# Lecture 2:
# Image Classification pipeline

# Course Goals

- Fundamental Concepts
  - Homeworks 1-4
- Practical Programming Experience
  - Paper implementation
  - Final Project
- Software development with a group
- Literature Review
- Scientific Experimentation

# Assignments and Grading

- 10% Homework #1
- 10% Homework #2
- 10% Homework #3
- 10% Homework #4
- 10% Reading summaries posted to class blog
- 10% Paper presentation(s), including partial system implementation or testing
  - Pick partner for paper presentation
  - Look at list of suggested papers, email top 2 or 3 picks to instructor
  - Papers will be assigned next week
- 40% Semester Project
  - Groups will be assigned randomly after first homework is graded
  - If you have a special case, please come to office hours to discuss

# Programming Requirements

- Prereqs: Python, all homework assignments are in Python. Deep learning functions will be written by you!
- Deep learning packages (Final Project): Caffe, TensorFlow, Torch, Theano…
    - Will be discussed later this semester
    - Feel free to get started experimenting
    - Come to office hours with questions or post on Piazza
    - Caffe, TensorFlow, Torch installed as part of start-up script
- You will be sharing the AWS instances
    - You can also apply for AWS, Google Cloud, Azure education credits
    - To check who else is using the CPUs use `top` or `htop`
    - Check GPU usage with `nvidia-smi`

# Admin

First assignment is out!
It is due Thursday Fed. 2
It includes:

- Write/train/evaluate a kNN classifier
- Write/train/evaluate a Linear Classifier (SVM and Softmax)
- Write/train/evaluate a 2-layer Neural Network (backpropagation!)
- Requires writing numpy/Python code

