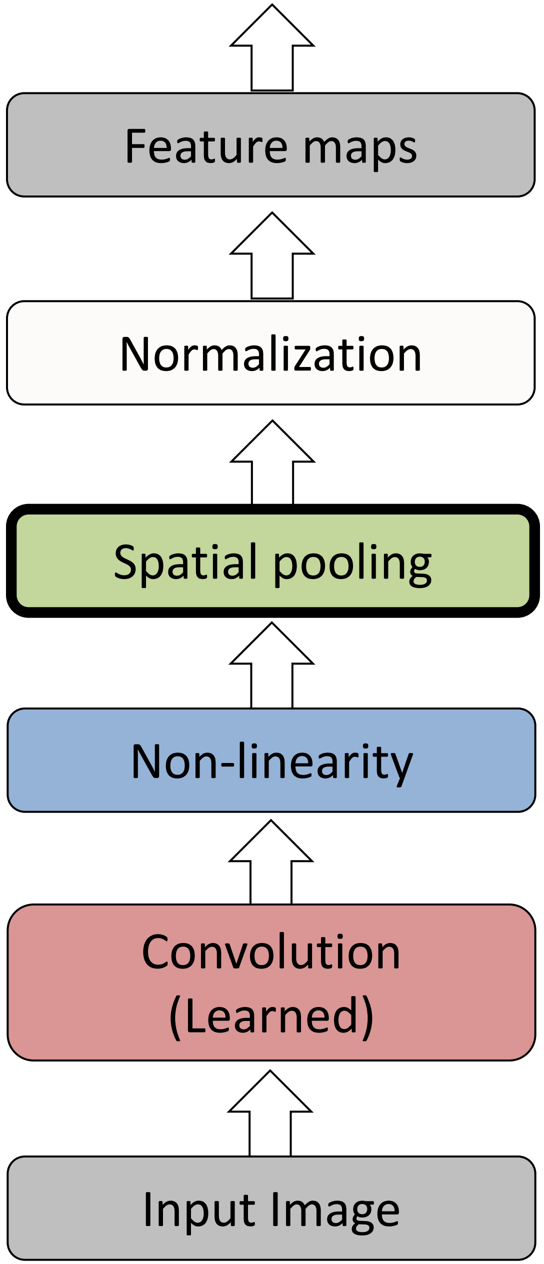


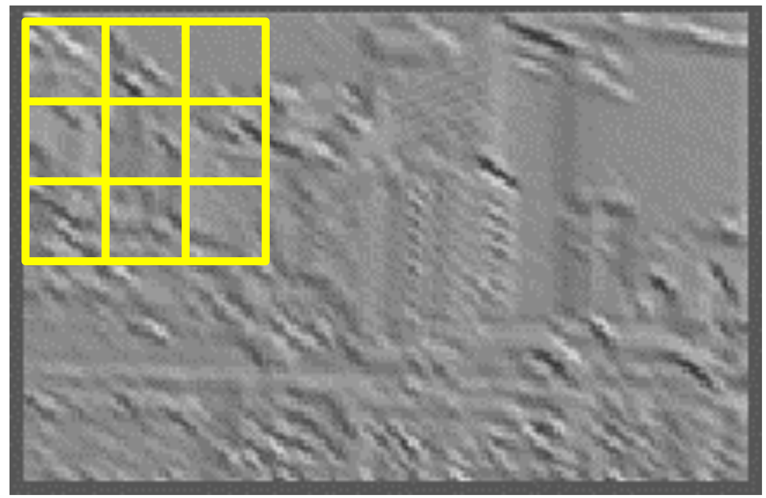
Rectified Linear Unit (ReLU)



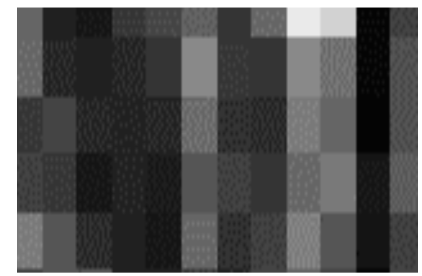
slide credit: S. Lazebnik

Convolutional Neural Networks





Max pooling

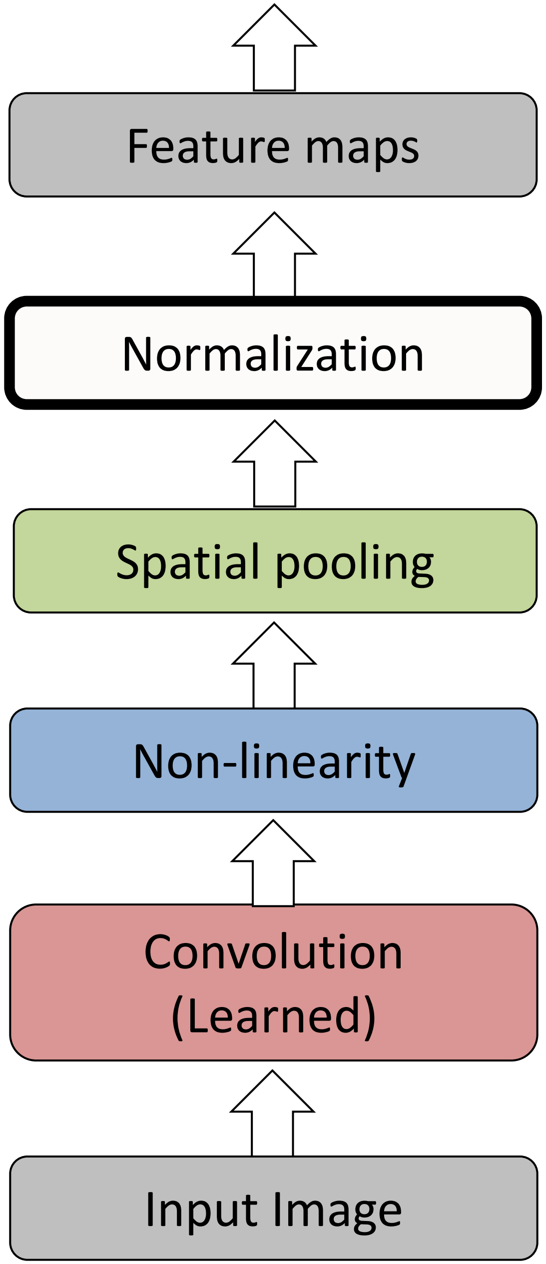


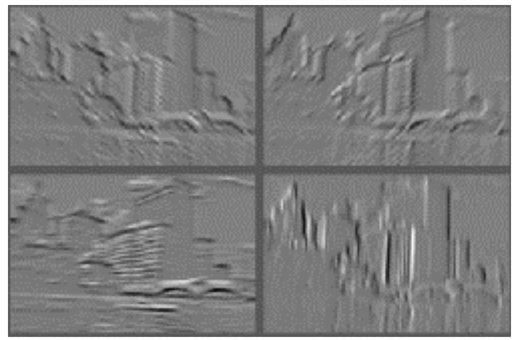
Max-pooling: a non-linear down-samp

Provide *translation invariance*

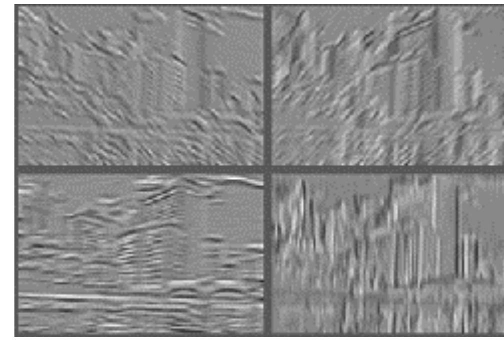
slide credit: S. Lazebnik

Convolutional Neural Networks





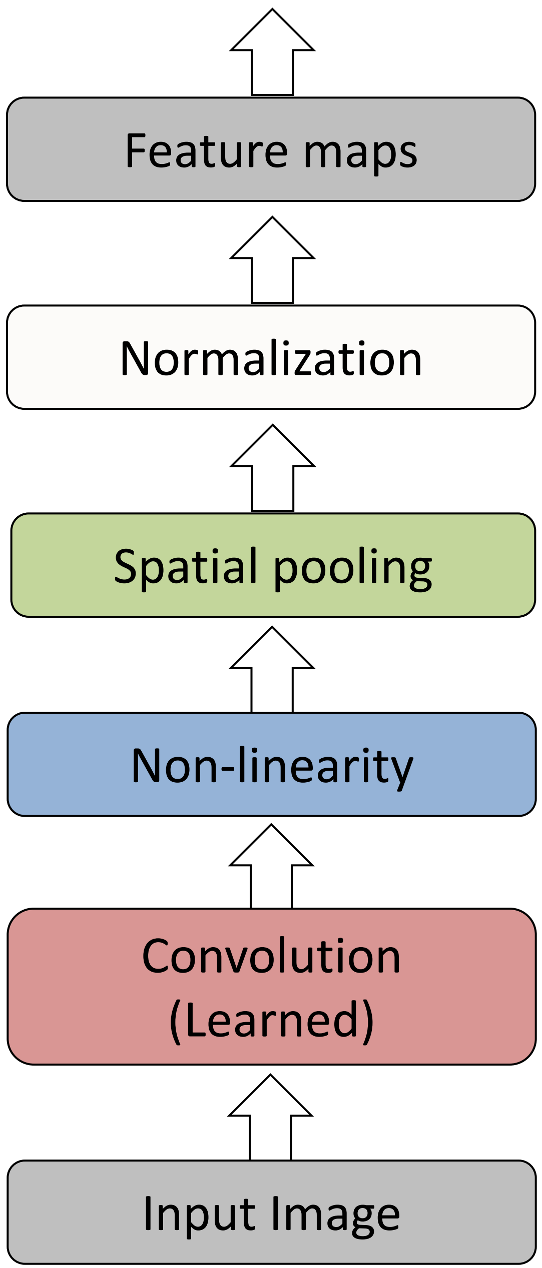
Feature Maps



Feature Maps After Contrast Normalization

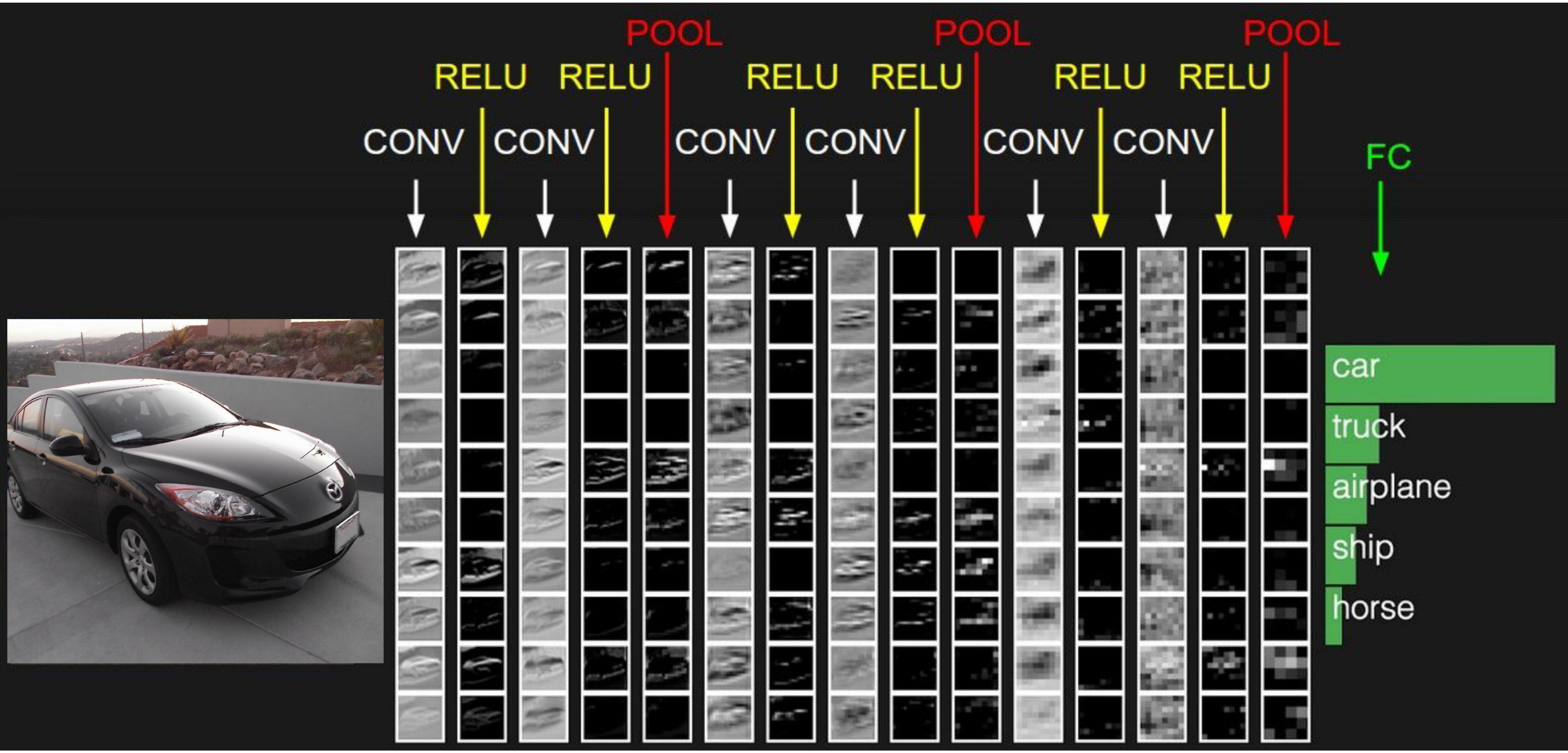
slide credit: S. Lazebnik

Convolutional Neural Networks



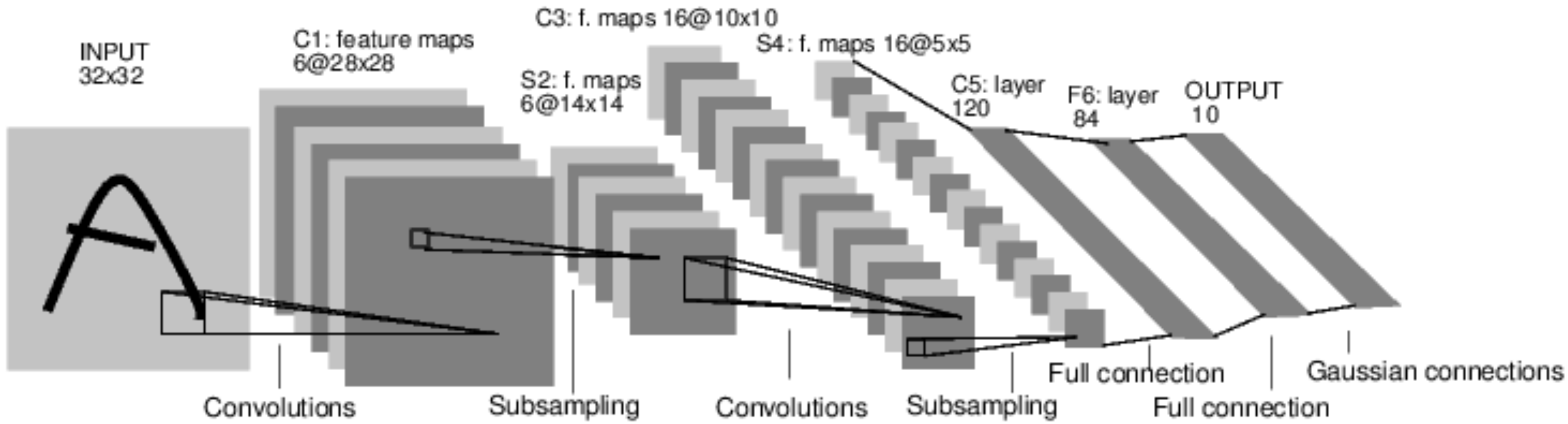
slide credit: S. Lazebnik

preview:


