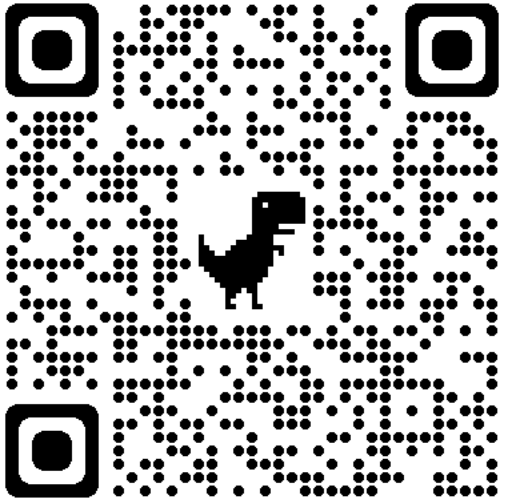
Lecture 18:
Introduction to Recognition

COMP 590/776: Computer Vision

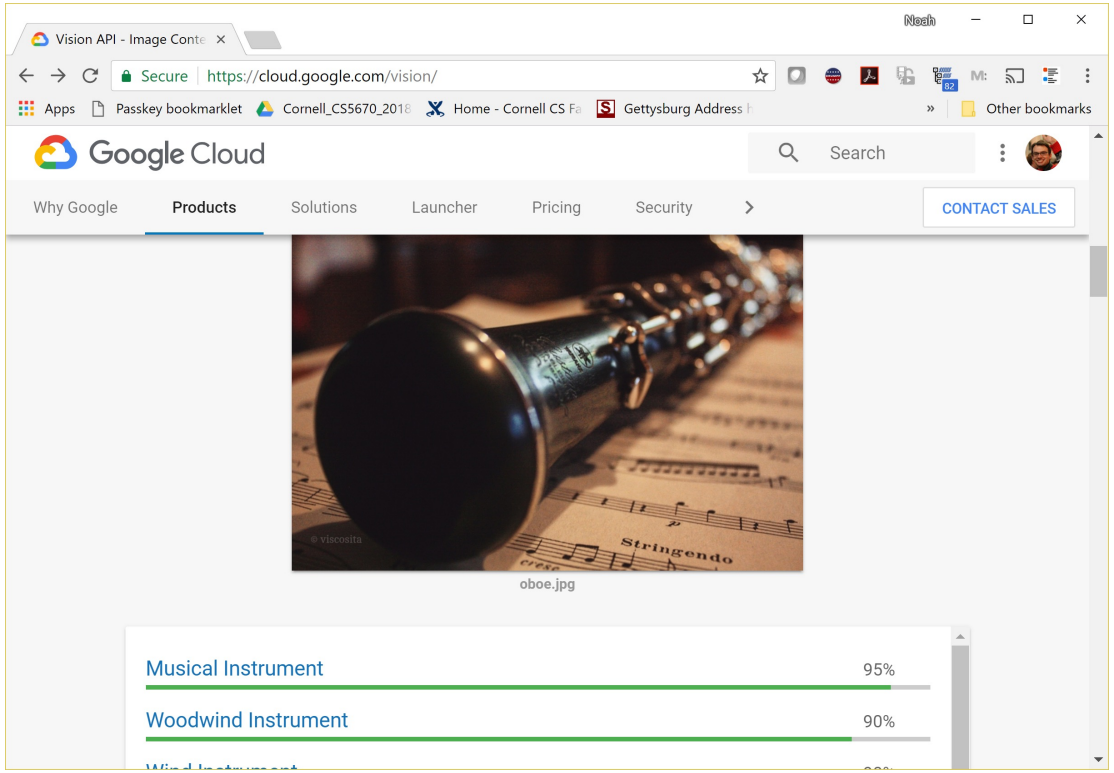
Instructor: Soumyadip (Roni) Sengupta

TA: Mykhailo (Misha) Shvets



Course Website:
Scan Me!

Image classification demo



https://cloud.google.com/vision/docs/drag-and-drop

See also:
https://aws.amazon.com/rekognition/
https://www.clarifai.com/
https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/
…

What is “Recognition”?

Next few slides adapted from Li, Fergus, & Torralba’s excellent short course on category and object recognition

What is “Recognition”?

- Verification: is that a lamp?

What is “Recognition”?

- Verification: is that a lamp?
- Detection: where are the people?

What is “Recognition”?

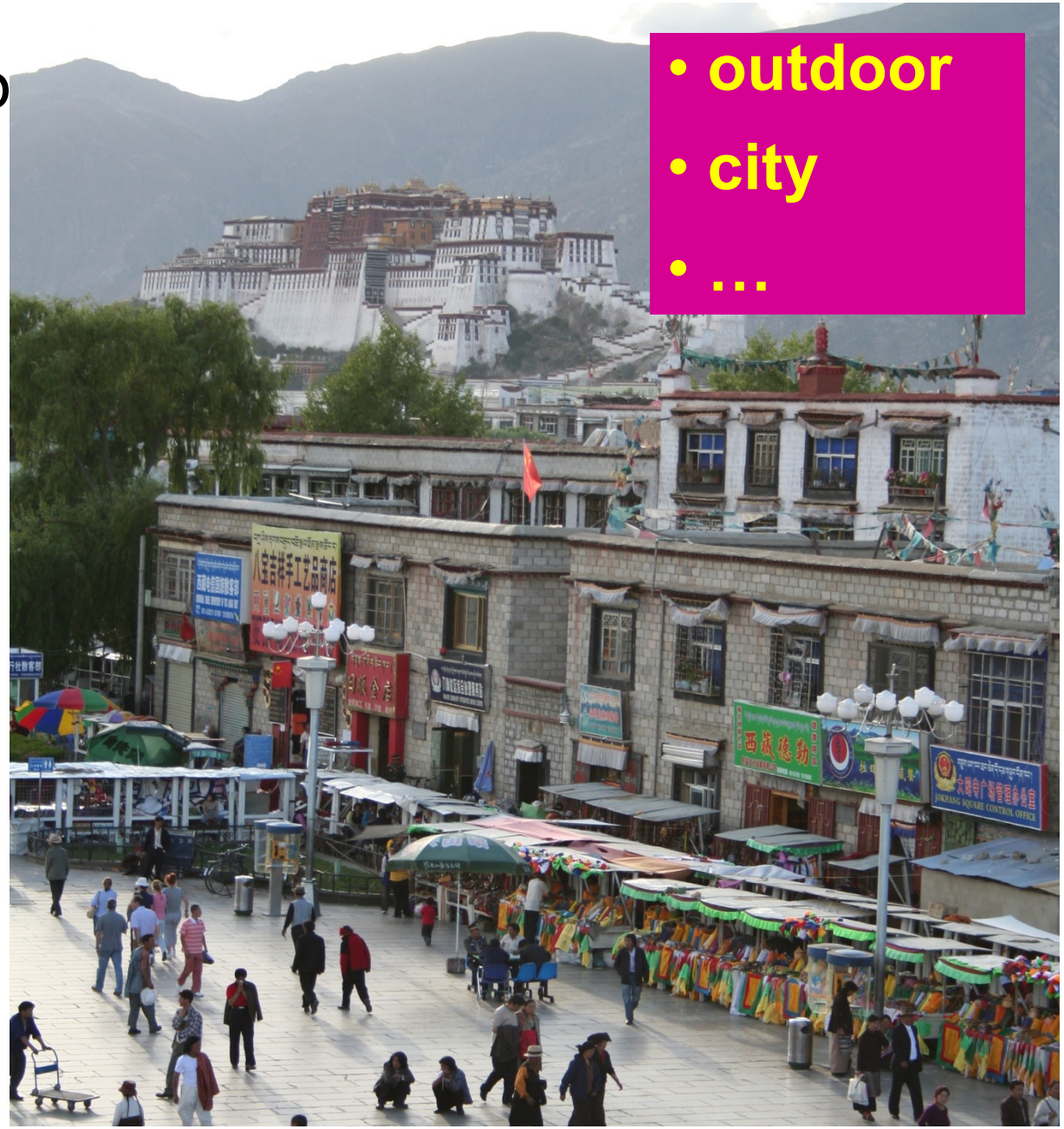
- Verification: is that a lamp?
- Detection: where are the people?
- Identification: is that Potala Palace?

What is “Recognition”?

- Verification: is that a lamp?
- Detection: where are the people?
- Identification: is that Potala Palace?
- Object categorization

What is “Recognition”?

- Verification: is that a lamp?
- Detection: where are the people?
- Identification: is that Potala Palace?
- Object categorization
- Scene and context categorization



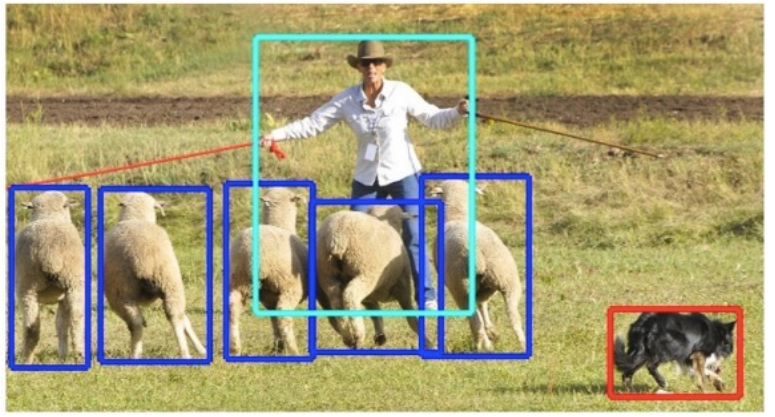
What is “Recognition”?

- Verification: is that a lamp?
- Detection: where are the people?
- Identification: is that Potala Palace?
- Object categorization
- Scene and context categorization
- Activity / Event Recognition

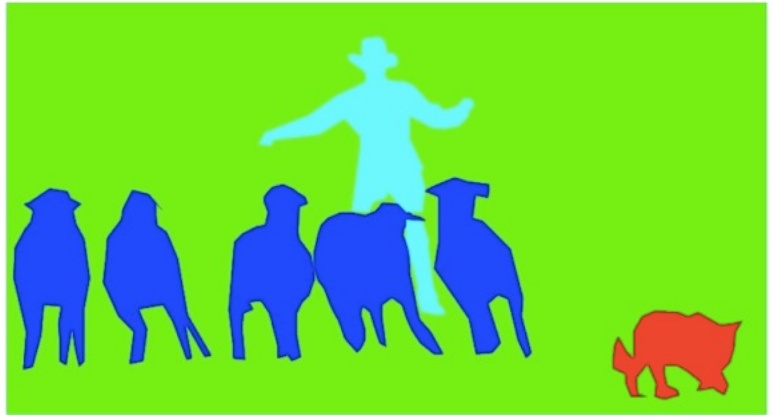
Recognition: What type of output?

Image classification

Object detection



Semantic segmentation



Instance segmentation

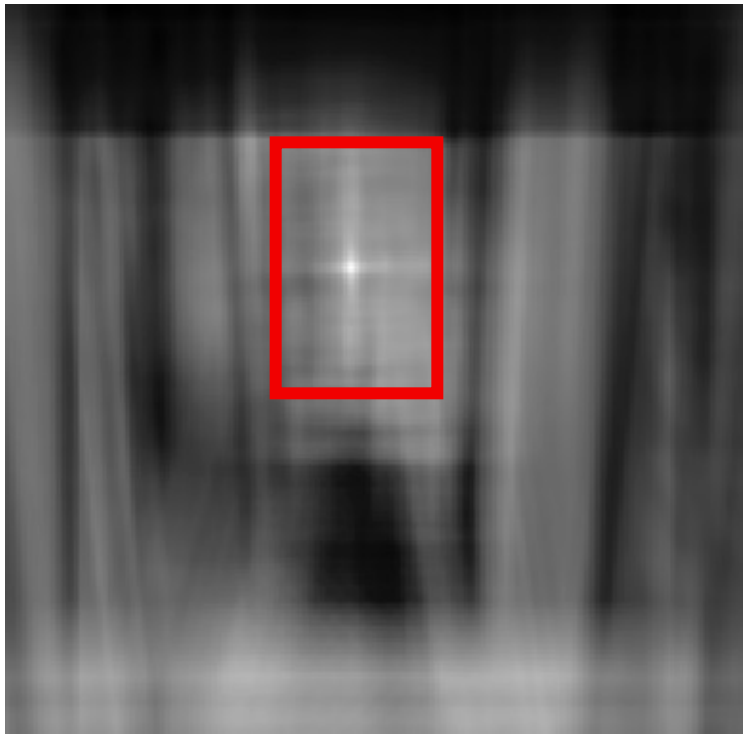
- And beyond: depth/3D structure prediction, image description, etc.

Object recognition: Is it really so hard?

This is a chair

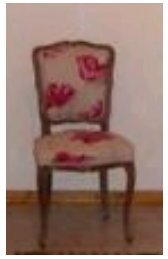
Find the chair in this image

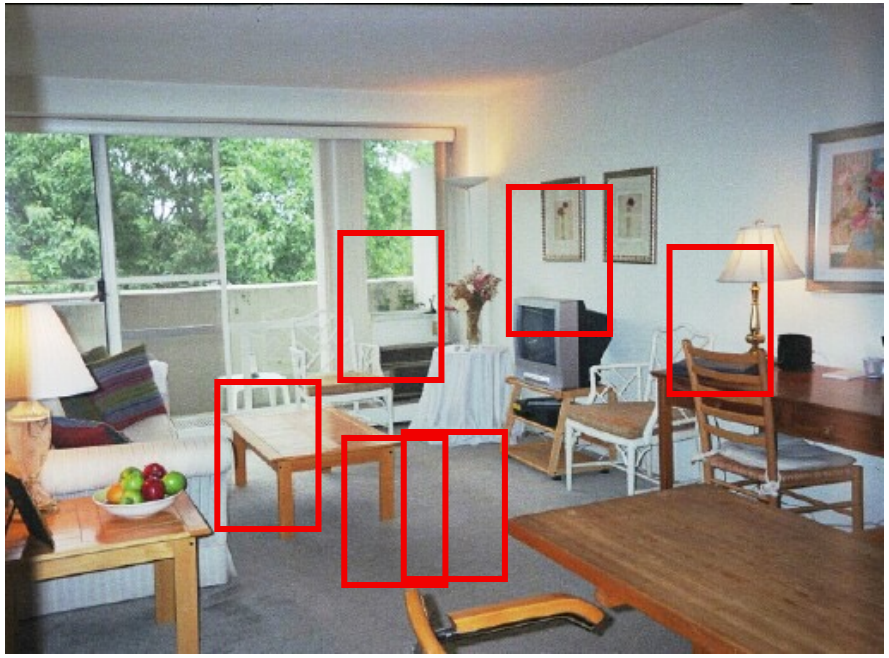
Output of normalized correlation

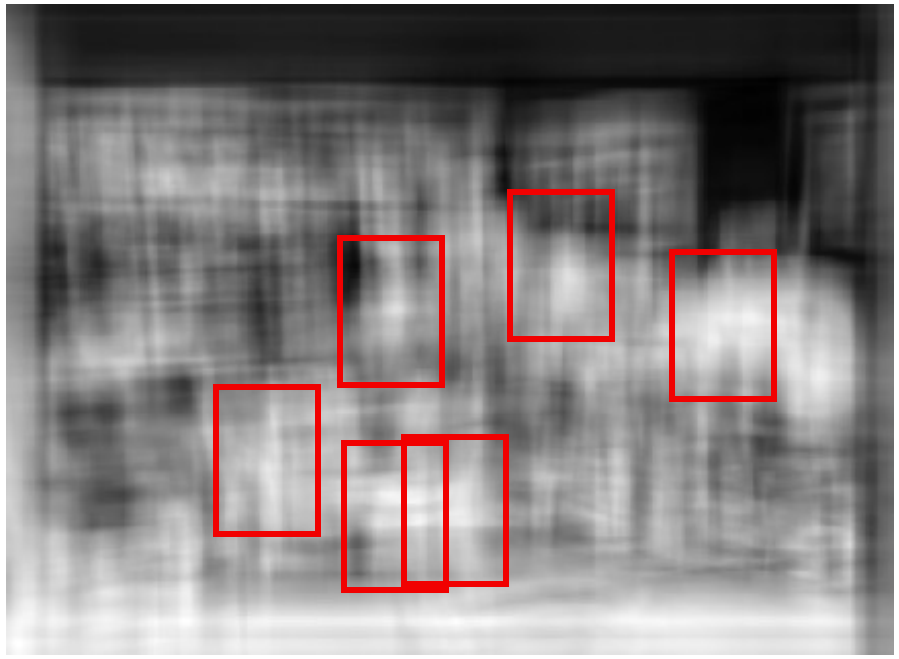


Object recognition: Is it really so hard?

Find the chair in this image



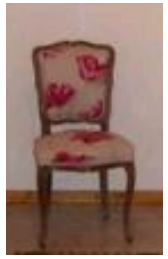




Pretty much garbage:
Simple template matching is not going to do the trick

Object recognition: Is it really so hard?

Find the chair in this image



A “popular method is that of template matching, by point to point correlation of a model pattern with the image pattern. These techniques are inadequate for three-dimensional scene analysis for many reasons, such as occlusion, changes in viewing angle, and articulation of parts.” Nivatia & Binford, 1977.

Why not use SIFT matching for everything?

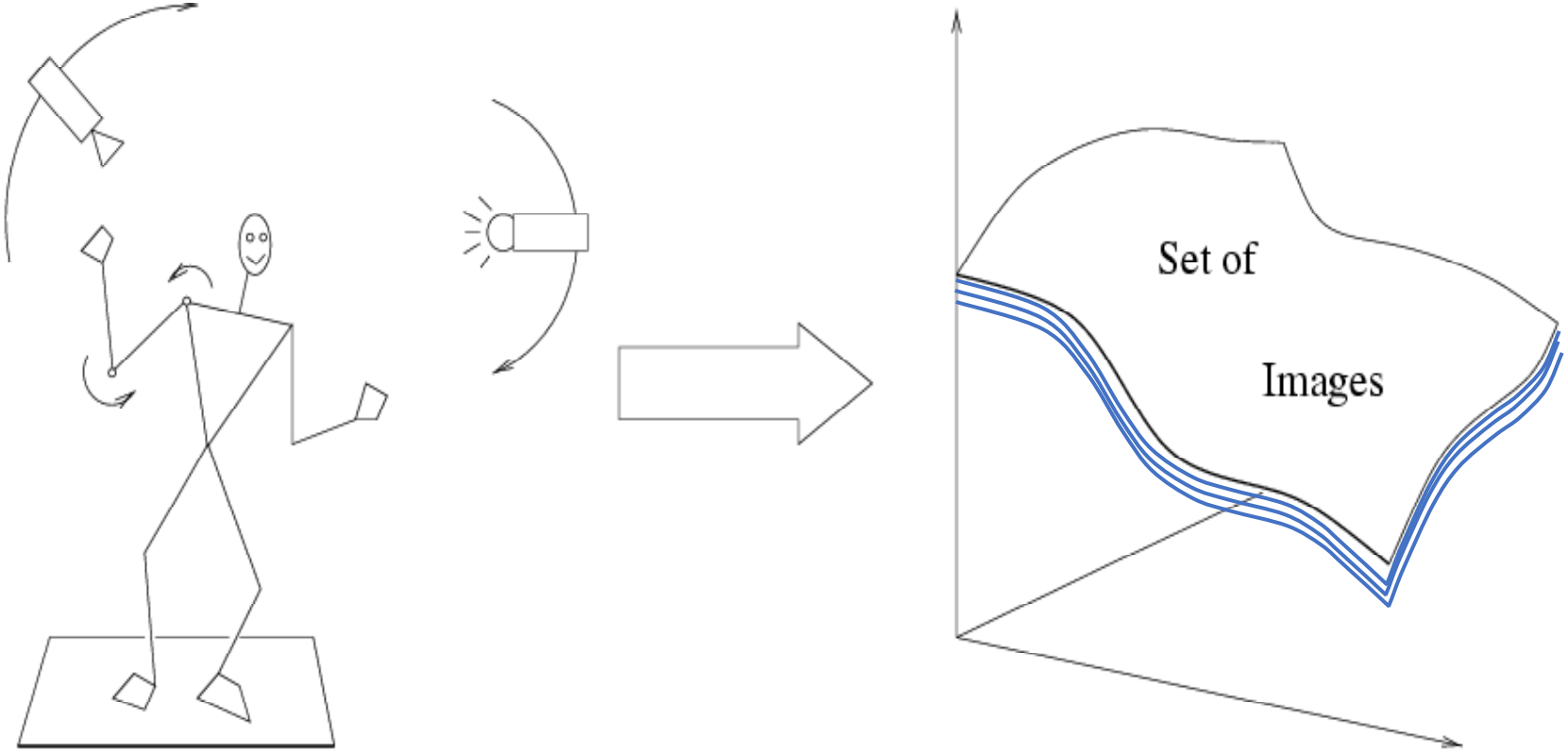
- Works well for object *instances* (or distinctive images such as logos)

- Not great for generic object *categories*

And it can get a lot harder

Brady, M. J., & Kersten, D. (2003). Bootstrapped learning of novel objects. J Vis, 3(6), 413-422

Why is recognition hard?



