CS 4803 / 7643: Deep Learning

Topics:

- Image Classification
- Supervised Learning view
- K-NN

Dhruv Batra

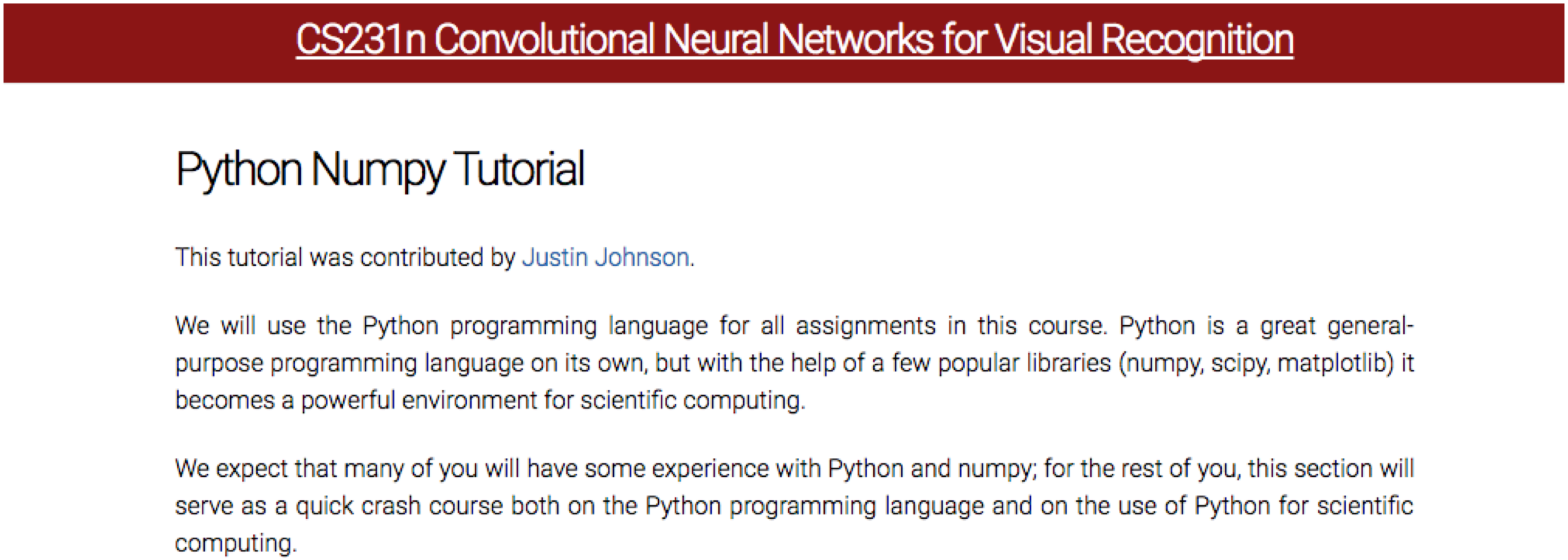
Georgia Tech

Administrativia

- Piazza
  - 165/222 people signed up. Please use that for questions.

