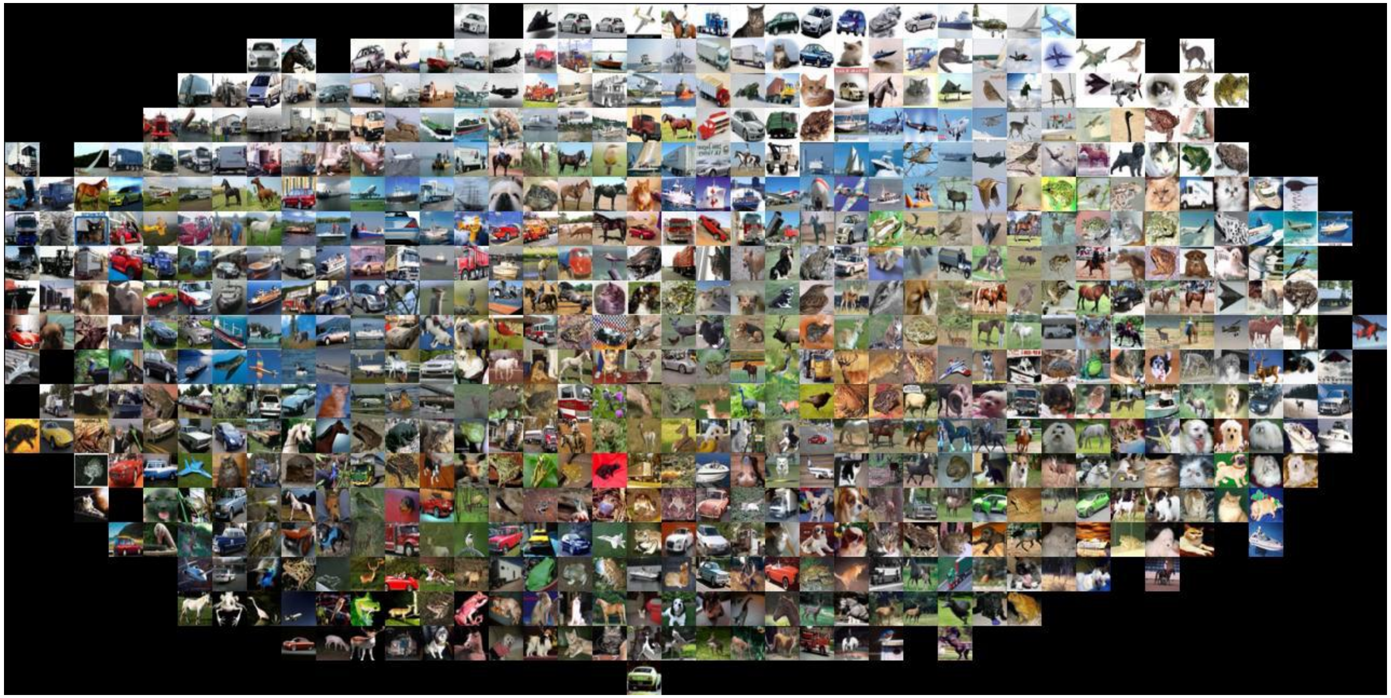
CS5670: Computer Vision

Noah Snavely

Image Classification



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

Announcements

- Project 4 to be released shortly
- Vote for Project 3 artifacts!
  - Deadline: midnight tonight

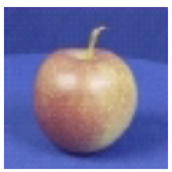
Today

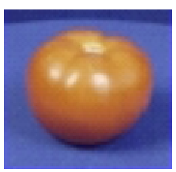
- Image classification pipeline
- Training, validation, testing
- Nearest neighbor classification
- Linear classification
- Score function and loss function

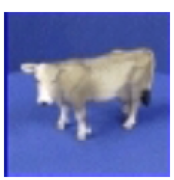
- Building up to CNNs for learning
  - Next 2-4 lectures on deep learning

Image Classification:
A core task in Computer Vision

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

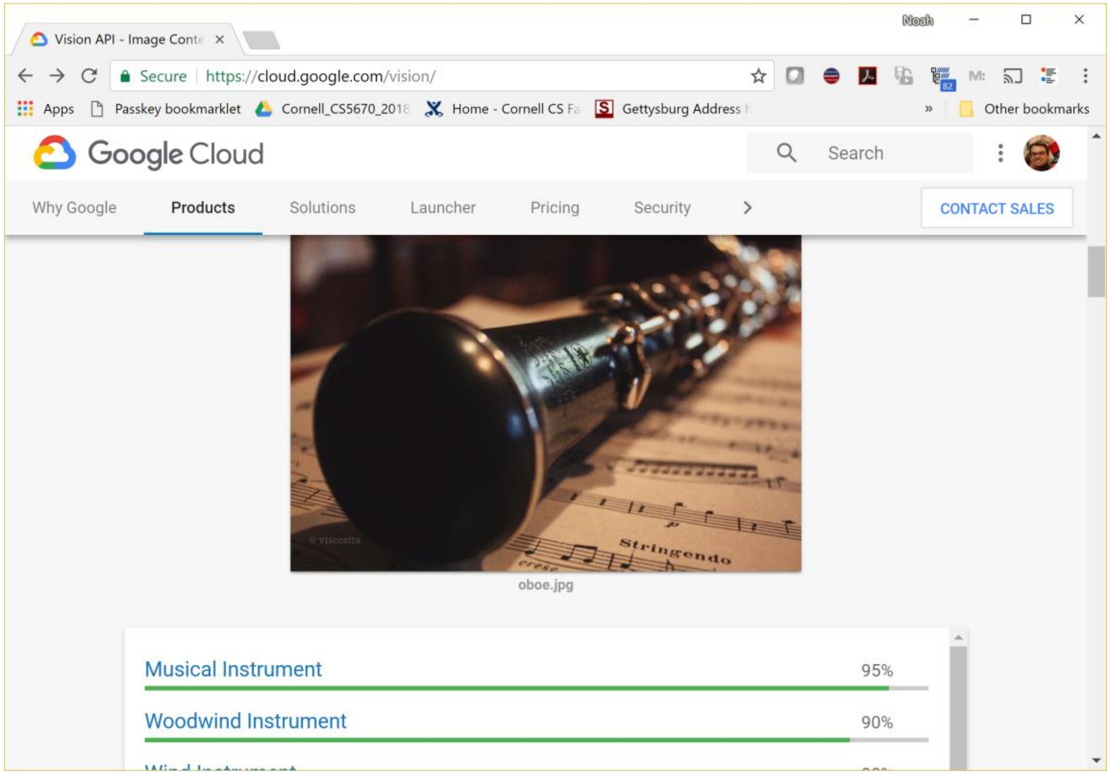
f() = “apple”

