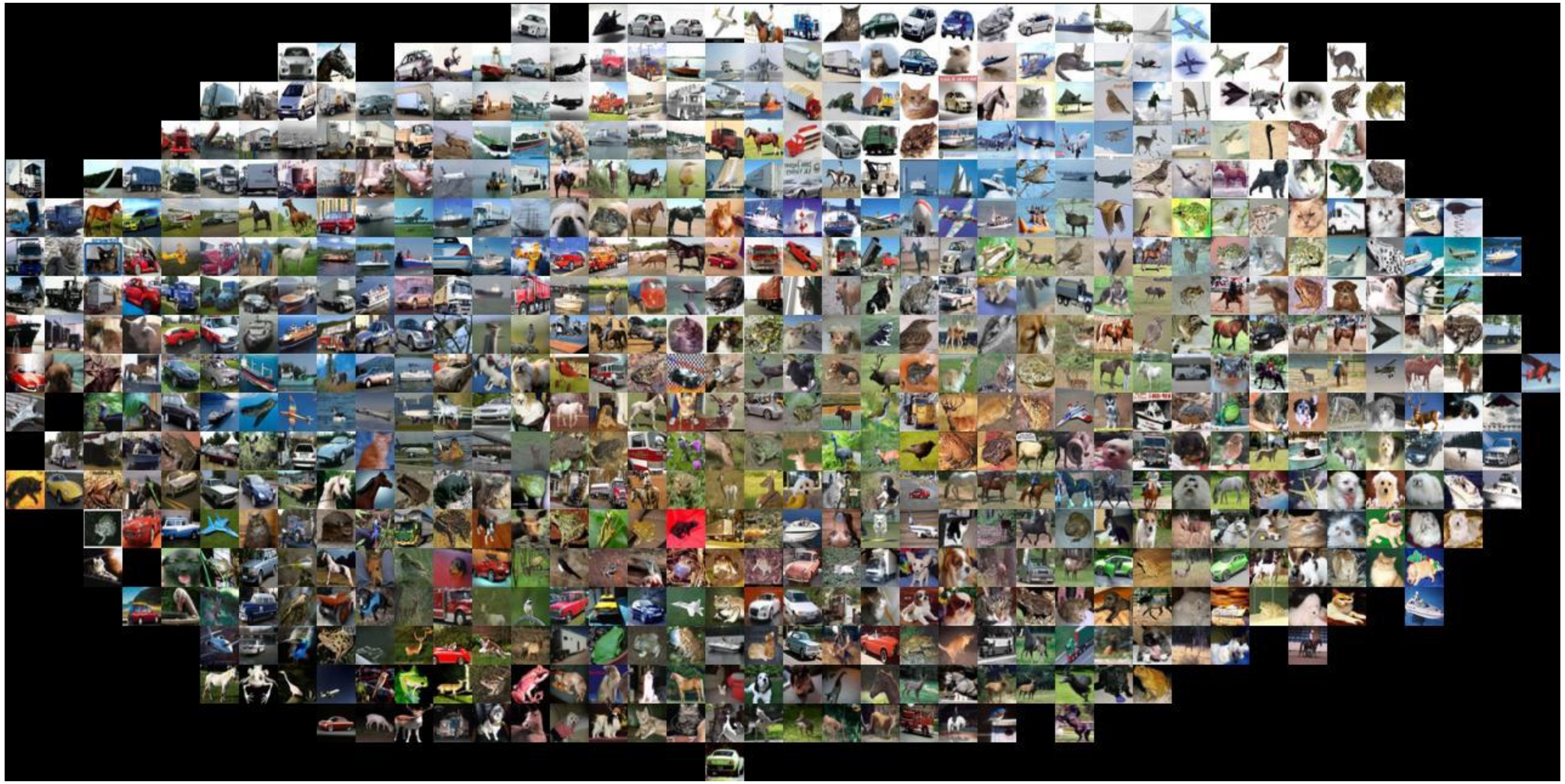
CS5670: Computer Vision

Image Classification



Some Slides from Fei-Fei Li, Justin Johnson, Serena Yeung
http://vision.stanford.edu/teaching/cs231n/

Announcements

- Project 4 due tomorrow, 4/20, by 7pm (both code and output)
- Final exam – default plan is to release on May 12, 2021, with due date of May 17, 2021
  - Please see EdStem to register for CIVS for a poll collecting input on other options
  - https://civs1.civs.us/cgi-bin/opt_in.pl
  - Please vote by tomorrow, Tuesday April 20, by 10am

- April 23-26 (Friday – Monday) are Wellness Days
- Project 5 to be released Monday May 3 (in two weeks), due Tuesday May 11 at 7pm

References

- Stanford CS231N
  - http://cs231n.stanford.edu/
- Many slides courtesy of Abe Davis

Image Classifiers in a Nutshell

- Input: an image
- Output: the class label for that image

```
def classifier(image):
     //Do some stuff
     return class_label;
```

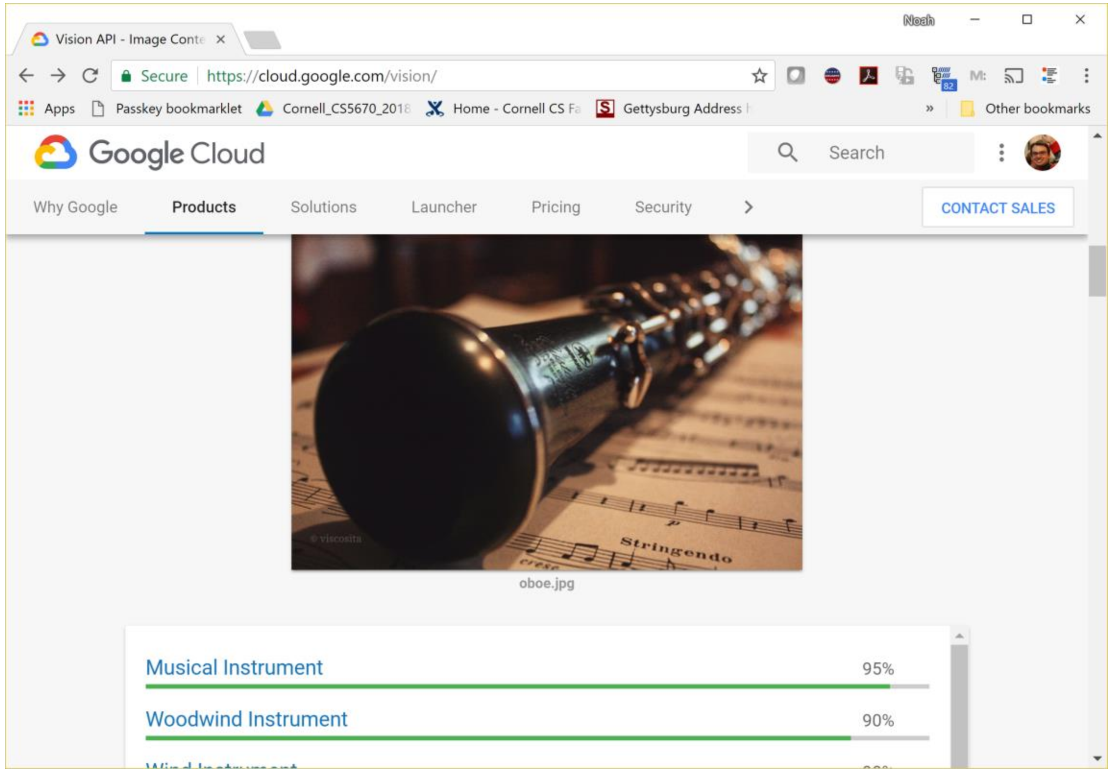
- Label is generally one or more of the discrete labels used in training
  - e.g. {cat, dog, cow, toaster, apple, tomato, truck, … }

$f($  $) =$ “Cat”

$f($  $) =$ “Dog”

$f($  $) =$ “Toaster”