- Gradescope/Canvas
  - Anybody not have access?
  - See note on Piazza

Python+Numpy Tutorial



http://cs231n.github.io/python-numpy-tutorial/

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

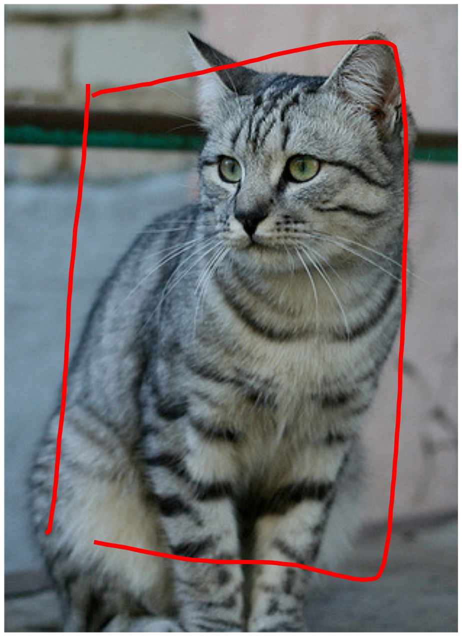
Plan for Today

- Image Classification
- Supervised Learning view
- K-NN

- Next time: Linear Classifiers

Image Classification

Image Classification: A core task in Computer Vision

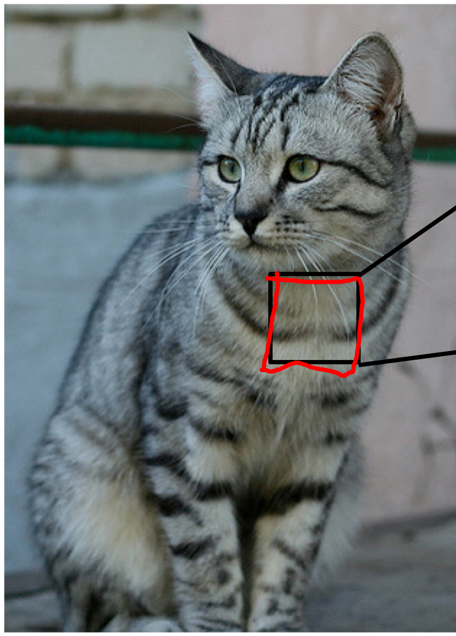


(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

→ cat

The Problem: Semantic Gap

3



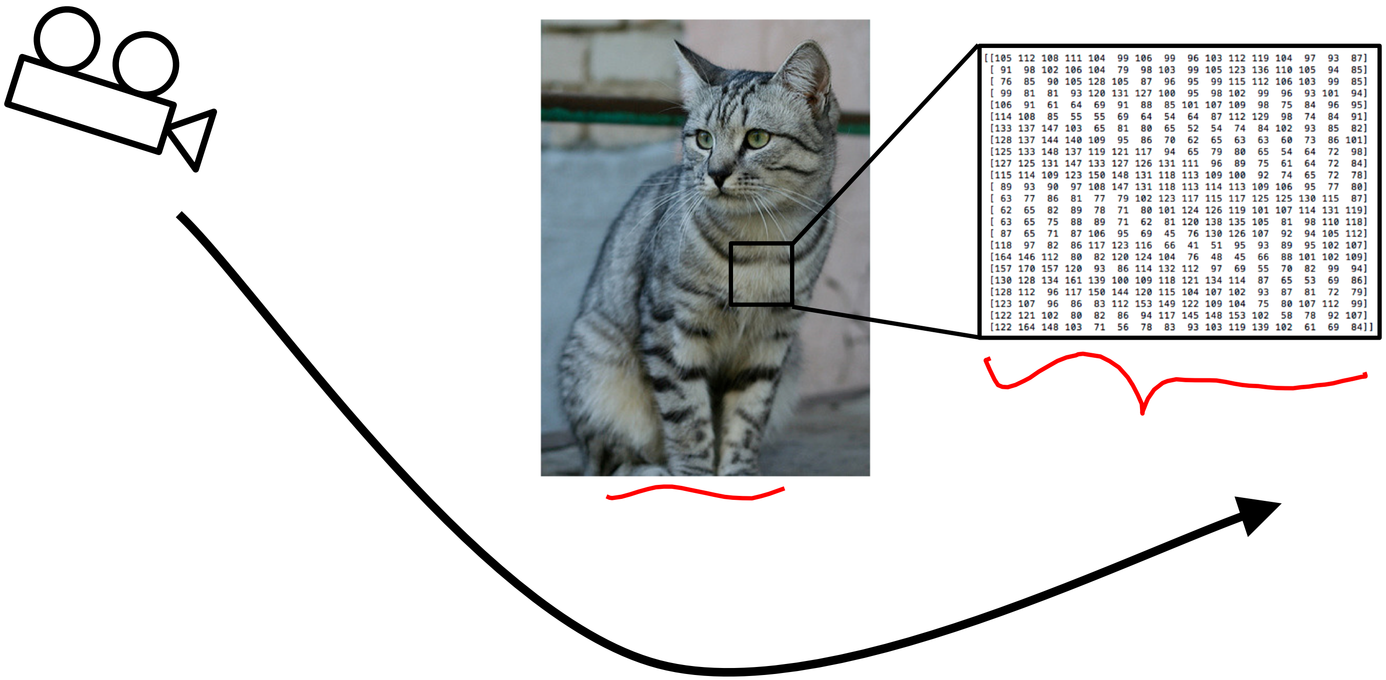

```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

What the computer sees

An image is just a big grid of numbers between [0, 255]:

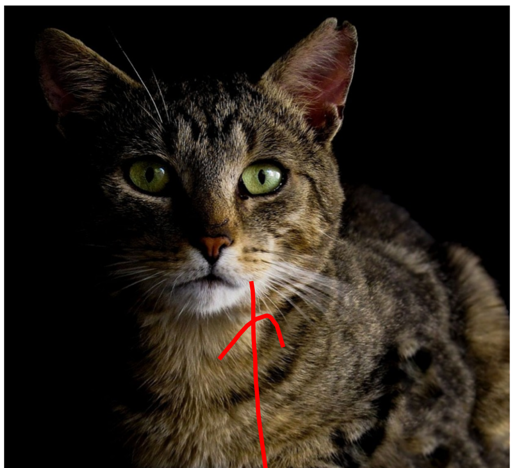
e.g. 800 x 600 x 3
(3 channels RGB)

Challenges: Viewpoint variation

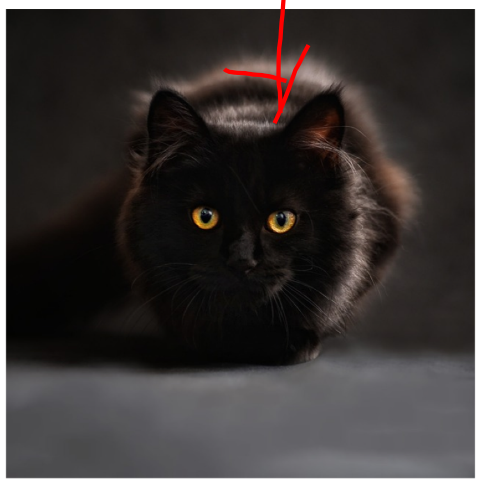


All pixels change when the camera moves!

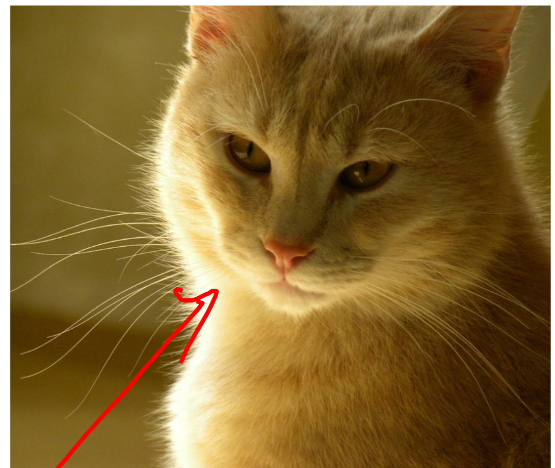
Challenges: Illumination



This image is CC0 1.0 public domain



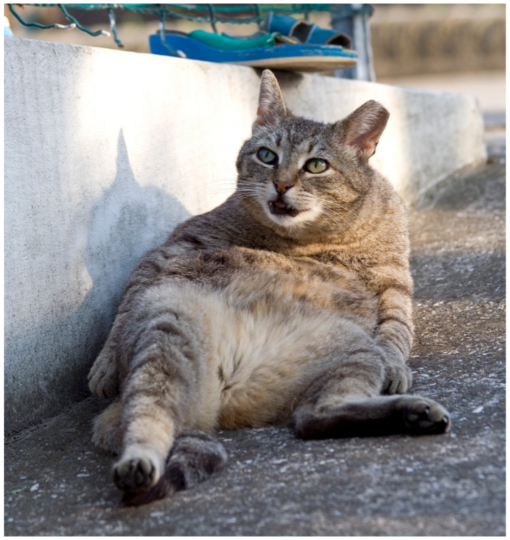
This image is CC0 1.0 public domain

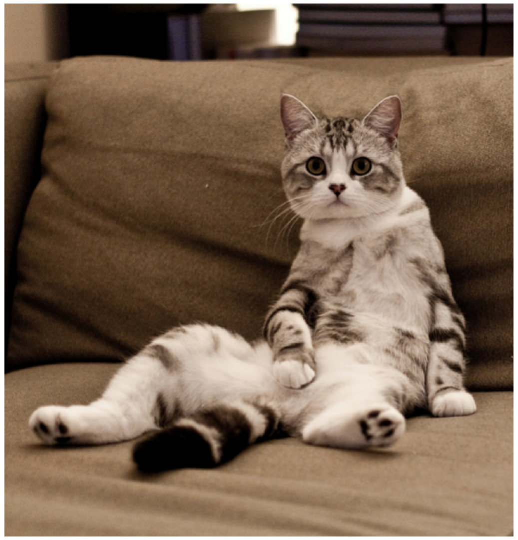


This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

Challenges: Deformation





Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Challenges: Occlusion

Challenges: Background Clutter

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

An image classifier

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

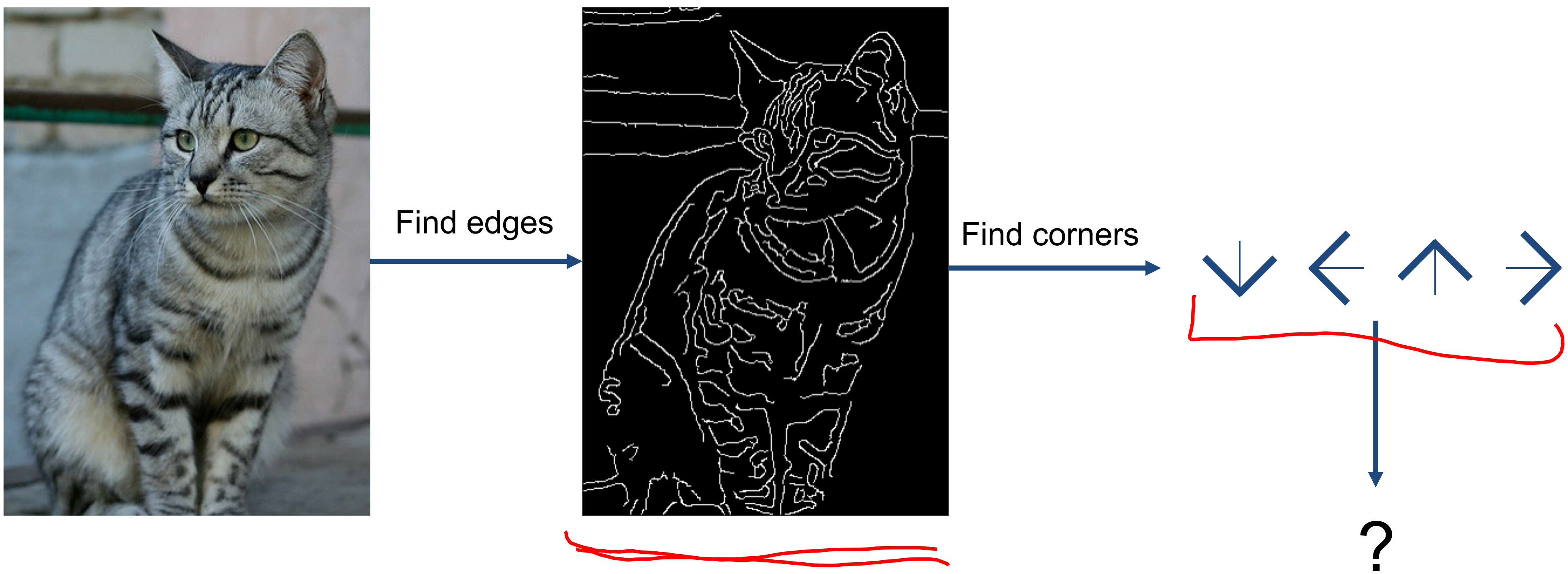
list

output list

Unlike e.g. sorting a list of numbers,

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

Attempts have been made



