CS 4803 / 7643: Deep Learning

Topics:

- Image Classification
- Supervised Learning view
- K-NN
- Linear Classifier

Zsolt Kira

Georgia Tech

Last Time

- High-level intro to what deep learning is

- Fast brief of logistics
  - Requirements: ML, math (linear algebra, calculus), programming (python)
  - Grades: 80% PS/HW, 20% Project, Piazza Bonus
  - Project: Topic of your choosing (related to DL), groups of 3-4 with separated undergrad/grad
  - 7 free late days
  - 1 week re-grading period
  - **No Cheating**

- PS0 out, due **Tuesday 01/14 11:55pm**
  - Graded pass/fail
  - Intended to do on your own
  - Don't worry if rusty! It's OK to need a refresher on various subjects to do it. Some of it (e.g. last question) is more suitable for graduate students.
  - If not registered, email staff for gradescope account

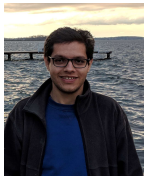
- Look through slides on website for all details

Current TAs

Sameer Dharur

MS-CS student

https://www.linkedin.com/in/sameerdharur/



Rahul Duggal

2nd year CS PhD student

http://www.rahulduggal.com/

Patrick Grady

2nd year Robotics PhD student

https://www.linkedin.com/in/patrick-grady

Anishi Mehta

MSCS student

https://www.linkedin.com/in/anishimehta

Yinquan Lu

2nd year MSCSE student

https://www.cc.gatech.edu/~jyang462/

Jiachen Yang

2nd year ML PhD

https://www.cc.gatech.edu/~jyang462/

- **New TAs:** Zhuoran Yu. Manas Sahni, (in process) Harish Kamath
- Official office hours coming soon (TA and instructor)
- For this & next week:
  - 11:30am-12:30pm **Friday 01/09** (Zhuoan Yu)
  - 11:30-12:30am on **Monday** (Patrick)
  - 11:30 AM to 12:30 PM on **Tuesdays**. (Sameer)
  - 4-5pm **Tuesday** (Jiachen)
  - 1:30-2:30 pm on **Wed**. (Anishi)
  - 11:30 AM to 12:30 PM on **Thursdays**. (Rahul)

Registration/Access

- Waitlist
  - Still a large waitlist for grad, still adding some capacity
- Canvas
  - Anybody not have access?
- Piazza
  - 110+ people signed up. Please use that for questions.

Website: http://www.cc.gatech.edu/classes/AY2020/cs7643_spring/

Piazza: https://piazza.com/gatech/spring2020/cs4803dl7643a/

Staff mailing list (personal questions): cs4803-7643-staff@lists.gatech.edu

Gradescope: https://www.gradescope.com/courses/78537

Canvas: https://gatech.instructure.com/courses/94450/

Course Access Code (Piazza): MWXKY8

Prep for HW1: Python+Numpy Tutorial

CS231n Convolutional Neural Networks for Visual Recognition

Python Numpy Tutorial

This tutorial was contributed by Justin Johnson.

We will use the Python programming language for all assignments in this course. Python is a great general-purpose programming language on its own, but with the help of a few popular libraries (numpy, scipy, matplotlib) it becomes a powerful environment for scientific computing.

We expect that many of you will have some experience with Python and numpy; for the rest of you, this section will serve as a quick crash course both on the Python programming language and on the use of Python for scientific computing.

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

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

Plan for Today

- Reminder:
  - What changed to enable DL
- Some Problems with DL
- Image Classification
- Supervised Learning view
- K-NN
- (Beginning of) Linear Classifiers

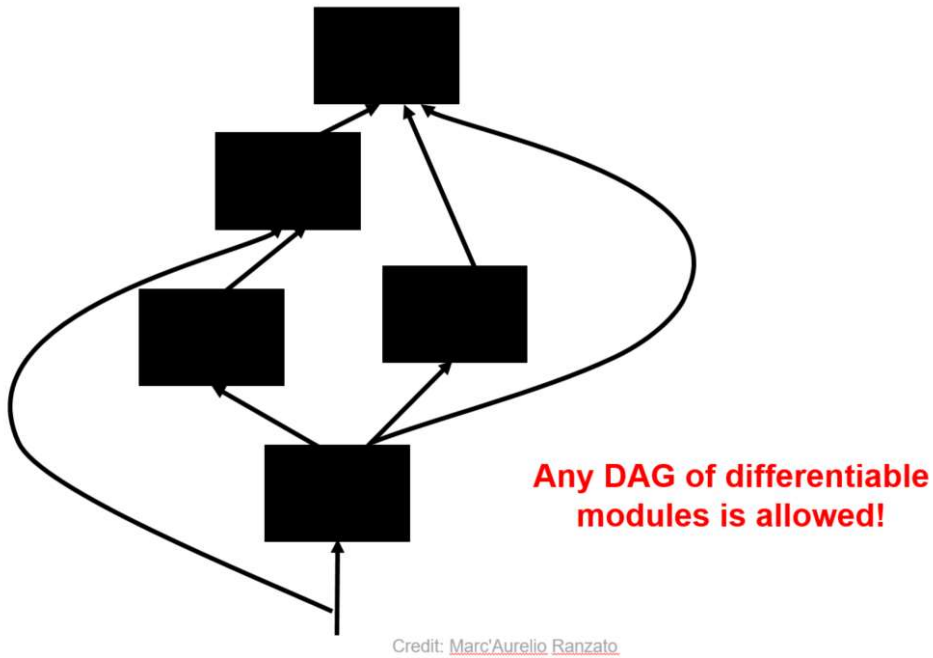
Reminder: What Deep Learning Is

- We will learn a complex non-linear **hierarchical** (compositional) function in an **end-to-end manner**

- (Hierarchical) Compositionality
  - Cascade of non-linear transformations
  - Multiple layers of representations

- End-to-End Learning
  - Learning (goal-driven) representations
  - Learning to feature extraction

- Distributed Representations
  - No single neuron "encodes" everything
  - Groups of neurons work together



What Changed?

- Few people saw this combination coming: **gigantic growth in data and processing** to enable **depth and feature learning**
  - Combined with specialized hardware (**gpus**) and open-source/distribution (**arXiv, github**)

- If the input features are poor, so will your result be
  - If your model is poor, so will your result be
    - If your optimizer is poor, so will your result be

- Now we have methods for feature learning that works (after some finesse)
  - Still have to guard against overfitting (very complex functions!)
  - Still tune hyper-parameters
  - Still design neural network architectures
  - Lots of research to automate this too, e.g. via reinforcement learning!

Problems with Deep Learning

- Problem#1: Non-Convex! Non-Convex! Non-Convex!
  - Depth>=3: most losses non-convex in parameters
  - Theoretically, all bets are off
  - Leads to stochasticity
    - different initializations → different local minima

- Standard response #1
  - "Yes, but all interesting learning problems are non-convex"
  - For example, human learning
    - Order matters → wave hands → non-convexity

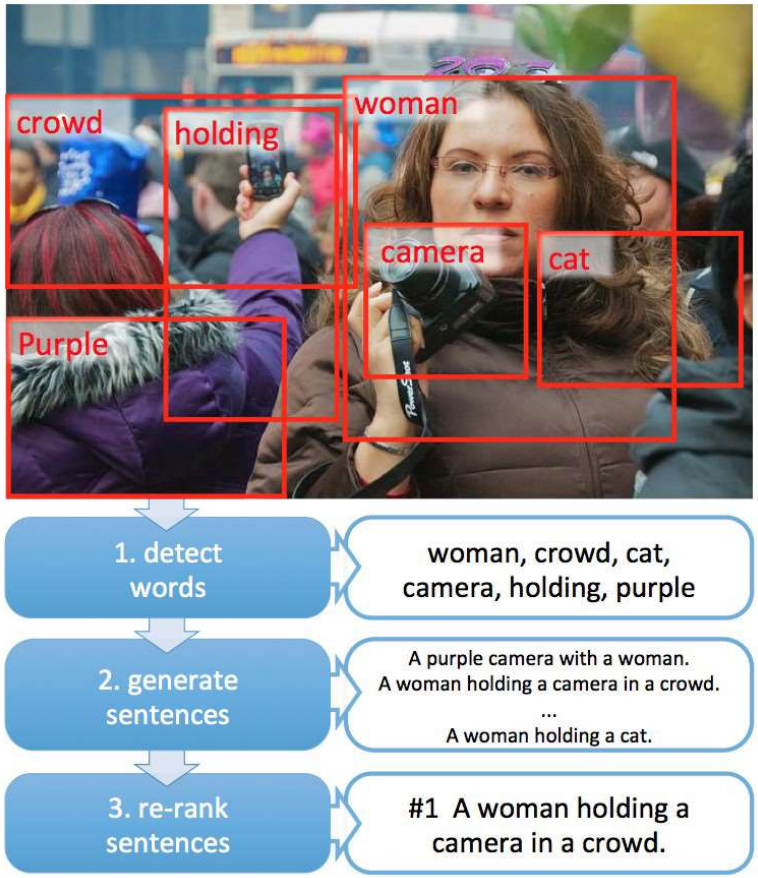
- Standard response #2
  - "Yes, but it often works!"

Problems with Deep Learning

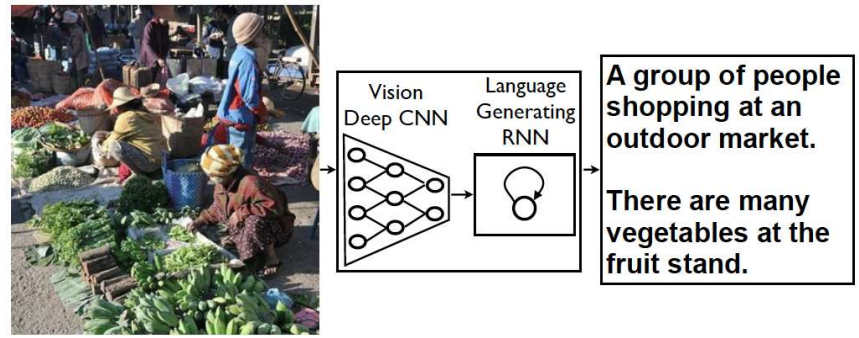
- Problem#2: Lack of interpretability
  - Hard to track down what's failing
  - Pipeline systems have "oracle" performances at each step
  - In end-to-end systems, it's hard to know why things are not working

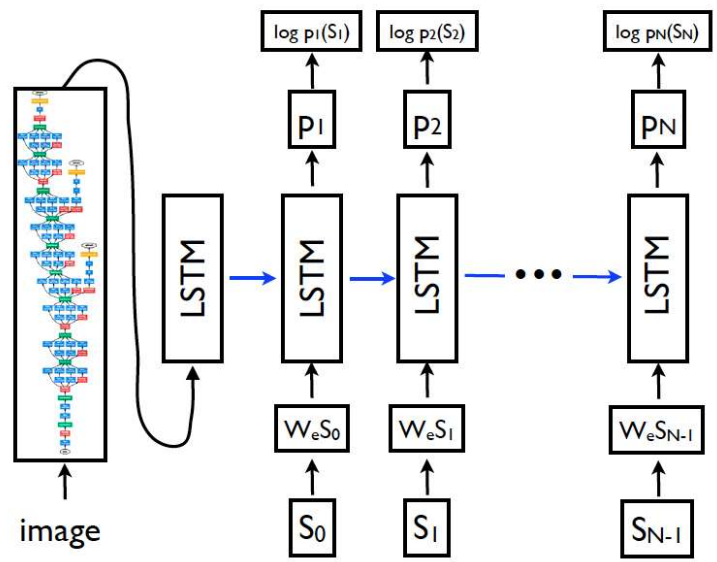
Problems with Deep Learning

- Problem#2: Lack of interpretability



[Fang et al. CVPR15]





[Vinyals et al. CVPR15]

Pipeline ⟷ End-to-End

Problems with Deep Learning

- Problem#2: Lack of interpretability
  - Hard to track down what's failing
  - Pipeline systems have "oracle" performances at each step
  - In end-to-end systems, it's hard to know why things are not working

- Standard response #1
  - Tricks of the trade: visualize features, add losses at different layers, pre-train to avoid degenerate initializations…
  - "We're working on it"

- Standard response #2
  - "Yes, but it often works!"

Problems with Deep Learning

- Problem#3: Lack of easy reproducibility
  - Direct consequence of stochasticity & non-convexity

- Standard response #1
  - It's getting much better
  - Standard toolkits/libraries/frameworks now available
  - Caffe, Theano, (Py)Torch

- Standard response #2
  - "Yes, but it often works!"

Yes it works, but how?