Image classification demo



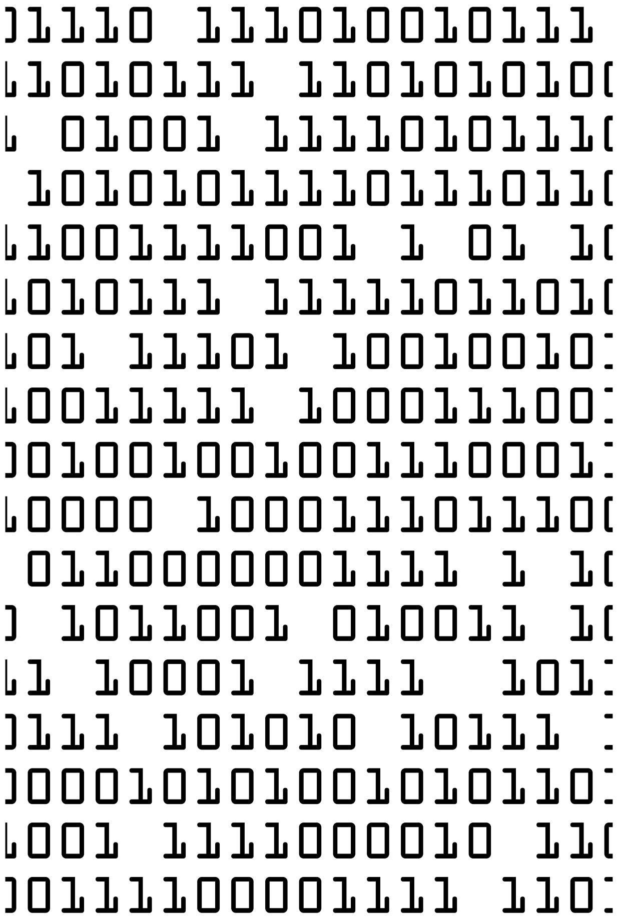
Musical Instrument	95%
Woodwind Instrument	90%

https://cloud.google.com/vision/docs/drag-and-drop

See also:
https://aws.amazon.com/rekognition/
https://www.clarifai.com/
https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/
…

The Semantic Gap

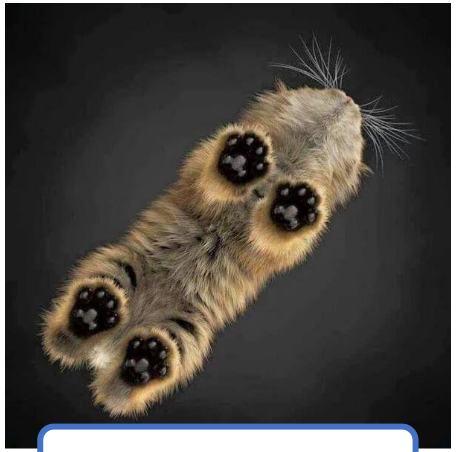
What we see

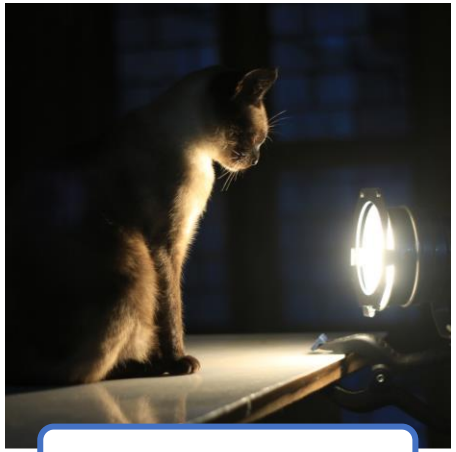


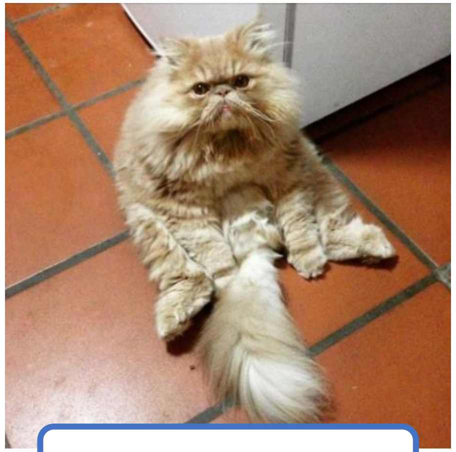
What the computer sees

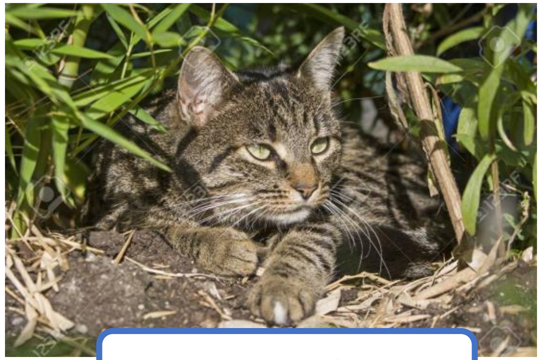
Variation Makes Recognition Hard

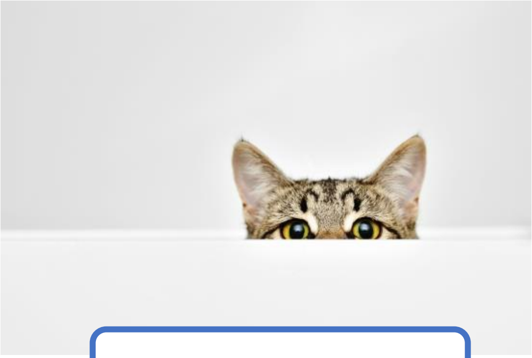
- The same class of object can appear *very* differently in different images


Viewpoint Variation


Lighting Variation


Deformation


Background Clutter


Occlusion

The Problem is Under-constrained

- Distinct realities can produce the same image…
- We generally can’t compute the “right” answer, but we can compute the most likely one…
- We need some kind of prior to condition on. We can learn this prior from data:

$$f(x) = \underset{\ell_x}{argmax}\ P(\ell_x | data)$$

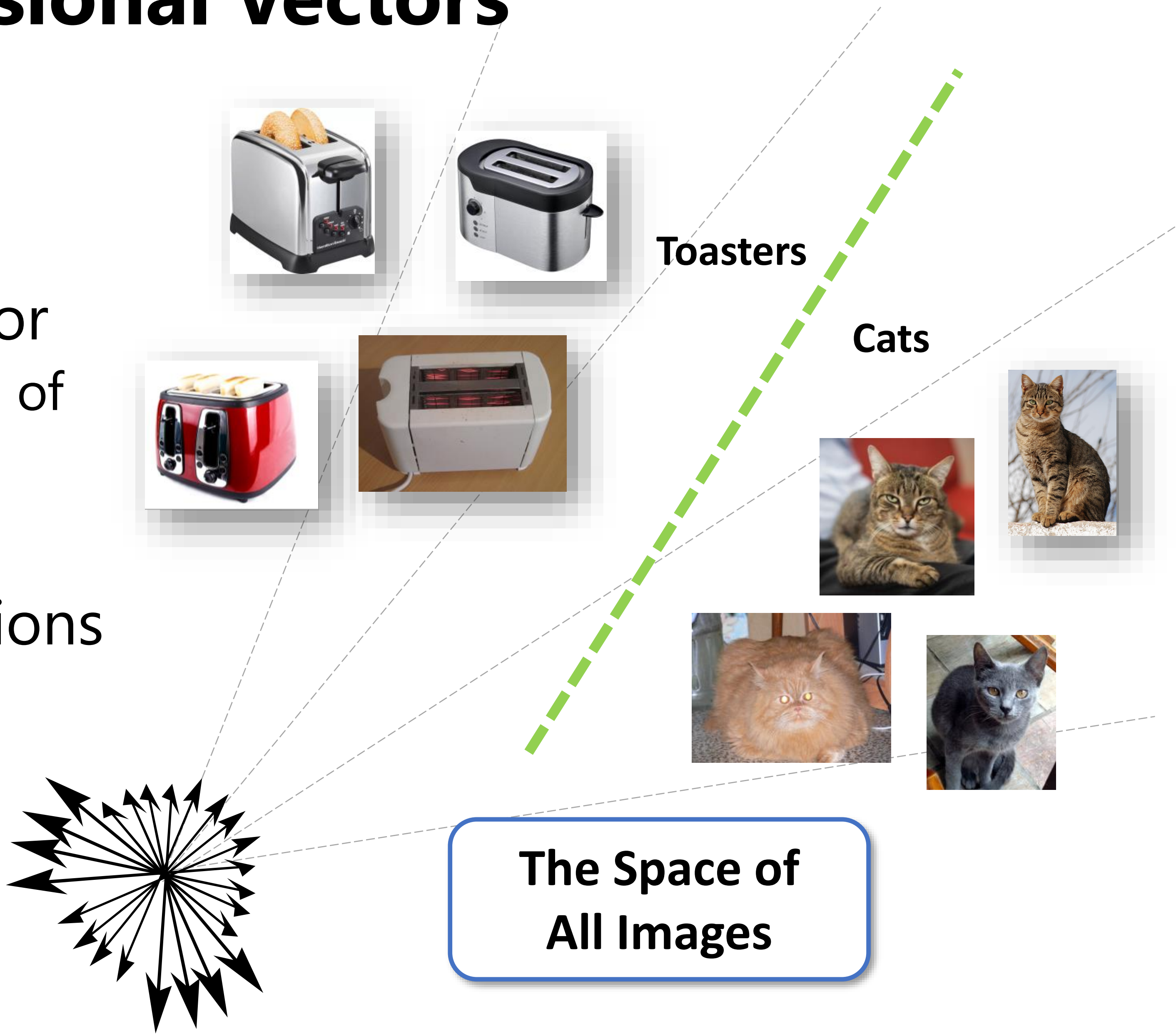
Images As High-Dimensional Vectors

- An image is just a bunch of numbers
- Let's stack them up into a vector
    - Our training data is just a bunch of high-dimensional points now