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/

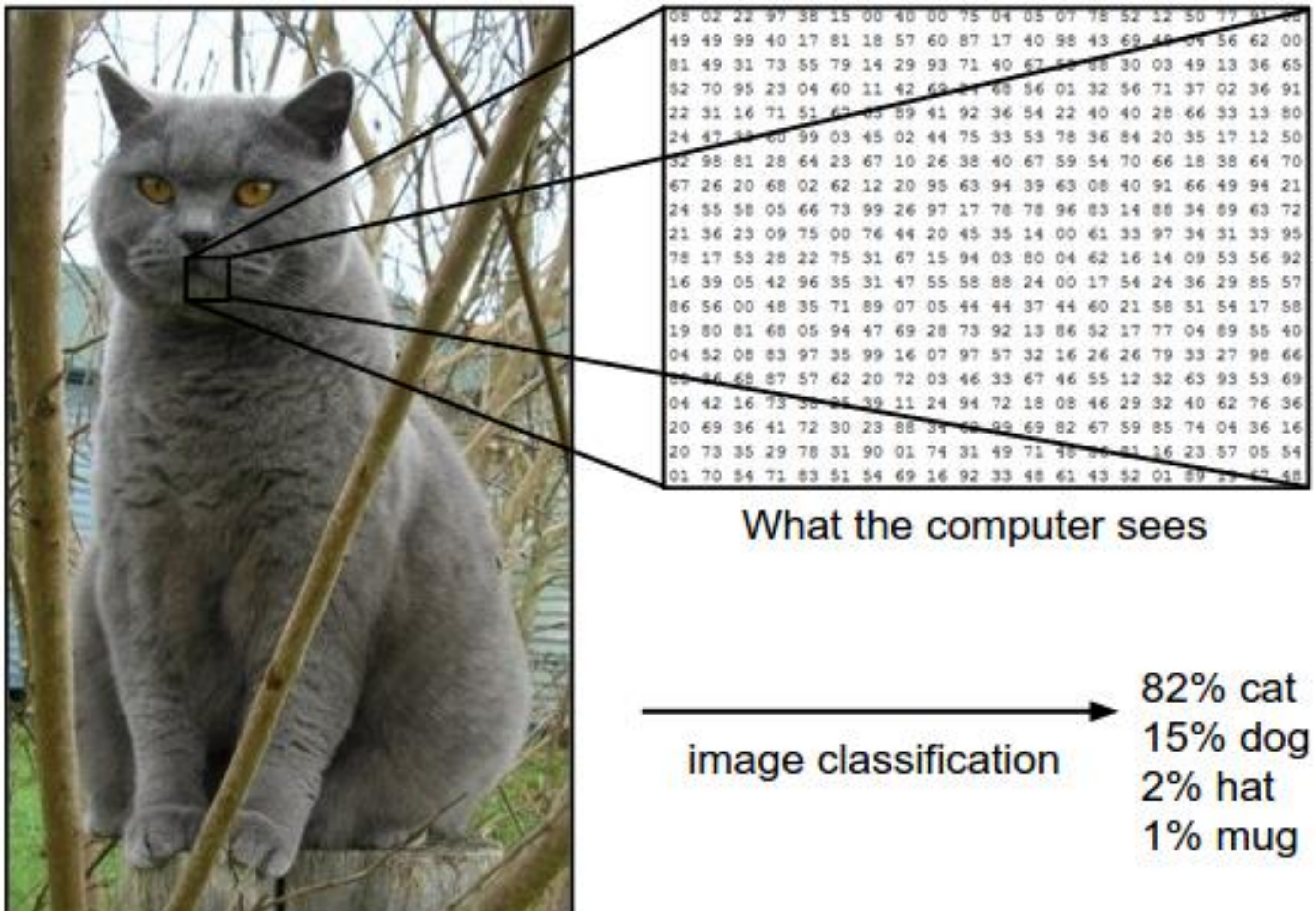
…

Image Classification

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

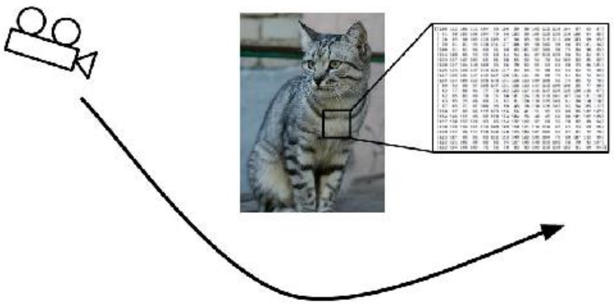
→ cat

Image Classification: Problem



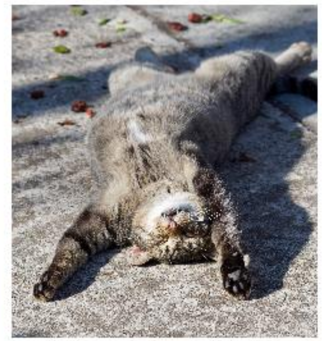
Recall from last time: Challenges of recognition

Viewpoint



Illumination

Deformation



Occlusion

Clutter

Intraclass Variation

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 Yeung

Data-driven approach

- Collect a database of images with labels
- Use ML to train an image classifier
- Evaluate the classifier on test images

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

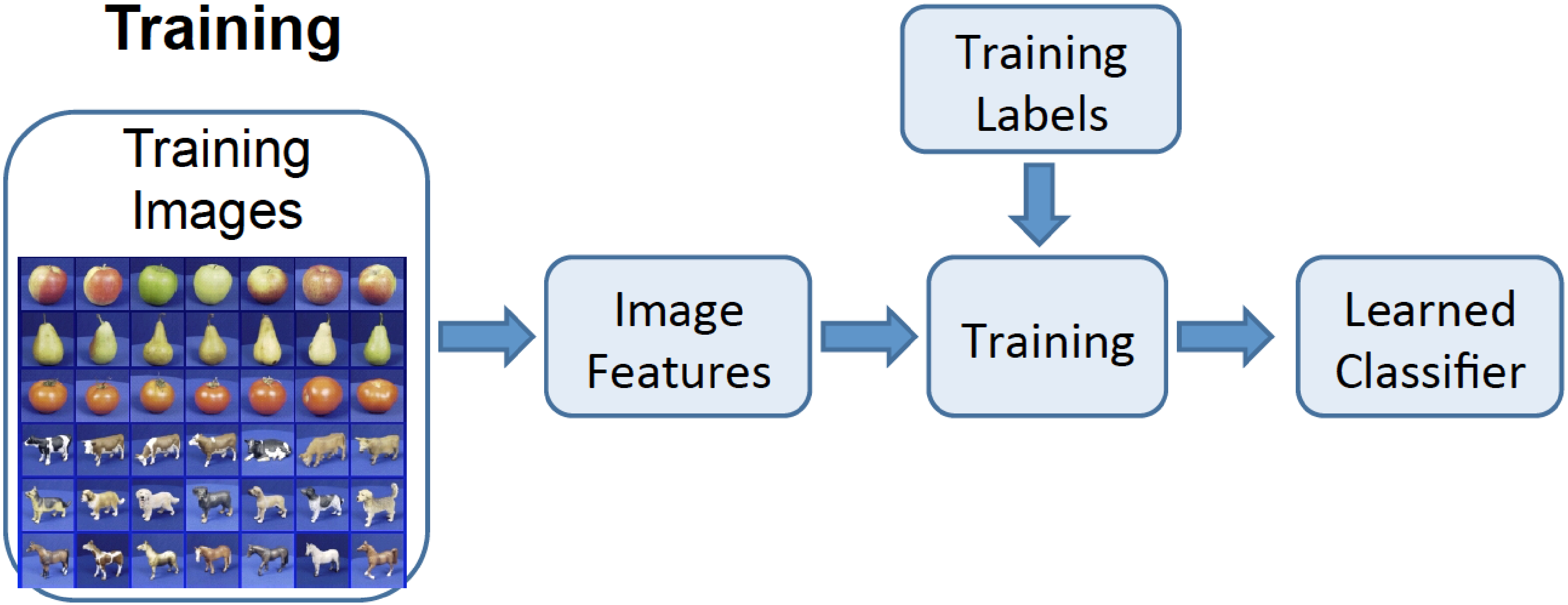
Data-driven approach

- Collect a database of images with labels
- Use ML to train an image classifier
- Evaluate the classifier on test images