Compute: Can use your own laptops, or Tufts' AWS instance

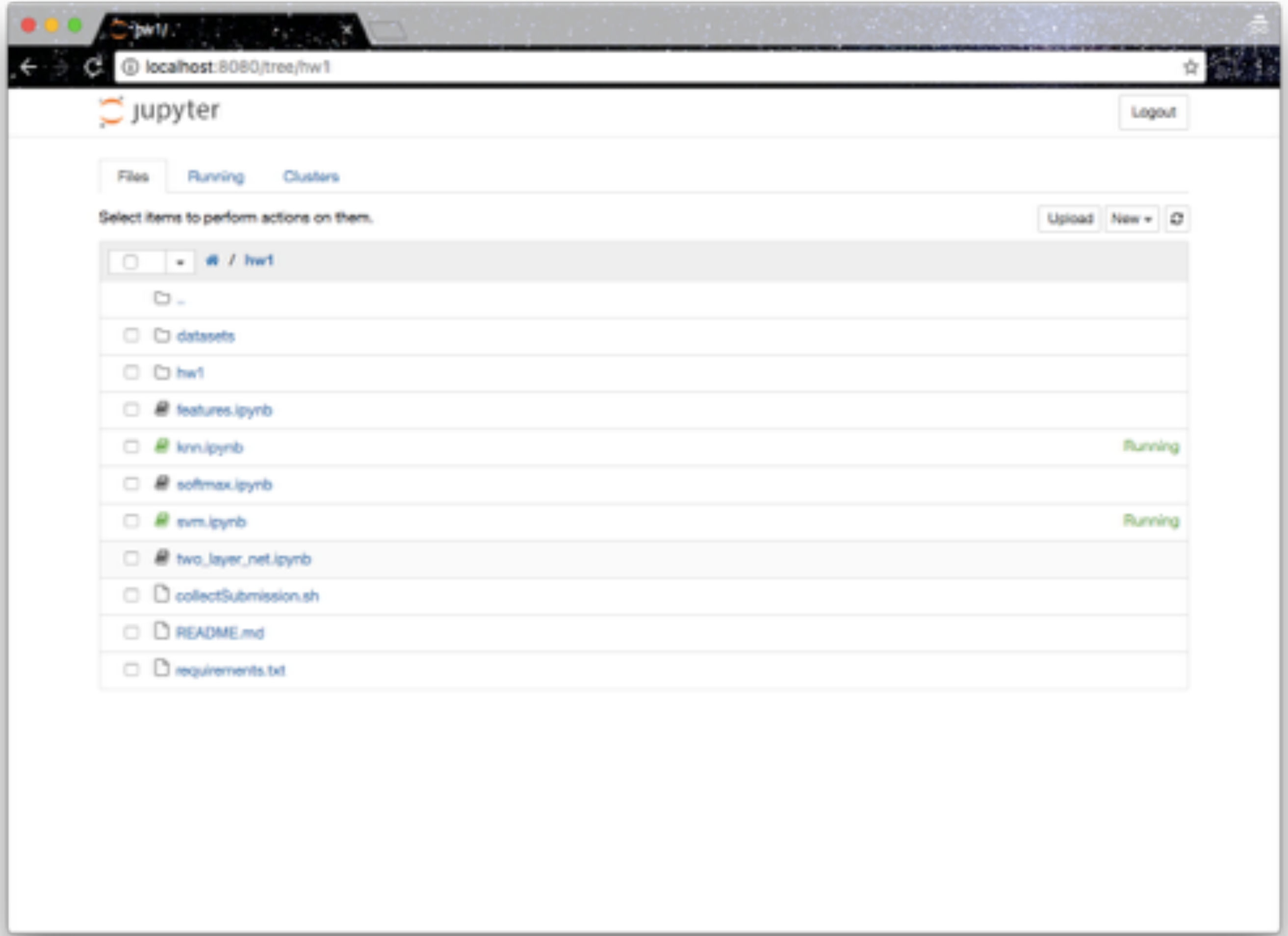
# Getting Set Up - Tutorials

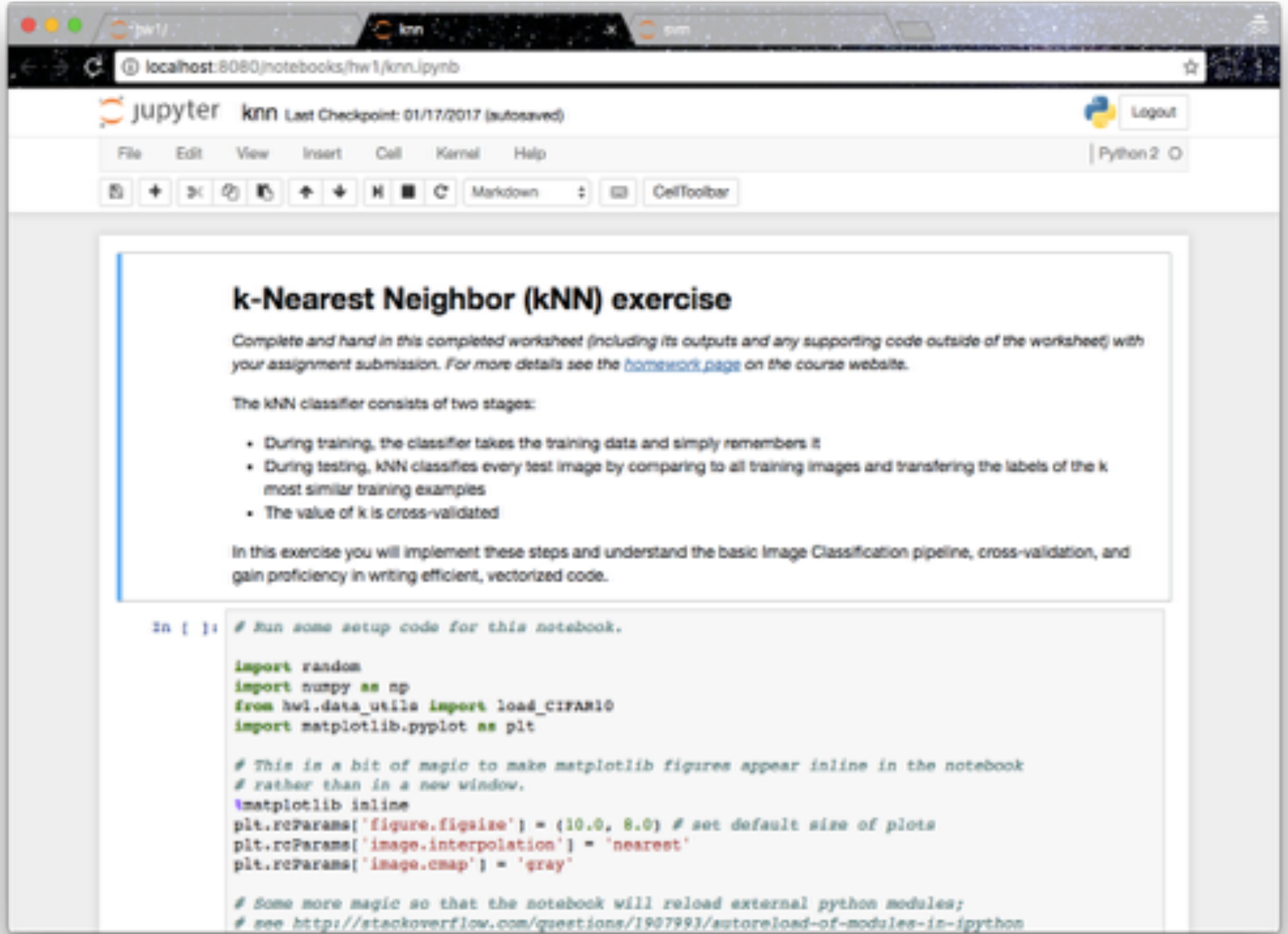
- Python/Numpy
- VirtualEnv

https://comp150dl.github.io/notes/

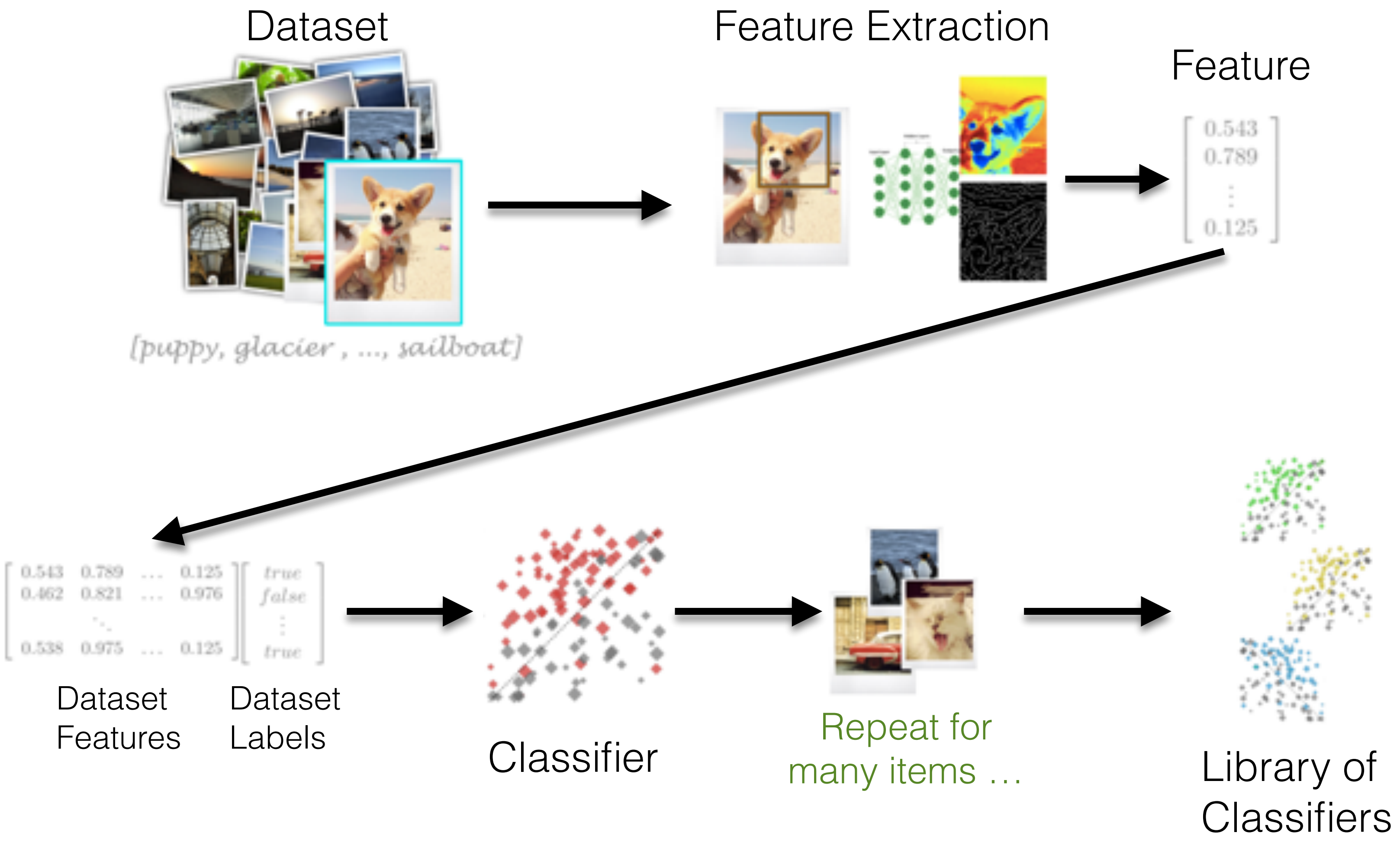
- Vectorized Operations: Using Slices in Python

```
# 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
```


localhost:8080/tree/hw1
jupyter
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
/ hw1
datasets
hw1
features.ipynb
knn.ipynb
Running
softmax.ipynb
svm.ipynb
Running
two_layer_net.ipynb
collectSubmission.sh
README.md
requirements.txt


localhost:8080/notebooks/hw1/knn.ipynb
jupyter knn Last Checkpoint: 01/17/2017 (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Python 2
Markdown
CellToolbar
k-Nearest Neighbor (kNN) exercise
Complete and hand in this completed worksheet (including its outputs and any supporting code outside of the worksheet) with your assignment submission. For more details see the homework page on the course website.
The kNN classifier consists of two stages:
During training, the classifier takes the training data and simply remembers it
During testing, kNN classifies every test image by comparing to all training images and transfering the labels of the k most similar training examples
The value of k is cross-validated
In this exercise you will implement these steps and understand the basic Image Classification pipeline, cross-validation, and gain proficiency in writing efficient, vectorized code.
In [ ]:
# Run some setup code for this notebook.
import random
import numpy as np
from hw1.data_utils import load_CIFAR10
import matplotlib.pyplot as plt
# This is a bit of magic to make matplotlib figures appear inline in the notebook
# rather than in a new window.
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'
# Some more magic so that the notebook will reload external python modules;
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython

# Image Classification Training Pipeline



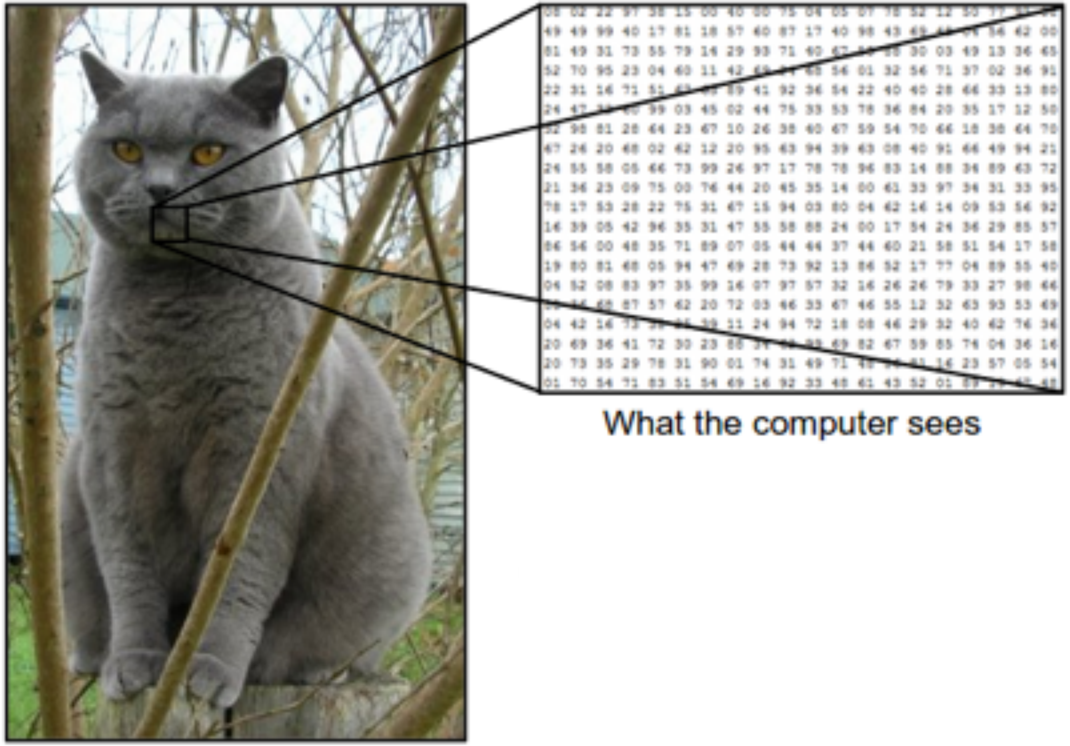
(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