The Space of All Images

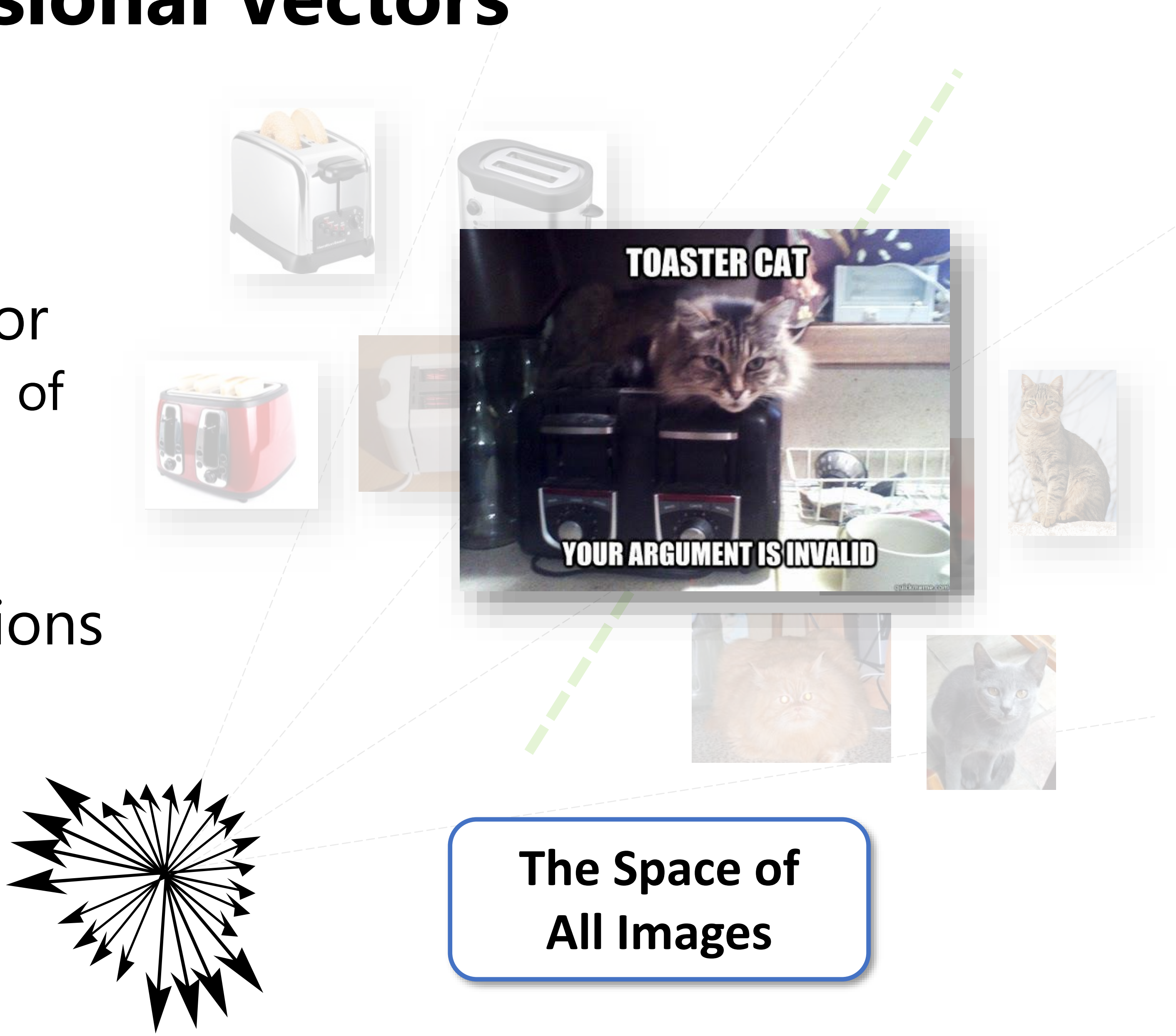
Images As High-Dimensional Vectors

- An image is just a bunch of numbers
- Let’s stack them up into a vector
  - Our training data is just a bunch of high-dimensional points now
- Divide space into different regions for different classes



Images As High-Dimensional Vectors

- An image is just a bunch of numbers
- Let’s stack them up into a vector
  - Our training data is just a bunch of high-dimensional points now
- Divide space into different regions for different classes



The Space of All Images

Images As High-Dimensional Vectors

- An image is just a bunch of numbers
- Let’s stack them up into a vector
  - Our training data is just a bunch of high-dimensional points now

- Divide space into different regions for different classes

or

- Define a distribution over space for each class

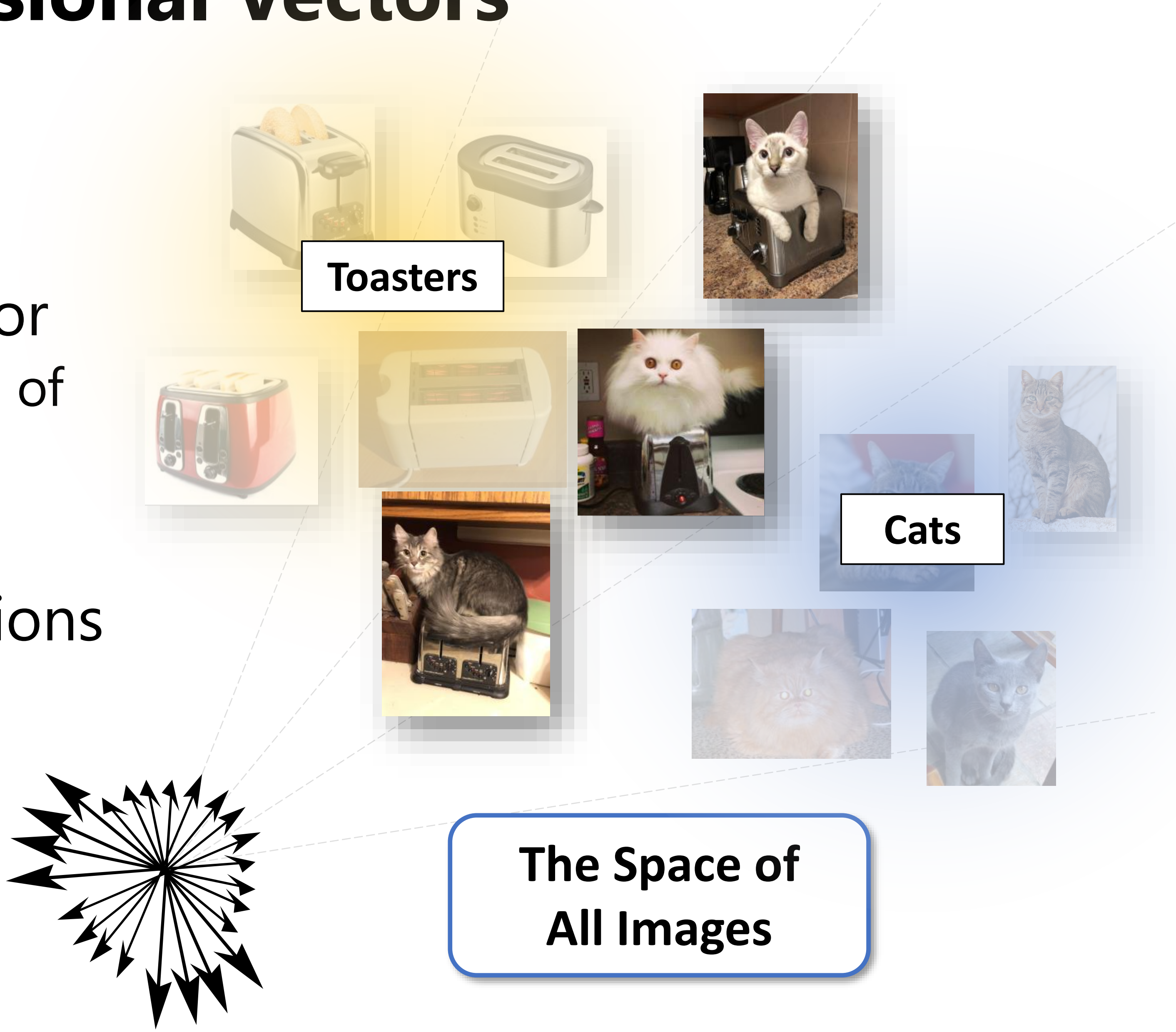
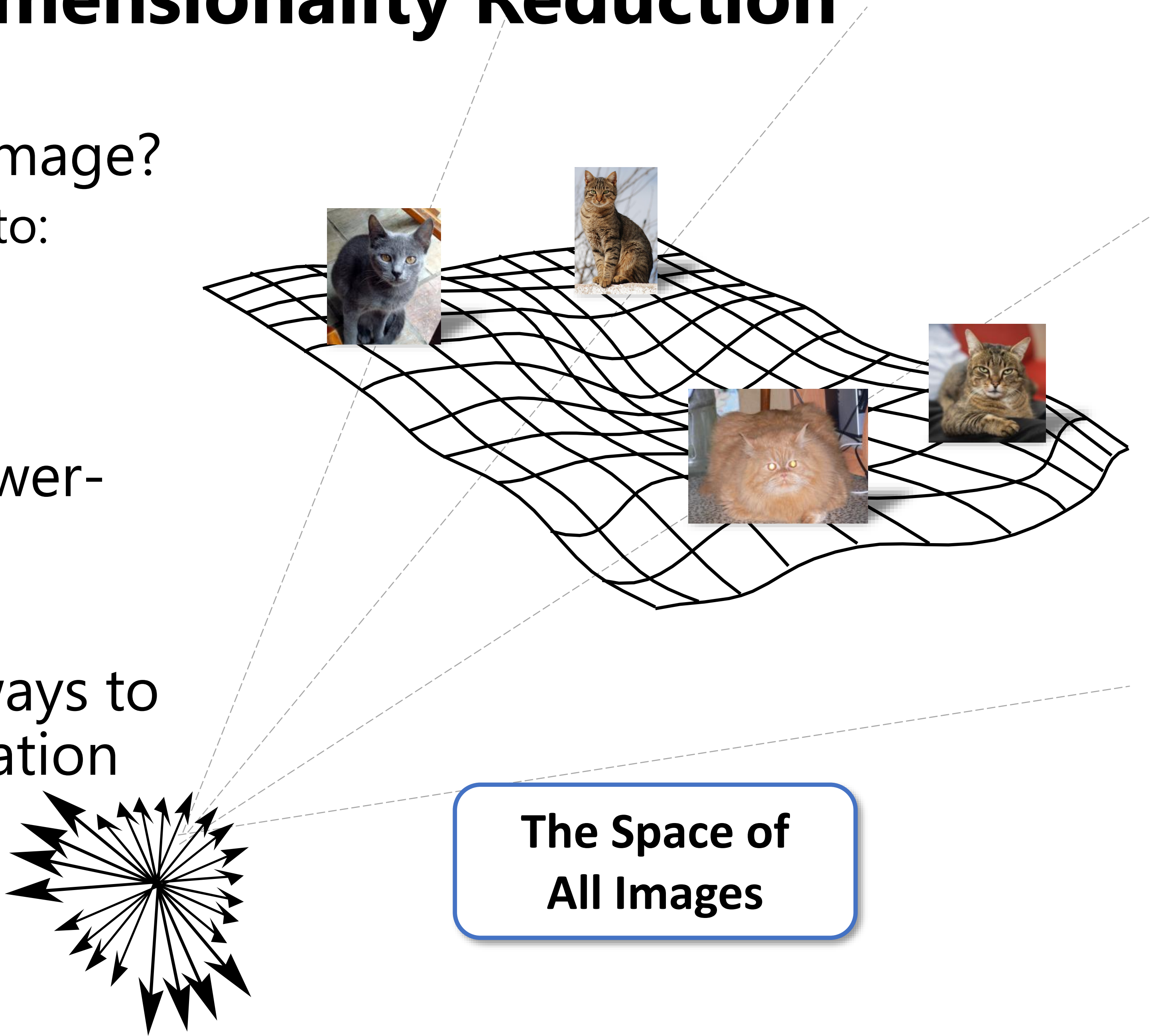