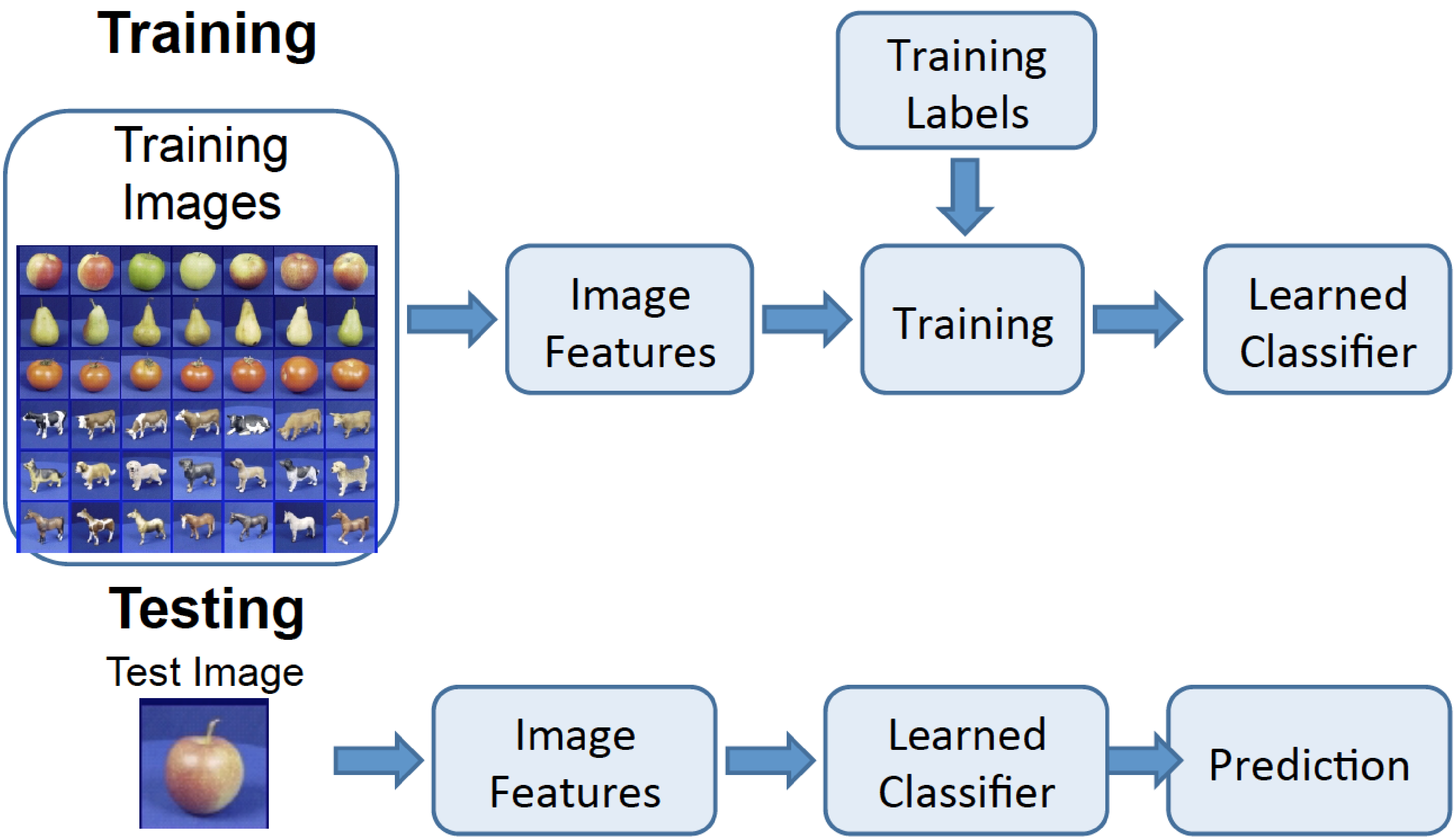
```
def train(train_images, train_labels):
    # build a model of images -> labels

def predict(image):
    # evaluate the model on the image
    return class_label
```

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

Classifiers

- Nearest Neighbor
- kNN (“k-Nearest Neighbors”)
- Linear Classifier
- SVM (Support Vector Machine)
- …

First: 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, each image is tiny: 32x32
10,000 test images.

CIFAR-10 and NN results

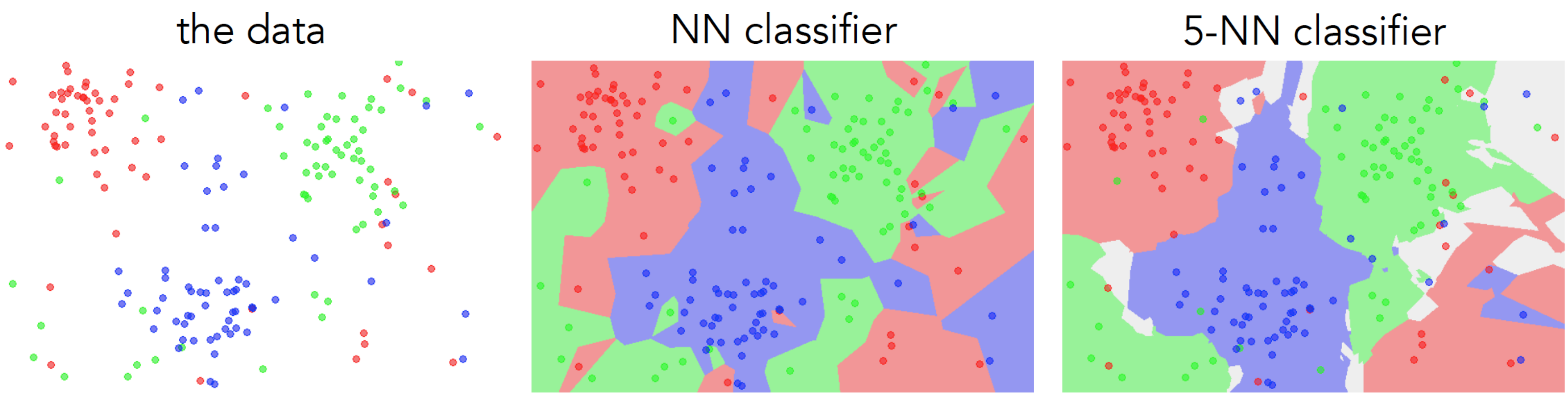
Example dataset: **CIFAR-10**
10 labels
50,000 training images
10,000 test images.