→ cat

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

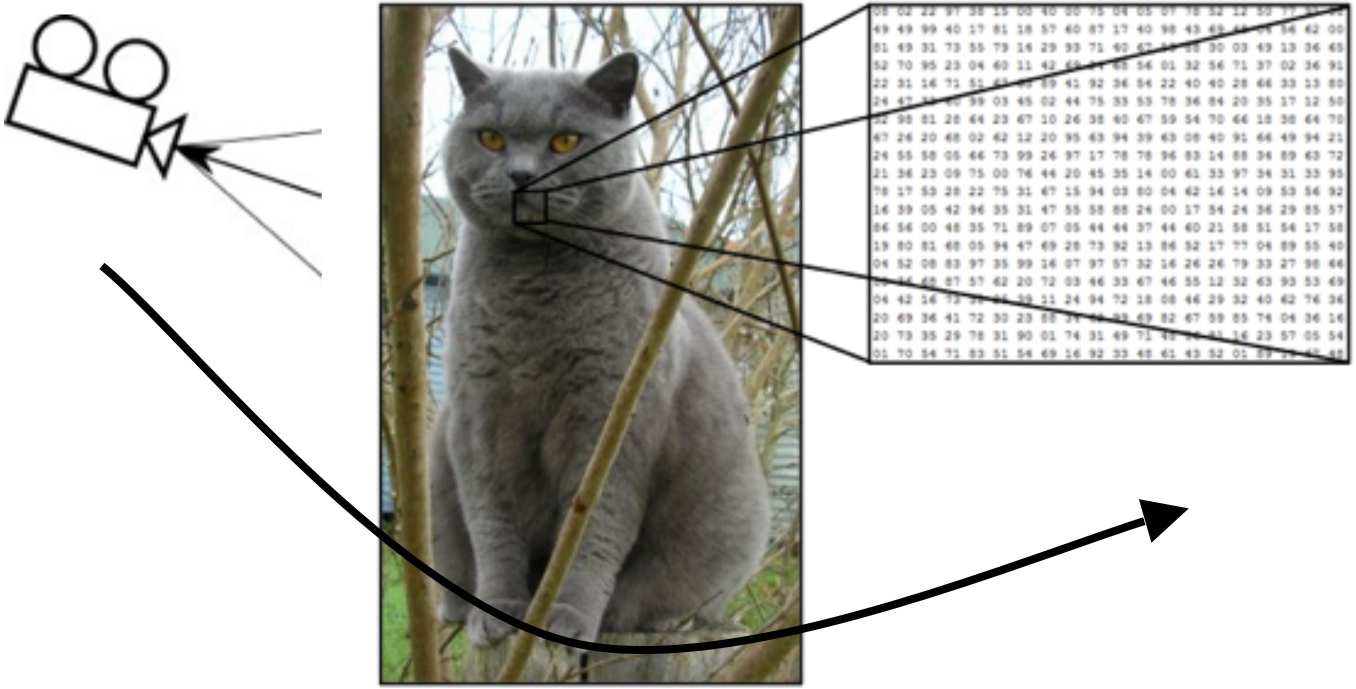
E.g.
300 x 100 x 3

(3 for 3 color channels RGB)



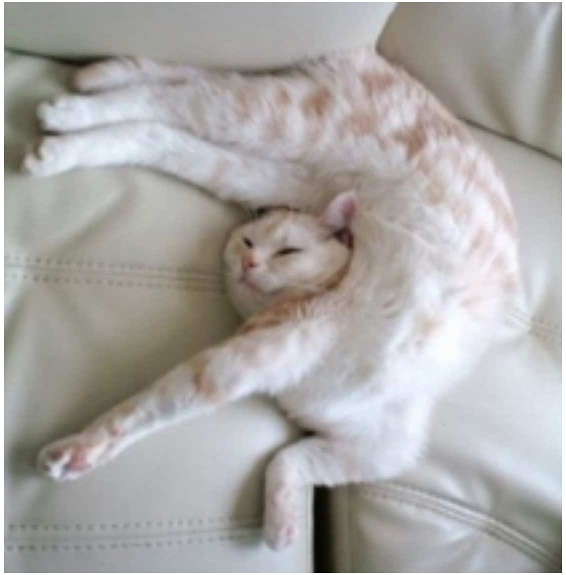
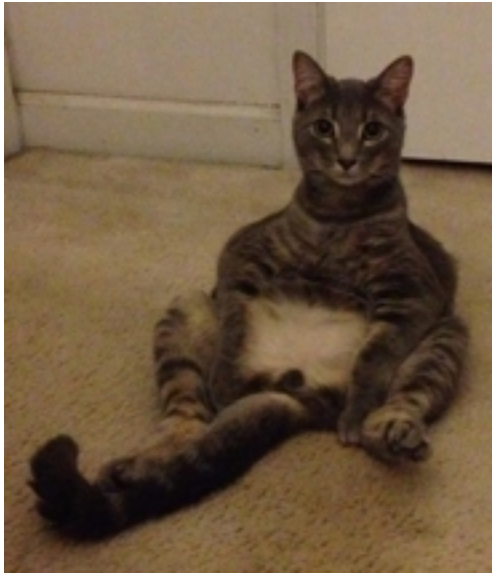
What the computer sees

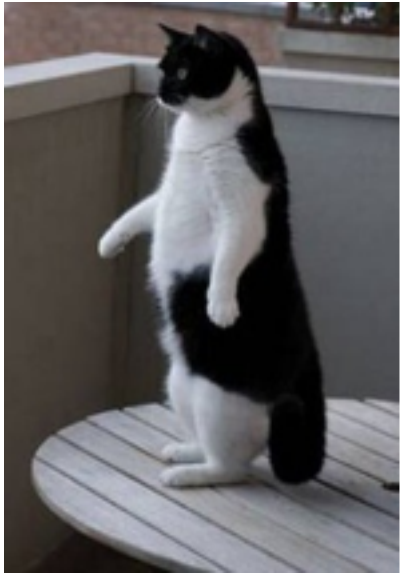
# Challenges: Viewpoint Variation



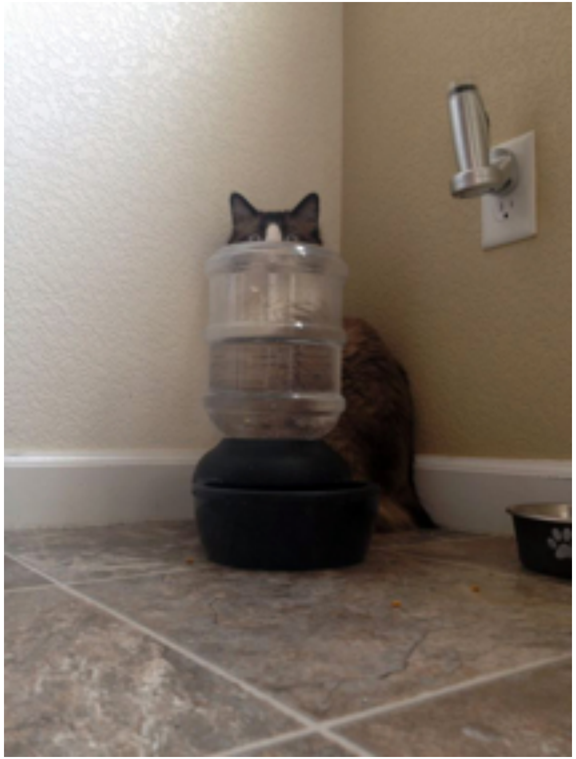
# Challenges: Illumination

# Challenges: Deformation



# Challenges: Occlusion



# Challenges: Background clutter

# Challenges: Intraclass variation

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

# An image classifier

```
def predict(image):
    # ????
    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.

# Data Driven Approach

- Collect a dataset of images and labels
- Use Machine Learning to train an image classifier
- 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**


* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

# First classifier: **Nearest Neighbor Classifier**

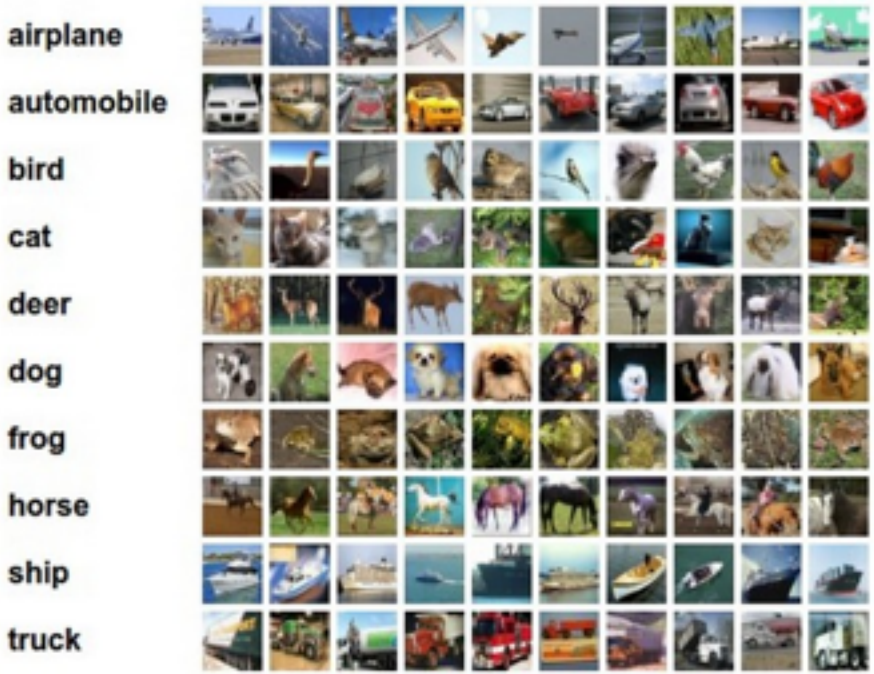
```python
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