Image Features and Dimensionality Reduction

- How high-dimensional is an image?
  - Let's consider an iPhone X photo:
    - 4032 x 3024 pixels
    - Every pixel has 3 colors
    - 36,578,304 pixels (36.5 Mega pixels)
- In practice, images sit on a lower-dimensional manifold
- Think of image features and dimensionality reduction as ways to represent images by their location on such manifolds



The Space of All Images

Training & Testing a Classifier

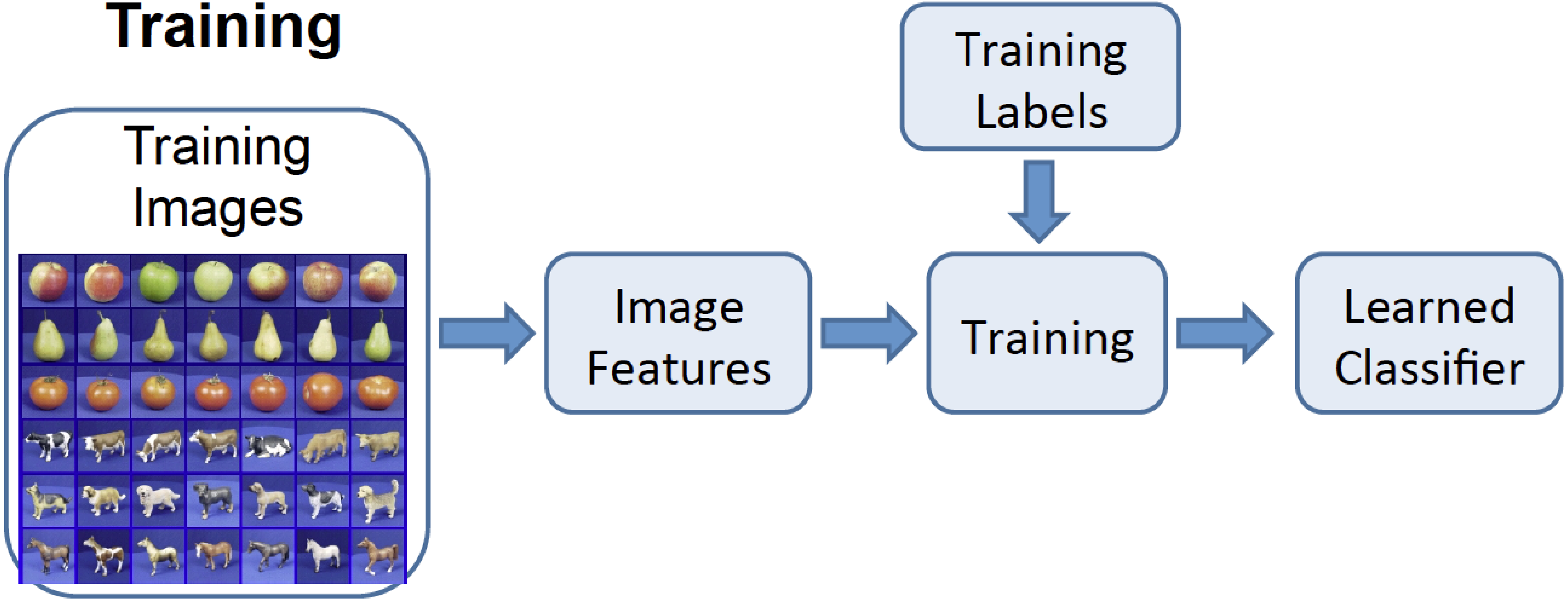
- Collect a database of images with labels
- Use ML to train an image classifier
- Evaluate the classifier on test images

Example training set

Slide from Andrej Karpathy and Fei-Fei Li   http://vision.stanford.edu/teaching/cs231n/

