

# Lecture 2:

# Image Classification pipeline

# Administrative

First assignment will come out tonight (or tomorrow at worst)

It is due **January 20** (i.e. in two weeks). Handed in through CourseWork

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

**Warning:** don't work on assignments from last year!

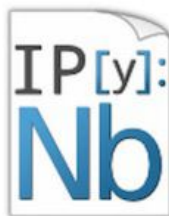
Compute: Can use your own laptops, or Terminal.com

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

## CS231n Convolutional Neural Networks for Visual Recognition

### Python Numpy Tutorial

```
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
```



## CS231N-A1

by yukez

Jan 7 2015

🔥 0

🔗 Share

### Snapshot Description

Assignment #1 of CS231N Winter 2014-15

### Tags

ipython notebook

python

NBview

Python Notebook

Terminal Notebook

The snapshot owner has not provided a README

Leave a comment

Post

### Start a Terminal

#### Terminal Name

CS231N-A1

medium

#### CPU's

2

#### RAM (GB)

3.2

#### Disk (GB)

10

#### Price (\$/hr)

0.124

Start as Temporary Terminal ? ☐

Auto-pause ? ☒

▶ Start

zyk09021FilesEditorTerminalBrowser

Terminal

Browser

root@yukez2:/home/notebooks/assignment1-master# ls  
README.md cs231n features.ipynb knn.ipynb softmax.ipynb svm.ipynb  
root@yukez2:/home/notebooks/assignment1-master#

localhost:8888/notebooks/

## Notebook

NotebooksRunningClusters

To import a notebook, drag the file onto the listing below or [click here](#).

assignment1-master

StarterNB.ipynb

New NotebookDelete

Upload FilesUpload Folder

or drag and drop files here

# Image Classification: a core task in Computer Vision



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



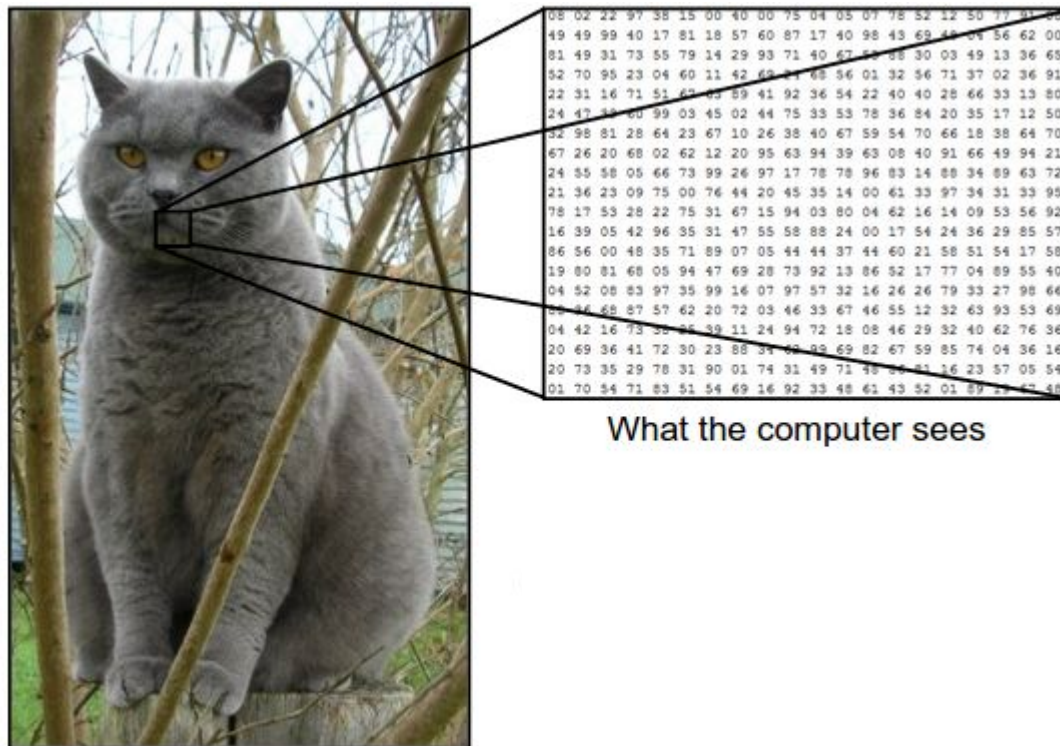
cat

# The problem: *semantic gap*

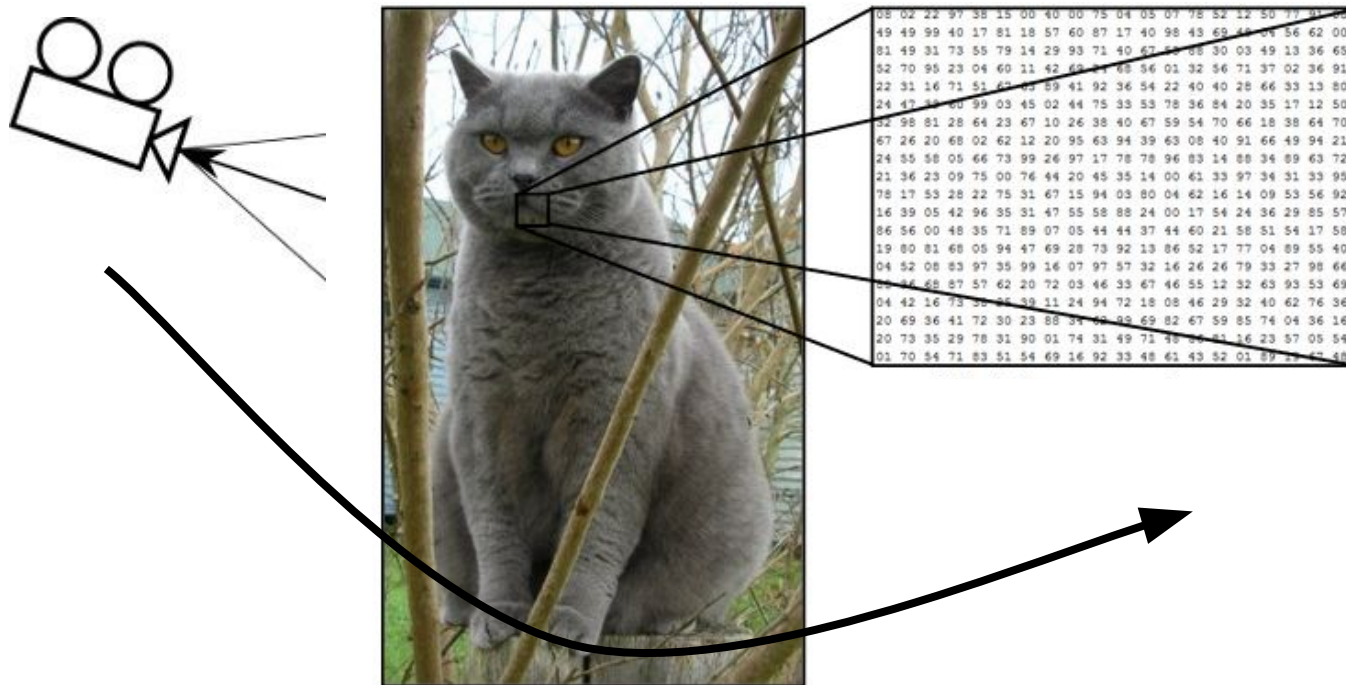
Images are represented as  
3D arrays of numbers, with  
integers between  $[0, 255]$ .

