Lecture 2: Data Driven Approach, kNN, Linear Classification 1

박사과정 김성빈 chengbinjin@inha.edu,
지도교수 김학일 교수 hikim@inha.ac.kr
인하대학교 컴퓨터비전 연구실

Assignment #1

Q1: k-Nearest Neighbor classifier (20 points)

The IPython Notebook **knn.ipynb** will walk you through implementing the kNN classifier.

Q2: Training a Support Vector Machine (25 points)

The IPython Notebook **svm.ipynb** will walk you through implementing the SVM classifier.

Q3: Implement a Softmax classifier (20 points)

The IPython Notebook **softmax.ipynb** will walk you through implementing the Softmax classifier.

Q4: Two-Layer Neural Network (25 points)

The IPython Notebook **two_layer_net.ipynb** will walk you through the implementation of a two-layer neural network classifier.

Q5: Higher Level Representations: Image Features (10 points)

The IPython Notebook **features.ipynb** will walk you through this exercise, in which you will examine the improvements gained by using higher-level representations as opposed to using raw pixel values.

Q6: Cool Bonus: Do something extra! (+10 points)

Implement, investigate or analyze something extra surrounding the topics in this assignment, and using the code you developed. For example, is there some other interesting question we could have asked? Is there any insightful visualization you can plot? Or anything fun to look at? Or maybe you can experiment with a spin on the loss function? If you try out something cool we'll give you up to 10 extra points and may feature your results in the lecture.

- Python 2.7
- Anaconda 2
- **Ubuntu** or Windows
- We will have <u>10 minutes</u> python tutorial to help all of u become python master!

Image Classification: a core task in Computer Vision

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

cat

A tiny Delta on top of **Image Classification**

- Object detection
- Image captioning
- Segmentation
- Others

Slide from CS231n

Why is this problem hard?

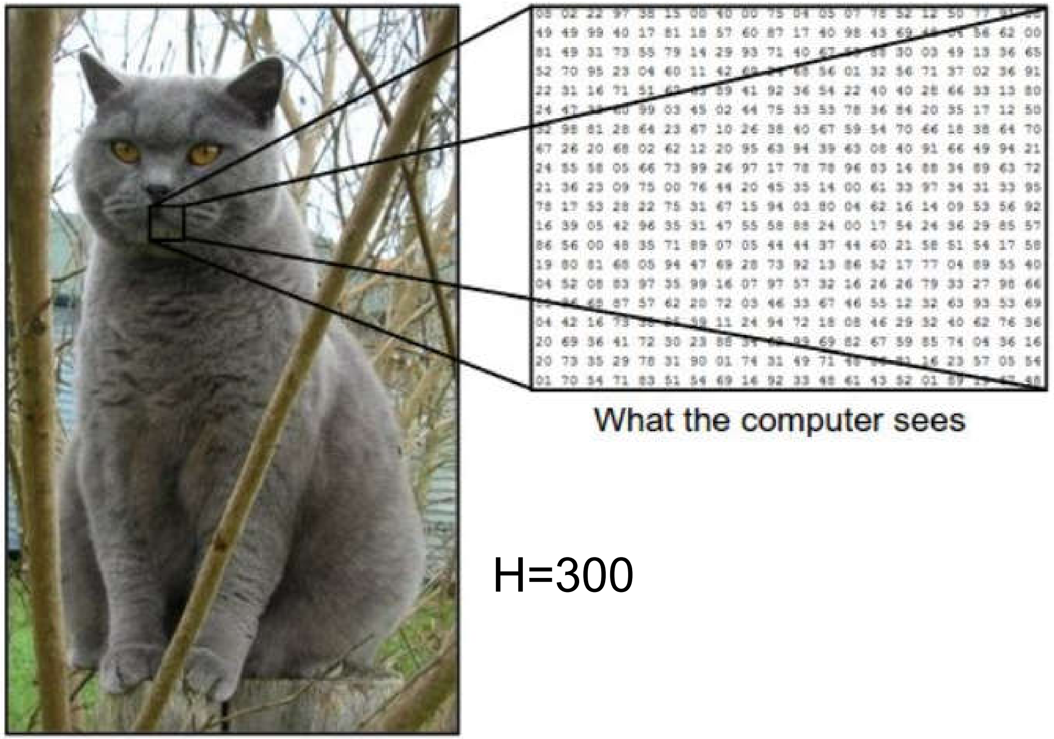
The problem:

- Semantic gap

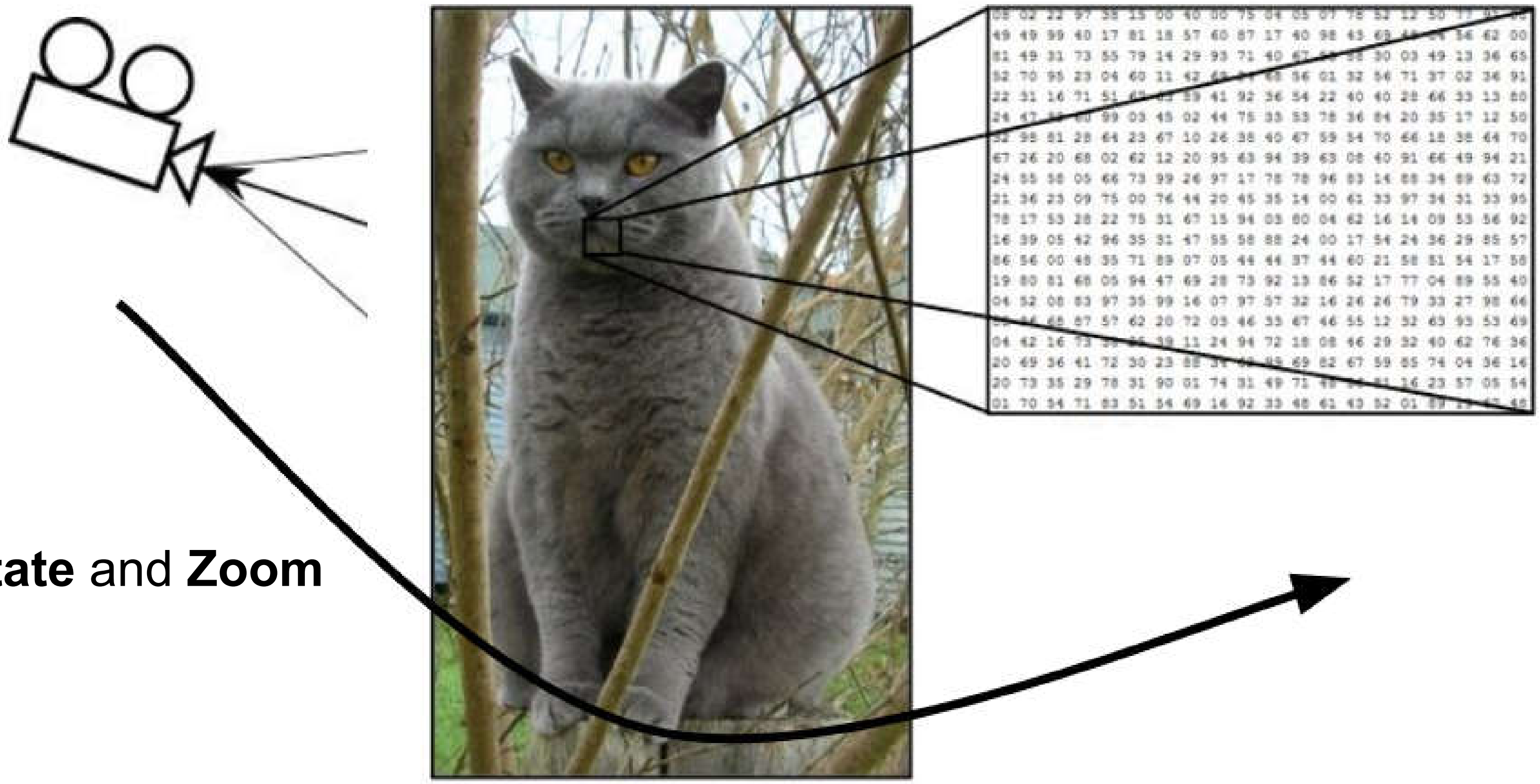
Images are represented as 3D arrays of numbers, with integers between **[0, 255]**.

E.g.

300 x 100 x 3 (RGB)