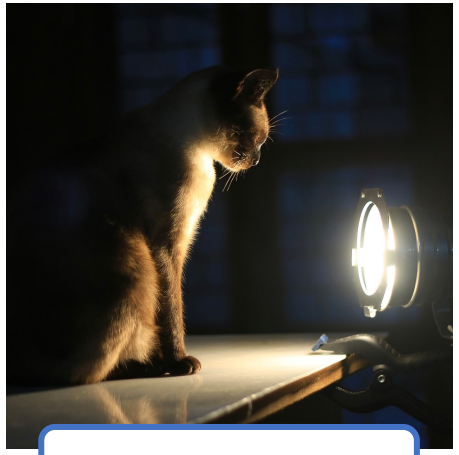
Variability: Camera position,
Illumination,
Shape,
etc…

Challenge: lots of potential classes

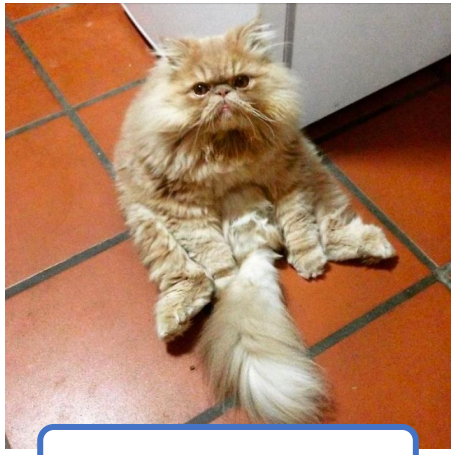
Variation Makes Recognition Hard

- The same class of object can appear *very* differently in different images

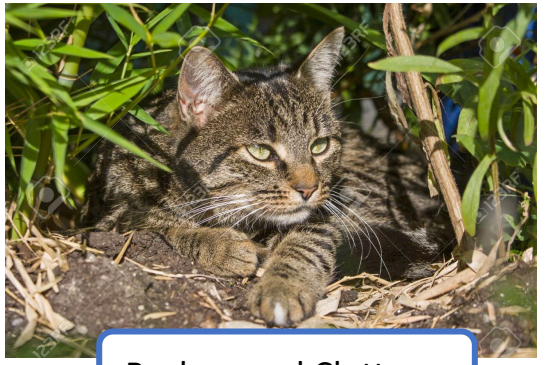
Viewpoint Variation



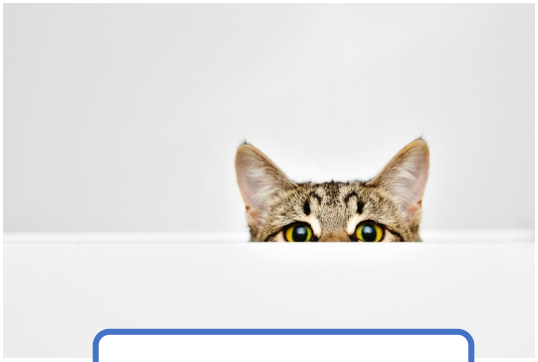
Lighting Variation



Deformation



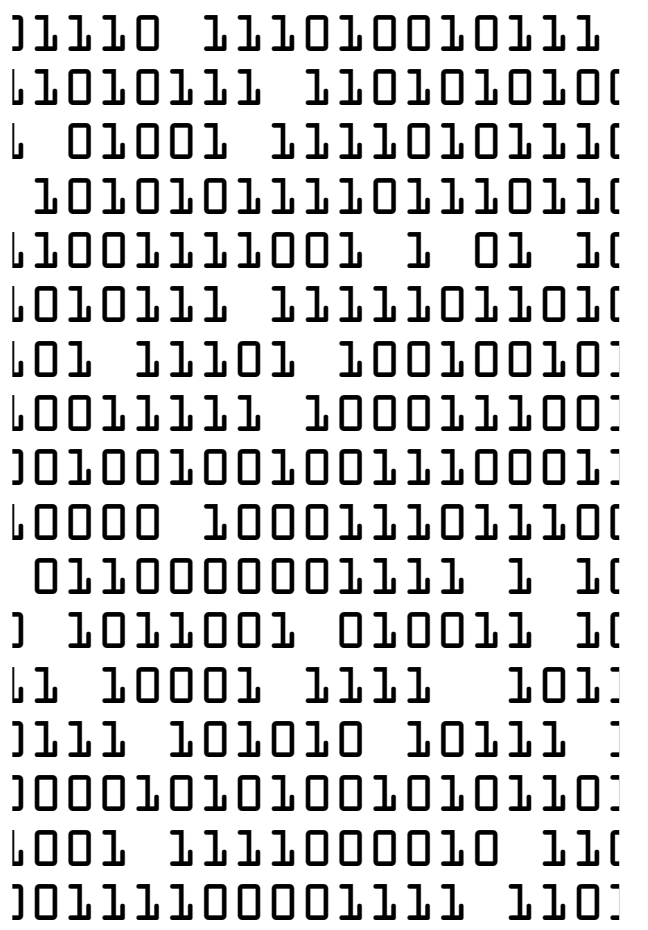
Background Clutter



Occlusion

The Semantic Gap

What we see



What the computer sees

Image Classifiers in a Nutshell

- Input: an image
- Output: the class label for that image

```
def classifier(image):
    //Do some stuff
    return class_label;
```

- Label is generally one or more of the discrete labels used in training
  - e.g. {cat, dog, cow, toaster, apple, tomato, truck, … }

$f($  $) =$ “Cat”

$f($  $) =$ “Dog”

$f($  $) =$ “Toaster”

The Problem is Under-constrained

- Distinct realities can produce the same image…
- We generally can't compute the "right" answer, but we can compute the most likely one…
- We need some kind of prior to condition on. We can learn this prior from data:

$$f(x) = \underset{\ell_x}{argmax}\ P(\ell_x | data)$$

What Matters in Recognition?

- Data
  - More is always better (as long as it is good data)
  - Annotation is the hard part

- Representation
  - Low level: SIFT, HoG, GIST, edges
  - Mid level: Bag of words, sliding window, deformable model
  - High level: Contextual dependence
  - Deep learned features

- Learning Techniques
  - E.g. choice of classifier or inference method

What Matters in Recognition?

- Data
  - More is always better (as long as it is good data)
  - Annotation is the hard part

- Representation
  - Low level: SIFT, HoG, GIST, edges
  - Mid level: Bag of words, sliding window, deformable model
  - High level: Contextual dependence
  - Deep learned features

- Learning Techniques
  - E.g. choice of classifier or inference method

24 Hrs in Photos

Flickr Photos From 1 Day in 2011

https://www.kesselskramer.com/project/24-hrs-in-photos/

Data Sets

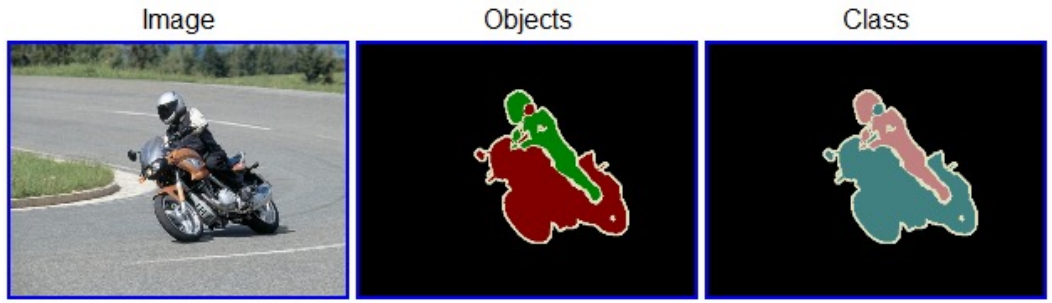
- PASCAL VOC
  - *Not* Crowdsourced, bounding boxes, 20 categories
- **ImageNet**
  - **Huge, Crowdsourced, Hierarchical, *Iconic* objects**
- SUN Scene Database, Places
  - *Not* Crowdsourced, 397 (or 720) scene categories
- LabelMe (Overlaps with SUN)
  - Sort of Crowdsourced, Segmentations, Open ended
- SUN *Attribute* database (Overlaps with SUN)
  - Crowdsourced, 102 attributes for every scene
- OpenSurfaces
  - Crowdsourced, materials
- **Microsoft COCO**
  - **Crowdsourced, large-scale objects**

… and many more https://paperswithcode.com/datasets?task=image-classification

The PASCAL Visual Object Classes Challenge 2009 (VOC2009)

- 20 object categories (aeroplane to TV/monitor)

- Three challenges:
  - Classification challenge (is there an X in this image?)
  - Detection challenge (draw a box around every X)
  - Segmentation challenge (which class is each pixel?)

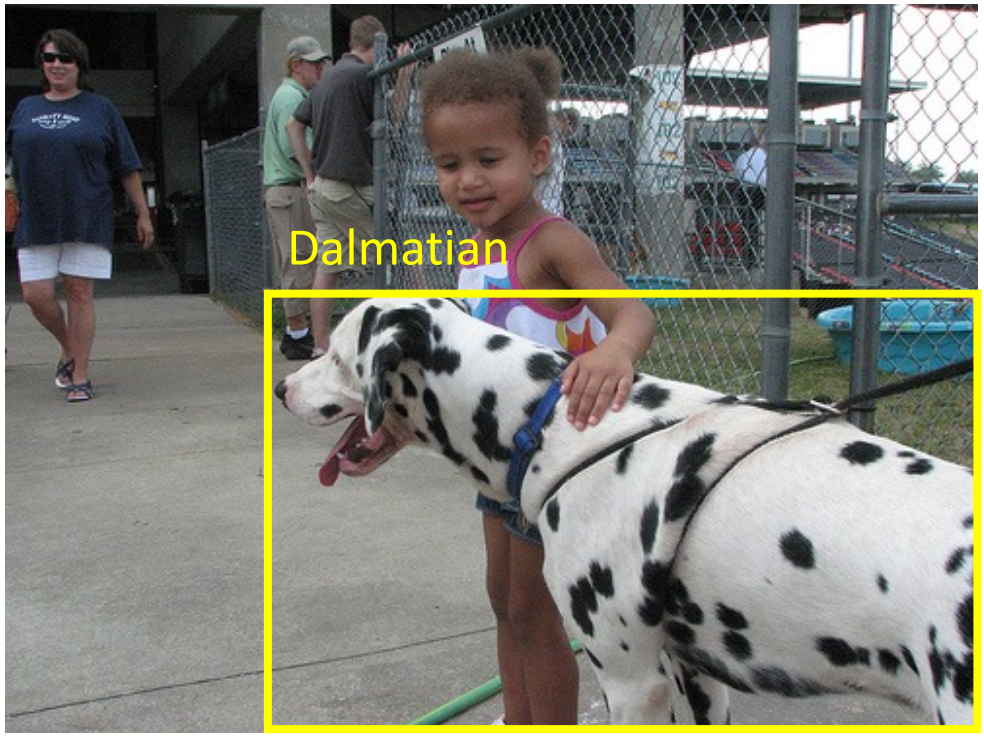


Large Scale Visual Recognition Challenge (ILSVRC)

2010-2017

~~20 object classes~~ ~~22,591 images~~

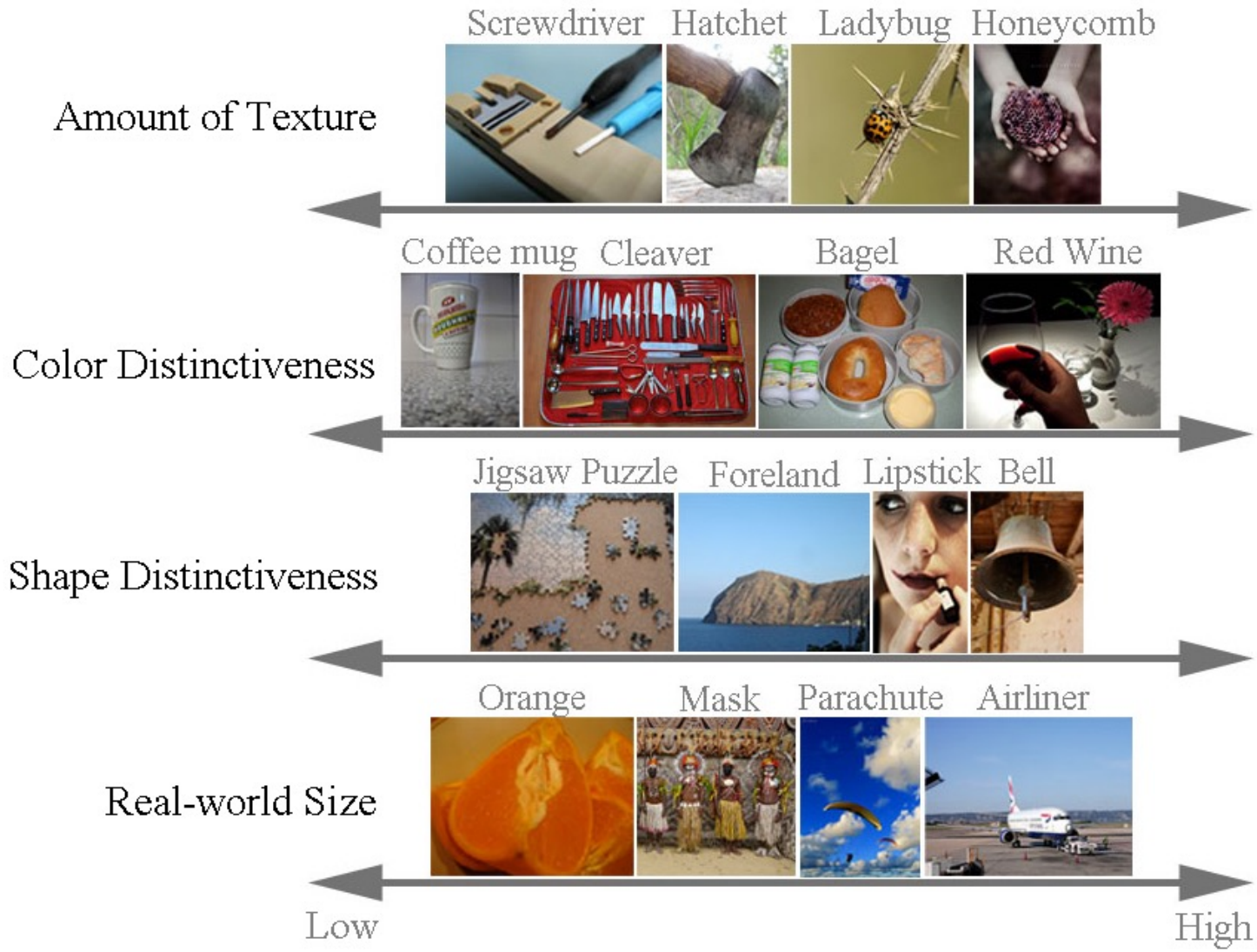
1000 object classes **1,431,167 images**



http://image-net.org/challenges/LSVRC/{2010,2011,2012}

Variety of object classes in ILSVRC

Variety of object classes in ILSVRC



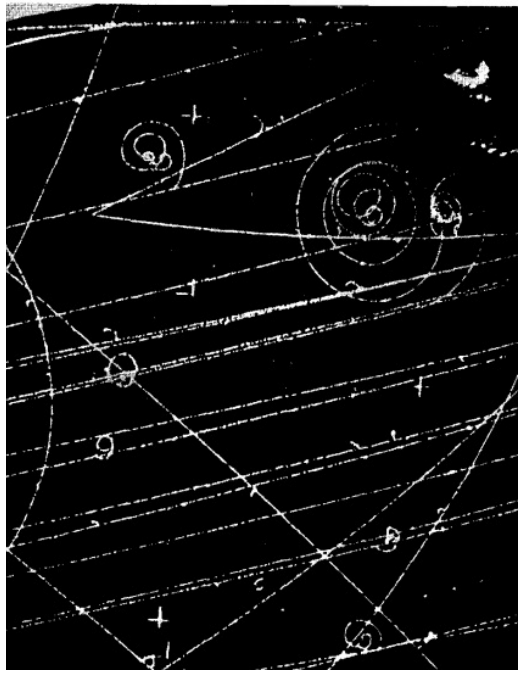
What Matters in Recognition?