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

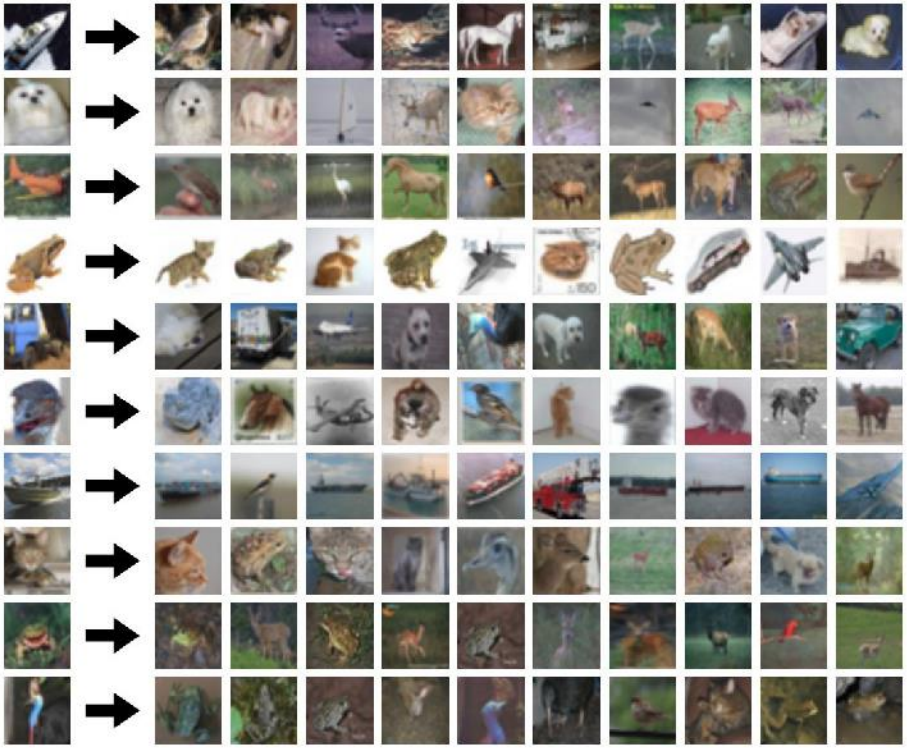
Slides from Andrej Karpathy and Fei-Fei Li
http://vision.stanford.edu/teaching/cs231n/

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?

How to find the most similar training image? What is the distance metric?

L1 distance: $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$

Where I_1 denotes image 1, and p denotes each pixel

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

=

pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

$\rightarrow$ 456

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

Choice of distance metric

- Hyperparameter

L1 (Manhattan) distance

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

L2 (Euclidean) distance

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

- Two most commonly used special cases of p-norm

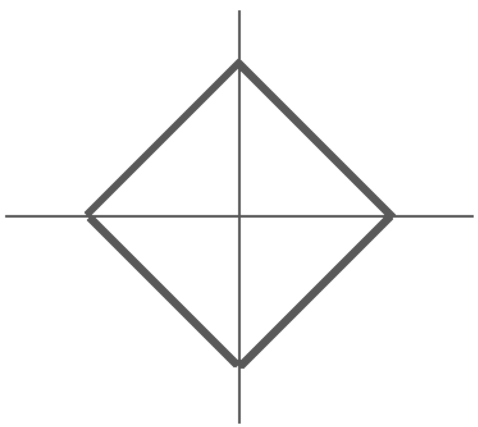
$$||x||_p = \left(|x_1|^p + \cdots + |x_n|^p\right)^{\frac{1}{p}} \qquad p \geq 1, x \in \mathbb{R}^n$$

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

K-Nearest Neighbors: Distance Metric

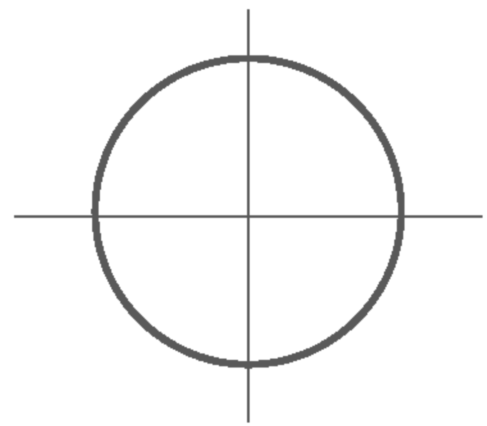
L1 (Manhattan) distance

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



L2 (Euclidean) distance

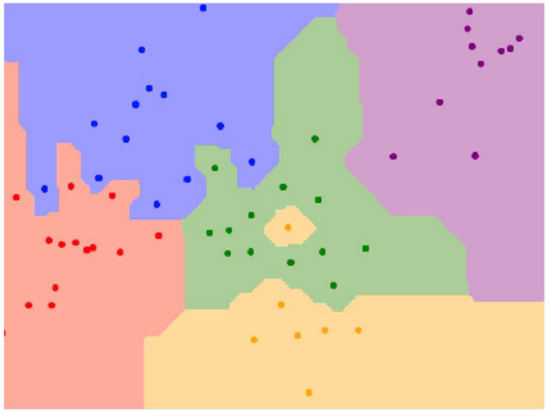
$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$



K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

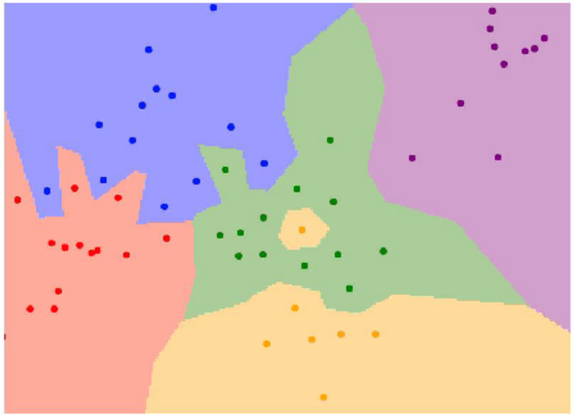
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