E.g.  
 $300 \times 100 \times 3$

(3 for 3 color channels RGB)



# Challenges: Viewpoint Variation





# Challenges: Illumination



# Challenges: Deformation



# Challenges: Occlusion



# Challenges: Background clutter





# Challenges: Intraclass variation



# An image classifier

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

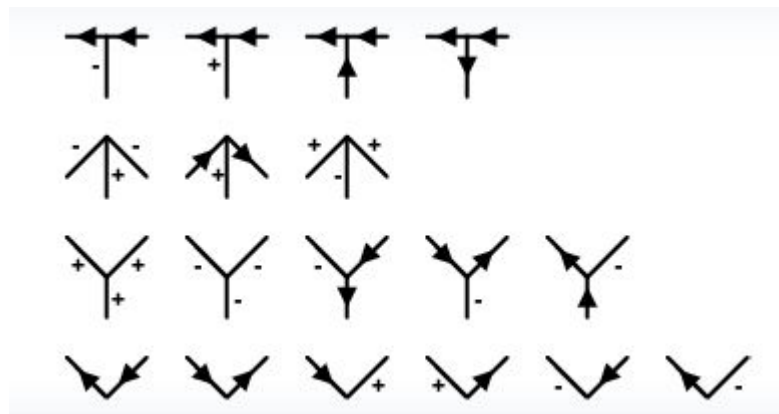
Unlike e.g. sorting a list of numbers,

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

Attempts have been made



???

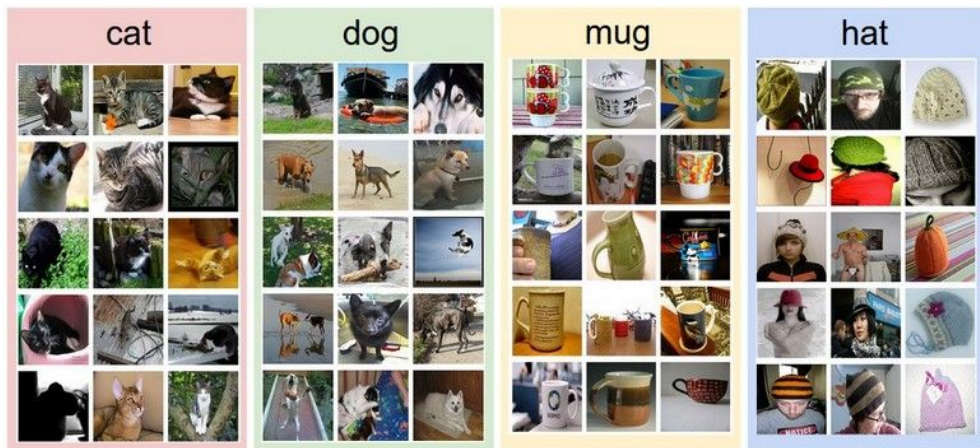


# Data-driven approach:

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





# 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

# Example dataset: **CIFAR-10**

**10** labels

**50,000** training images, each image is tiny: 32x32

**10,000** test images.

airplane



automobile



bird



cat



deer



dog



frog



horse



ship



truck



# Example dataset: **CIFAR-10**

**10** labels

**50,000** training images

**10,000** test images.

airplane



automobile



bird



cat



deer



dog



frog



horse



ship



truck



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

test image					training image					pixel-wise absolute value differences				
56	32	10	18		10	20	24	17		46	12	14	1	
90	23	128	133		8	10	89	100		82	13	39	33	
24	26	178	200	-	12	16	178	170	=	12	10	0	30	
2	0	255	220		4	32	233	112		2	32	22	108	add → 456



## Nearest Neighbor classifier

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

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

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

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

        return Ypred
```

## Nearest Neighbor classifier

```
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

## 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

```
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?**



```
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

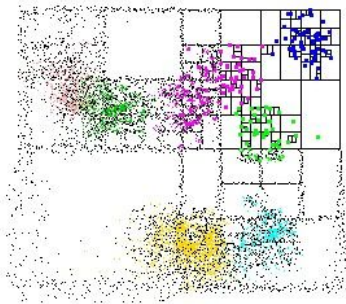
find approximate nearest neighbors quickly

## 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!
- The macOS X users can install flann through 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