- Data
  - More is always better (as long as it is good data)
  - Annotation is the hard part

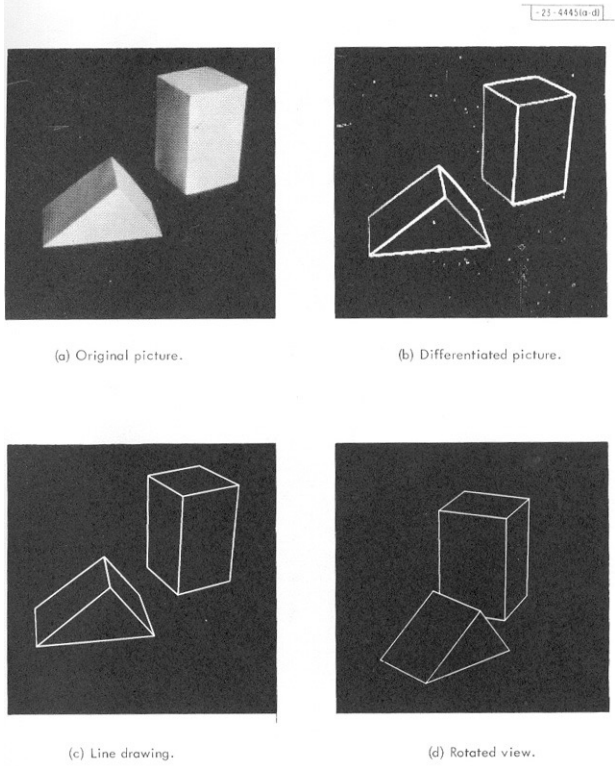
- Representation
  - **Low level: SIFT, HoG, GIST, edges**
  - **Mid level: Bag of words, sliding window, deformable model**
  - **High level: Contextual dependence**
  - Deep learned features

- Learning Techniques
  - E.g. choice of classifier or inference method

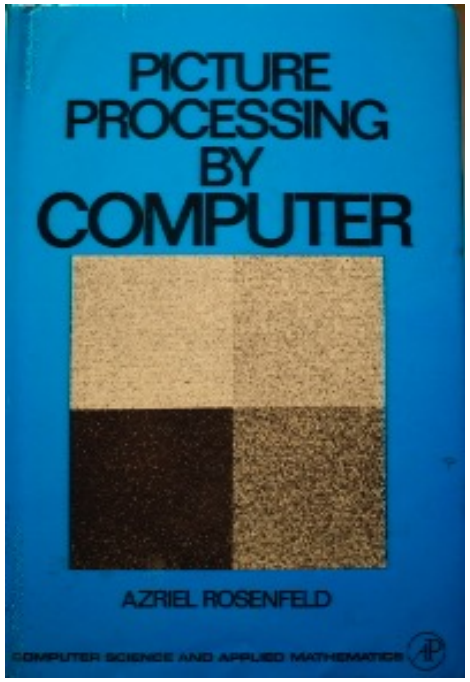
Recall: Origins of computer vision



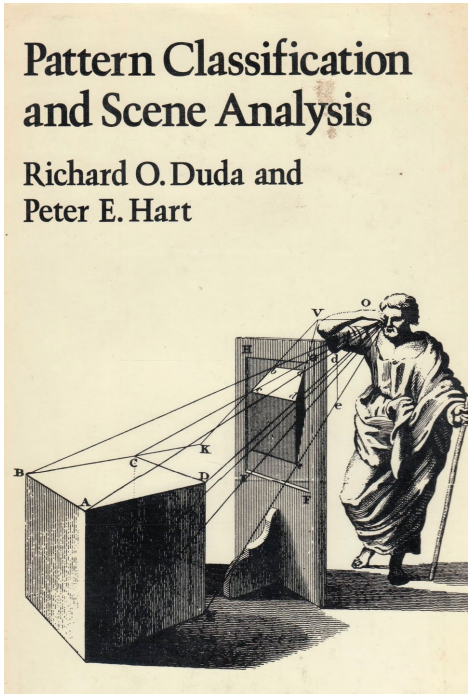
Hough, 1959



Roberts, 1963

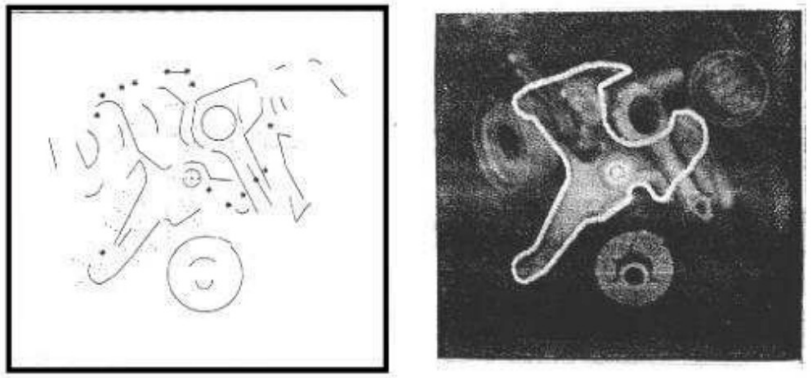


Rosenfeld, 1969

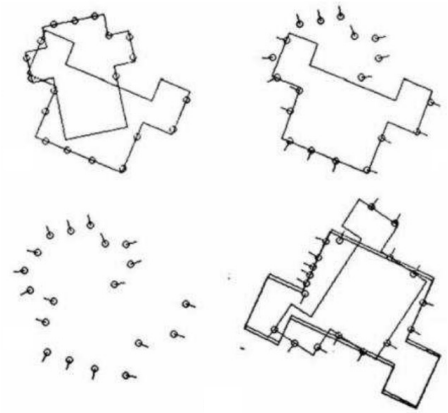


Duda & Hart, 1972

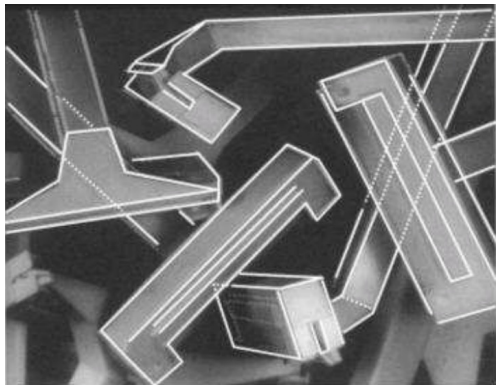
History of recognition: Geometric alignment



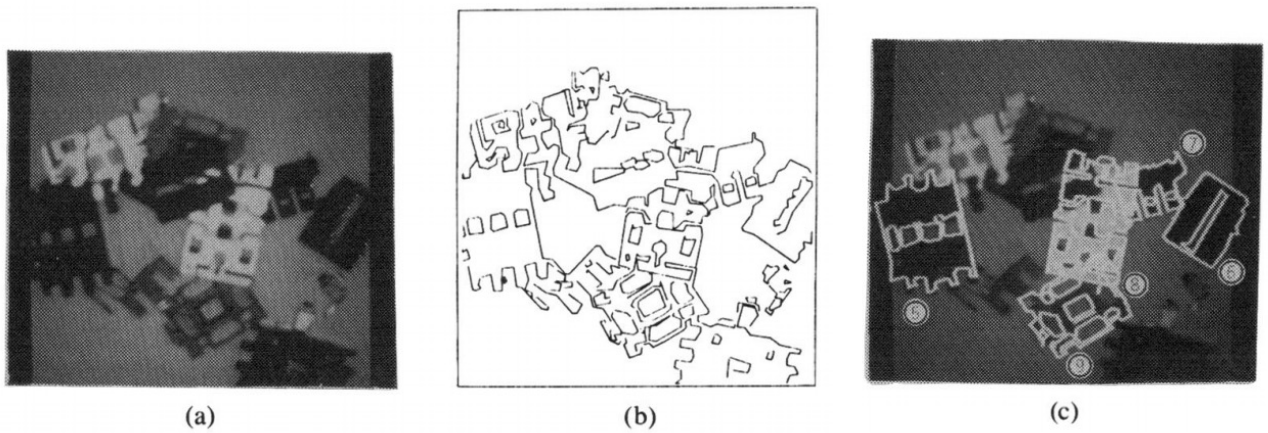
Perkins (1978)



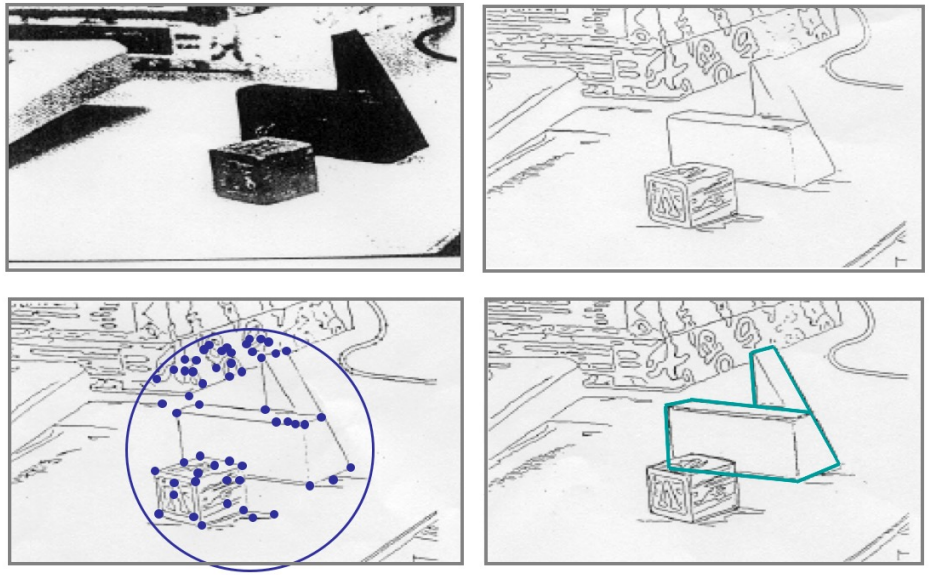
Grimson & Lozano-Perez (1984)



Lowe (1985)

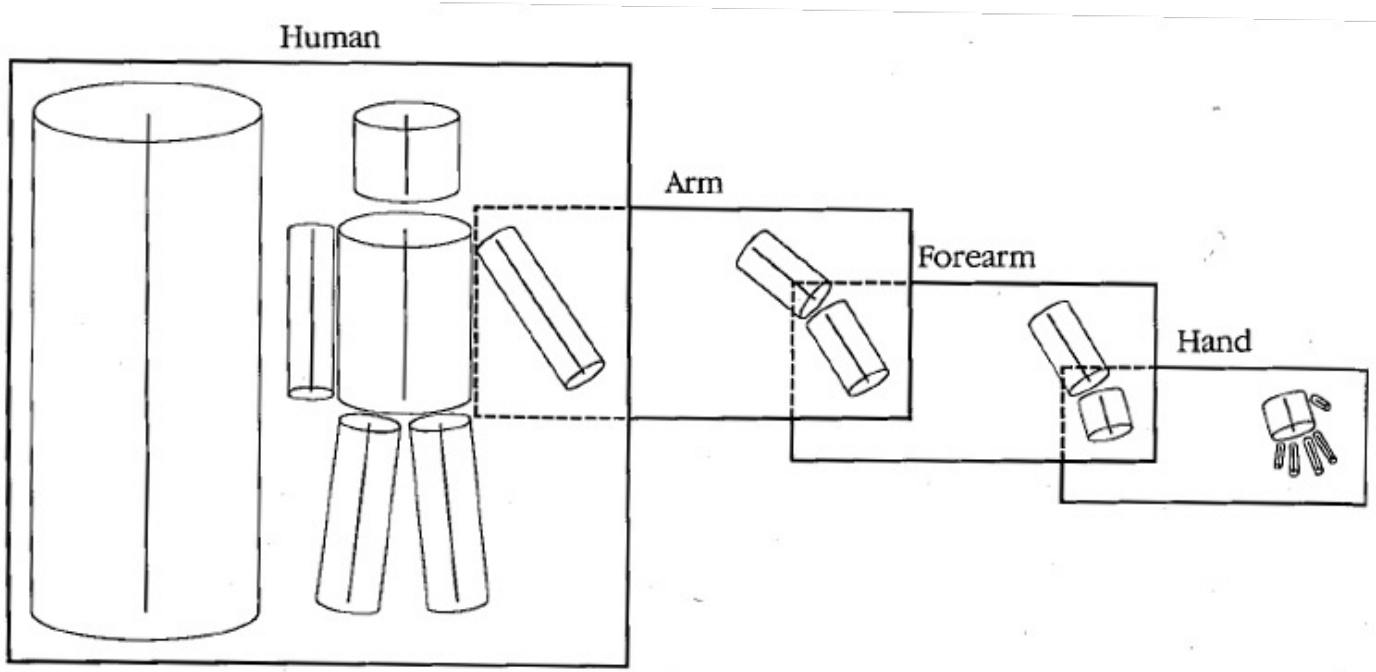


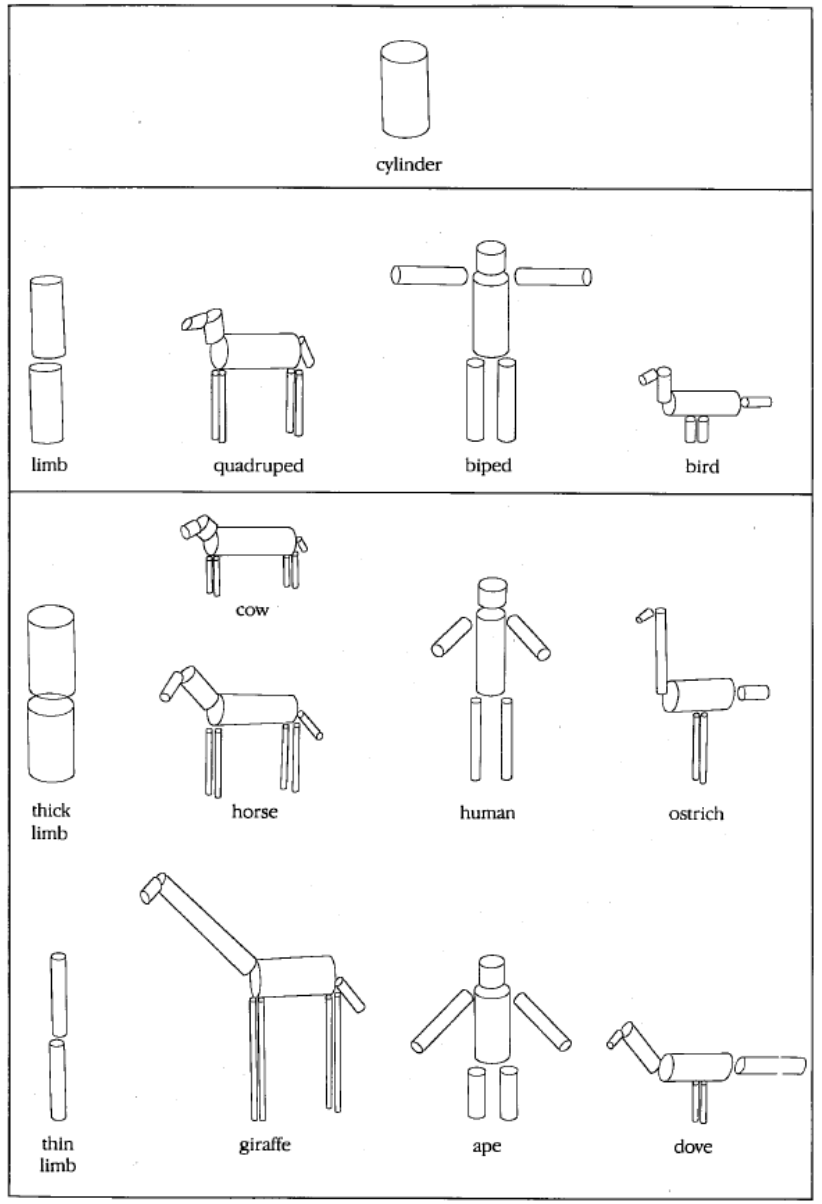
Ayache & Faugeras (1986)



Huttenlocher & Ullman (1987)

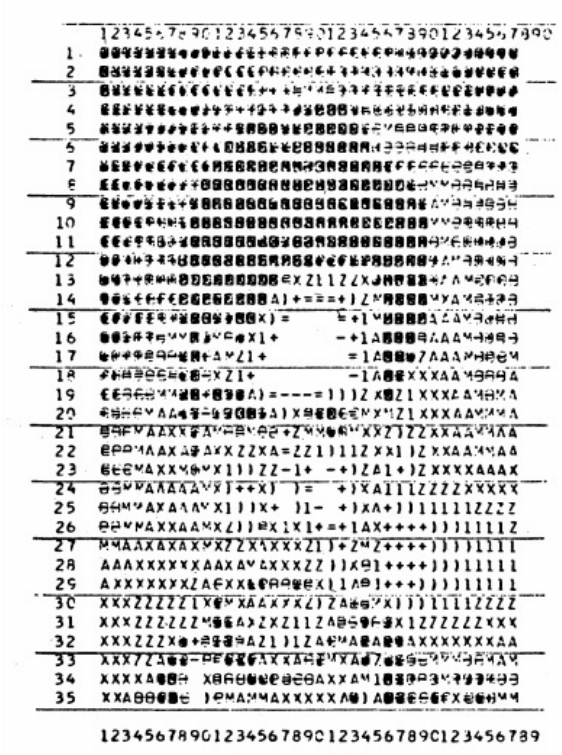
History of recognition: Hierarchies of parts



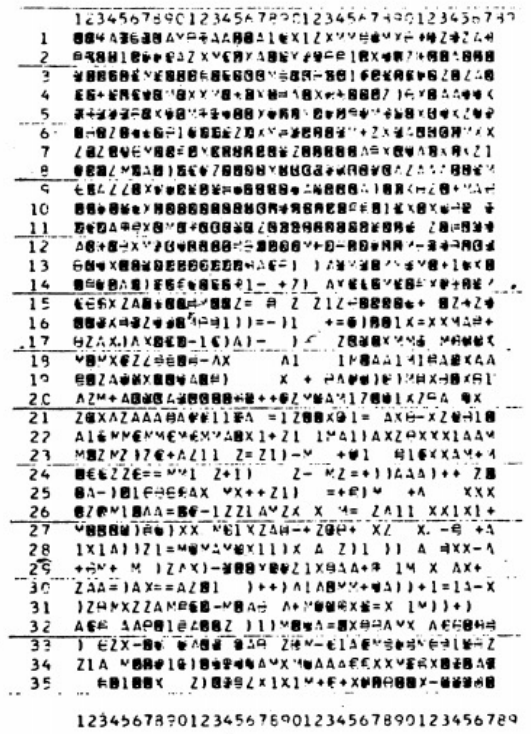


Figures from Marr's Vision (1982)

History of recognition: Deformable templates

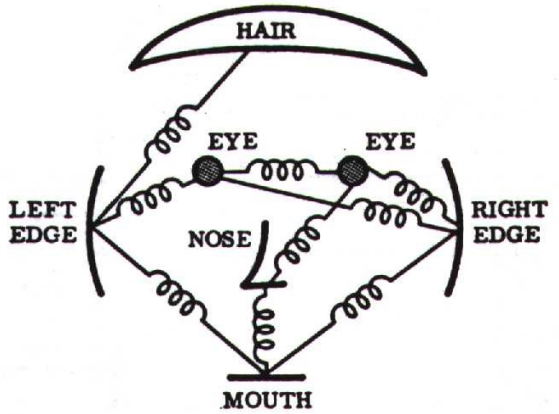


Original picture.



Noisy picture (sensed scene) as used in experiment.

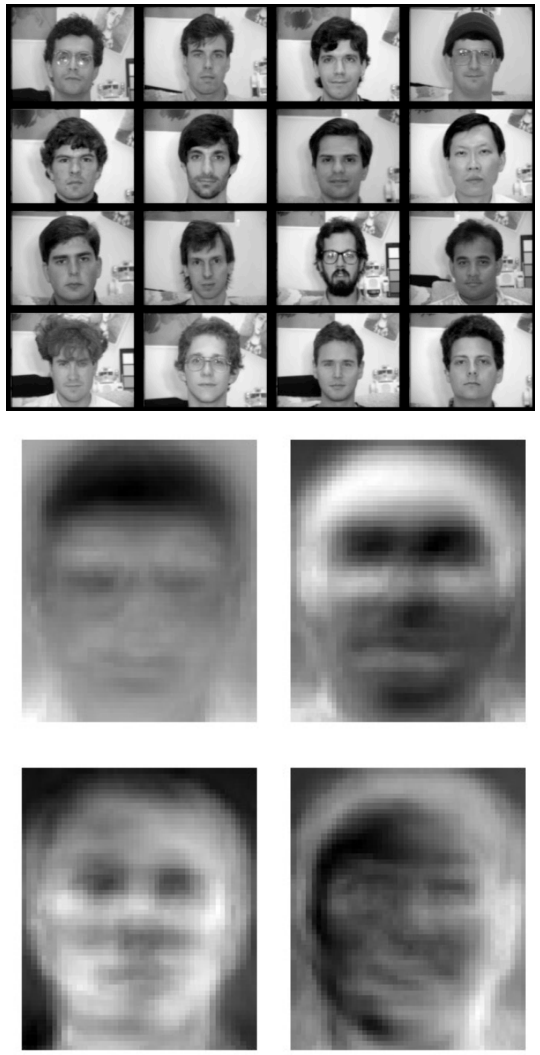
L(EV)A for hair. (Density at a point is proportional to probability that hair is present at that location.)



