CSCI 497P/597P: Computer Vision

Scott Wehrwein

Image Classification and Recognition

Reading

- http://cs231n.github.io/classification/

Announcements

- P4 out tonight(?)

Goals

- Understand some of the reasons why image recognition is hard.
- Understand the standard ML pipeline for image classification problems:
  - Represent images as feature vectors
  - Learn a classifier function from labeled data
  - Classify novel images using the learned classifier
- Understand KNN classifier and why it doesn't work so well on images.
  - Performance
  - Curse of dimensionality
- Understand the importance of splitting data into train/val/test sets when developing algorithms and tuning hyperparameters.

Image classification

- Given an image, produce a label
- Label can be:
  - 0/1 or yes/no: *Binary classification*
  - one-of-k: *Multiclass classification*
  - 0/1 for each of k concepts: *Multilabel classification*

Image classification - Binary classification

Is this a dog?

Yes

Image classification - Multiclass classification

Which of these is it:
dog, cat or zebra?
Dog

Image classification - Multilabel classification

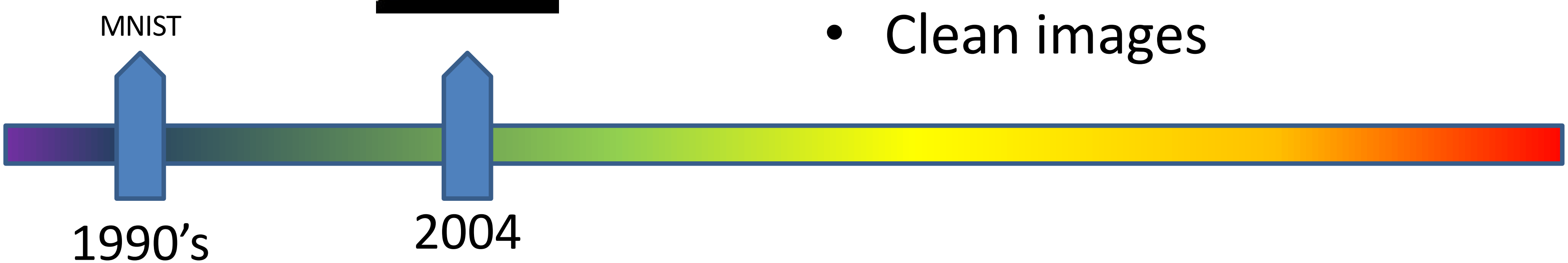
Is this a dog? Yes
Is this furry? Yes
Is this sitting down? Yes

A history of classification : MNIST

- 2D
- 10 classes
- 6000 examples per class

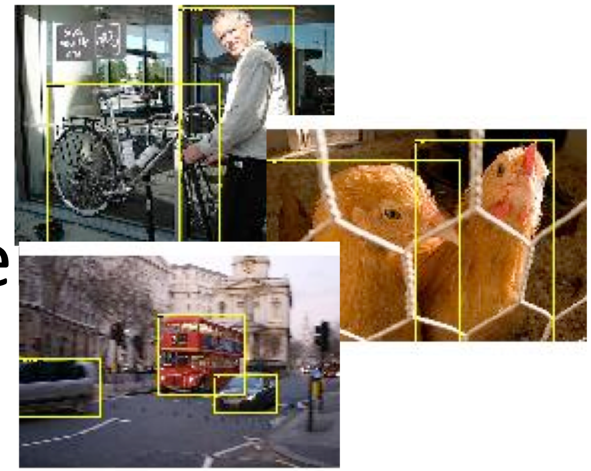
A history of classification : Caltech 101

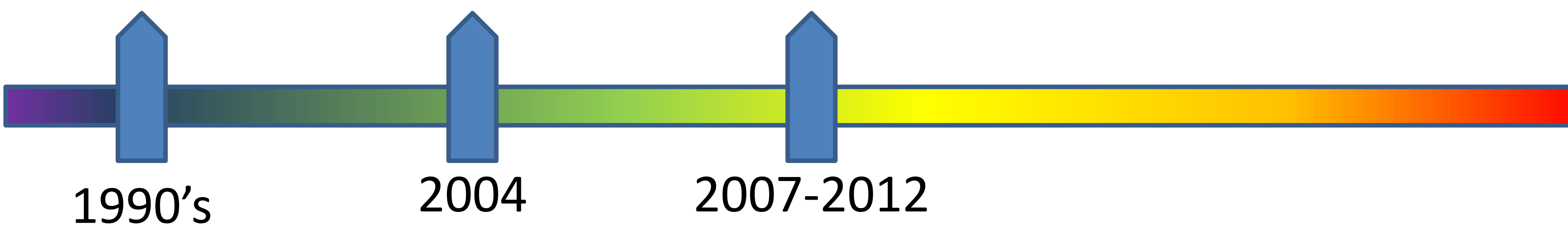
- 101 classes
- 10 classes
- 30 examples per class
- Strong category-specific biases
- Clean images



A history of classification: PASCAL VOC

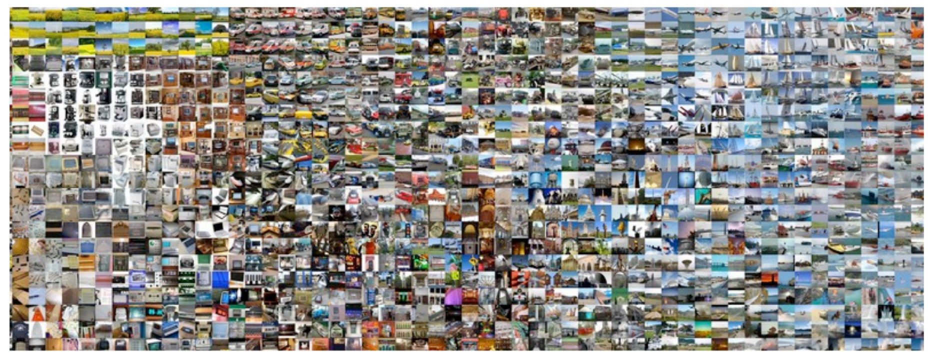
- 20 classes
- ~500 examples pe class
- Clutter, occlusion, natural scenes

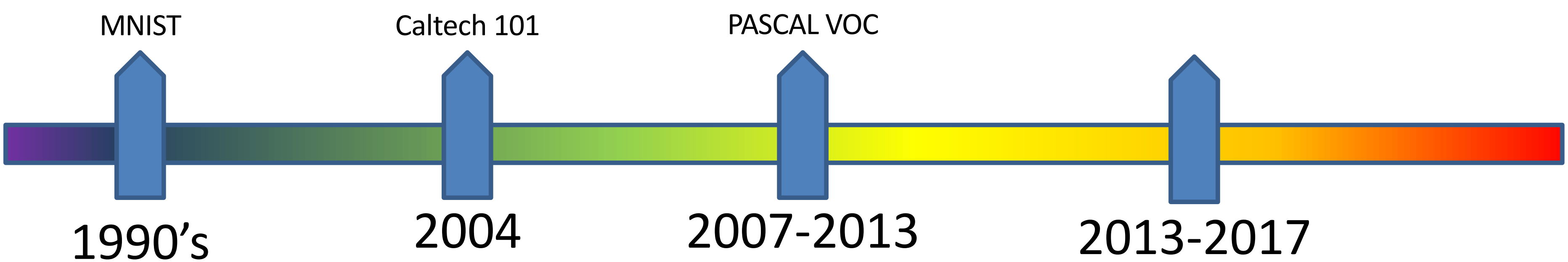




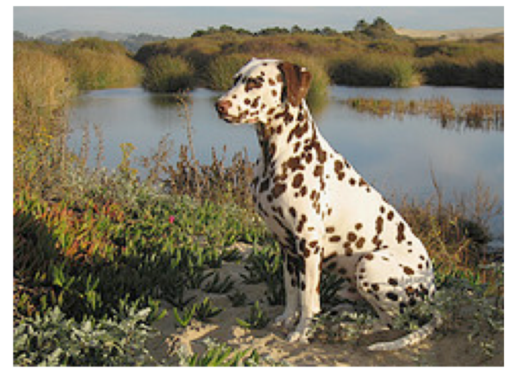
A history of classification: ImageNet

- 1000 classes
- ~1000 examples per class
- Mix of cluttered and clean images



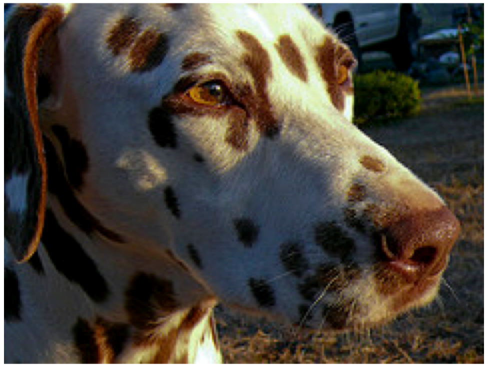


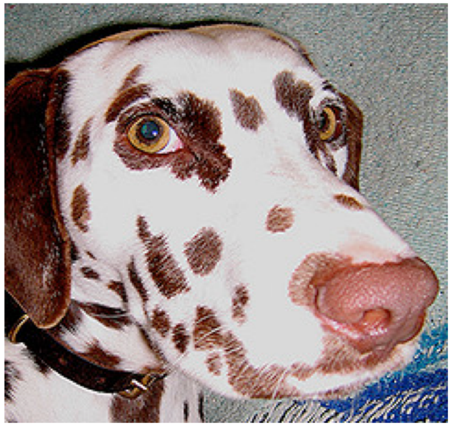
Why is recognition hard?