K = 1

L2 (Euclidean) distance

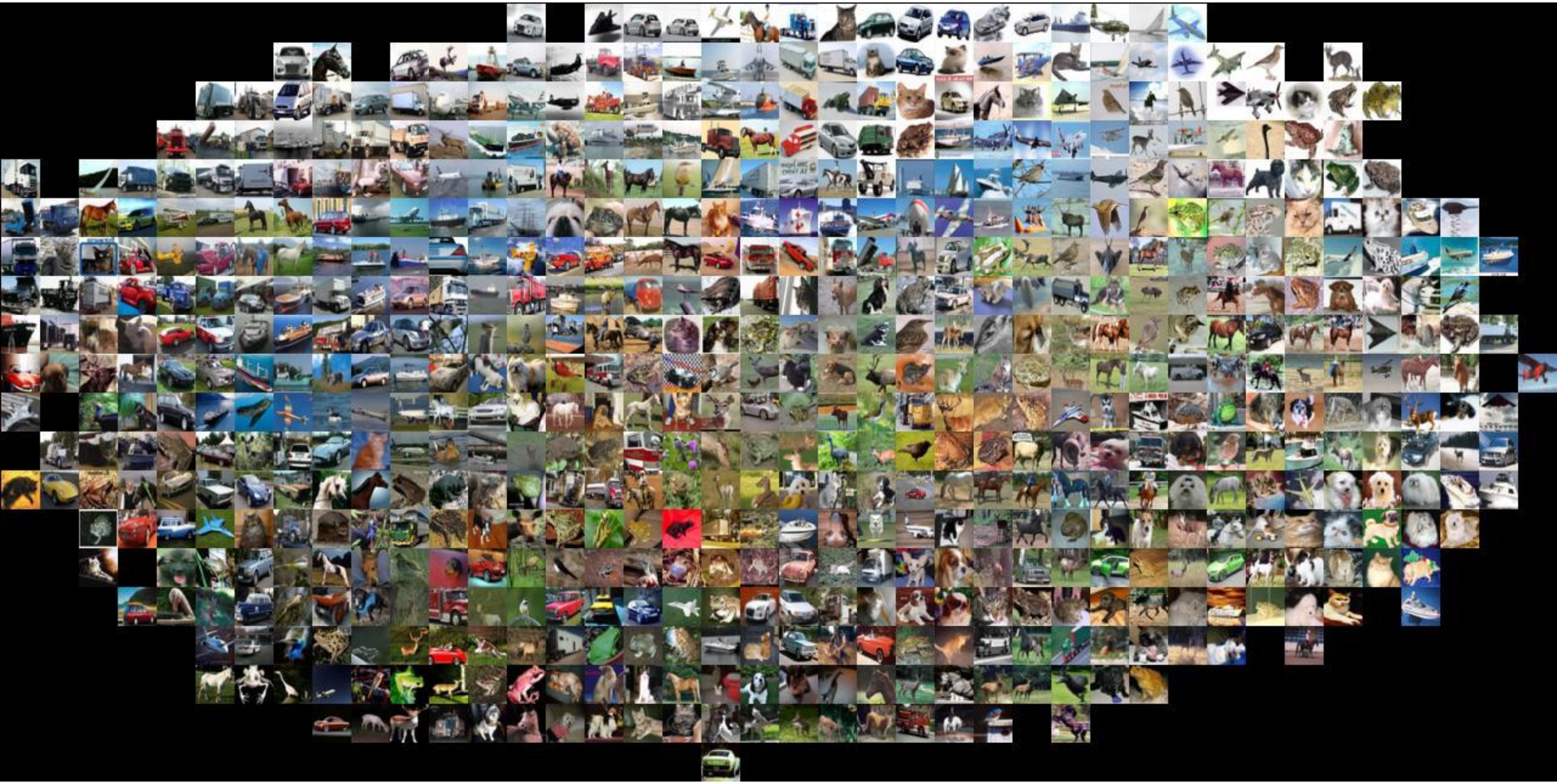
$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$



K = 1

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

Visualization: L2 distance



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

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

Your Dataset

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

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

Your Dataset

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

train	test

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

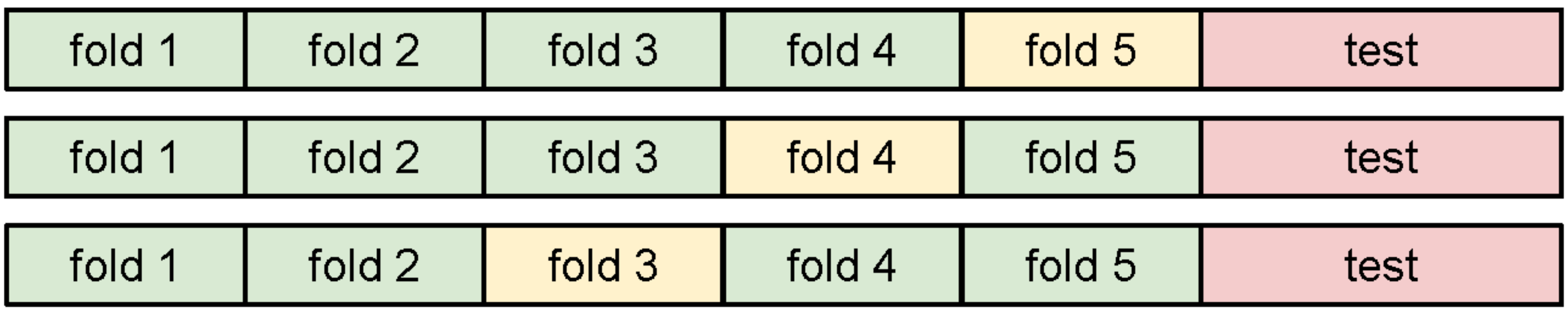
BAD: No idea how algorithm will perform on new data

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

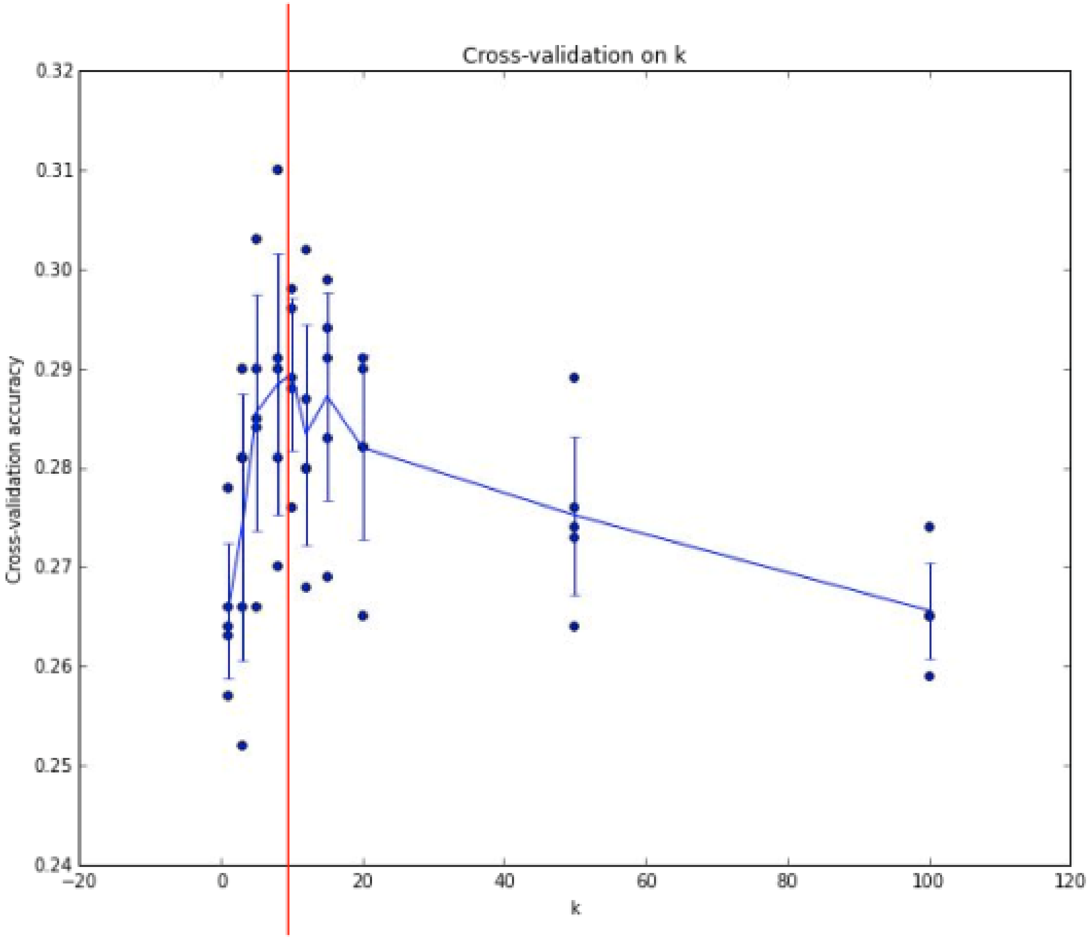
Better!