John Canny, "A Computational Approach to Edge Detection", IEEE TPAMI 1986

ML: A Data-Driven Approach

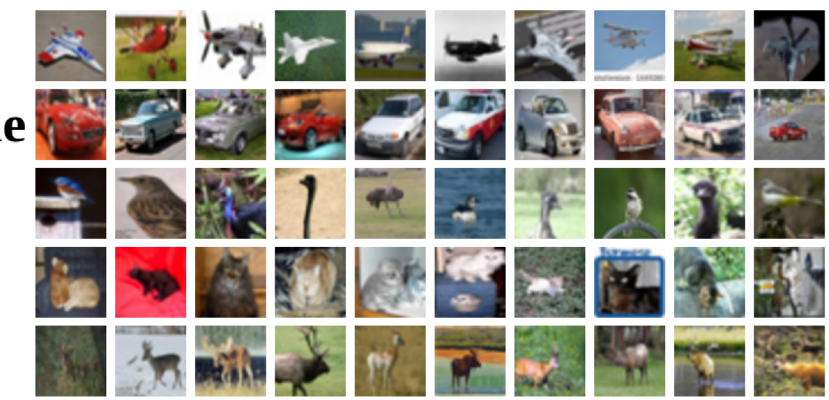
1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

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

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

Example training set

airplane
automobile
bird
cat
deer



Notation

Scalars: $x, y, z, i, \in \mathbb{R}^1$

$\vec{x}, \vec{y} \quad \in \mathbb{R}^d$

Matrices X, Y

R.V, sets

input dim d

output dim K

parameters $\vec{w}, \vec{\theta} \in \mathbb{R}^d$

#samples n, N

Supervised Learning

- Input: $\vec{x}$ $\in \mathbb{R}^1$ $\mathbb{R}^d$ $\mathbb{R}^{H \times W \times 3}$ (images, text, emails…) $V = \{1, \ldots, d\}$ $\{\text{'a'}, \text{'the'}, \ldots\}$
- Output: y $\in \{+1, -1\}$ (spam or non-spam…)
  $\{1, \ldots, K\}$ $\in \mathbb{R}^K$
- (Unknown) Target Function
  - f: X → Y (the "true" mapping / reality)
- Data
  - { (x_1,y_1), (x_2,y_2), …, (x_N,y_N) }
    x_1, $f(x_1)$

Goal: find f | Predict $f(\vec{x})$ at new $\vec{x}$

Supervised Learning

Model Class / Hypothesis Set

$$\underline{\mathcal{H}} = \{\underbrace{h : X \to Y}\} \qquad \underbrace{h(\vec{x})}_{\vec{w}} = y$$

$$\underline{\text{Loss}(h, D)} = \frac{1}{N} \sum_{i=1}^{N} L_i(h(x_i), y_i)$$

Learning = Search / Opt in MC

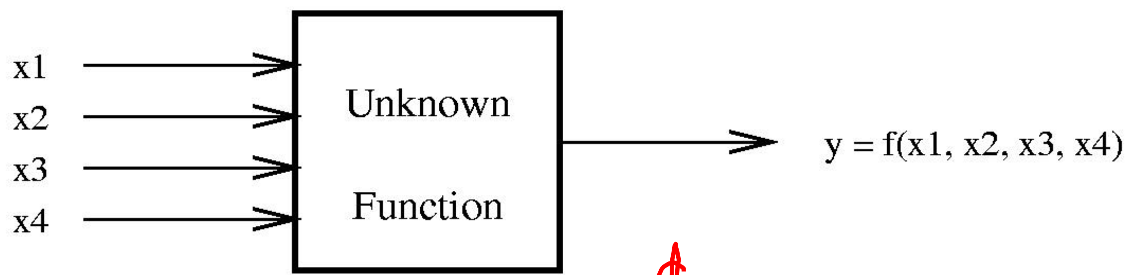
$$= \boxed{\operatorname{argmin}}_{\boxed{h \in \mathcal{H}}} \boxed{\text{Loss}}(h, D)$$

Supervised Learning

- Input: x (images, text, emails…)
- Output: y (spam or non-spam…)
- (Unknown) Target Function
  - f: X → Y (the "true" mapping / reality)
- Data
  - $\{ (x_1,y_1), (x_2,y_2), \ldots, (x_N,y_N) \}$
- Model / Hypothesis Class
  - H = {h: X → Y}