Challenges: Viewpoint Variation

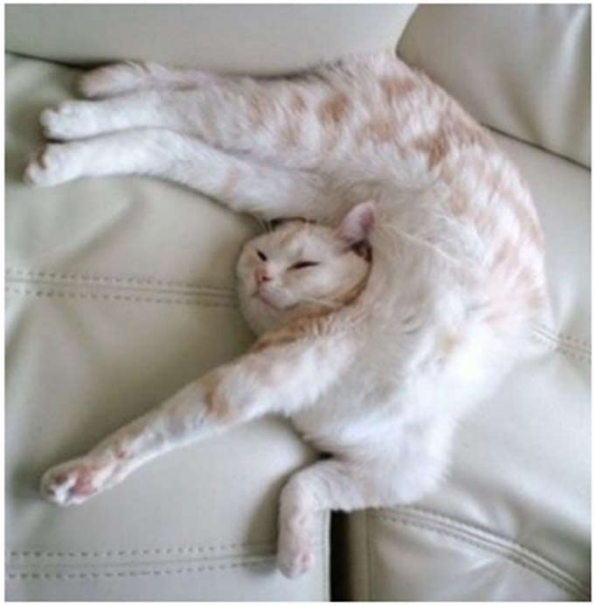


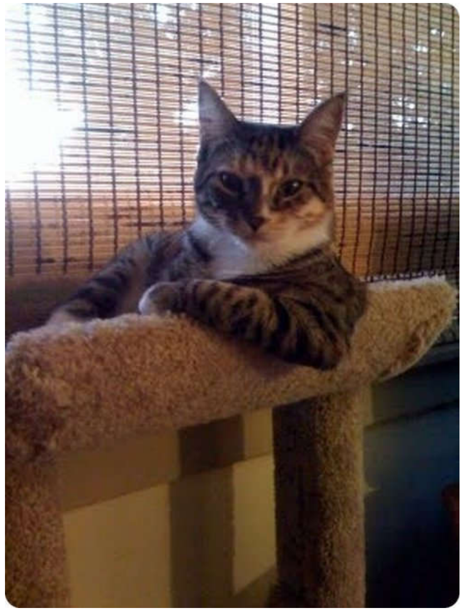
Rotate and **Zoom**

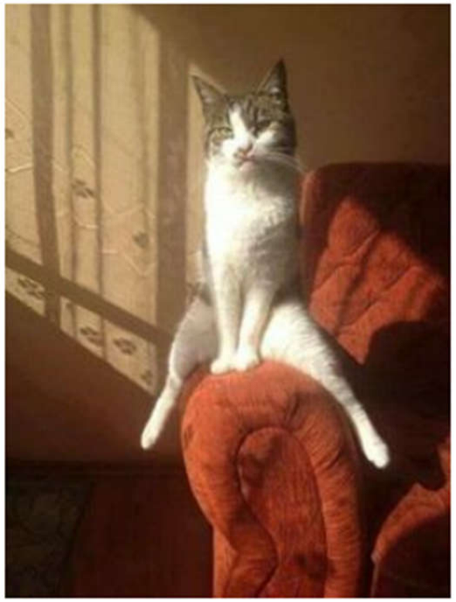
Challenges: Illumination

Slide from CS231n

Challenges: Deformation








Challenges: Occlusion




Challenges: Background Clutter

Challenges: Intraclass Variation

Slide from CS231n

An Image Classifier

Cat: 0

Dog: 1

Human: 2

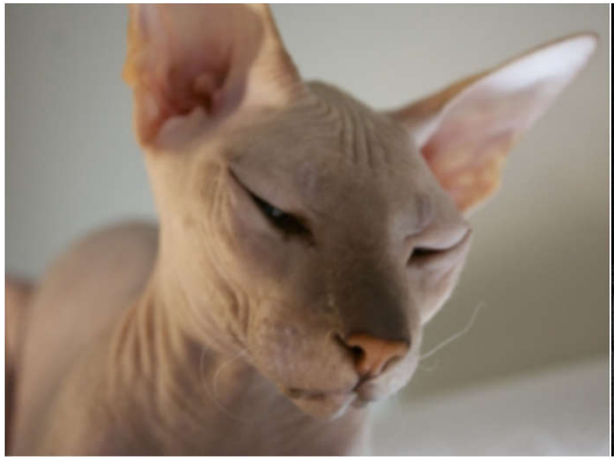
…

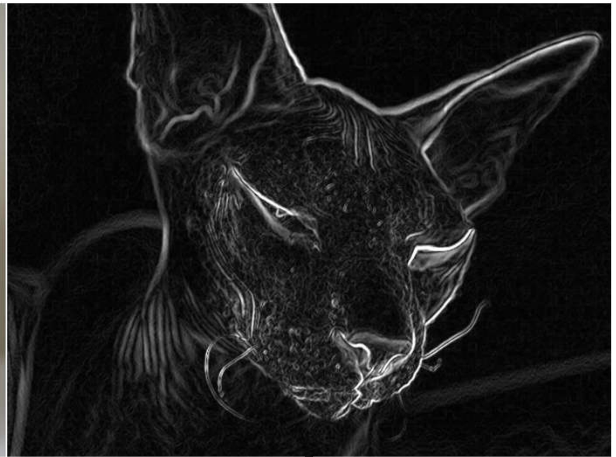
```
def predict(image):
    # ???
    return class_label
```

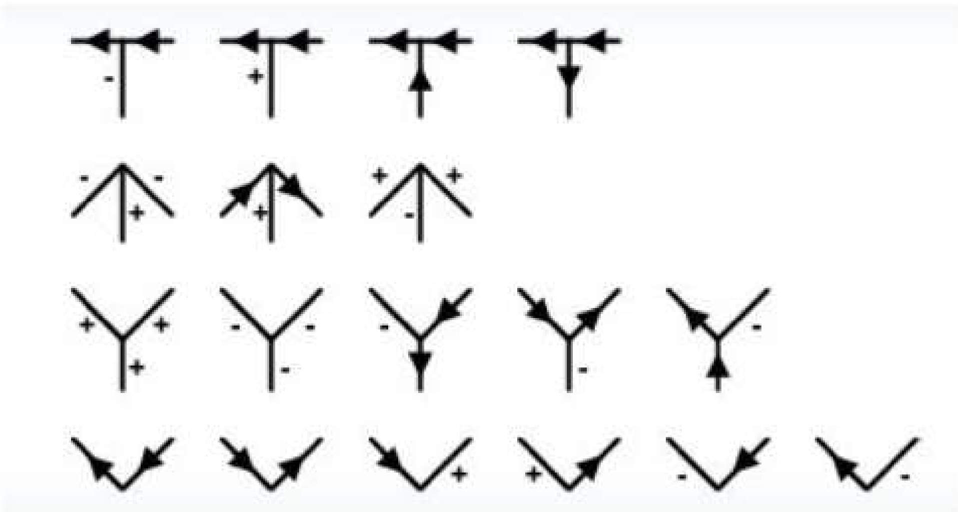
- Unlike e.g. sorting a list of numbers (according to increase or decrease)
- **No obvious way to hard-code the algorithm for recognizing a cat, or other classes**

Slide from CS231n

Attempts have been made







???

Rule-based

Data-Driven Approach

1. **Collect a dataset** of images and labels
2. Use Machine Learning to **train an image classifier**
3. Evaluate the classifier on a withheld set of test images

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

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

Example training set

Slide from CS231n

First Classifier: **Nearest Neighbor Classifier**

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

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

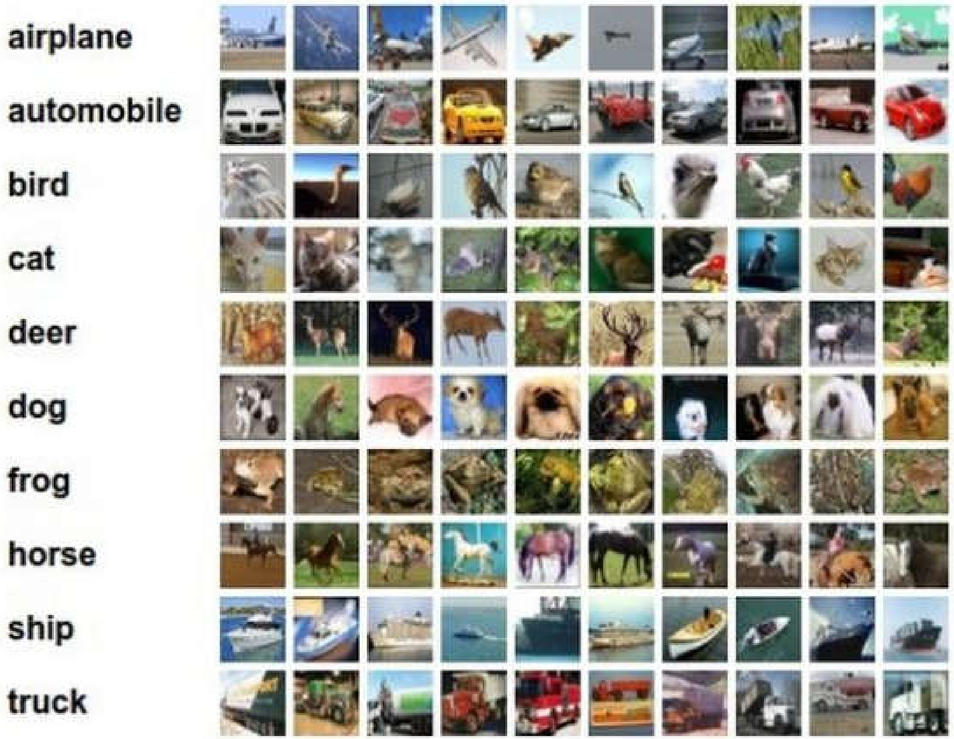
Remember all training images and their labels

Predict the **label of the most similar training image**

Slide from CS231n

Example Dataset: **CIFAR-10**

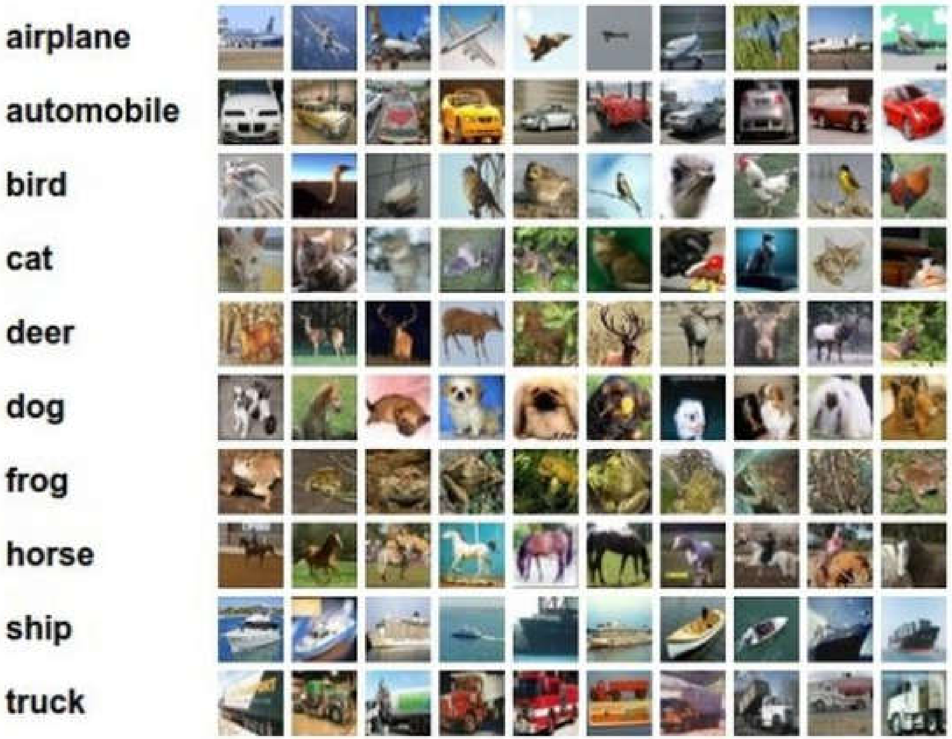
- **10** labels (classes)
- **50,000** training images
- **10,000** test images.