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

Case Study: LeNet-5

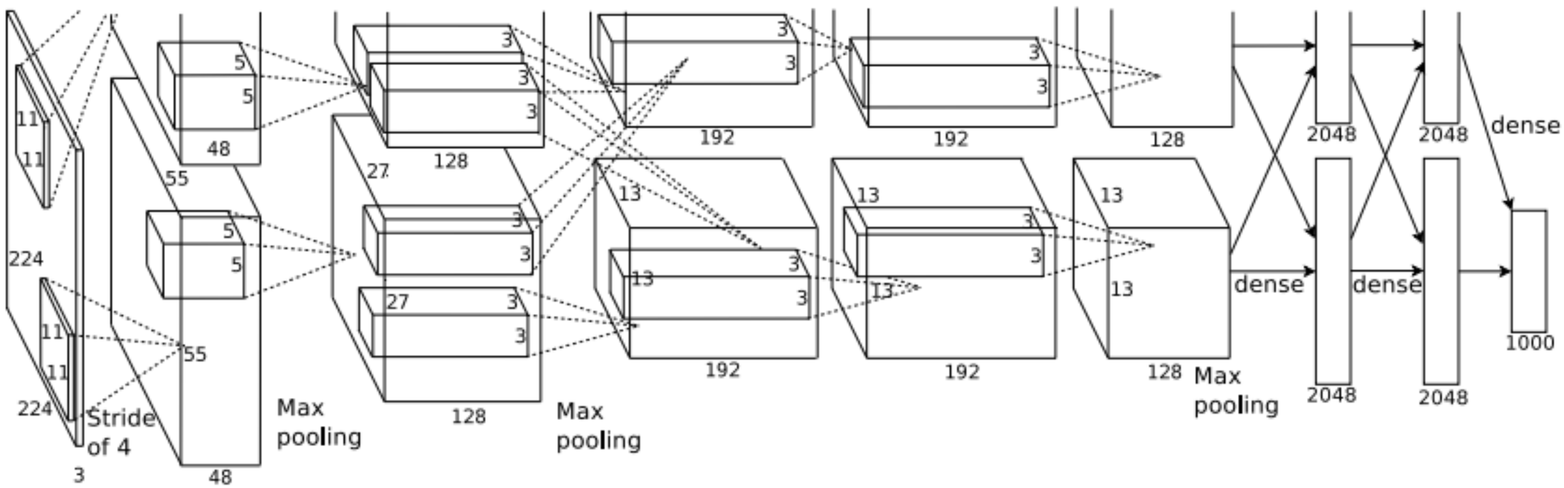
[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Case Study: AlexNet

[Krizhevsky et al. 2012]

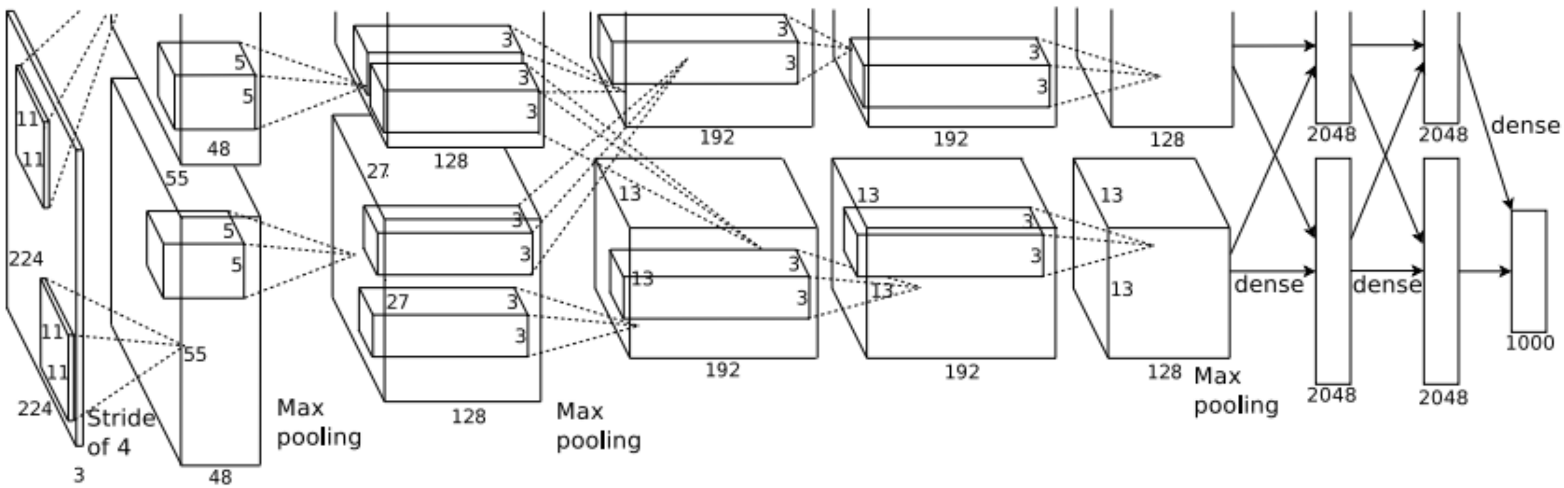


Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Case Study: AlexNet

[Krizhevsky et al. 2012]



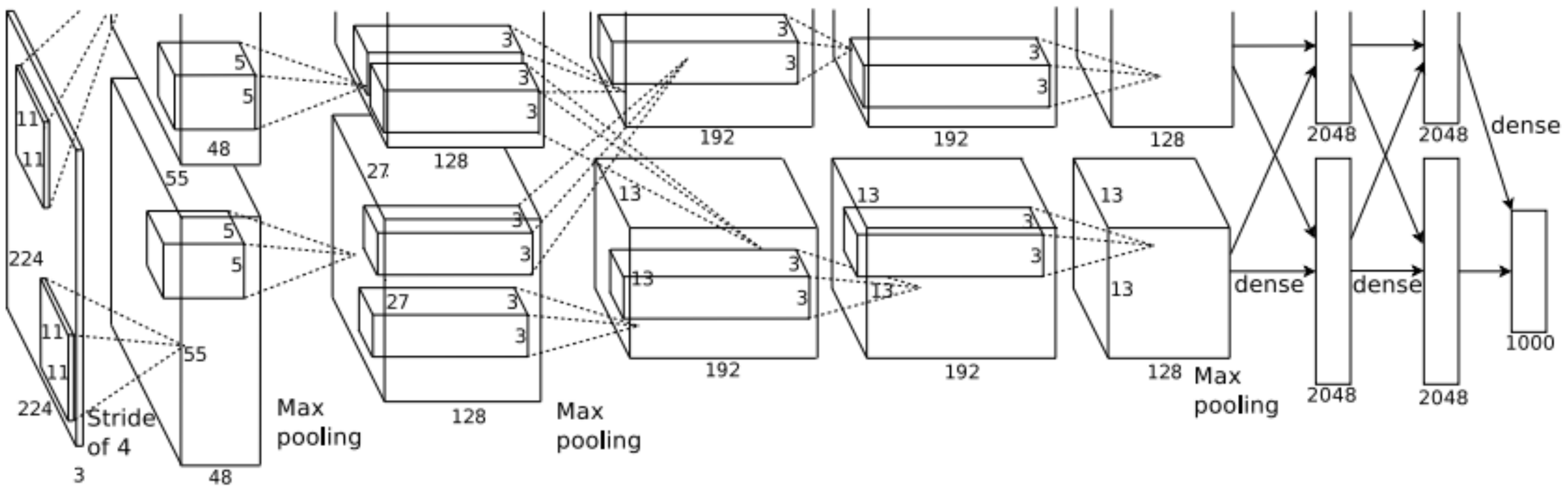
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

Case Study: AlexNet

[Krizhevsky et al. 2012]

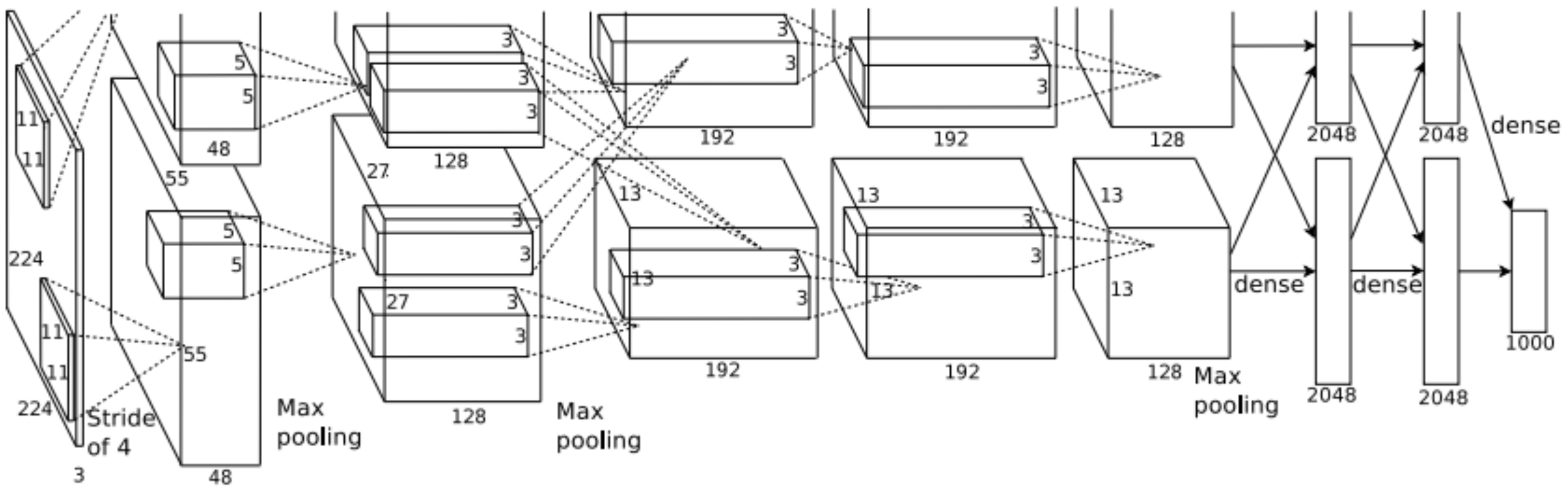


Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

Case Study: AlexNet

[Krizhevsky et al. 2012]



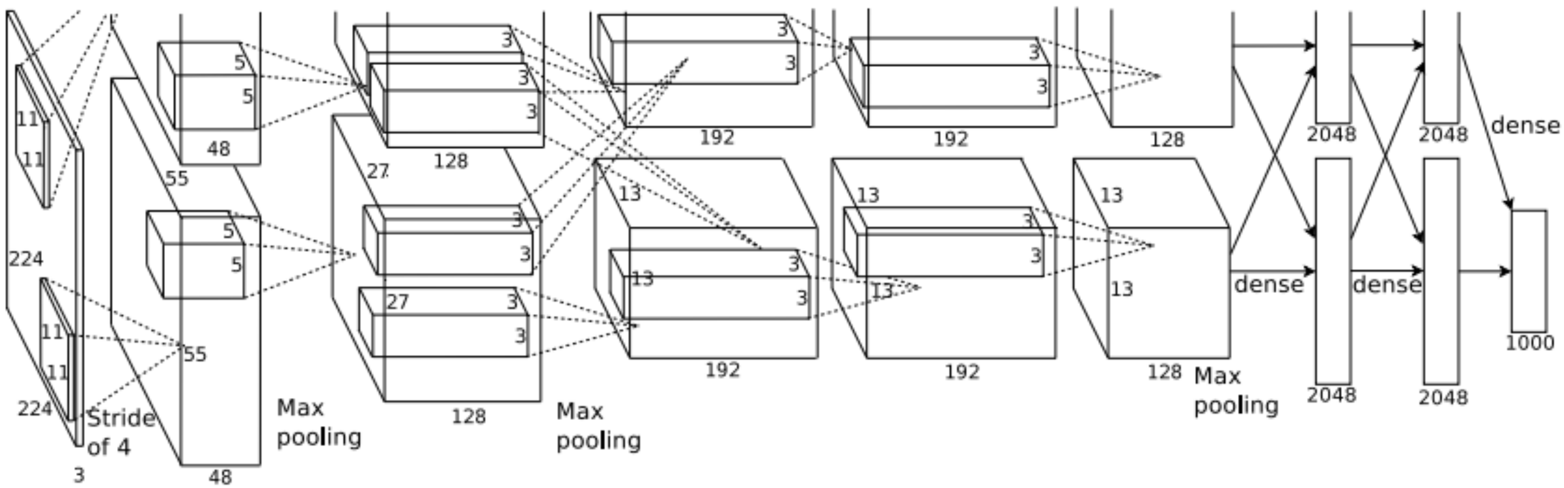
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: (55-3)/2+1 = 27

Case Study: AlexNet

[Krizhevsky et al. 2012]



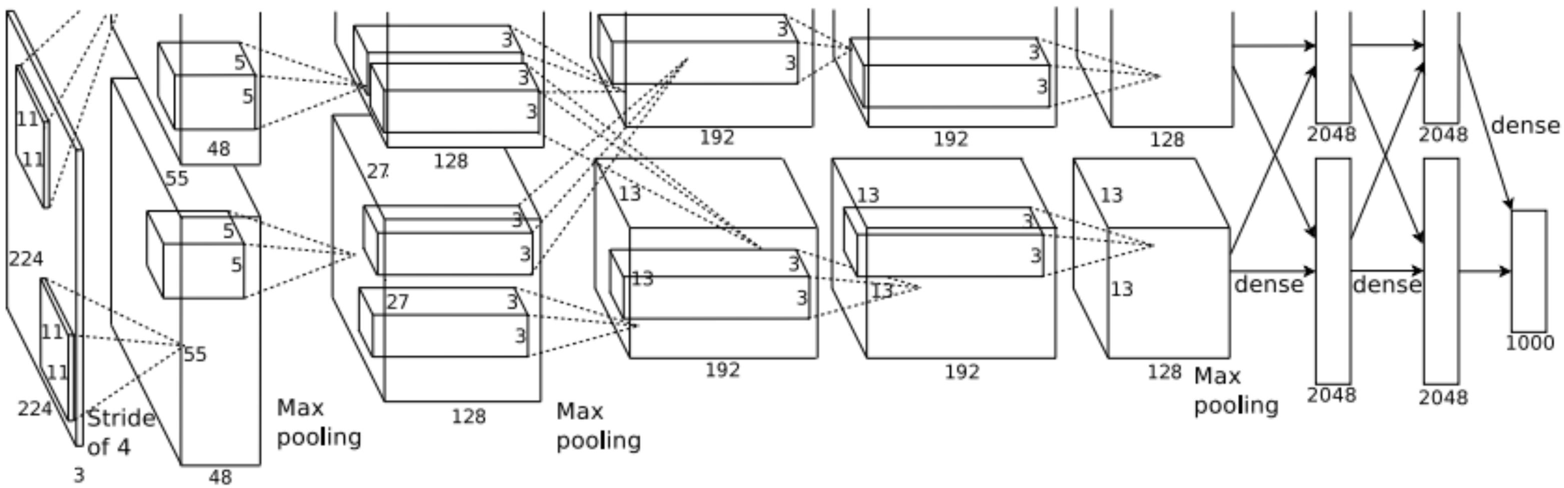
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

Q: what is the number of parameters in this layer?