Training & Testing a Classifier

Training



Training Images

Image Features

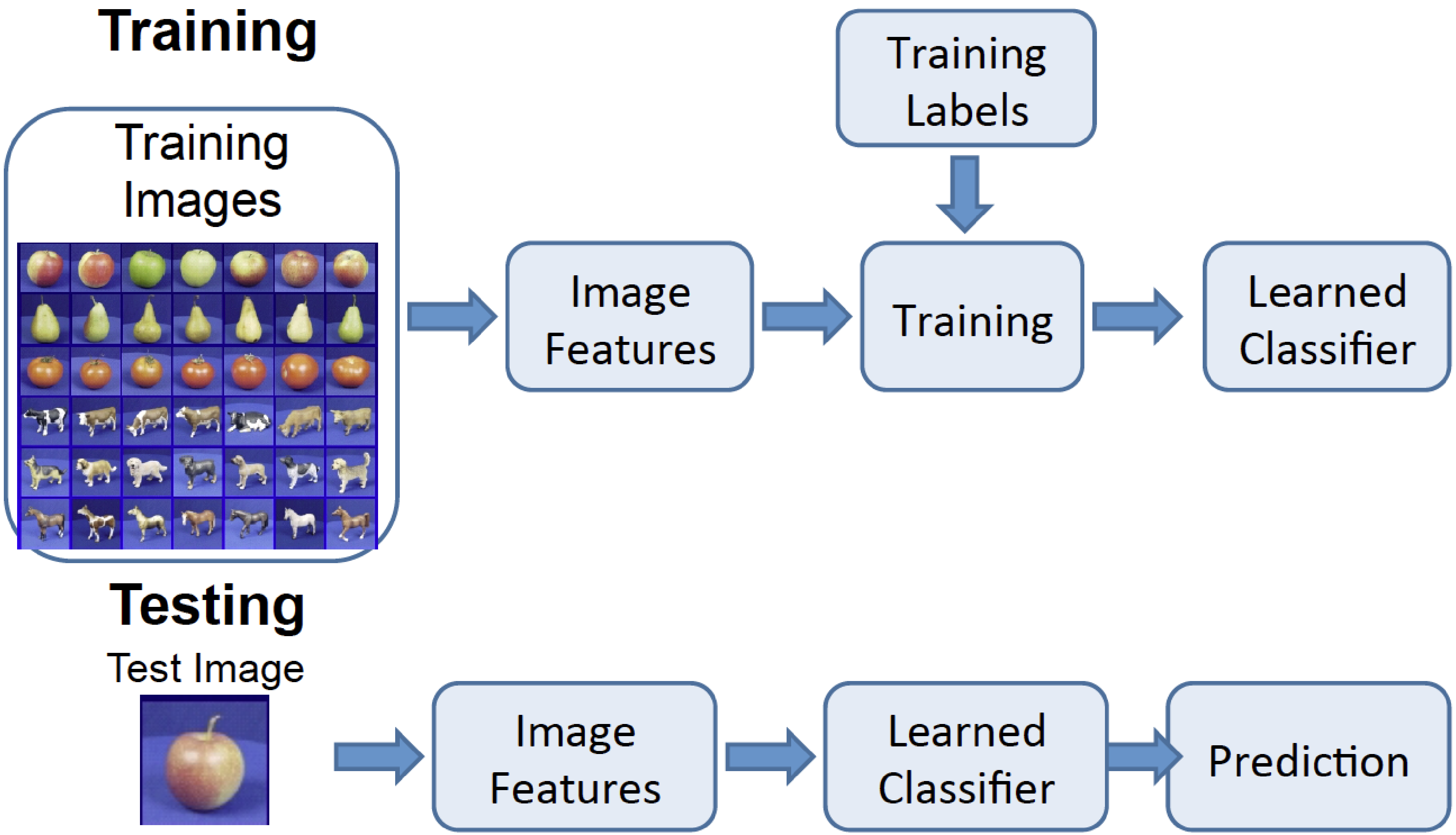
Training Labels

Training

Learned Classifier

Dataset: ETH-80, by B. Leibe Slide credit: D. Hoiem, L. Lazebnik

Training & Testing a Classifier



Training

Training Images

Image Features

Training Labels

Training

Learned Classifier

Testing

Test Image

Image Features

Learned Classifier

Prediction

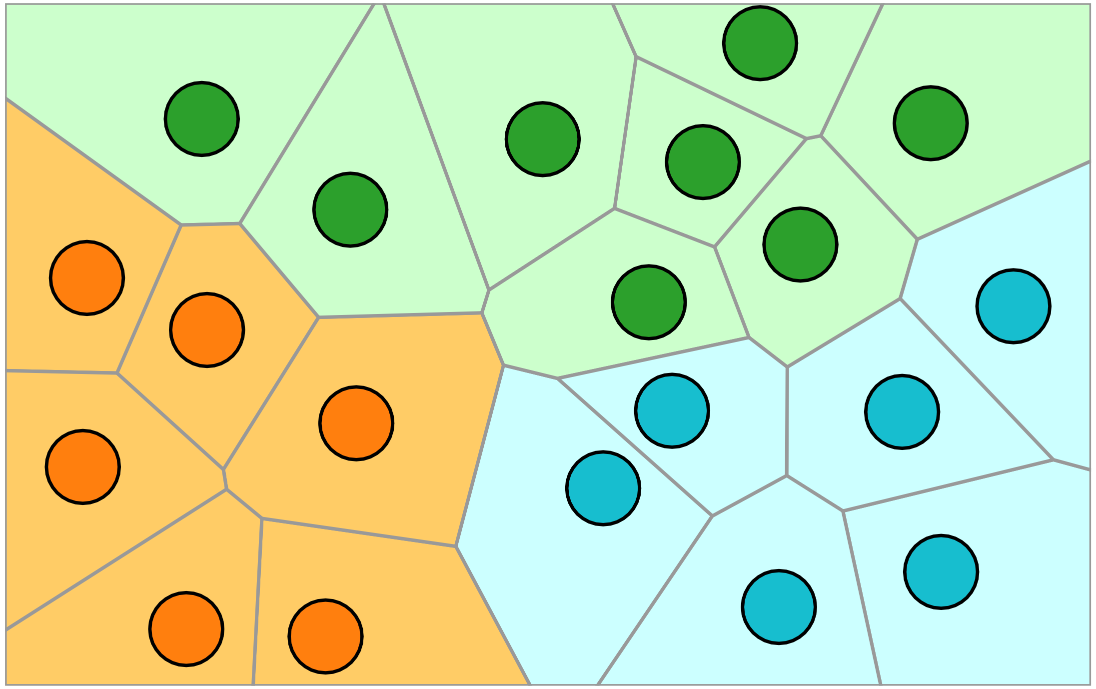
Dataset: ETH-80, by B. Leibe   Slide credit: D. Hoiem, L. Lazebnik

Classifiers

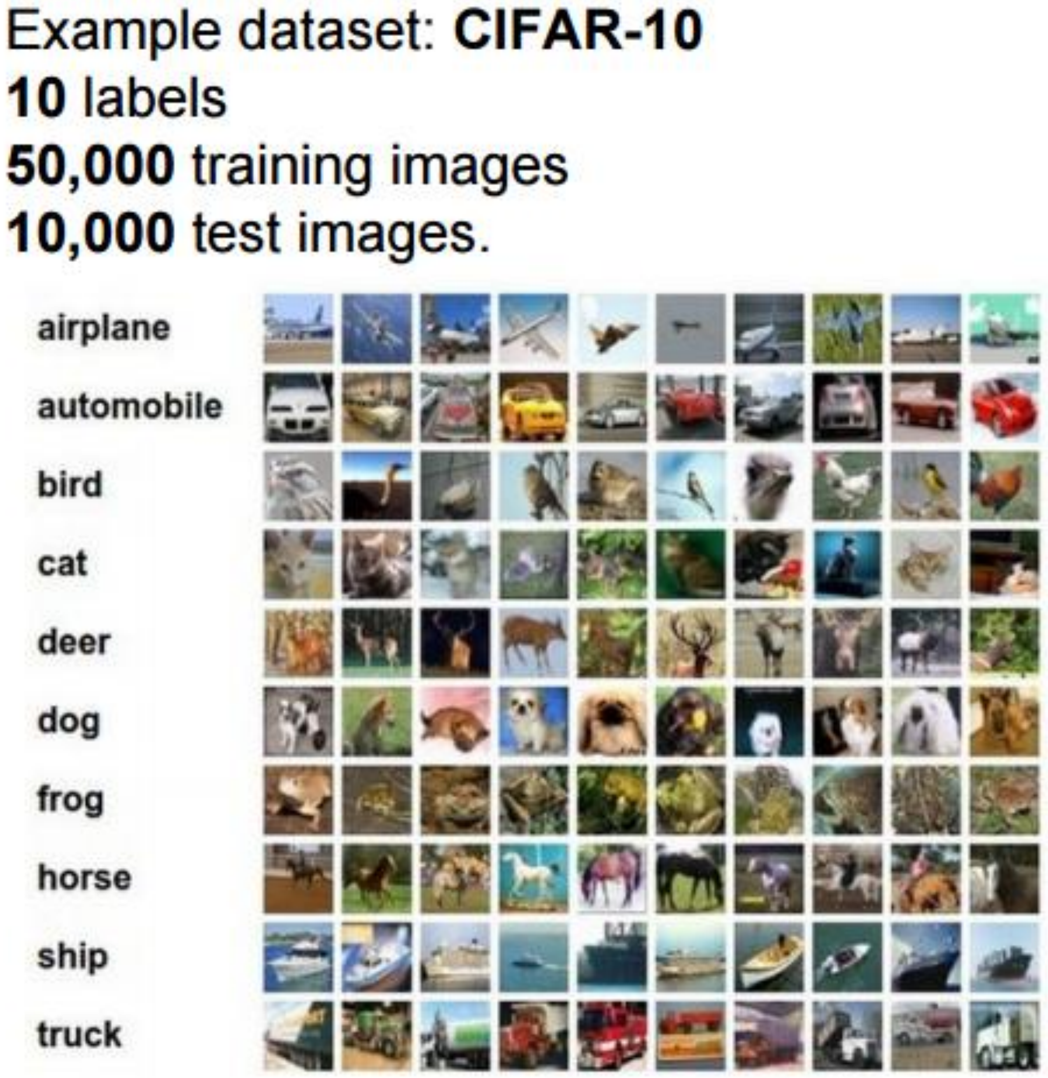
- Nearest Neighbor
- kNN (“k-Nearest Neighbors”)
- Linear Classifier
- Neural Network
- Deep Neural Network
- …