Pose variation

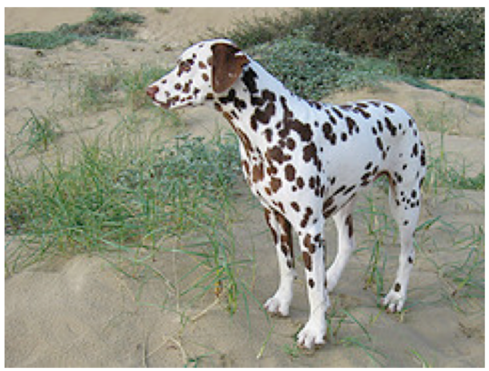
Why is recognition hard?





Lighting variation

Why is recognition hard?

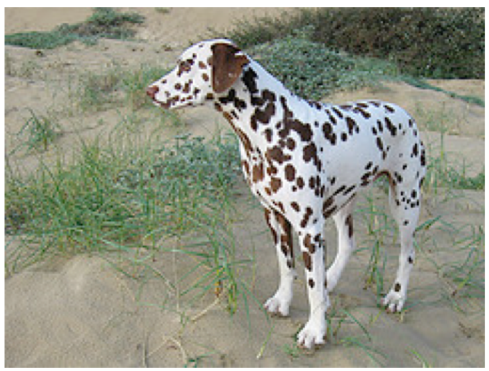


Scale variation

Why is recognition hard?

Clutter and occlusion

Why is recognition hard?



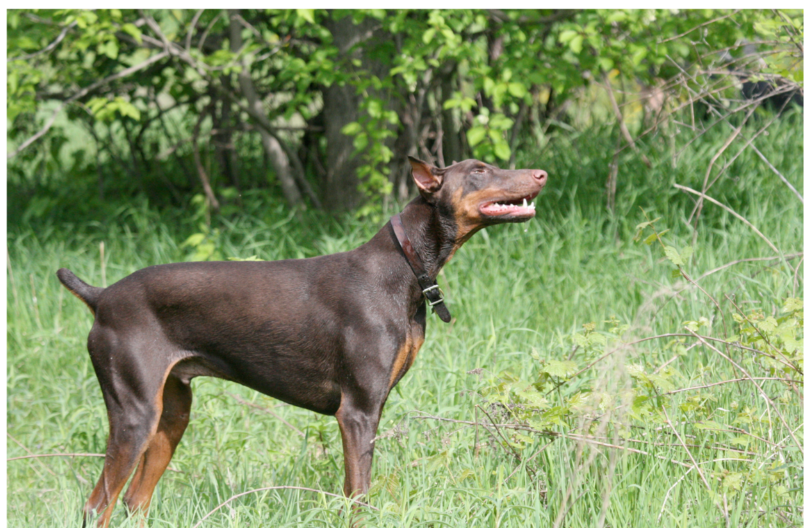
Intrinsic intra-class variation

Why is recognition hard?

Inter-class similarity

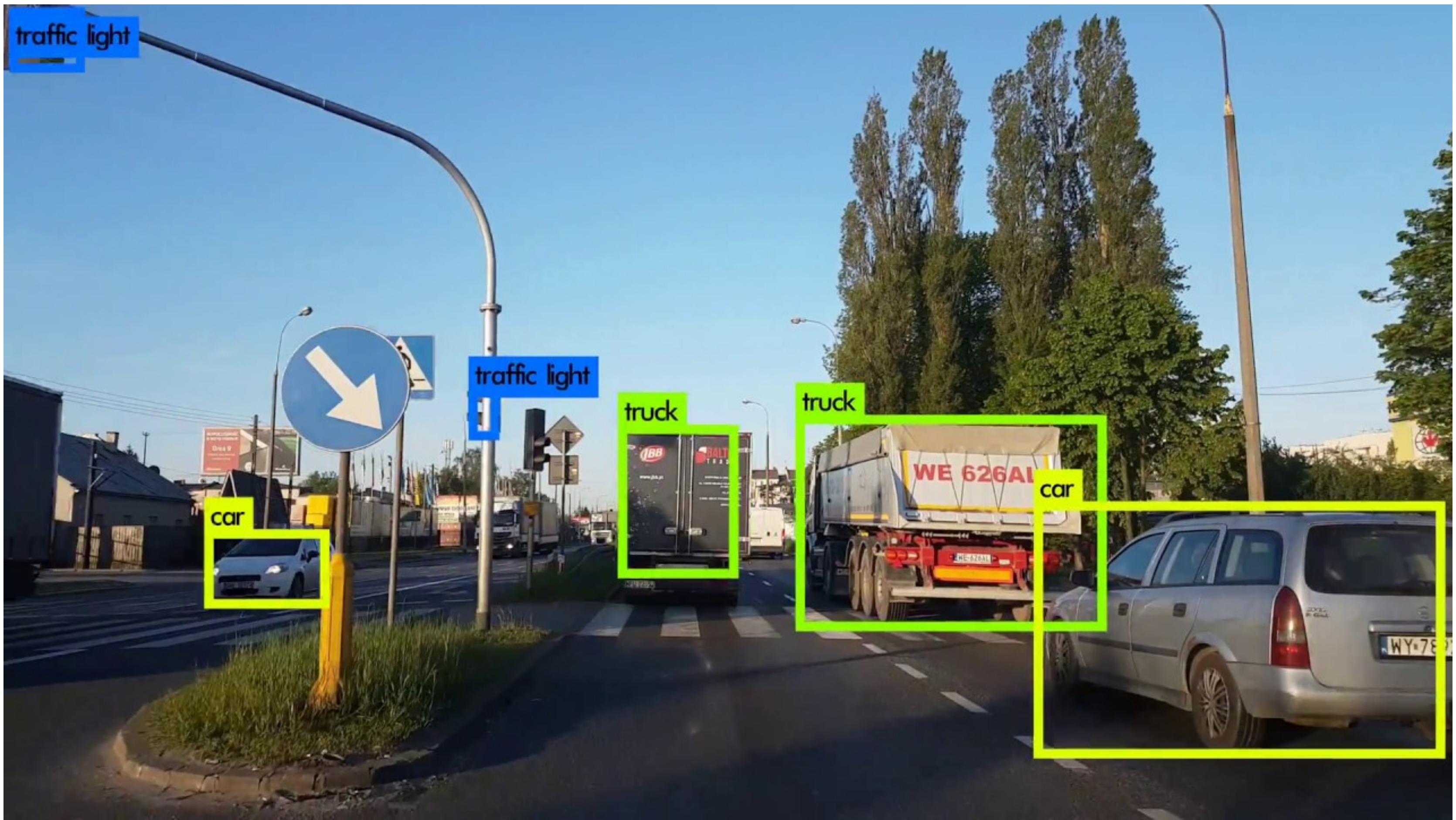
The language of recognition

- Boundaries of classes are often fuzzy
- “A dog is an animal with four legs, a tail and a snout”
- Really?