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

Remember all training images and their labels

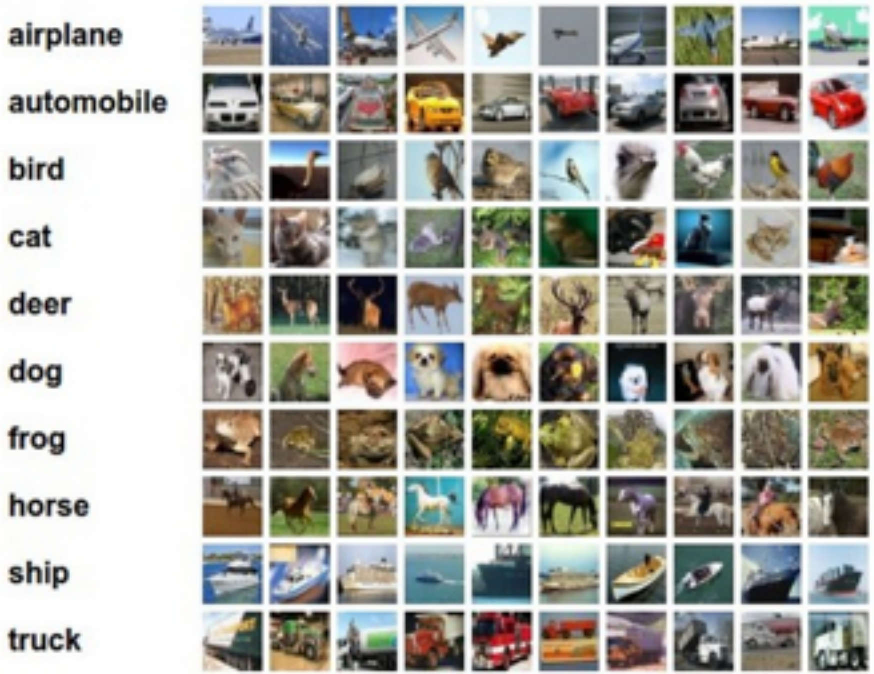
Predict the label of the most similar training image

Example dataset: **CIFAR-10**
**10** labels
**50,000** training images, each image is tiny: 32x32
**10,000** test images.




* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford 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

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

## 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
```

```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

remember the training data

# 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
```

for every test image:

- find nearest train image with L1 distance
- predict the label of nearest training image

# 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
```