First: Nearest Neighbor (NN) Classifier

- Train
  - Remember all training images and their labels
- Predict
  - Find the closest (most similar) training image
  - Predict its label as the true label



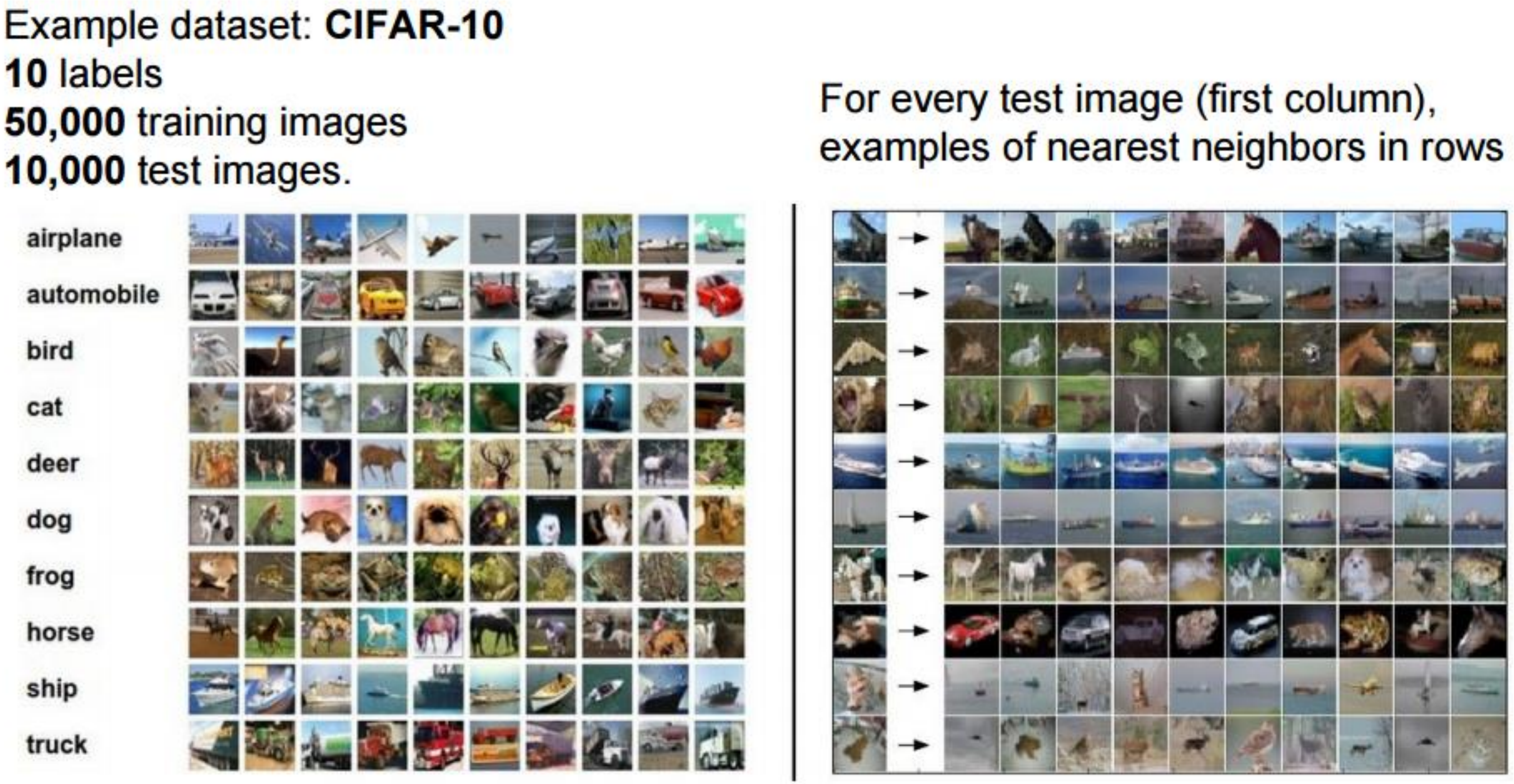
CIFAR-10 and NN results



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

Slides from Andrej Karpathy and Fei-Fei Li
http://vision.stanford.edu/teaching/cs231n/

CIFAR-10 and NN results



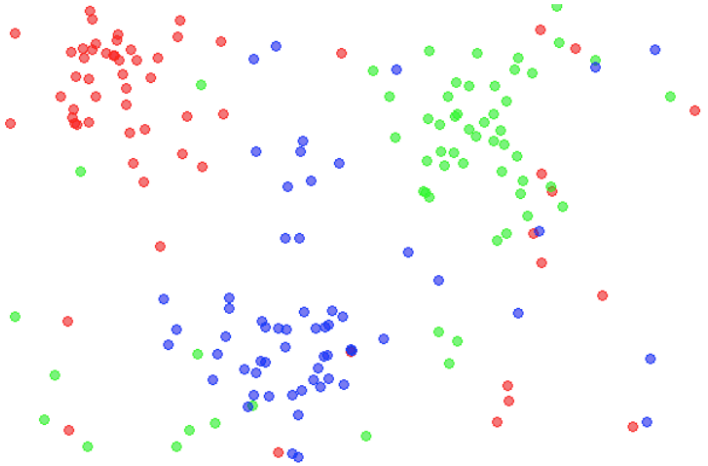
Example dataset: **CIFAR-10**
10 labels
50,000 training images
10,000 test images.

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

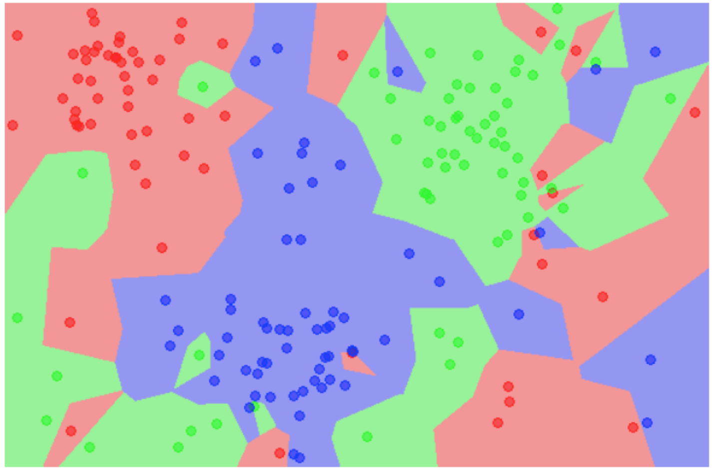
Slides from Andrej Karpathy and Fei-Fei Li
http://vision.stanford.edu/teaching/cs231n/

k-nearest neighbor

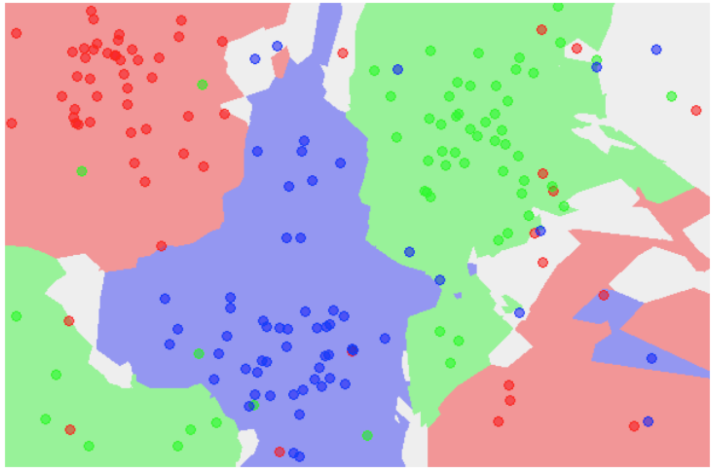
- Find the k closest points from training data
- Take **majority vote** from K closest points