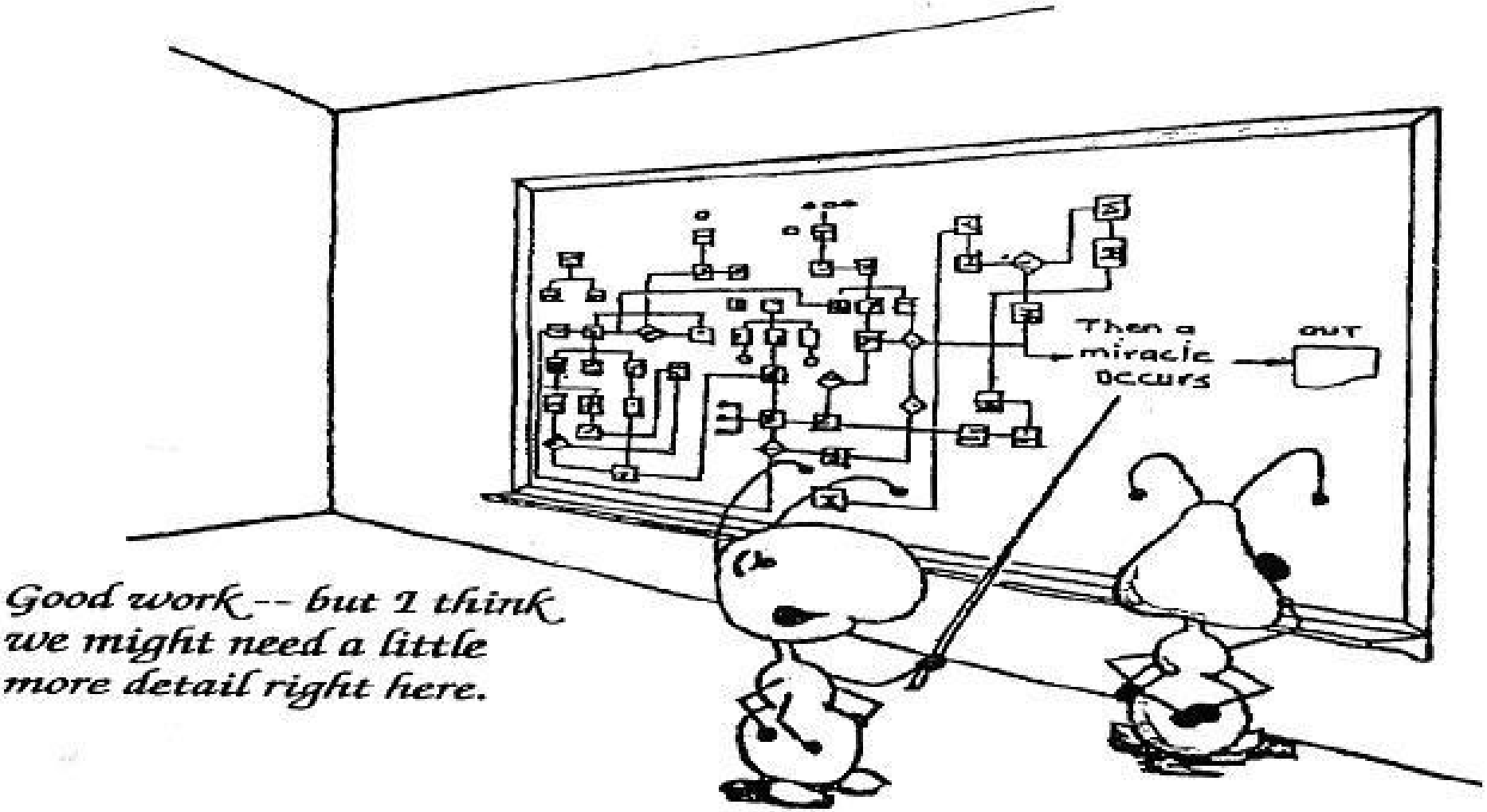


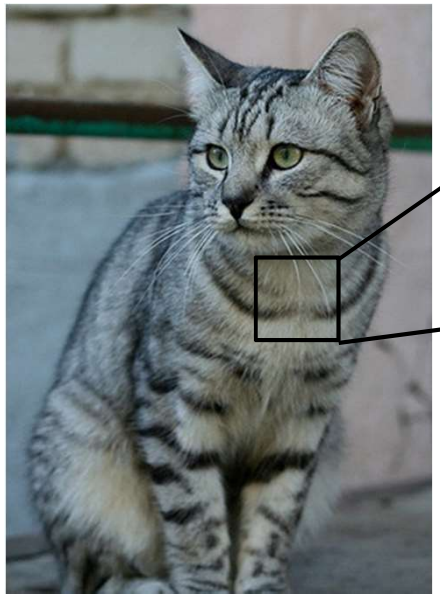
Image Classification

Image Classification: A core task in Computer Vision

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

→ cat

The Problem: Semantic Gap




```
[[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]]
```

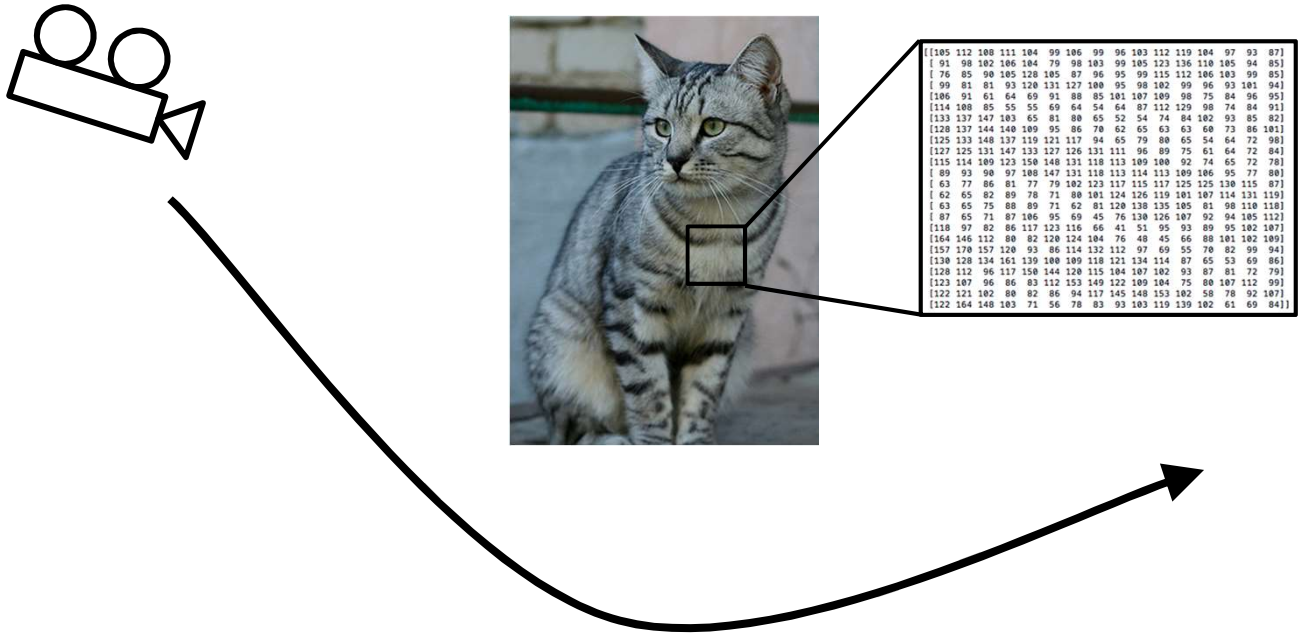
What the computer sees

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

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

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

Challenges: Viewpoint variation



All pixels change when the camera moves!