Case Study: AlexNet

[Krizhevsky et al. 2012]

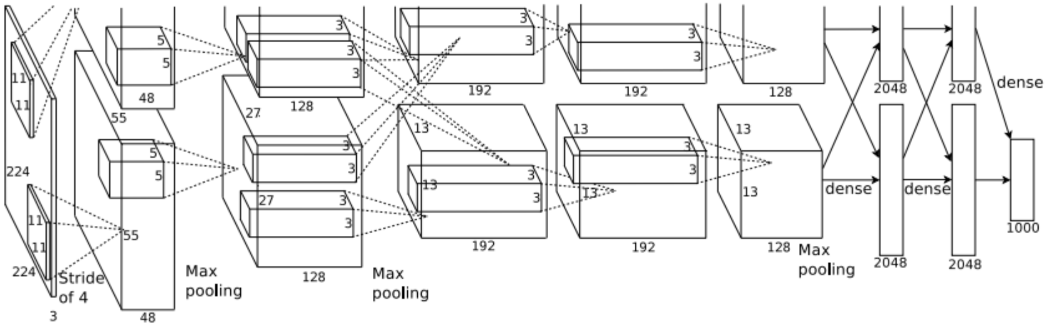


Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

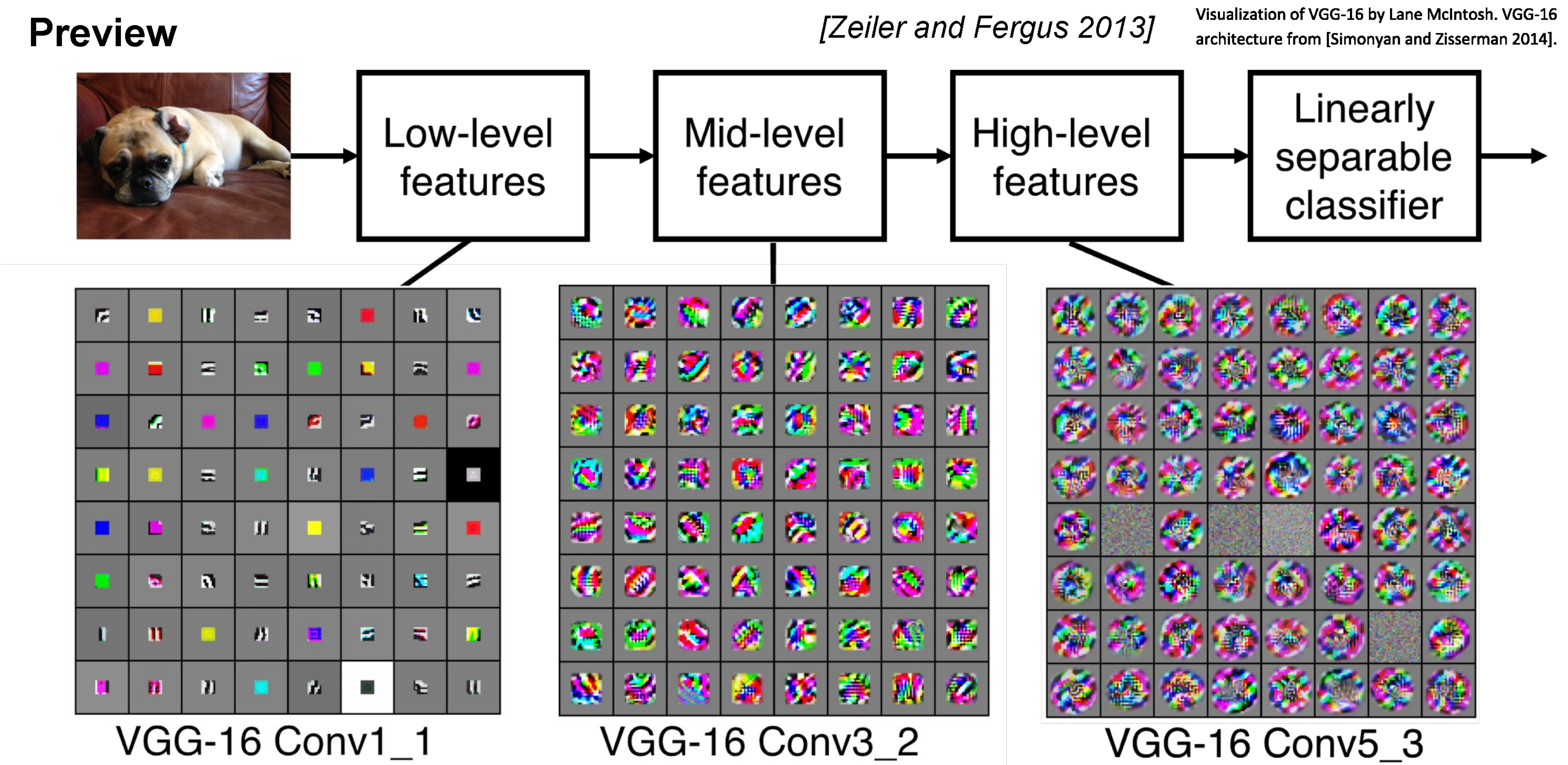
Case Study: AlexNet

[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

What do the filters look like?



Summary

- Multi-layer neural network is a non-linear classifier

- CNN = a multi-layer neural network with

  Local connectivity

  Weight sharing

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

Part V

Training CNNs

Multi-layer Neural Network

- A non-linear classifier
- **Training:** find network weights **w** to minimize the error between true training labels y_i and estimated labels $f_{\boldsymbol{w}}(\boldsymbol{x_i})$,

$$L(\boldsymbol{w}) = \frac{1}{n} \sum_i l(y_i, f_{\boldsymbol{w}}(\boldsymbol{x_i}))$$

For example:

  - L2 loss for regression
  - Cross-entropy loss for classification

- Minimization can be done by gradient descent provided f is differentiable
- This training method is called **back-propagation**

Training CNN with gradient descent

- A CNN as composition of functions

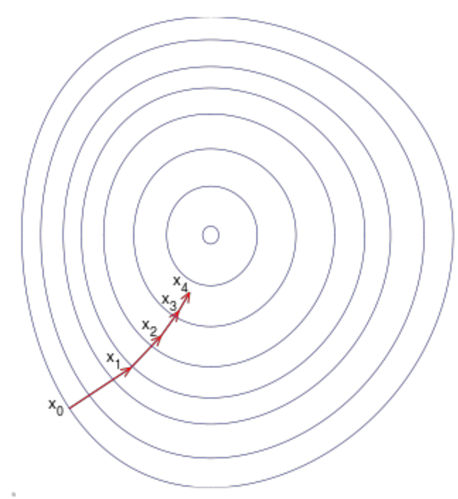
$$f_{\boldsymbol{w}}(\boldsymbol{x}) = f_L(\dots\ (f_2(f_1(\boldsymbol{x}; \boldsymbol{w}_1); \boldsymbol{w}_2)\ \dots; \boldsymbol{w}_L)$$

- Parameters

$$\boldsymbol{w} = (\boldsymbol{w_1}, \boldsymbol{w_2}, \dots \boldsymbol{w_L})$$

- Empirical loss function

$$L(\boldsymbol{w}) = \frac{1}{n}\sum_i l(y_i, f_{\boldsymbol{w}}(\boldsymbol{x_i}))$$



- Gradient descent

$$\boldsymbol{w^{t+1}} = \boldsymbol{w^t} - \eta_t \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{w}}(\boldsymbol{w^t})$$

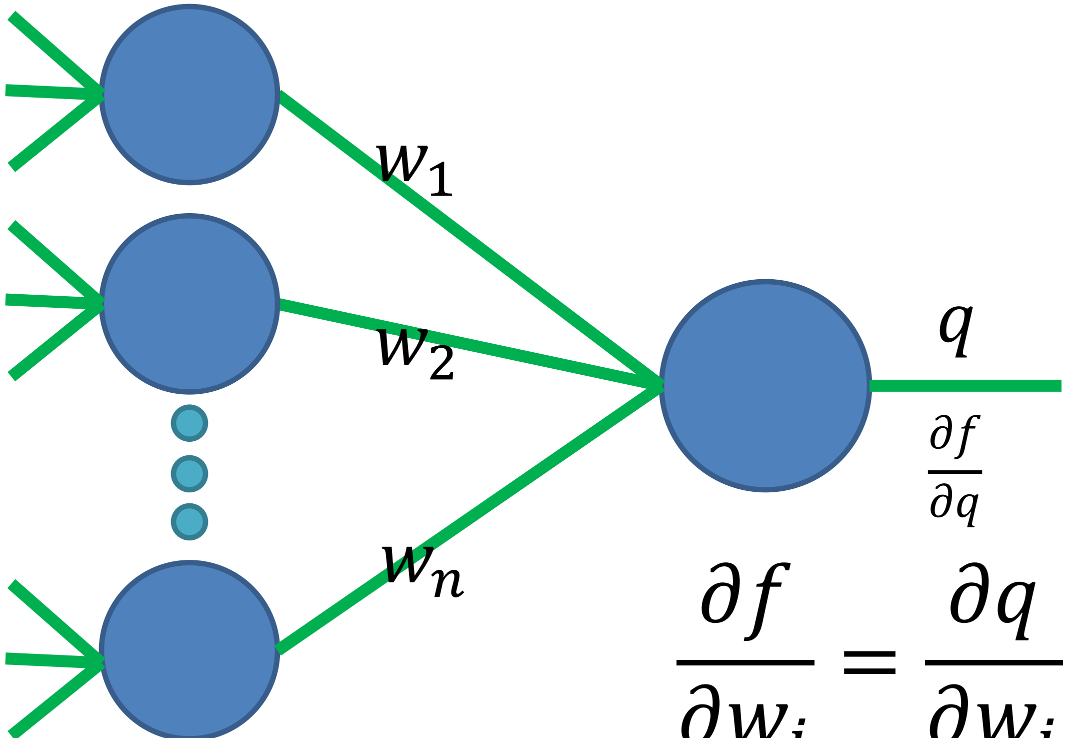
New weight

Old weight

Learning rate

Gradient

Backpropagation (recursive chain rule)



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

Local gradient

Gate gradient

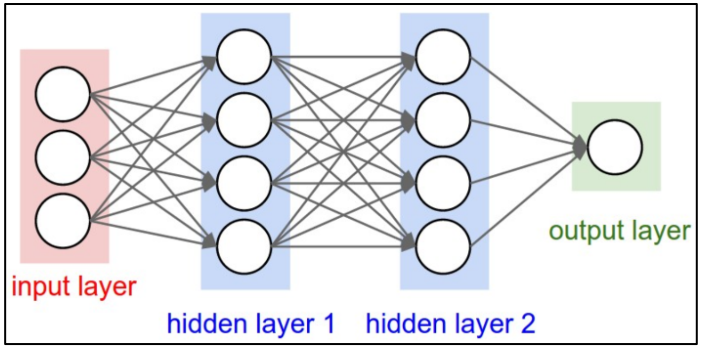
Can be computed during forward pass

The gate receives this during backprop

Stochastic Gradient Descent (SGD)

Loop:

1. **Sample** a batch of data
2. **Forward** prop it through the graph, get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient



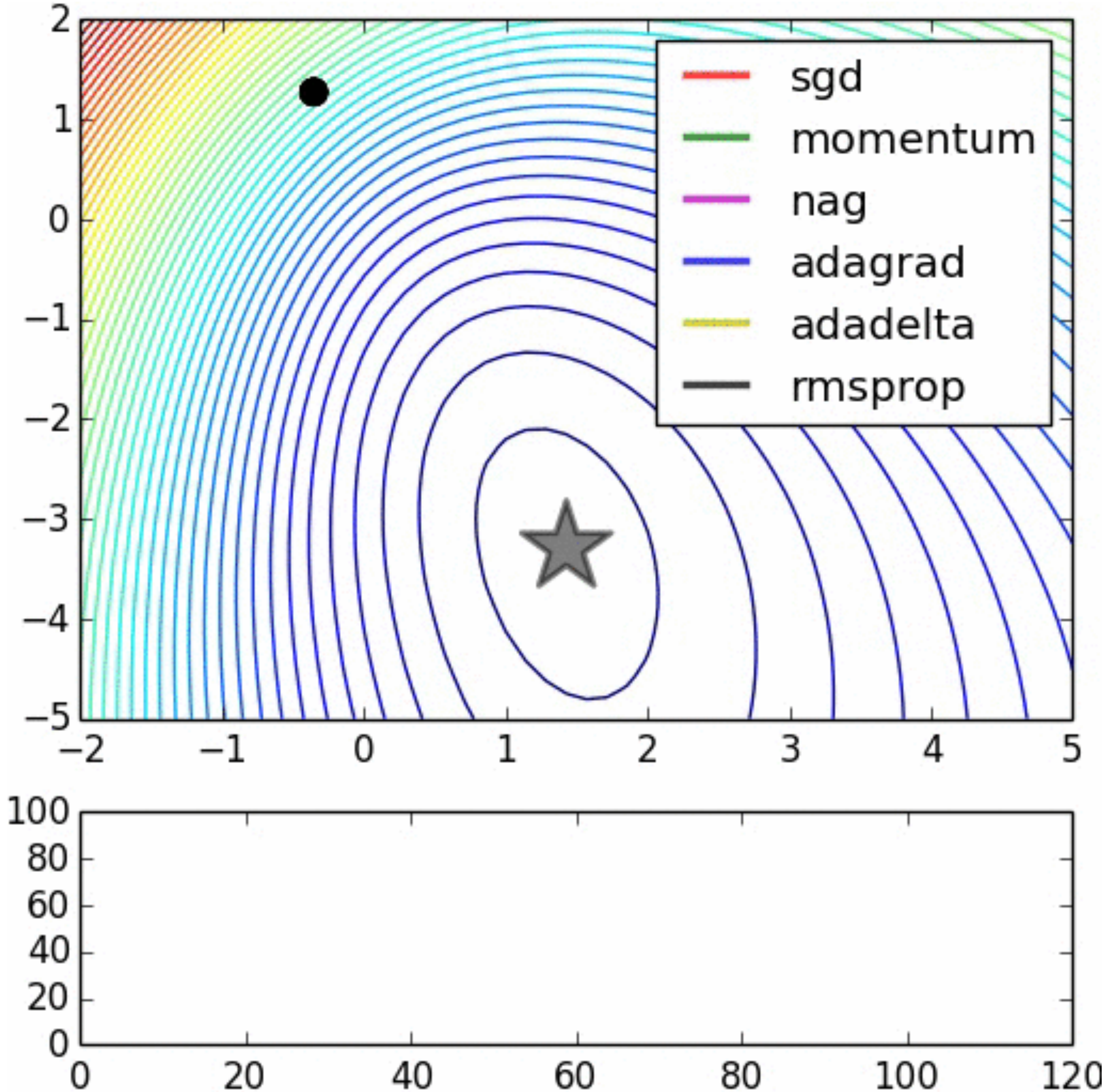


Image credits: Alec Radford

Architecture and hyper-parameters

- How many layers to use?
- How many filters in each layer?
- What are the best batch size and learning rate?