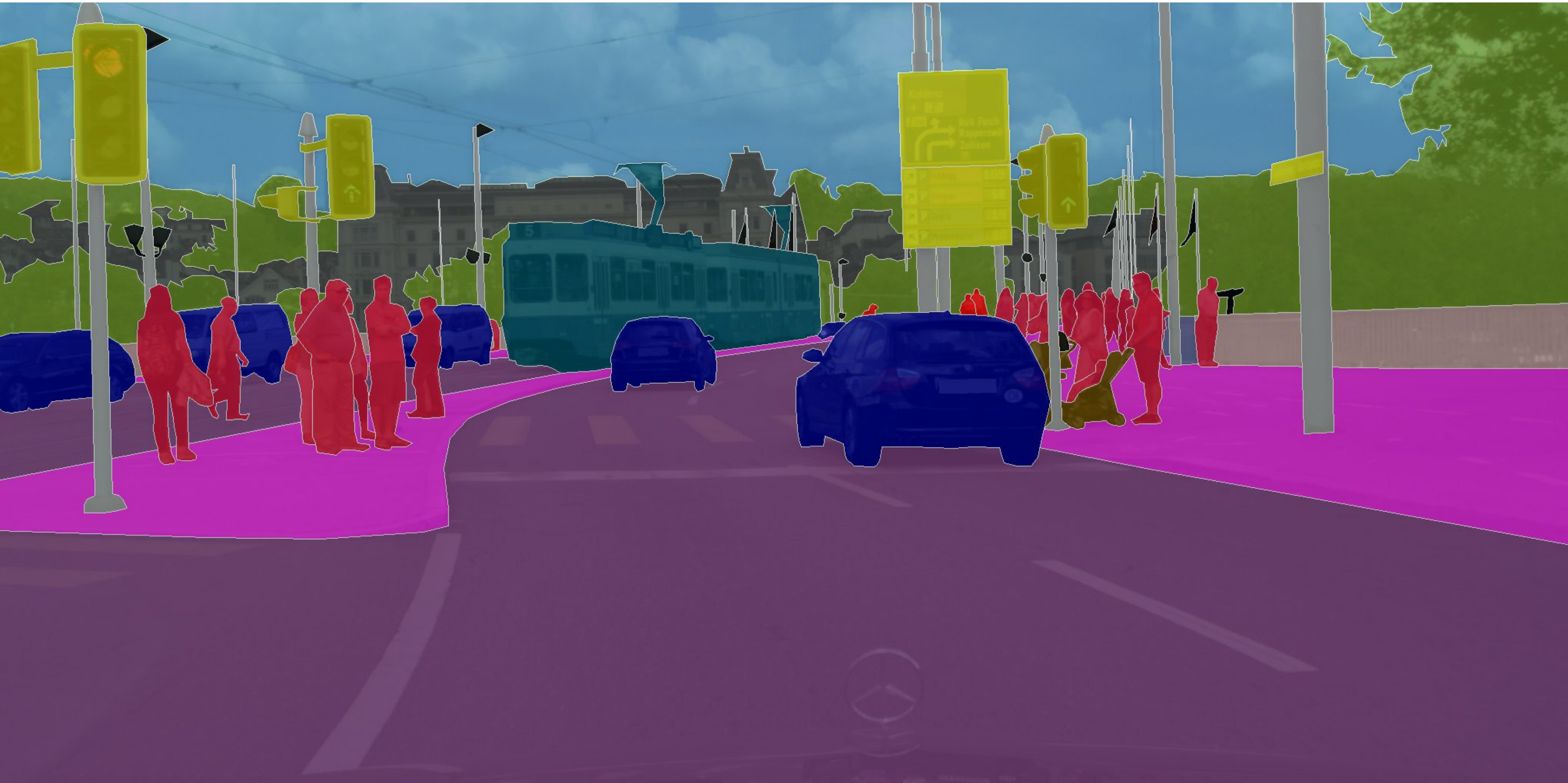
Other Recognition Problems

- Object Detection



Other Recognition Problems

- Semantic Segmentation



Other Recognition Problems

- Instance Segmentation

Semantic Segmentation

Instance Segmentation

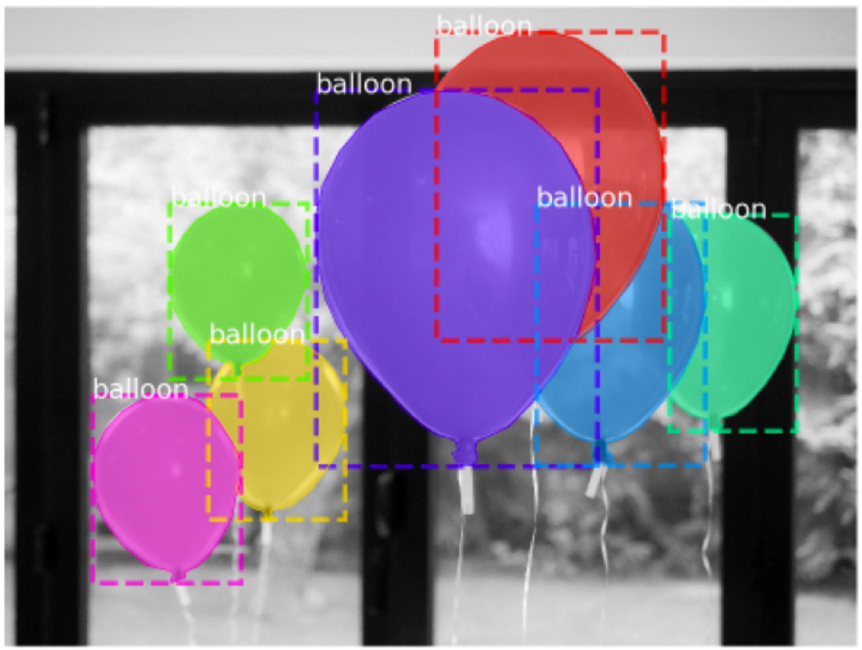


Image: https://www.jeremyjordan.me/evaluating-image-segmentation-models/

Other Recognition Problems

- Action Recognition

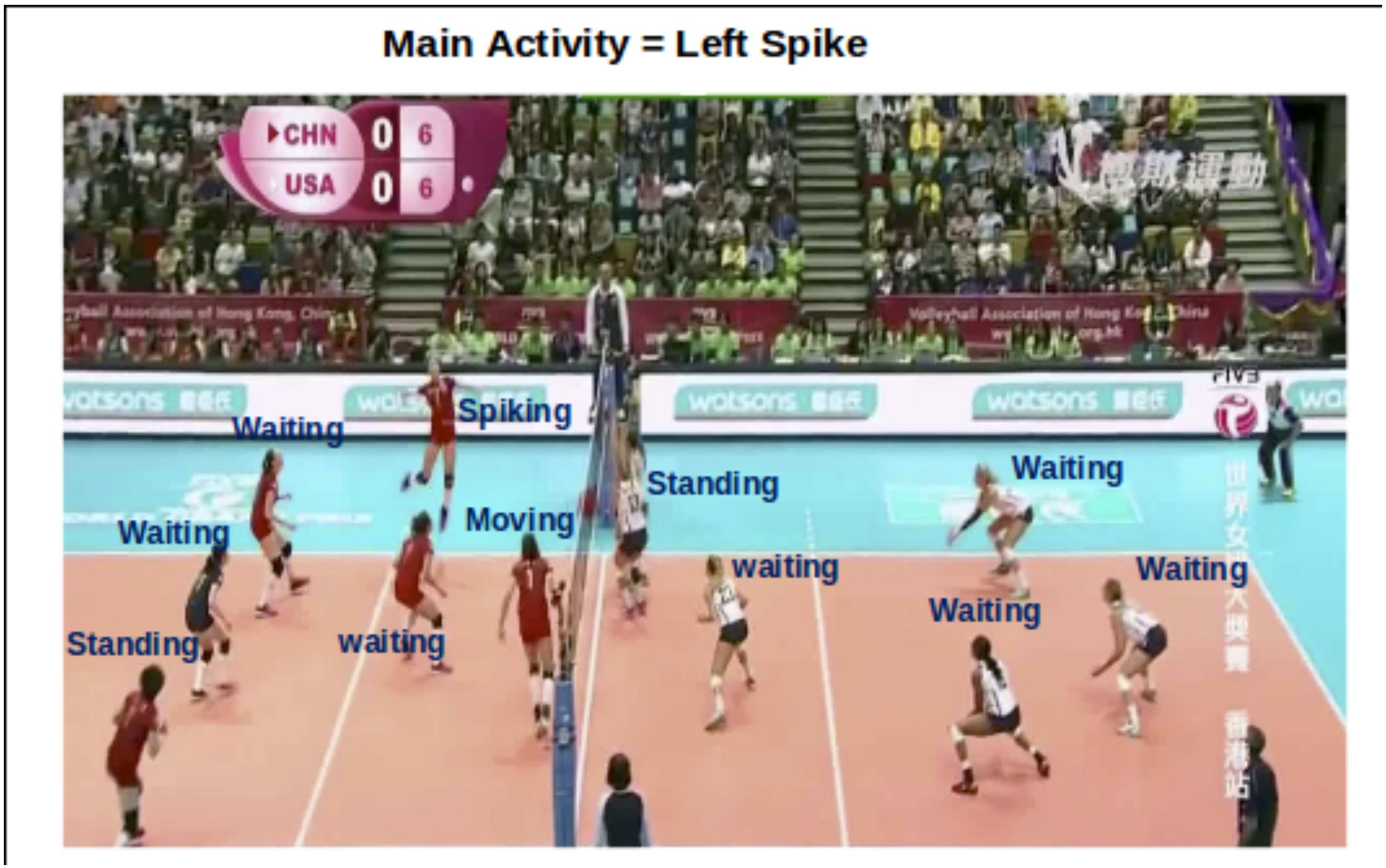


Image: http://nguyenducminhkhoi.com/project/action_recognition/

How are we going to solve this?