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

Challenges: Illumination

This image is CC0 1.0 public domain

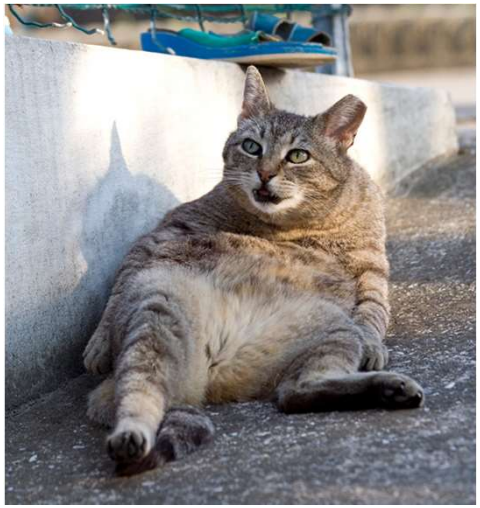
This image is CC0 1.0 public domain

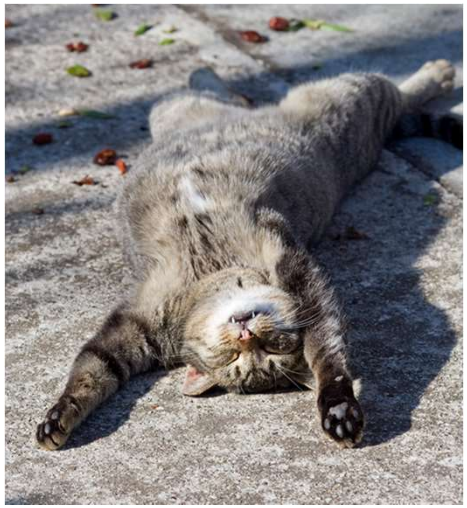
This image is CC0 1.0 public domain

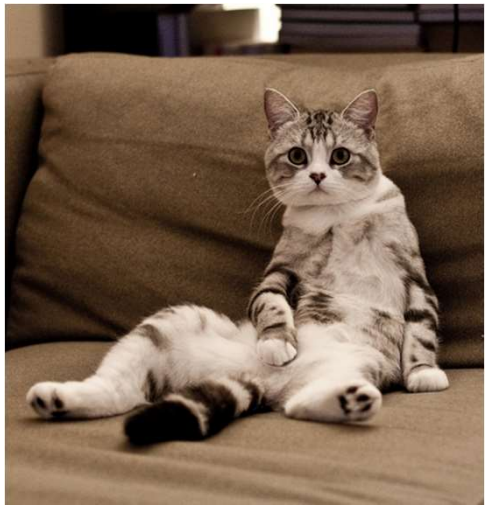
This image is CC0 1.0 public domain

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

Challenges: Deformation







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

Challenges: Occlusion

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

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
```

Unlike e.g. sorting a list of numbers,

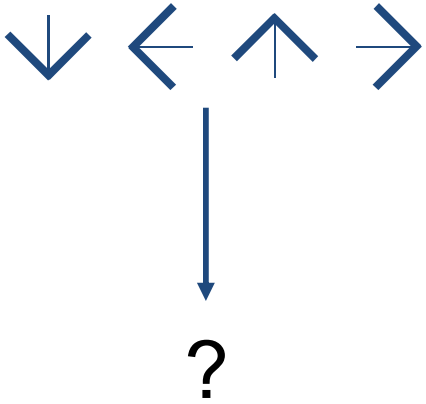
no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

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

Attempts have been made

Find edges

Find corners



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

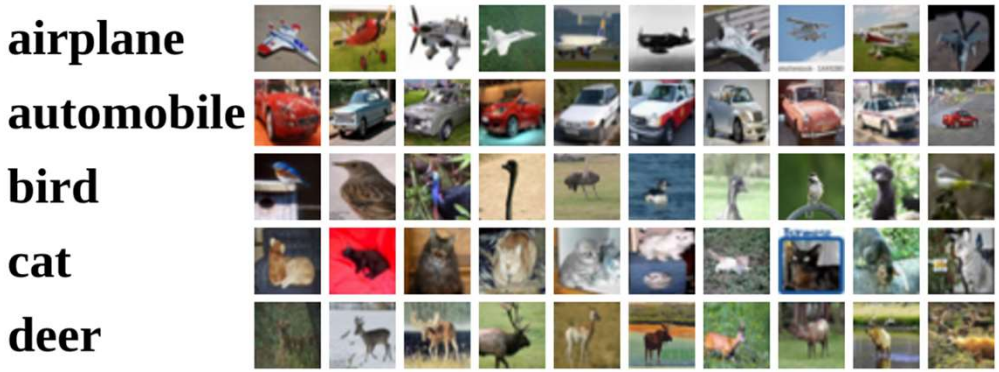
ML: A Data-Driven Approach

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



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

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: X → Y
  - $y = h(x) = \text{sign}(w^T x)$
- Learning = Search in hypothesis space
  - Find best h in model class.

Procedural View

- Training Stage:
  - Training Data { (x,y) } → f (Learning)

- Testing Stage
  - Test Data x → f(x) (Apply function, Evaluate error)

Statistical Estimation View

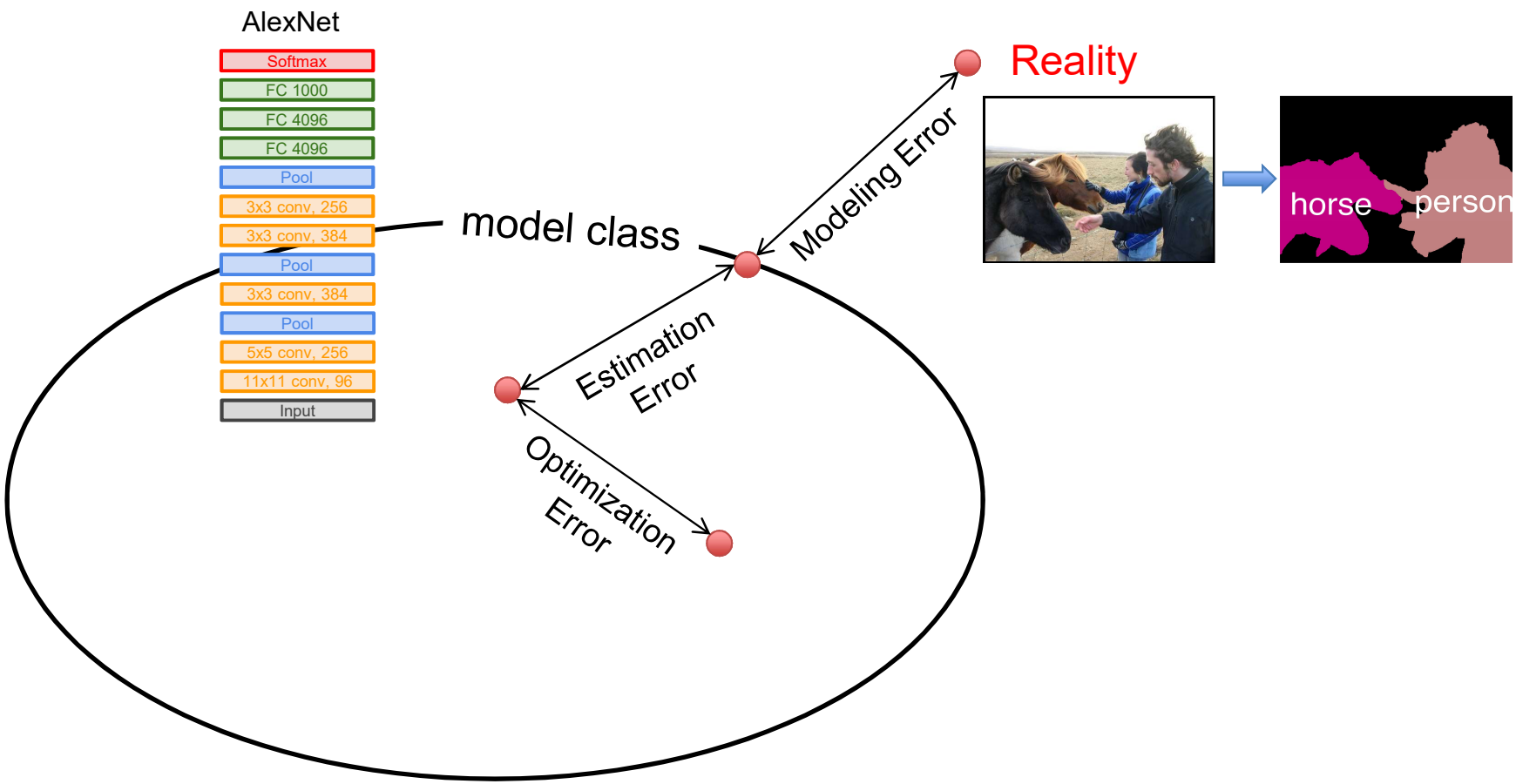
- Probabilities to rescue:
  - X and Y are *random variables*
  - D = $(x_1,y_1), (x_2,y_2), \ldots, (x_N,y_N)$ ~ P(X,Y)

- 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

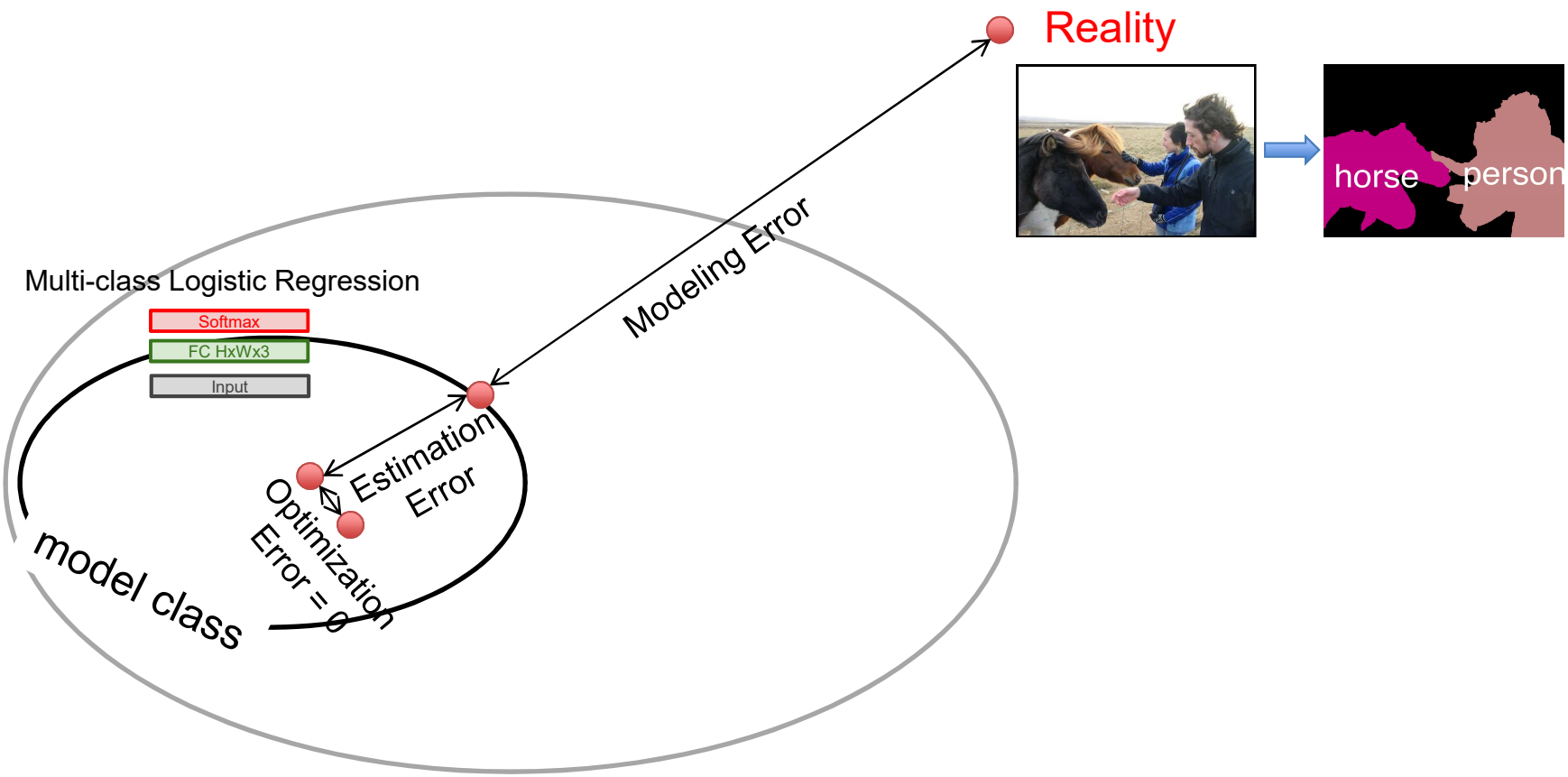
Error Decomposition

- Reality

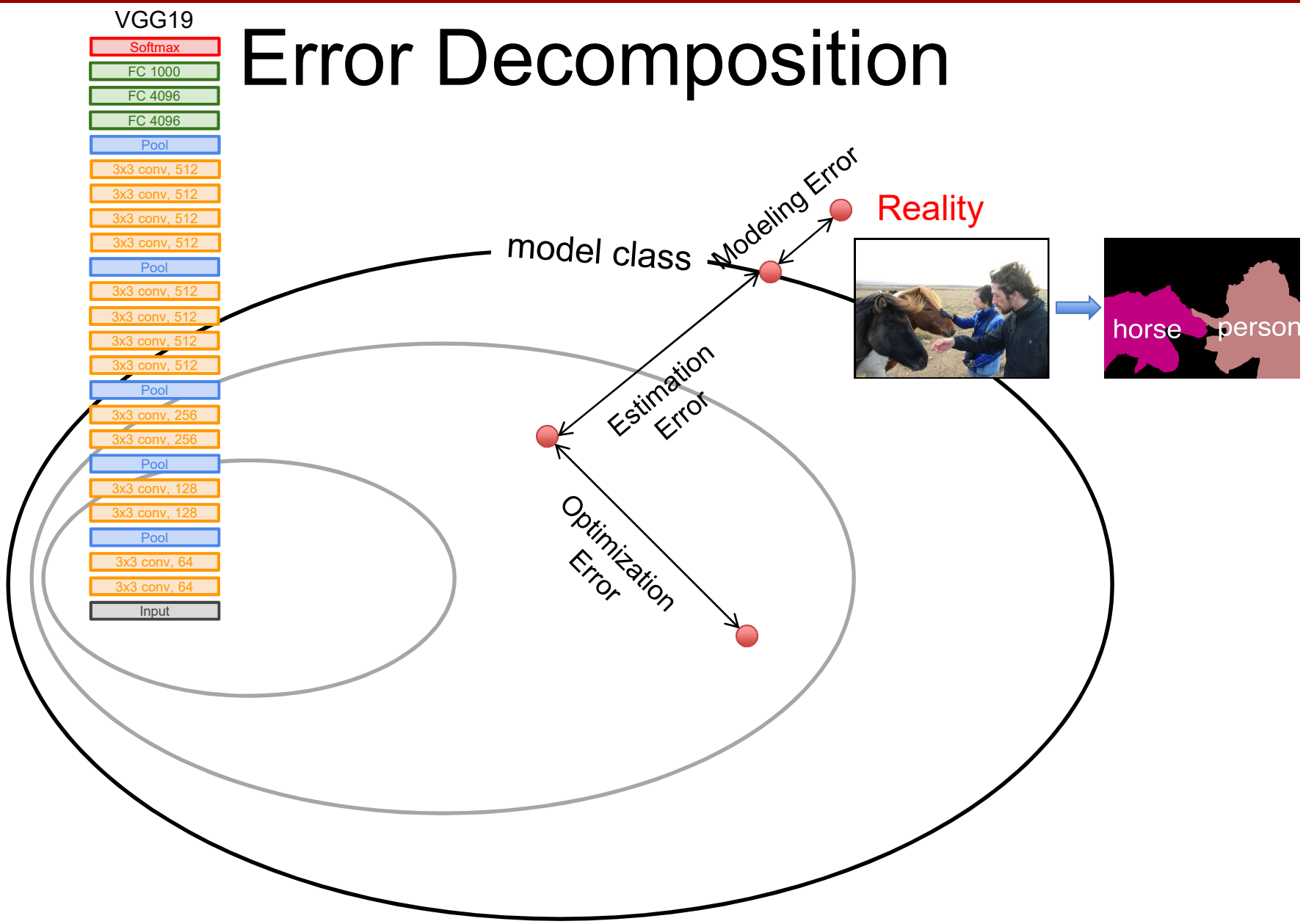
Error Decomposition



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

Guarantees

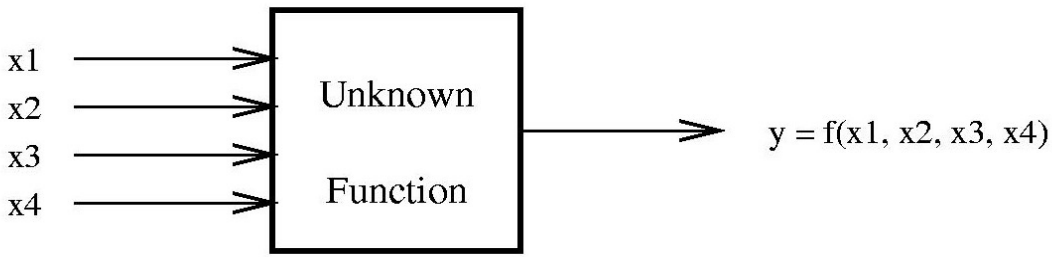
- 20 years of research in Learning Theory oversimplified:

- If you have:
  - Enough training data D
  - and H is not too complex
  - then *probably* we can generalize to unseen test data

- Note: Several ways to measure complexity
  - Vapnik–Chervonenkis dimension
  - Rademacher complexity

Learning is hard!

A Learning Problem

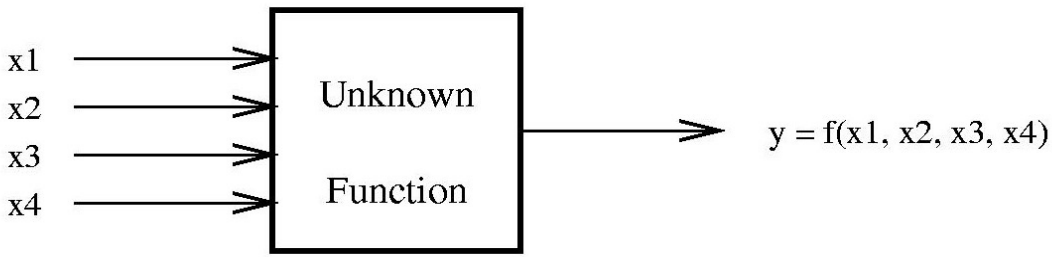


Example	x_1	x_2	x_3	x_4	y
1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	1	1	1
4	1	0	0	1	1
5	0	1	1	0	0
6	1	1	0	0	0
7	0	1	0	1	0

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

First classifier: **Nearest Neighbor**

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

→ Memorize all data and labels

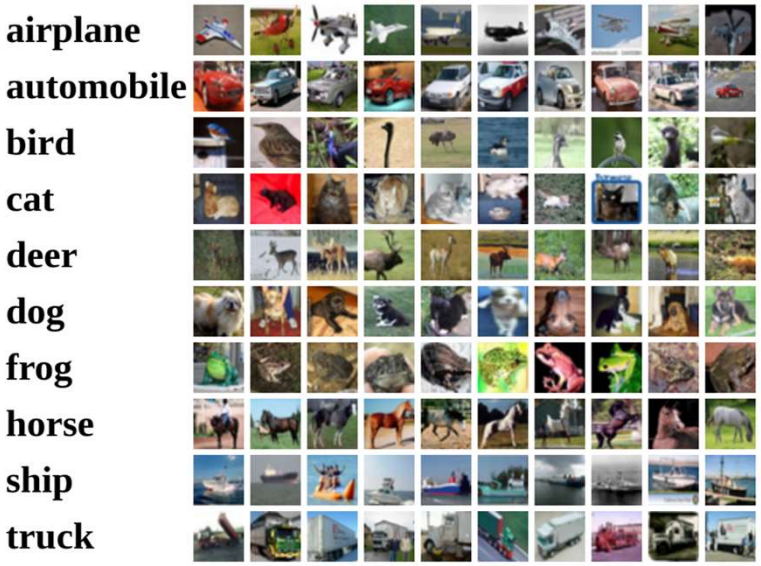
```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