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

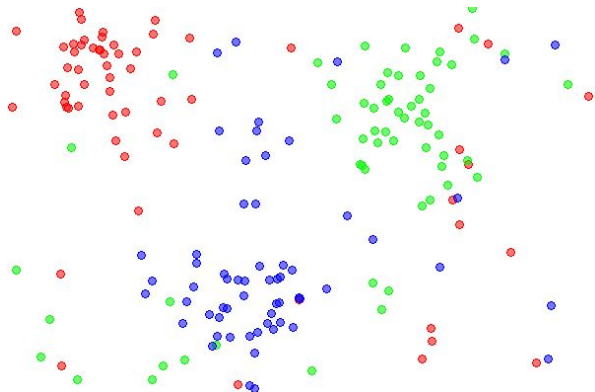
L2 (Euclidean) distance

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

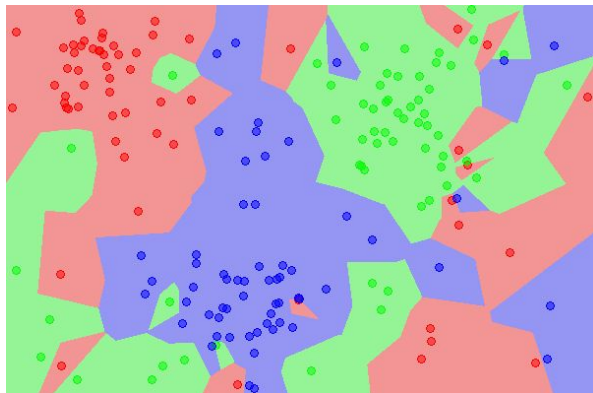
# k-Nearest Neighbor

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

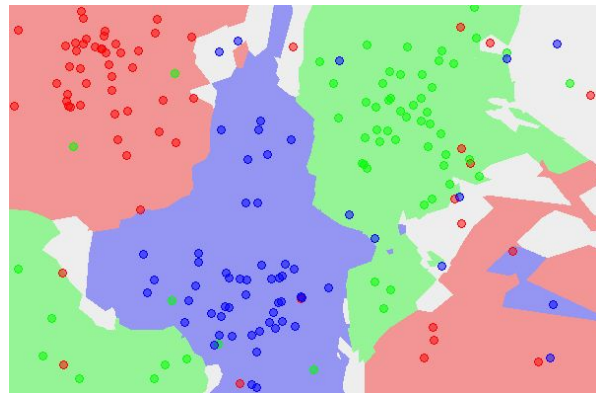
the data



NN classifier



5-NN classifier



[http://en.wikipedia.org/wiki/K-nearest\\_neighbors\\_algorithm](http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)



# Example dataset: **CIFAR-10**

**10** labels

**50,000** training images

**10,000** test images.

airplane



automobile



bird



cat



deer



dog



frog



horse



ship



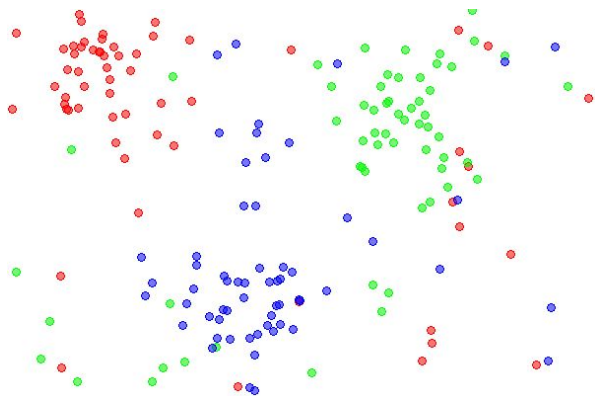
truck



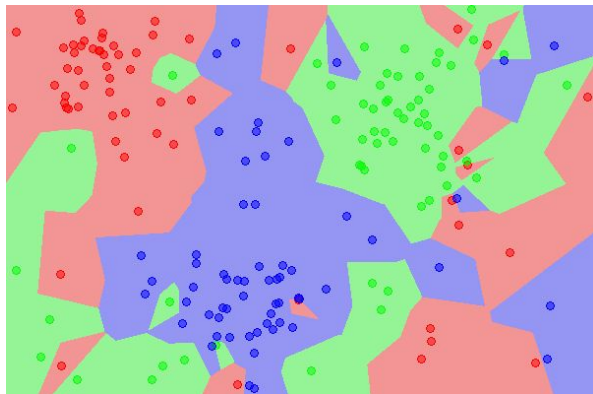
For every test image (first column),  
examples of nearest neighbors in rows



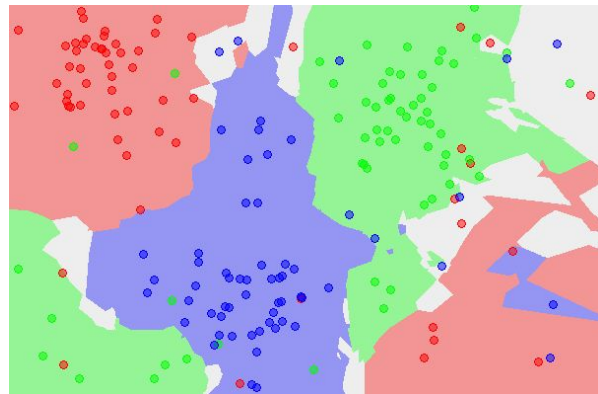
the data



NN classifier

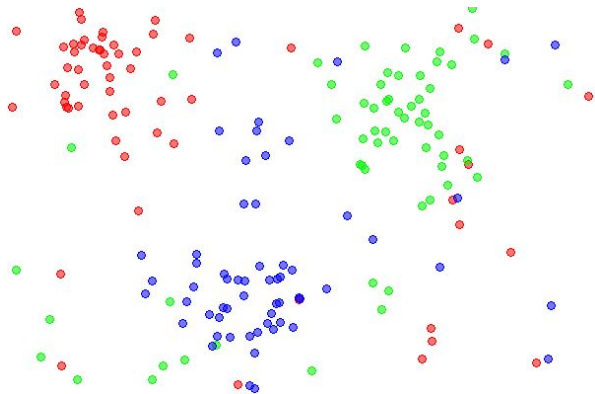


5-NN classifier

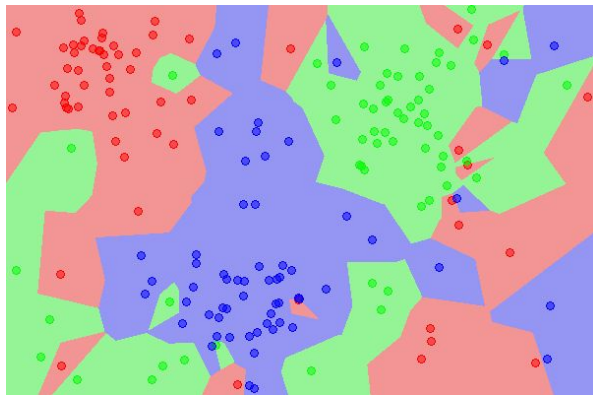


Q: what is the accuracy of the nearest neighbor classifier on the training data, when using the Euclidean distance?