An image classifier

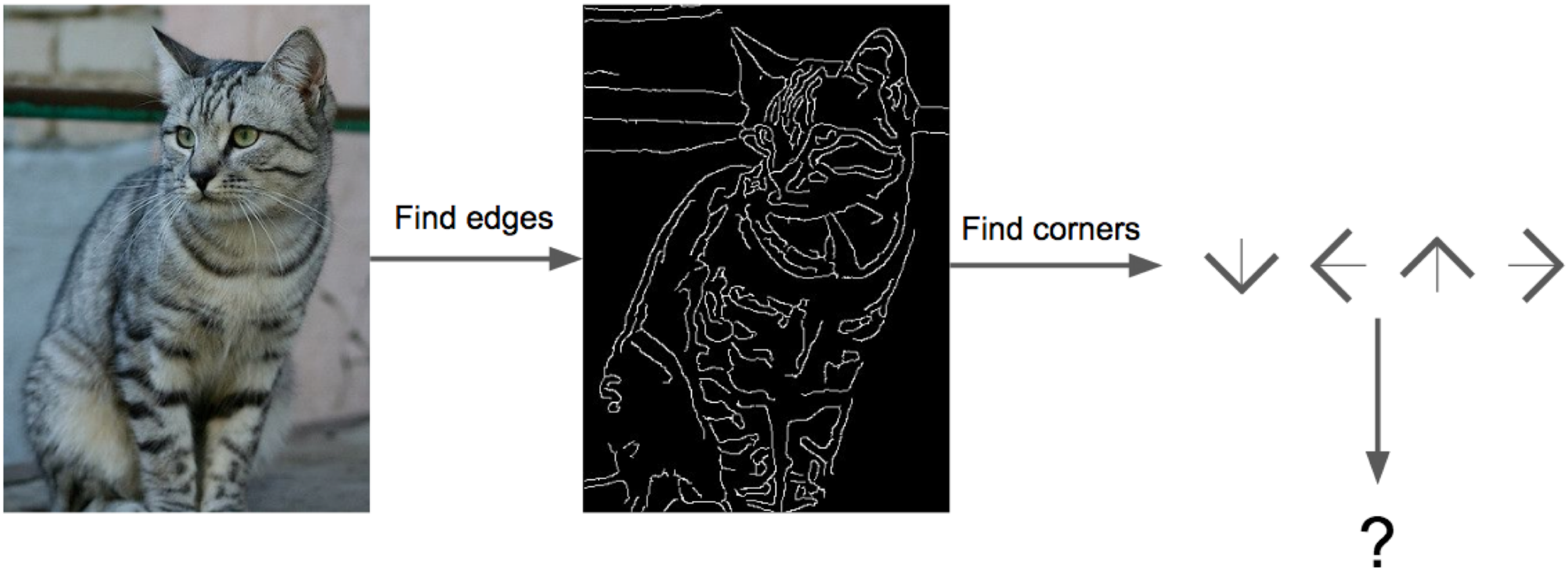
```python
def classify_image(image):
  # Some magic here?
  return class_label
```

Unlike e.g. sorting a list of numbers,

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

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

Attempts have been made



Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

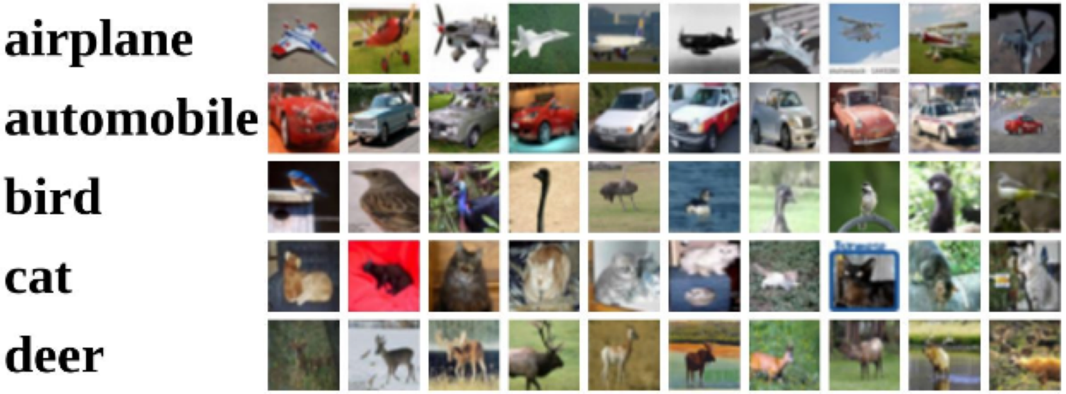
Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

```
def train(images, labels):
  # Machine learning!
  return model
```

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

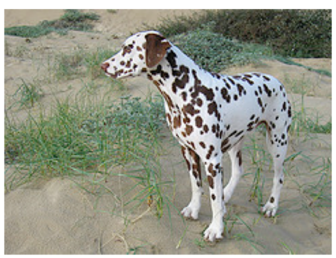
Example training set



Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

Representing Images

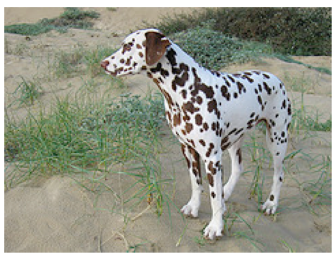
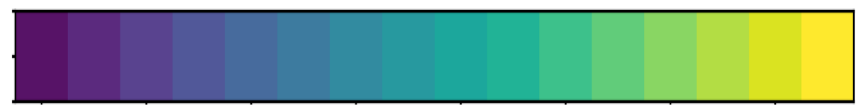
- We have images; ML works on vectors.
- To do machine learning, we need a function that takes an image and converts it into a vector.

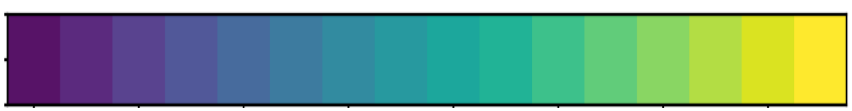
ϕ () = 

- Given an image, use ϕ to get a vector representing a point in high dimensional space

Classifying Images

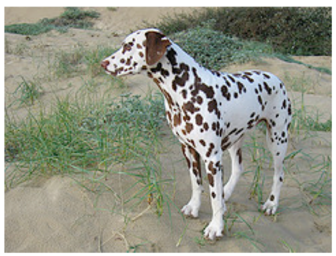
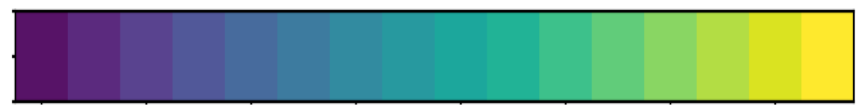
- Given an image, use ϕ to get a vector and plot it as a point in high dimensional space

ϕ () = 

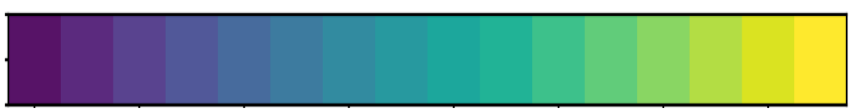
- Then, use a *classifier* function to map feature vectors to class labels:
- h() = ”dog”

Classifying Images: Pipeline

1. Represent the image in some *feature space*

ϕ () = 

2. Classify the image based on its feature representation.

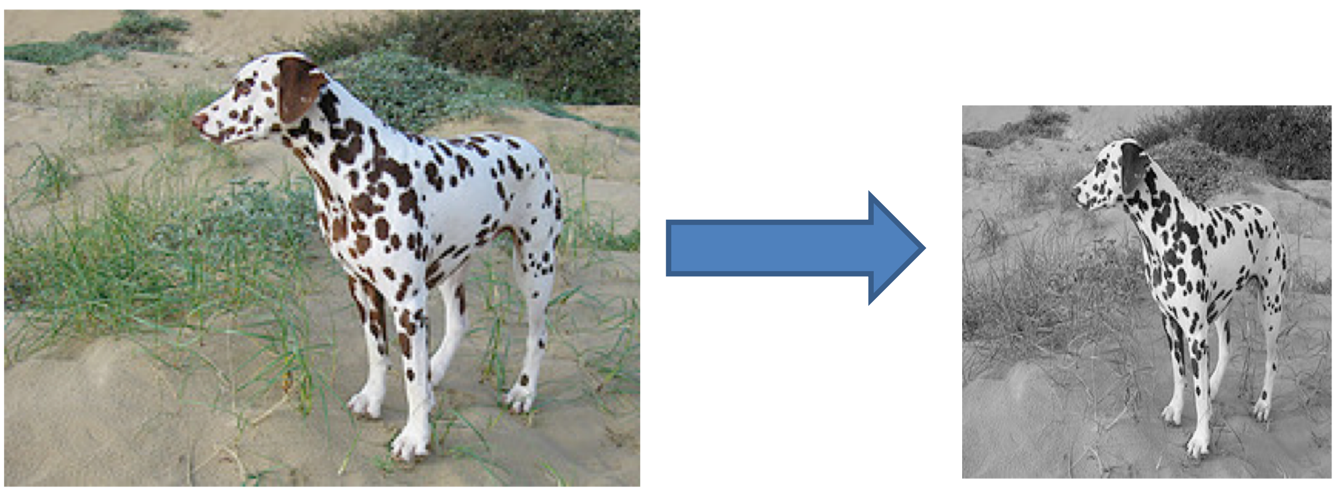
- h() = ”dog”