→ Predict the label of the most similar training image

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

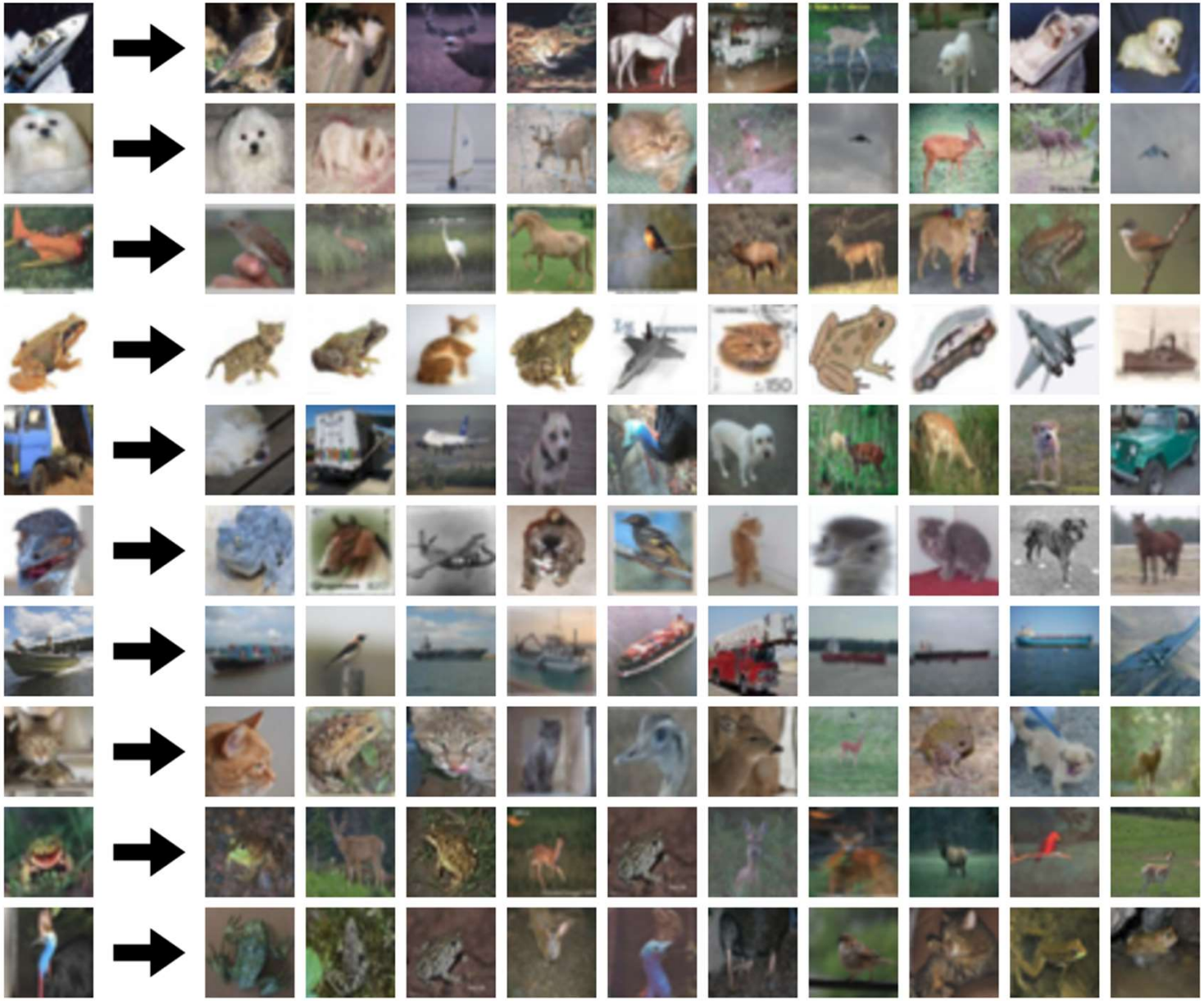
Example Dataset: **CIFAR10**

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

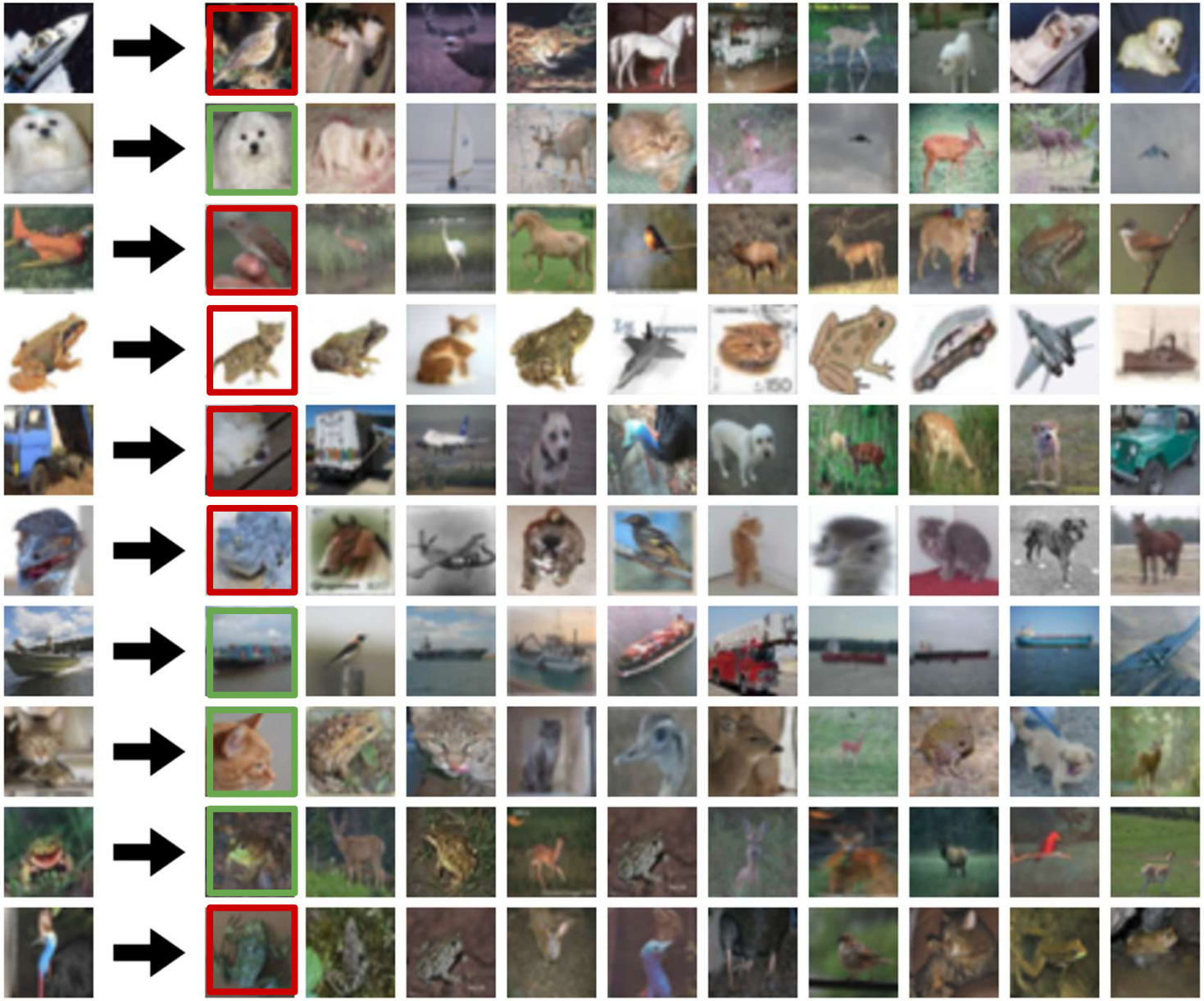


Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, Technical Report, 2009.

Nearest Neighbours



Nearest Neighbours



Instance/Memory-based Learning

Four things make a memory based learner:

- *A distance metric*
- *How many nearby neighbors to look at?*
- *A weighting function (optional)*
- *How to fit with the local points?*

1-Nearest Neighbour

Four things make a memory based learner:

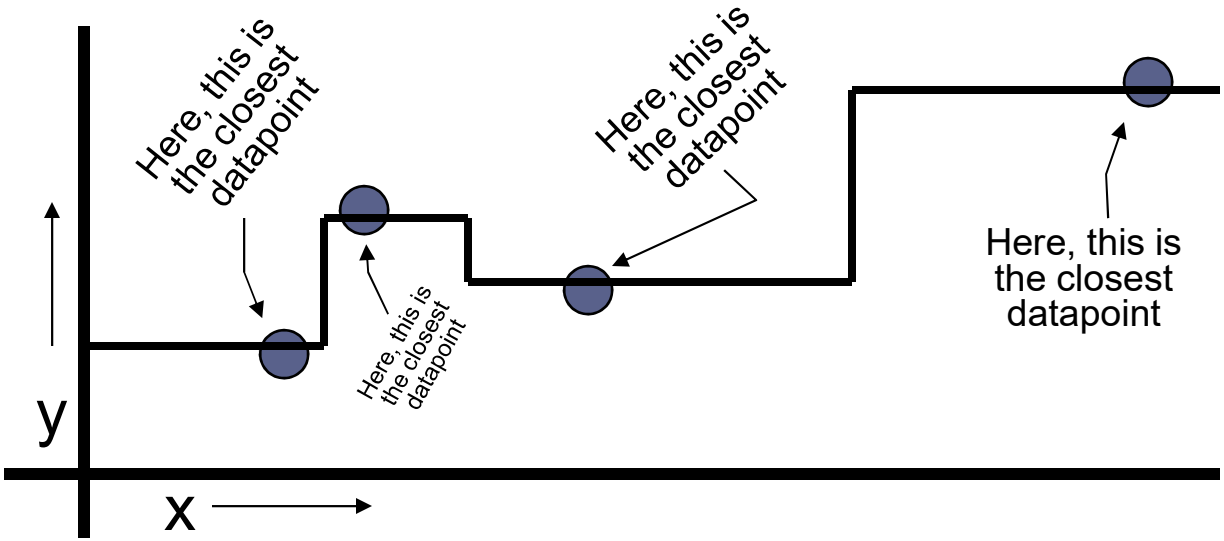
- *A distance metric*
  - **Euclidean (and others)**
- *How many nearby neighbors to look at?*
  - **1**
- *A weighting function (optional)*
  - **unused**
- *How to fit with the local points?*
  - **Just predict the same output as the nearest neighbour.**

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

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

$\xrightarrow{\text{add}}$ 456

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

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

Memorize training data

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

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

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?