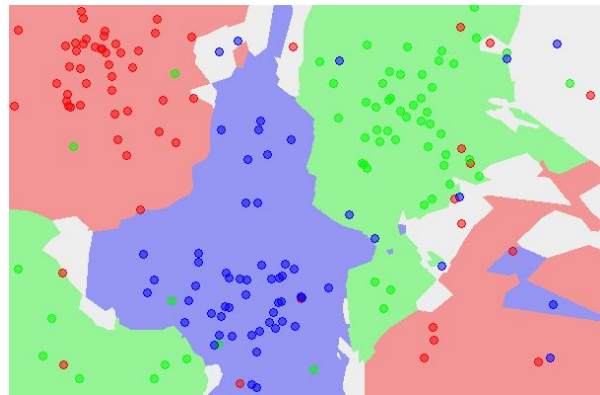
the data



NN classifier



5-NN classifier



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?

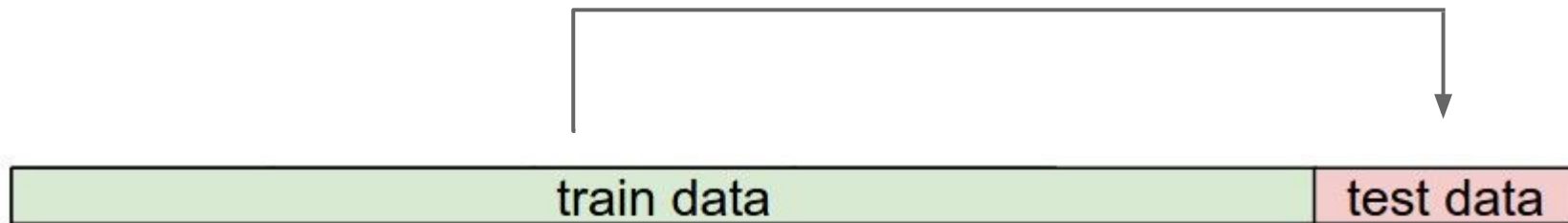
What is the best value of **k** to use?

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

Very problem-dependent.

Must try them all out and see what works best.

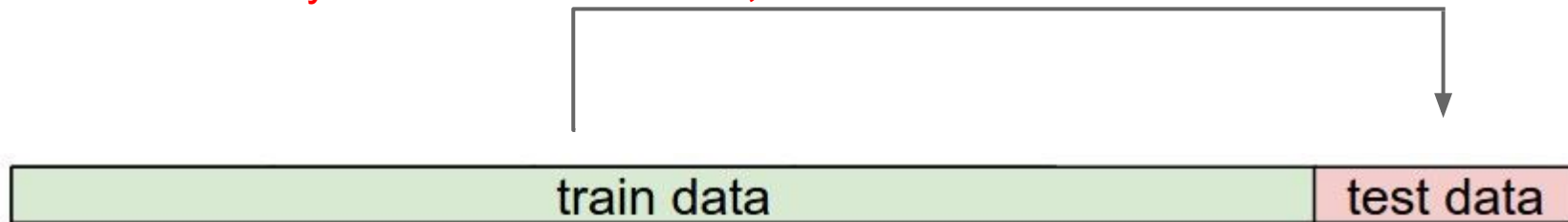
Try out what hyperparameters work best on test set.

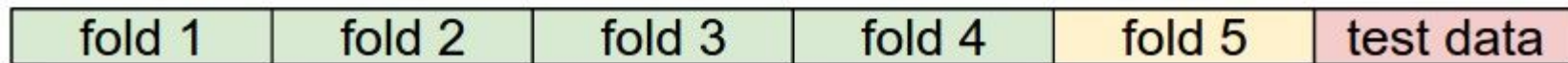
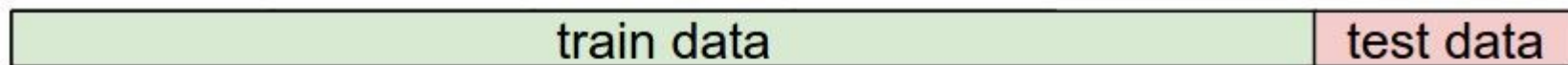


Trying out what hyperparameters work best on test set:

Very bad idea. The test set is a proxy for the generalization performance!

Use only **VERY SPARINGLY**, at the end.