Two important pieces

- The feature extractor (ϕ)
- The classifier (*h*)

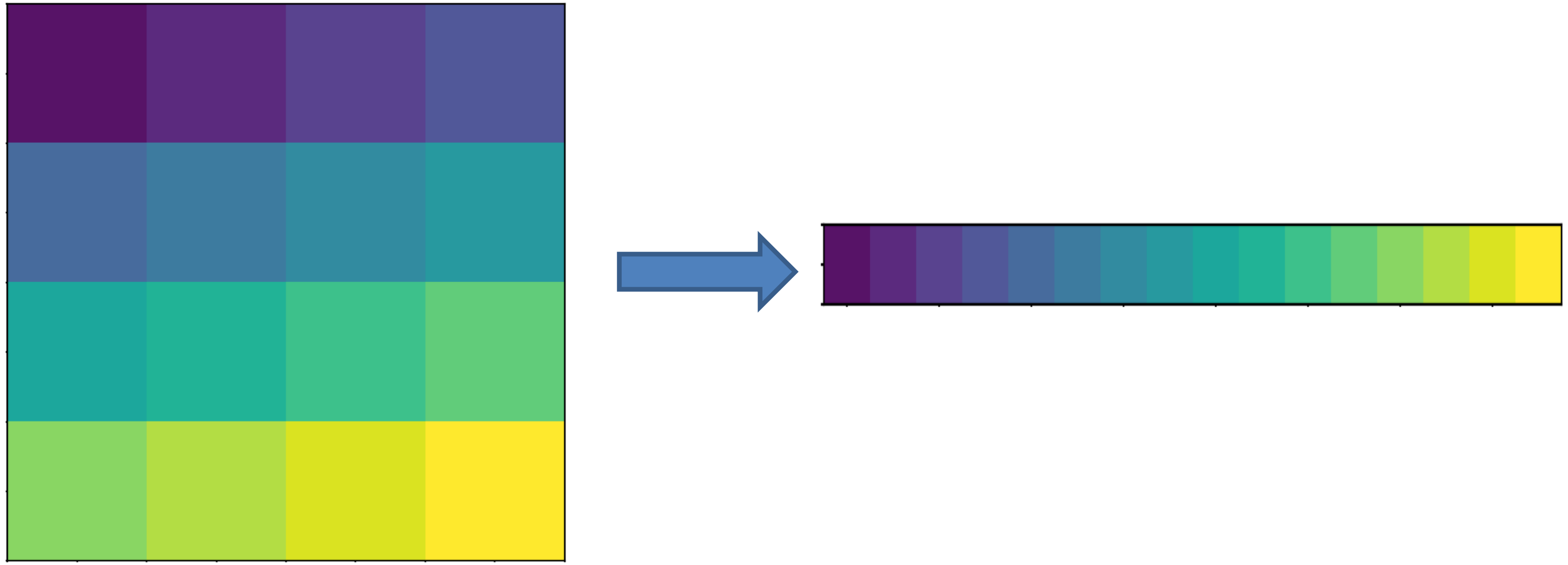
Let's make the simplest possible ϕ

- Represent an image as a vector in $\mathbb{R}^d$
- Step 1: convert image to gray-scale and resize to fixed size



Feature space: representing images as vectors

- Step 2: Flatten 2D array into 1D vector



Let’s make the simplest possible *h*

- h(x) = ”dog”
- Okay, let’s get a little less simple than that.

Let’s make a very simple *h*

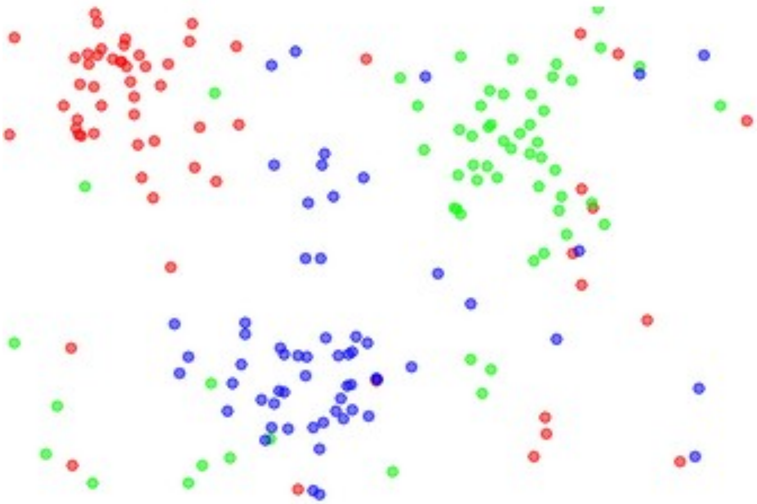
- h(x) = ”dog”
- Okay, let’s get a little less simple than that.

- I’ve never seen x before, but I’ve seen a bunch of other things.

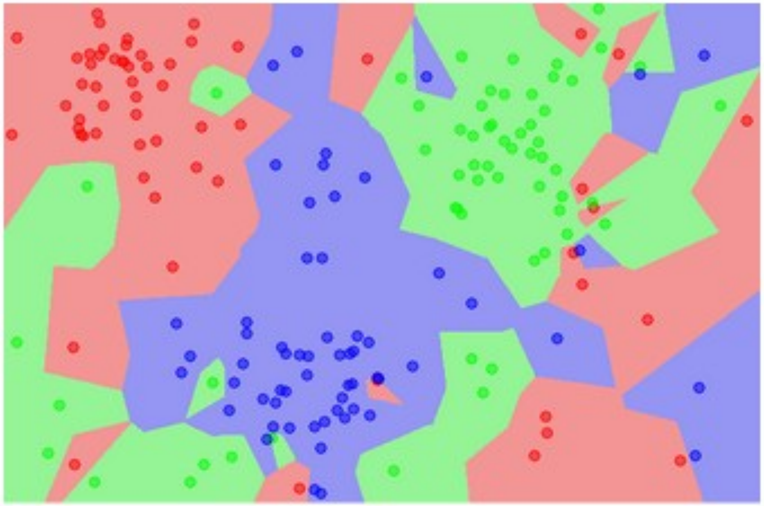
- h(x) = the label of the most similar thing to x of all the things I’ve seen.
  - assumption: **similar** data points have **similar** labels

A Simple *h*: Nearest Neighbor Classifier

the data



NN classifier



```
def train(images, labels):
  # Machine learning!
  return model
```