the data

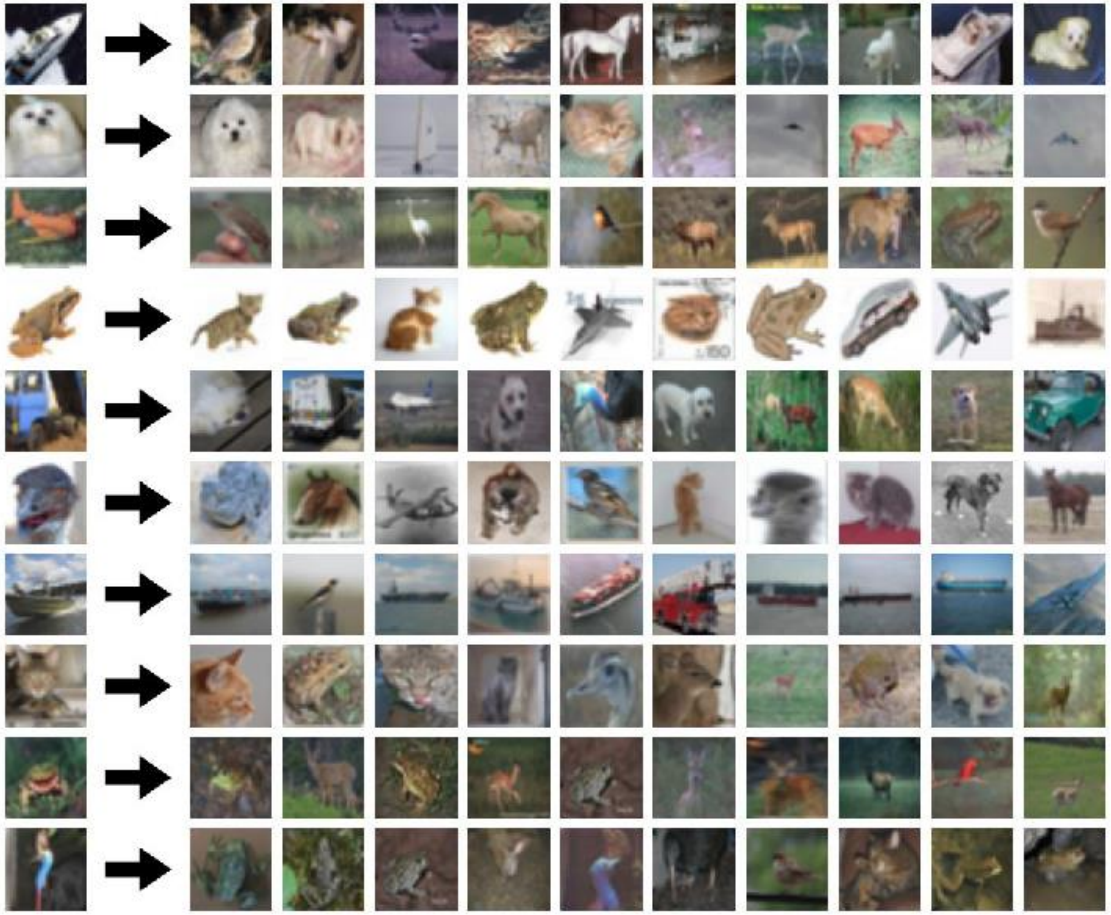


NN classifier



5-NN classifier

What does this look like?



What does this look like?

How to Define Distance Between Images

L1 distance:

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

Where I_1 denotes image 1, and p denotes each pixel

test image

training image

pixel-wise absolute value differences

$$\left| \begin{array}{|c|c|c|c|} \hline 56 & 32 & 10 & 18 \\ \hline 90 & 23 & 128 & 133 \\ \hline 24 & 26 & 178 & 200 \\ \hline 2 & 0 & 255 & 220 \\ \hline \end{array} - \begin{array}{|c|c|c|c|} \hline 10 & 20 & 24 & 17 \\ \hline 8 & 10 & 89 & 100 \\ \hline 12 & 16 & 178 & 170 \\ \hline 4 & 32 & 233 & 112 \\ \hline \end{array} \right| = \begin{array}{|c|c|c|c|} \hline 46 & 12 & 14 & 1 \\ \hline 82 & 13 & 39 & 33 \\ \hline 12 & 10 & 0 & 30 \\ \hline 2 & 32 & 22 & 108 \\ \hline \end{array} \rightarrow 456$$

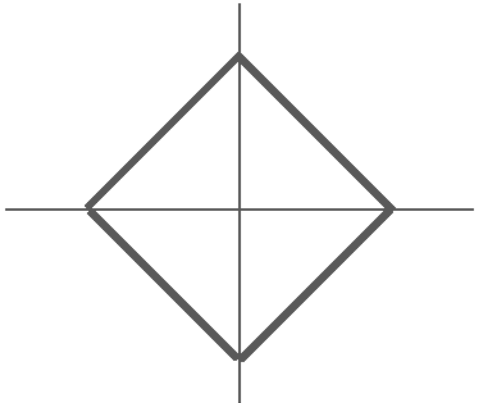
Slides from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Choice of distance metric

- Hyperparameter

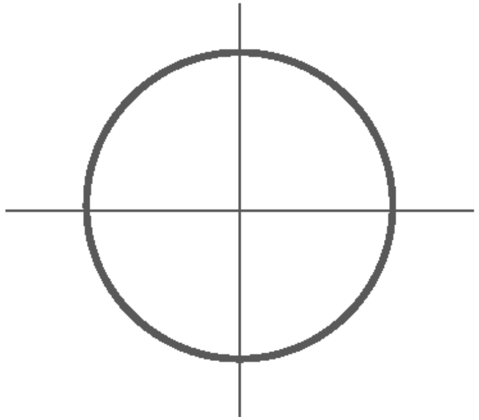
L1 (Manhattan) distance

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



L2 (Euclidean) distance

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



- Two most commonly used special cases of p-norm

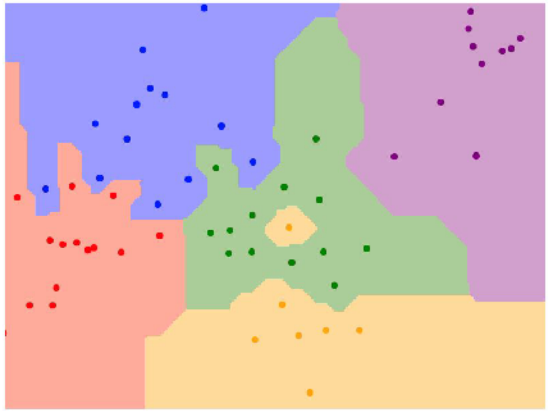
$$||x||_p = \left(|x_1|^p + \cdots + |x_n|^p\right)^{\frac{1}{p}} \qquad p \geq 1, x \in \mathbb{R}^n$$

Slide composited from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

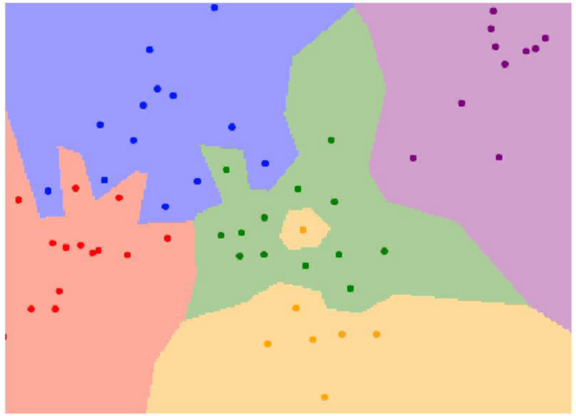
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



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

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

Hyperparameters

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

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

- How do we set them?
    - One option: try them all and see what works best

Setting Hyperparameters

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

Your Dataset

Slide composited from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

Your Dataset

Slide composited from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

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

Slide composited from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

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

BAD: No idea how algorithm will perform on new data

Slide composited from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Setting 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

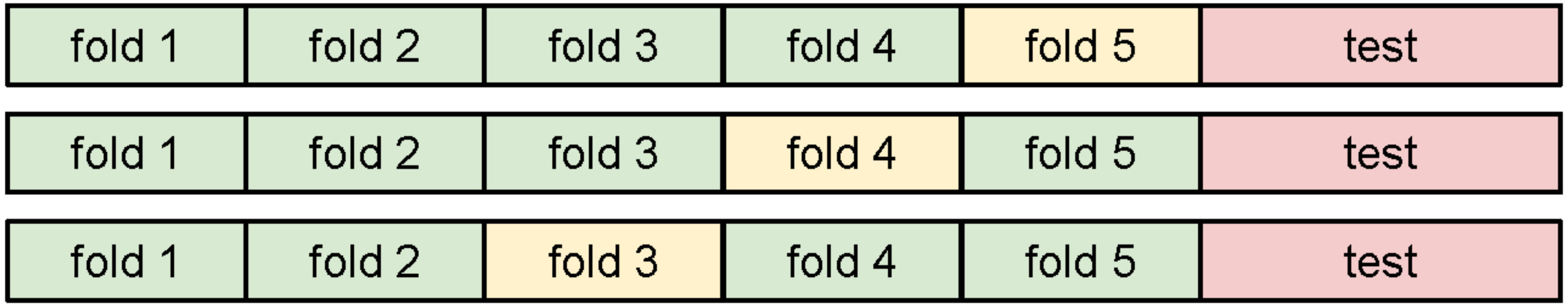
test

Slide composited from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Setting Hyperparameters