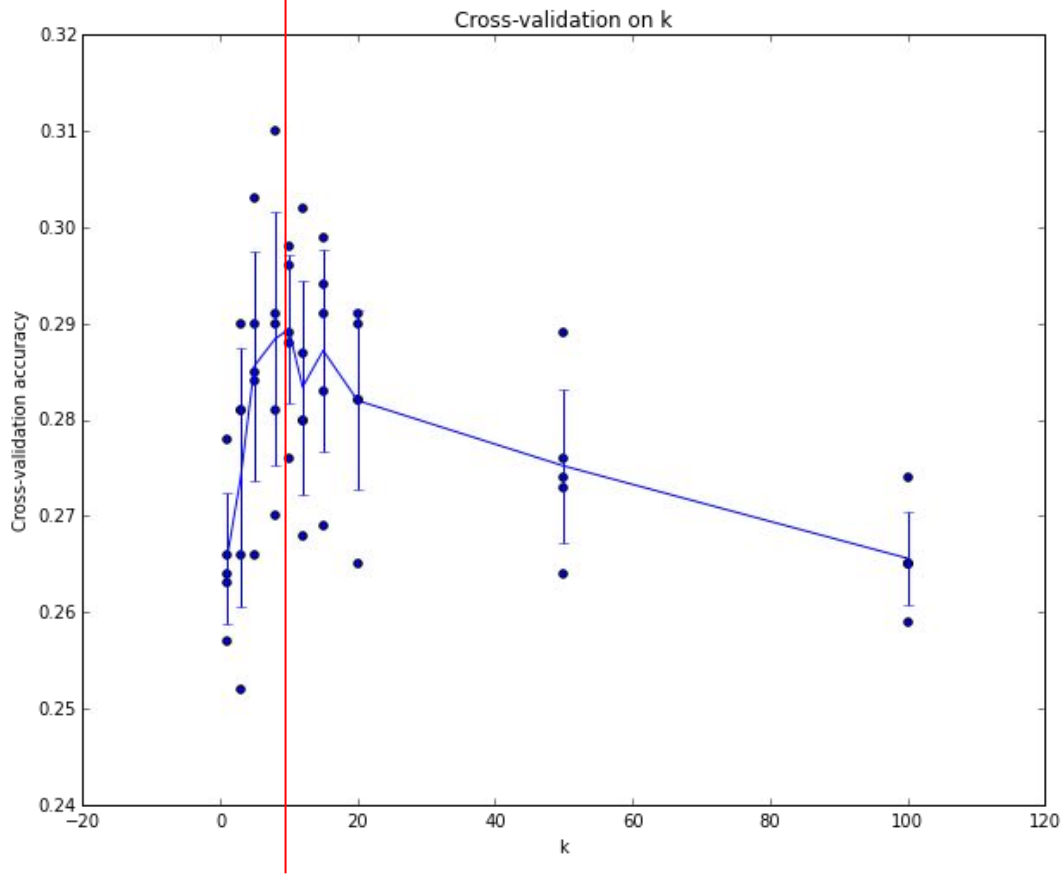


**Validation data**  
use to tune hyperparameters



## 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**.

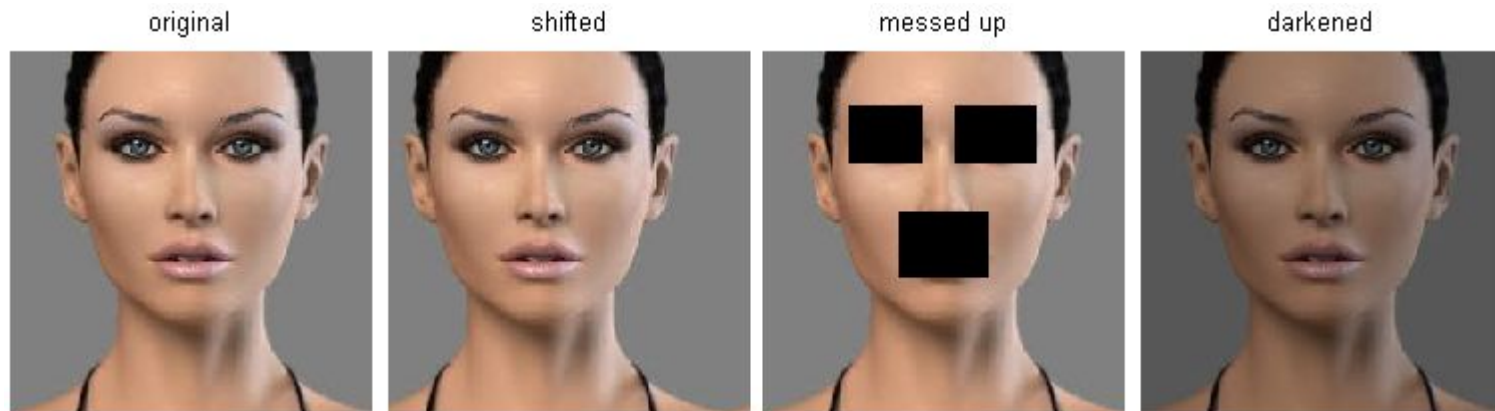
Each point: single  
outcome.

The line goes  
through the mean, bars  
indicated standard  
deviation

(Seems that  $k \approx 7$  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



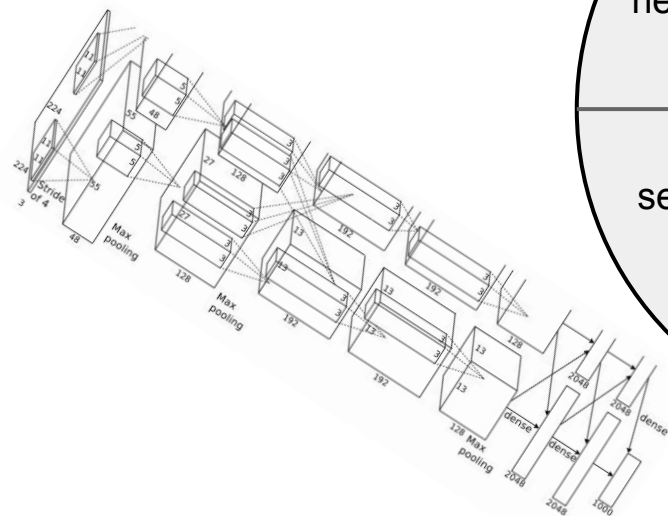
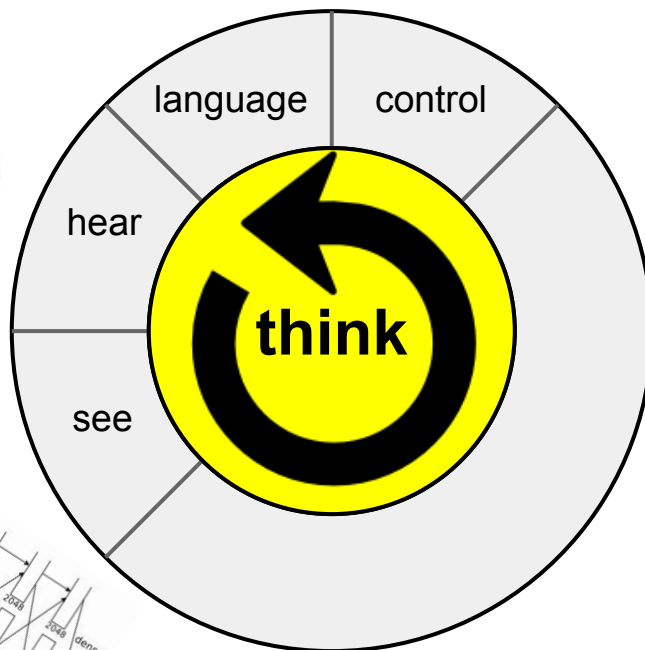
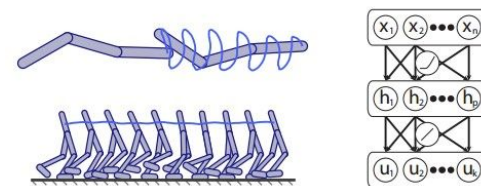
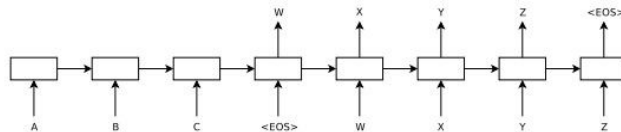
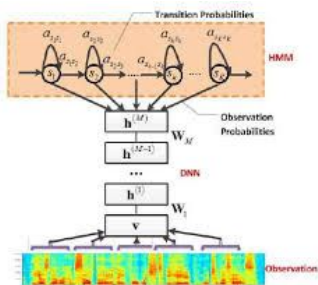
(all 3 images have same L2 distance to the one on the left)