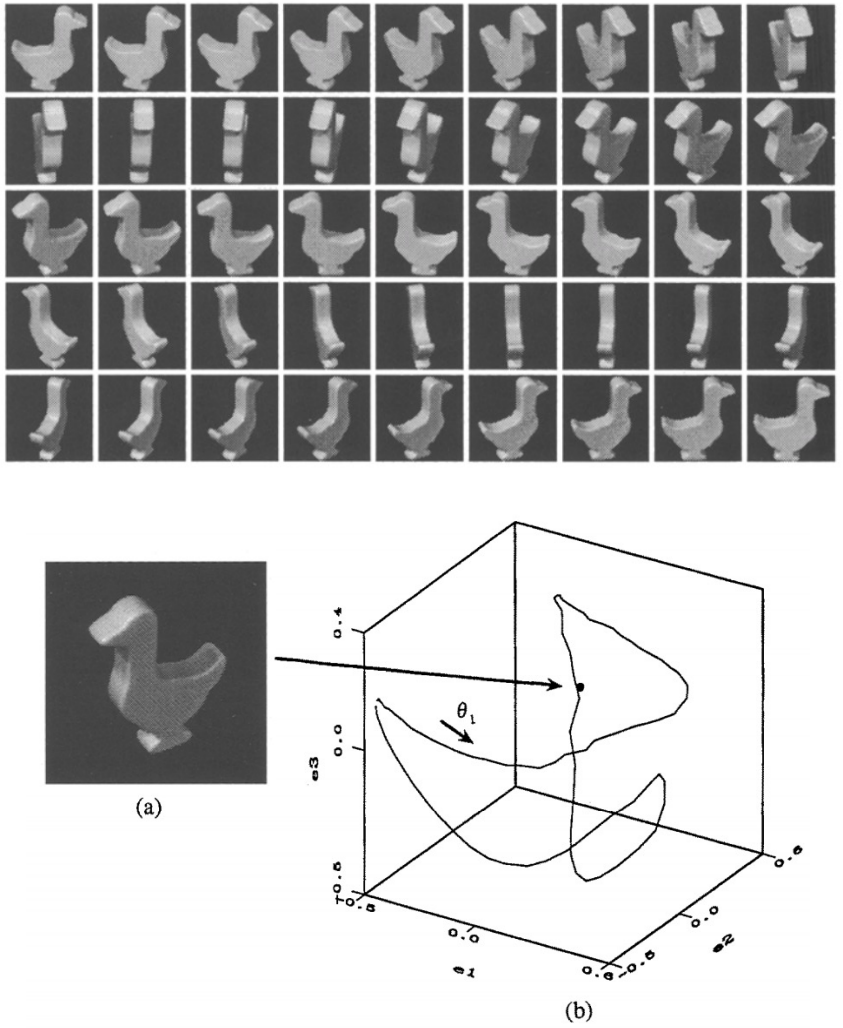
HAIR WAS LOCATED AT (11, 21)
L/EDGE WAS LOCATED AT (25, 11)
R/EDGE WAS LOCATED AT (25, 24)
L/EYE WAS LOCATED AT (21, 15)
R/EYE WAS LOCATED AT (21, 21)
NOSE WAS LOCATED AT (26, 18)
MOUTH WAS LOCATED AT (29, 17)

M. Fischler and R. Elschlager, The representation and matching of pictorial structures, IEEE Trans. on Computers, 1973

History of recognition: Appearance-based models



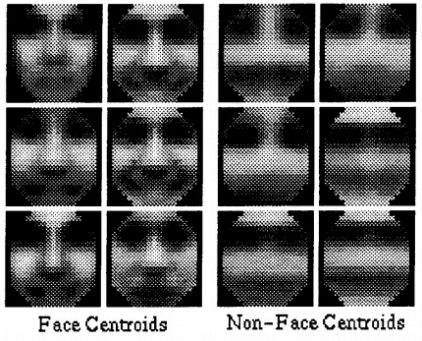
M. Turk and A. Pentland, Face recognition using eigenfaces, CVPR 1991



H. Murase and S. Nayar, Visual learning and recognition of 3-d objects from appearance, IJCV 1995

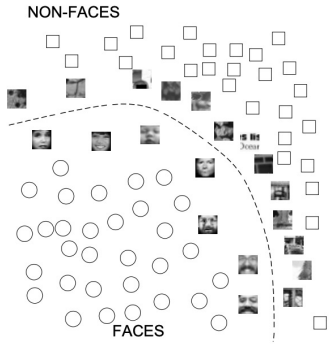
History of recognition: Features and classifiers

Appearance manifolds
+ neural network



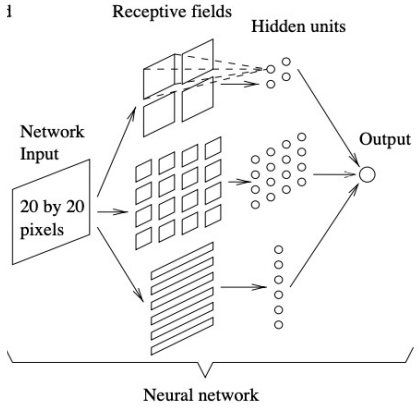
Sung & Poggio (1994)

Support vector machines



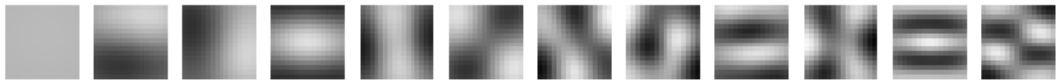
Osuna, Freund, Girosi (1997)

Neural network



Rowley, Baluja, Kanade (1998)

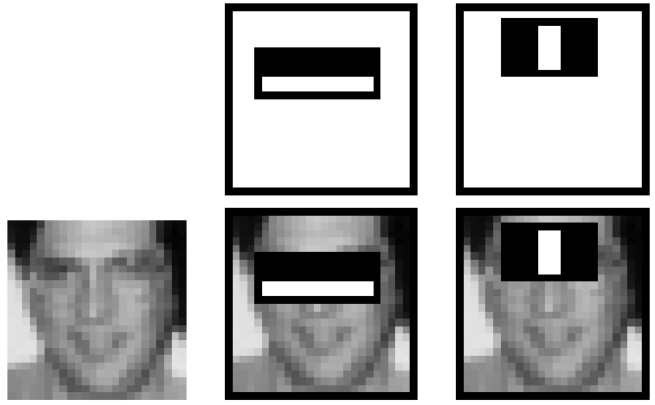
Statistics of feature responses, probabilistic classifier



$$\prod_{j=1}^{n_{magn}} \prod_{i=1}^{n_{subs}} \frac{P(q1_i^j \mid \text{object}) P(pos_i^{j'} \mid q2_i, \text{object})}{P(q1_i^j \mid \overline{\text{object}})} \underset{\overline{\text{object}}}{\overset{\text{object}}{>\!<}} \lambda = \frac{P(\overline{\text{object}})}{P(\text{object})}$$

Schneiderman & Kanade (1998)

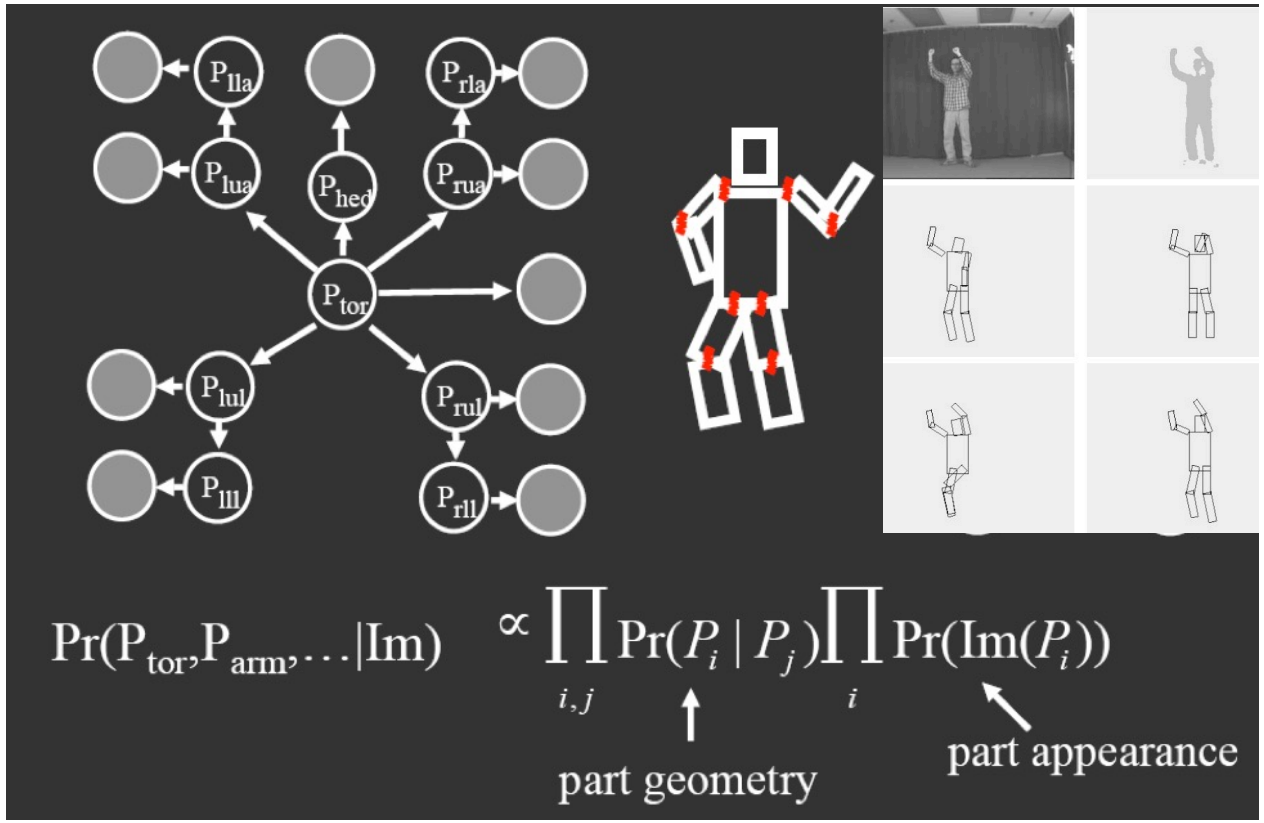
Rectangle features, boosting



Viola & Jones (2001)

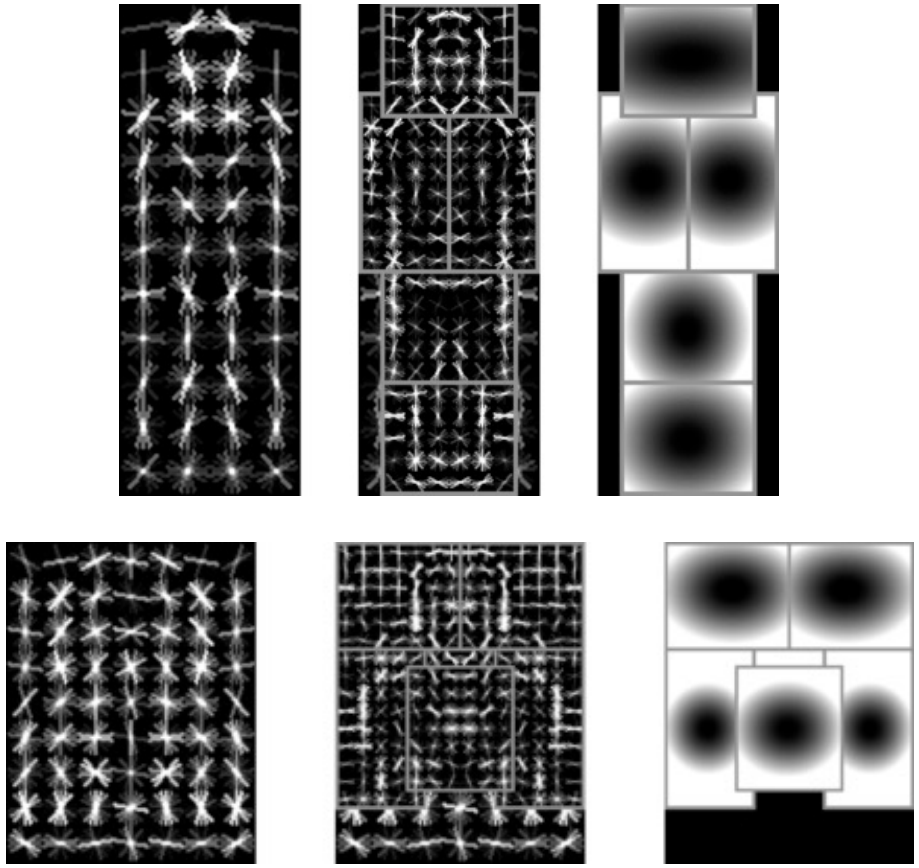
History of recognition: Deformable templates

Pictorial structures revisited



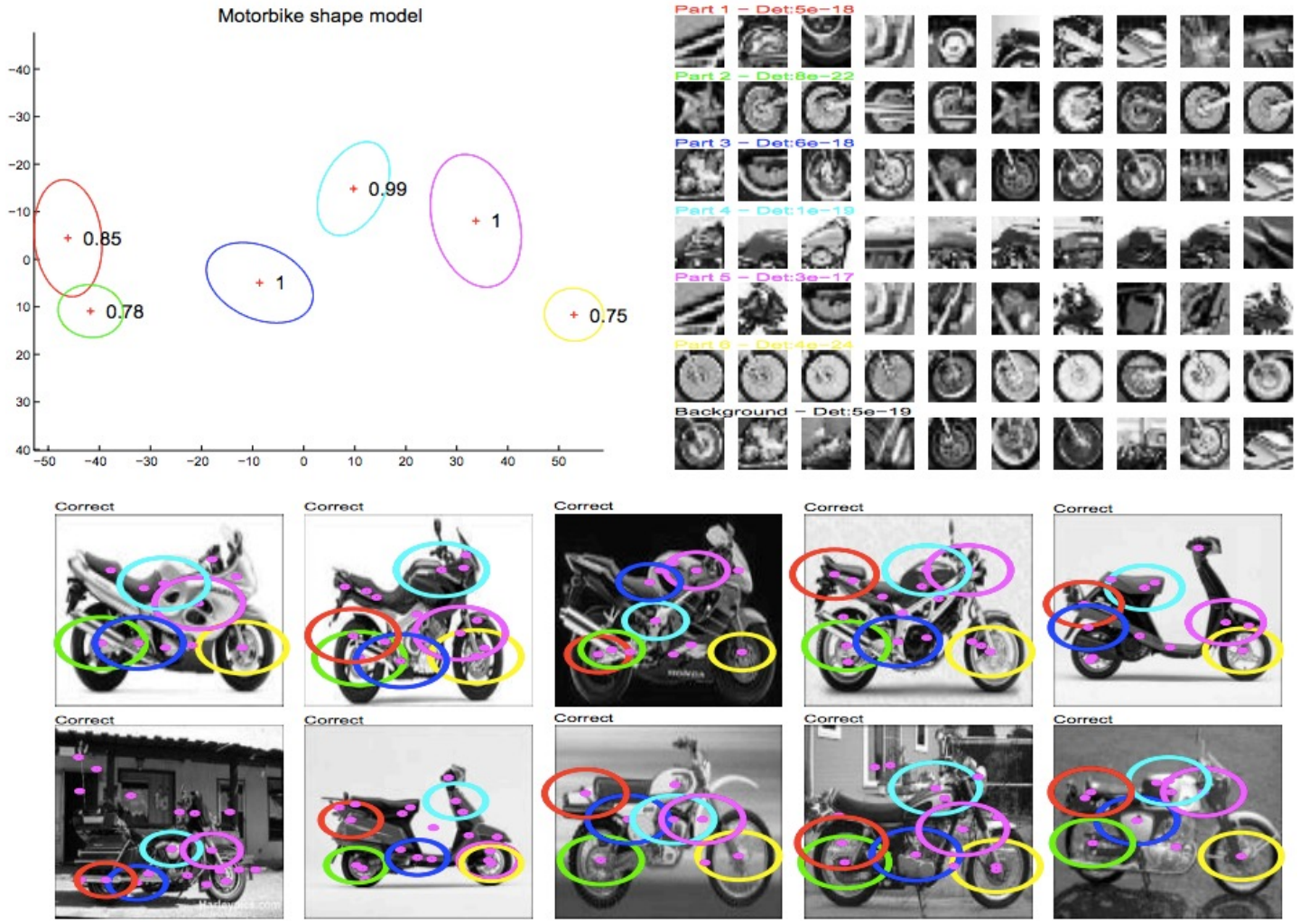
Felzenszwalb & Huttenlocher (2000)

Discriminatively trained deformable part-based models



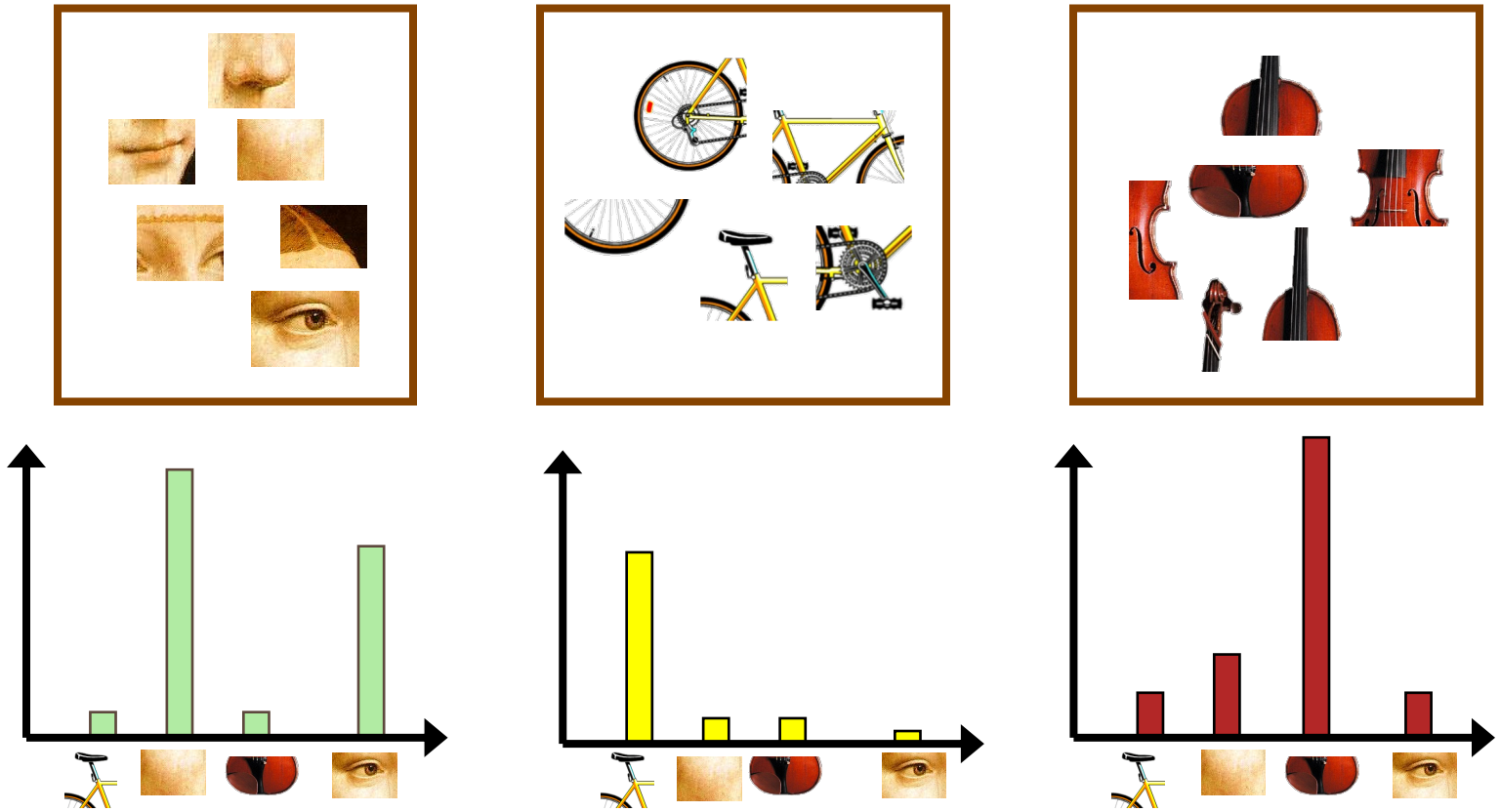
Felzenszwalb et al. (2008)

History of recognition: Constellation models



Weber, Welling & Perona (2000), Fergus, Perona & Zisserman (2003)