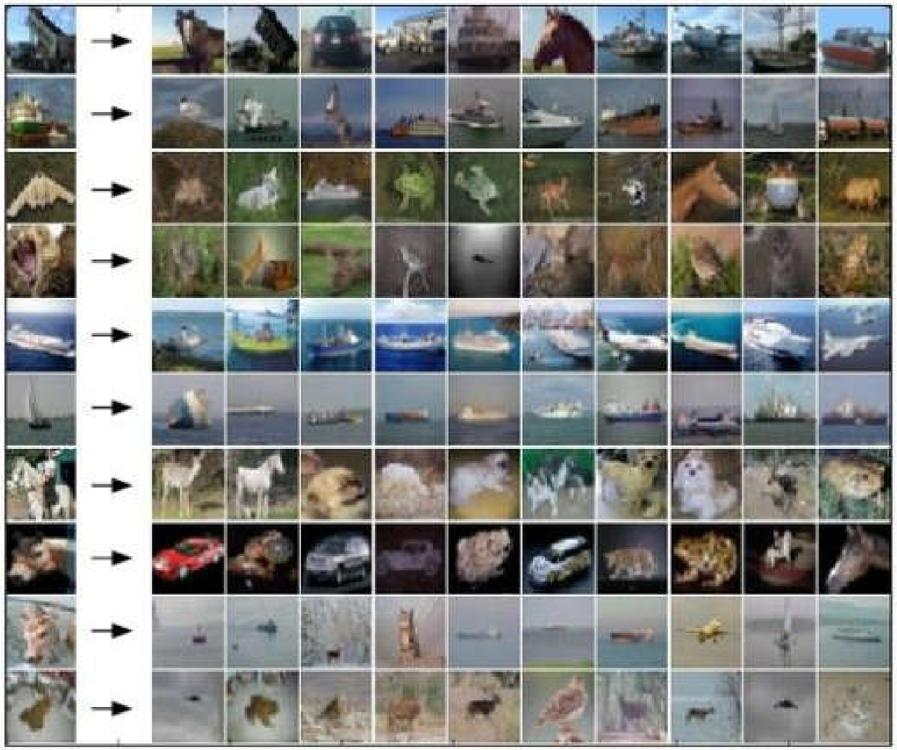
Example Dataset: **CIFAR-10**

- **10** labels (classes)
- **50,000** training images
- **10,000** test images.
- For every test image (first column), examples of nearest neighbors in rows

Training process



Test process



Slide from CS231n

Distance Metric

- How do we compare the images? What is the **distance metric**?

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

(Manhattan distance)

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 from CS231n

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
```

Slide from CS231n

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
```

Remember the training data

Slide from CS231n

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 every test image:

- Find nearest train image with L1 distance
- Predict the label of nearest training image

Slide from CS231n

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
```

Question #1: How does the classification speed depend on the size of training data?

Slide from CS231n

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
```

Question #1: How does the classification speed depend on the size of training data?

This is **backwards:**

- Test time performance is usually much more important in practice
- CNNs flip this: expensive training, cheap test evaluation

Slide from CS231n

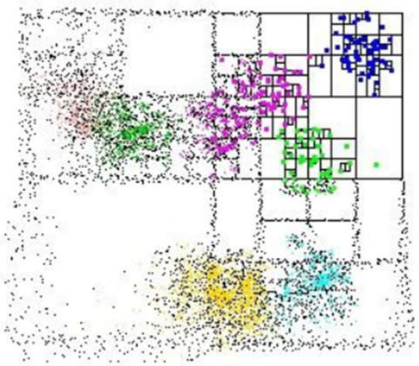
Aside: Approximate Nearest Neighbor

- Find approximate nearest neighbors quickly

FLANN

ANN: A Library for Approximate Nearest Neighbor Searching

David M. Mount and **Sunil Arya**
Version 1.1.2
Release Date: Jan 27, 2010



ANN

What is ANN?

ANN is a library written in C++, which supports data structures and algorithms for both exact and approximate nearest neighbor searching in arbitrarily high dimensions.

In the nearest neighbor problem a set of data points in d-dimensional space is given. These points are preprocessed into a data structure, so that given any query point q, the nearest or generally k nearest points of P to q can be reported efficiently. The distance between two points can be defined in many ways. ANN assumes that distances are measured using any class of distance functions called Minkowski metrics. These include the well known Euclidean distance, Manhattan distance, and max distance.

Based on our own experience, ANN performs quite efficiently for point sets ranging in size from thousands to hundreds of thousands, and in dimensions as high as 20. (For applications in significantly higher dimensions, the results are rather spotty, but you might try it anyway.)

The library implements a number of different data structures, based on kd-trees and box-decomposition trees, and employs a couple of different search strategies.

The library also comes with test programs for measuring the quality of performance of ANN on any particular data sets, as well as programs for visualizing the structure of the geometric data structures.

FLANN - Fast Library for Approximate Nearest Neighbors

- Home
- News
- Publications
- Download
- Changelog
- Repository

What is FLANN?

FLANN is a library for performing fast approximate nearest neighbor searches in high dimensional spaces. It contains a collection of algorithms we found to work best for nearest neighbor search and a system for automatically choosing the best algorithm and optimum parameters depending on the dataset.

FLANN is written in C++ and contains bindings for the following languages: C, MATLAB and Python.

News

- (14 December 2012) Version 1.8.0 is out bringing incremental addition/removal of points to/from indexes
- (20 December 2011) Version 1.7.0 is out bringing two new index types and several other improvements.
- You can find binary installers for FLANN on the Point Cloud Library project page. Thanks to the PCL developers!
- Mac OS X users can install flann though MacPorts (thanks to Mark Moll for maintaining the Portfile)
- New release introducing an easier way to use custom distances, kd-tree implementation optimized for low dimensionality search and experimental MPI support
- New release introducing new C++ templated API, thread-safe search, save/load of indexes and more.
- The FLANN license was changed from LGPL to BSD.

How fast is it?

In our experiments we have found FLANN to be about one order of magnitude faster on many datasets (in query time), than previously available approximate nearest neighbor search software.

Publications

More information and experimental results can be found in the following papers:
- Marius Muja and David G. Lowe: **"Scalable Nearest Neighbor Algorithms for High Dimensional Data"**. Pattern Analysis and Machine Intelligence (PAMI), Vol. 36, 2014. [PDF] [BibTeX]
- Marius Muja and David G. Lowe: **"Fast Matching of Binary Features"**. Conference on Computer and Robot Vision (CRV) 2012. [PDF] [BibTeX]
- Marius Muja and David G. Lowe, **"Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration"**, in International Conference on Computer Vision Theory and Applications (VISAPP'09), 2009 [PDF] [BibTeX]

Slide from CS231n

Hyperparameter (Metric)

- The choice of distance is a **hyperparameter** common choices:

L1 (Manhattan) distance

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

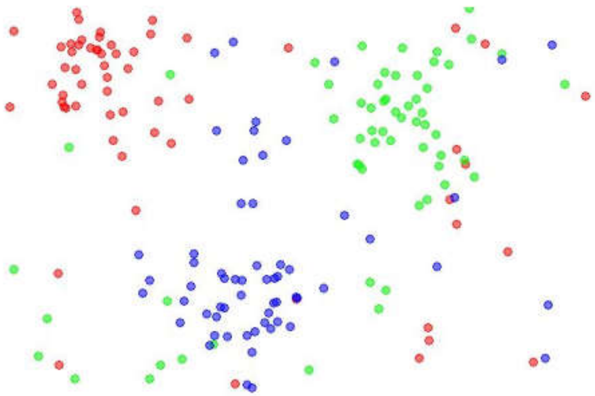
L2 (Euclidean) distance

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

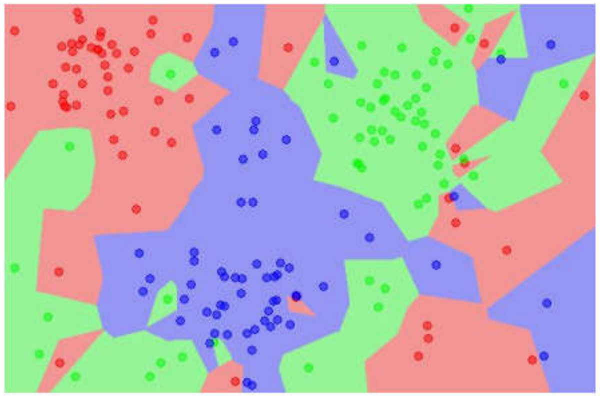
K-Nearest Neighbor

- Find the k nearest images, have them **vote on the label**

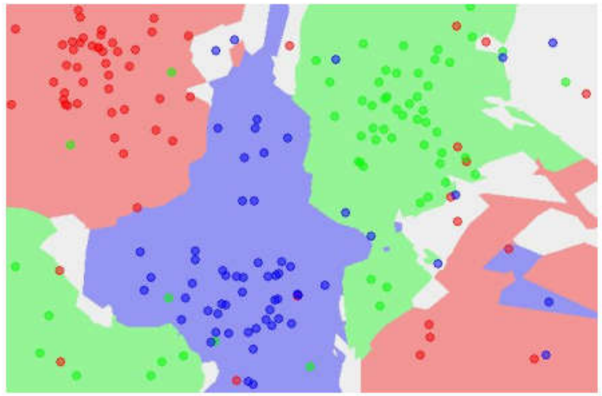
The data



NN classifier



5-NN classifier

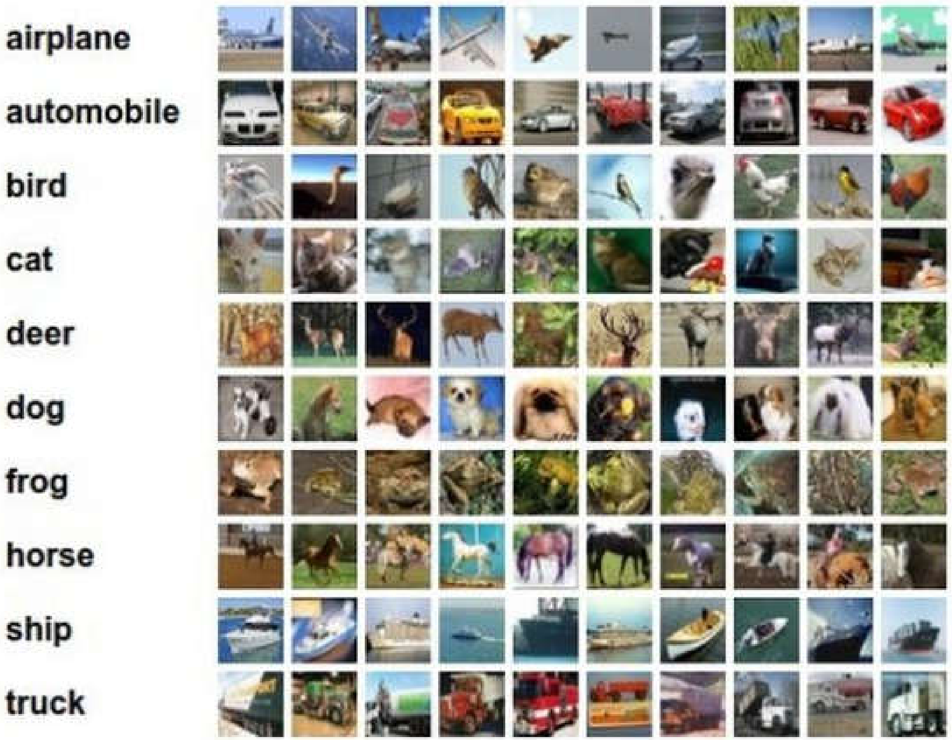


https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Slide from CS231n

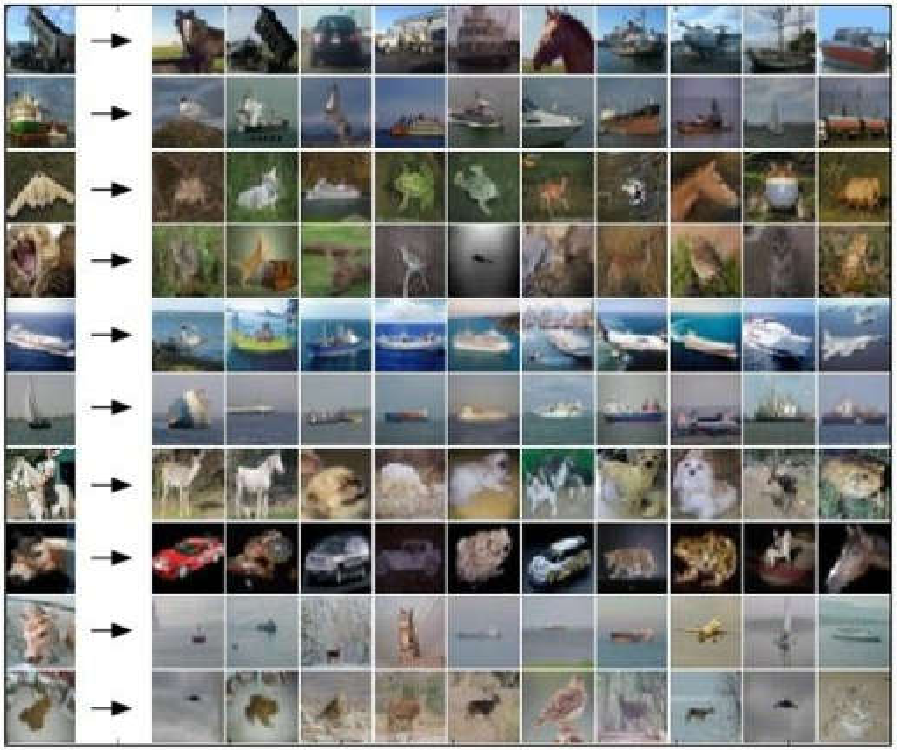
Example Dataset: **CIFAR-10**

- **10** labels
- **50,000** training images
- **10,000** test images.



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

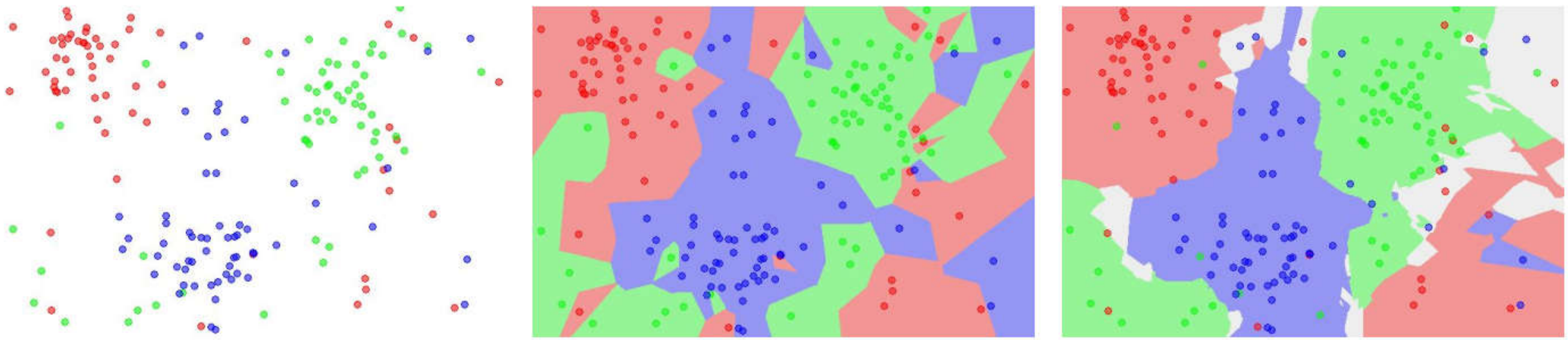
Majority Vote



Slide from CS231n

Nearest Neighbor (NN)

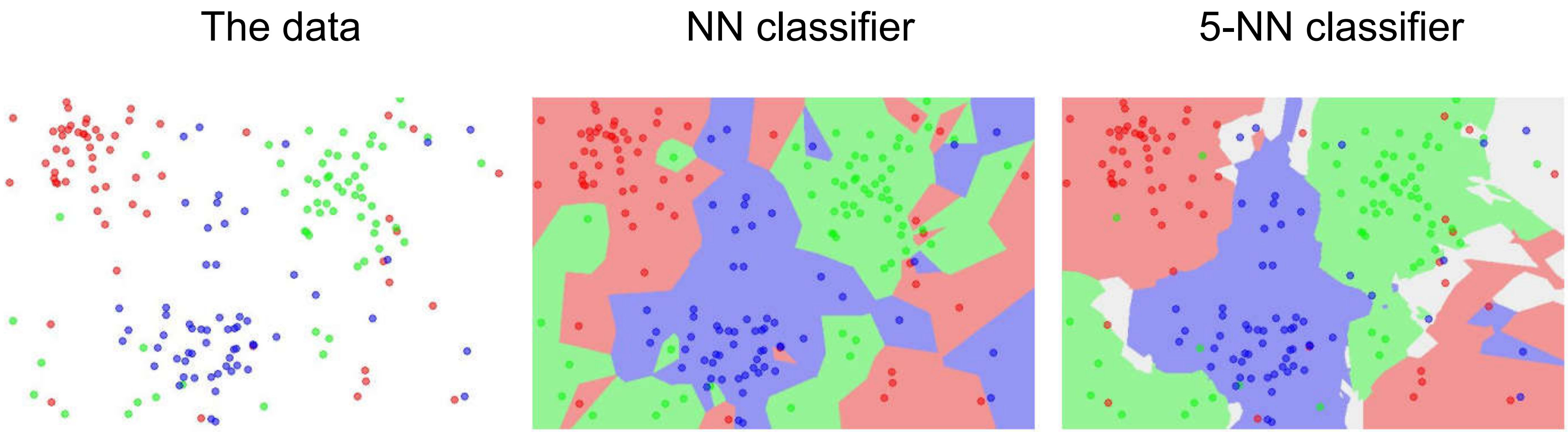
The data | NN classifier | 5-NN classifier



Question #2: What is the accuracy of the nearest neighbor classifier on the training data, when using the Euclidean distance?

L2 (Euclidean) distance $$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$

Nearest Neighbor (NN)

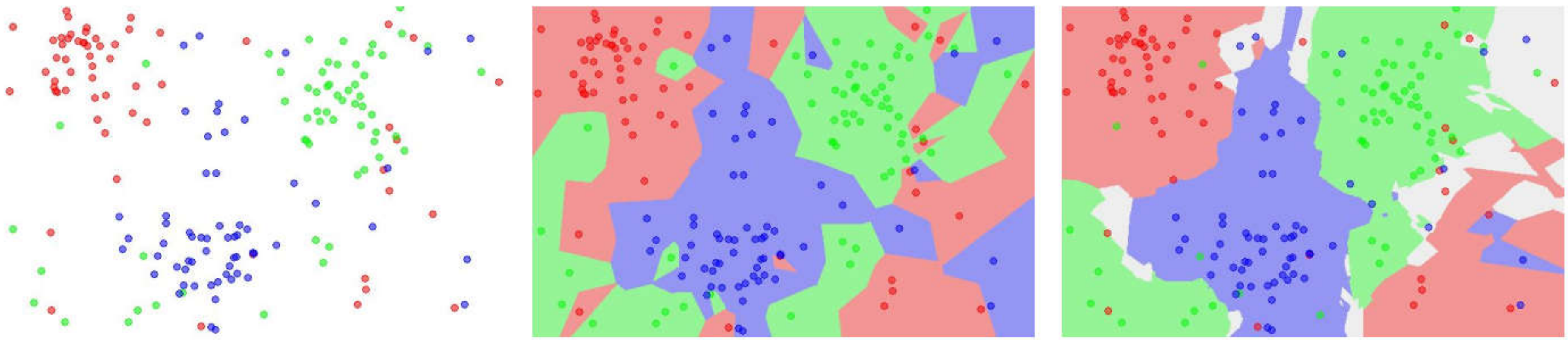


Question #3: What is the accuracy of the nearest neighbor classifier on the training data, when using the Manhattan distance?

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

Nearest Neighbor (NN)

The data | NN classifier | 5-NN classifier



Question #4: What is the accuracy of the k-nearest neighbor classifier on the training data?

Hyperparameters

- What is the best **distance** to use? (L1 or L2, …)
- What is the best value of **k** to use? (1, 3, 5, 7, …)

i.e. How do we set the **hyperparameters**?

Slide from CS231n

Hyperparameters

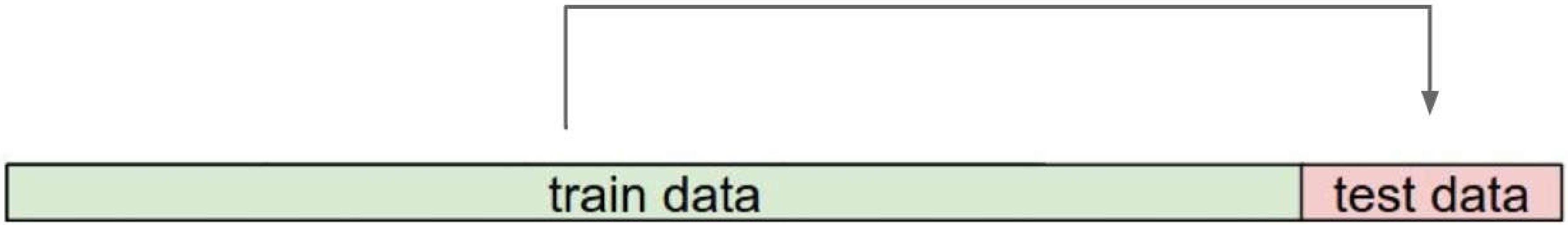
- What is the best **distance** to use? (L1 or L2, …)
- What is the best value of **k** to use? (1, 3, 5, 7, …)

i.e. How do we set the **hyperparameters**?