# 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.



# Linear Classification



# Neural Networks practitioner





"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."



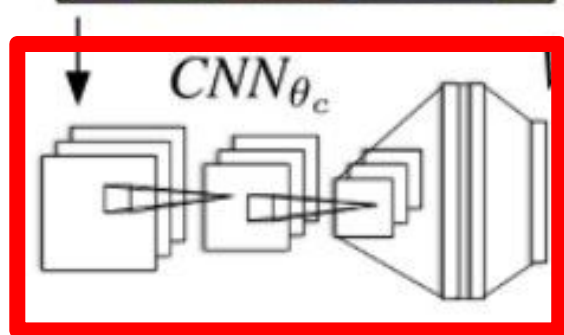
"black and white dog jumps over bar."



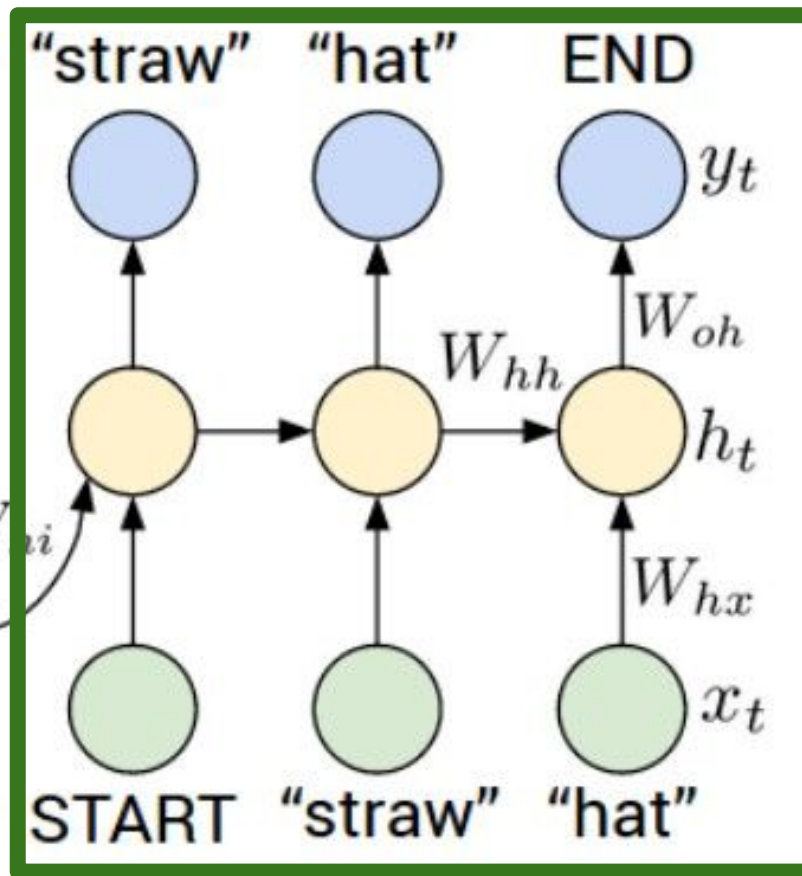
"young girl in pink shirt is swinging on swing."



"man in blue wetsuit is surfing on wave."



**CNN**



**RNN**



airplane



automobile



bird



cat



deer



dog



frog



horse



ship



truck



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



# Parametric approach



image    parameters

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



**10** numbers,  
indicating class  
scores

**[32x32x3]**

array of numbers 0...1  
(3072 numbers total)