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)

```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

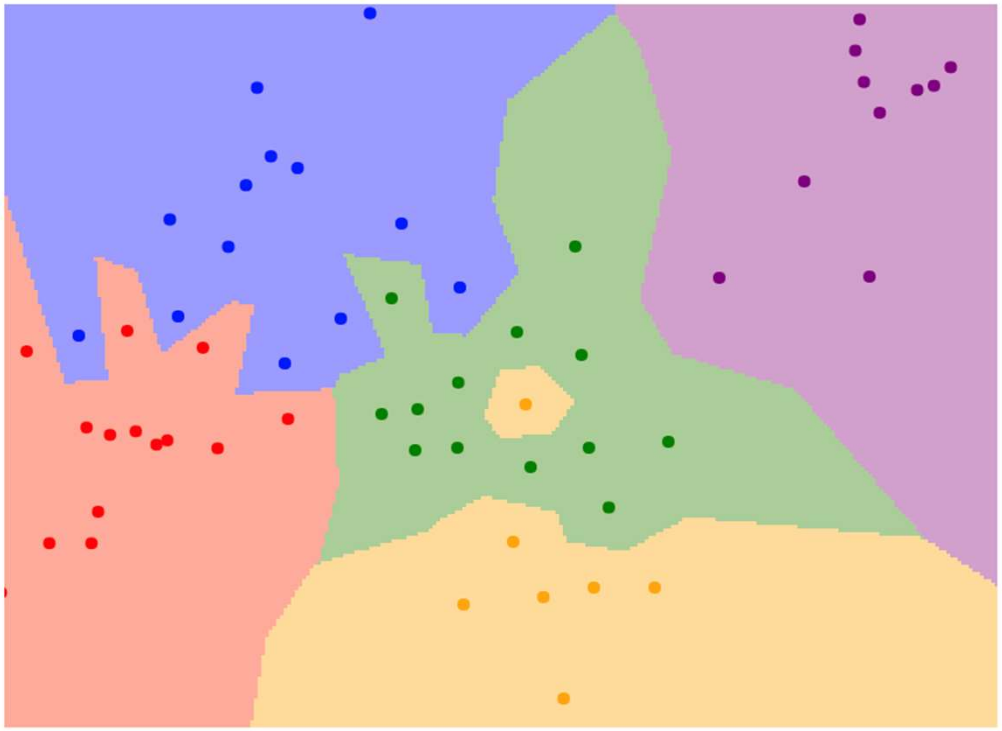
Q: With N examples, how fast are training and prediction?

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

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

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

What does this look like?



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

Nearest Neighbour

- Demo 1
  - http://vision.stanford.edu/teaching/cs231n-demos/knn/
- Demo 2
  - http://www.cs.technion.ac.il/~rani/LocBoost/

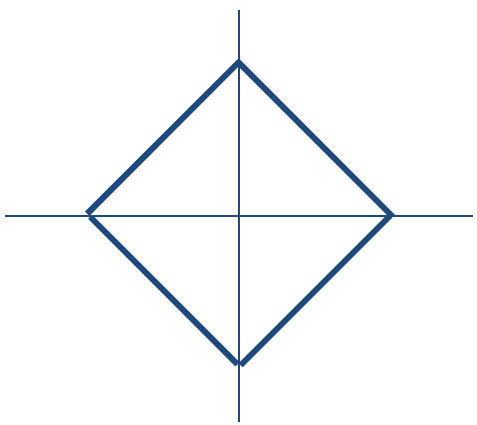
Parametric vs Non-Parametric Models

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

K-Nearest Neighbors: Distance Metric

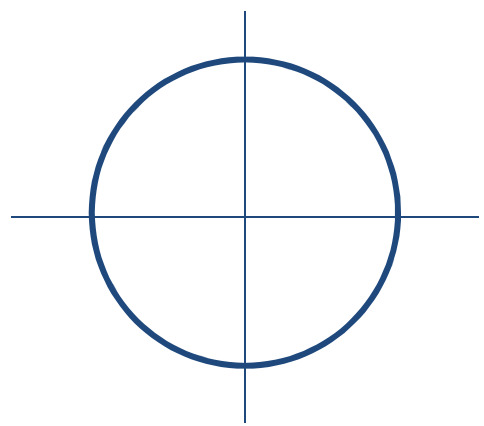
L1 (Manhattan) distance

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



L2 (Euclidean) distance

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

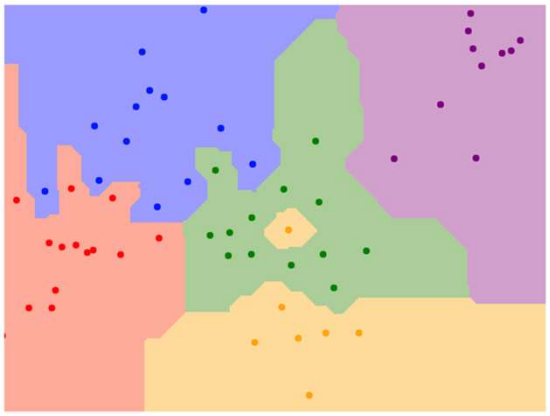


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

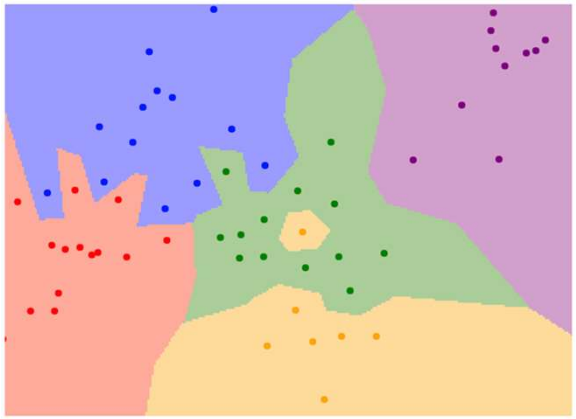
$$d_1(I_1, I_2) = \sum_p \left| I_1^p - I_2^p \right|$$



K = 1

L2 (Euclidean) distance

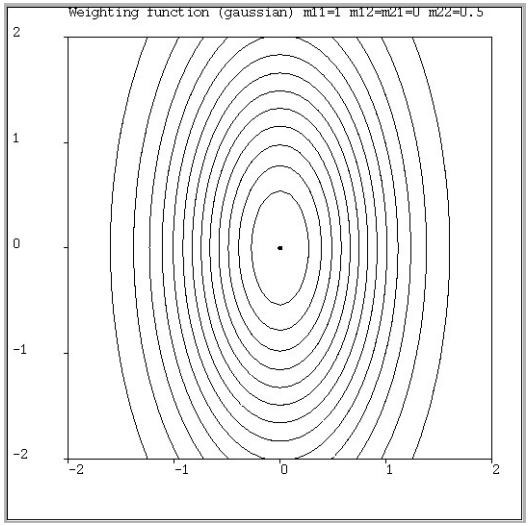
$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p \right)^2}$$



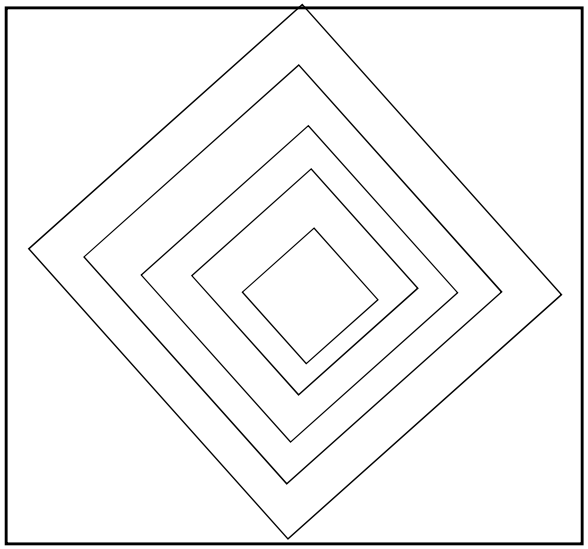
K = 1

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

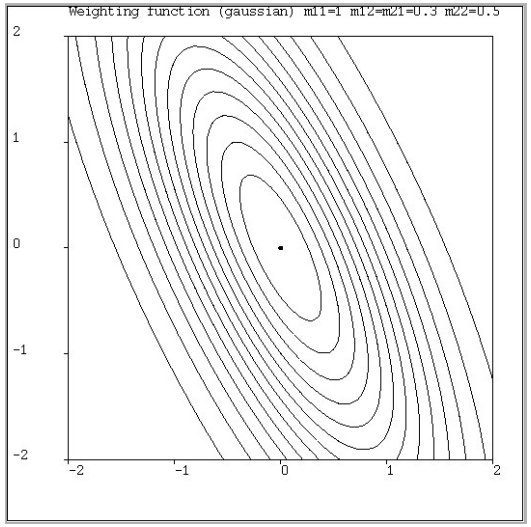
K-Nearest Neighbors: Distance Metric



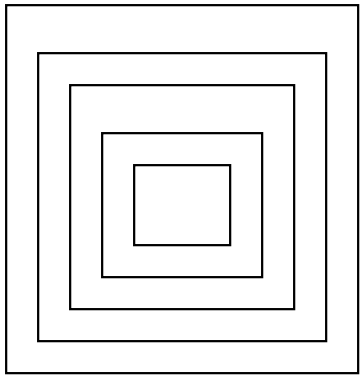
Scaled Euclidian (L_2)



L_1 norm (absolute)



Mahalanobis



$L_{infinity}$ *(max) norm*

More Distance Metrics

$$D(\mathrm{x},\mathrm{x}') = \sqrt{\sum_i S_i^{-1}\left(x_i - x'_i\right)^2}$$

Or equivalently,

$$D(\mathrm{x},\mathrm{x}') = \sqrt{(\mathrm{x}-\mathrm{x}')^T S^{-1}(\mathrm{x}-\mathrm{x}')}$$

where

$$S = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \sigma_N^2 \end{bmatrix}$$

Other Metrics…

- Mahalanobis, Rank-based, Correlation-based (Stanfill+Waltz, Maes' Ringo system…)

Slide by Andrew W. Moore

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

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