History of recognition: Bags of keypoints

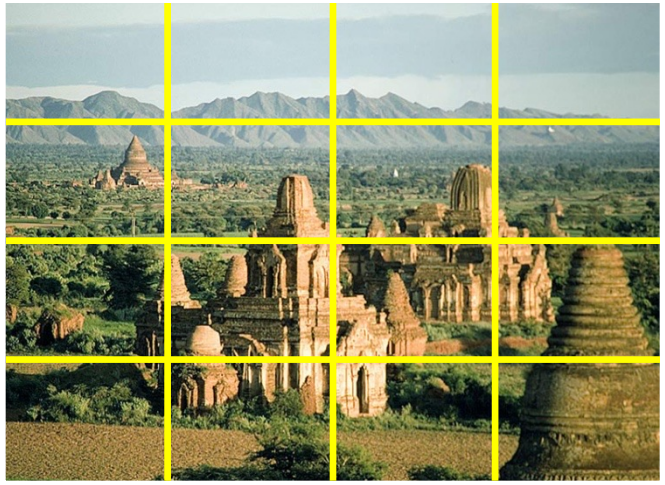


Csurka et al. (2004), Willamowski et al. (2005), Grauman & Darrell (2005), Sivic et al. (2003, 2005)

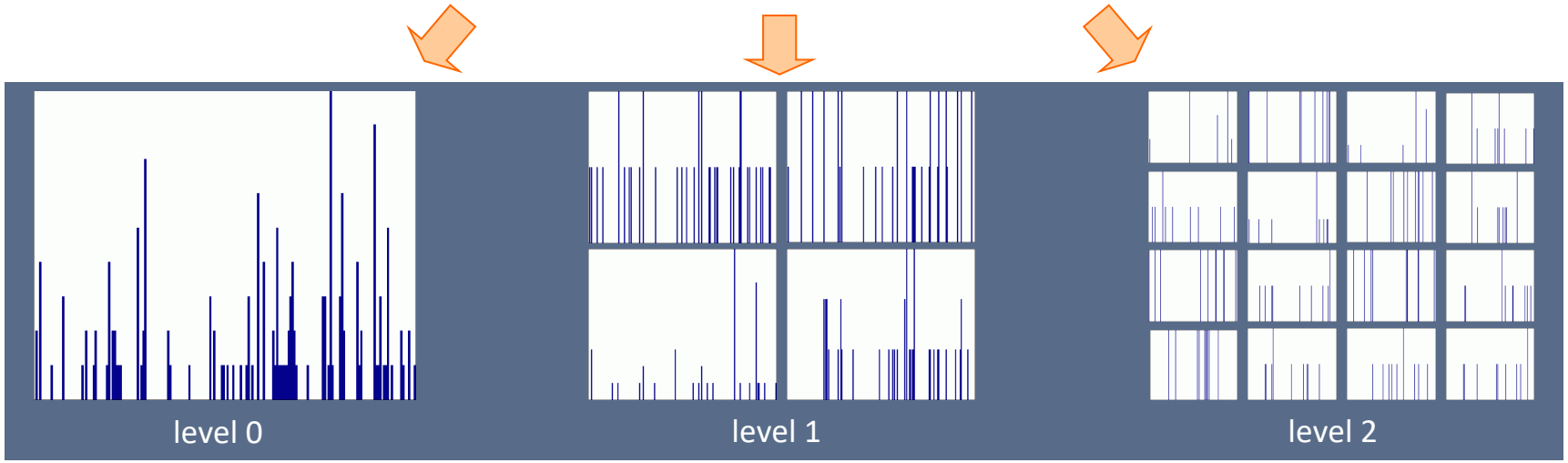
Inspired by Bag of Words features from NLP

Spatial pyramids

- Orderless pooling of local features over a coarse grid



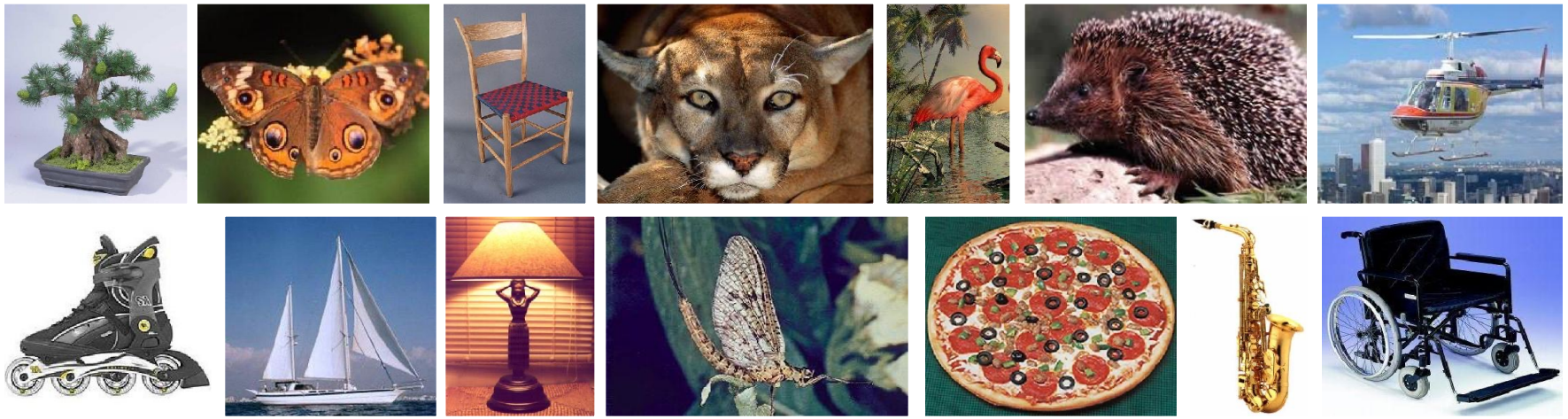
Pyramid representations are very popular in Vision, and still used in conjunction with deep learning.



Lazebnik, Schmid & Ponce (CVPR 2006)

Spatial pyramids

- Caltech101 classification results:

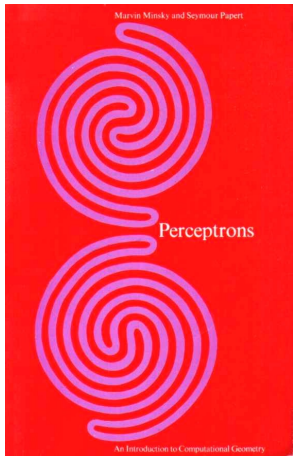


	Weak features (16)		Strong features (200)	
Level	Single-level	Pyramid	Single-level	Pyramid
0	15.5 ±0.9		41.2 ±1.2	
1	31.4 ±1.2	32.8 ±1.3	55.9 ±0.9	57.0 ±0.8
2	47.2 ±1.1	49.3 ±1.4	63.6 ±0.9	**64.6** ±0.8
3	52.2 ±0.8	**54.0** ±1.1	60.3 ±0.9	64.6 ±0.7

History of recognition: Neural networks

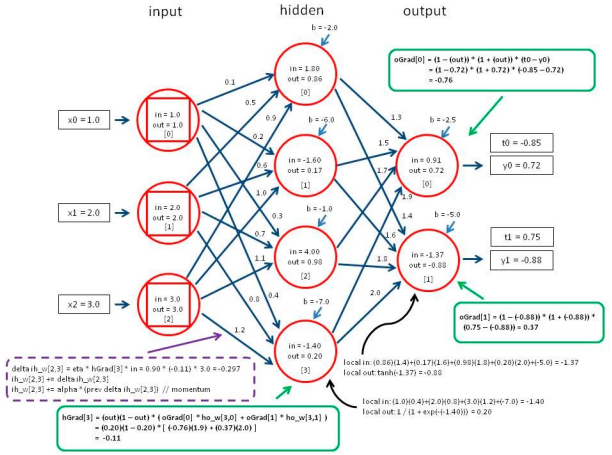
Perceptrons

Rosenblatt (1958)



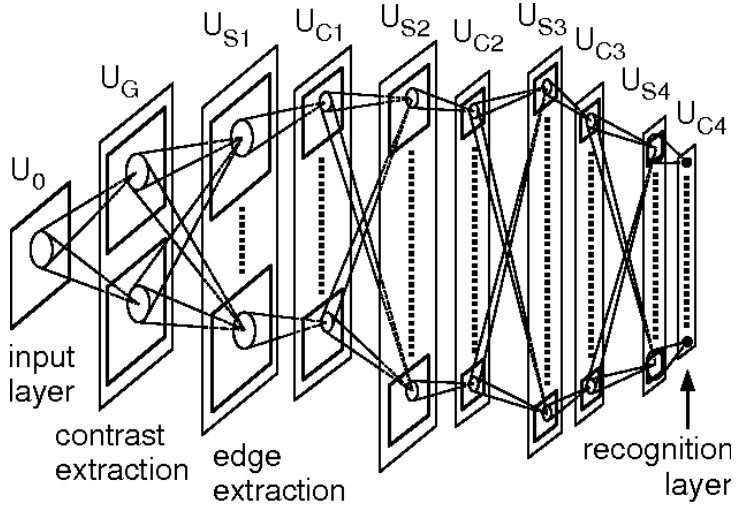
Minsky & Papert (1969)

Back-propagation



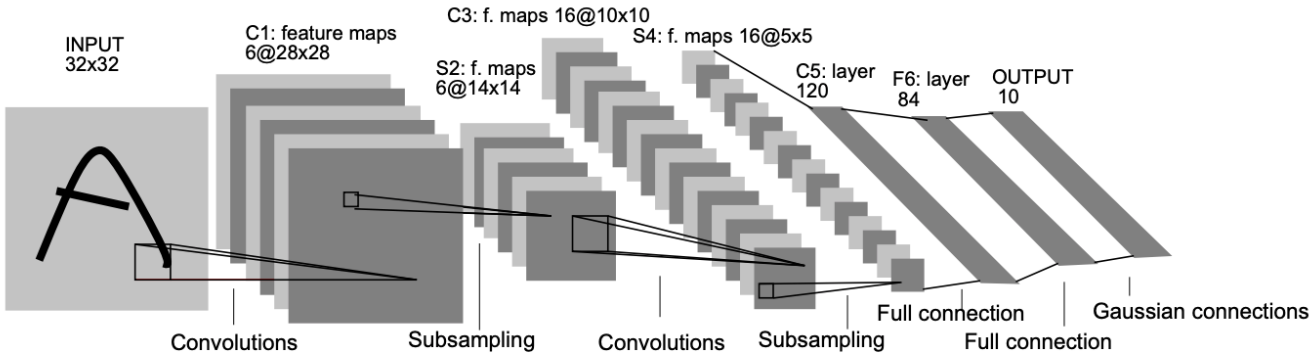
Rumelhart, Hinton & Williams (1986)

Neocognitron



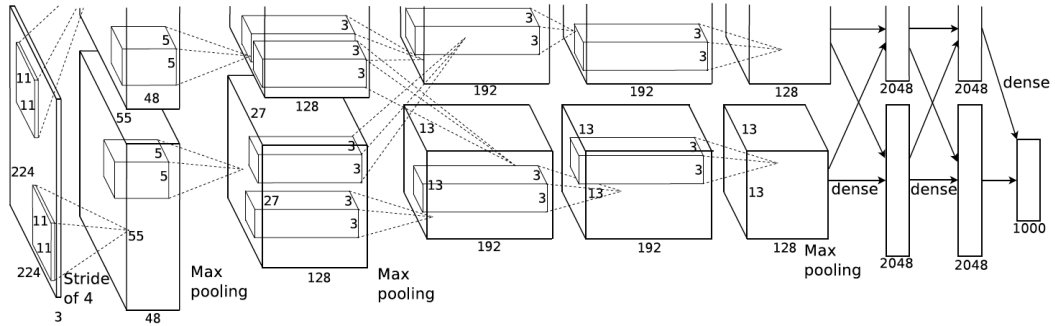
Fukushima (1980)

LeNet-5



LeCun et al. (1998)

AlexNet

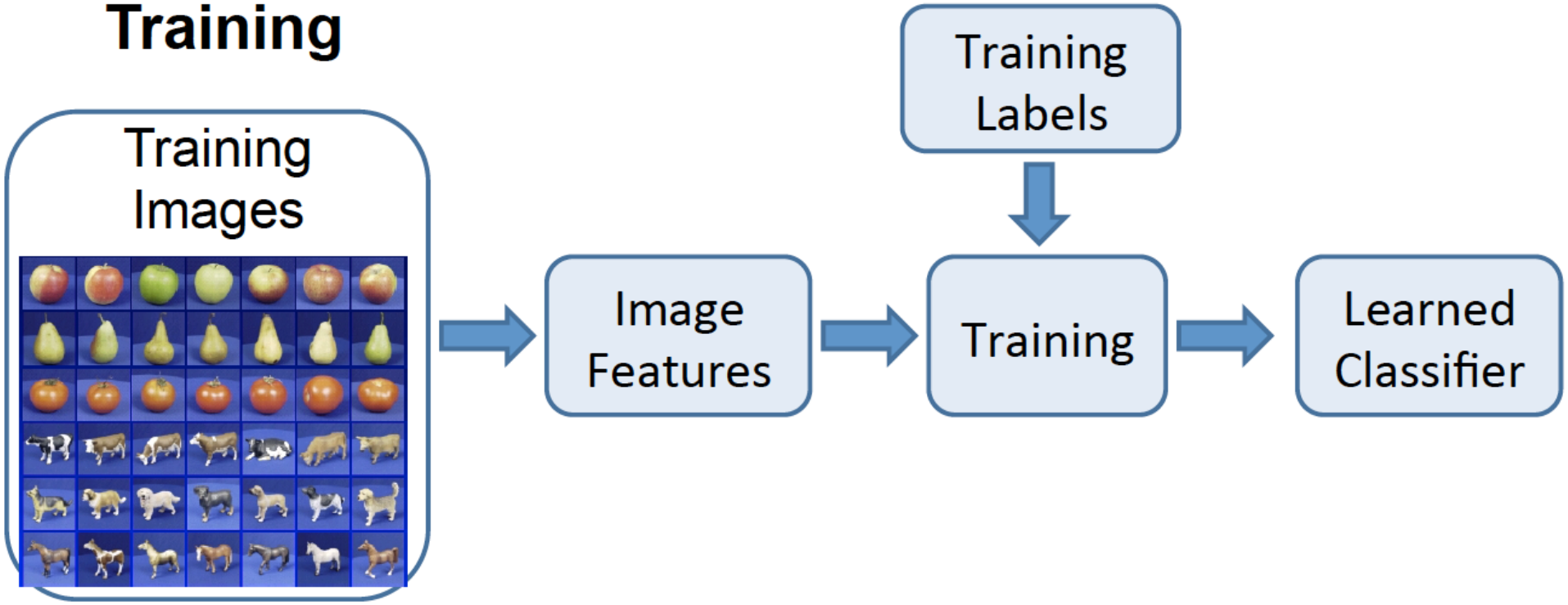


Krizhevsky et al. (2012)

What Matters in Recognition?

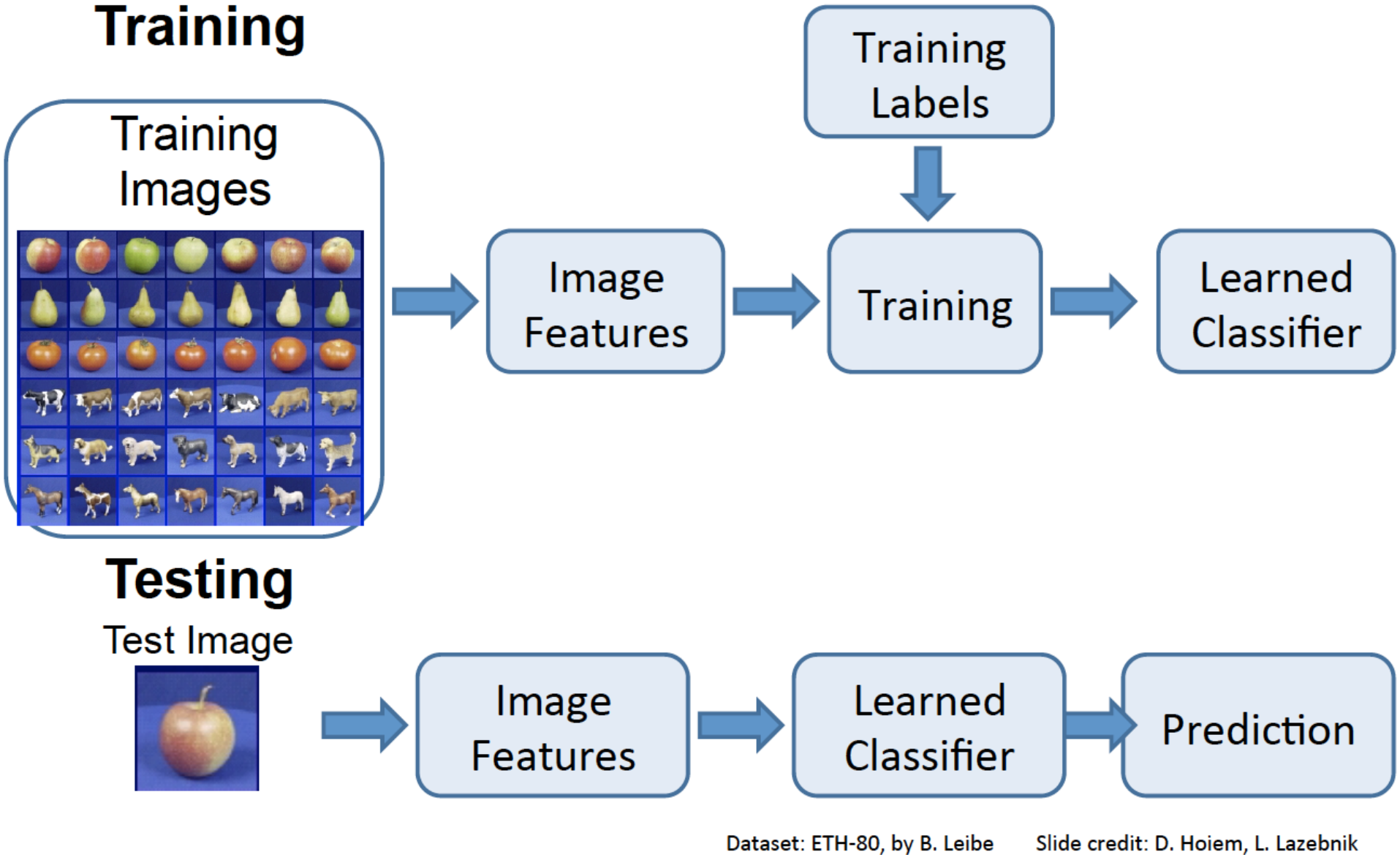
- Data
  - More is always better (as long as it is good data)
  - Annotation is the hard part
- Representation
  - Low level: SIFT, HoG, GIST, edges
  - Mid level: Bag of words, sliding window, deformable model
  - High level: Contextual dependence
  - Deep learned features
- Learning Techniques
  - E.g. choice of classifier or inference method