**Q: how does the classification speed depend on the size of the training data?**

```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: how does the classification speed depend on the size of the training data?
**linearly :(**

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

# Aside: Approximate Nearest Neighbor

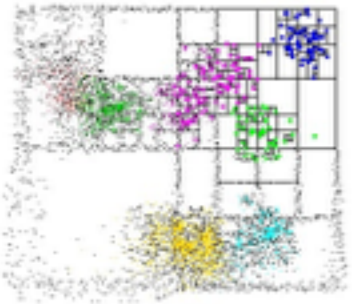
find approximate nearest neighbors quickly

## ANN: A Library for Approximate Nearest Neighbor Searching

David M. Mount and Sunil Arya

Version 1.1.2

Release Date: Jan 27, 2010



### 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]

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

L1 (Manhattan) distance

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

L2 (Euclidean) distance

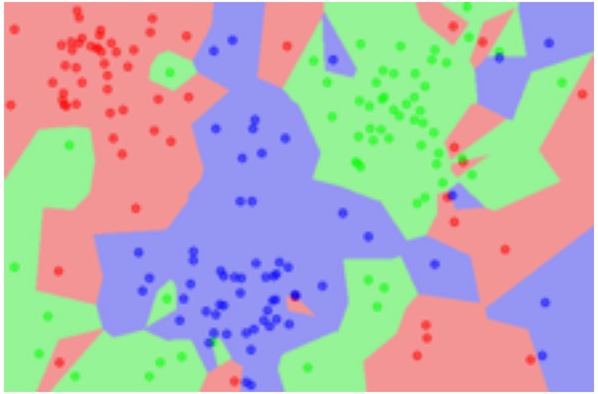
$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$

# k-Nearest Neighbor

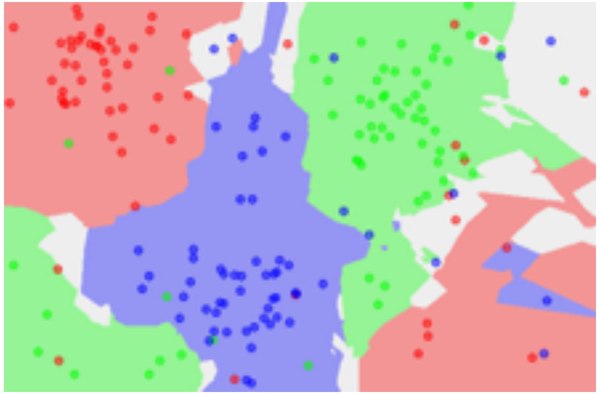
## find the k nearest images, have them vote on the label

the data

NN classifier

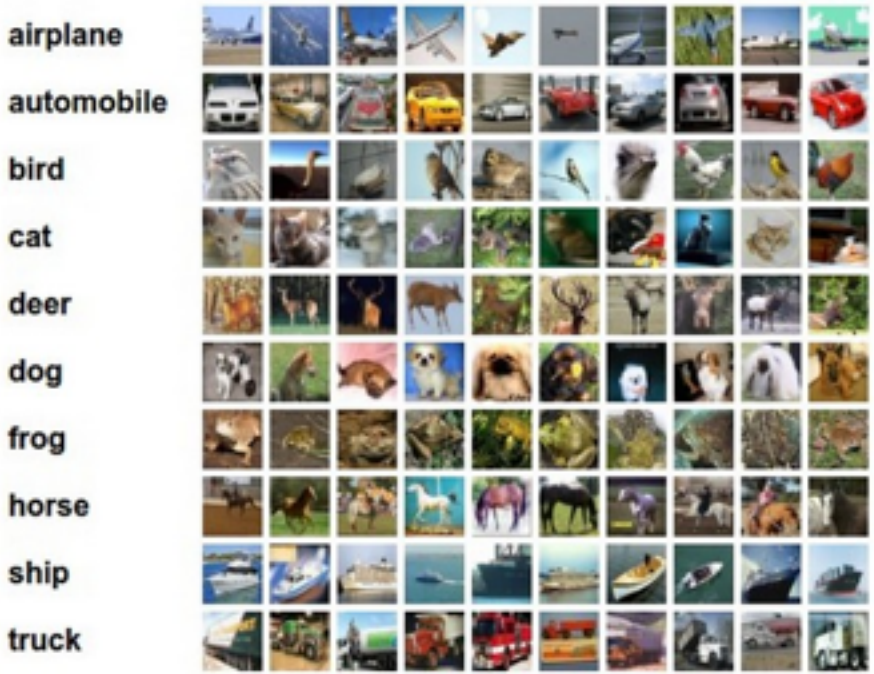


5-NN classifier



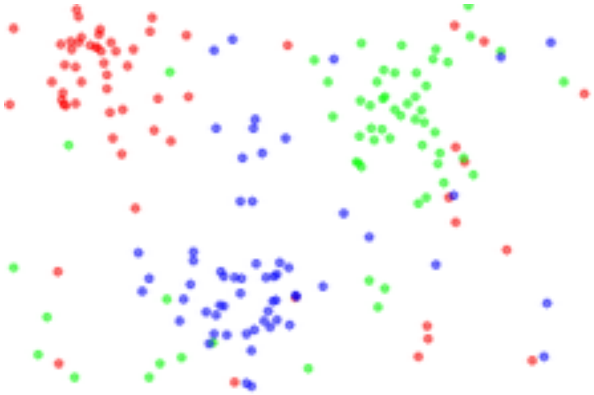
http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

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

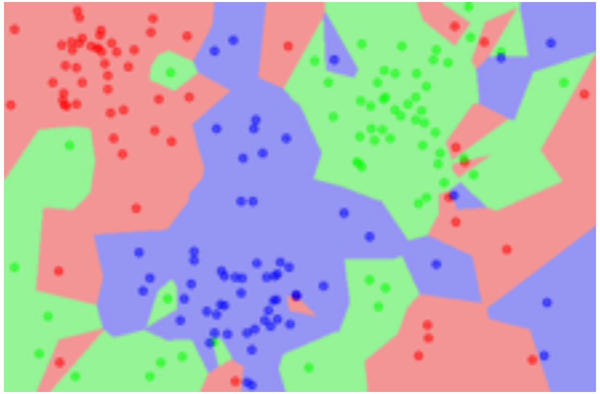



For every test image (first column), examples of nearest neighbors in rows

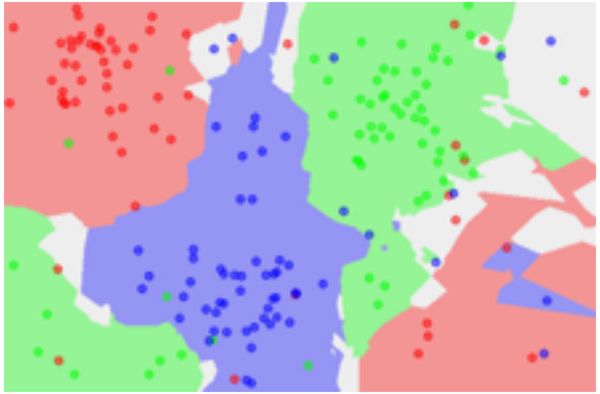
the data



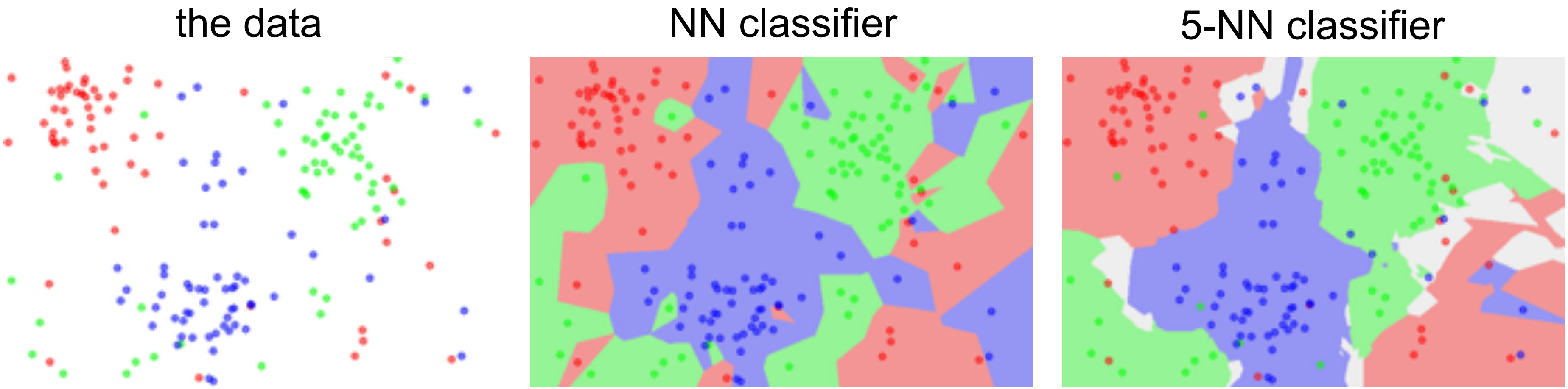
NN classifier



5-NN classifier



Q: what is the accuracy of the nearest neighbor classifier on the training data, when using the Euclidean distance?




Q2: what is the accuracy of the **k**-nearest neighbor classifier on the training data?

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

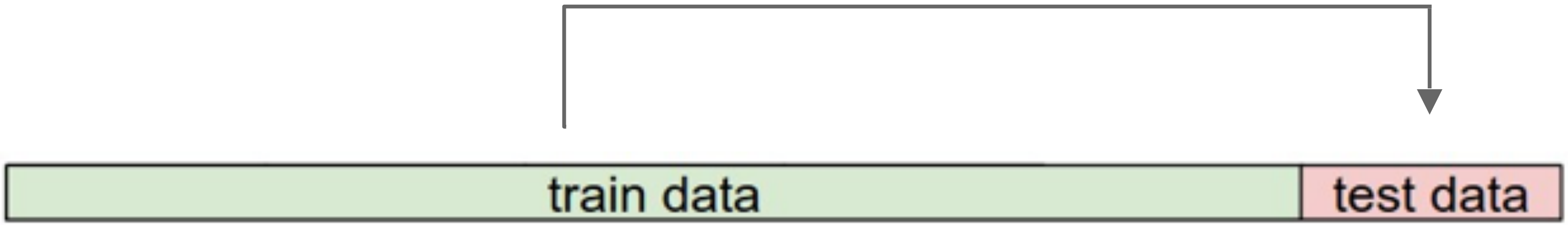
i.e. how do we set the **hyperparameters**?

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

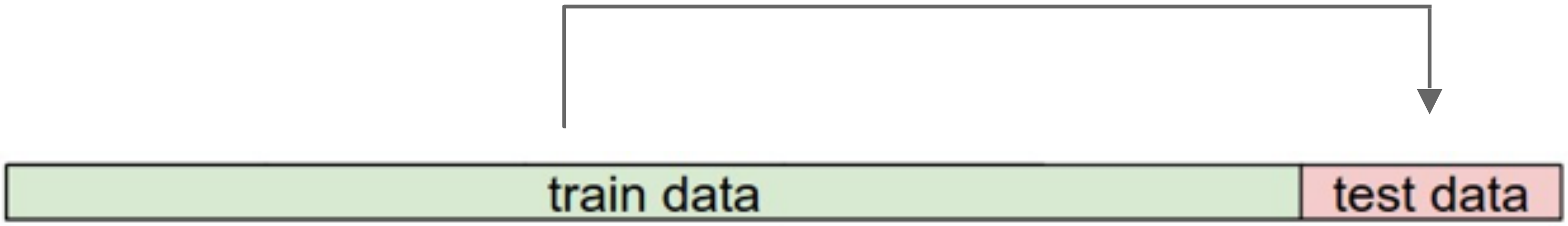
i.e. how do we set the **hyperparameters**?

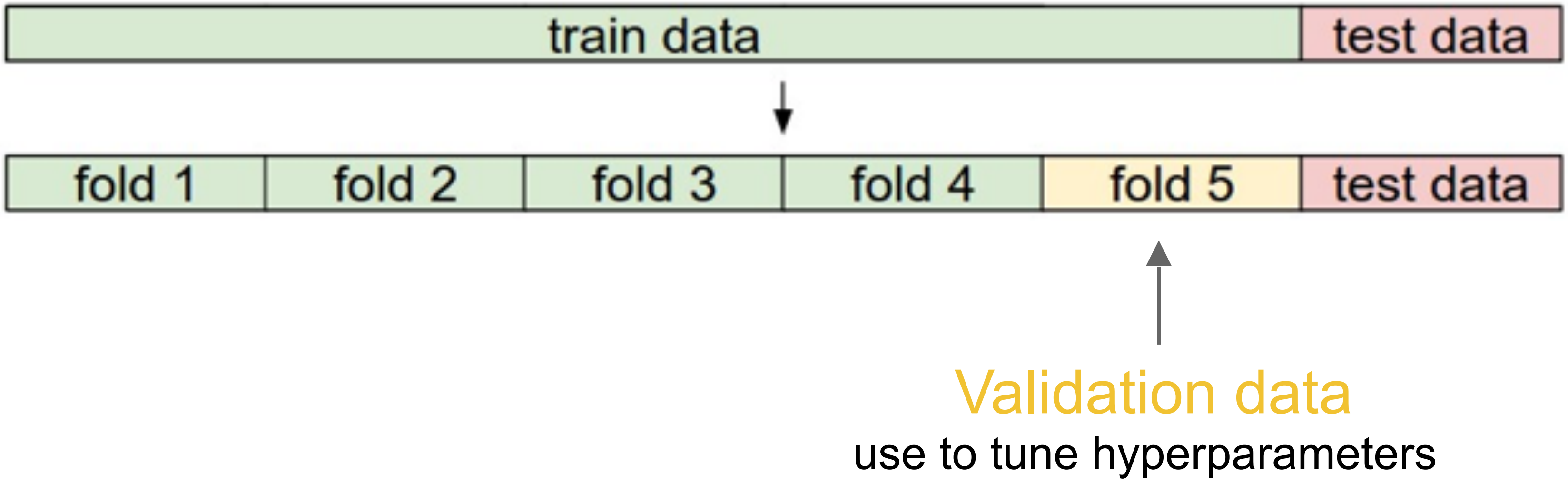
Very problem-dependent.
Must try them all out and see what works best.

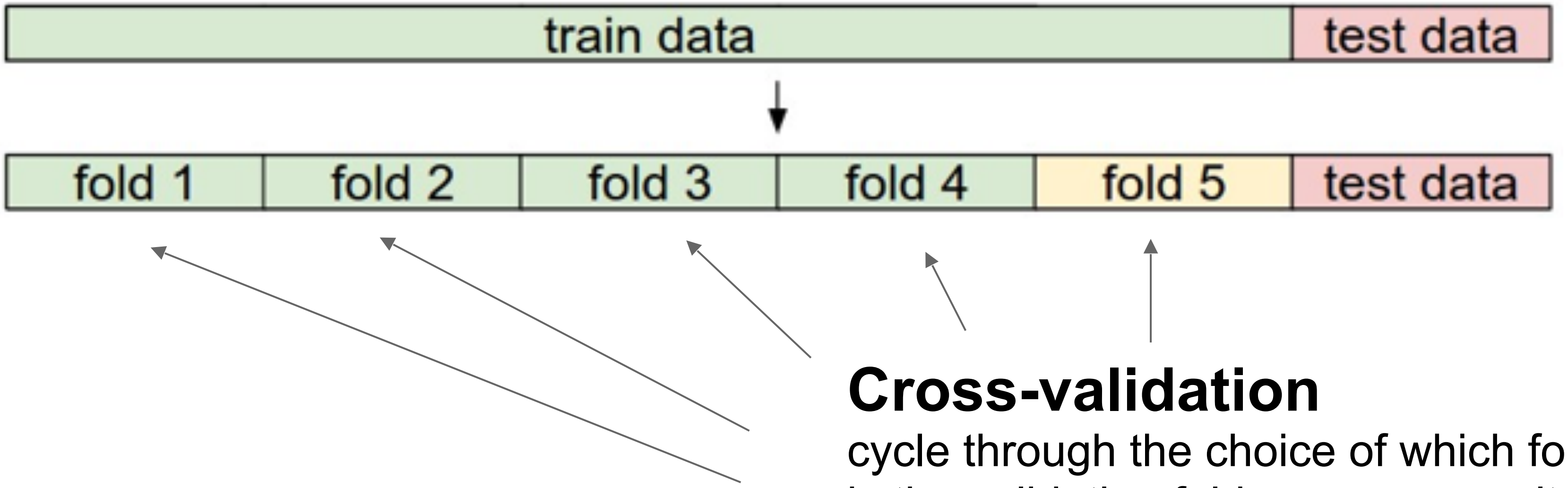