Setting Hyperparameters

Idea #4: **Cross-Validation**: Split data into **folds**,
try each fold as validation and average the results



Useful for small datasets, but not used too frequently in deep learning



Example of
5-fold cross-validation
for the value of **k.**

Each point: single
outcome.

The line goes
through the mean, bars
indicated standard
deviation

(Seems that k ~= 7 works best
for this data)

Recap: How to pick hyperparameters?

- Methodology
  - Train and test
  - Train, validate, test

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

kNN -- Complexity and Storage

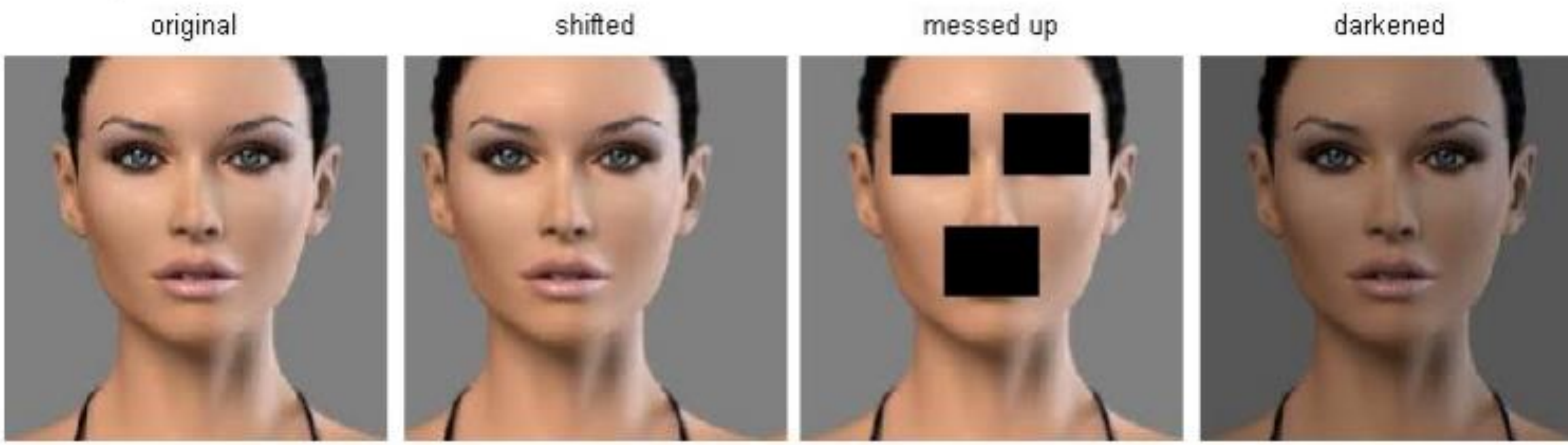
- N training images, M test images

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

- Hmm…
  - Normally need the opposite
  - Slow training (ok), fast testing (necessary)

k-Nearest Neighbor on images **never used.**

- 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)

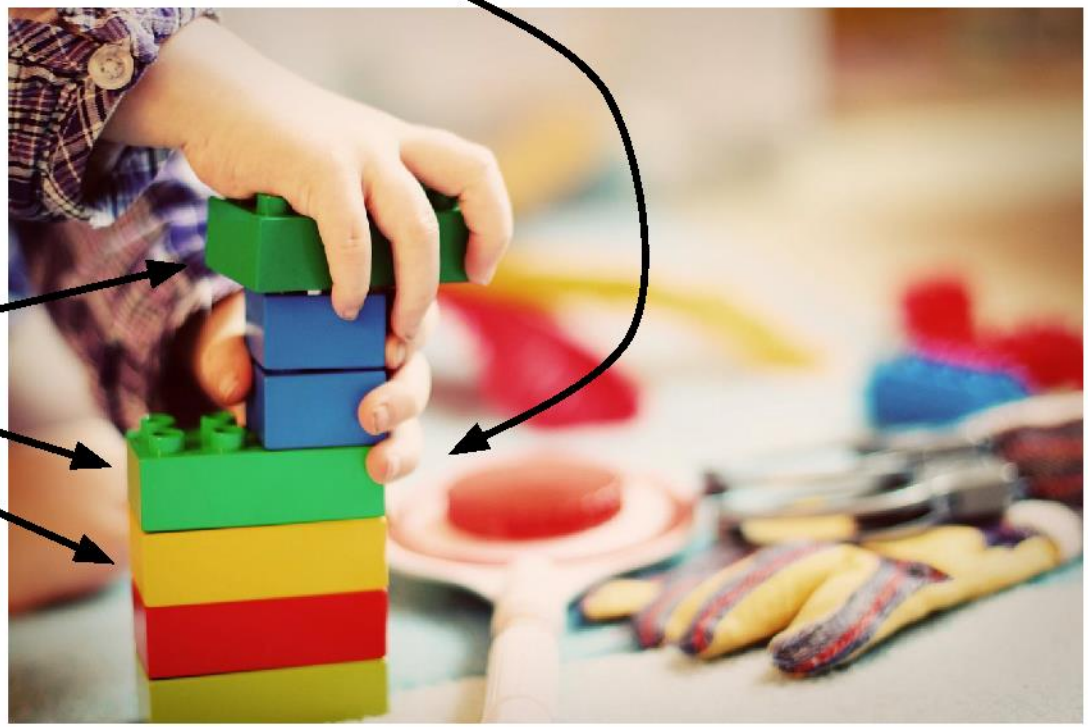
k-Nearest Neighbors: Summary

- 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!