  - e.g. $y = h(x) = \text{sign}(w^T x)$
- Learning = Search in hypothesis space
  - Find best ~~g~~ h in model class.

Learning is hard!

A Learning Problem



#fn

$= 2^{2^d} - n$

$2^d - n$

Example	x_1	x_2	x_3	x_4	y
1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	1	1	1
4	1	0	0	1	1
5	0	1	1	0	0
6	1	1	0	0	0
7	0	1	0	1	0

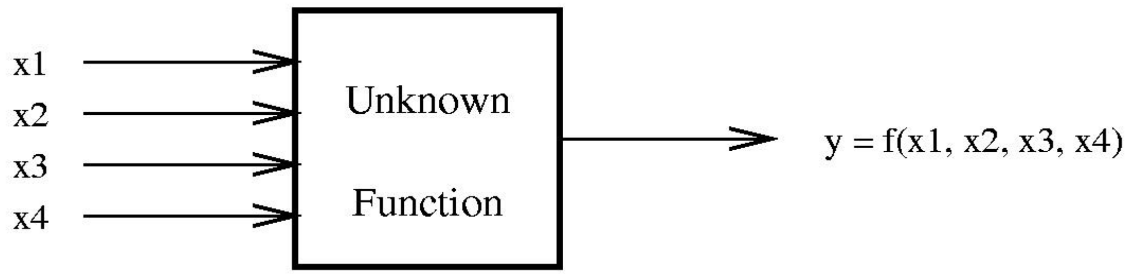
x_1	x_2	x_3	x_4	y
0	1	1	1	0, 1

2^{16}

Learning is hard!

- No assumptions = No learning

A Learning Problem



Example	x_1	x_2	x_3	x_4	y
1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	1	1	1
4	1	0	0	1	1
5	0	1	1	0	0
6	1	1	0	0	0
7	0	1	0	1	0

Procedural View

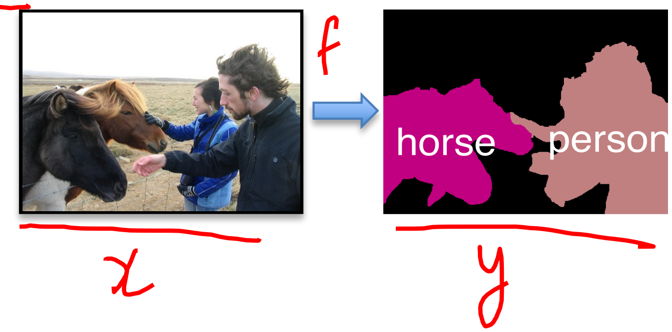
- Training Stage:
  - Training Data $\{ (x_i,y_i) \} \rightarrow h$ (Learning)
- Testing Stage
  - Test Data $x \rightarrow h(x)$ (Apply function, Evaluate error)

Statistical Estimation View

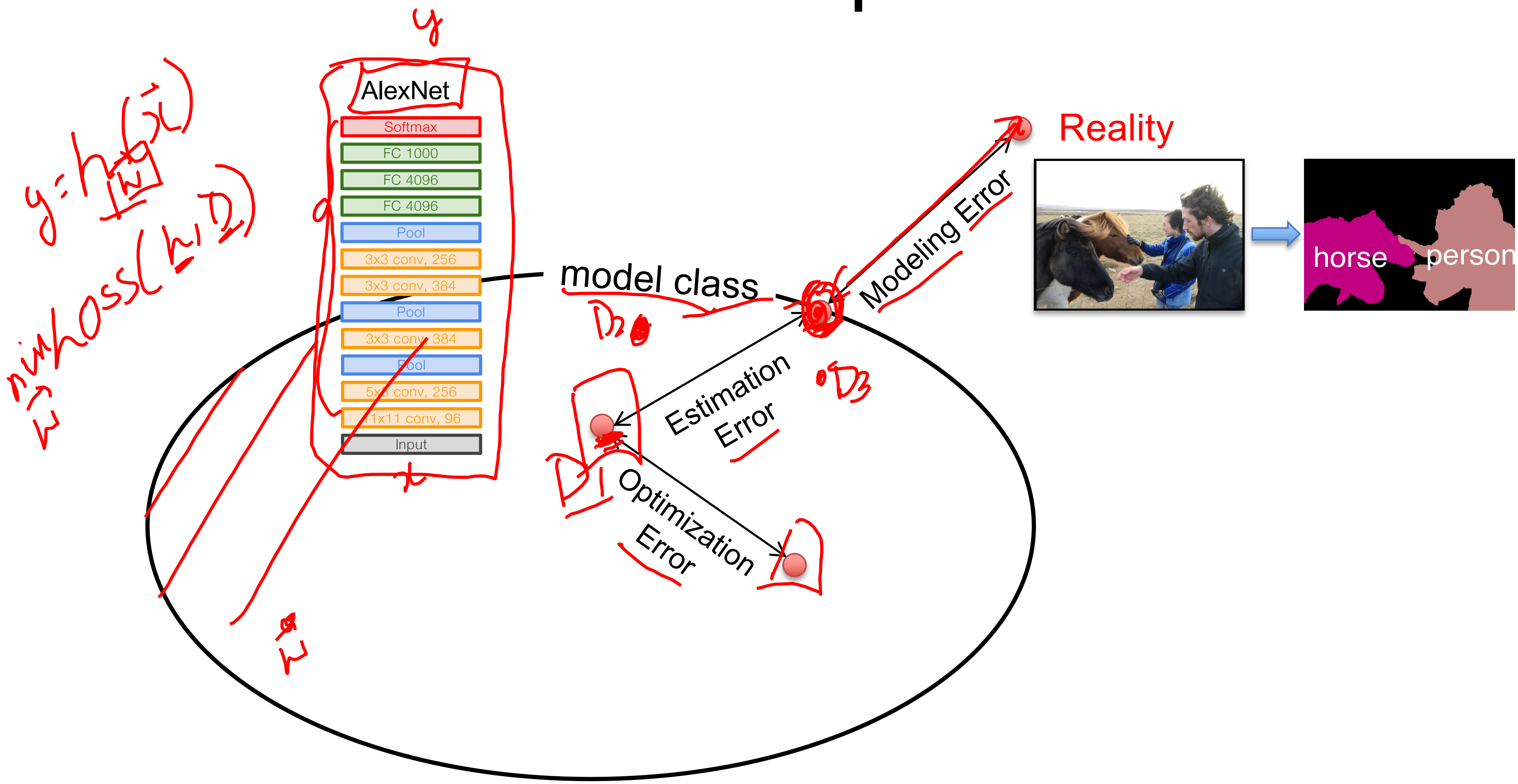
- Probabilities to rescue:
  - X and Y are *random variables*
  - $D = (x_1,y_1), (x_2,y_2), \ldots, (x_N,y_N) \quad \sim P(X,Y)$
- IID: Independent Identically Distributed
  - Both training & testing data sampled IID from P(X,Y)
  - Learn on training set
  - Have some hope of *generalizing* to test set

Error Decomposition

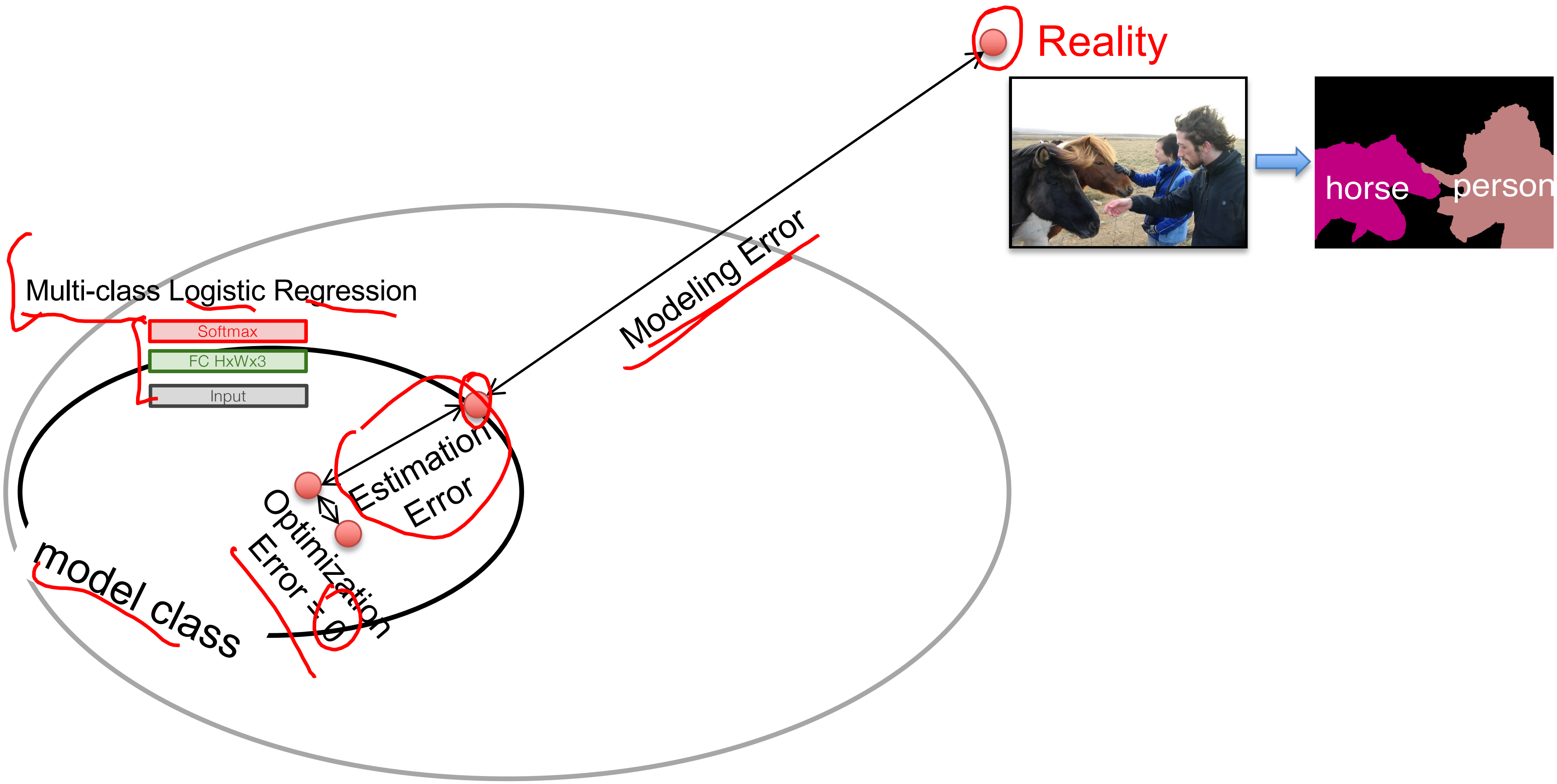
- Reality



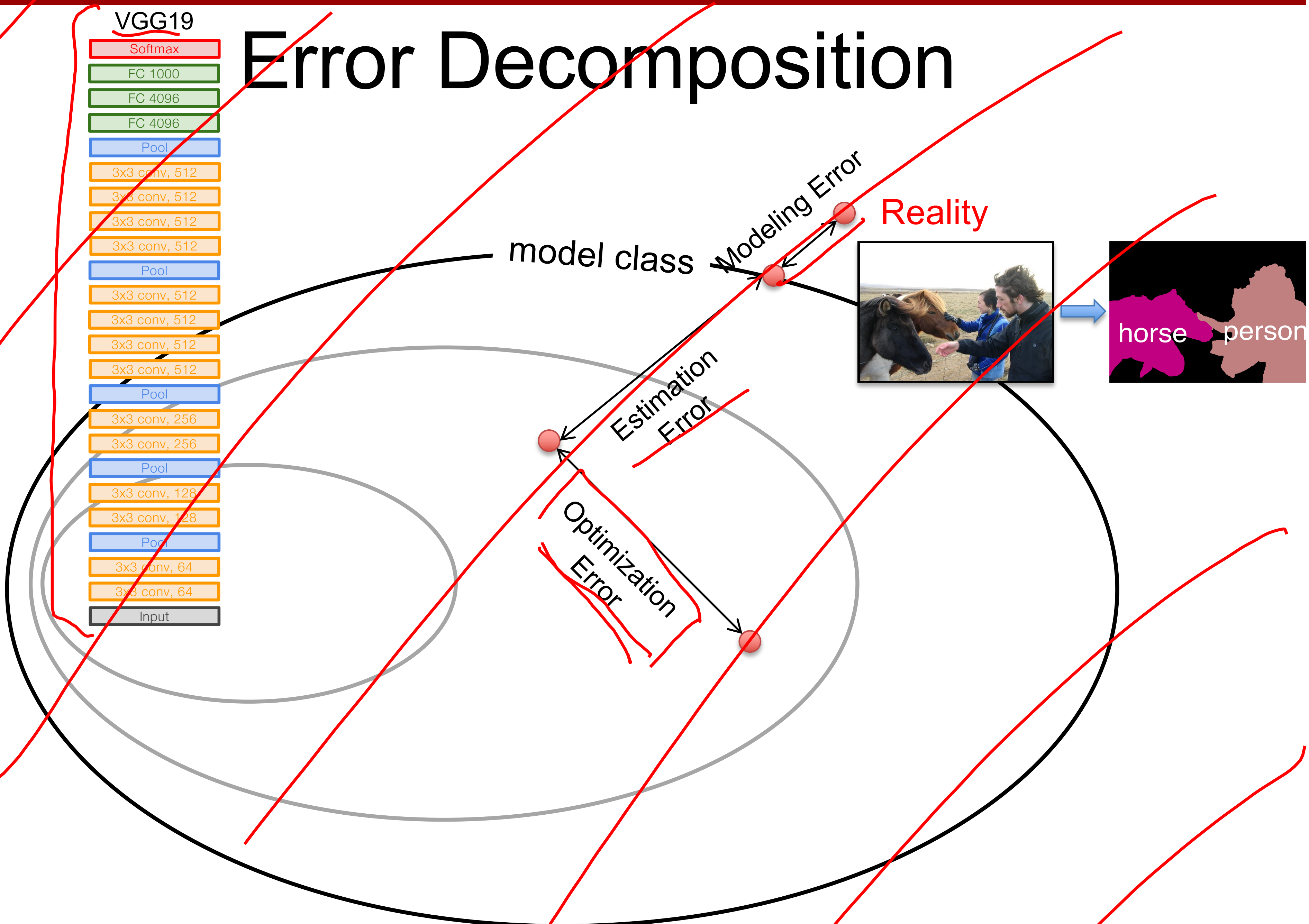
Error Decomposition



Error Decomposition



Error Decomposition



Error Decomposition

- Approximation/Modeling Error
  - You approximated reality with model
- Estimation Error
  - You tried to learn model with finite data
- Optimization Error
  - You were lazy and couldn't/didn't optimize to completion
- Bayes Error
  - Reality just sucks

First classifier: **Nearest Neighbor**

D