→ Memorize all data and labels

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

→ Predict the label of the most similar training image

Figures: Fei-Fei Li, Justin Johnson, & Serena Yeung

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Nearest Neighbor classifier

Memorize training data

Nearest Neighbor classifier

```
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

For each test image:
- Find closest train image
- Predict label of nearest image

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

What’s the runtime of train?

What’s the runtime of predict?

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

Nearest Neighbor Classifier

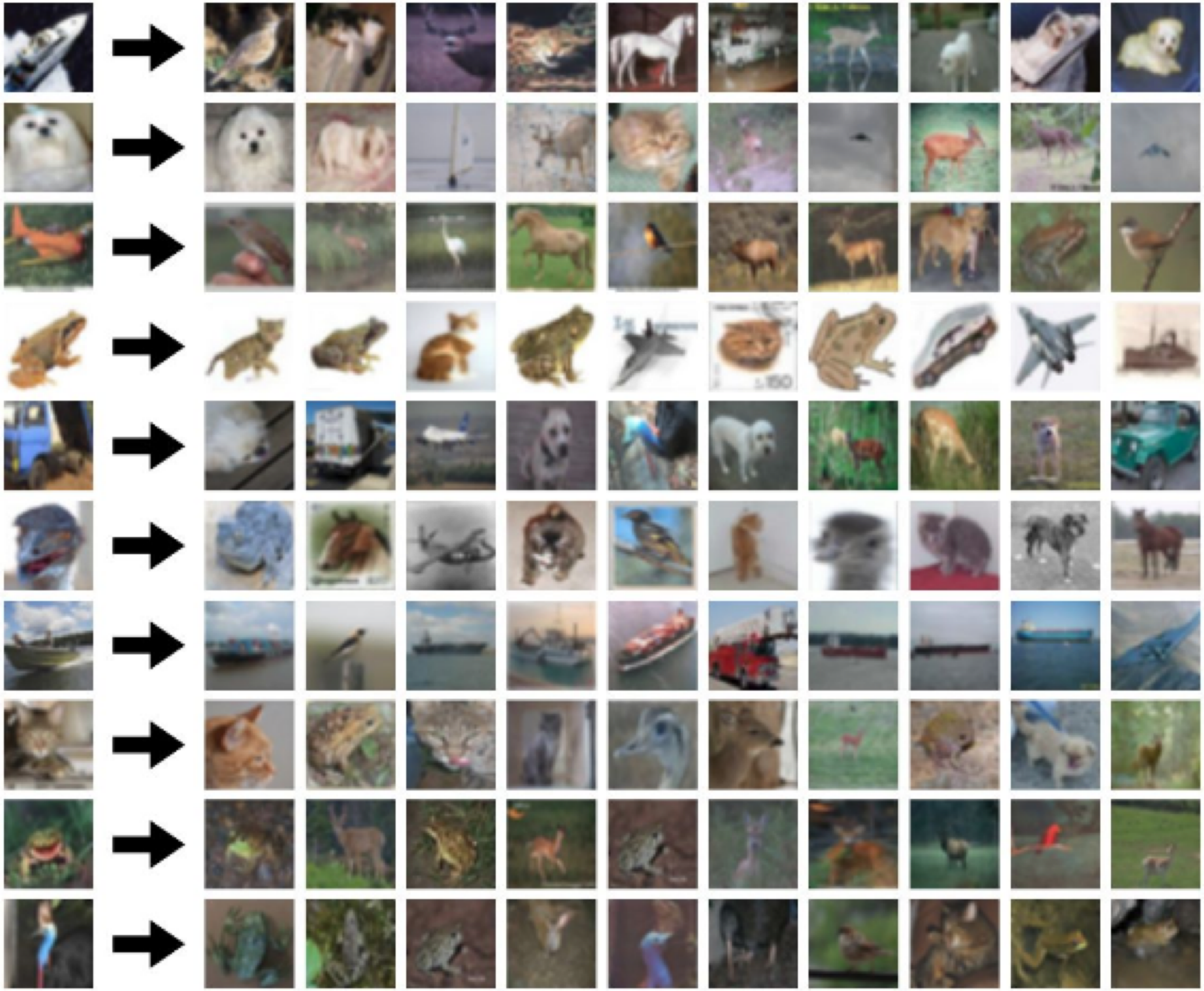
```
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

What’s the runtime of train?
O(1)

What’s the runtime of predict?
O(N)

Ideally, it’d be the other way around:

- slow training
- fast prediction



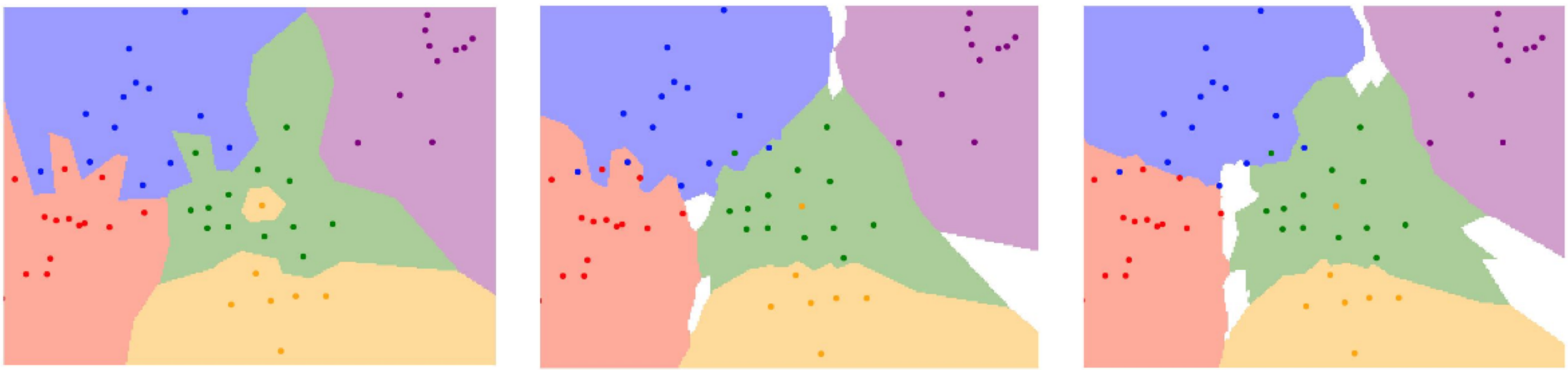
Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

An improvement: K nearest neighbors

K-Nearest Neighbors

Instead of copying label from nearest neighbor,
take **majority vote** from K closest points



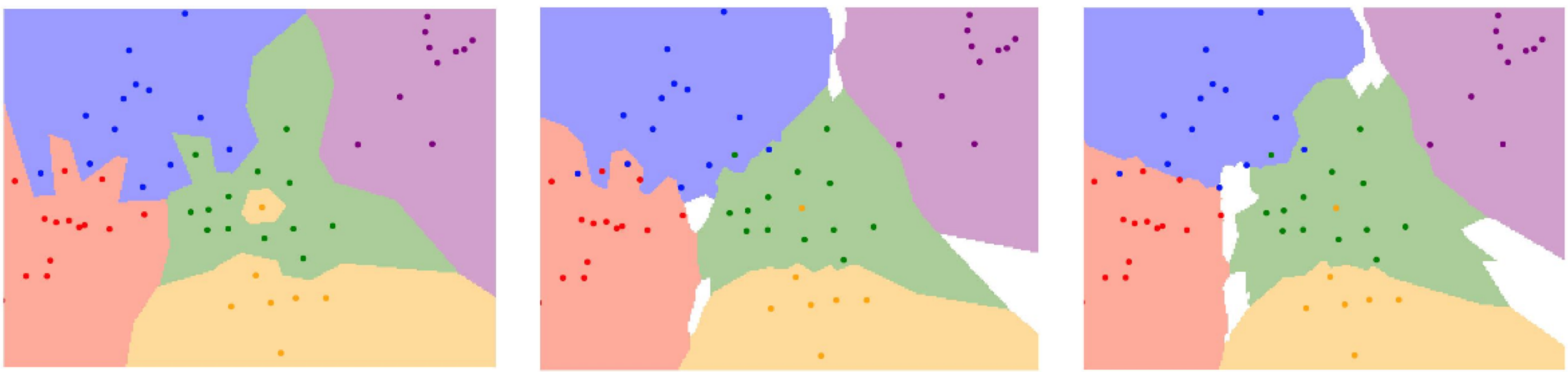
K = 1 K = 3 K = 5

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

An improvement: K nearest neighbors

K-Nearest Neighbors

Instead of copying label from nearest neighbor,
take **majority vote** from K closest points



K = 1 K = 3 K = 5

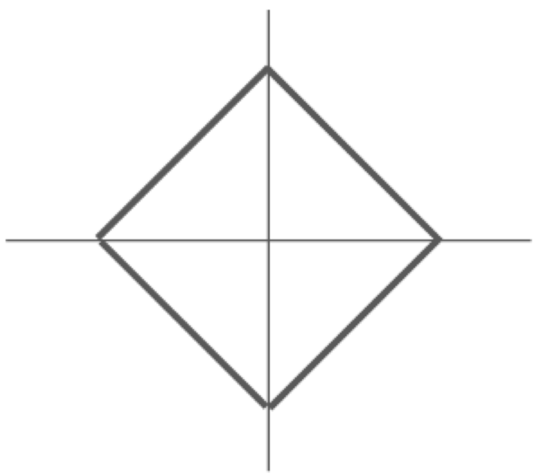
- What do we mean by "nearest" anyway?