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

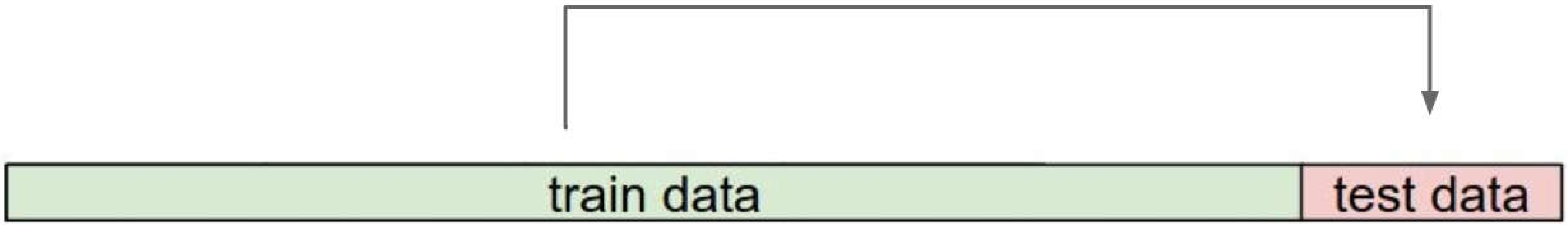
Slide from CS231n

How to Find Hyperparameters?



- Trying out what hyperparameters work best on test set

How to Find Hyperparameters?

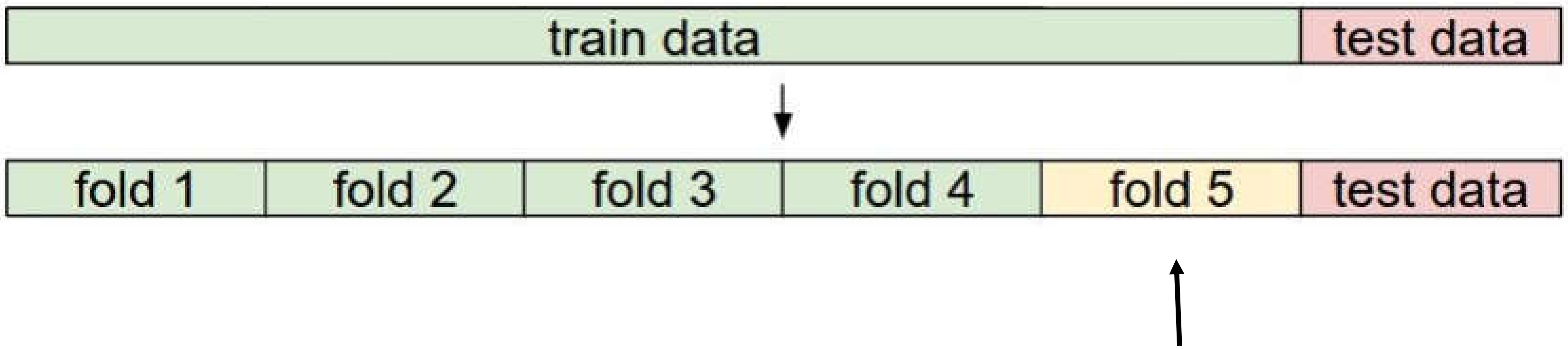


- Trying out what hyperparameters work best on test set

- Very bad idea.
- The test set is a proxy for the generalization performance!
- Use only **VERY SPARINGLY**, at the end.

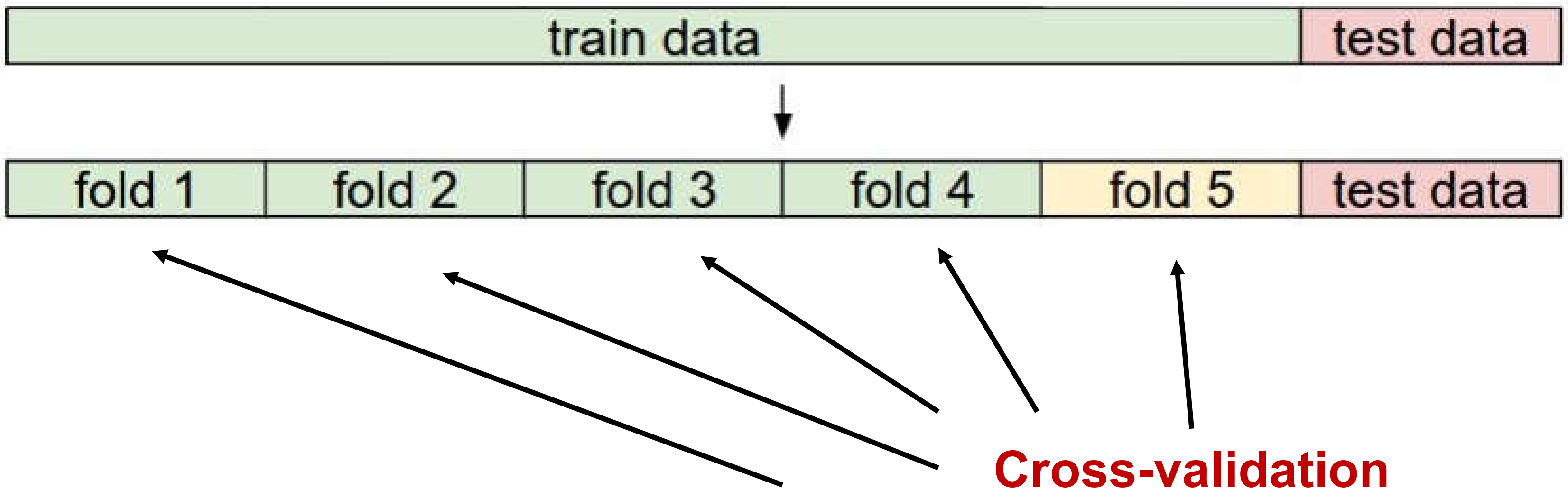
Slide from CS231n

Validation Set



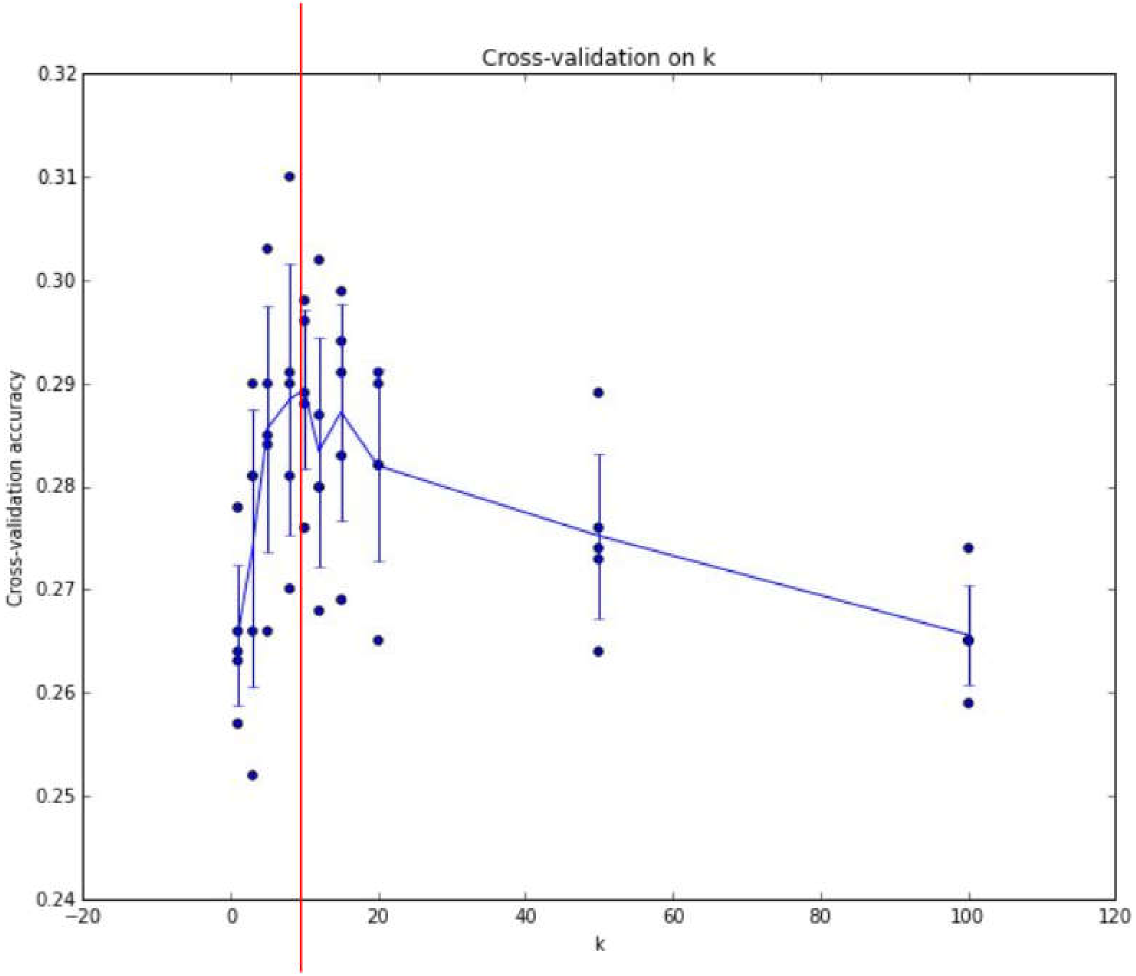
Validation set
use to tune hyperparameters

Cross-Validation



Cross-validation
Cycle through the choice of which fold is the validation fold, average results