```
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

Example Dataset: **CIFAR10**

10 classes
50,000 training images
10,000 testing images

Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, Technical Report, 2009.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

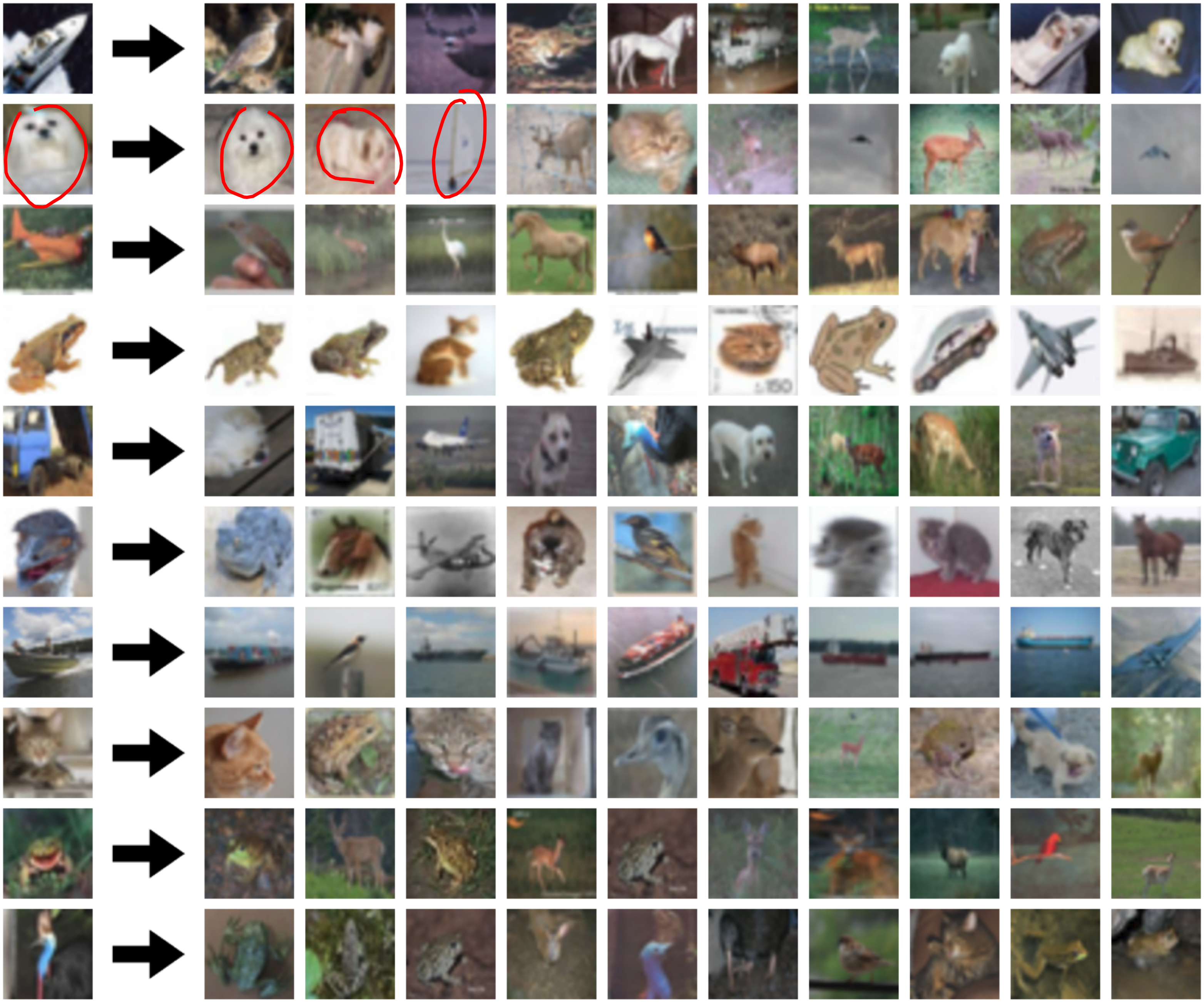
Example Dataset: **CIFAR10**

10 classes
50,000 training images
10,000 testing images

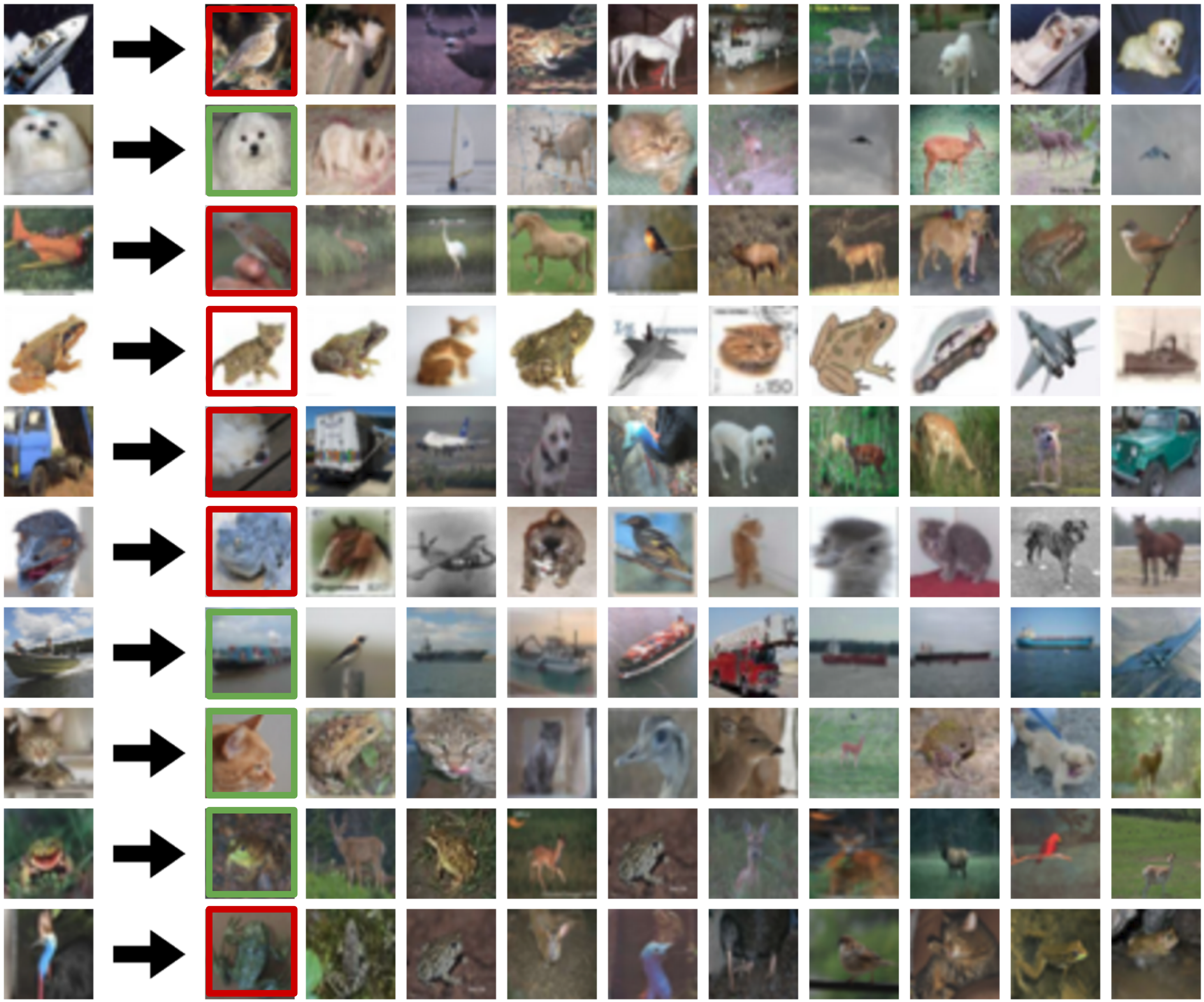
Test images and nearest neighbors

Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Technical Report, 2009.

Nearest Neighbours



Nearest Neighbours



Instance/Memory-based Learning

Four things make a memory based learner:

- *A distance metric* $d(\vec{x}_i, \vec{x}_j)$
- *How many nearby neighbors to look at?* $\vec{x}$ $\underbrace{\cdot\ \cdot\ -\ \cdot}$ k
- *A weighting function (optional)*
- *How to fit with the local points?*

1-Nearest Neighbour

Four things make a memory based learner:

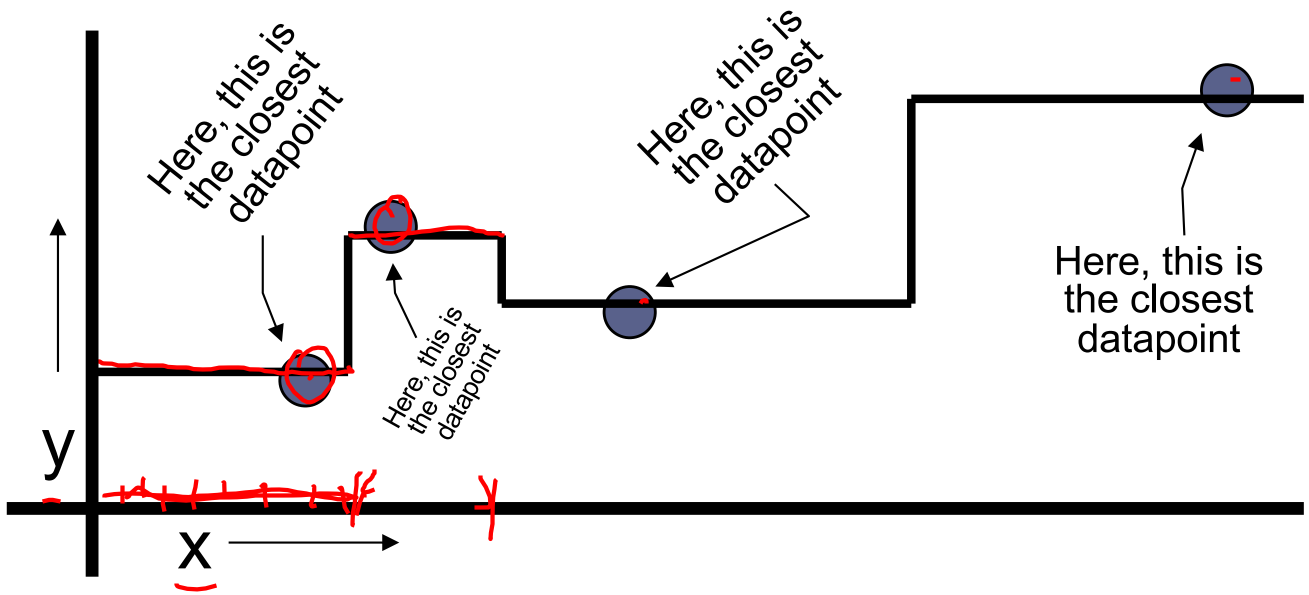
- *A distance metric*
  - **Euclidean (and others)**
- *How many nearby neighbors to look at?*
  - **1**
- *A weighting function (optional)*
  - **unused**
- *How to fit with the local points?*
  - **Just predict the same output as the nearest neighbour.**

k-Nearest Neighbour

Four things make a memory based learner:

- *A distance metric*
  - **Euclidean (and others)**
- *How many nearby neighbors to look at?*
  - **k**
- *A weighting function (optional)*
  - **unused**
- *How to fit with the local points?*
  - **Just predict the average output among the nearest neighbours.**

1-NN for Regression



Distance Metric to compare images

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

$$\left|\;\overset{\text{test image}}{\begin{array}{|c|c|c|c|}\hline 56 & 32 & 10 & 18 \\ \hline 90 & 23 & 128 & 133 \\ \hline 24 & 26 & 178 & 200 \\ \hline 2 & 0 & 255 & 220 \\ \hline \end{array}} - \overset{\text{training image}}{\begin{array}{|c|c|c|c|}\hline 10 & 20 & 24 & 17 \\ \hline 8 & 10 & 89 & 100 \\ \hline 12 & 16 & 178 & 170 \\ \hline 4 & 32 & 233 & 112 \\ \hline \end{array}}\;\right| = \overset{\text{pixel-wise absolute value differences}}{\begin{array}{|c|c|c|c|}\hline 46 & 12 & 14 & 1 \\ \hline 82 & 13 & 39 & 33 \\ \hline 12 & 10 & 0 & 30 \\ \hline 2 & 32 & 22 & 108 \\ \hline \end{array}} \xrightarrow{\text{add}} 456$$

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
```

Nearest Neighbor classifier

```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
```

Memorize training data

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

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

```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

Q: With N examples, how fast are training and prediction?

```
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

Q: With N examples, how fast are training and prediction?

A: Train O(1), predict O(N)

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