Try out what hyperparameters work best on test set.



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.

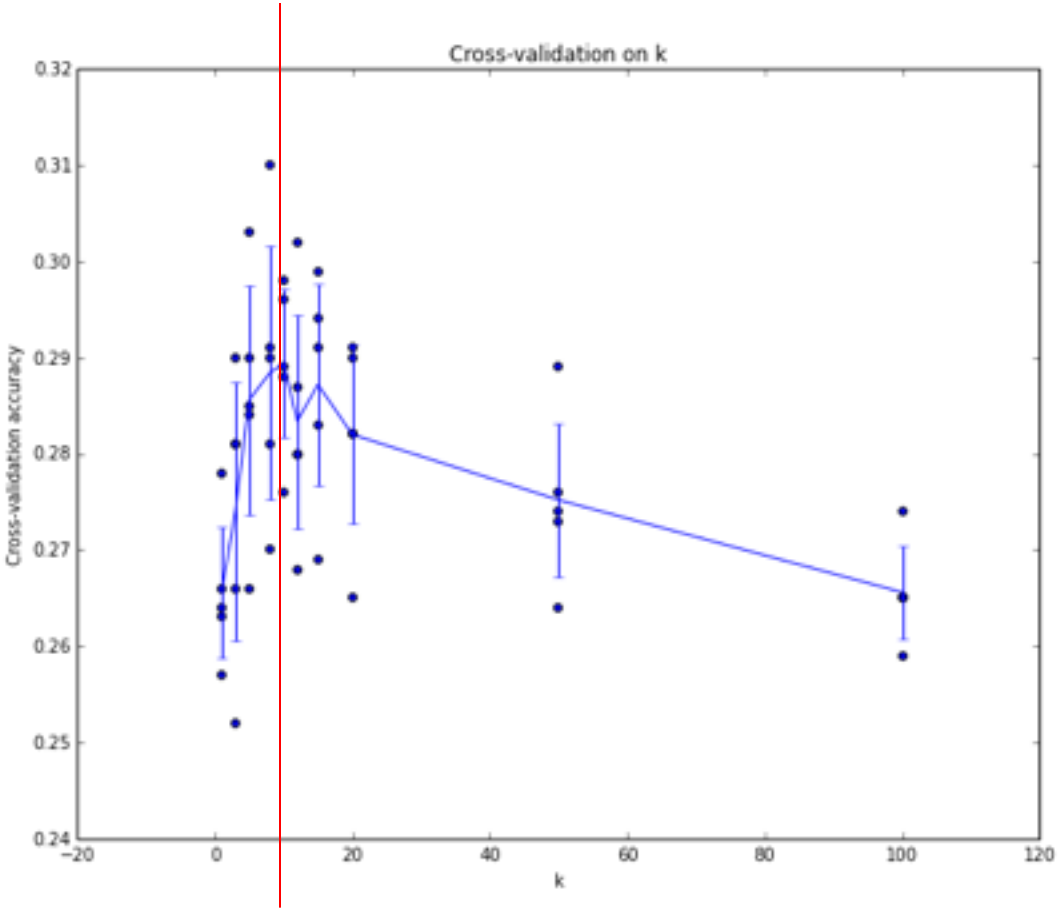



train data
test data
fold 1
fold 2
fold 3
fold 4
fold 5
test data
Validation data
use to tune hyperparameters




**Cross-validation**

cycle through the choice of which fold is the validation fold, average results.




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)

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

- terrible performance at test time
- distance metrics on level of whole images can be very unintuitive


(all 3 images have same L2 distance to the one on the left)

# 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.

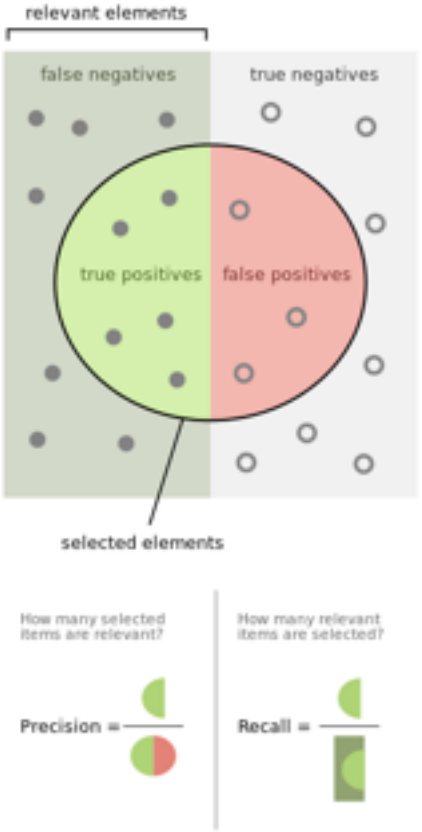
# Aside: Precision and Recall

- Sometimes we are interested in more than **accuracy.**

- **Precision:** true positives/ total positives, ex: out of 50 images marked ‘cat’, 10 were correct, Prec = 0.2

- **Recall:** true positives/ total population, ex: out of 100 cat images in the test set, 10 were marked ‘cat’, Rec = 0.1

- Average Precision: The average precision value over range of imposed recall values 0-1.0.



# Linear Classification

"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."

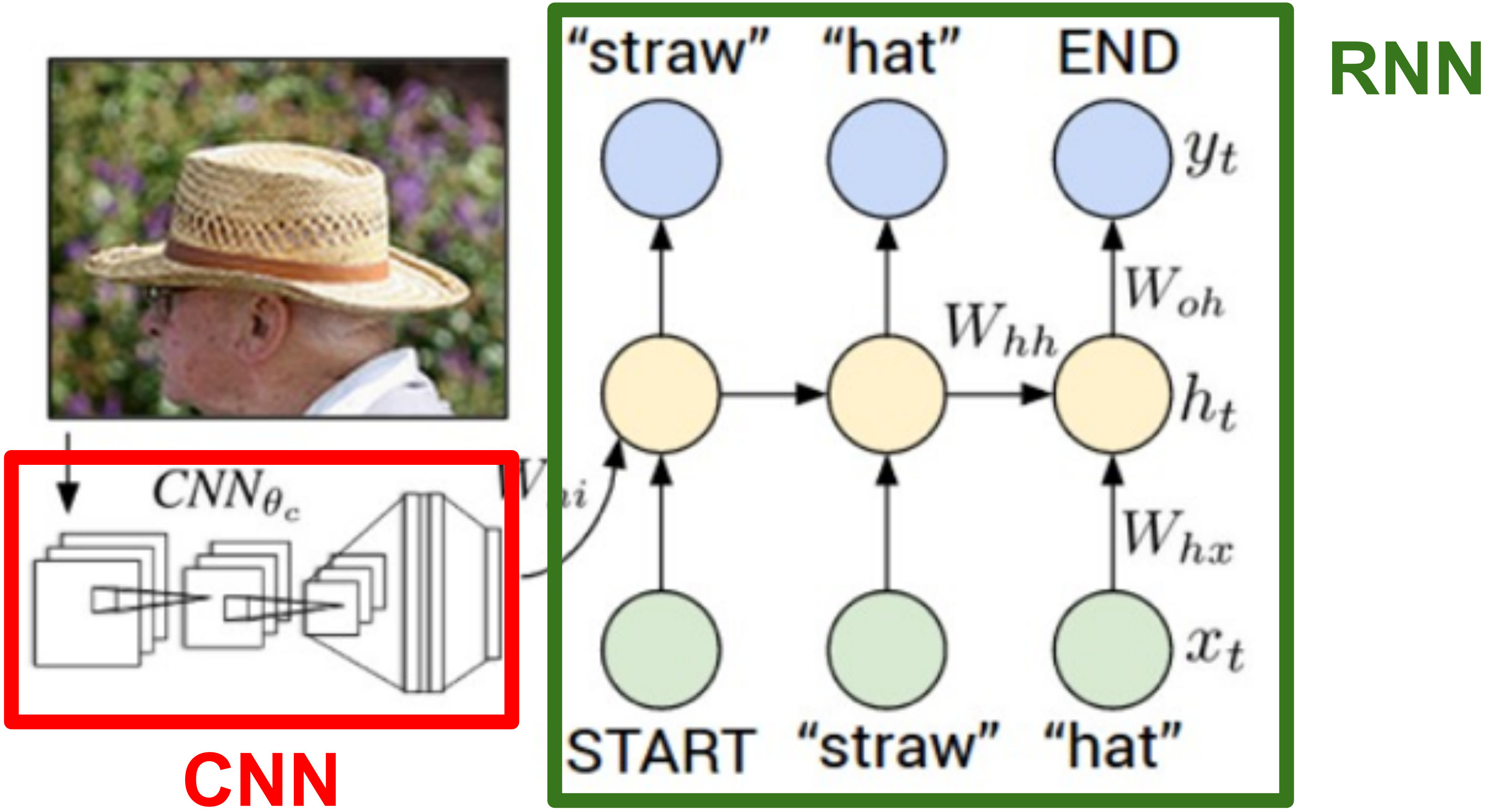
"girl in pink dress is jumping in air."

"black and white dog jumps over bar."

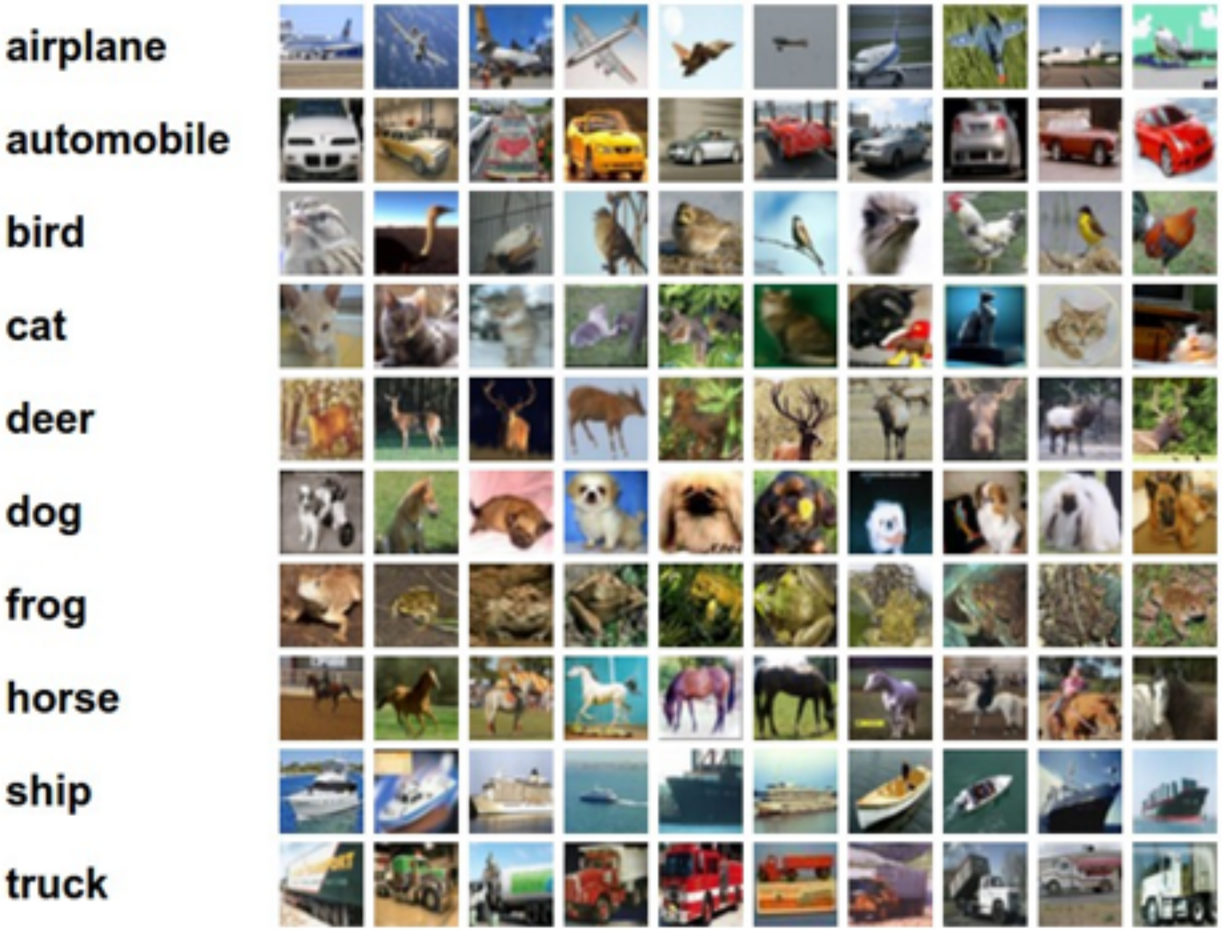
"young girl in pink shirt is swinging on swing."

"man in blue wetsuit is surfing on wave."

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n



"straw"
"hat"
END
$y_t$
$W_{hh}$
$W_{oh}$
$h_t$
$CNN_{\theta_c}$
$W_{hx}$
$x_t$
START
"straw"
"hat"
RNN
CNN




Example dataset: **CIFAR-10**
**10** labels
**50,000** training images
each image is **32x32x3**
**10,000** test images.

# Parametric approach

image parameters

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

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

**10** numbers, indicating class scores

* Original slides borrowed from Andrej Karpathy and Li Fei-Fei, Stanford cs231n

# Parametric approach: **Linear classifier**

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