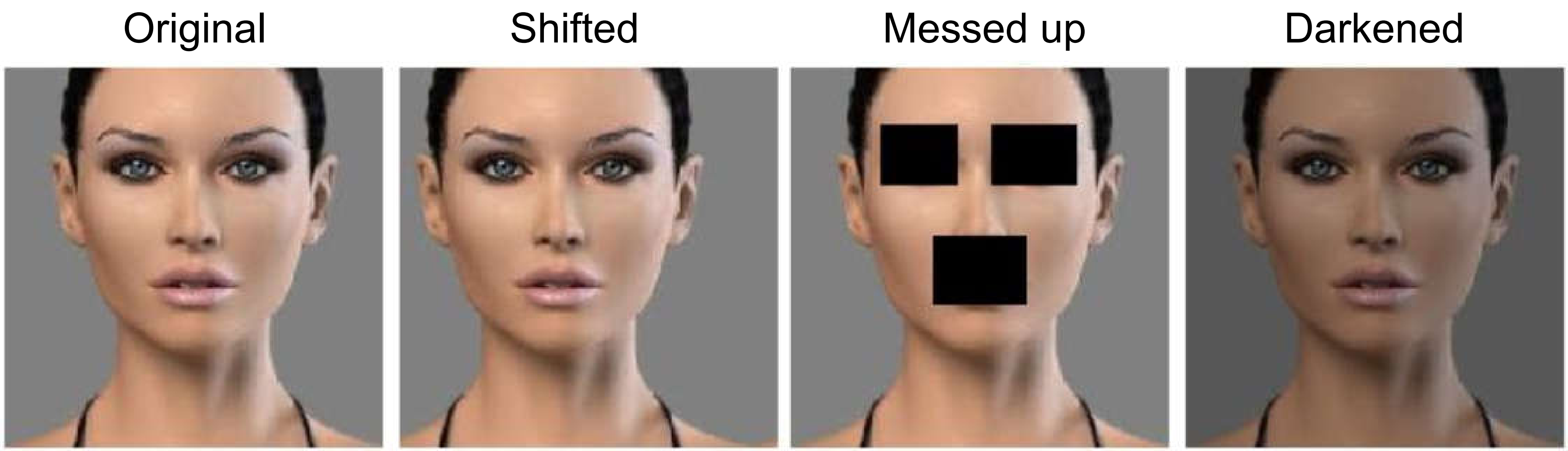
Cross-Validation



- 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

Never Use k-Nearest Neighbor in Practice

- k-Nearest Neighbor on images **never used** (pixel-based)
  - ➢ **Terrible performance** at test time
  - ➢ Distance metrics on level of whole images can be very **unintuitive**



- All 3 images have same **L2 distance** to the one on the left

Summary

- **Image Classification:** We are given a **Training Set** of labeled images, asked to predict labels on **Test Set**. Common to report the **Accuracy** of predictions (fraction of correctly predicted images)

- We introduced the **k-Nearest Neighbor Classifier**, which predicts the labels based on nearest images in the training set

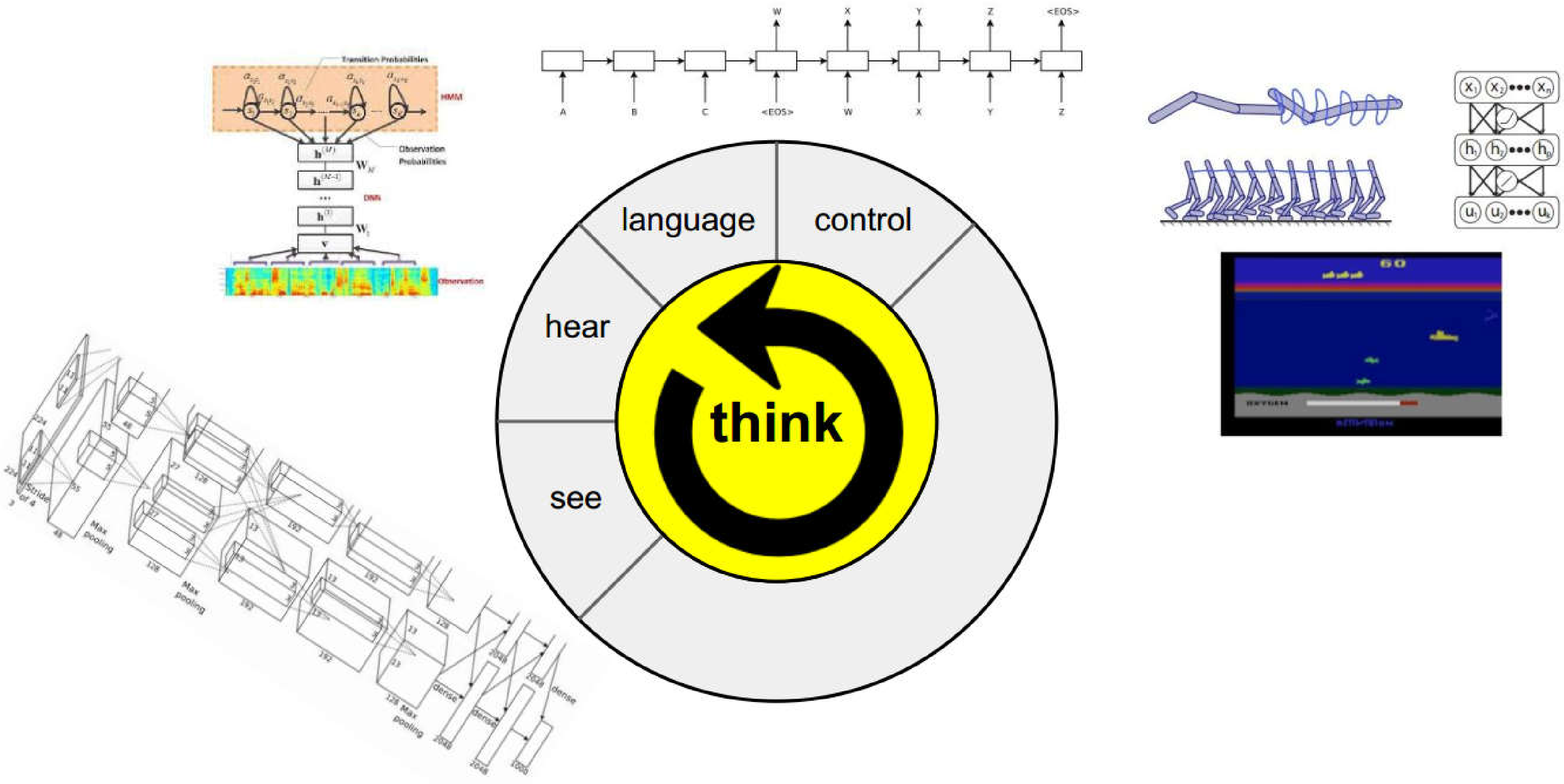
- We saw that the choice of distance and the value of k are **hyperparameters** that are tuned using a **validation set**, or through **cross-validation** if the size of the data is small.

- Once the **best set of hyperparameters** is chosen, the classifier is evaluated **once on the test set**, and reported as the performance of kNN on that data.

Slide from CS231n

Linear Classification

Deep Learning in Different Areas



Slide from CS231n

Lego Block

Neural Networks practitioner

Image Captioning

- Neural network look at the image and create **a description of the image**

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"girl in pink dress is jumping in air."

"black and white dog jumps over bar."

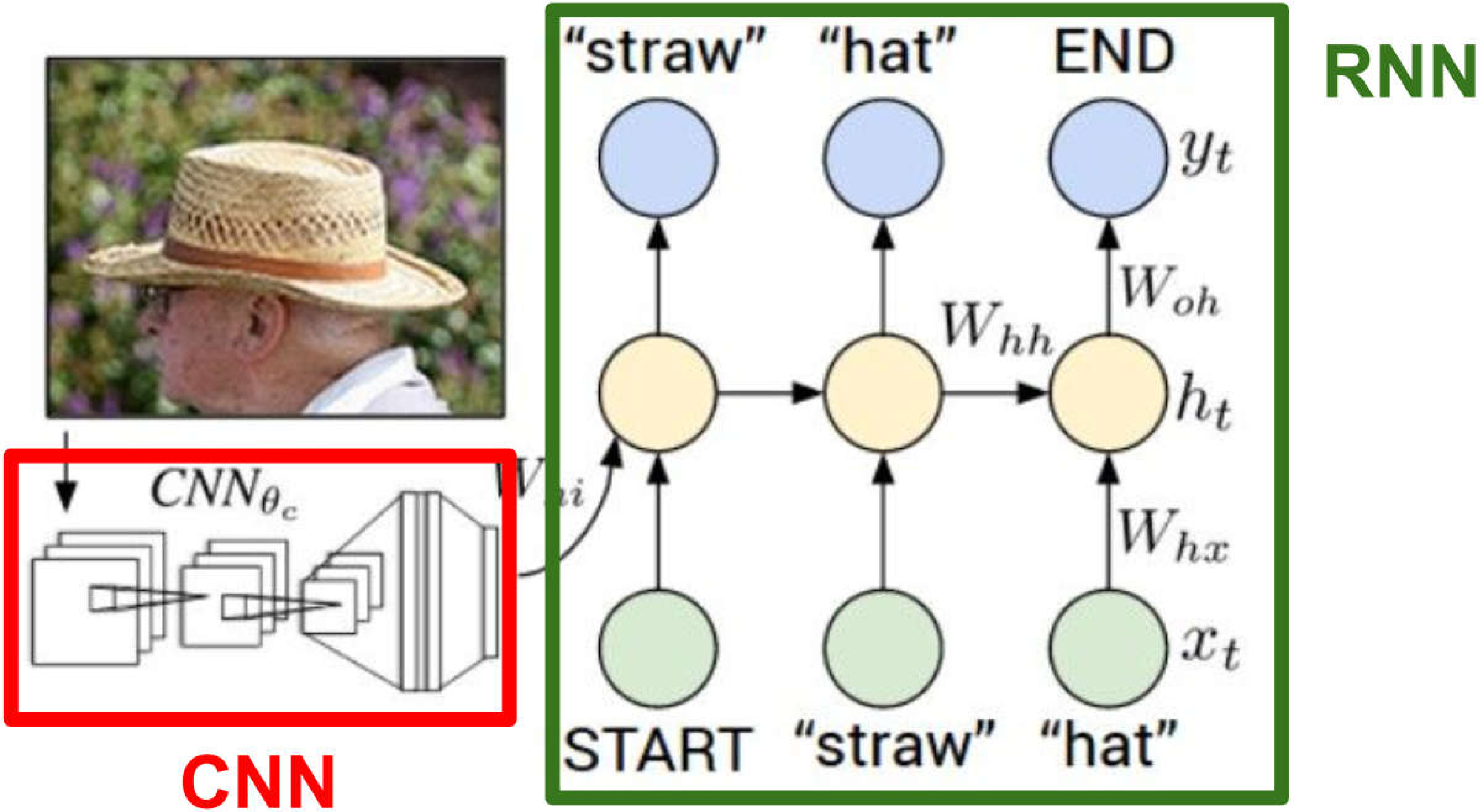
"young girl in pink shirt is swinging on swing."

"man in blue wetsuit is surfing on wave."