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

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

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

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

Hyperparameters

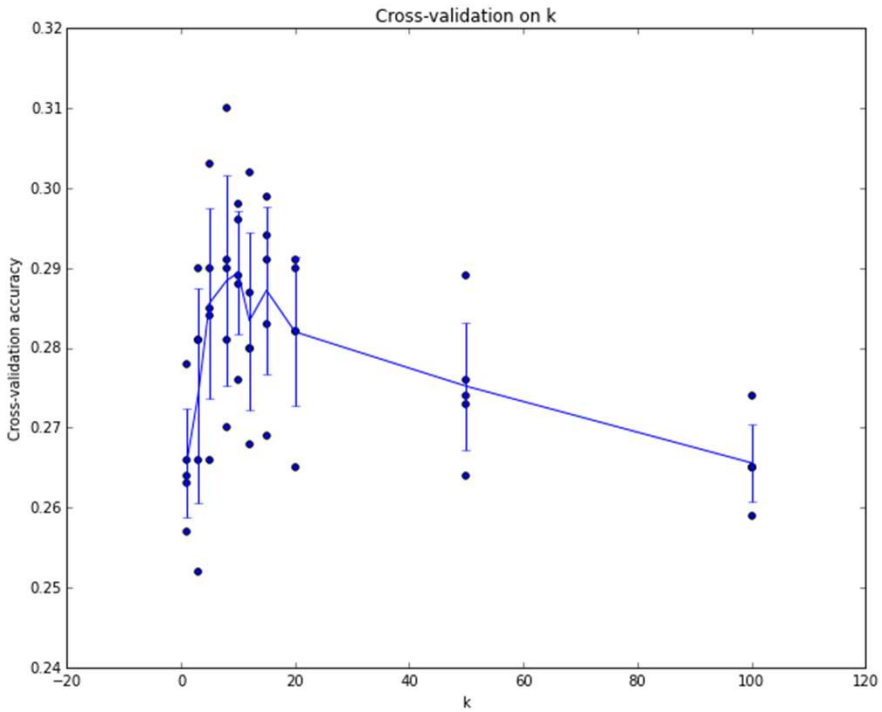
Your Dataset

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



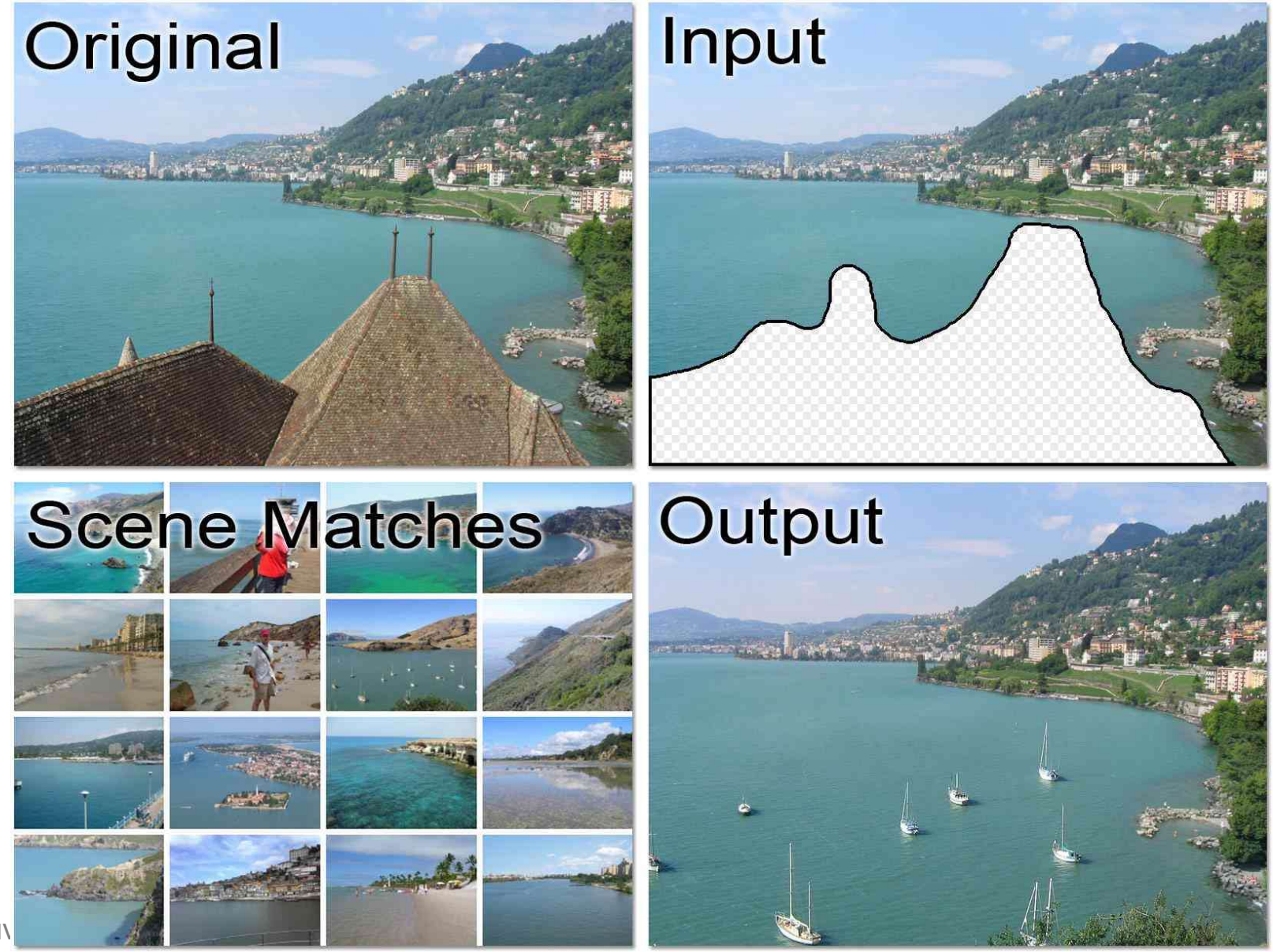
Example of
5-fold cross-validation
for the value of **k.**

Each point: single
outcome.

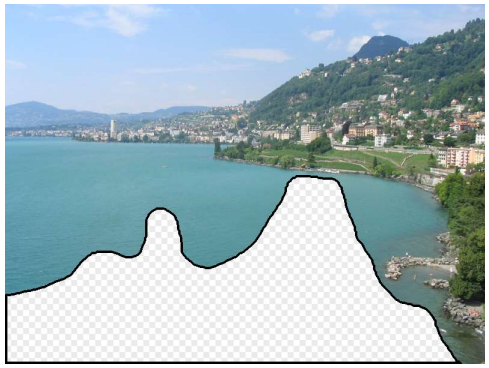
The line goes
through the mean, bars
indicated standard
deviation

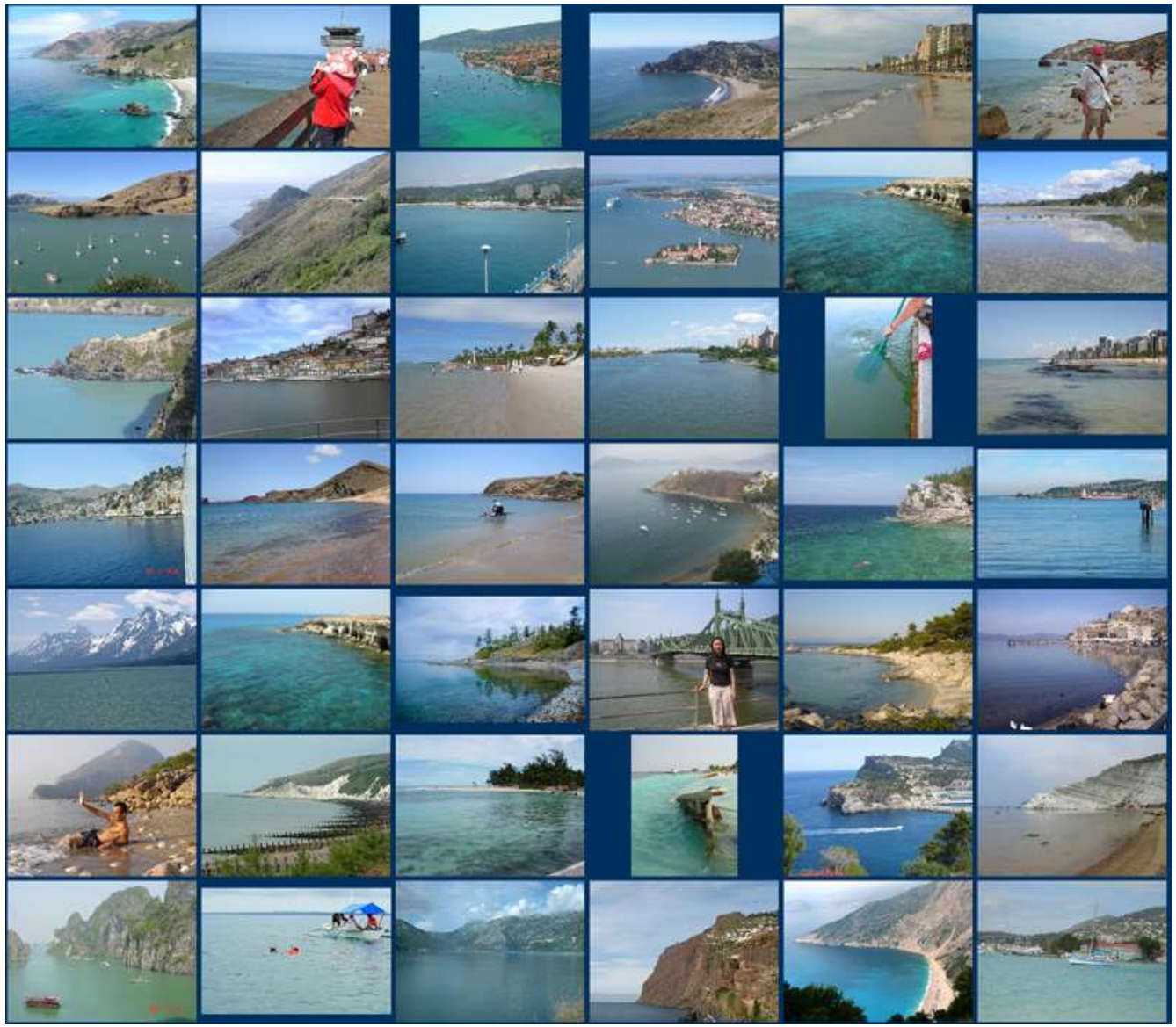
(Seems that k ~= 7 works best
for this data)

Scene Completion [Hayes & Efros, SIGGRAPH07]




Hays and Efros, SIGGRAPH 2007

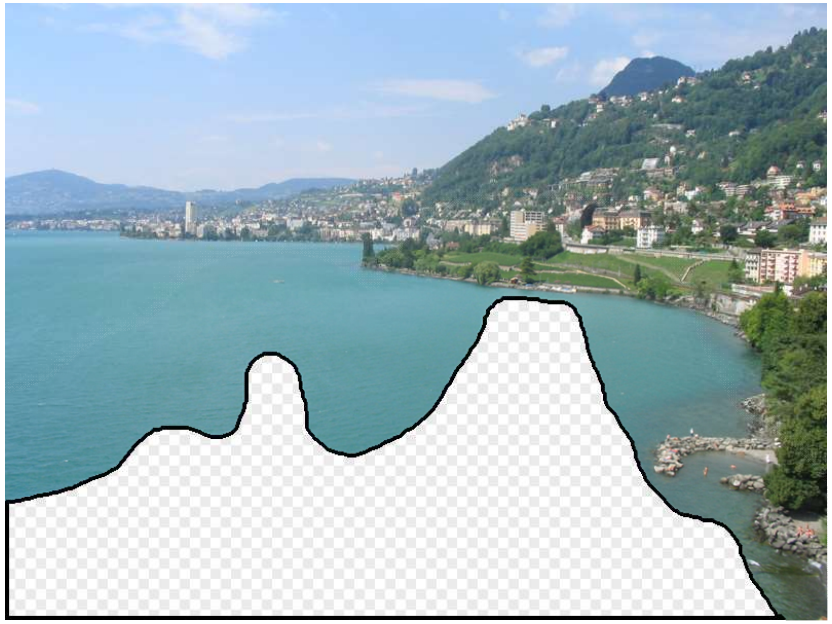




… 200 total

Hays and Efros, SIGGRAPH 2007

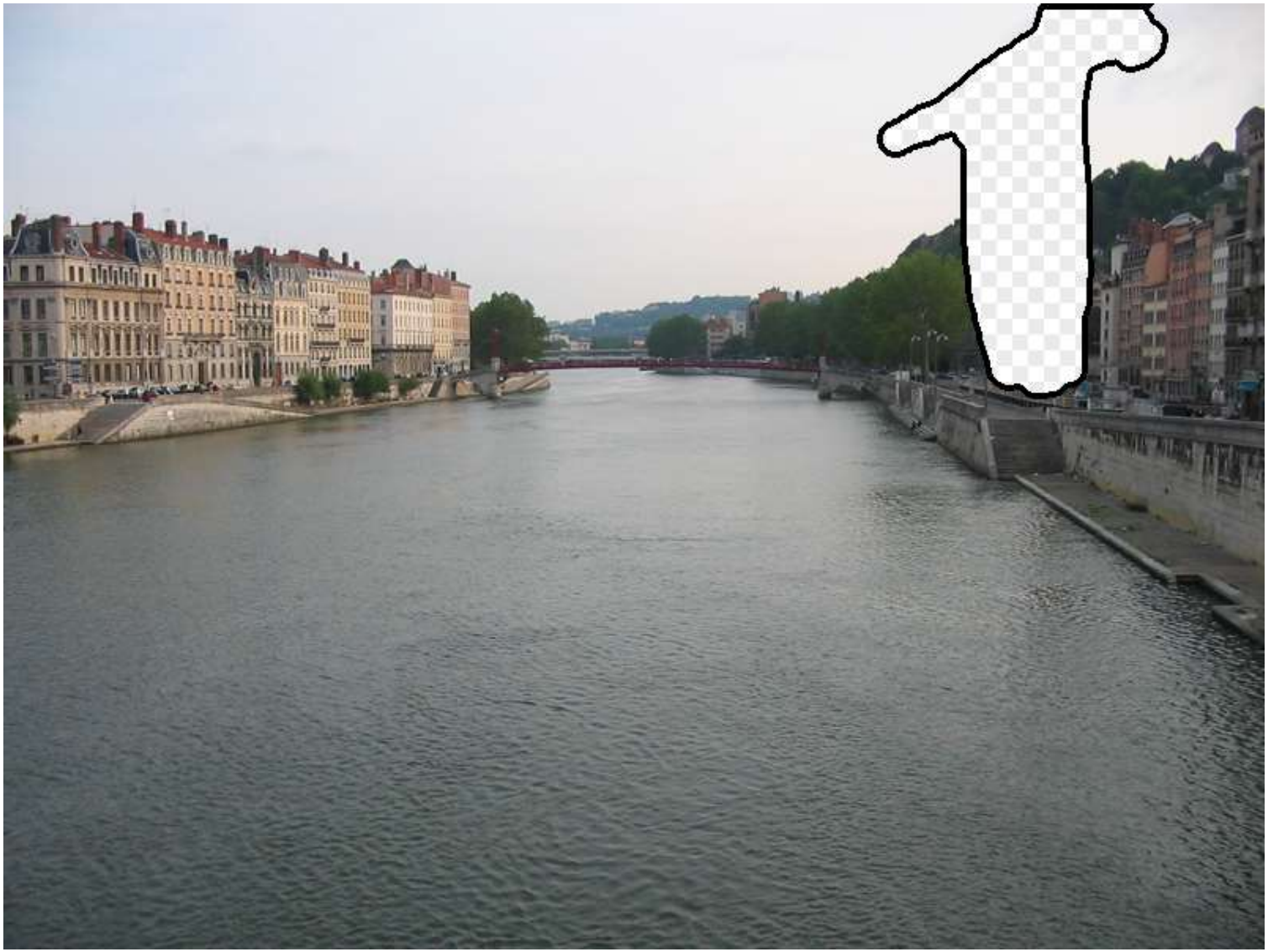
Context Matching



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

Problems with Instance-Based Learning

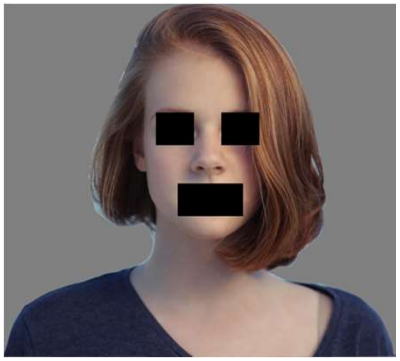
- Expensive
  - No Learning: most real work done during testing
  - For every test sample, must search through all dataset – very slow!
  - Must use tricks like approximate nearest neighbour search
- Doesn't work well when large number of irrelevant features
  - Distances overwhelmed by noisy features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

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)

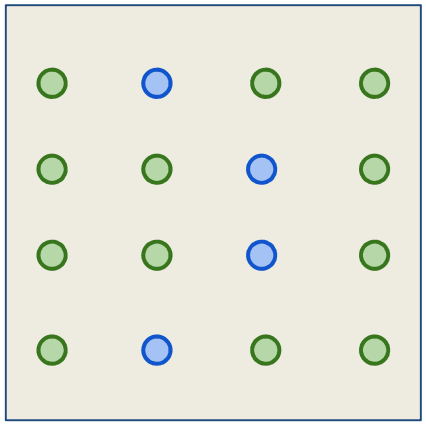
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

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

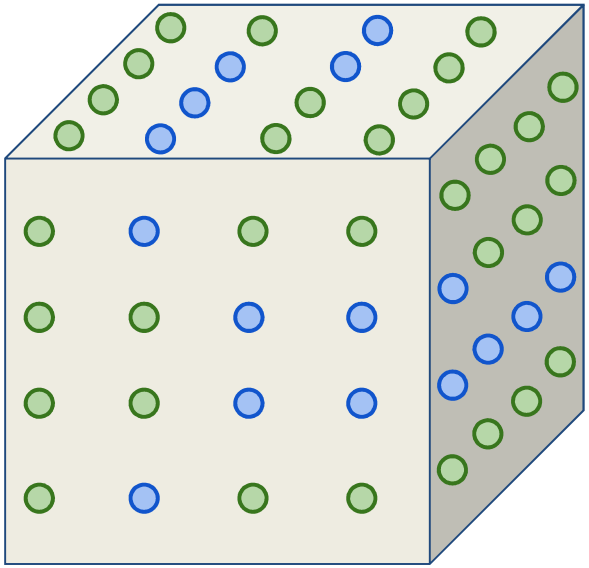
- Curse of dimensionality

Dimensions = 1
Points = 4

Dimensions = 2
Points = 4^2



Dimensions = 3
Points = 4^3



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

Curse of Dimensionality

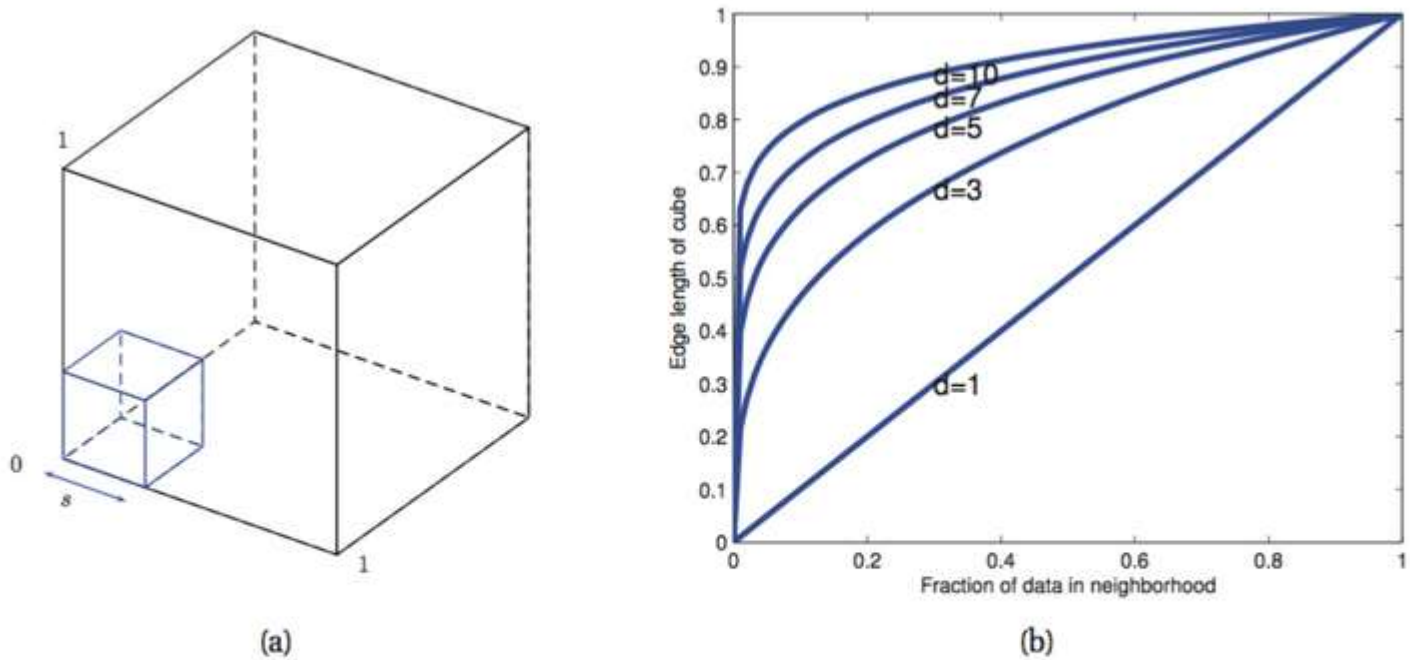


Figure 1.16 Illustration of the curse of dimensionality. (a) We embed a small cube of side s inside a larger unit cube. (b) We plot the edge length of a cube needed to cover a given volume of the unit cube as a function of the number of dimensions. Based on Figure 2.6 from (Hastie et al. 2009). Figure generated by `curseDimensionality`.