```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
```

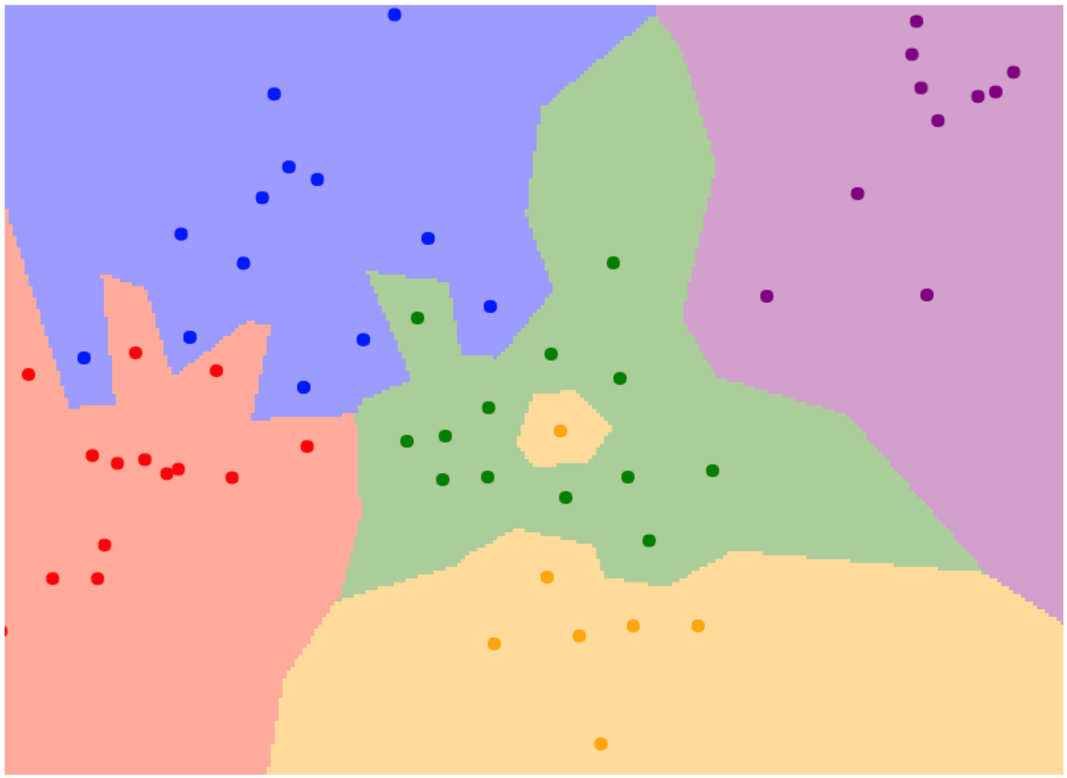
Nearest Neighbor classifier

Q: With N examples, how fast are training and prediction?

A: Train O(1),
predict O(N)

This is bad: we want classifiers that are **fast** at prediction; **slow** for training is ok

What does this look like?



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Nearest Neighbour

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

Parametric vs Non-Parametric Models

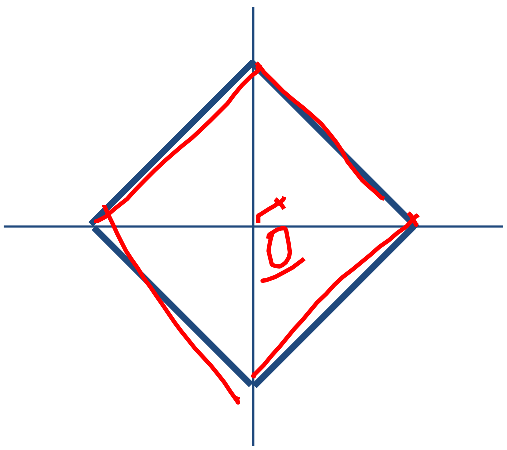
- Does the capacity (size of hypothesis class) grow with size of training data?
  - Yes = Non-Parametric Models
  - No = Parametric Models

$H = \{ h : X \rightarrow Y \}$

K-Nearest Neighbors: Distance Metric

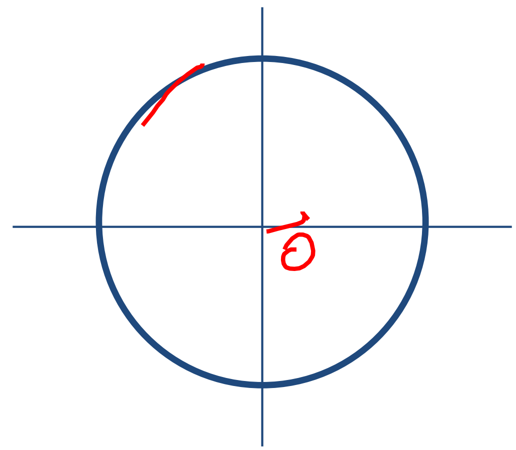
L1 (Manhattan) distance

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



L2 (Euclidean) distance

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



Hyperparameters

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

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

Hyperparameters

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

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

Very problem-dependent.
Must try them all out and see what works best.

Hyperparameters

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

Your Dataset

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Hyperparameters

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

BAD: K = 1 always works perfectly on training data

Your Dataset

Hyperparameters