Slide from CS231n

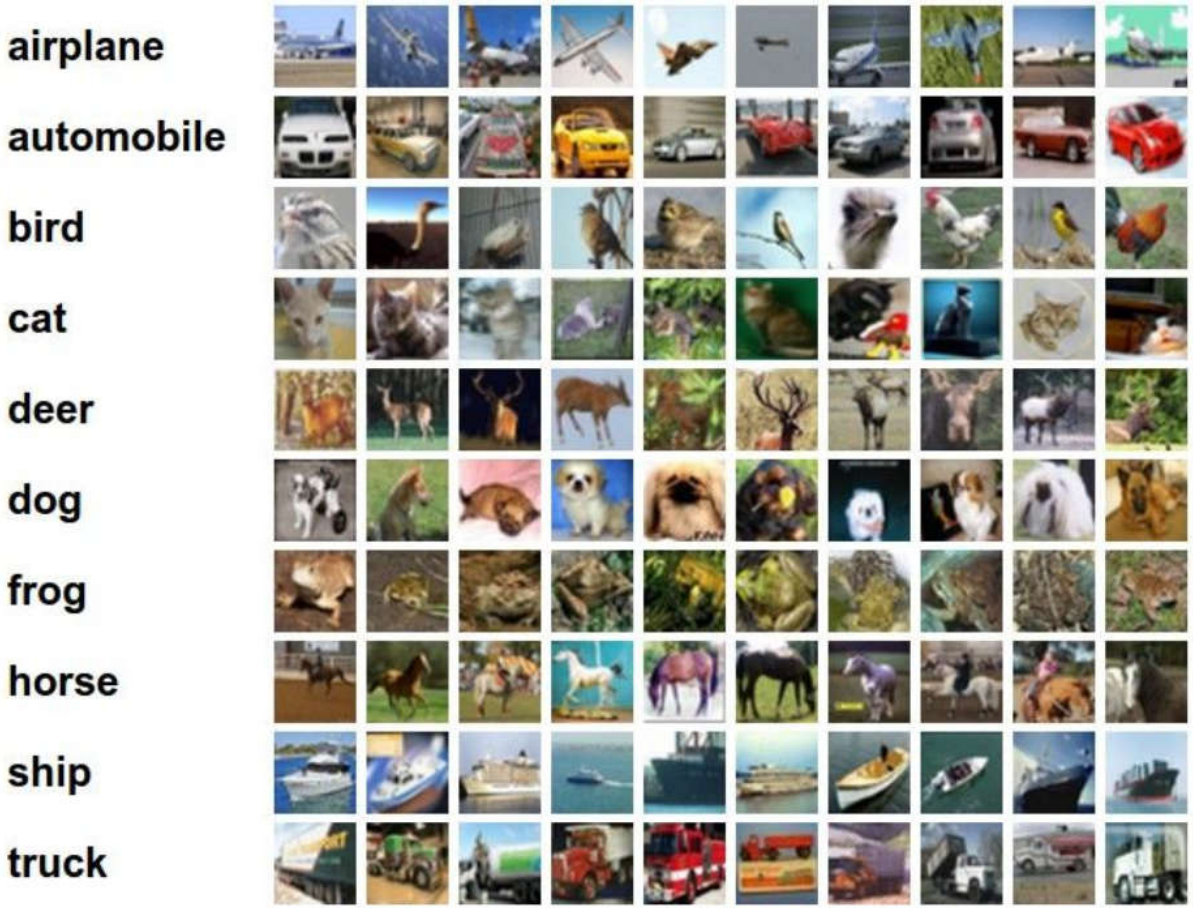
Details in Image Captioning

- We will have full understanding **image captioning** through this course roughly



Slide from CS231n

CIFAR10 Linear Classification



- **10** labels
- **50,000** training images
- **10,000** test images
- **32x32x3**

Slide from CS231n

Parametric Approach

Image Parameters

$f(\mathbf{x}, \mathbf{W}) = \textbf{scores}$

10 numbers,
indicating class scores

[32x32x3]

(3072 numbers total)

- **Linear model**
- Neural network
- Convolutional neural network

- k-nearest neighbor is a **non-parametric approach**, there's no parameters that we're going to optimizing over.

Parametric Approach: Linear Classifier

[32x32x3]

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

10 numbers,
indicating class scores

Parametric Approach: Linear Classifier

[32x32x3]

3072x1

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

10x1 **10x3072**

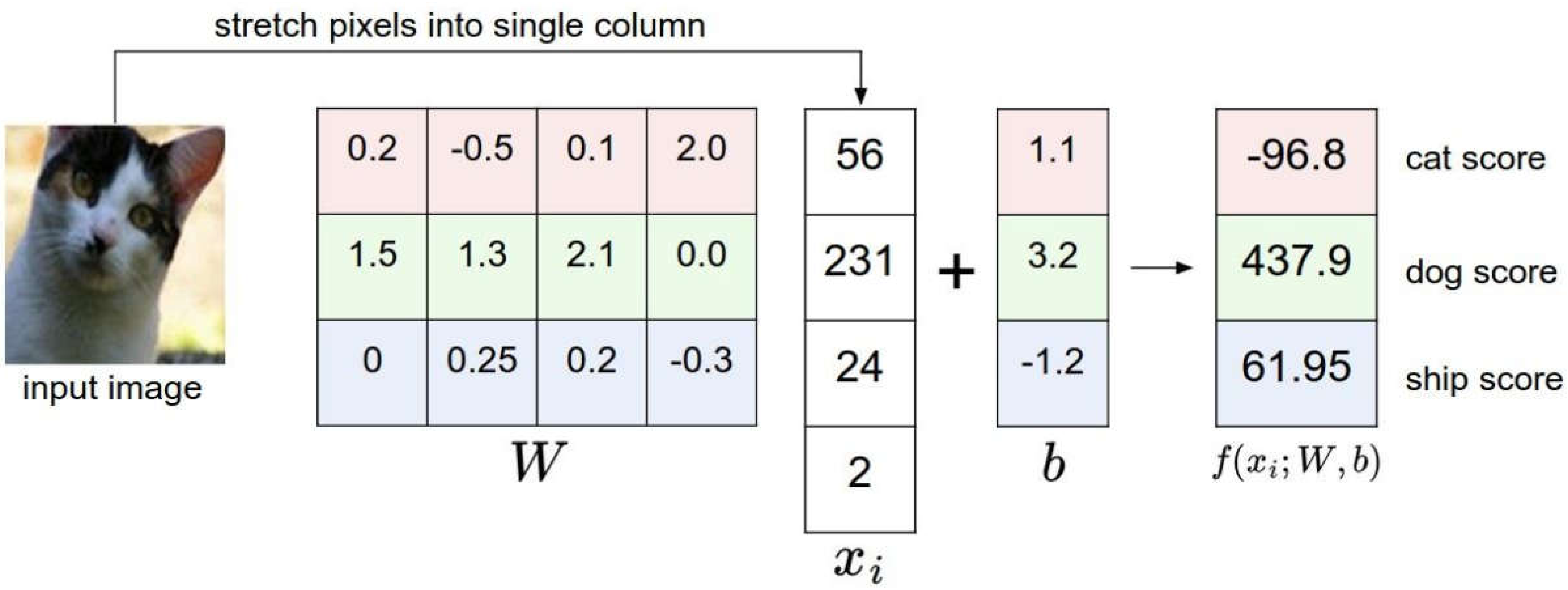
10 numbers,
indicating class scores

Parameters, or “weights”