- How do we set them?
  - One option: try them all and see what works best

Data split

Idea #1: Choose hyperparameters that work best on the data

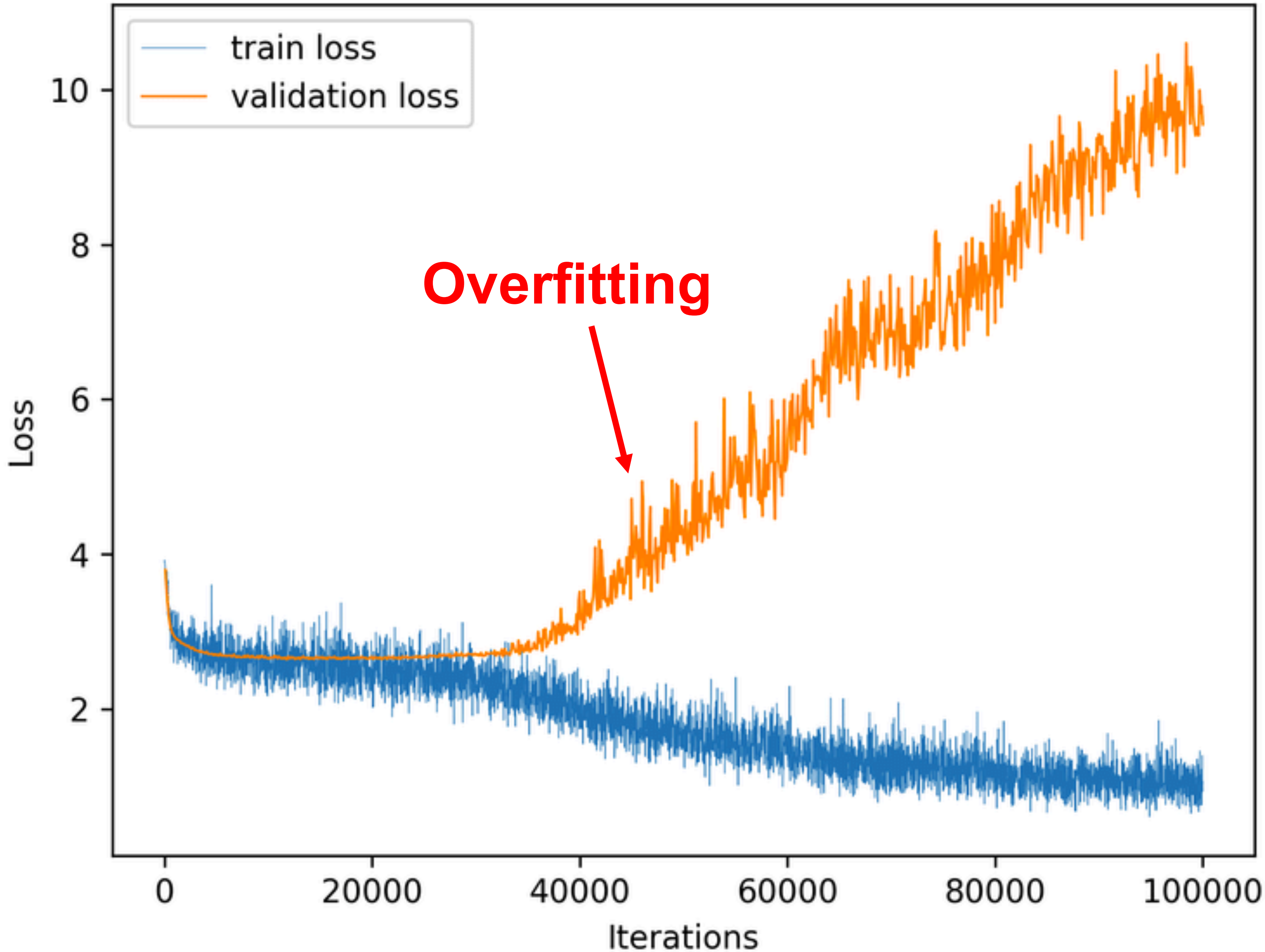
Bad: you can always decrease training loss by using a larger network

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

Bad: no idea how it will perform on new data

Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Best!

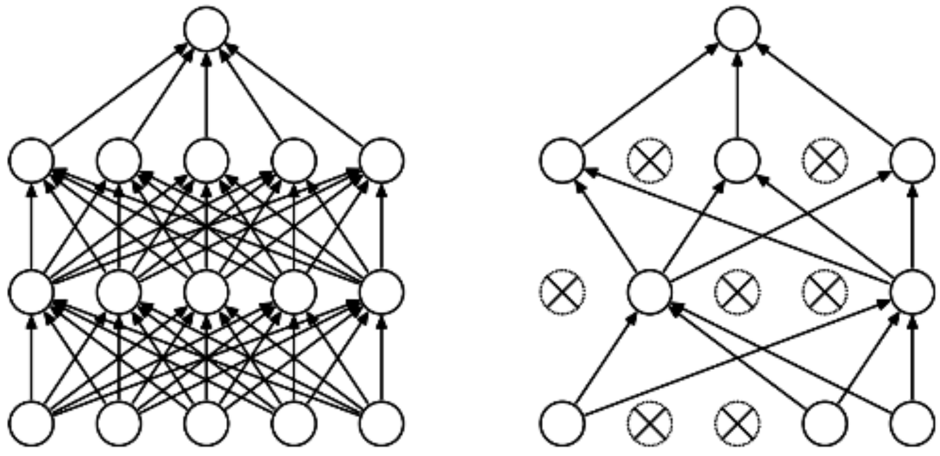


train loss
validation loss
Overfitting
Loss
Iterations
10
8
6
4
2
0
20000
40000
60000
80000
100000

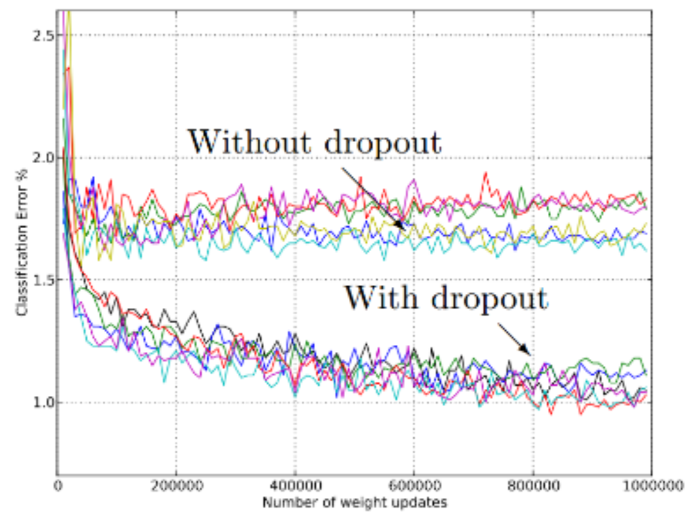
Recap: How to pick hyperparameters?

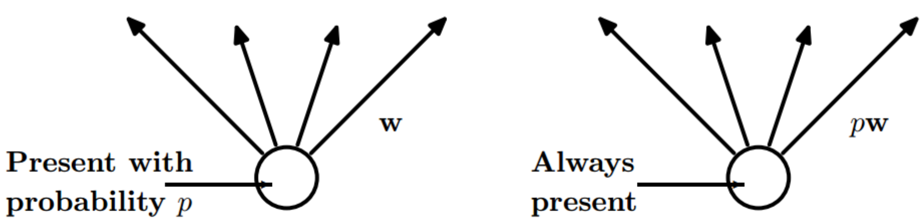
- Train for original model
- Validate to find hyperparameters
- Test to understand generalizability

Dropout



(a) Standard Neural Net (b) After applying dropout.





(a) At training time (b) At test time

Main Idea: approximately combining exponentially many different neural network architectures efficiently

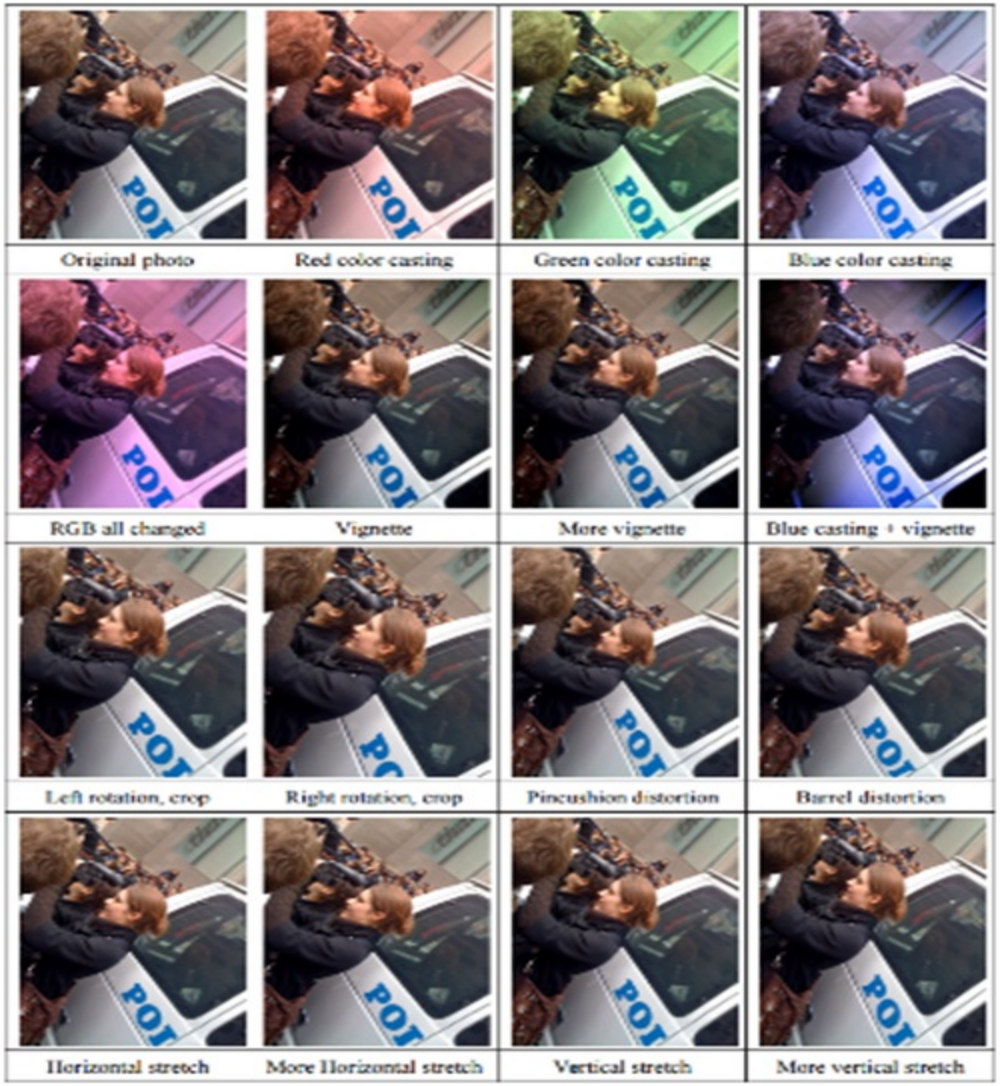
Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SVM on Fisher Vectors of Dense SIFT and Color Statistics	-	-	27.3
Avg of classifiers over FVs of SIFT, LBP, GIST and CSIFT	-	-	26.2
Conv Net + dropout (Krizhevsky et al., 2012)	40.7	18.2	-
Avg of 5 Conv Nets + dropout (Krizhevsky et al., 2012)	38.1	16.4	16.4

Table 6: Results on the ILSVRC-2012 validation/test set.

Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

Data Augmentation (Jittering)

- Create *virtual* training samples
  - Horizontal flip
  - Random crop
  - Color casting
  - Geometric distortion



Deep Image [Wu et al. 2015]

Part VI

History and Recent Advances

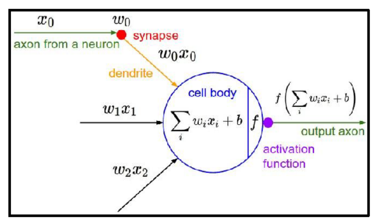
A bit of history...

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

A bit of history...

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

recognizable math

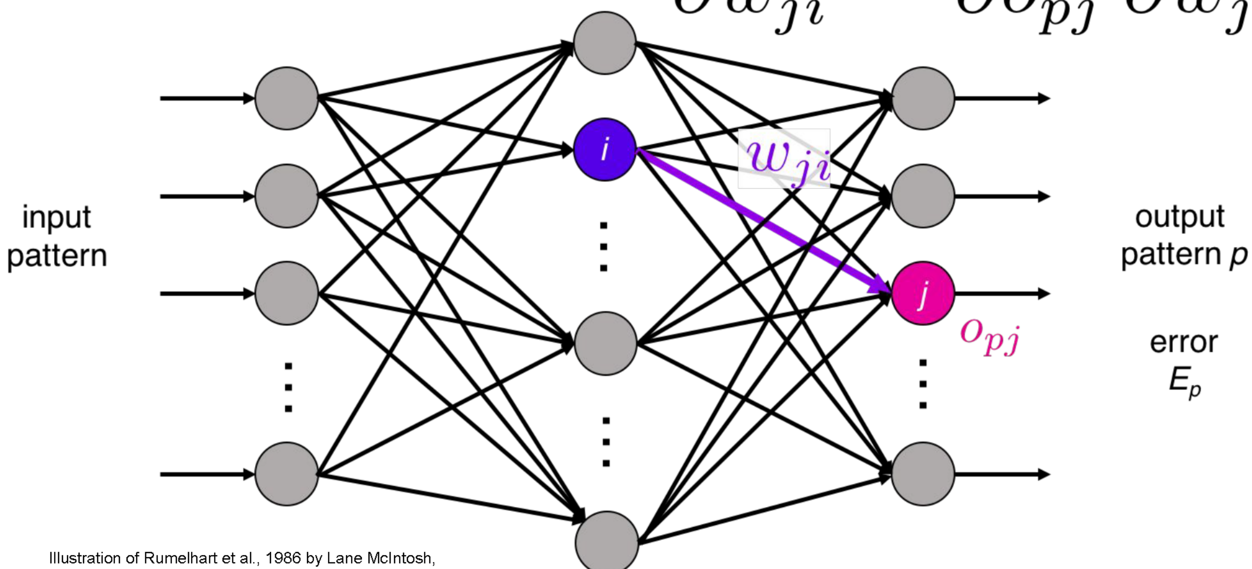
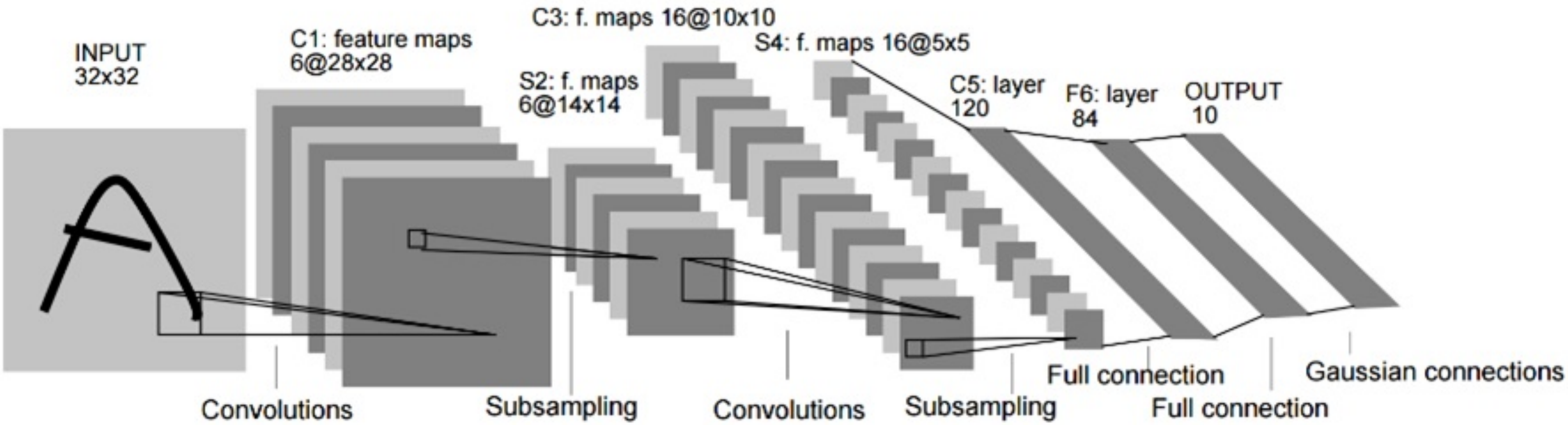