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

BAD: K = 1 always works perfectly on training data

Your Dataset

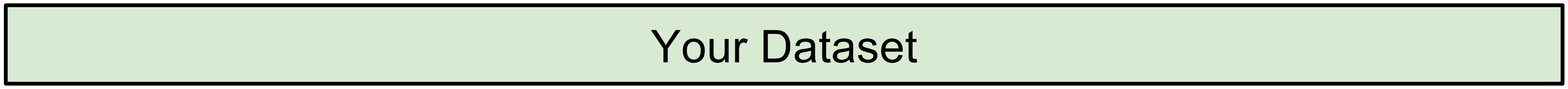
Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

train | test

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

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!

Acc

train	validation	test

K^* d^*

K d

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Hyperparameters

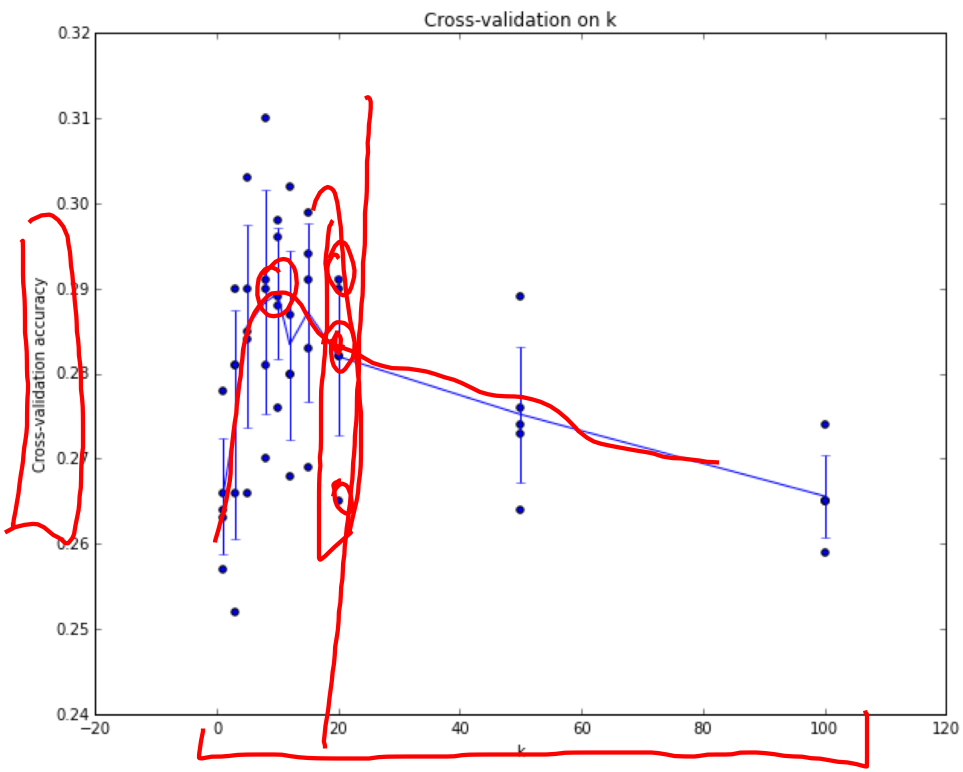
Your Dataset

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

fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

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

Setting Hyperparameters



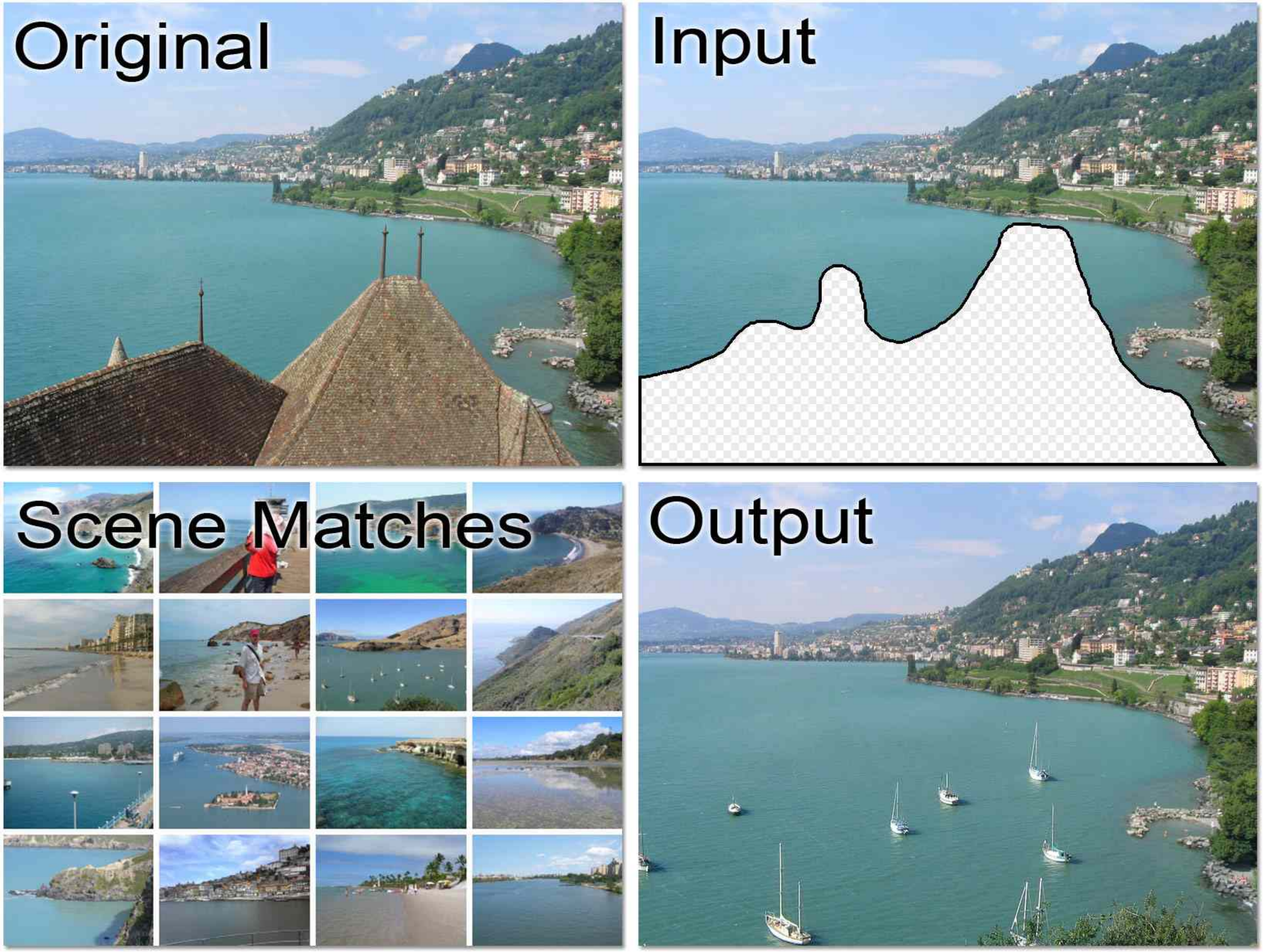
Example of
5-fold cross-validation
for the value of **k**.

Each point: single
outcome.

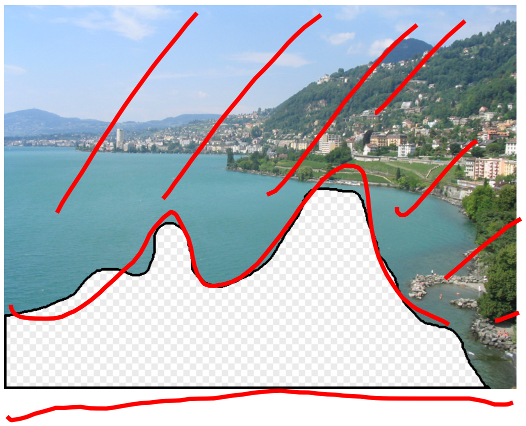
The line goes
through the mean, bars
indicated standard
deviation

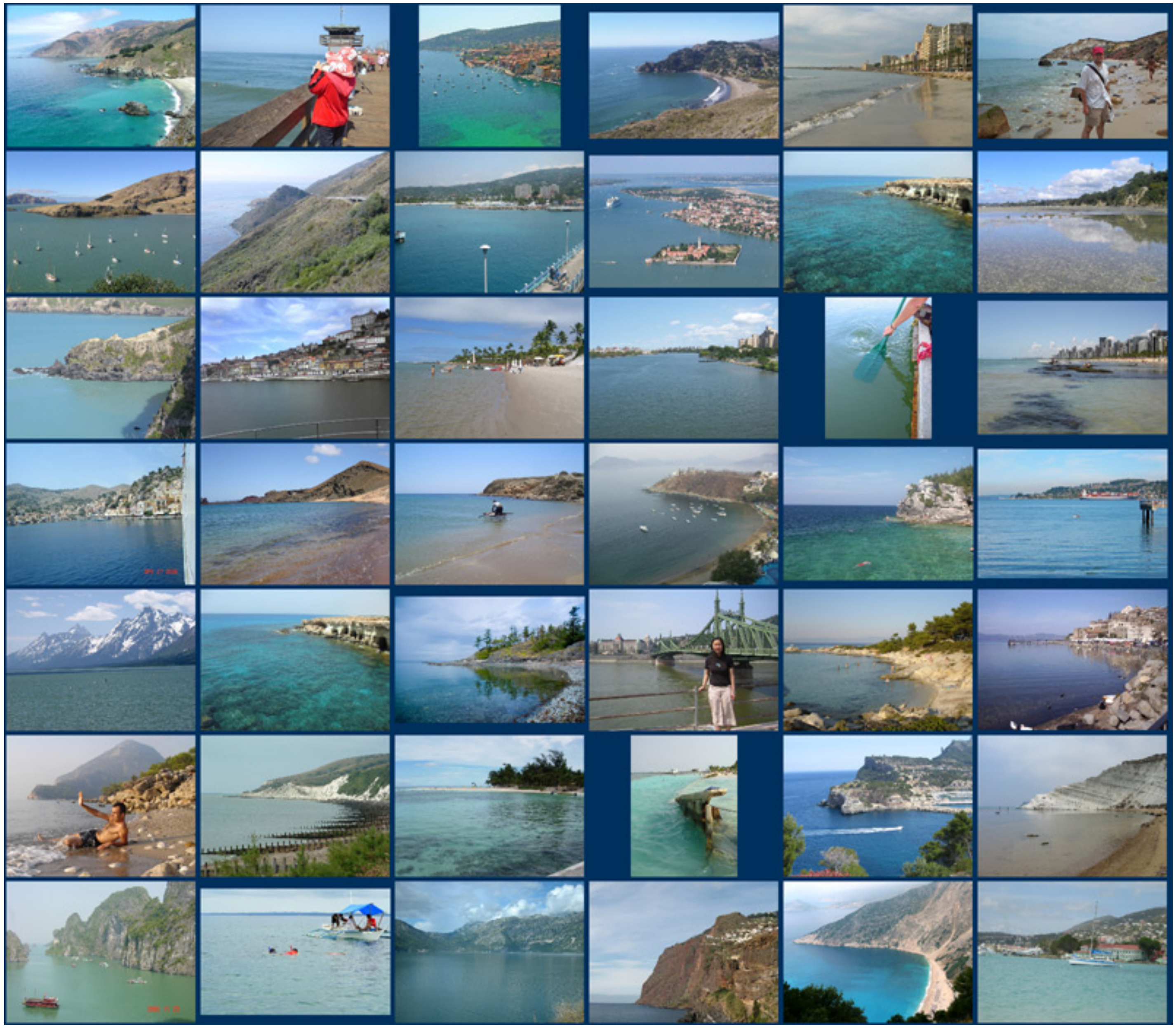
(Seems that k ~= 7 works best
for this data)

Scene Completion [Hayes & Efros, SIGGRAPH07]




Hays and Efros, SIGGRAPH 2007

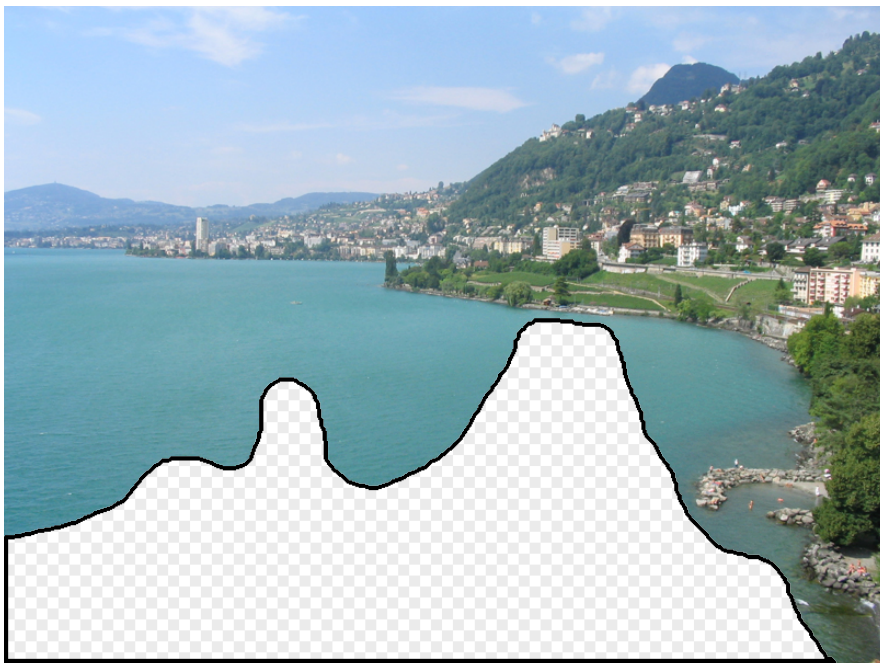




… 200 total

Hays and Efros, SIGGRAPH 2007

Context Matching



Hays and Efros, SIGGRAPH 2007

Graph cut + Poisson blending

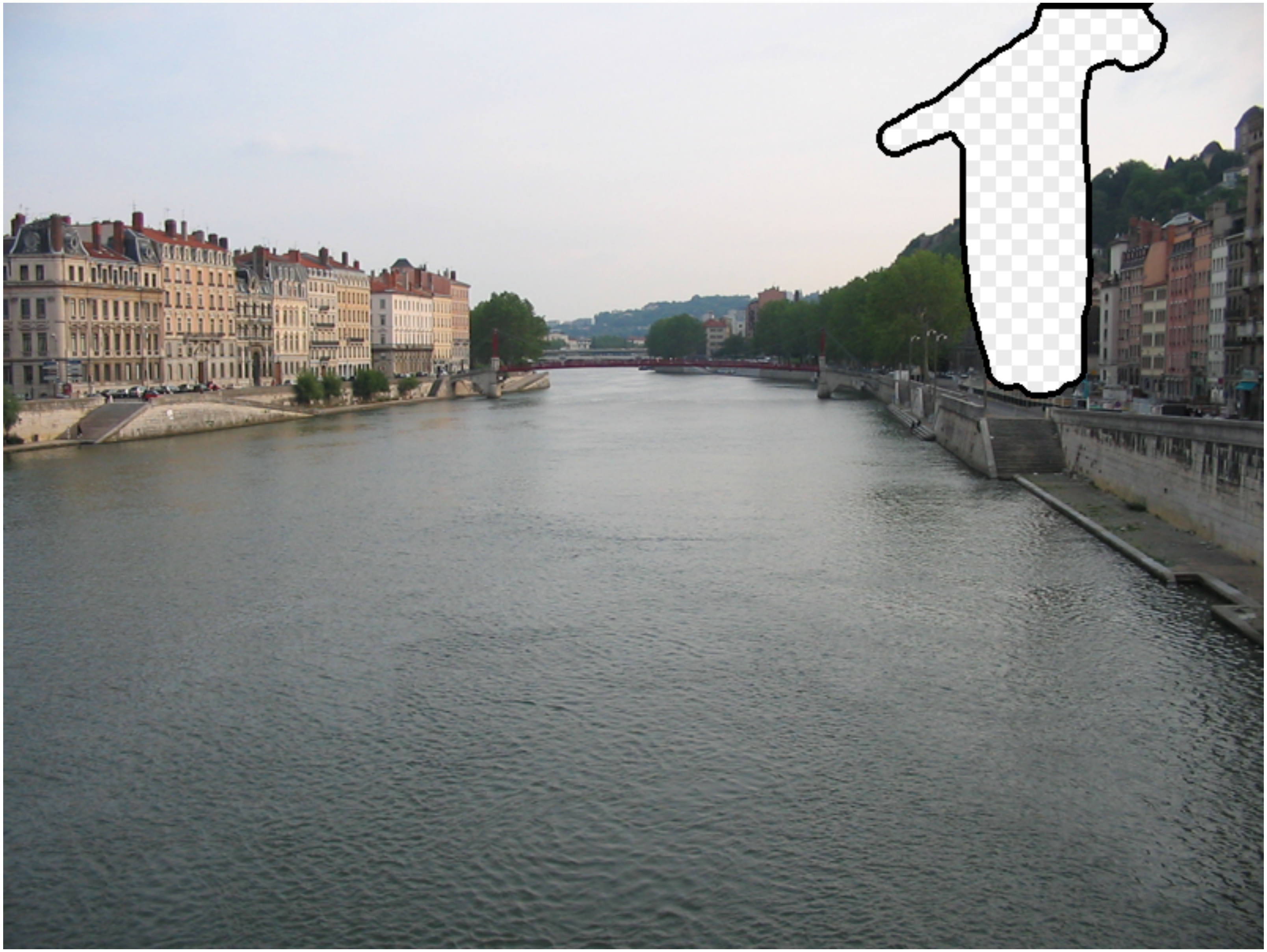
Hays and Efros, SIGGRAPH 2007

Hays and Efros, SIGGRAPH 2007

Hays and Efros, SIGGRAPH 2007

Hays and Efros, SIGGRAPH 2007

Hays and Efros, SIGGRAPH 2007



Hays and Efros, SIGGRAPH 2007

Hays and Efros, SIGGRAPH 2007

Problems with Instance-Based Learning

- Expensive