Linear classifiers

Neural Network

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

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

image parameters

$f(\mathbf{x}, \mathbf{W})$

10 numbers,
indicating class
scores

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

$$\boxed{f(x,W)} = \boxed{W}\boxed{x} \quad \text{3072x1} \qquad \boxed{(+b)} \text{ 10x1}$$

10x1 10x3072

10 numbers, indicating class scores

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

parameters, or “weights”

Linear Classifier

define a **score function**

data (image)

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

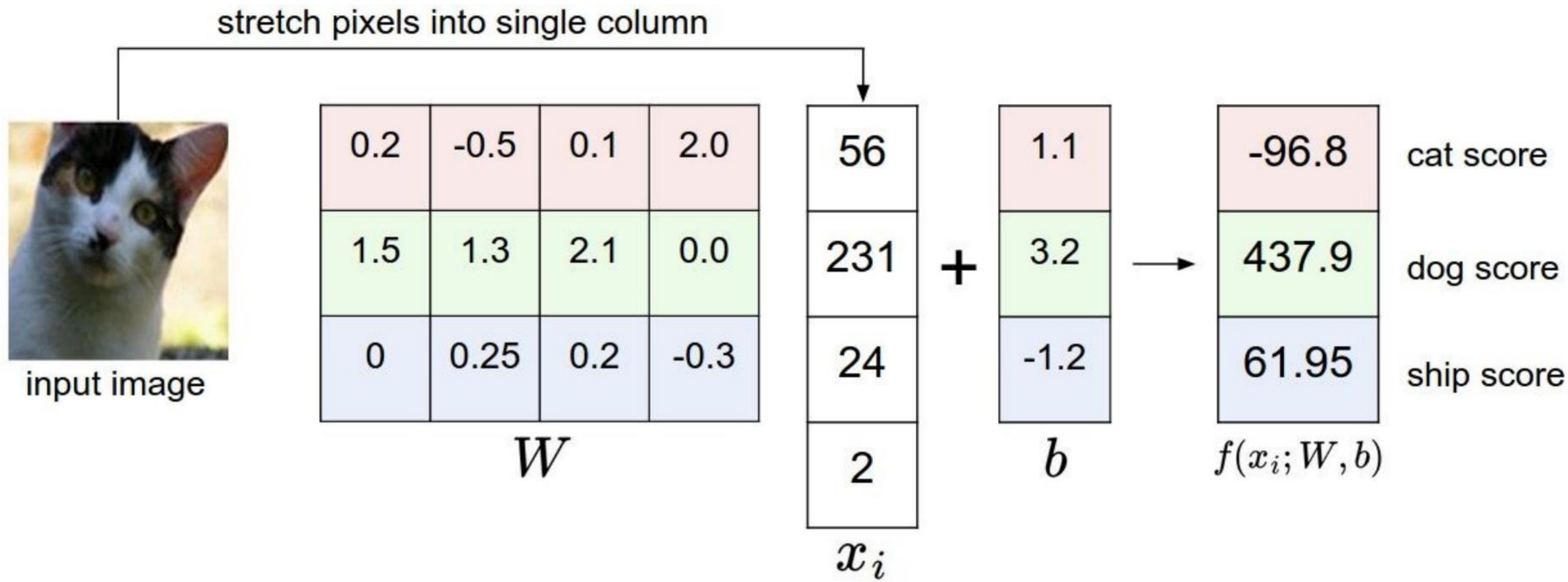
class scores

"weights"

"bias vector"

"parameters"

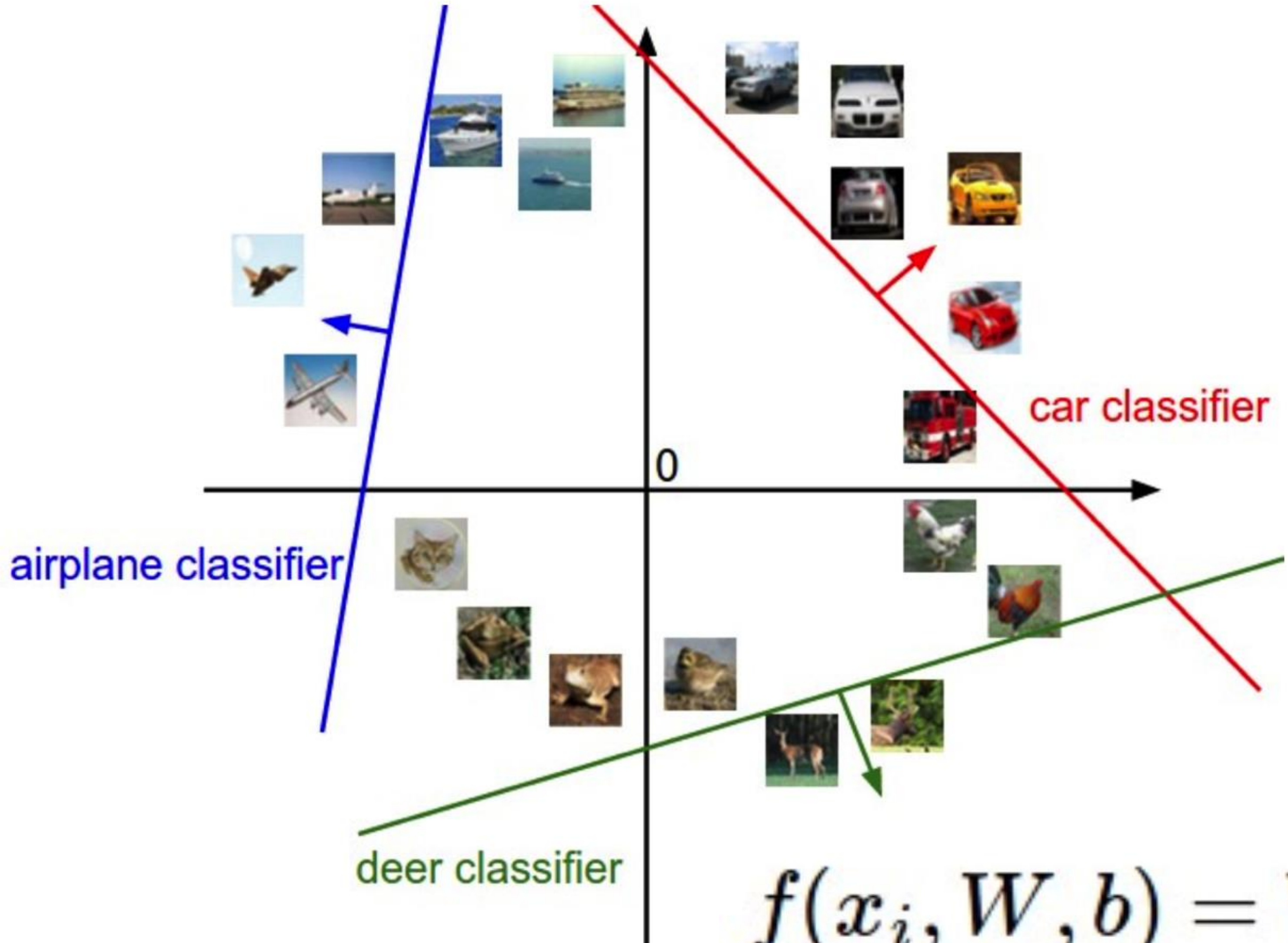
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