Illustration of Rumelhart et al., 1986 by Lane McIntosh, copyright CS231n 2017

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

LeNet [LeCun et al. 1998]



Gradient-based learning applied to document recognition [LeCun, Bottou, Bengio, Haffner 1998]

LeNet-1 from 1993

But neural networks were not that successful before

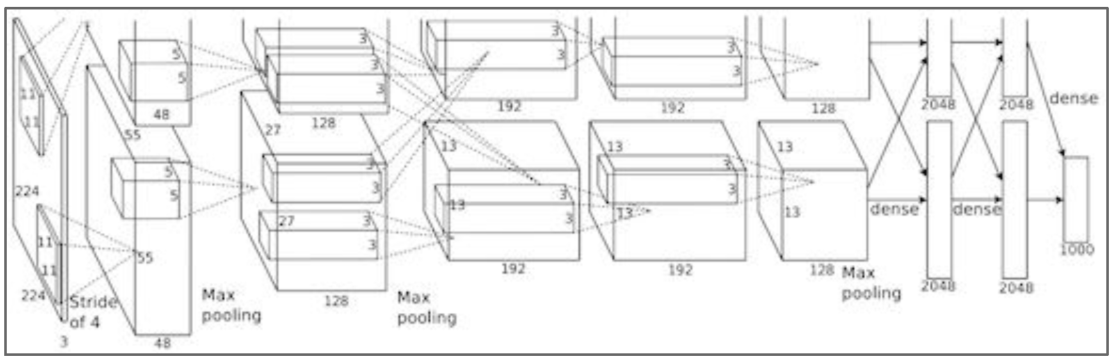
- Why?
  - Small training datasets lead to overfitting
  - Hard to train a deep neural network due to limited computational power

ImageNet dataset

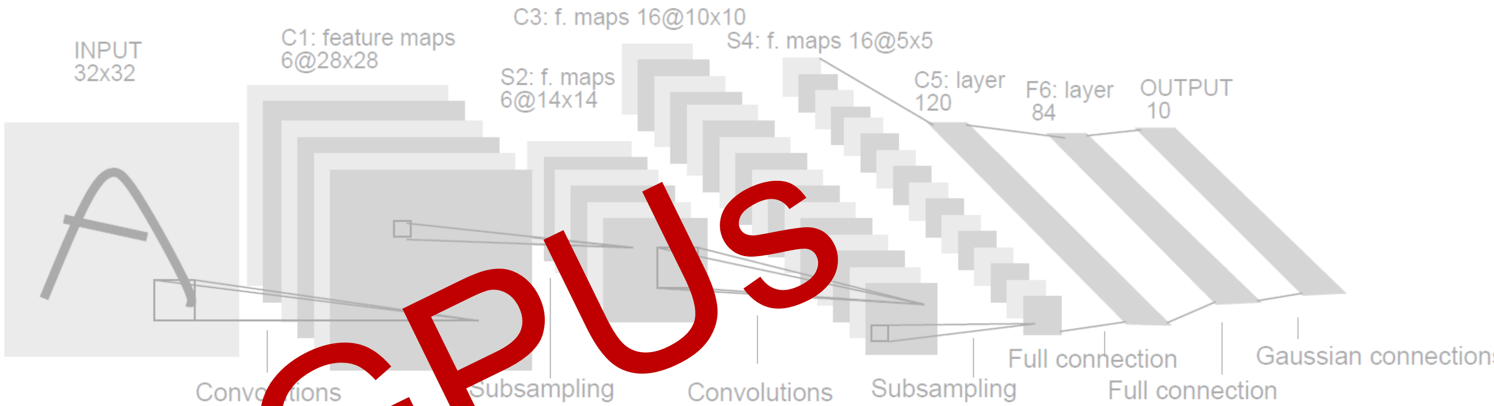
A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

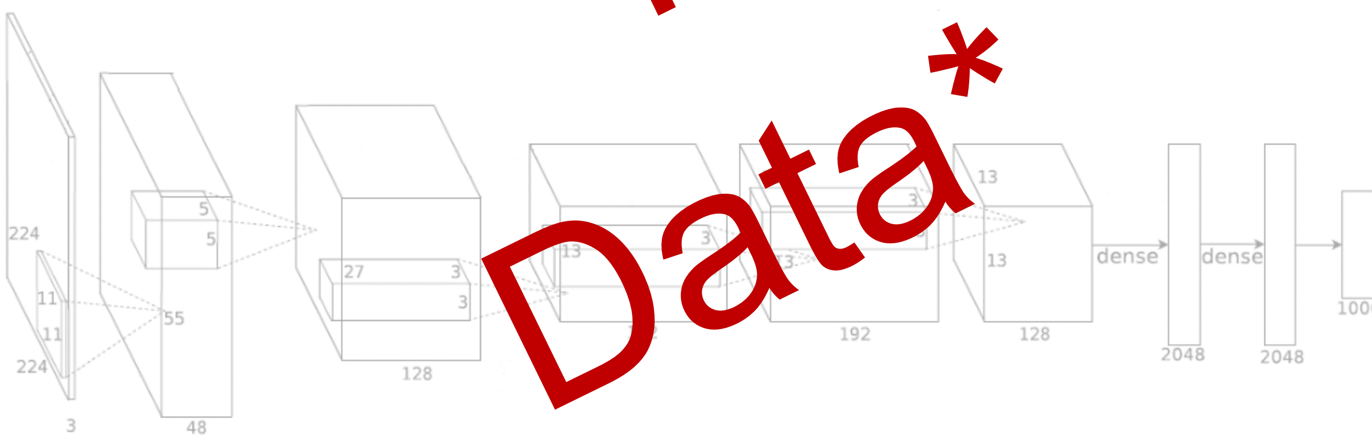


“AlexNet”



Gradient-Based Learning Applied to Document Recognition, LeCun, Bottou, Bengio and Haffner, Proc. of the IEEE, **1998**

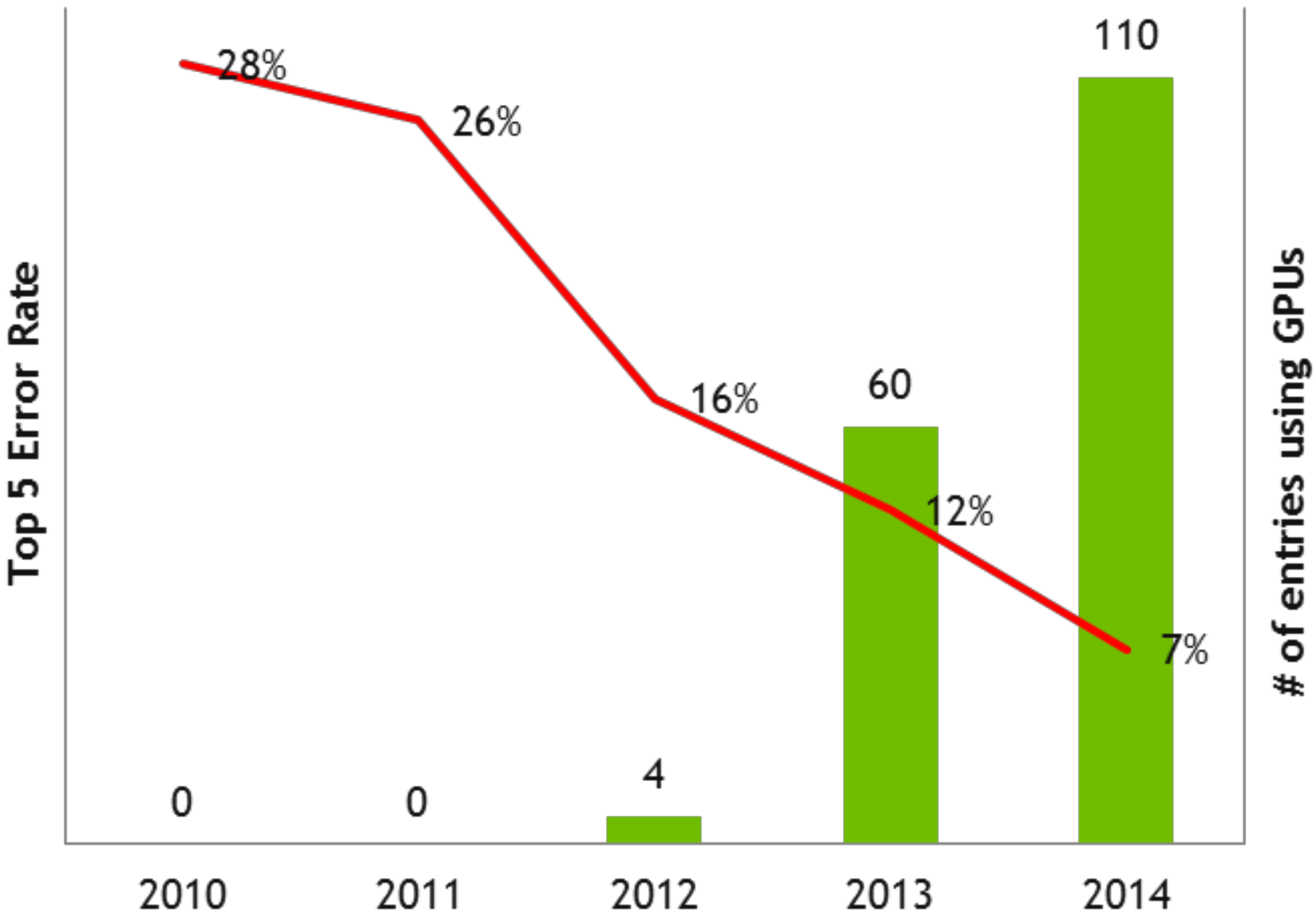
GPUs + Data*



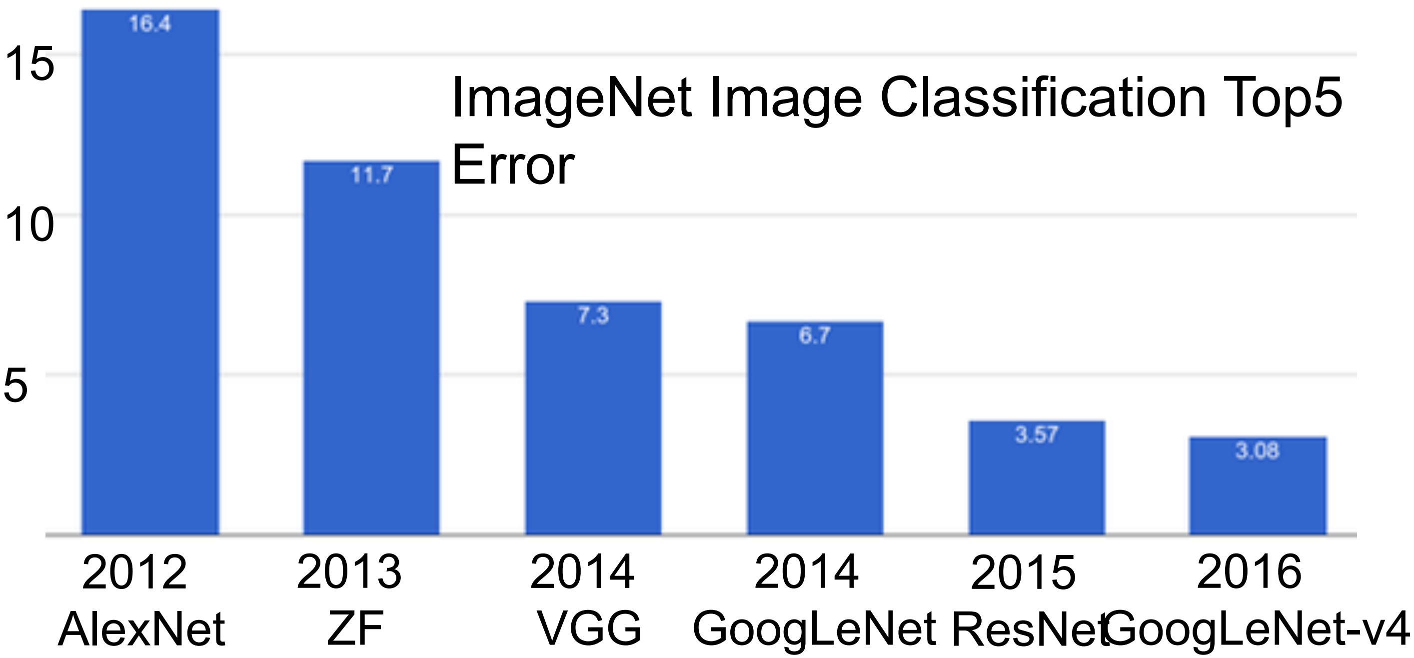
Imagenet Classification with Deep Convolutional Neural Networks, Krizhevsky, Sutskever, and Hinton, NIPS **2012**

Slide Credit: L. Zitnick

IMAGENET

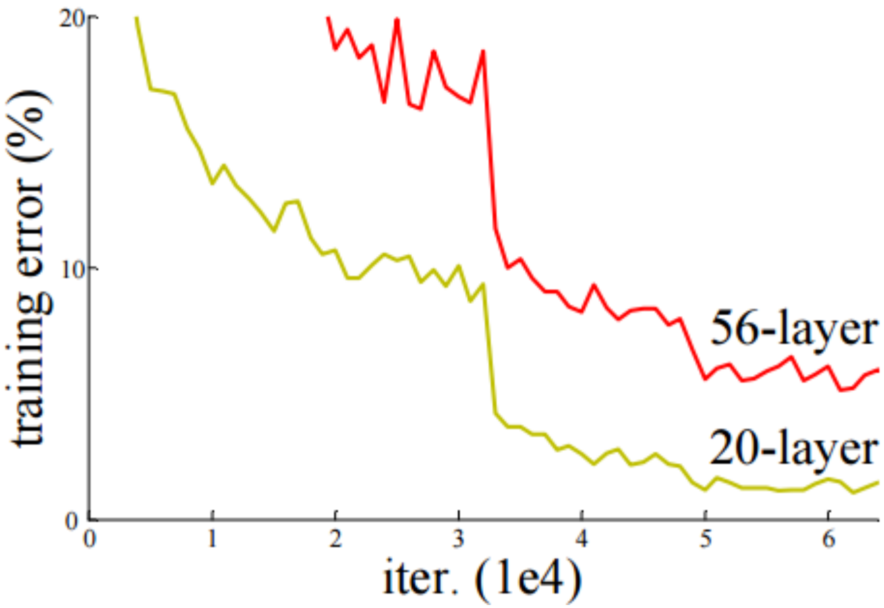

Top 5 Error Rate
# of entries using GPUs
28%
26%
16%
12%
7%
0
0
4
60
110
2010
2011
2012
2013
2014