Training & Testing a Classifier



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

Training & Testing a Classifier



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

Classifiers

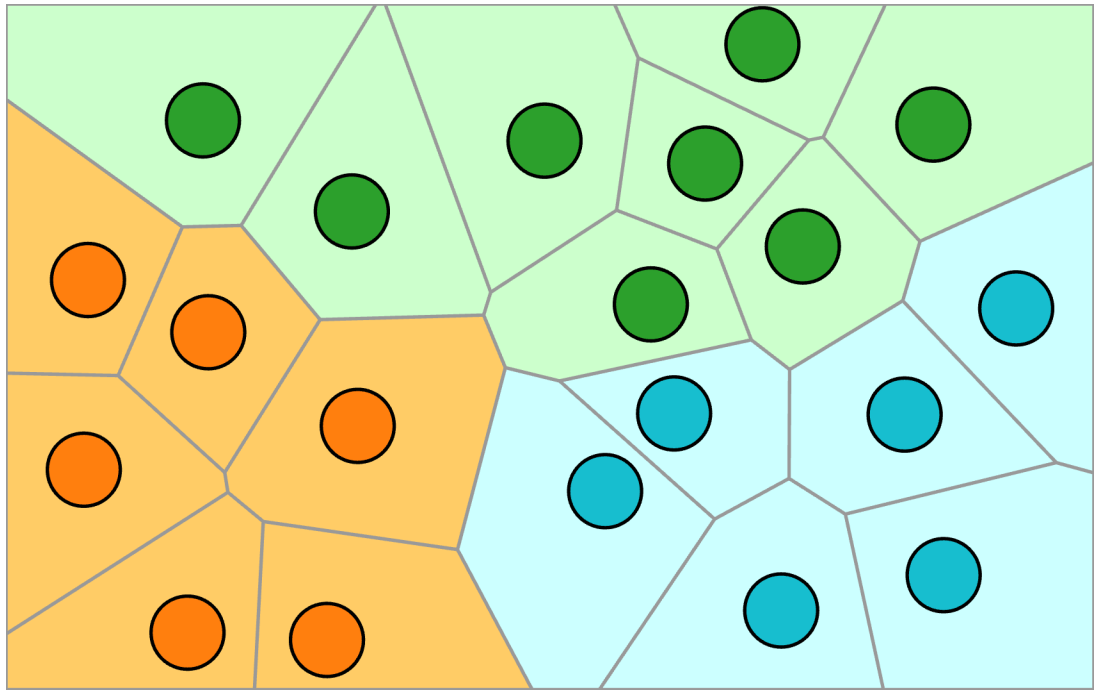
- Nearest Neighbor
- kNN (“k-Nearest Neighbors”)
- Linear Classifier
- Neural Network
- Deep Neural Network
- …

Classifiers

- Nearest Neighbor
- kNN (“k-Nearest Neighbors”)
- Linear Classifier
- Neural Network
- Deep Neural Network
- …

Nearest Neighbor (NN) Classifier

- Train
  - Remember all training images and their labels
- Predict
  - Find the closest (most similar) training image
  - Predict its label as the true label



CIFAR-10 and NN results

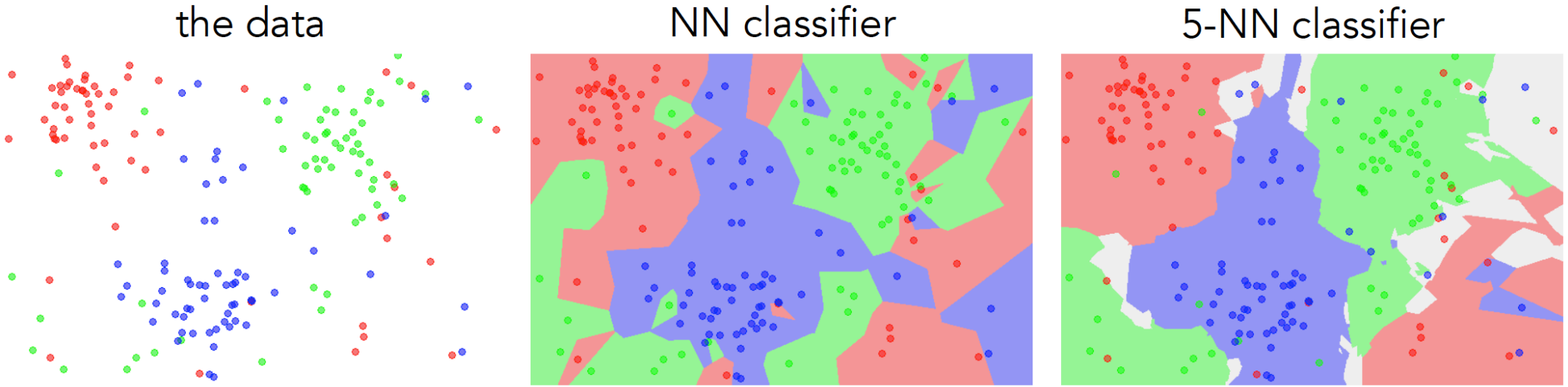
Example dataset: **CIFAR-10**
10 labels
50,000 training images
10,000 test images.

For every test image (first column), examples of nearest neighbors in rows

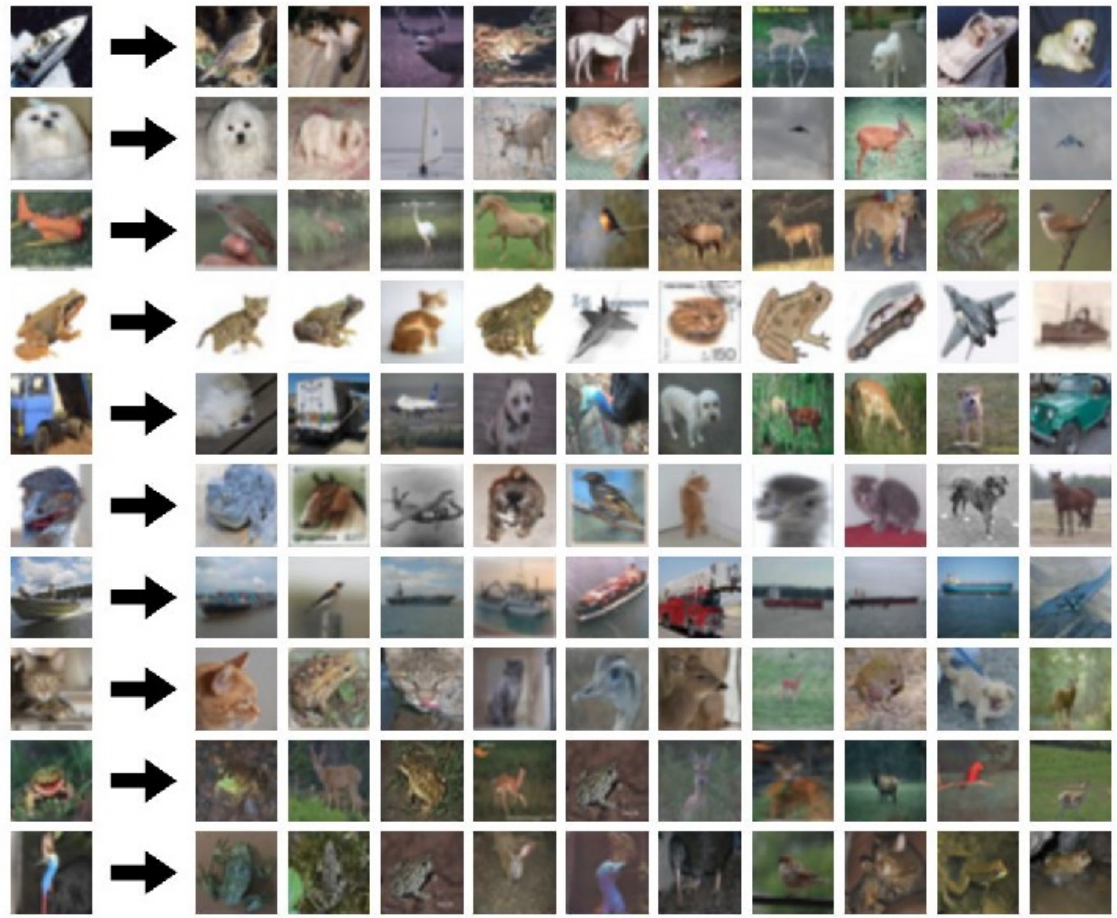
Slides from Andrej Karpathy and Fei-Fei Li
http://vision.stanford.edu/teaching/cs231n/

k-nearest neighbor

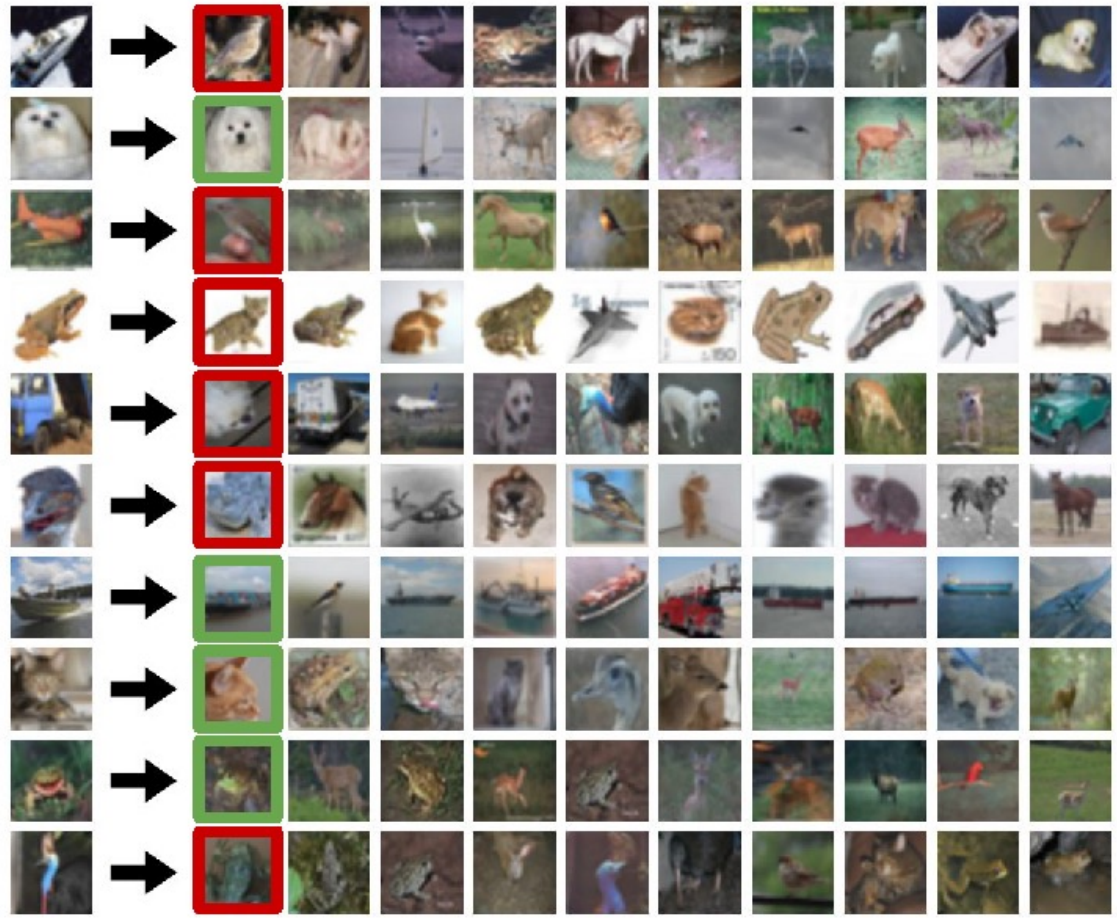
- Find the k closest points from training data
- Take **majority vote** from K closest points



What does this look like?



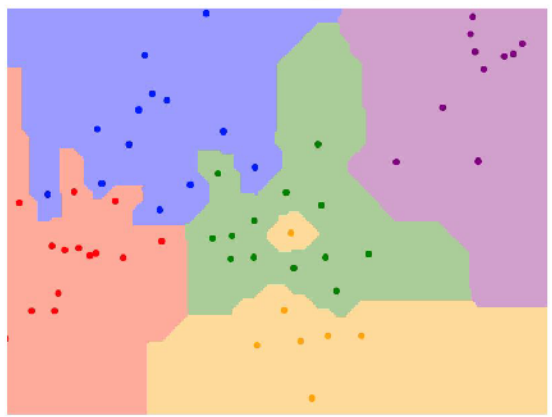
What does this look like?



K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

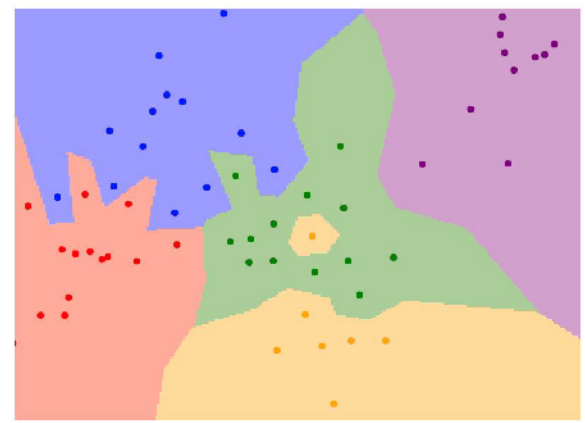
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



K = 1

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



K = 1

Demo: http://vision.stanford.edu/teaching/cs231n-demos/knn/

Hyperparameters

- What is the **best distance** to use?
- What is the **best value of k** to use?

- These are **hyperparameters**: choices about the algorithm that we set rather than learn

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

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

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

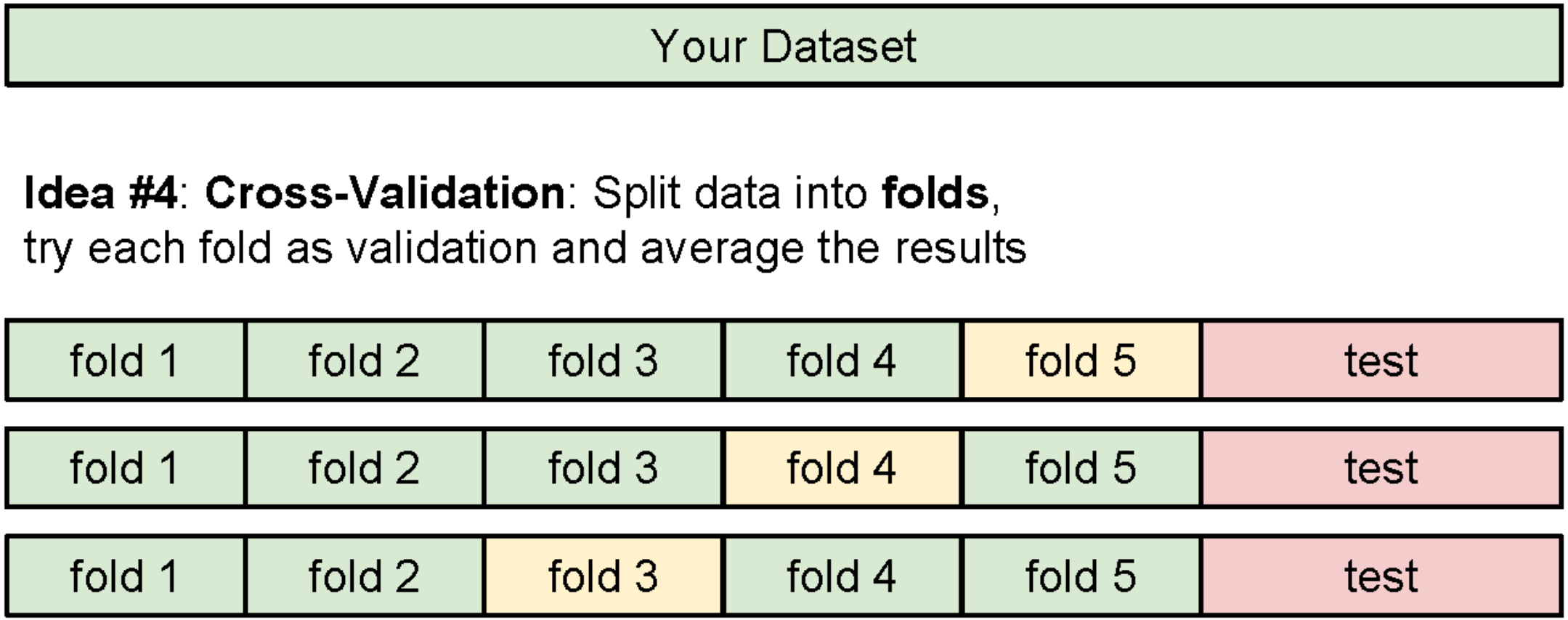
BAD: No idea how algorithm will perform on new data

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

Better!

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

Setting Hyperparameters

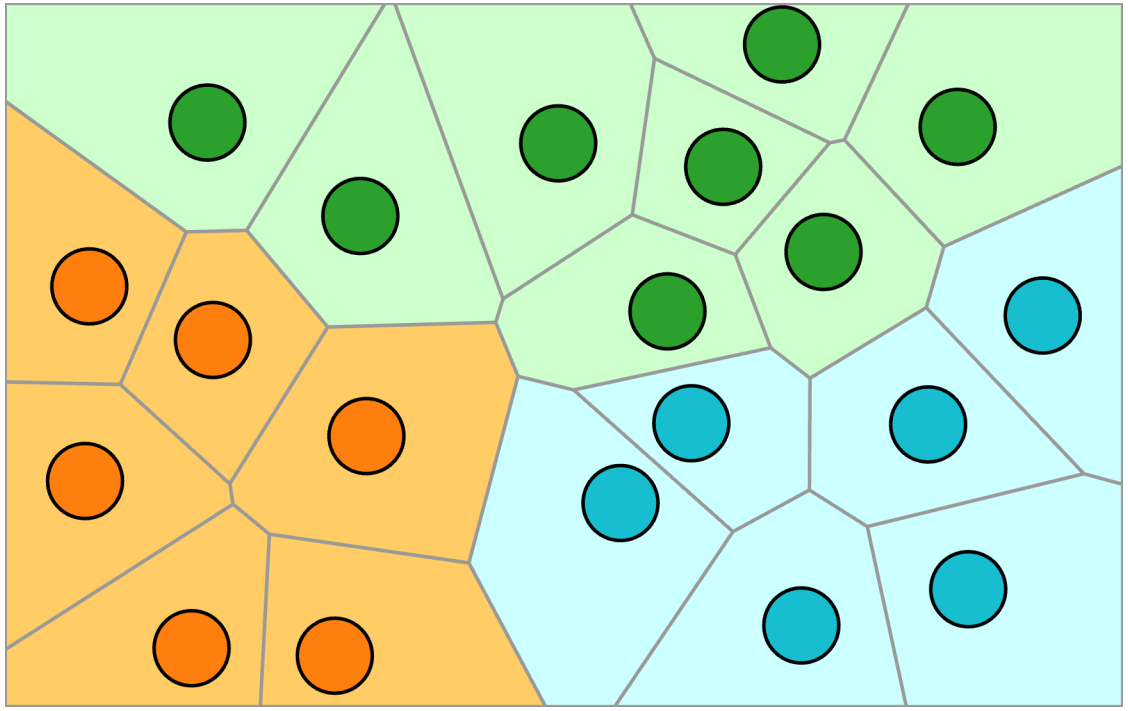


kNN -- Complexity and Storage

- N training images, M test images

- Training: O(1)
- Testing: O(MN)

- We often need the opposite:
  - Slow training is ok
  - Fast testing is necessary

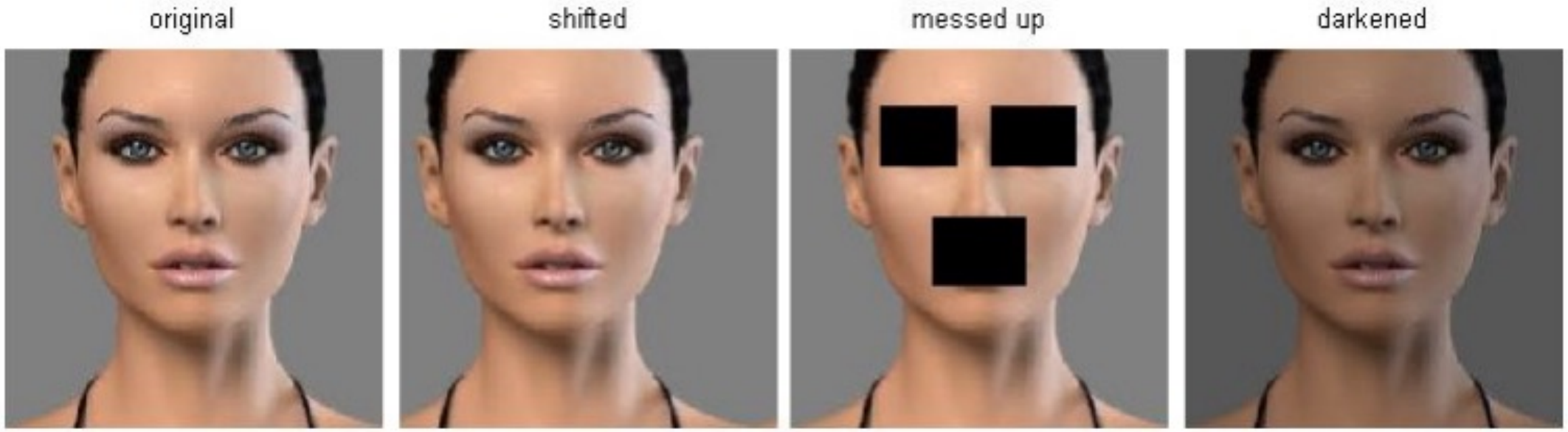


k-Nearest Neighbors: Summary

- In **image classification** we start with a **training set** of images and labels, and must predict labels on the **test set**
- The **K-Nearest Neighbors** classifier predicts labels based on nearest training examples
- Distance metric and K are **hyperparameters**
- Choose hyperparameters using the **validation set**; only run on the test set once at the very end!