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!

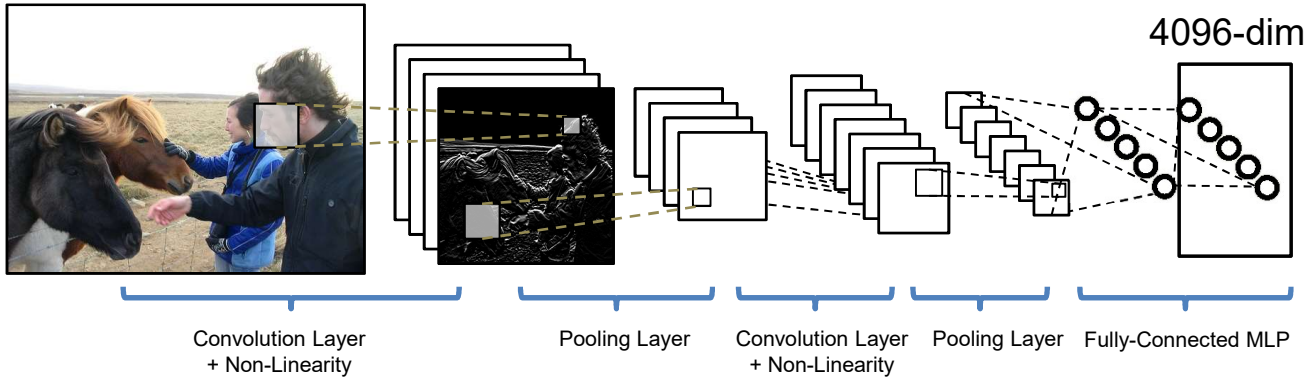
Neural Network

Linear classifiers

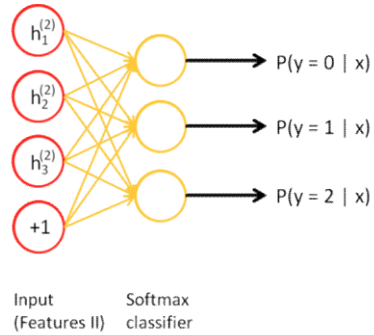
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Visual Question Answering

Image Embedding (VGGNet)

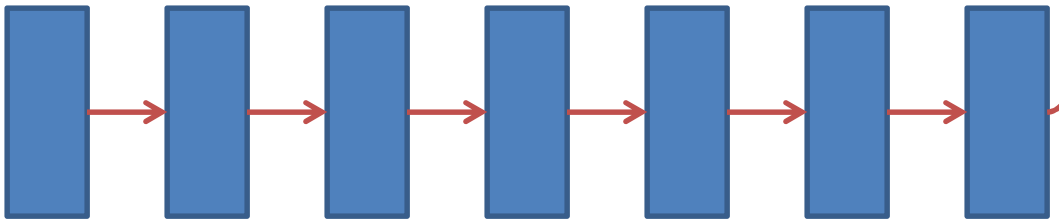


Neural Network
Softmax
over top K answers

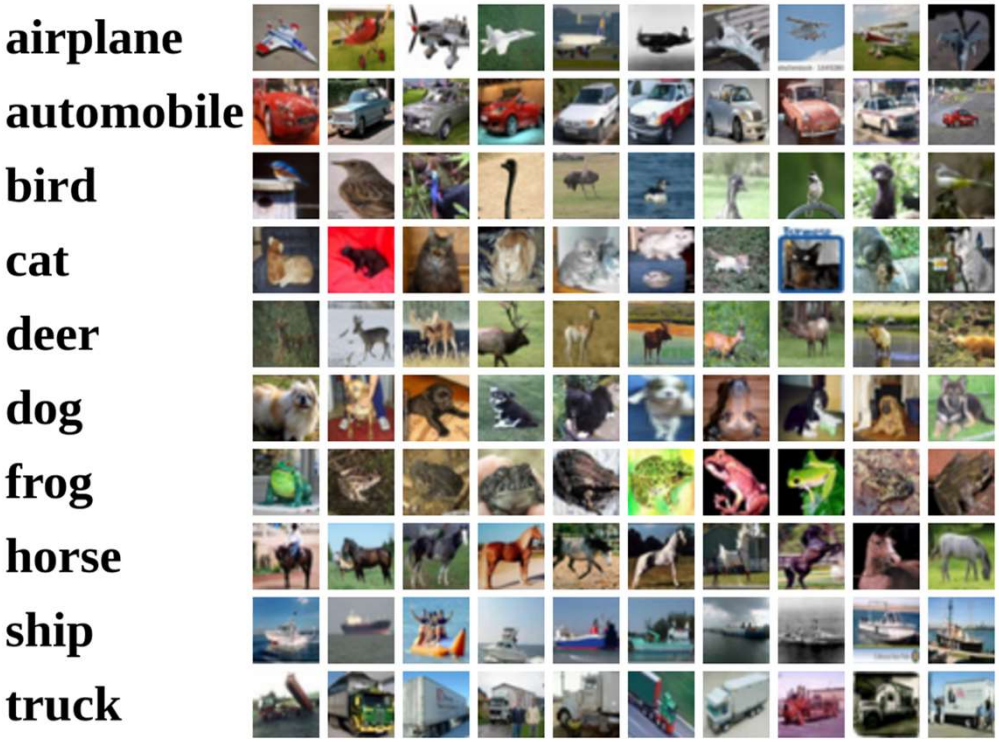


Question Embedding (LSTM)

"How many horses are in this image?"



Recall CIFAR10



50,000 training images
each image is **32x32x3**

10,000 test images.

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

Parametric Approach

Image

Array of **32x32x3** numbers
(3072 numbers total)

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

10 numbers giving
class scores

W

parameters
or weights

Parametric Approach: Linear Classifier

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

Image

Array of **32x32x3** numbers
(3072 numbers total)

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

10 numbers giving
class scores

W

parameters
or weights

Parametric Approach: Linear Classifier

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

3072x1

10x1 10x3072

Image

Array of **32x32x3** numbers
(3072 numbers total)

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

10 numbers giving
class scores

W

parameters
or weights

Parametric Approach: Linear Classifier

$$\underbrace{f(x,W)}_{10\times 1} = \underbrace{W}_{10\times 3072}\underbrace{x}_{3072\times 1} + \underbrace{b}_{10\times 1}$$

Image

Array of **32x32x3** numbers
(3072 numbers total)

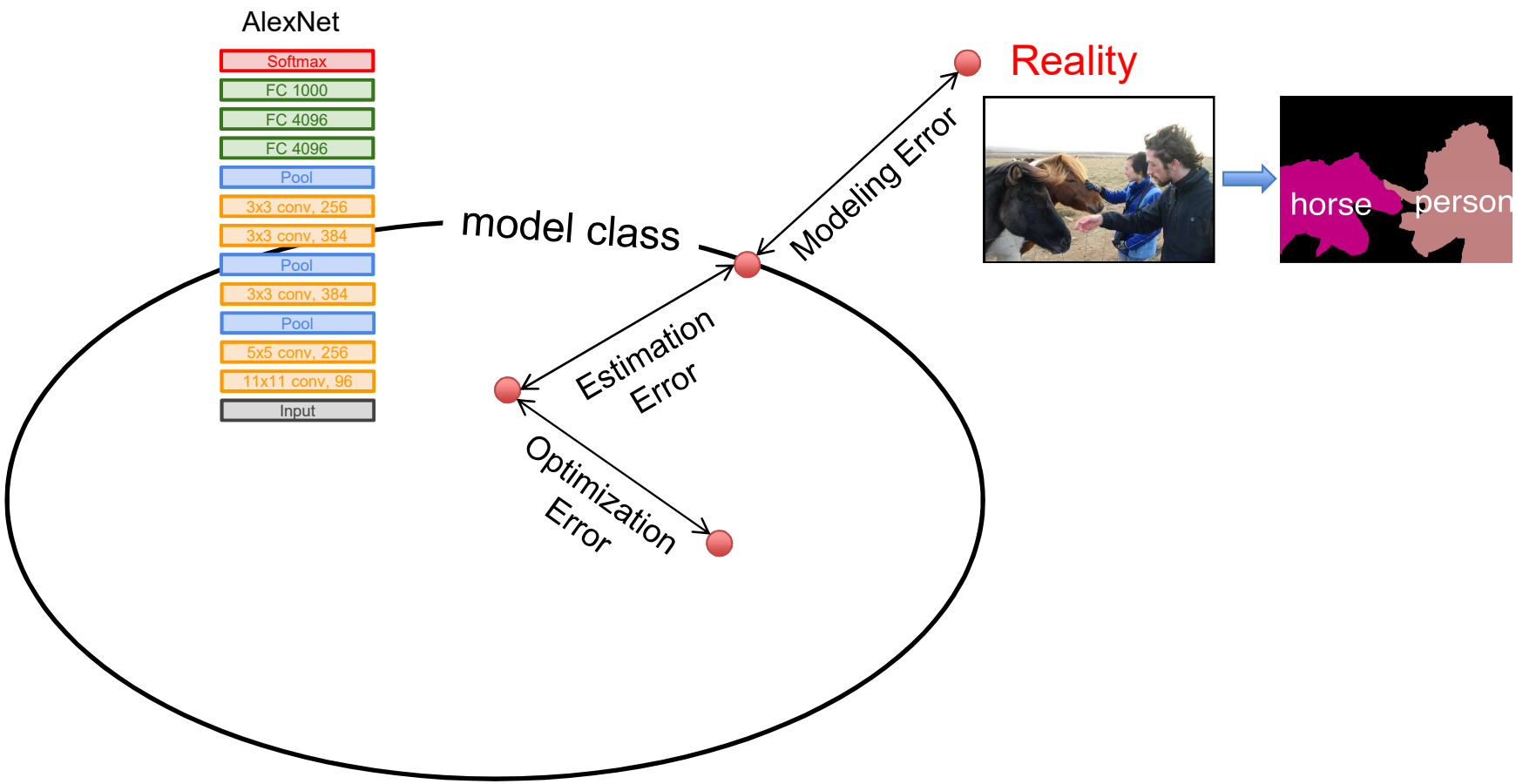
$$f(\mathbf{x},\mathbf{W})$$

10 numbers giving
class scores

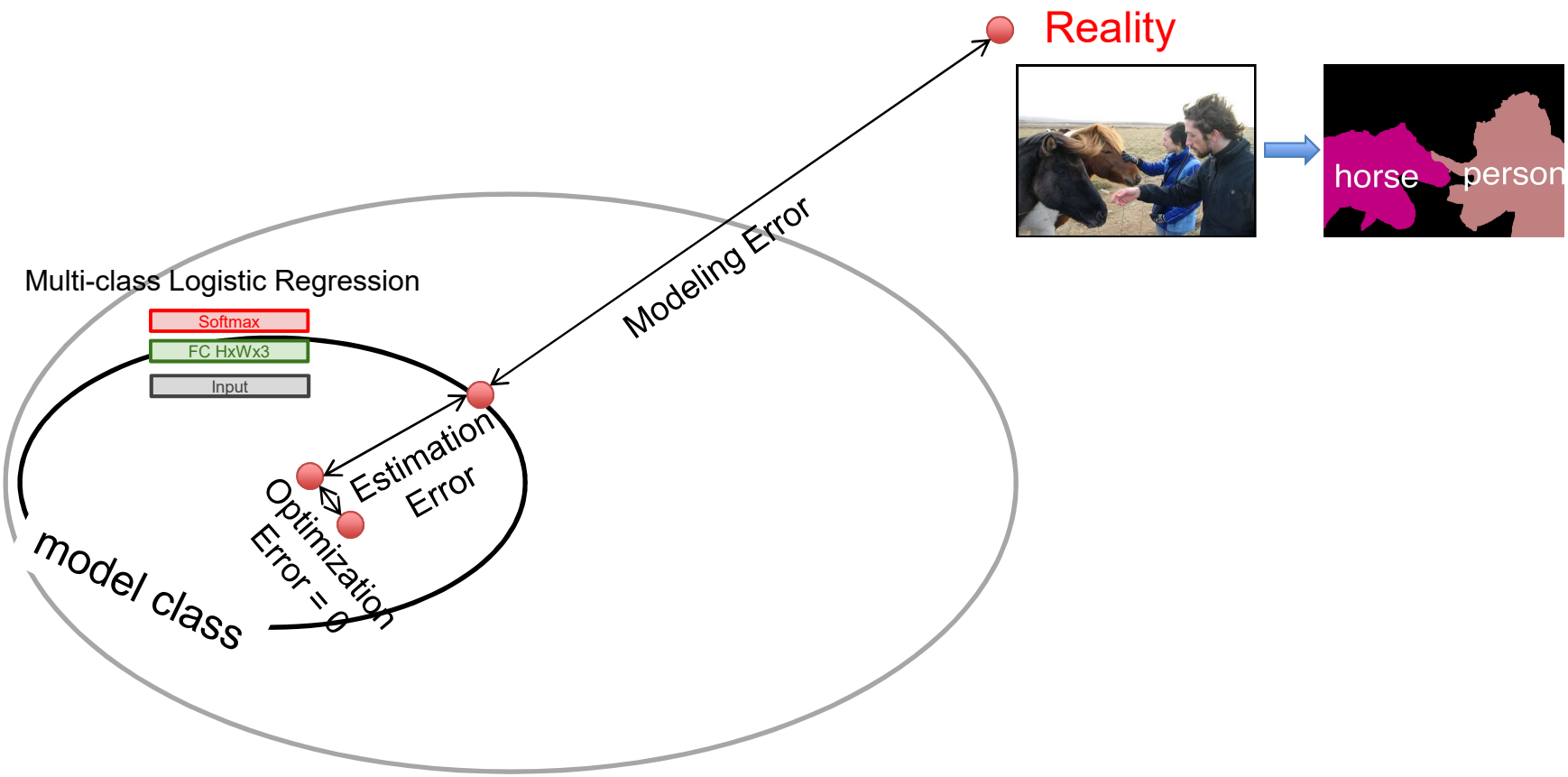
W

parameters
or weights

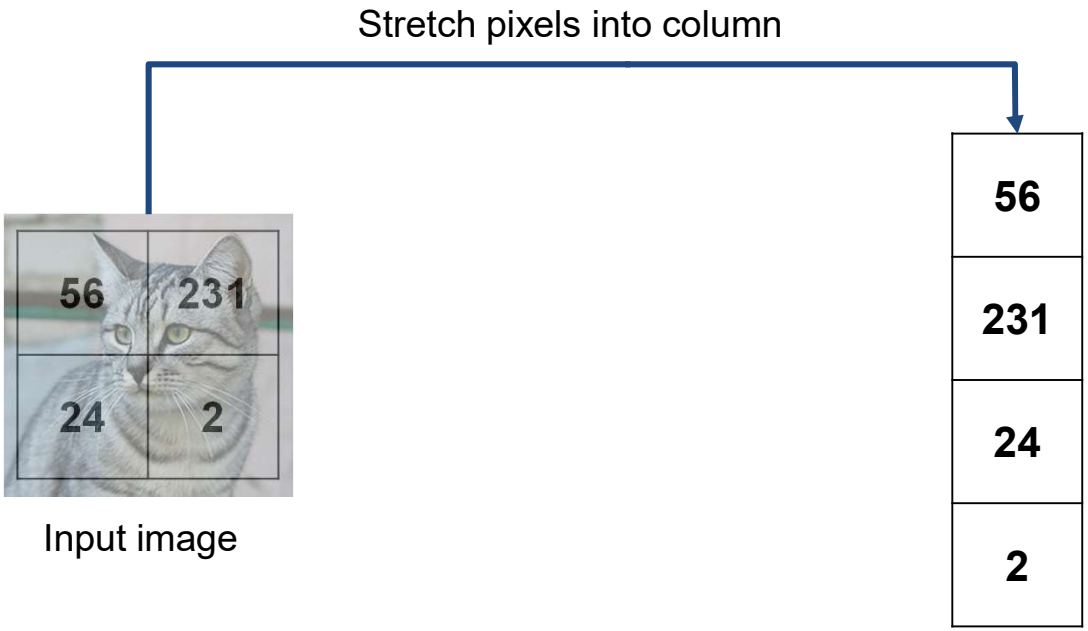
Error Decomposition



Error Decomposition

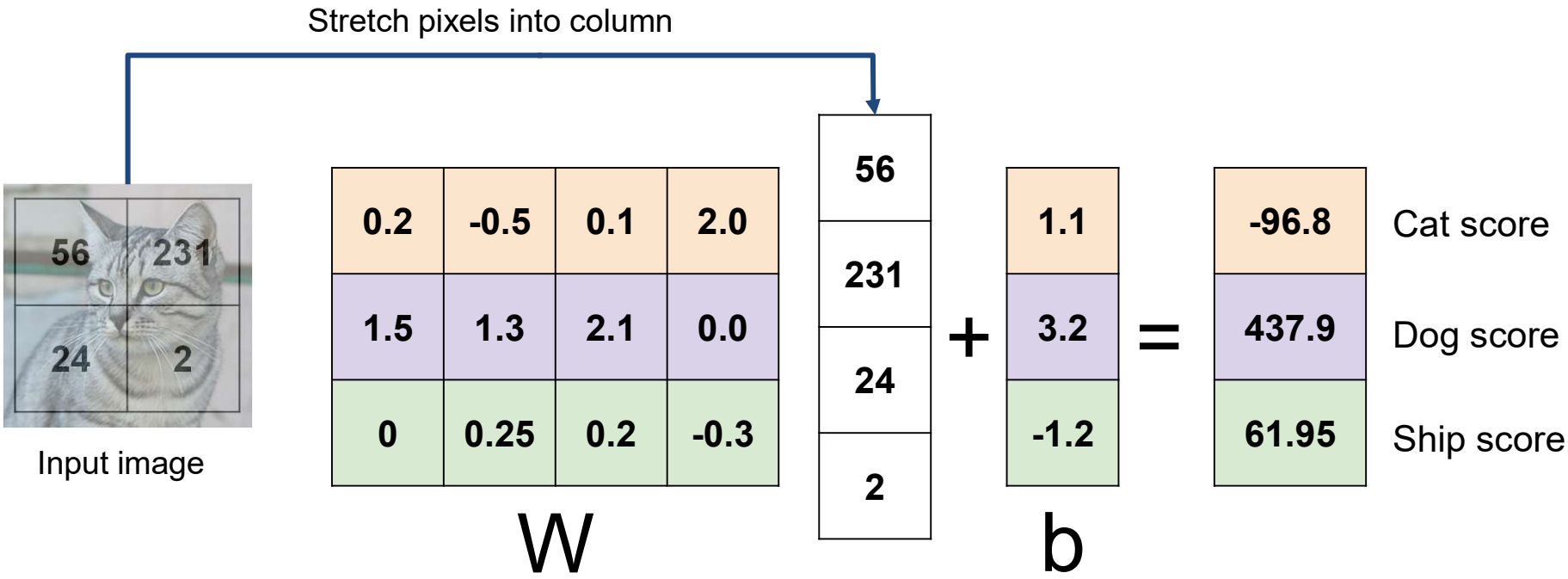


Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



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

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

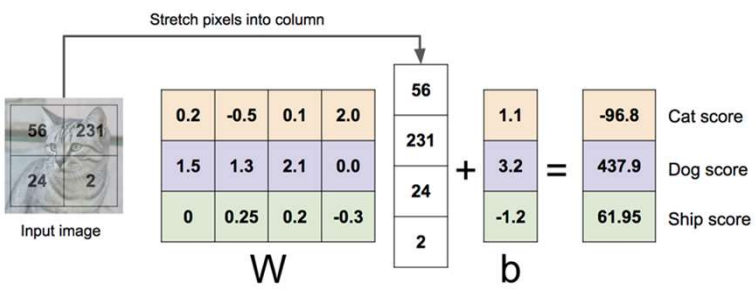