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

K-Nearest Neighbors: Distance Metric

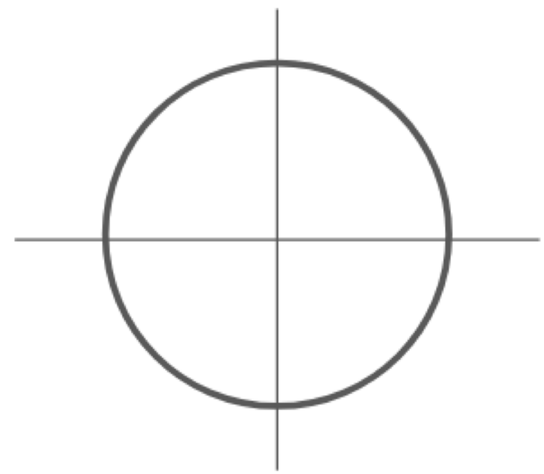
L1 (Manhattan) distance

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



L2 (Euclidean) distance

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

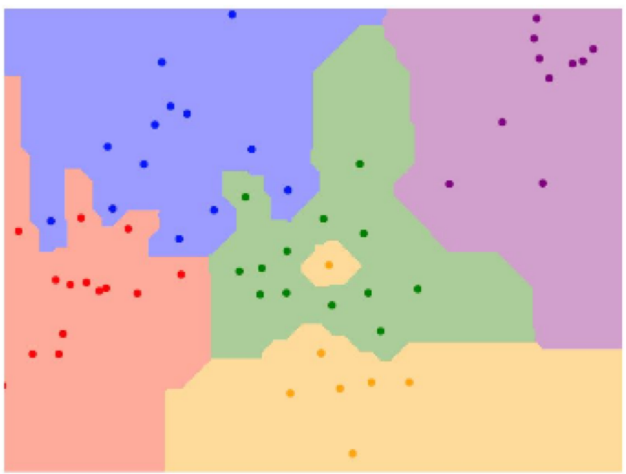


Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

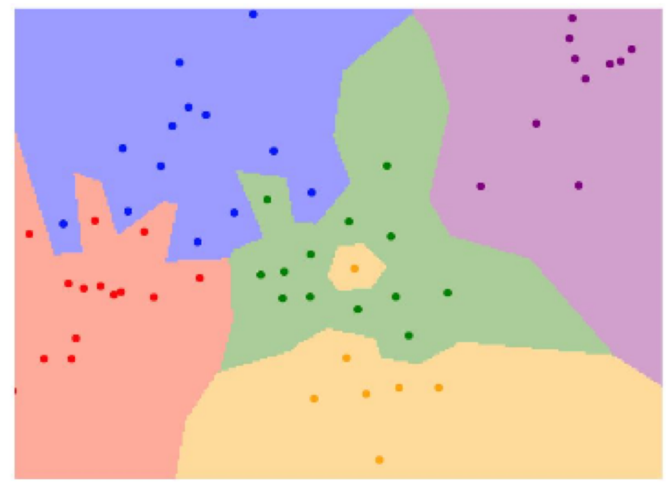
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



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

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

Demo

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

Simple Image Classification Algorithm

- ϕ : Convert to grayscale and unravel into a vector.

- *h*: Classify using majority label of the *k* nearest neighbors according to a distance metric *d*.

- k and d are **hyperparameters**. How do we know what to choose?
  - Depends on the problem
  - Usually no principled way to choose – trial and error is often the only way.

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

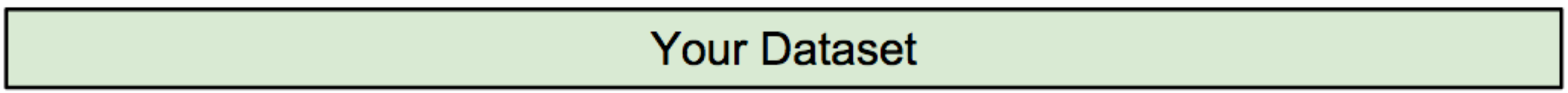
Your Dataset

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

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

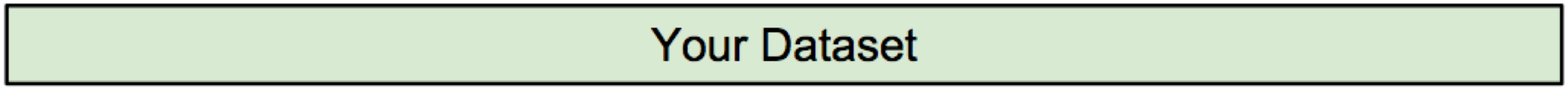
train	test

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

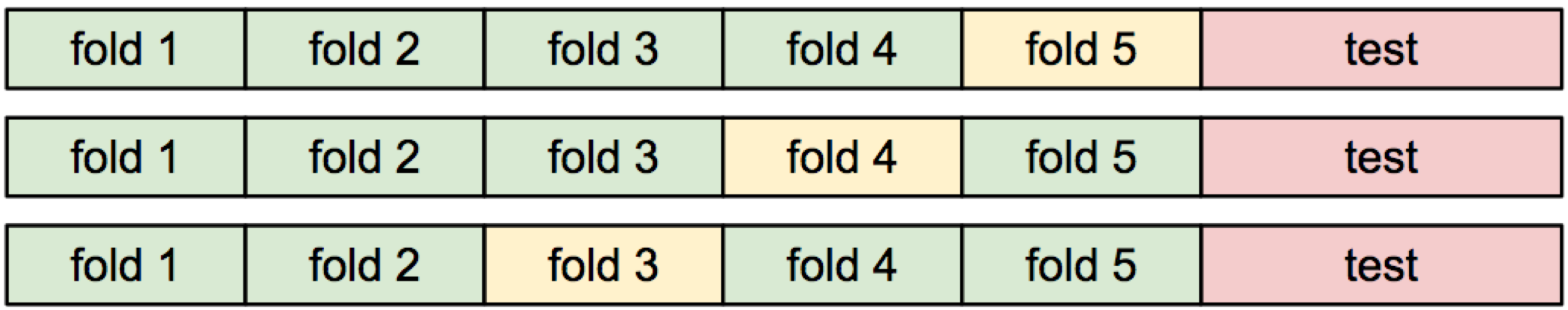
Better!

train	validation	test

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

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

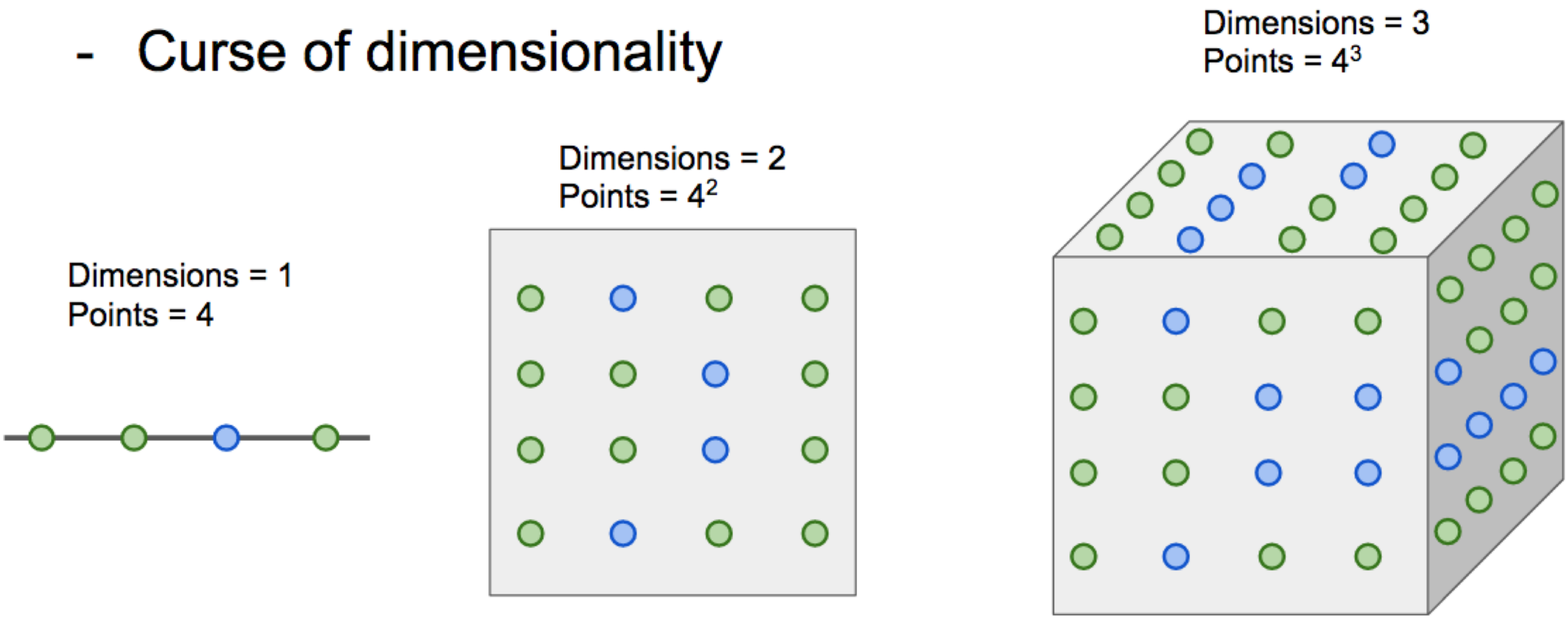
k-Nearest Neighbor on images **never used.**

- Very slow at test time
- Distance metrics on pixels are not informative

Original Boxed Shifted Tinted

(all 3 images have same L2 distance to the one on the left) k-Nearest Neighbor on images **never used.**