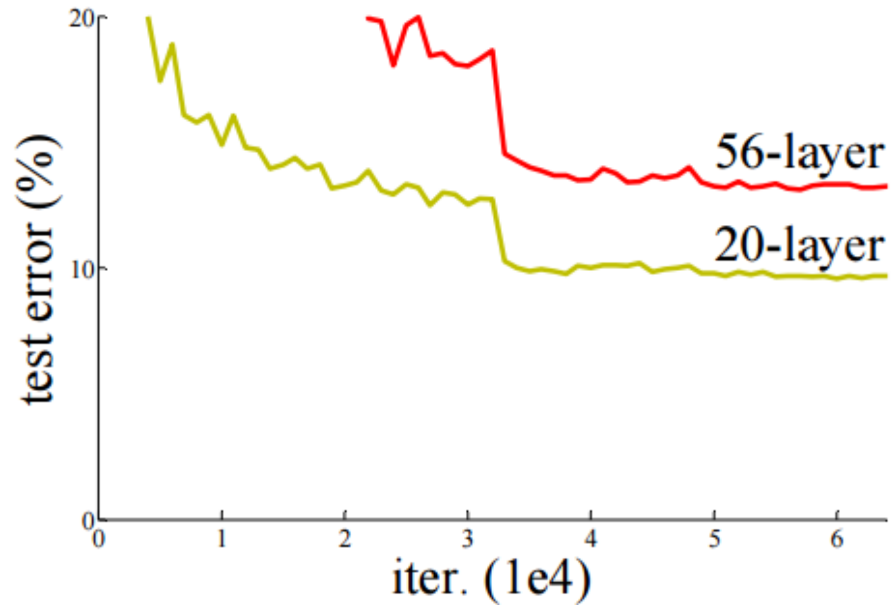
Progress on ImageNet



ResNet

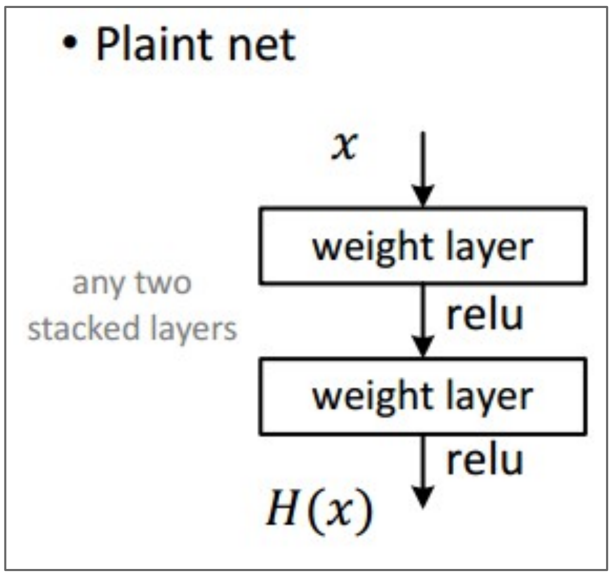
- Can we just increase the #layer?

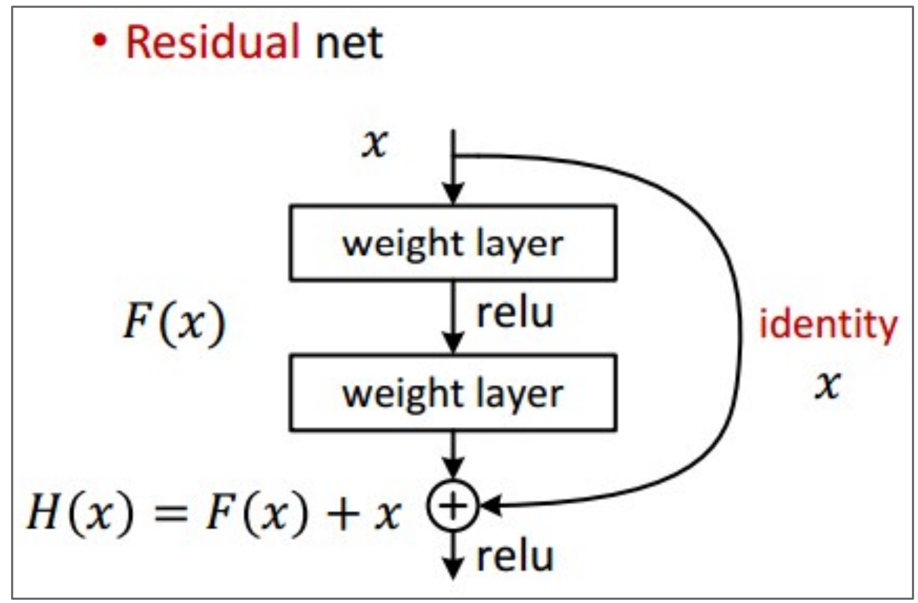




ResNet

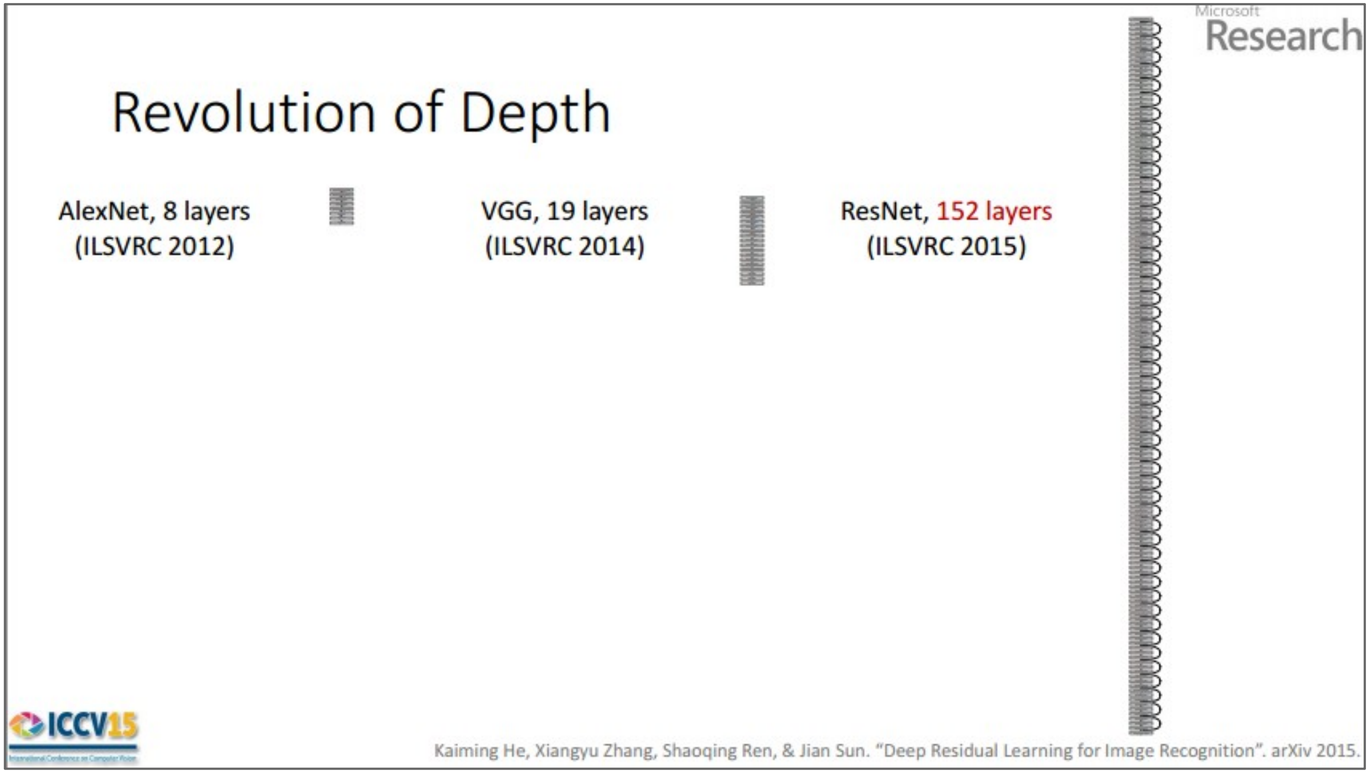
- How can we train very deep network?
  - Residual learning





ResNet

ILSVRC 2015 winner (3.6% top 5 error)

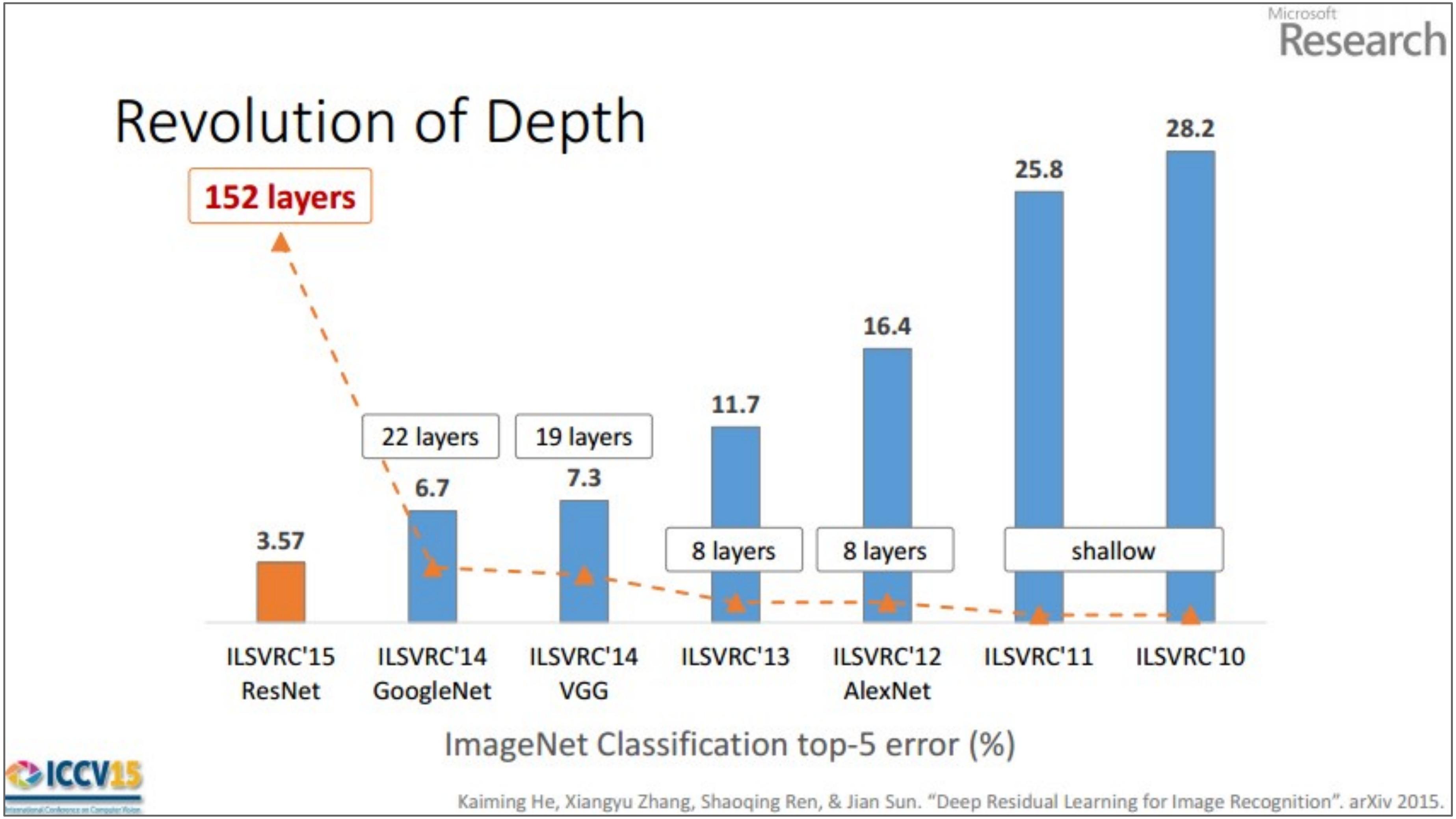


(slide from Kaiming He's recent presentation)

2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet! (even though it has 8x more layers)

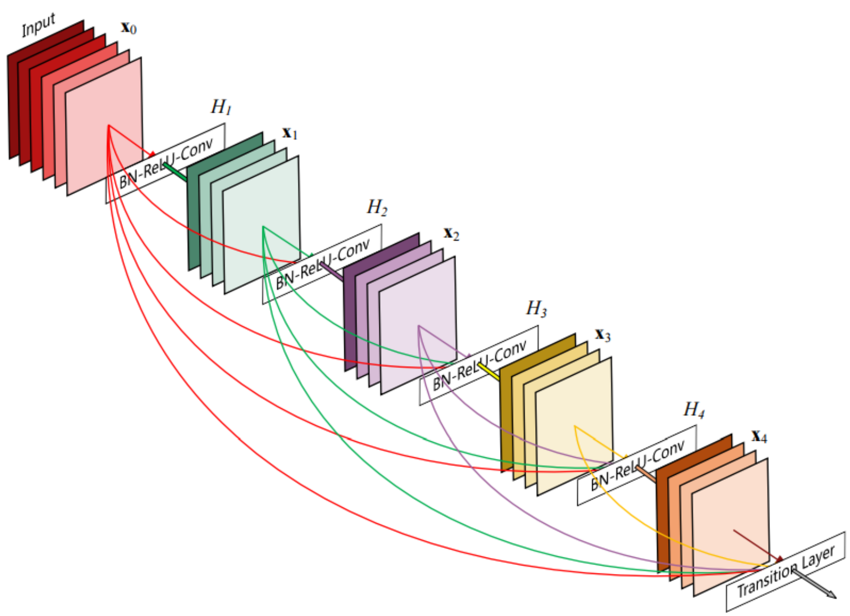
ResNet

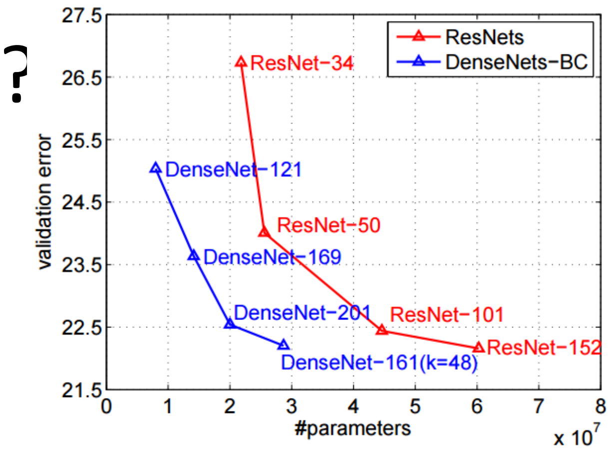


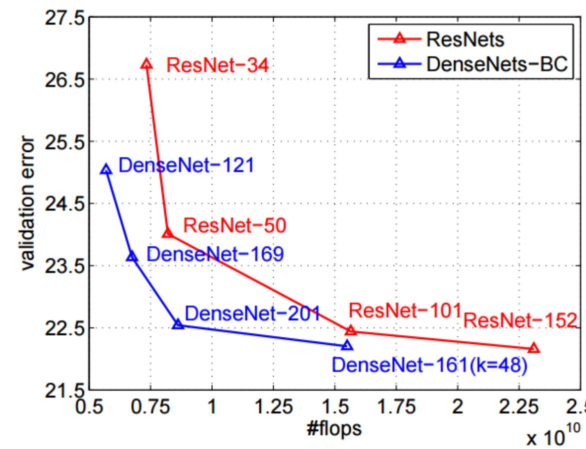
(slide from Kaiming He's recent presentation)

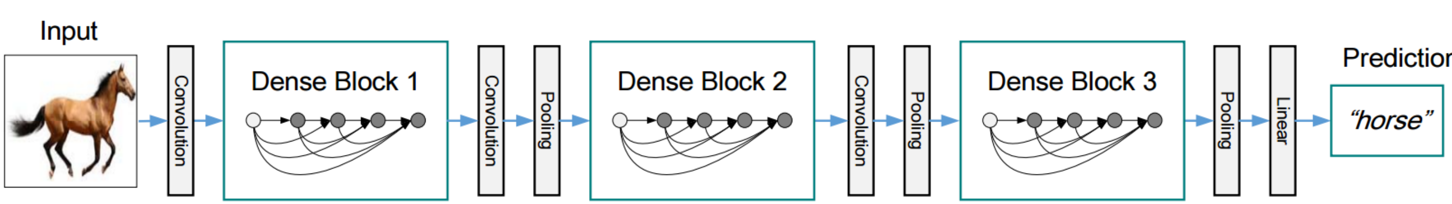
DenseNet

- Shorter connections (like ResNet) help
- Why not just connect them all?