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

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

<u>Algebraic Viewpoint</u>

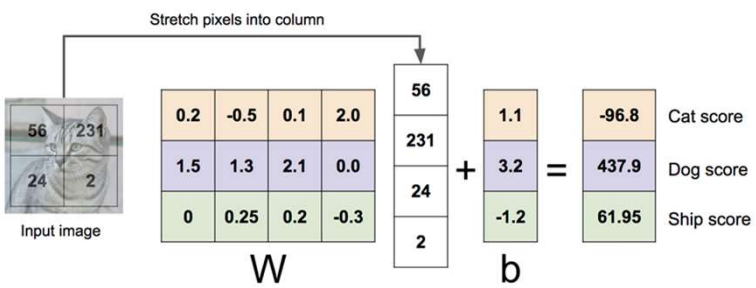
$$f(x,W) = Wx$$

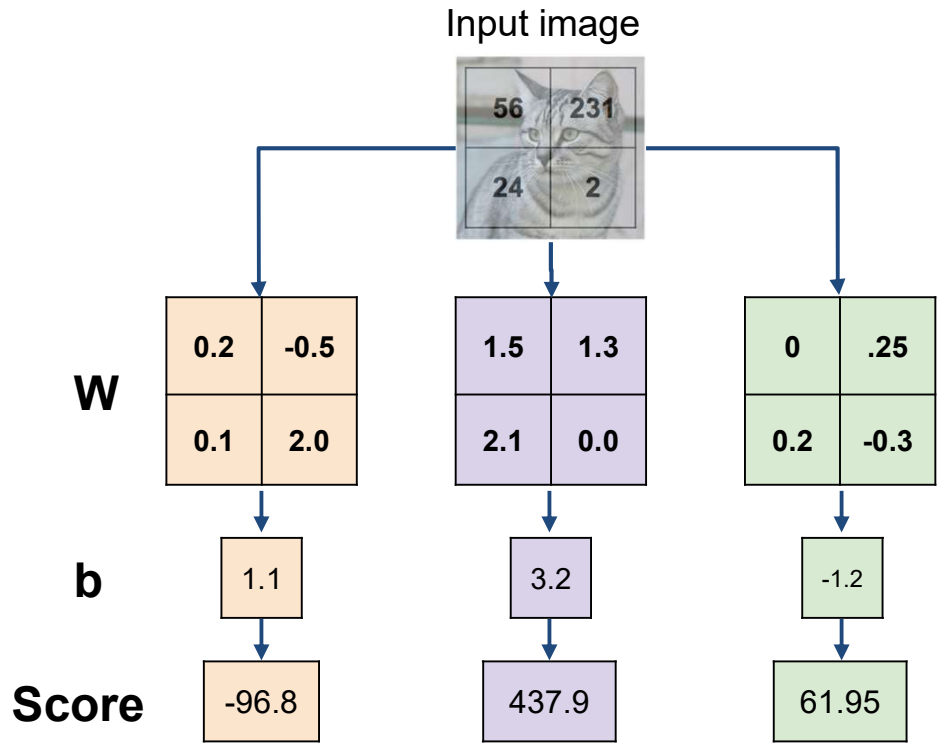


Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Algebraic Viewpoint

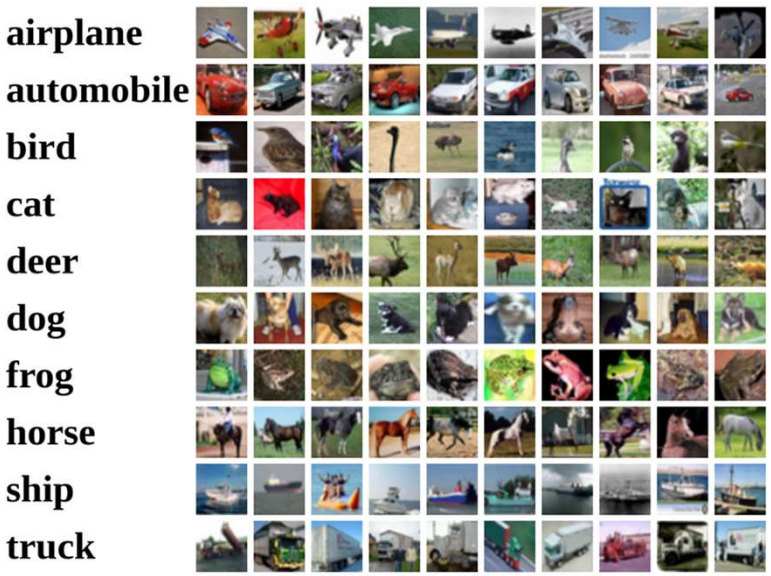
$$f(x,W) = Wx$$

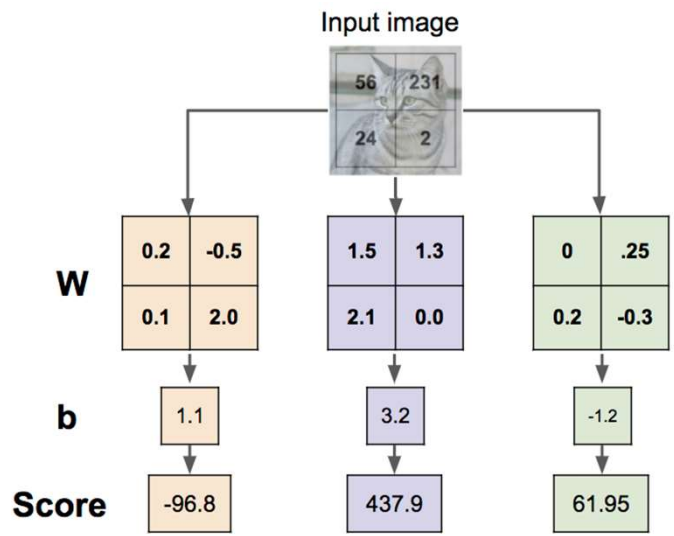




Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

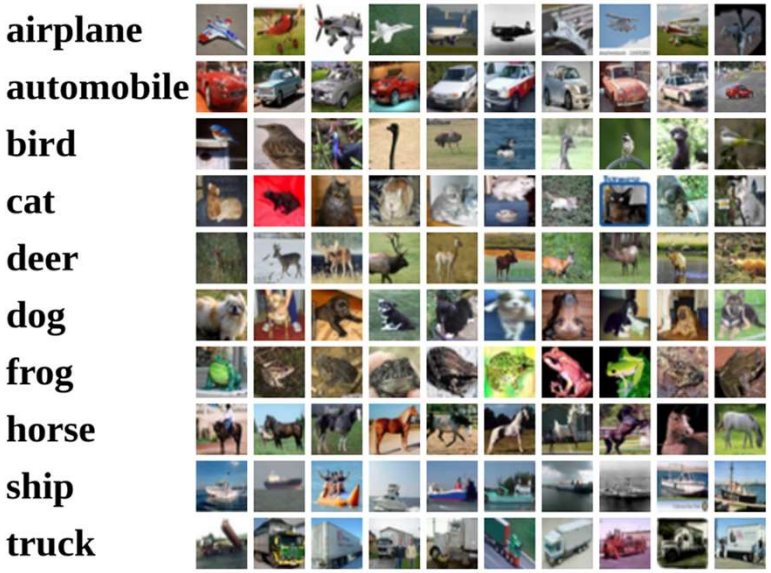
Interpreting a Linear Classifier

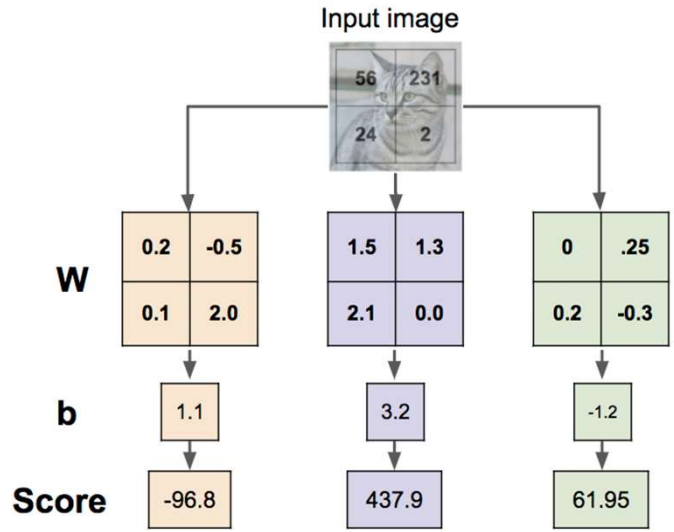


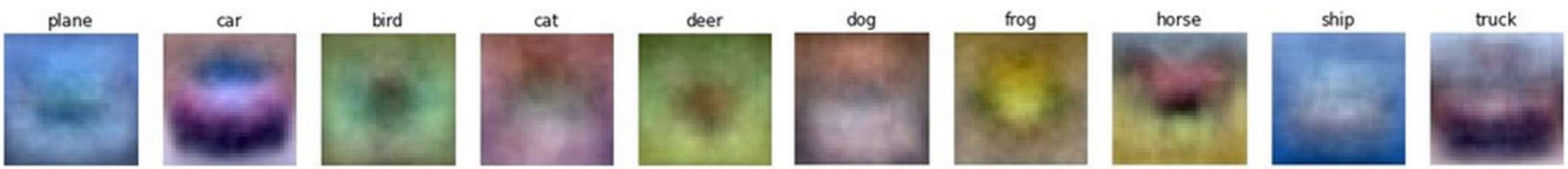


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Interpreting a Linear Classifier: Visual Viewpoint

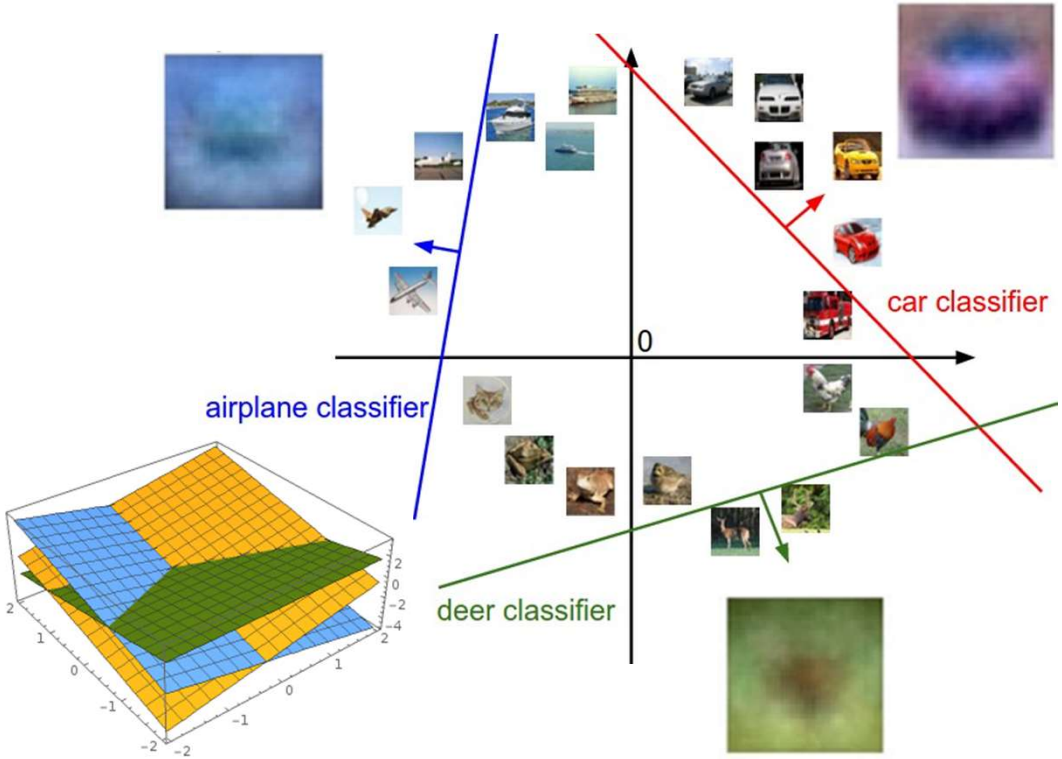






Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Interpreting a Linear Classifier: <u>Geometric Viewpoint</u>



Plot created using Wolfram Cloud

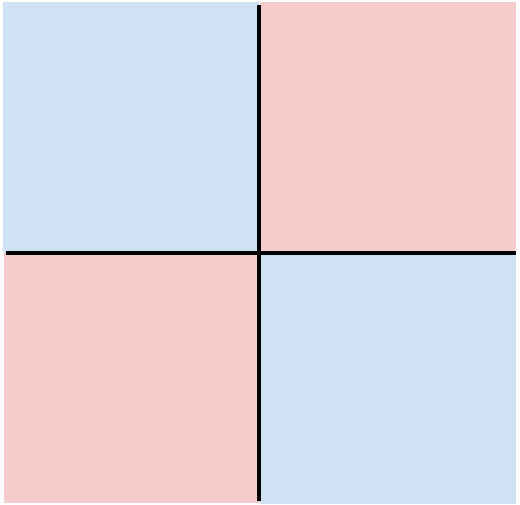
$$f(x,W) = Wx + b$$

Array of **32x32x3** numbers
(3072 numbers total)

Hard cases for a linear classifier

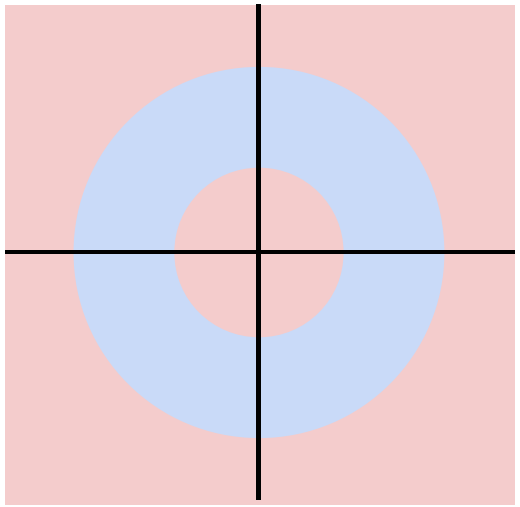
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



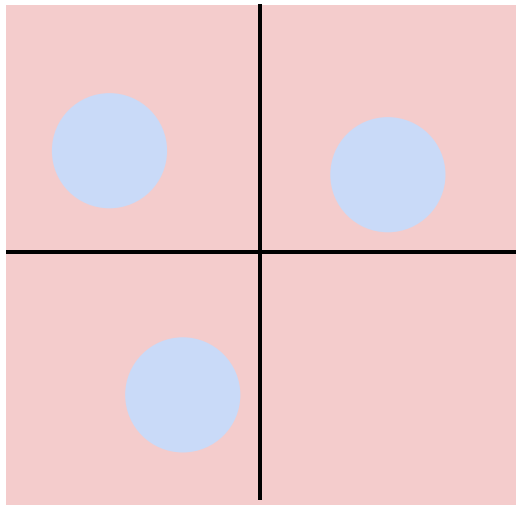
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

Class 2:
Everything else

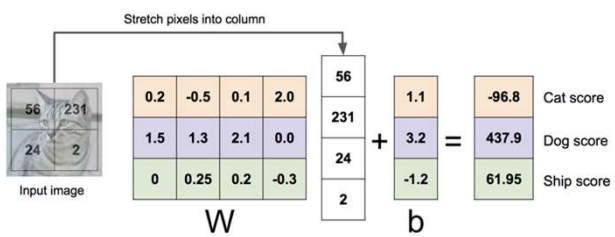


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Linear Classifier: Three Viewpoints

Algebraic Viewpoint

$f(x,W) = Wx$

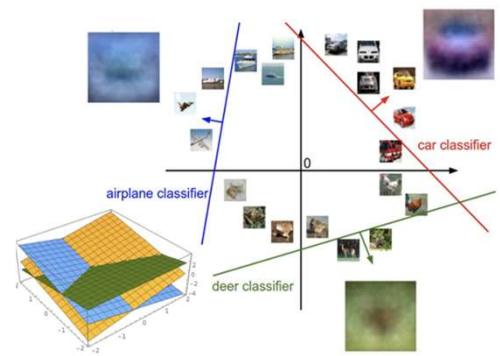


Visual Viewpoint

One template per class

Geometric Viewpoint

Hyperplanes cutting up space



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

So far: Defined a (linear) <u>score function</u>

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

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

Example class scores for 3 images for some W:

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

So far: Defined a (linear) <u>score function</u>

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

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)**

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