- Curse of dimensionality

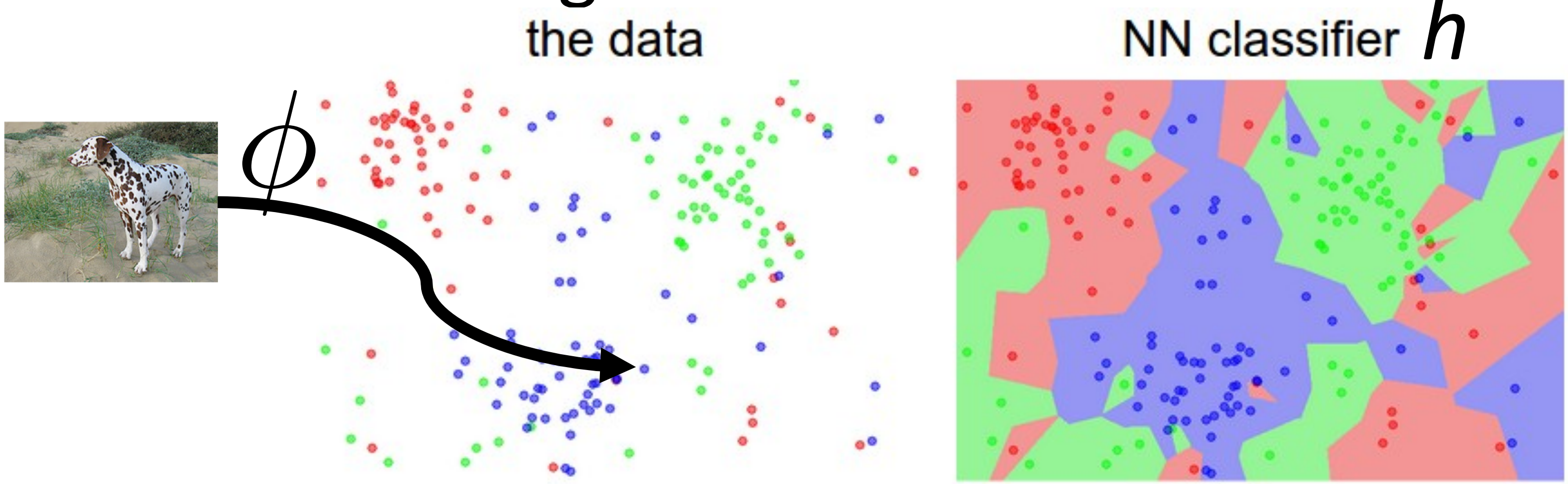


KNN: Bottom Line

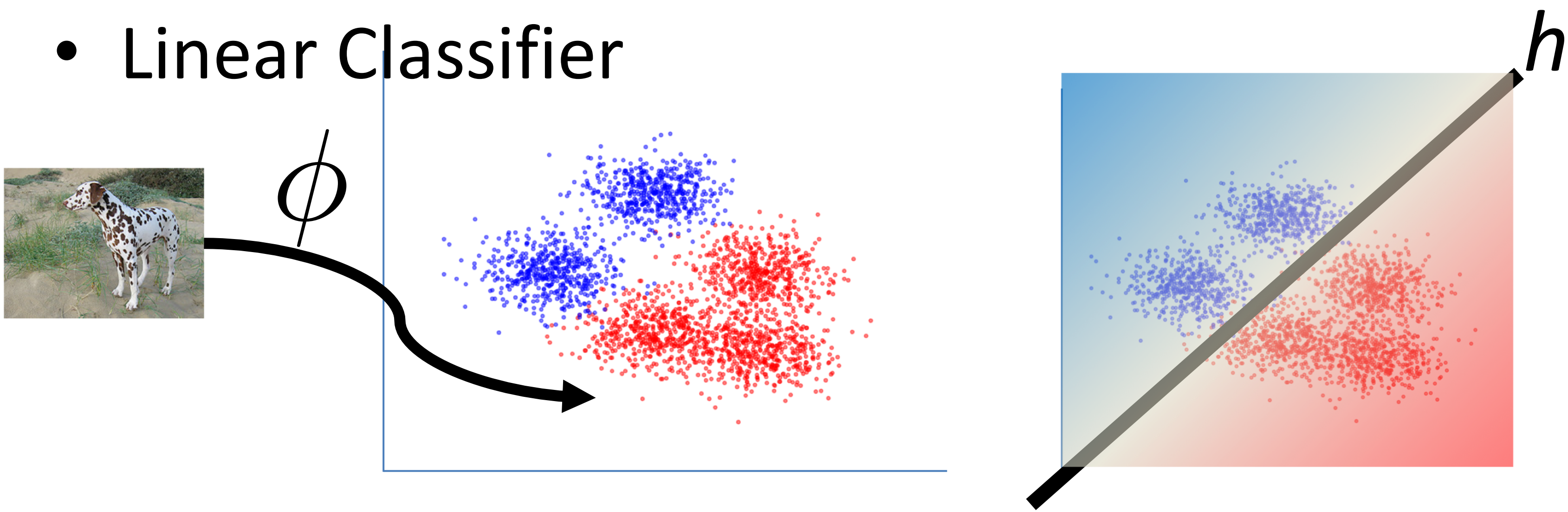
- Fast to train but slow to predict
- Distance metrics don’t behave well for high-dimensional image vectors

Classifying Images

- Nearest Neighbor Classifier



- Linear Classifier

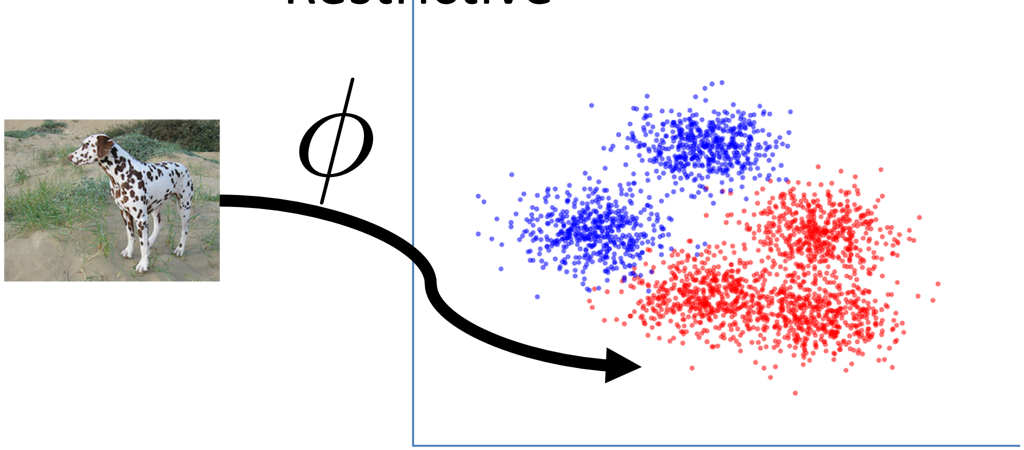


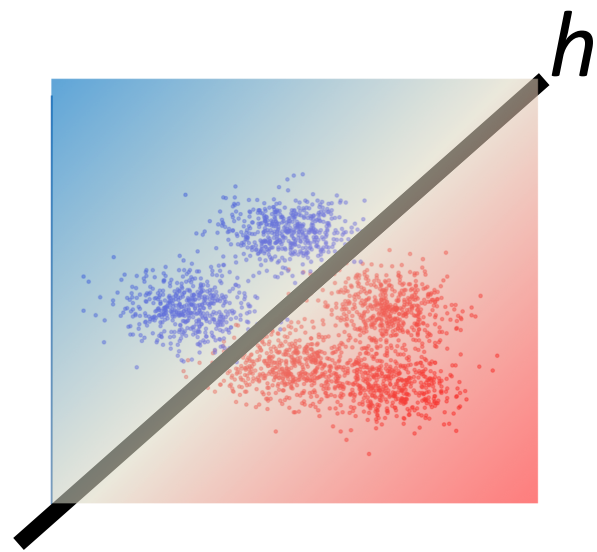
Linear classifiers

- Finding nearest neighbor is slow.
- Basic idea:
  - Training time: find a line that separates the data
  - Testing time: which side of the line is ϕ(x) on?

    +Fast to compute

    \- Restrictive





Some history of the Ante**deep**luvian Era

- Common pipeline from days of yore:
  - Detect corners and extract SIFT features
  - Collect features into a “bag of features”
  - (if you’re feeling fancy) maintain some spatial information
  - Somehow convert feature bag to fixed size
  - Apply linear classifier.

- Key idea: ϕ is designed by hand, while *h* is learned from data.

Some history of the Ante**deep**luvian Era

- Key idea: ϕ is designed by hand, while *h* is learned from data.

- Nowadays: learn both from data - "end-to-end": image goes in, label comes out.
  – Enabled only recently by bigger
    - labeled datasets
    - compute power (GPUs)