Slide from CS231n

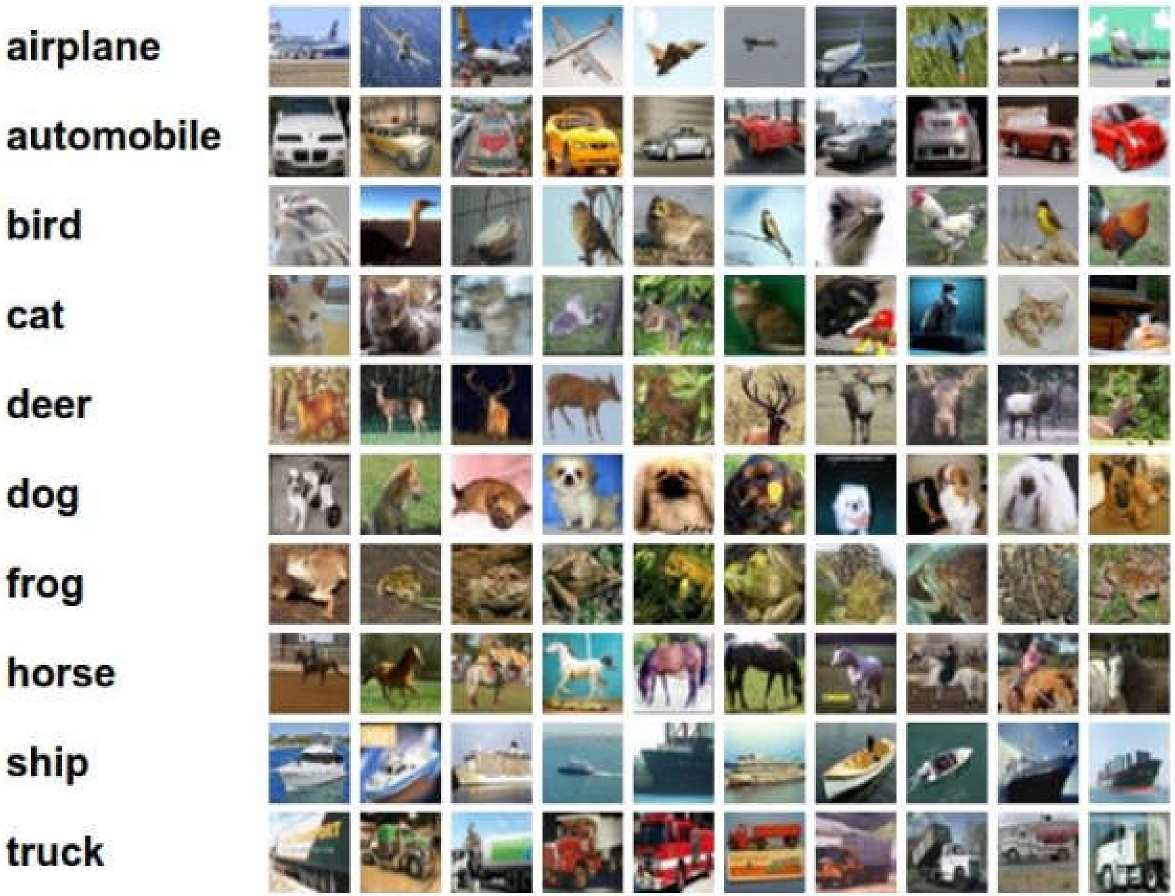
Example

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



Random setting of *W*

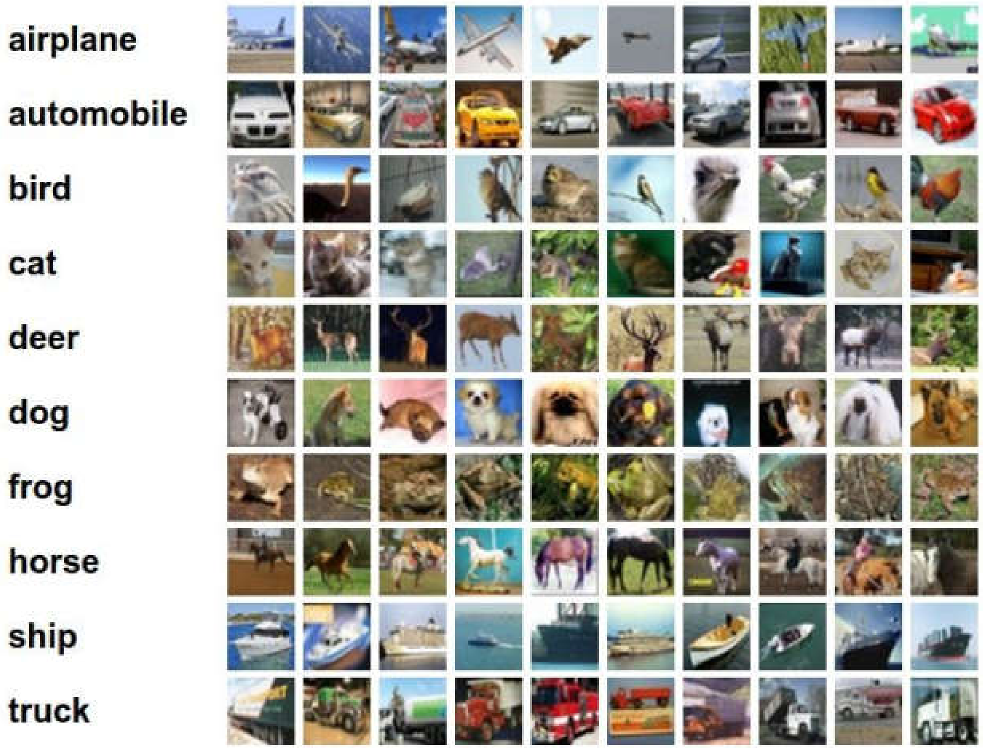
Interpreting a Linear Classifier



$$f\left(x_i, W, b\right) = Wx_i + b$$

Question #5: What does the linear classifier do?

Interpreting a Linear Classifier



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

10x3072

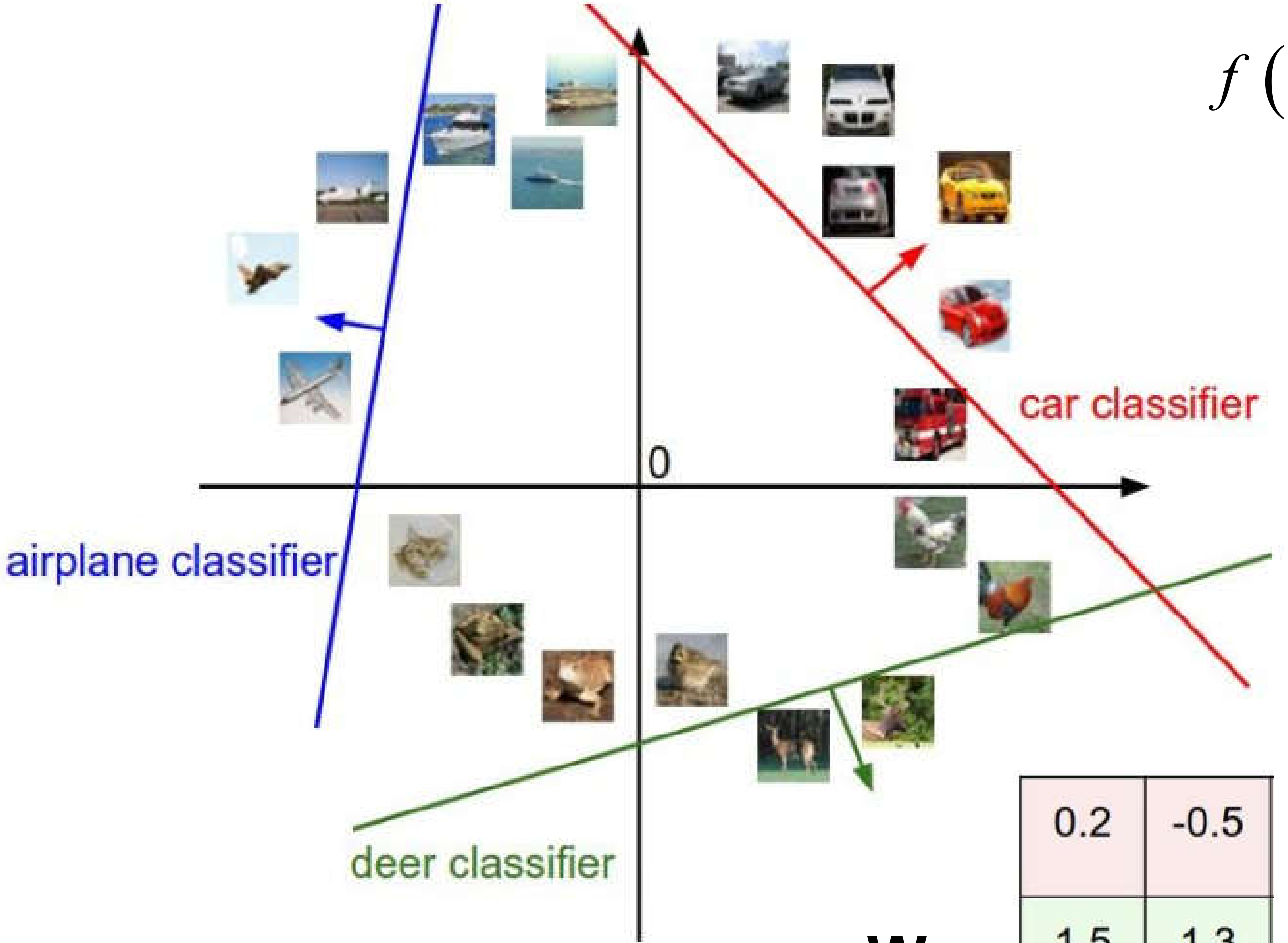
[32x32x3]

- Example trained weights of a linear classifier trained on CIFAR-10:

- Similar with **Template Matching**

Slide from CS231n

Interpreting a Linear Classifier



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

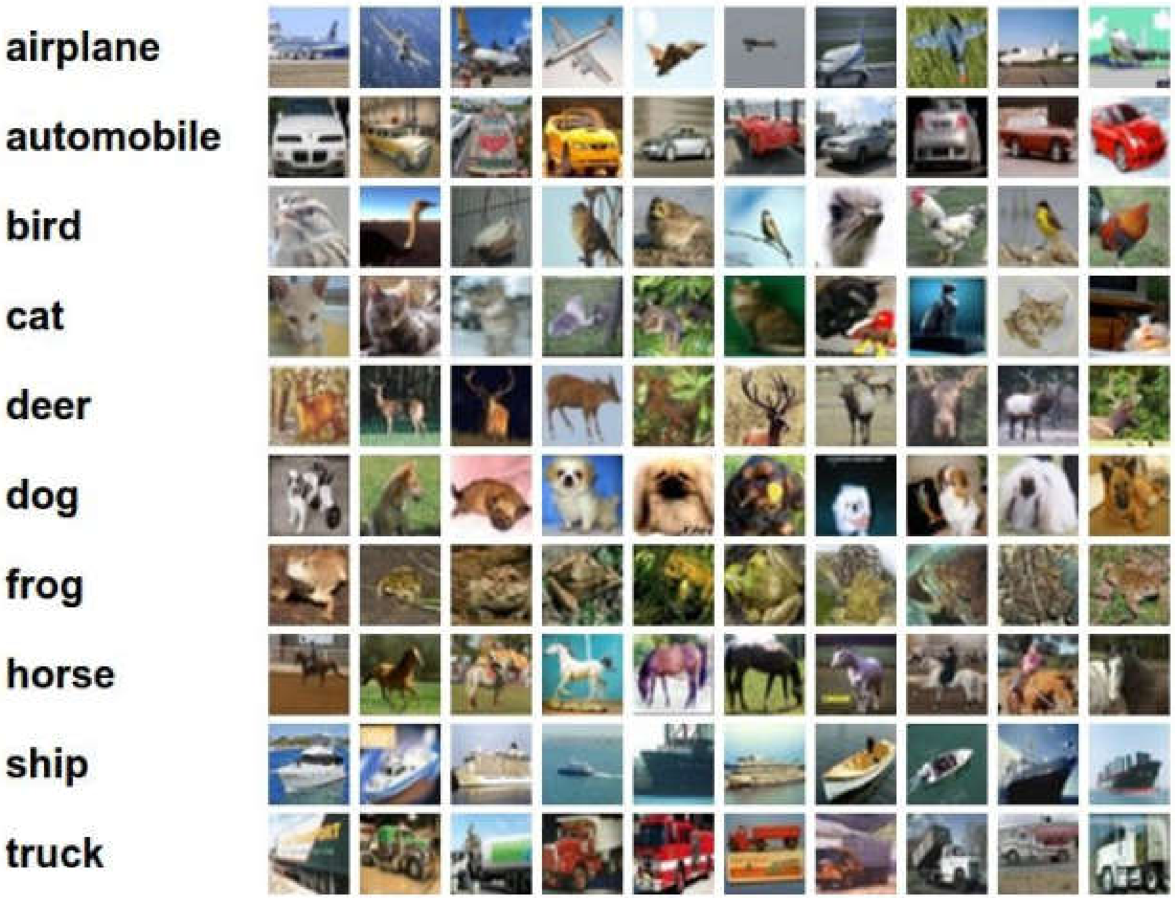
10x3072

$$W = \begin{bmatrix} 0.2 & -0.5 \\ 1.5 & 1.3 \\ 0 & 0.25 \end{bmatrix}$$

[32x32x3]
(3072 numbers total)

Slide from CS231n

Interpreting a Linear Classifier



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

- **Question #6:** What would be a very hard set of classes for a linear classifier to distinguish?

Linear Classifier

- We defined a (linear) **score function**: $f\left(x_i, W, b\right) = Wx_i + b$

Example class scores for 3 images, with a **random W**:

We will **define a loss function** that can measure how much good or bad for current **W**

We can **minimize loss** and **optimize W**

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

Slide from CS231n

Next Week

Coming up:

- Loss function
- Optimization
- ConvNets!

$$f\left(x_i, W, b\right) = Wx_i + b$$

Quantifying what it means to have a “good” W

Start with random W and find a W that minimizes the loss

Tweak the functional form of f

Slide from CS231n

Q & A