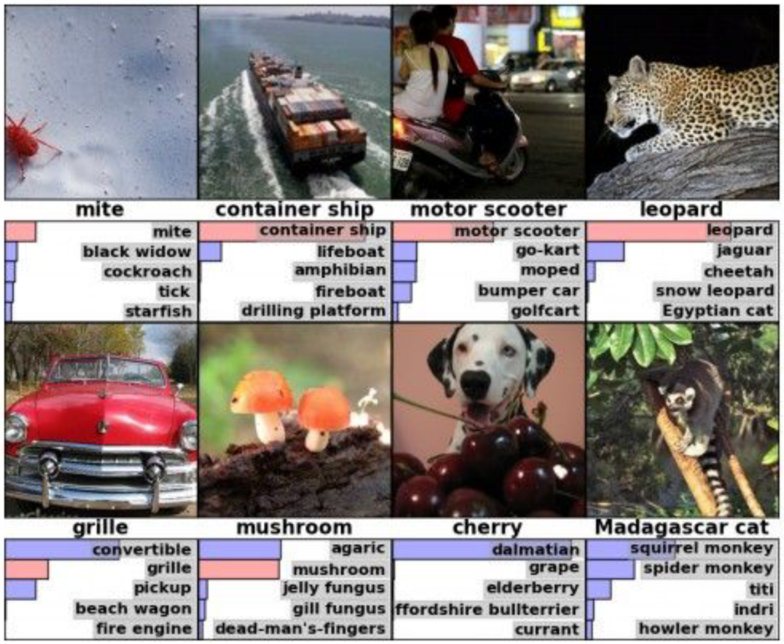






Fast-forward to today: ConvNets are everywhere

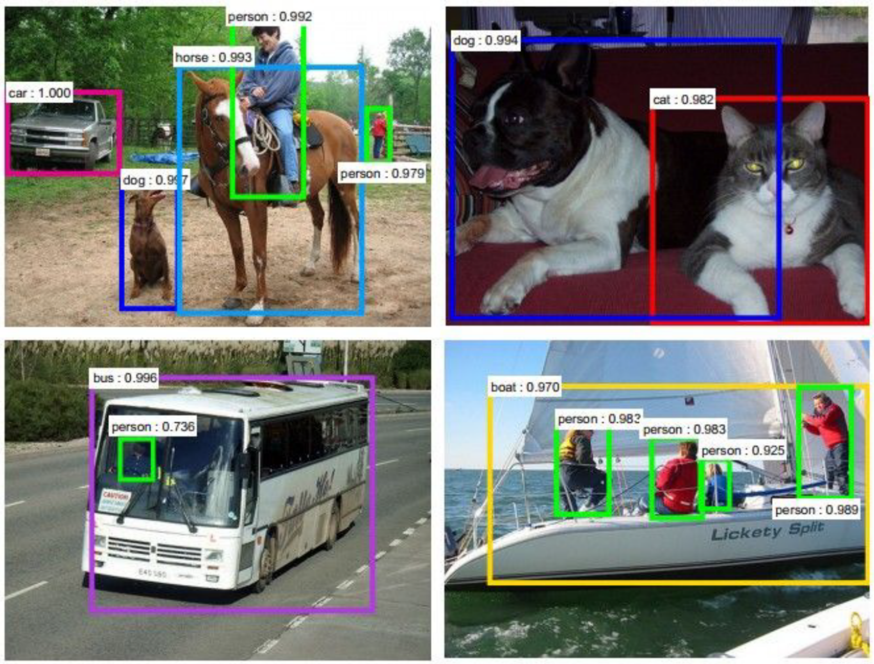
Classification



Retrieval

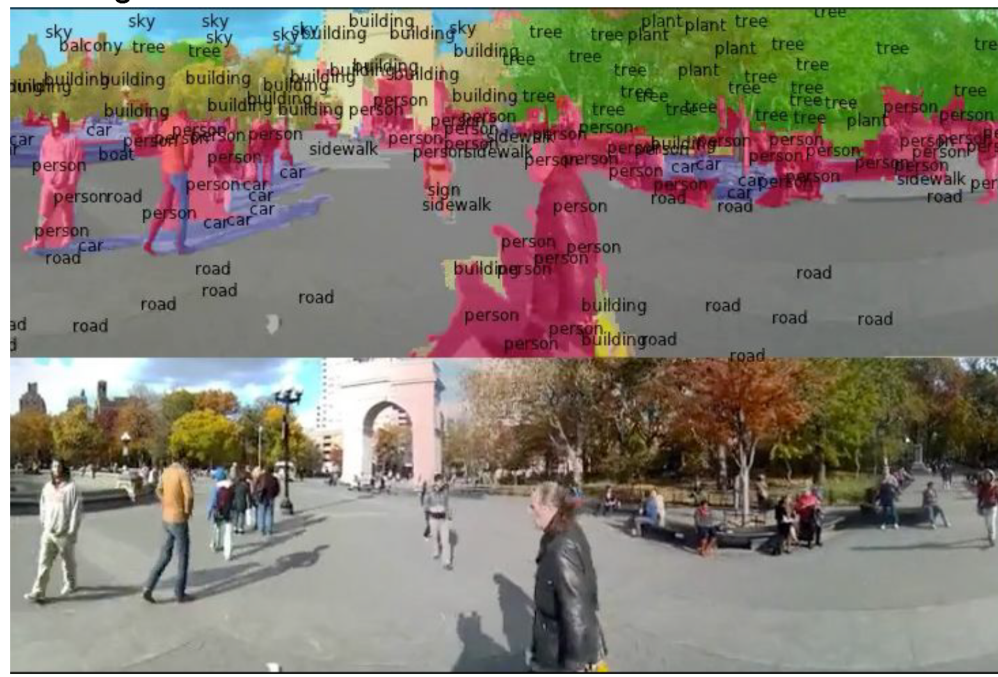
Fast-forward to today: ConvNets are everywhere

Detection



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

Segmentation



[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere

Photo by Lane McIntosh. Copyright CS231n 2017.

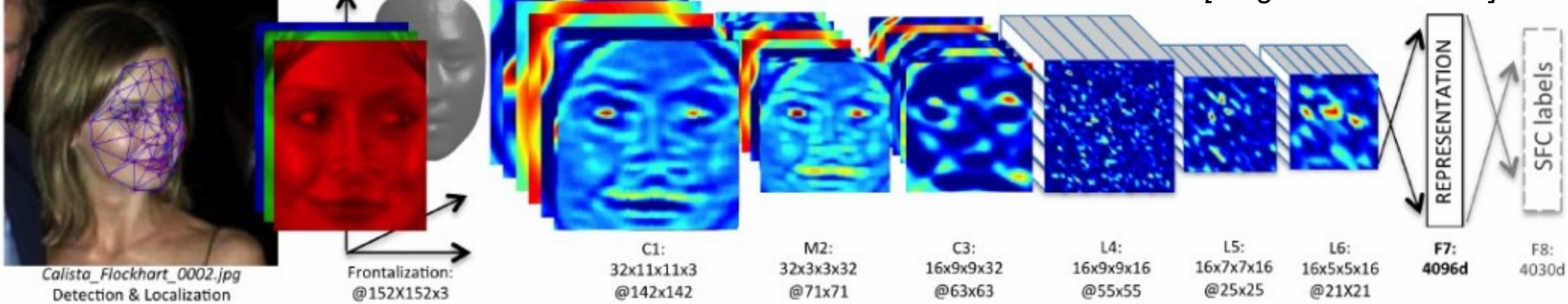
self-driving cars

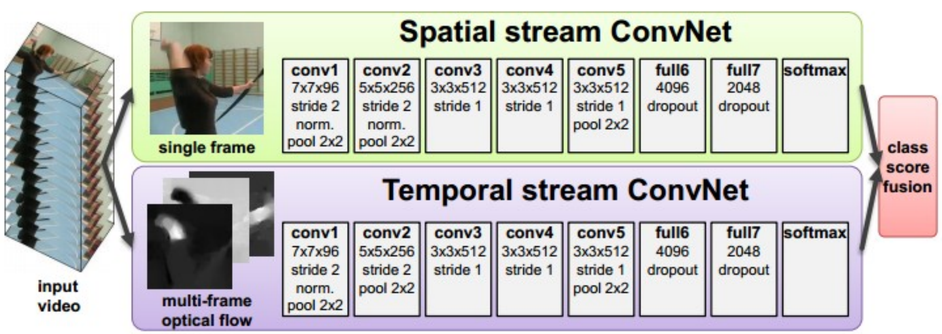
NVIDIA Tesla line
(these are the GPUs on rye01.stanford.edu)

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

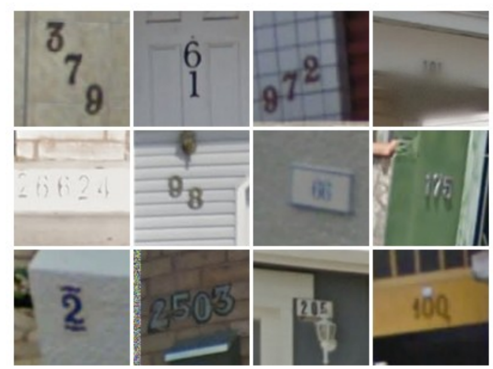
Fast-forward to today: ConvNets are everywhere

[Taigman et al. 2014]





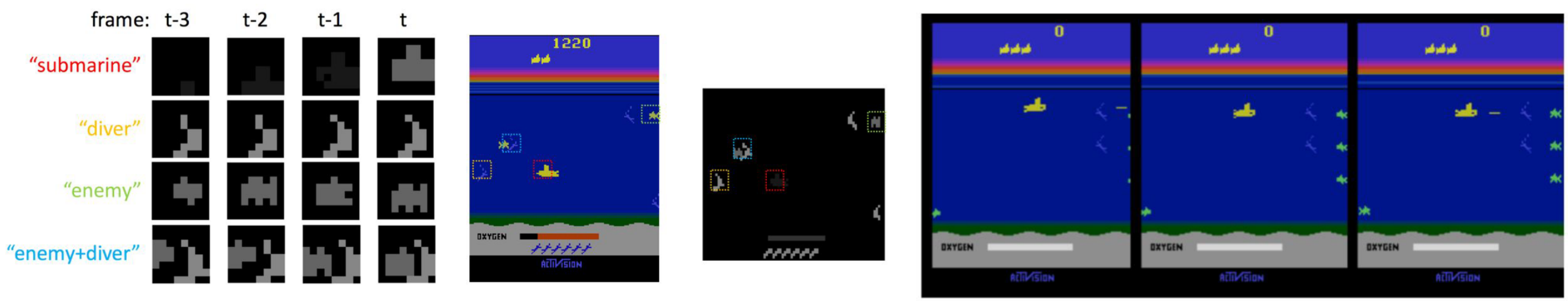
[Simonyan et al. 2014]



[Goodfellow 2014]

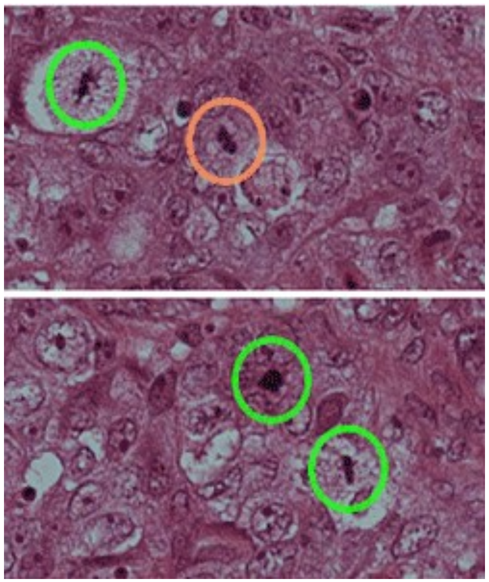
Fast-forward to today: ConvNets are everywhere

[Toshev, Szegedy 2014]



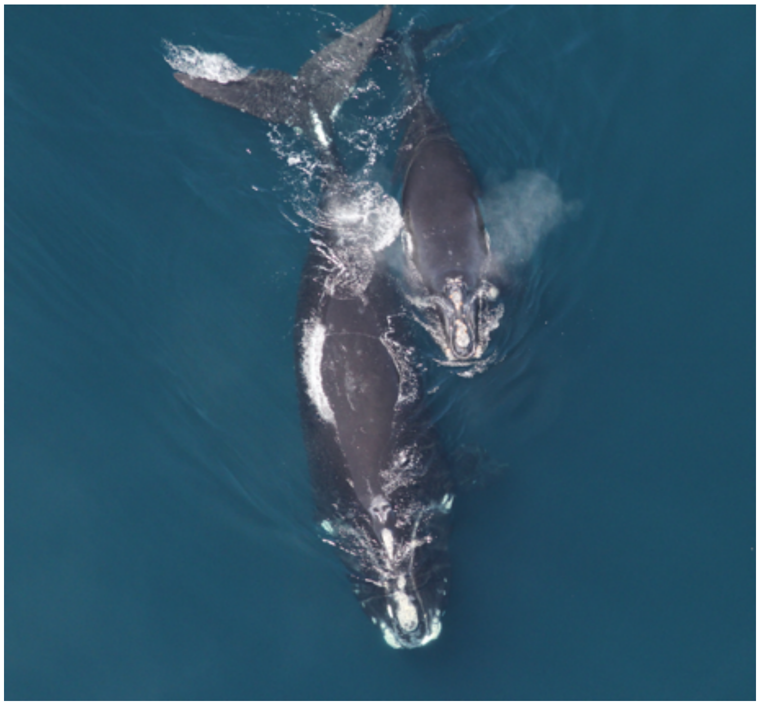
[Guo et al. 2014]

Fast-forward to today: ConvNets are everywhere



[Ciresan et al. 2013]