Problems with KNN: Distance Metrics

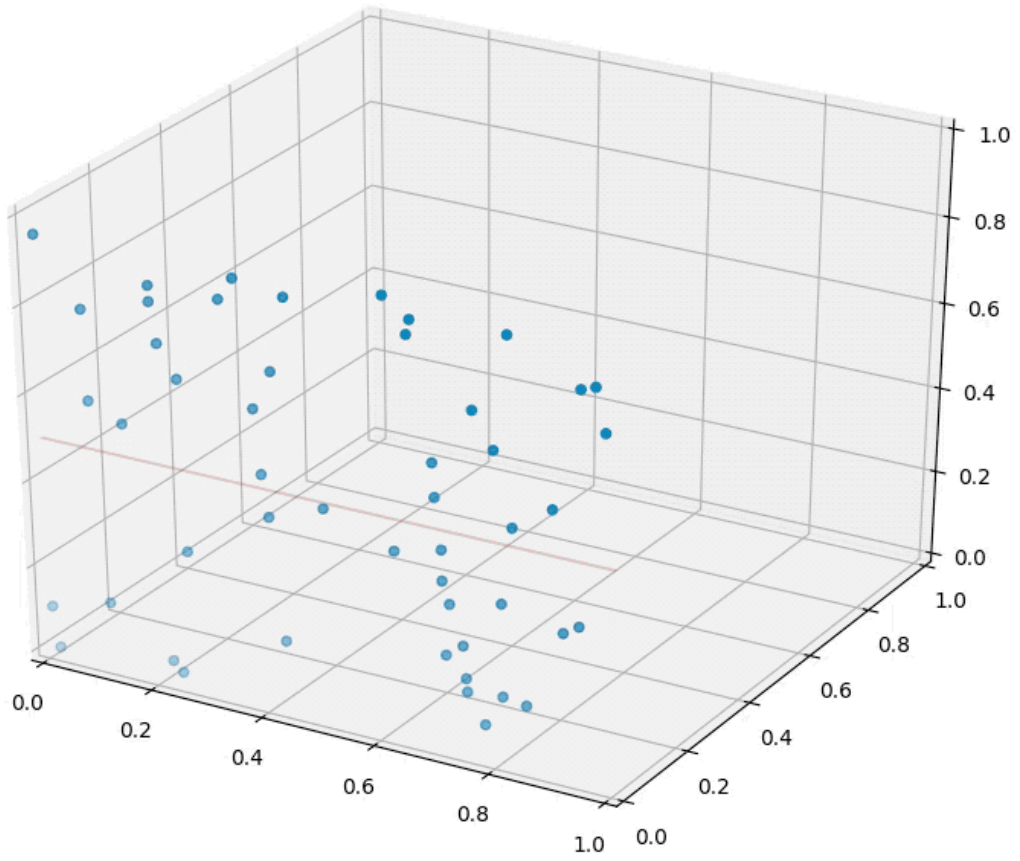
- terrible performance at test time
- distance metrics on level of whole images can be very unintuitive



(all 3 images have same L2 distance to the one on the left)

Problems with KNN: The Curse of Dimensionality

- As the number of dimensions increases, the same amount of data becomes more sparse.
- Amount of data we need ends up being exponential in the number of dimensions

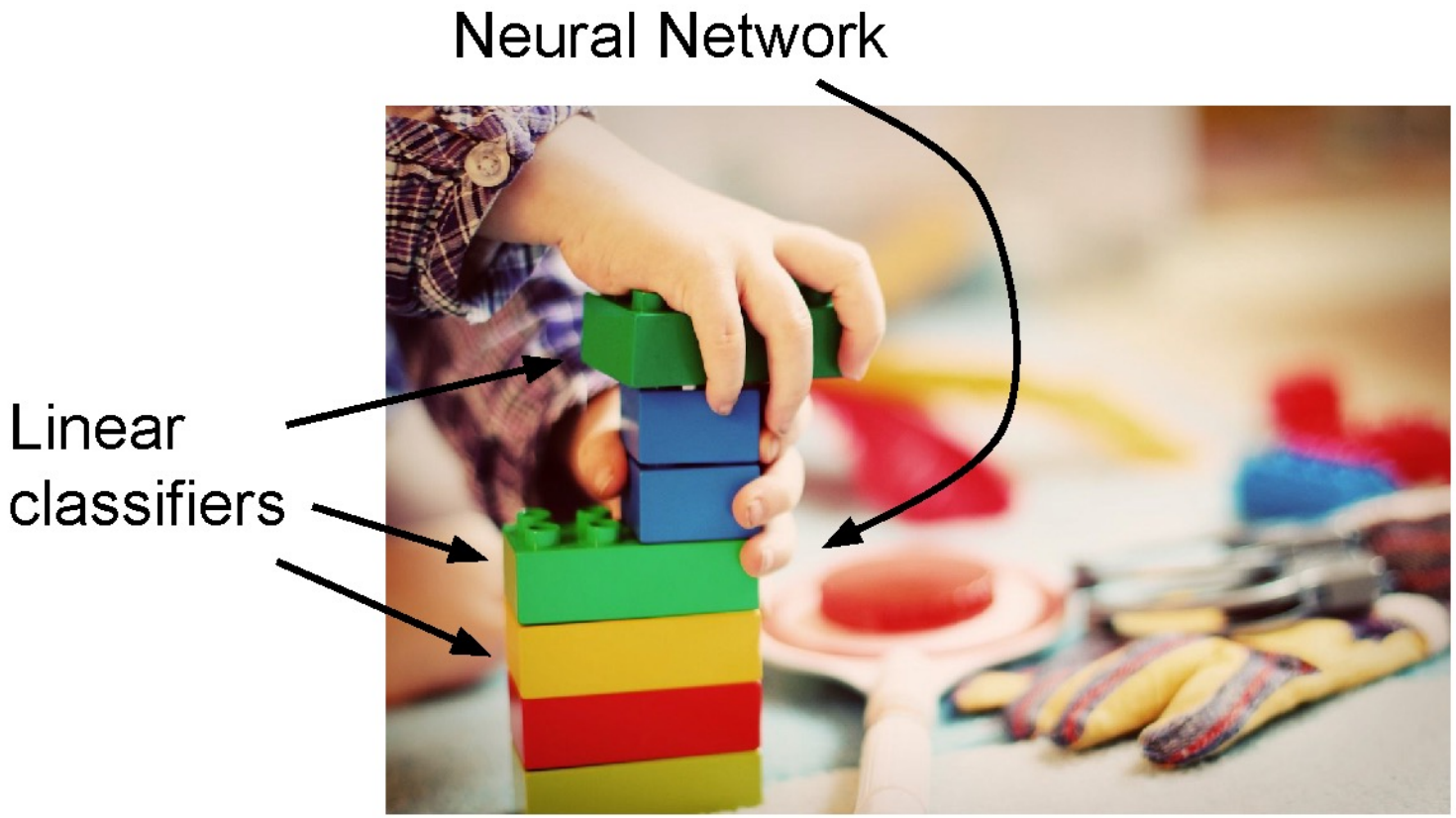


Animation from https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html

Classifiers

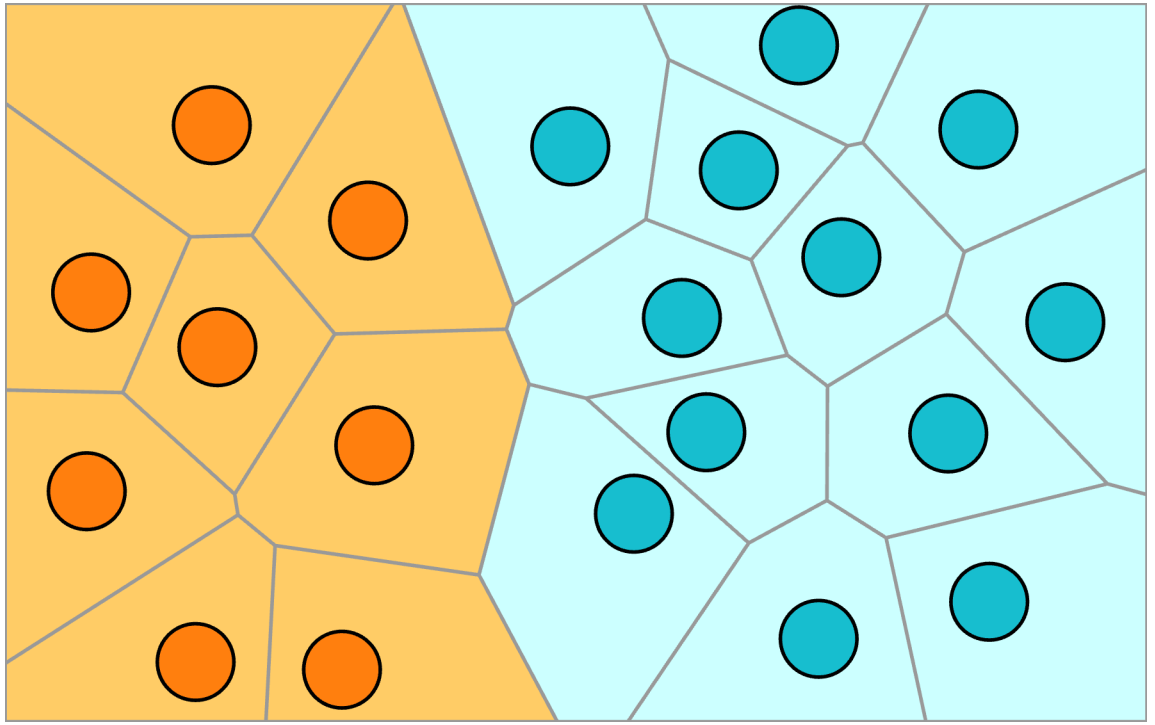
- Nearest Neighbor
- kNN (“k-Nearest Neighbors”)
- Linear Classifier
- Neural Network
- Deep Neural Network
- …

Linear Classifiers



Linear Classification vs. Nearest Neighbors

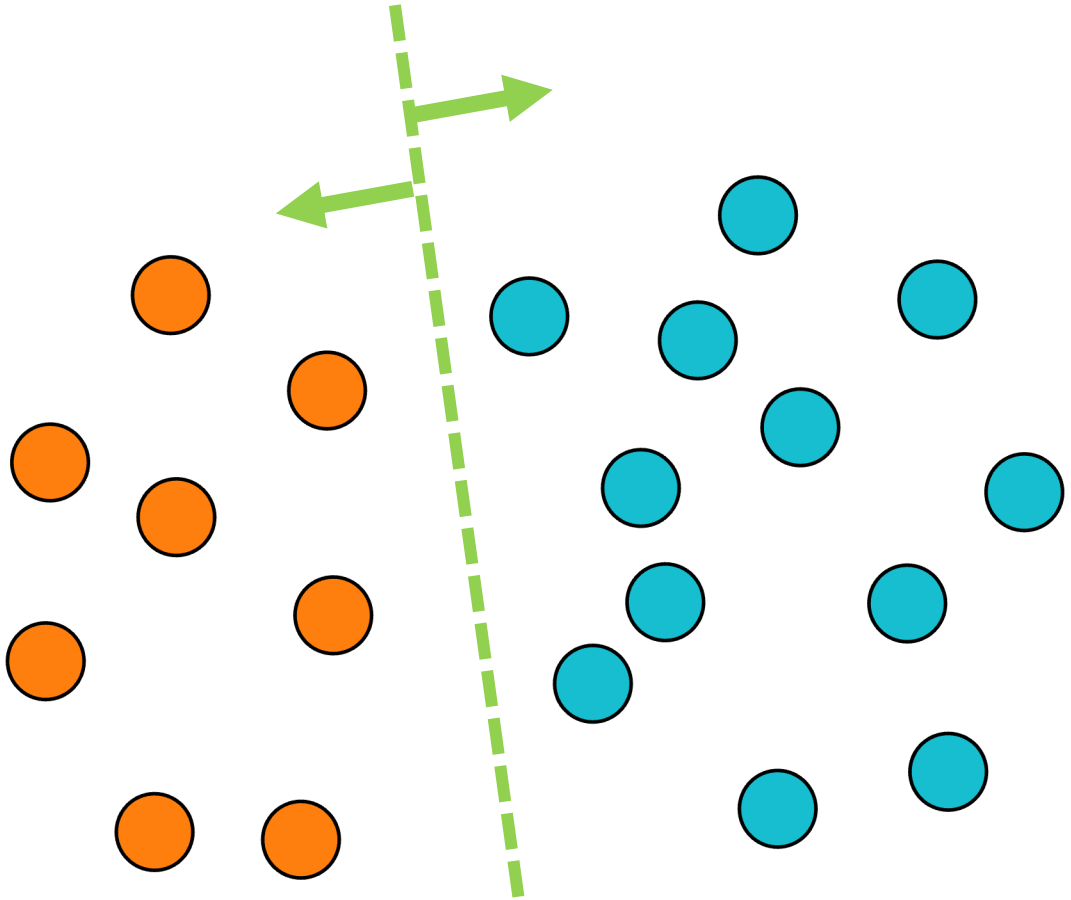
- Nearest Neighbors
  - Store every image
  - Find nearest neighbors at test time, and assign same class



Linear Classification vs. Nearest Neighbors

- Nearest Neighbors
  - Store every image
  - Find nearest neighbors at test time, and assign same class

- Linear Classifier
  - Store hyperplanes that best separate different classes
  - We can compute continuous class score by calculating (signed) distance from hyperplane



We can interpret this as a linear "score function" for each class.

Score functions

class scores

Parametric Approach

image parameters

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

10 numbers, indicating class scores

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

Parametric Approach: Linear Classifier

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

3072x1

10x1

10x3072

10 numbers, indicating class scores

[32x32x3] $= 3072$
array of numbers 0...1

parameters, or “weights”

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

Parametric Approach: Linear Classifier

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

3072x1

10x1

10x3072

10x1

10 numbers, indicating class scores

[32x32x3] $= 3072$
array of numbers 0...1

parameters, or “weights”

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

Linear Classifier

define a **score function**

data (image)

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

class scores

"weights"

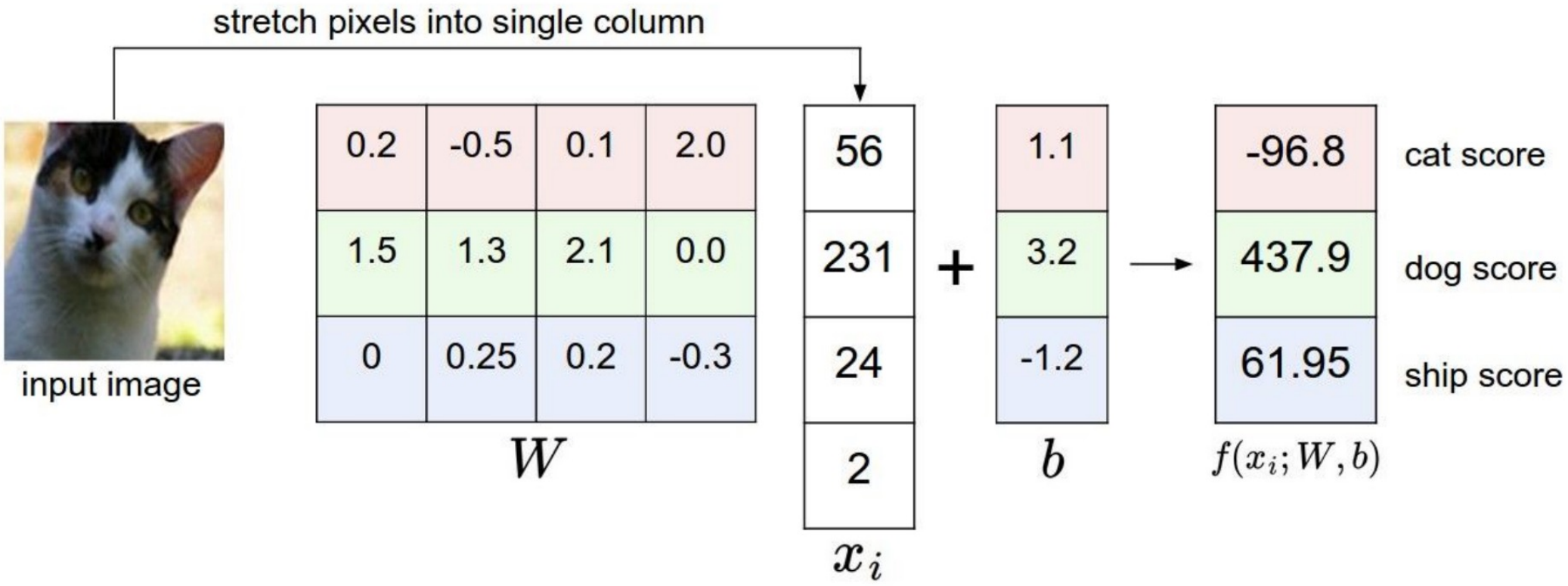
"bias vector"

"parameters"

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

Interpretation: Algebraic

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



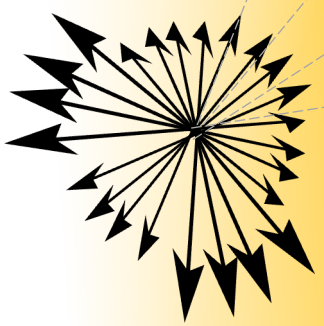
Slide adapted from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

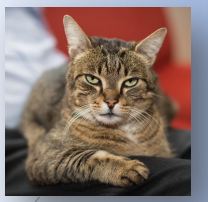
Interpretation: Geometric

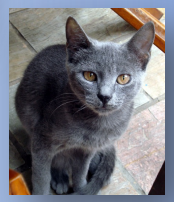
- Parameters define a hyperplane for each class:

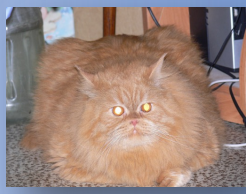
$$f(x_i, W, b) = Wx_i + b$$

- We can think of each class score as defining a distribution that is proportional to distance from the corresponding hyperplane









The Space of All Images

Interpretation: Template matching

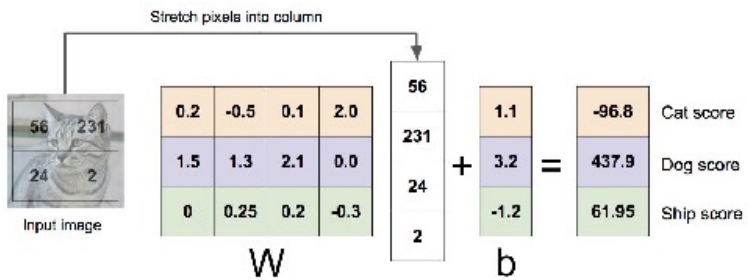
- We can think of the rows in W as templates for each class