Interpretation: Template matching

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

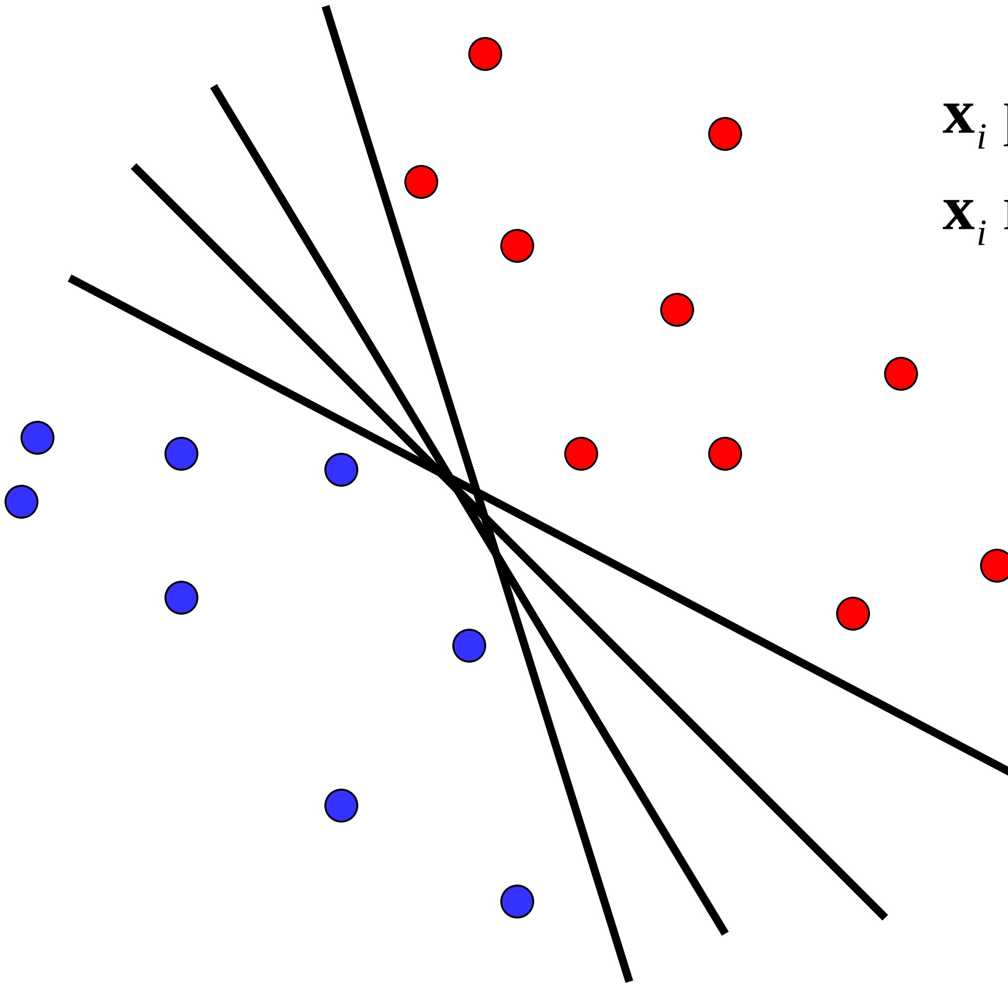
Geometric Interpretation



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

Linear classifiers

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



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

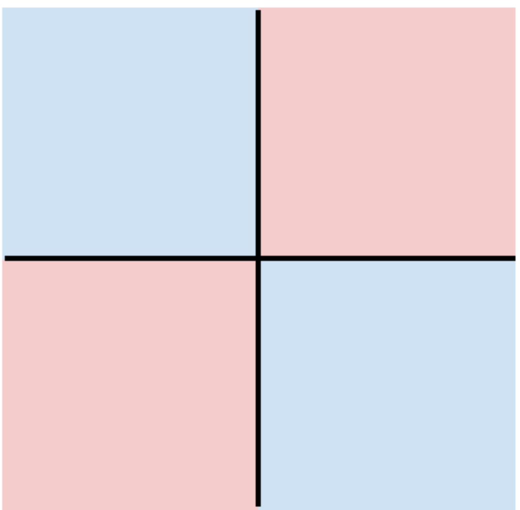
$$\mathbf{x}_i \text{ negative}: \quad \mathbf{x}_i \cdot \mathbf{w} + b < 0$$

Which hyperplane is best?
We will come back to this later

Hard cases for a linear classifier

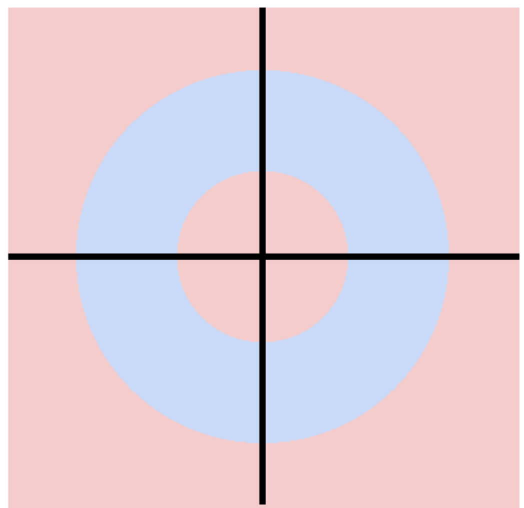
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



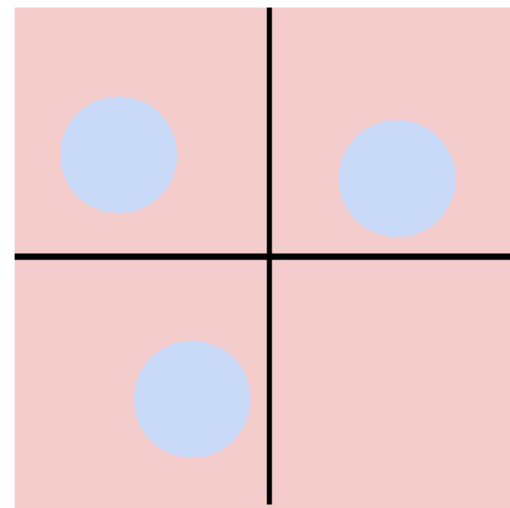
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

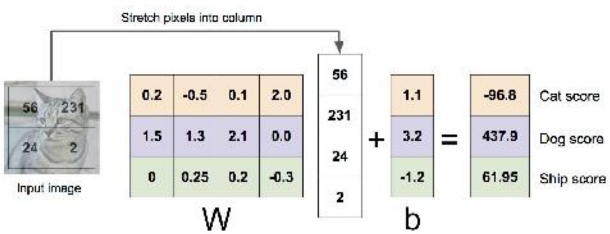
Class 2:
Everything else



Linear Classifier: Three Viewpoints

Algebraic Viewpoint

$f(x,W) = Wx$

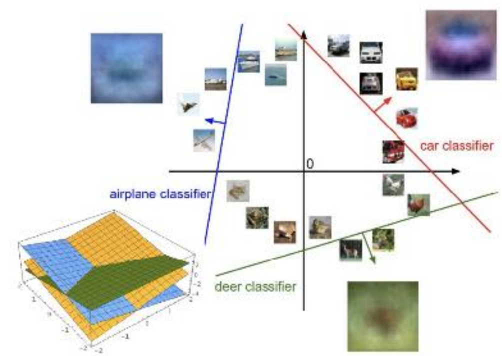


Visual Viewpoint

One template
per class

Geometric Viewpoint

Hyperplanes
cutting up space



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

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

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

Coming up:

- Loss function (quantifying what it means to have a "good" W)
- Optimization (start with random W and find a W that minimizes the loss)
- ConvNets! (tweak the functional form of f)