# Parametric approach: **Linear classifier**

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



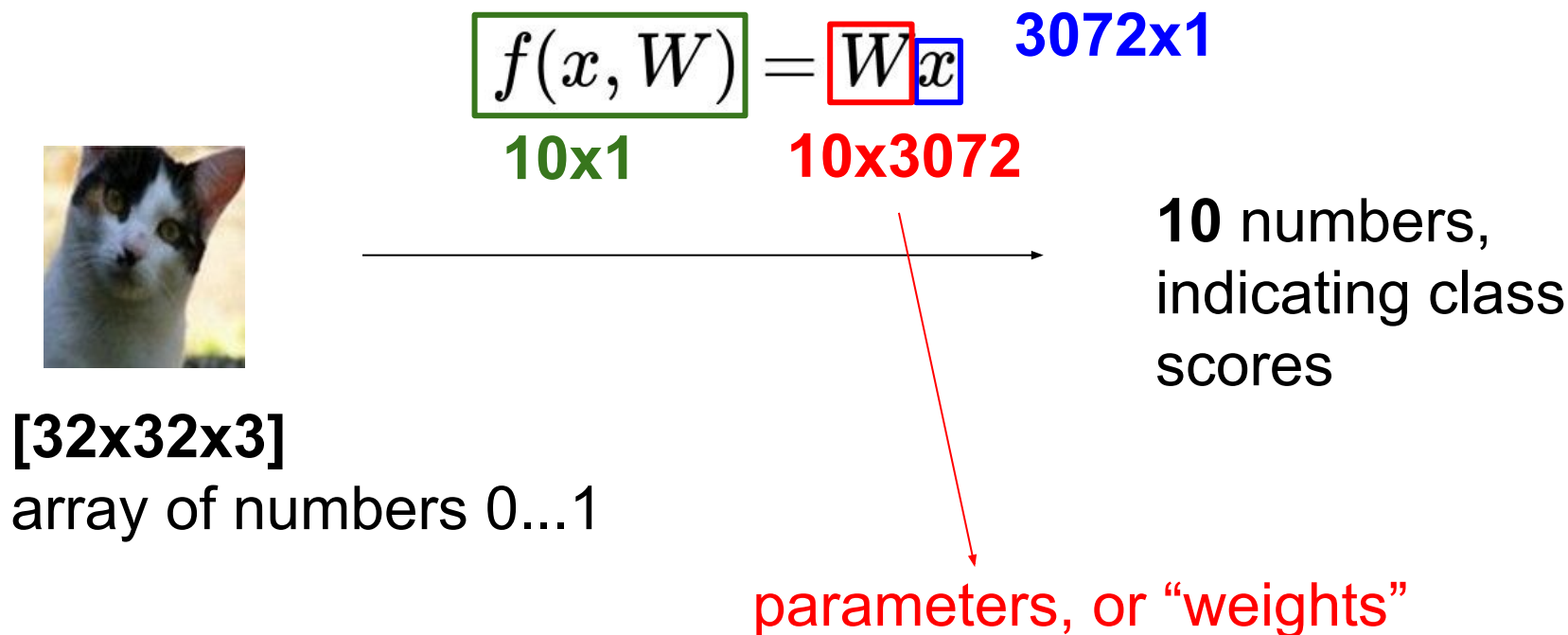
**[32x32x3]**

array of numbers 0...1

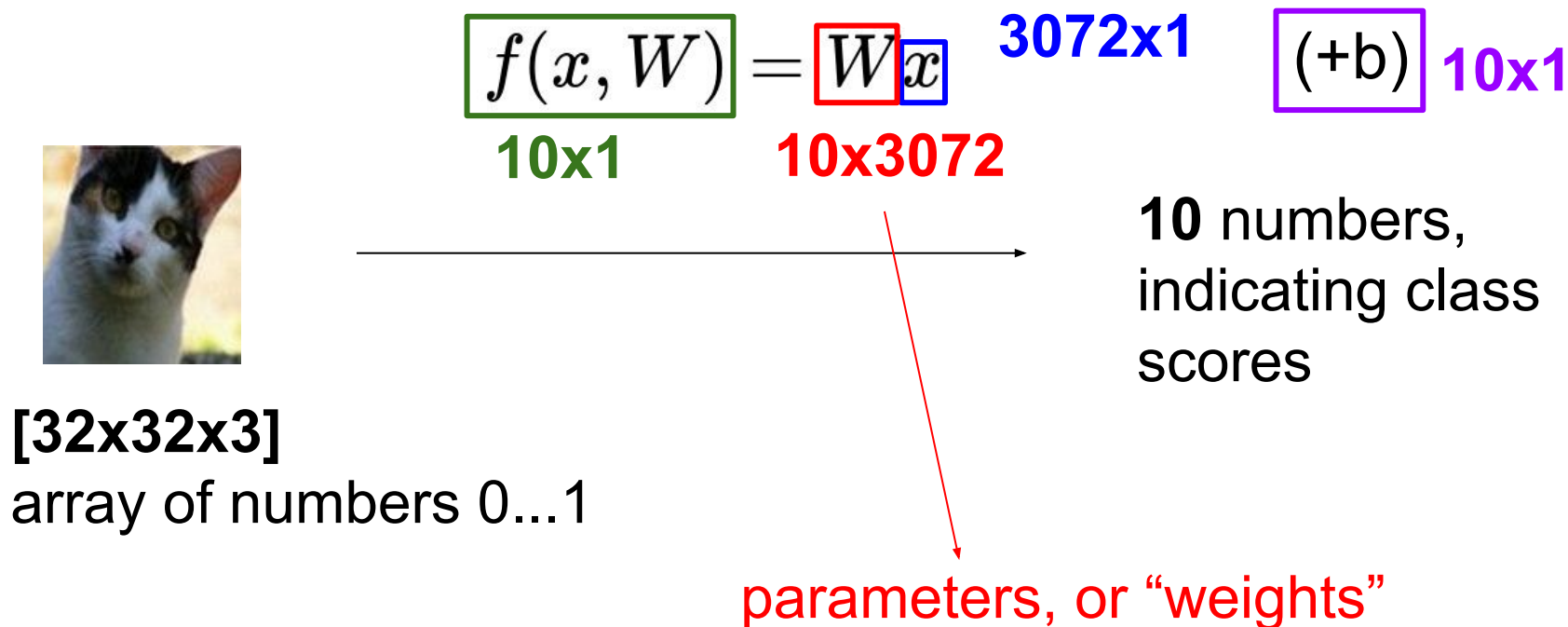


**10** numbers,  
indicating class  
scores

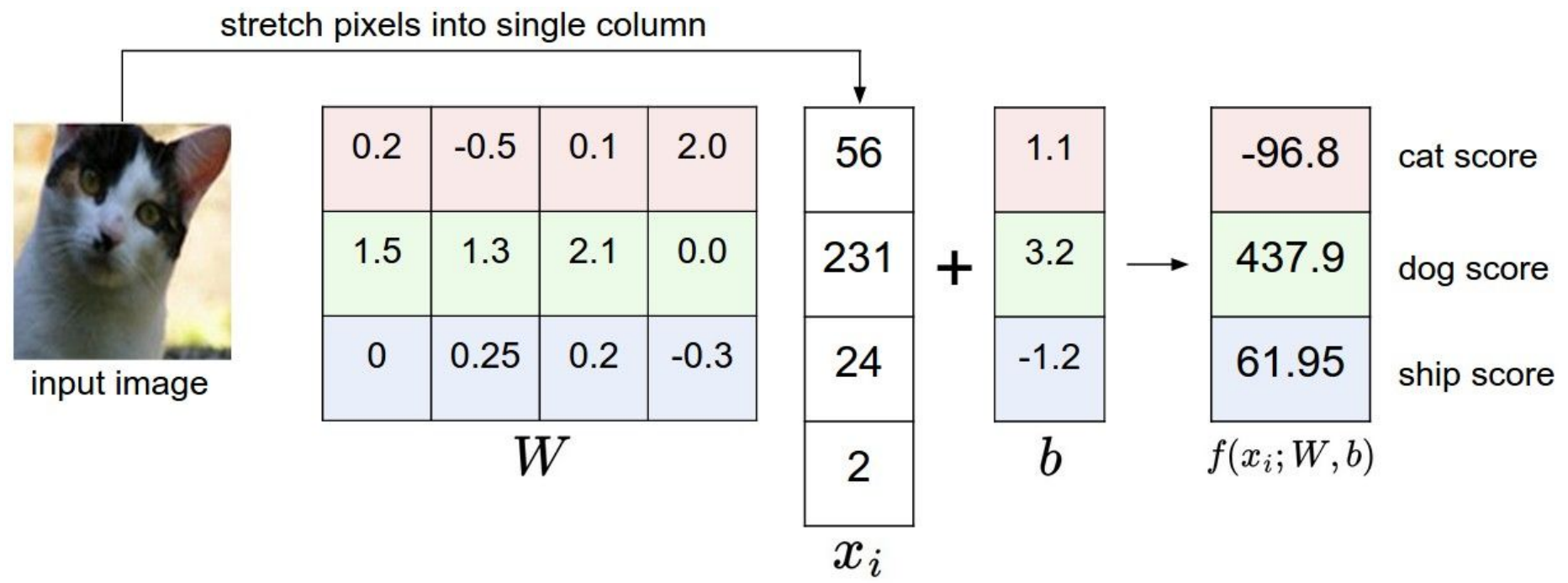
# Parametric approach: Linear classifier



# Parametric approach: Linear classifier



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



# Interpreting a Linear Classifier

airplane



automobile



bird



cat



deer



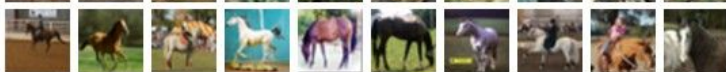
dog



frog



horse



ship



truck

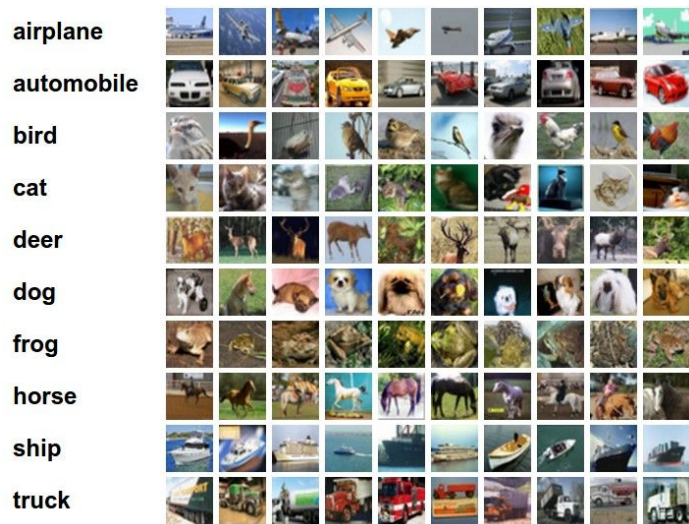


$$f(x_i, W, b) = Wx_i + b$$

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