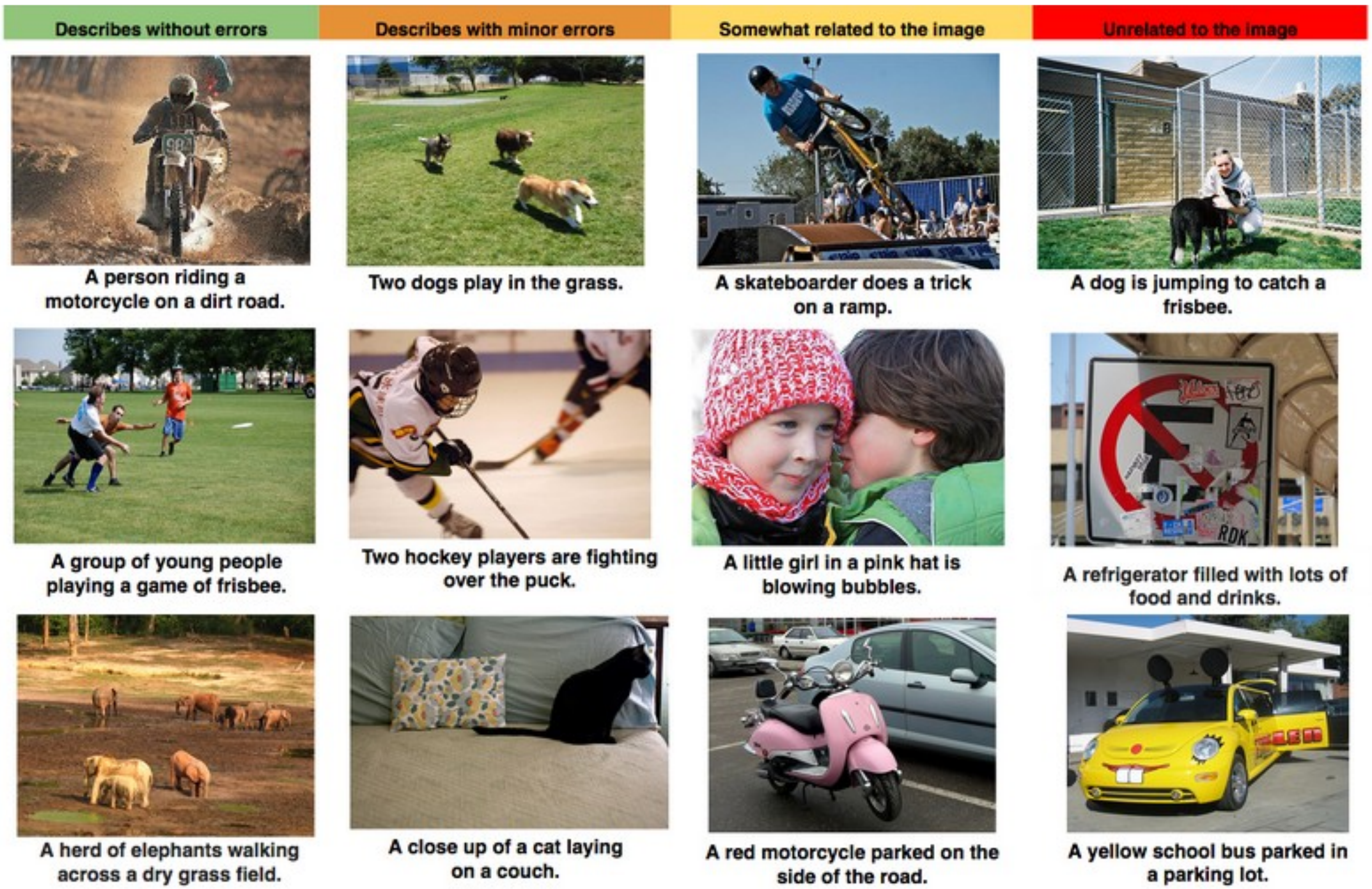
[Sermanet et al. 2011]
[Ciresan et al.]



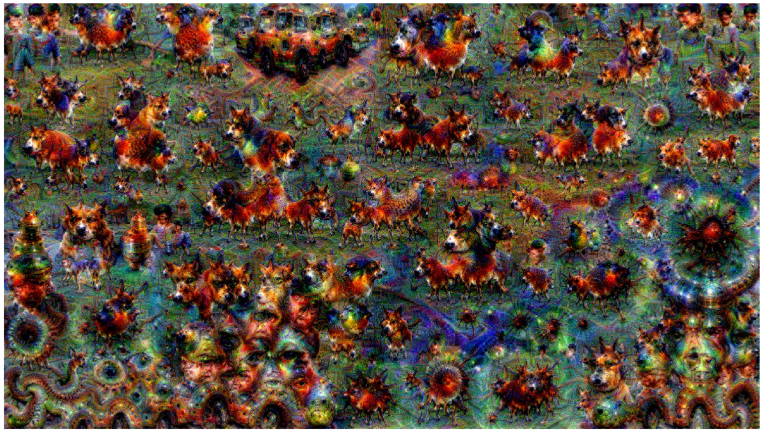
Whale recognition, Kaggle Challenge

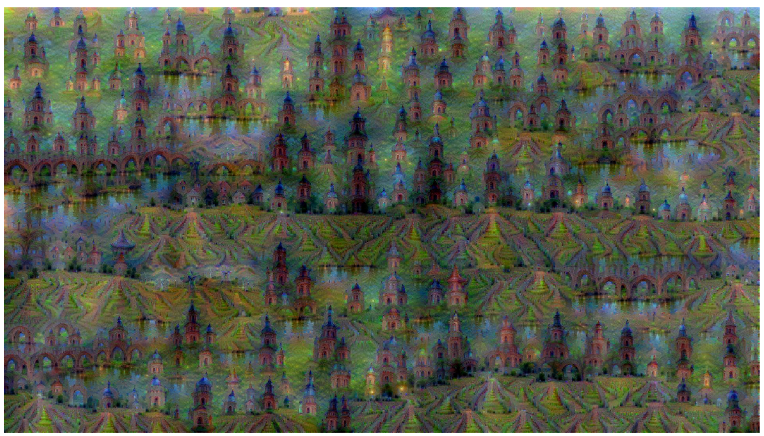
Mnih and Hinton, 2010

Image Captioning

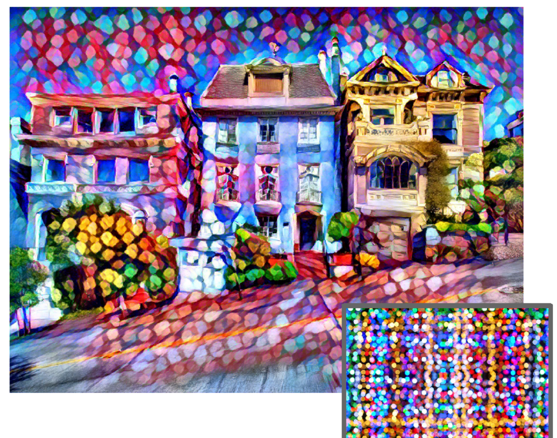


[Vinyals et al., 2015]





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

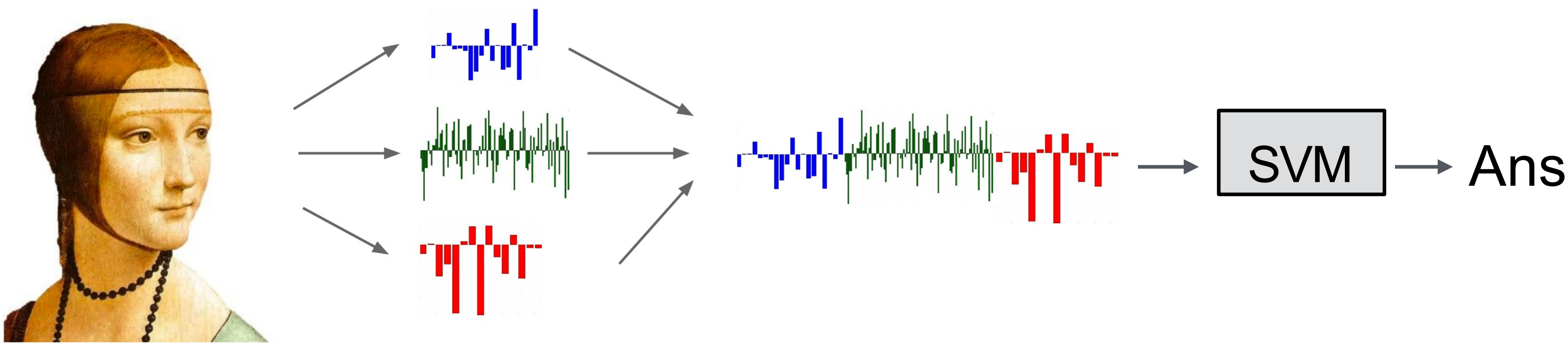


Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017; reproduced with permission

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

Summary

Recap: Life Before Deep Learning



Input Pixels

Extract Hand-Crafted Features

*Concatenate into a vector **x***

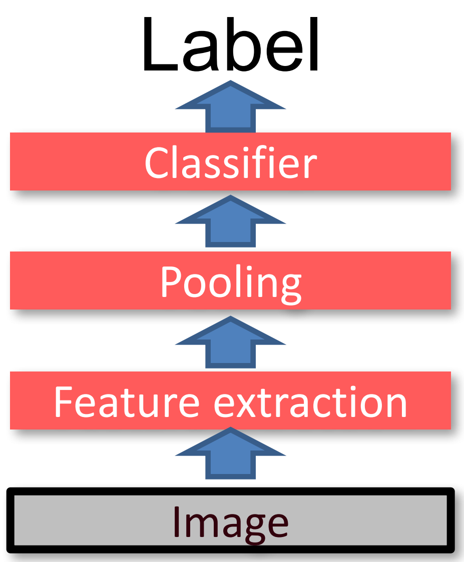
Linear Classifier

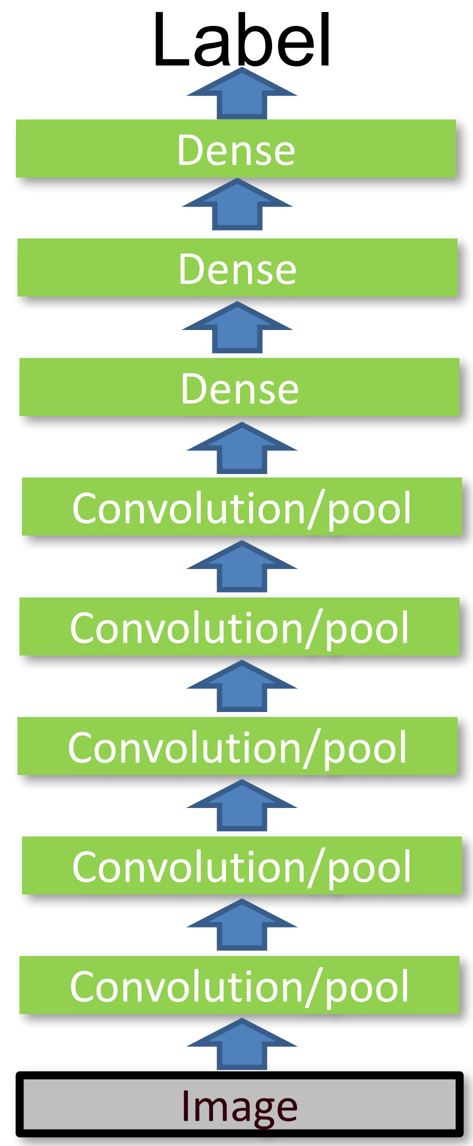
Figure: Karpathy 2016

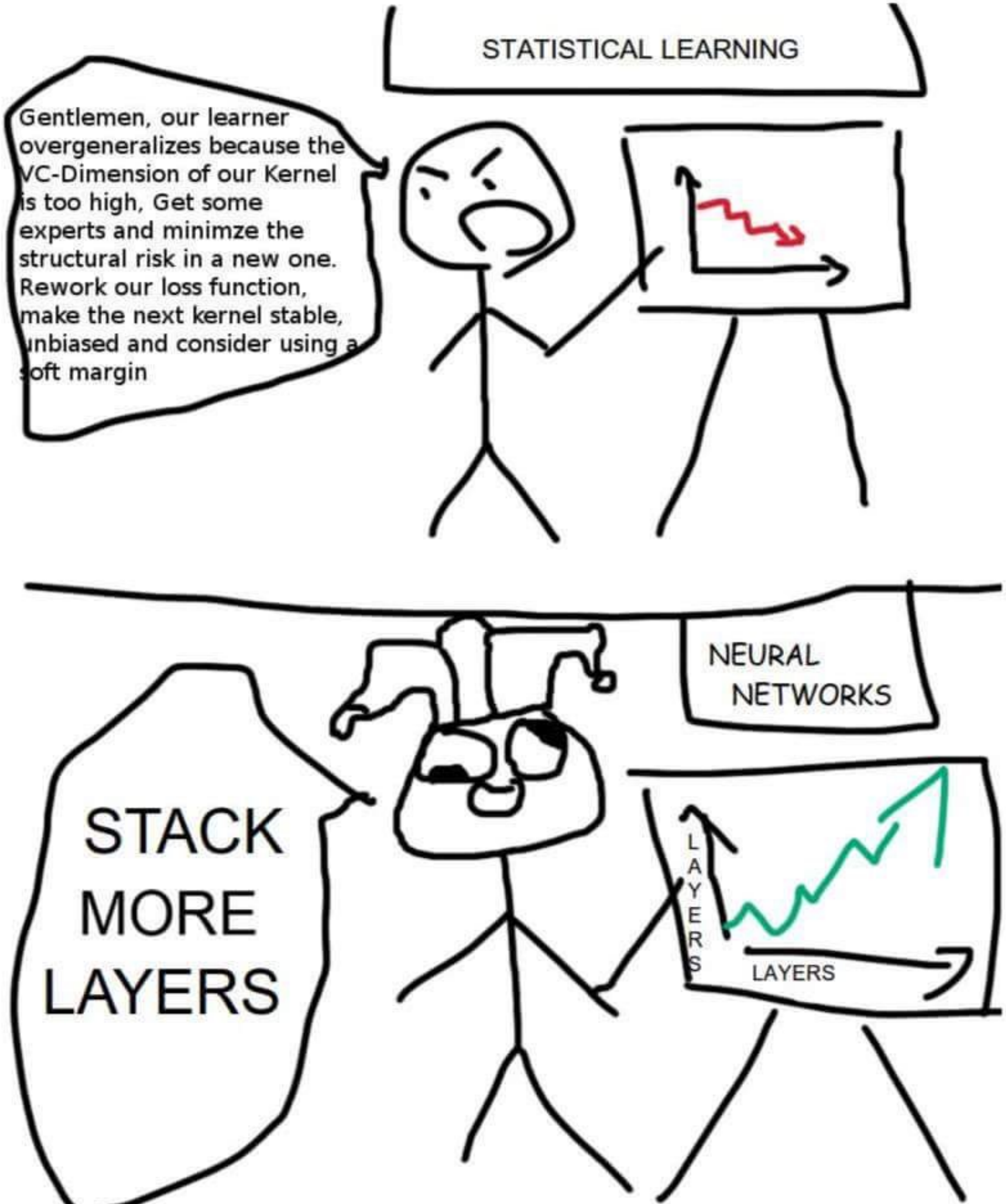
Why deep learning is powerful?

Convolutional filters are trained in a supervised manner by back-propagating classification error

Filters are learned from data instead of hand-crafted!






STATISTICAL LEARNING
Gentlemen, our learner overgeneralizes because the VC-Dimension of our Kernel is too high, Get some experts and minimze the structural risk in a new one. Rework our loss function, make the next kernel stable, unbiased and consider using a soft margin
NEURAL NETWORKS
STACK MORE LAYERS
LAYERS
LAYERS

Deep learning library

- TensorFlow
  - Research + Production
- PyTorch
  - Research
- Caffe2
  - Production

Resources

- http://deeplearning.net/
  - Hub to many other deep learning resources
- https://github.com/ChristosChristofidis/awesome-deep-learning
  - A resource collection deep learning
- https://github.com/kjw0612/awesome-deep-vision
  - A resource collection deep learning for computer vision
- http://cs231n.stanford.edu/syllabus.html
  - Nice course on CNN for visual recognition

Things to remember

- Supervised learning
  - Linear classifier, softmax, cross-entropy loss
- Neural network
  - Linear functions chained together and separated with non-linear activation functions
- Convolutional neural network (CNN)
  - Neural network with local connectivity and weight sharing
  - Convolution, nonlinearity, max pooling
- Training CNN
  - Back propagation, data split, dropout, data augmentation

2019 Turing Awards

Yann LeCun Geoffrey Hinton Yoshua Bengio