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

**10** numbers,
indicating class
scores

# Parametric approach: **Linear classifier**

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

**3072x1**

**10x1** **10x3072**

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

**10** numbers, indicating class scores

parameters, or "weights"

# Parametric approach: **Linear classifier**

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

**3072x1** **10x1**

**10x1** **10x3072**

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

**10** numbers, indicating class scores

parameters, or “weights”

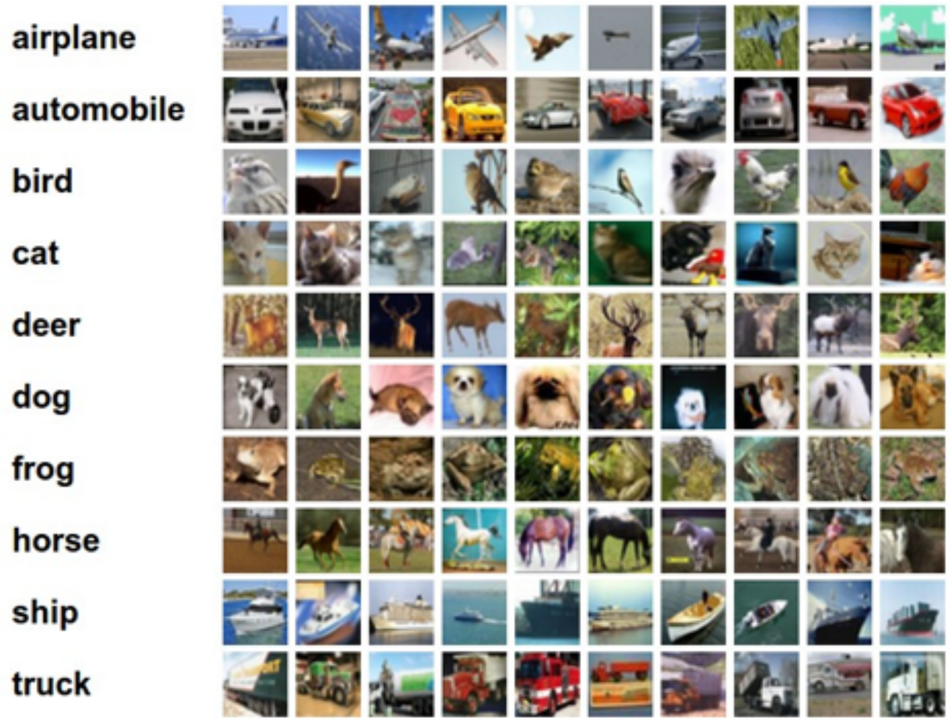
# Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

stretch pixels into single column

input image

$$\underbrace{\begin{bmatrix}0.2 & -0.5 & 0.1 & 2.0\\ 1.5 & 1.3 & 2.1 & 0.0\\ 0 & 0.25 & 0.2 & -0.3\end{bmatrix}}_{W}\underbrace{\begin{bmatrix}56\\ 231\\ 24\\ 2\end{bmatrix}}_{x_i}+\underbrace{\begin{bmatrix}1.1\\ 3.2\\ -1.2\end{bmatrix}}_{b}\rightarrow\underbrace{\begin{bmatrix}-96.8\\ 437.9\\ 61.95\end{bmatrix}}_{f(x_i;W,b)}\begin{array}{l}\text{cat score}\\ \text{dog score}\\ \text{ship score}\end{array}$$

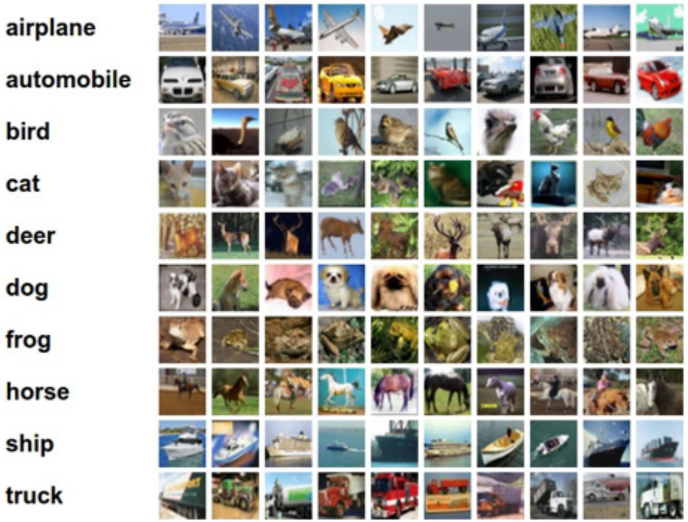
# Interpreting a Linear Classifier




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

Q: what does the linear classifier do, in English?

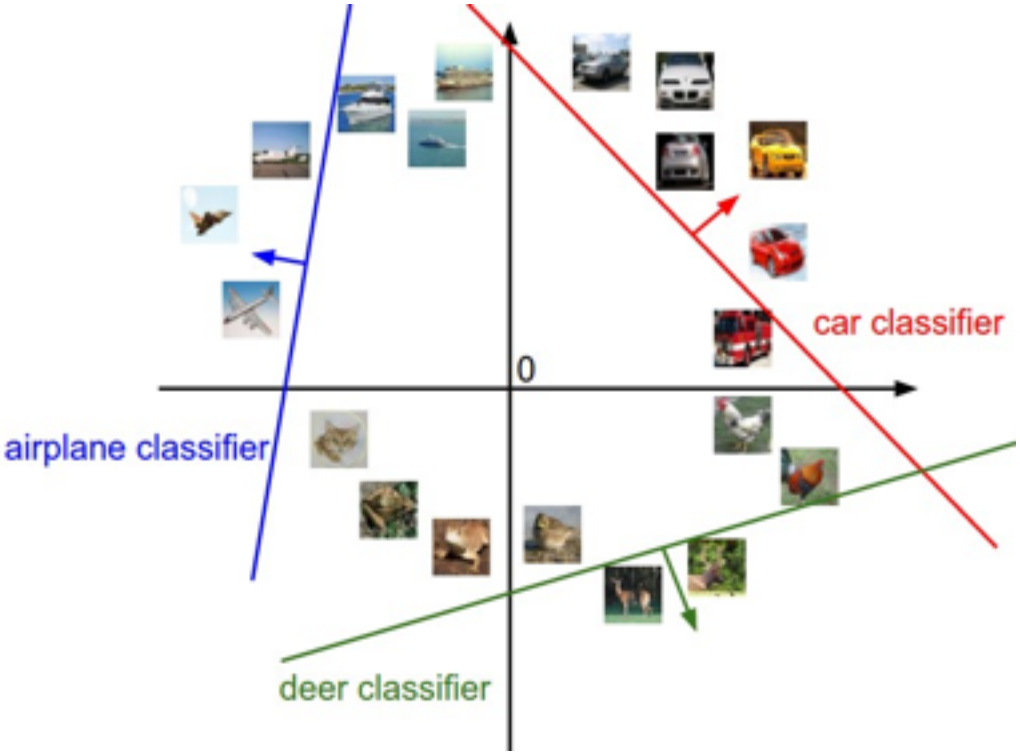
# Interpreting a Linear Classifier




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

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

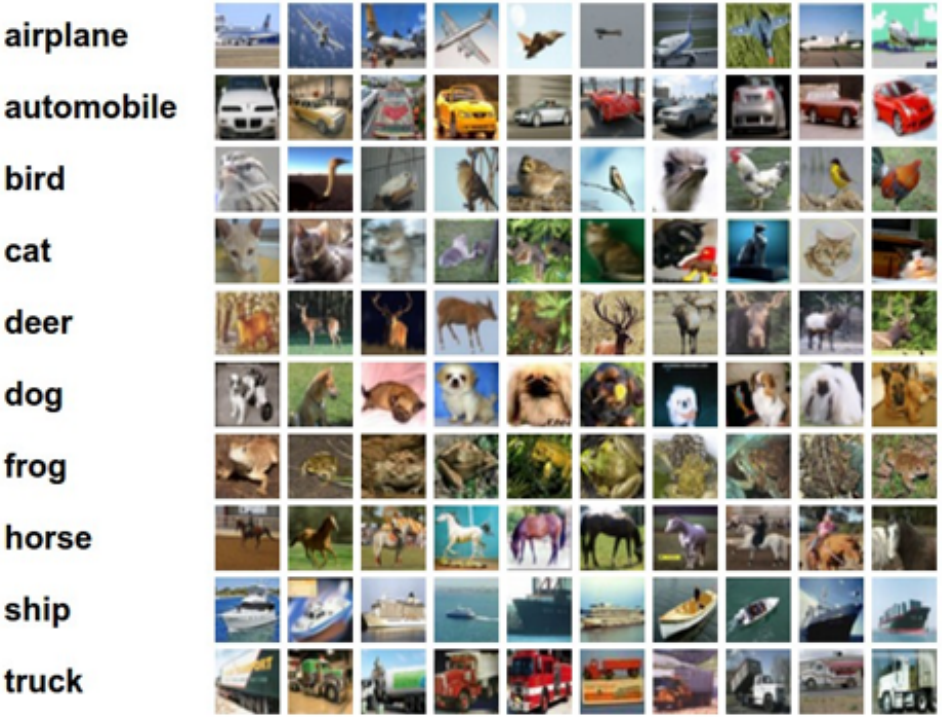
# Interpreting a Linear Classifier




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

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

# Interpreting a Linear Classifier




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

Q2: what would be a very hard set of classes for a linear classifier to distinguish?

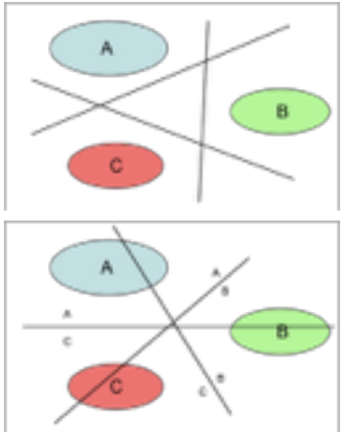
# 1 vs All Classifiers

- What if you have a new category?

- Option 1: Treat examples of that class as positives, all other classes negative