Rows of *W* in $f(x_i, W, b) = Wx_i + b$

Linear Classifier: Three Viewpoints

Algebraic Viewpoint

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

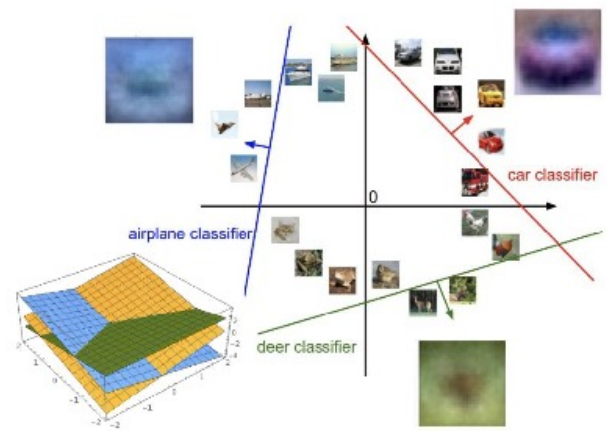


Visual Viewpoint

One template per class

Geometric Viewpoint

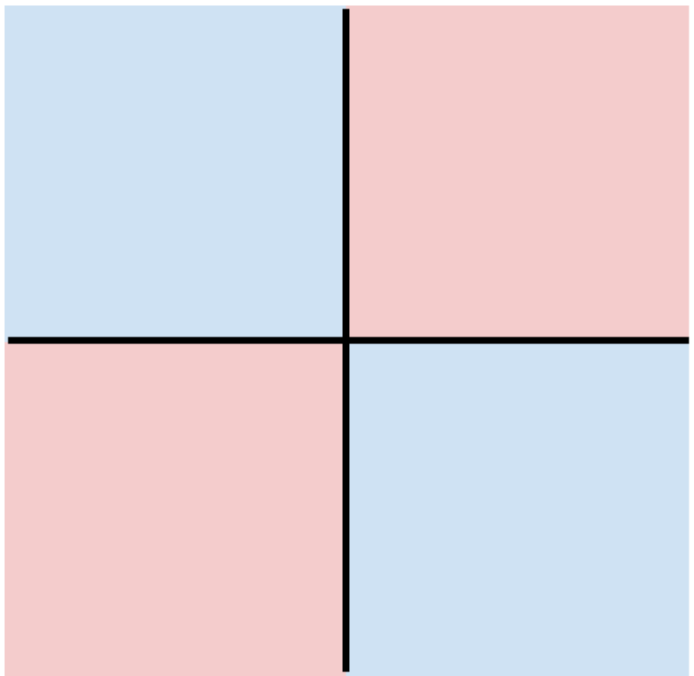
Hyperplanes cutting up space



Hard Cases for a Linear Classifier

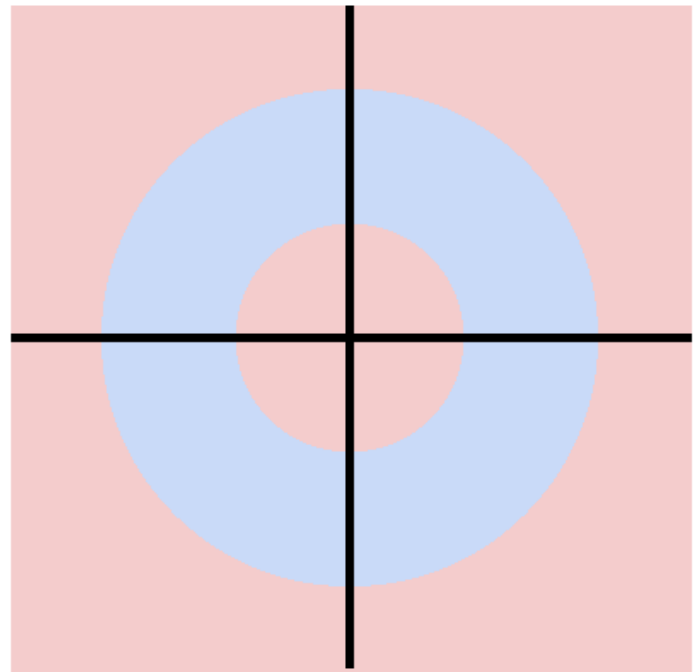
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



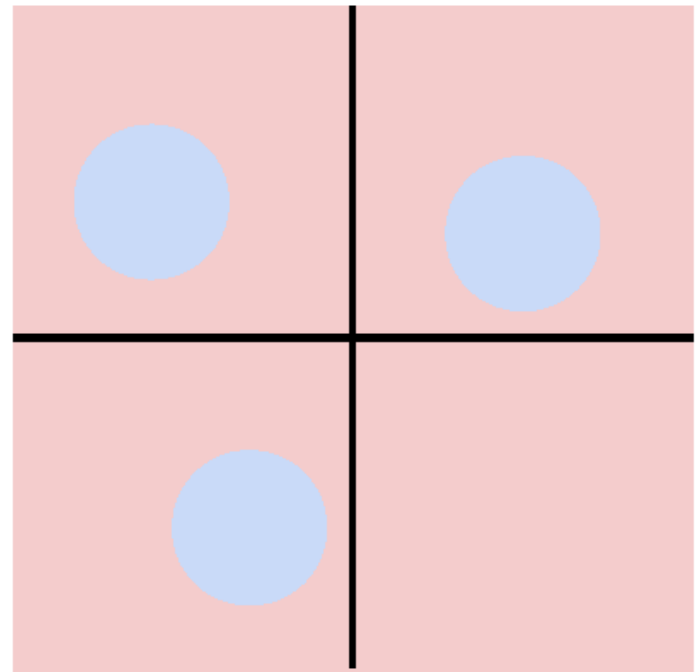
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

Class 2:
Everything else



So far: Defined a (linear) score function $f(x,W) = Wx + b$

Example class scores for 3 images for some W:

How can we tell whether this W is good or bad?



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

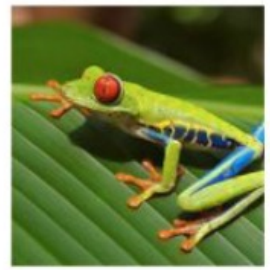
Recap

- Learning methods
  - k-Nearest Neighbors
  - **Linear classification**

- Classifier outputs a **score function** giving a score to each class
- How do we define how good a classifier is based on the training data? (Spoiler: define a *loss function*)

Linear classification





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)**

Loss functions

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:

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

Simpler example: binary classification

- Two classes (e.g., “cat” and “not cat”)
  - AKA “positive” and “negative” classes

cat

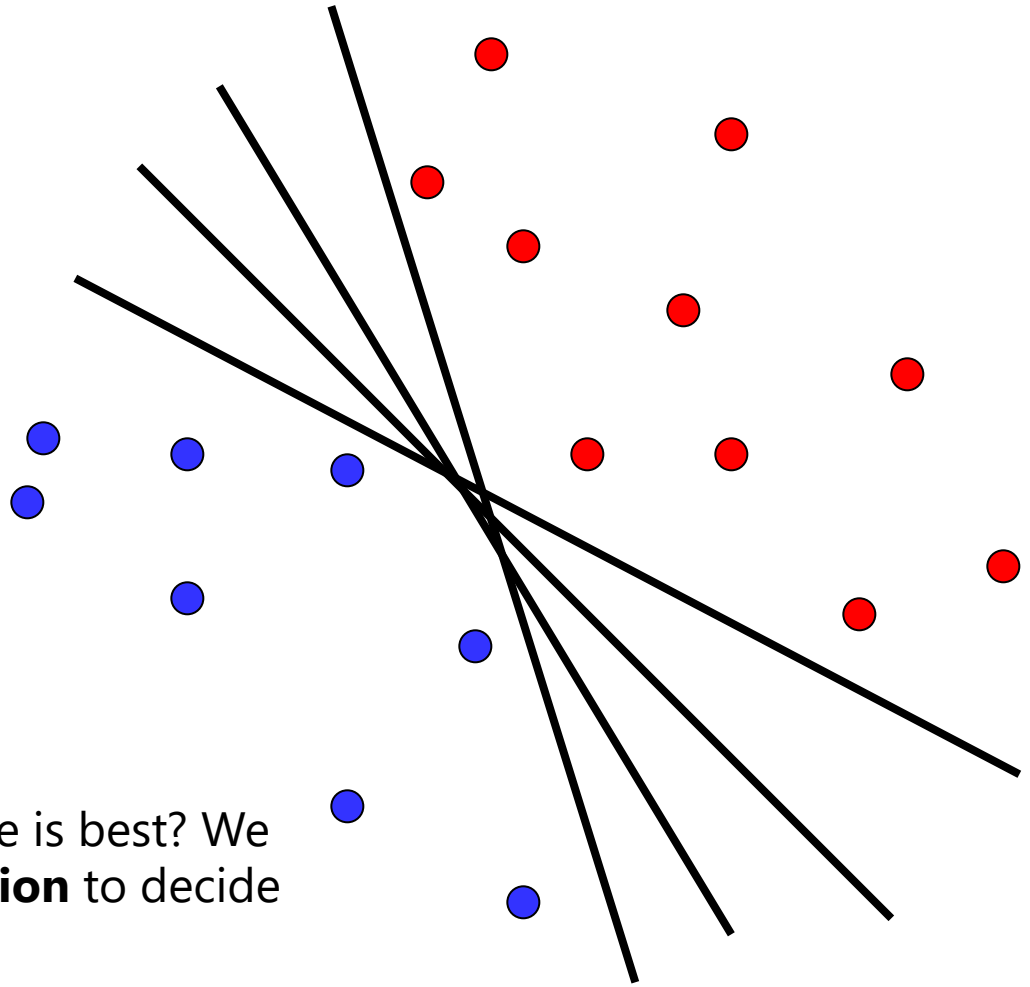
not cat

Linear classifiers

- Find linear function (*hyperplane*) to separate positive and negative examples

$$\mathbf{x}_i \text{ positive}: \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 0$$

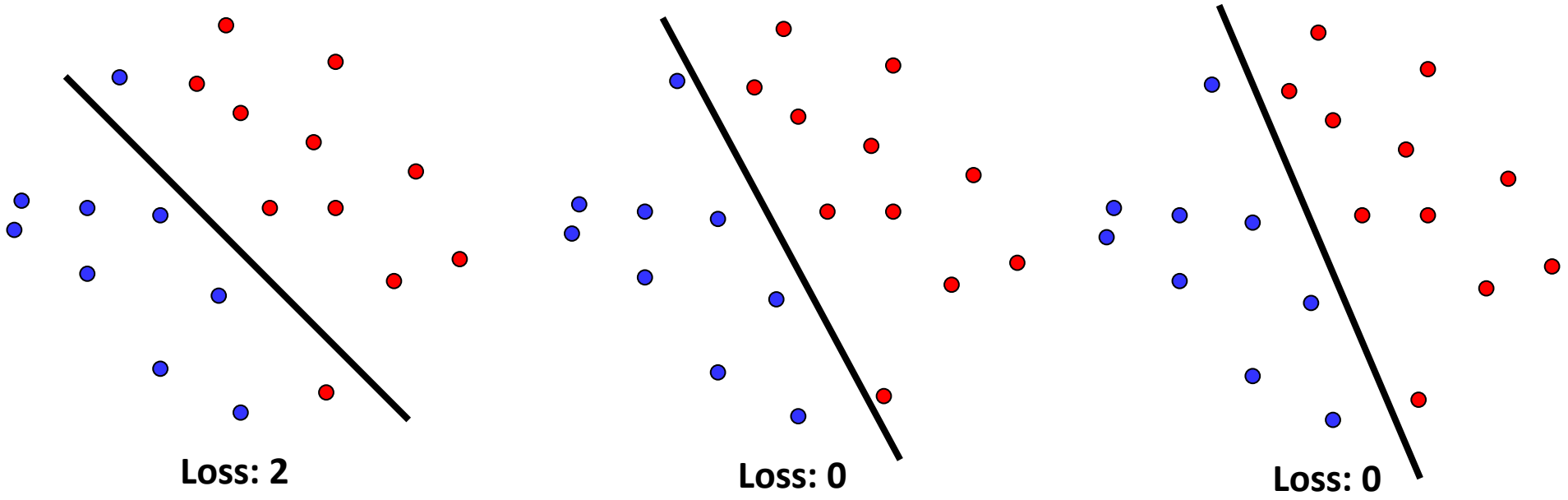
$$\mathbf{x}_i \text{ negative}: \quad \mathbf{x}_i \cdot \mathbf{w} + b < 0$$



Which hyperplane is best? We need a **loss function** to decide

What is a good loss function?

- One possibility: Number of misclassified examples
  - Problems: discrete, can't break ties
  - We want the loss to lead to *good generalization*
  - We want the loss to work for more than 2 classes



Softmax classifier

- Interpret Scores as unnormalized log probabilities of classes

$$f(x_i, W) = Wx_i$$ (score function)

$$\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$$ softmax function

Squashes values into *probabilities* ranging from 0 to 1

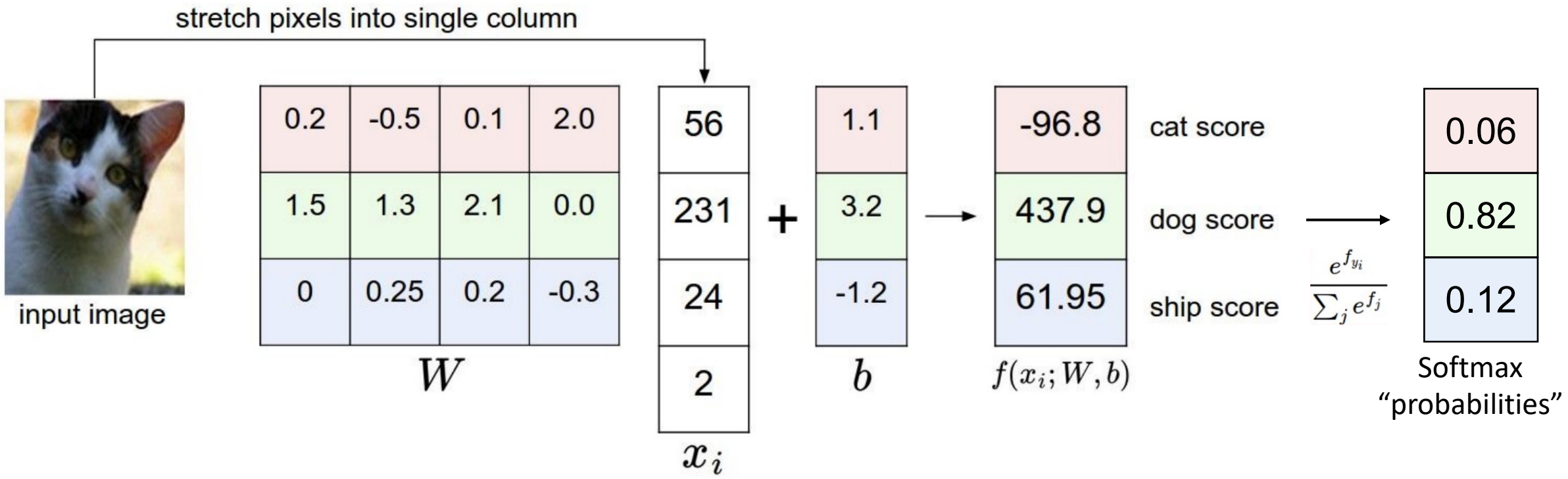
$$P(y_i \mid x_i; W)$$

Example with three classes:

$$[1, -2, 0] \rightarrow [e^1, e^{-2}, e^0] = [2.71, 0.14, 1] \rightarrow [0.7, 0.04, 0.26]$$

Softmax classifier

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



Cross-entropy loss

$$f(x_i, W) = Wx_i$$ (score function)

Cross-entropy loss

$$f(x_i, W) = Wx_i$$ (score function)

f_{y_i} : score of correct class

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \qquad L_i = -f_{y_i} + \log\sum_j e^{f_j}$$

We call L_i *cross-entropy loss*

Cross-entropy loss

$$f(x_i, W) = Wx_i$$ (score function)

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \qquad L_i = -f_{y_i} + \log\sum_j e^{f_j}$$

We call L_i *cross-entropy loss*

$P(y_i \mid x_i; W)$

i.e. we're minimizing the negative log likelihood.

Losses

- Cross-entropy loss is just one possible loss function
  - One nice property is that it reinterprets scores as probabilities, which have a natural meaning

- SVM (max-margin) loss functions also used to be popular
  - But currently, cross-entropy is the most common classification loss

Summary

- Have score function and loss function
  - Currently, score function is based on linear classifier
  - Next, will generalize to convolutional neural networks
- Find W and b to minimize loss

$$L = \frac{1}{N}\sum_i -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) + \lambda \sum_k \sum_l W_{k,l}^2$$

Average of cross-entropy loss
over all training examples

Regularization term

What's Still Hard?

- Fine-grain classification
  - How do we distinguish between more subtle class differences?

Animal->Bird->Oriole…

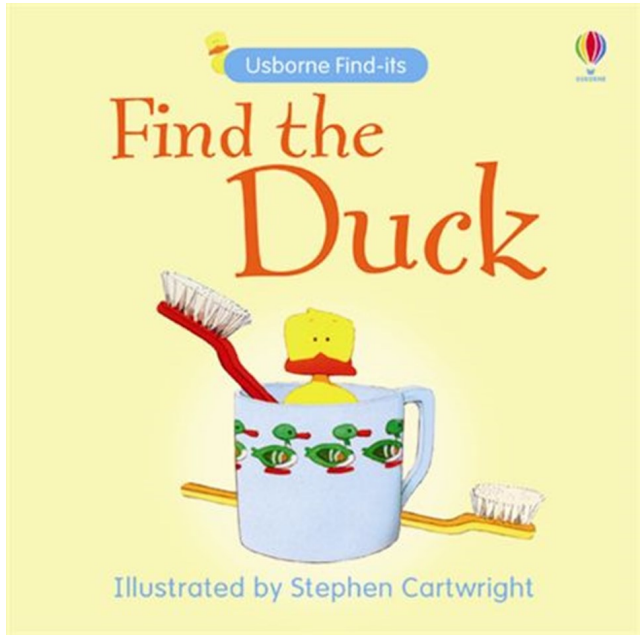
Baltimore Oriole

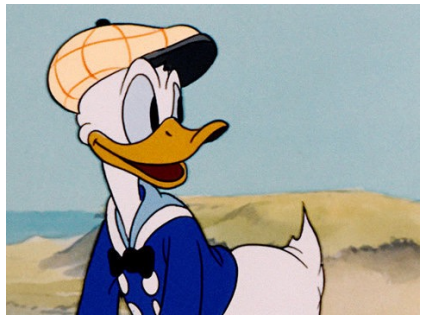
Hooded Oriole

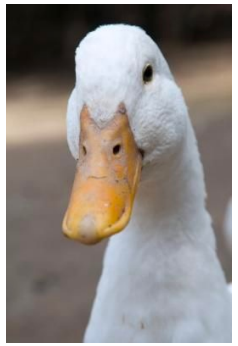
Scott Oriole

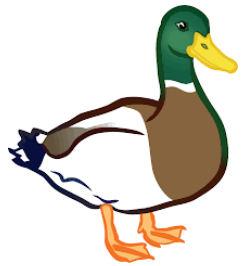
What's Still Hard?

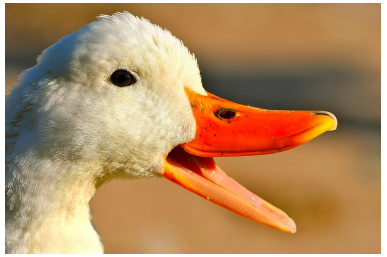
- Few shot learning
  - How do we generalize from only a small number of examples?

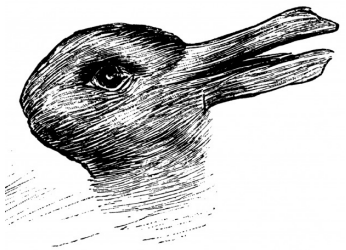












Slide Credits

- CS5670, Introduction to Computer Vision, **Cornell Tech, by Noah Snavely.**
- **CS 543 Computer Vision, by Stevlana Lazebnik, UIUC.**
- **EECS 442 Computer Vision, by Justin Johnson & David Fouhey, U Michigan.**