  - No Learning: most real work done during testing
  - For every test sample, must search through all dataset – very slow!
  - Must use tricks like approximate nearest neighbour search
- Doesn't work well when large number of irrelevant features
  - Distances overwhelmed by noisy features
- Curse of Dimensionality
  - Distances become meaningless in high dimensions
  - (See proof next)

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

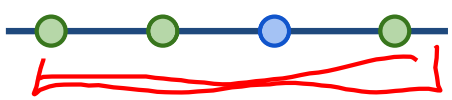
- Very slow at test time
- Distance metrics on pixels are not informative

Original | Boxed | Shifted | Tinted

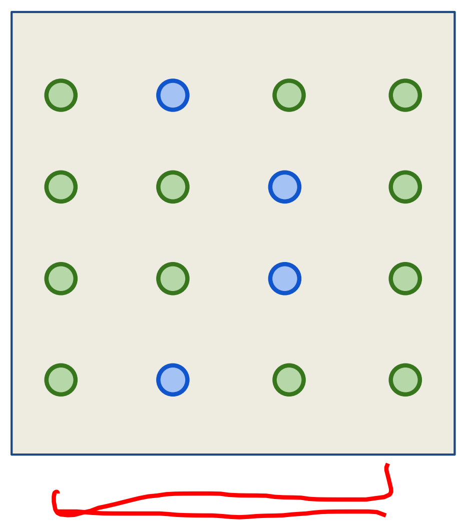
(all 3 images have same L2 distance to the one on the left) k-Nearest Neighbor on images **never used.**

- Curse of dimensionality

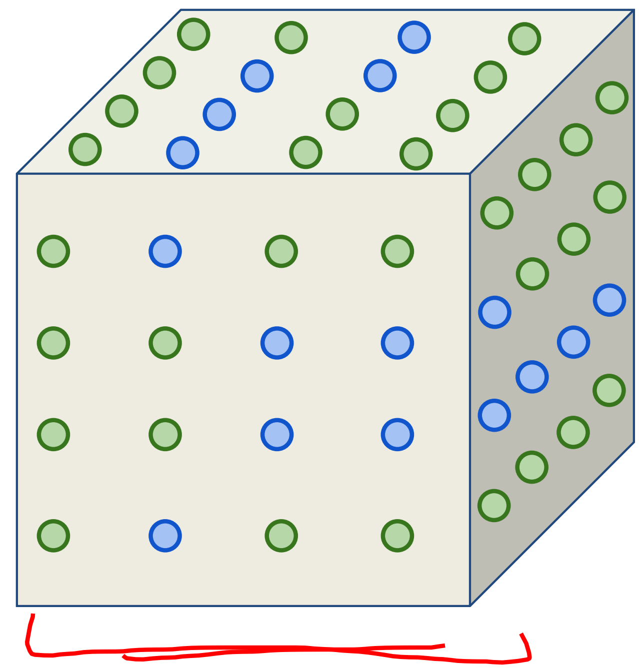
Dimensions = 1
Points = 4



Dimensions = 2
Points = 4^2



Dimensions = 3
Points = $\boxed{4^3}$



Curse of Dimensionality

- Consider: Sphere of radius 1 in d-dims
- Consider: an outer ε-shell in this sphere
- What is $\left[\dfrac{\textit{shell volume}}{\textit{sphere volume}}\right]$?

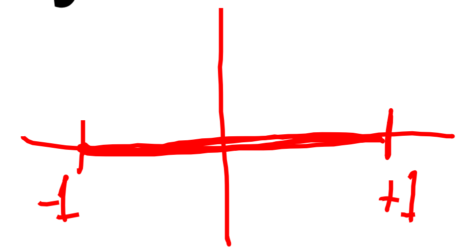
$2r$

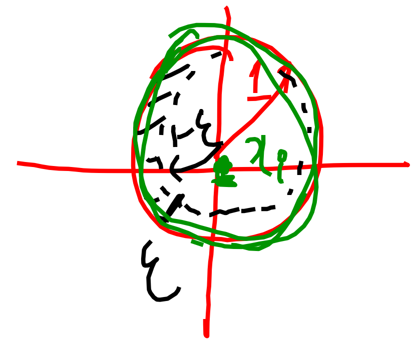
πr^2

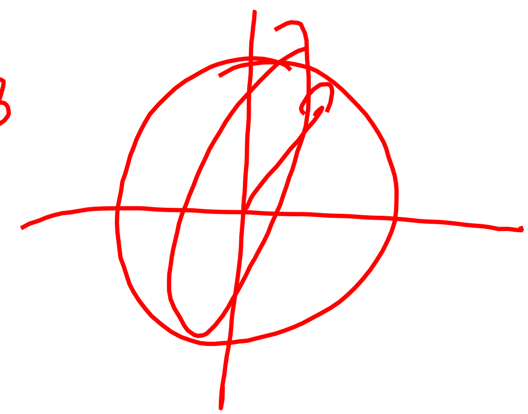
$\frac{4}{3}\pi r^3$

$\vdots$

$K_d r^d$







$$= \frac{K_d 1^d - K_d (1-\varepsilon)^d}{K_d 1^d}$$

$$\lim_{d \to \infty} = \underbrace{1}_{} - \underbrace{(1-\varepsilon)^d}_{} = \boxed{1}$$

Curse of Dimensionality

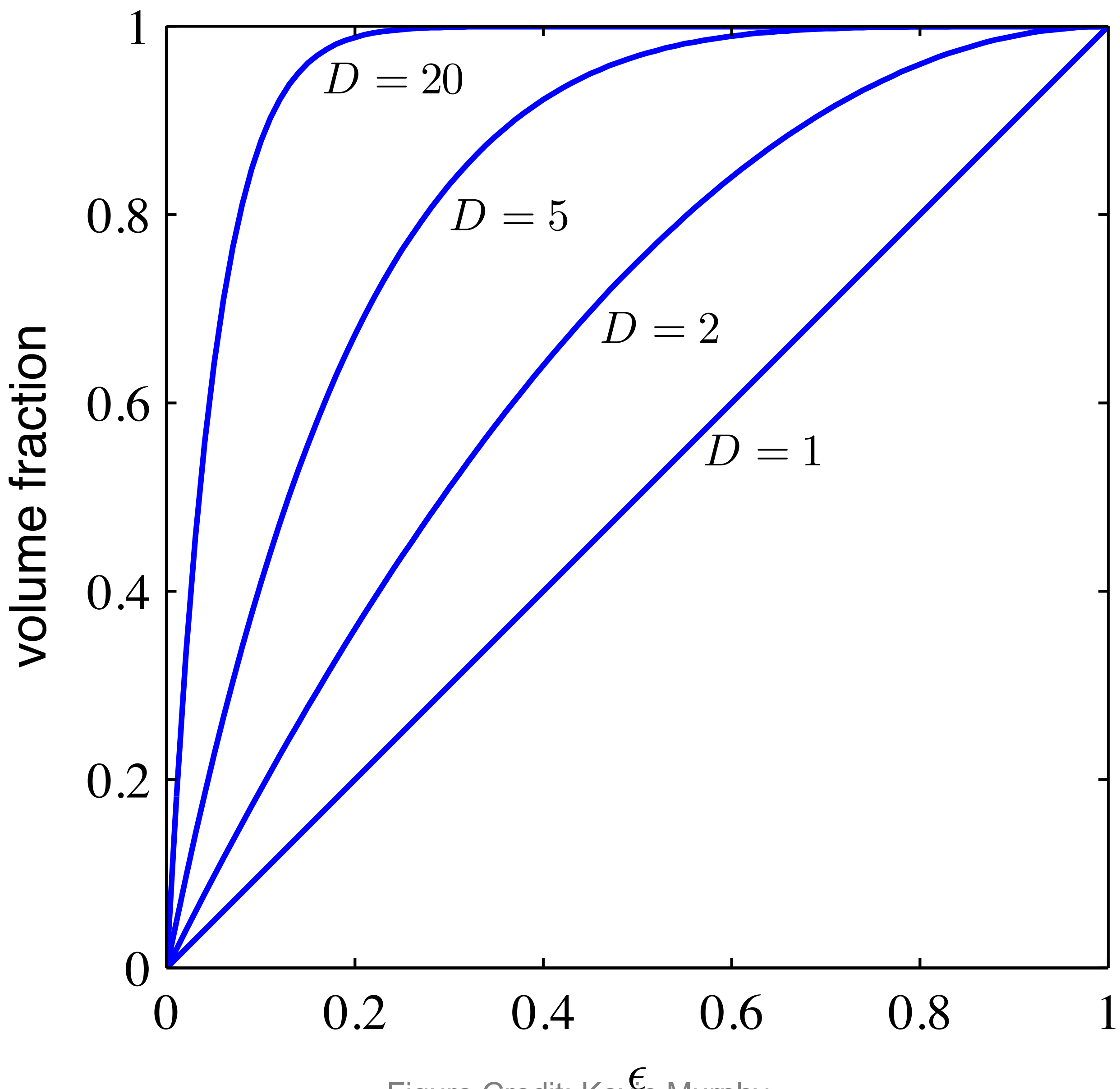


Figure Credit: Kevin Murphy

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!