# Interpreting a Linear Classifier

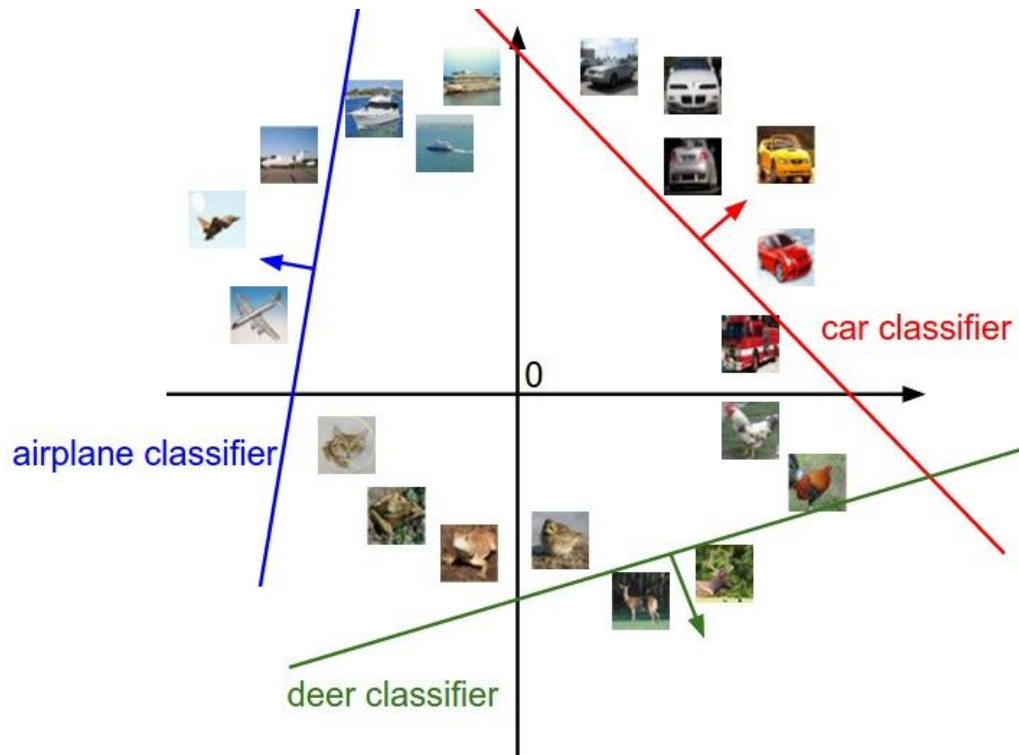


$$f(x_i, W, b) = Wx_i + b$$

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

airplane



automobile



bird



cat



deer



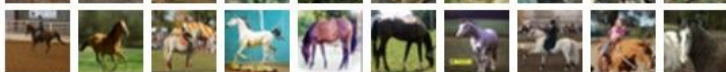
dog



frog



horse



ship



truck



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

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

**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	<b>6.04</b>	4.64
bird	0.09	5.31	2.65
cat	<b>2.9</b>	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	<b>-4.34</b>
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

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

# 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  $f$ )