- Option 2: Train N hyperplanes, where each linear classifier separates the new category from one of the existing categories (1 vs. Each

- Aside: Using a standard classification library like Sci-Kit Learn, you can also use non-linear kernels. This may be an option if you want a quick result using pre-trained features.

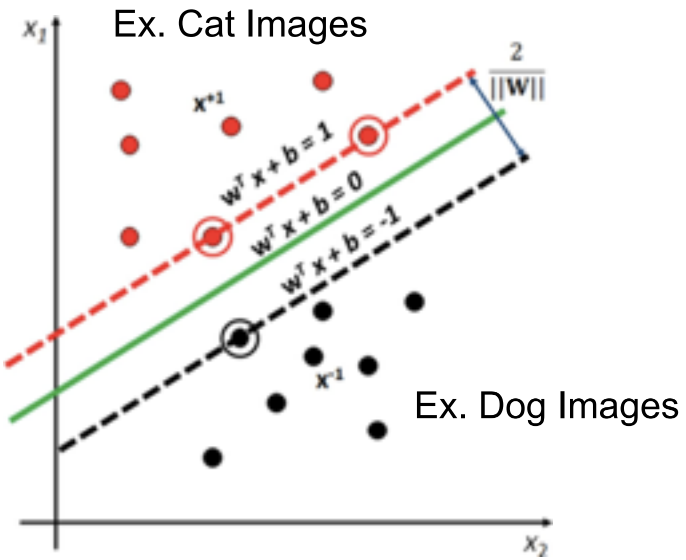



* images credit Ben Aisen 2006

# Margin and Offset (b)

- **Margin**: distance between the closest positive training item in the dataset and the hyperplane (line made by *Wx+b*), or distance between the first negative example and the hyperplane

- **Offset:** *b* is chosen so that the margin is the same on both positive and negative sides




* image credit Fujun Liu

# So far: We defined a (linear) **score function**: $f(x_i, W, b) = Wx_i + b$



Example class scores for 3 images, with a random 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 |

# How do we know if W is good?

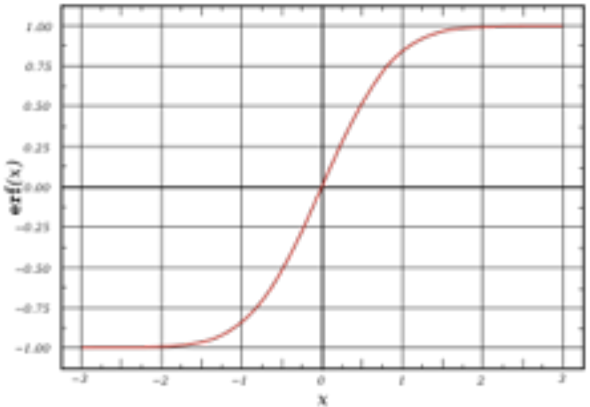
- For an example image *x*
- We want *Wx+b* to be strongly positive for a positive classification of *x*
- We want *Wx+b* to be strongly negative for a negative classification of *x*
- We want *Wx+b* to be a greater value for the correct class than for any other class
- **Next lecture:** how to use a **Loss Function** to find *(W,b)* parameters that satisfy these requirements
- Unlike NN classifiers, training for linear classifiers will take computation time
- Test time only requires one matrix multiplication, and is fast

# Off-the-Shelf 1 vs. All Classifiers

- If you have 1 vs. All linear classifiers trained separately, you can approximate comparing their outputs using a sigmoid function:

- $t = Wx+b$

$$S(t) = \frac{1}{1+e^{-t}}.$$




- The value of *S* approximates the probability of *x* belonging to a particular class. Relative rankings of class estimates will remain the same, but confusions between categories will be more obvious.

- This is not as good as using a multiclass Loss Function, but may be expedient for debugging. Training will be faster.

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

# Coming up:

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