Your Dataset

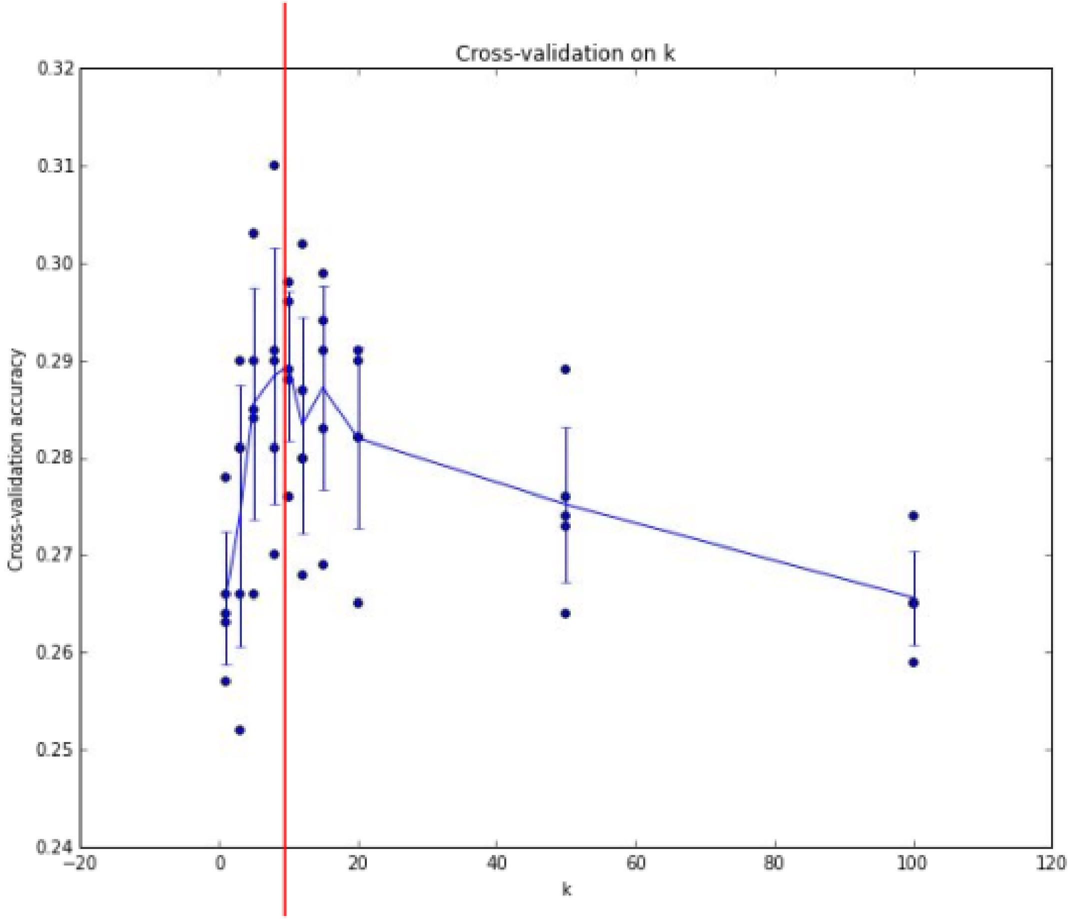
Idea #4: **Cross-Validation**: Split data into **folds**, try each fold as validation and average the results


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

Slide composited from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Hyperparameter Tuning



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)

Recap: How to pick hyperparameters?

- Methodology
  - Train and test
  - Train, validate, test

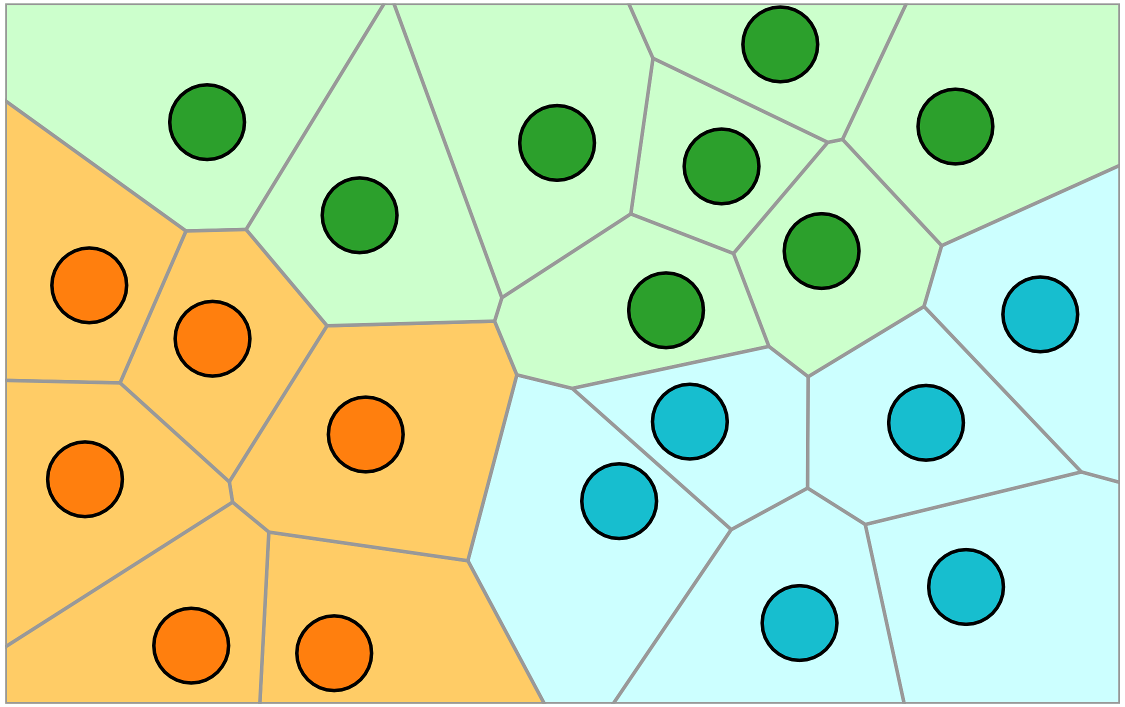
- Train for original model
- Validate to find hyperparameters
- Test to understand generalizability

kNN -- Complexity and Storage

- N training images, M test images

- Training: O(1)
- Testing: O(MN)

- We often need the opposite:
  - Slow training is ok
  - Fast testing is necessary

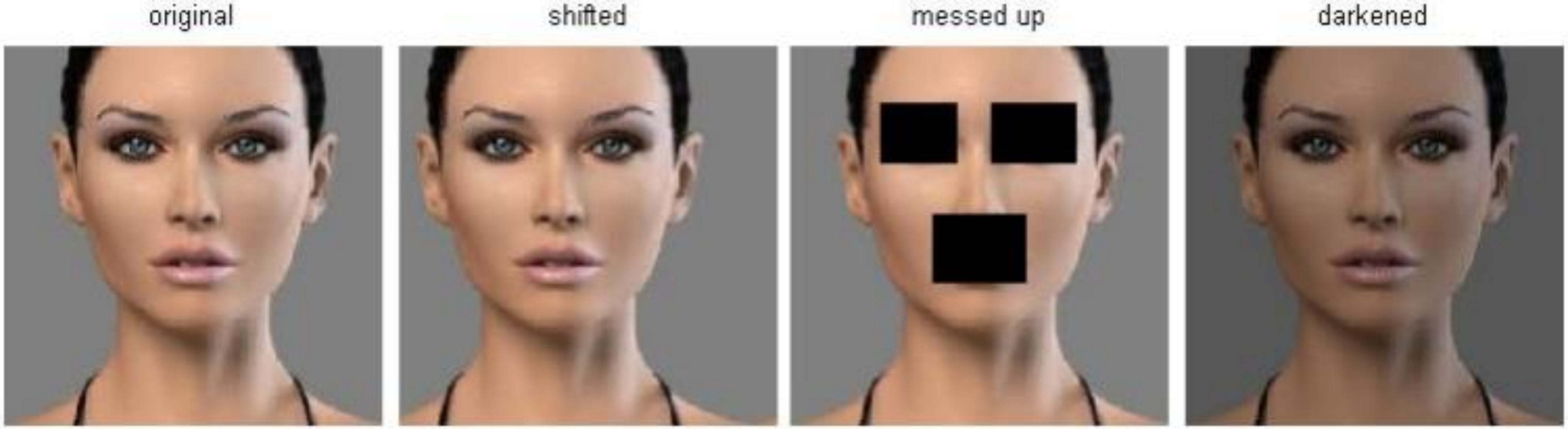


k-Nearest Neighbors: Summary

- 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!

Problems with KNN: Distance Metrics

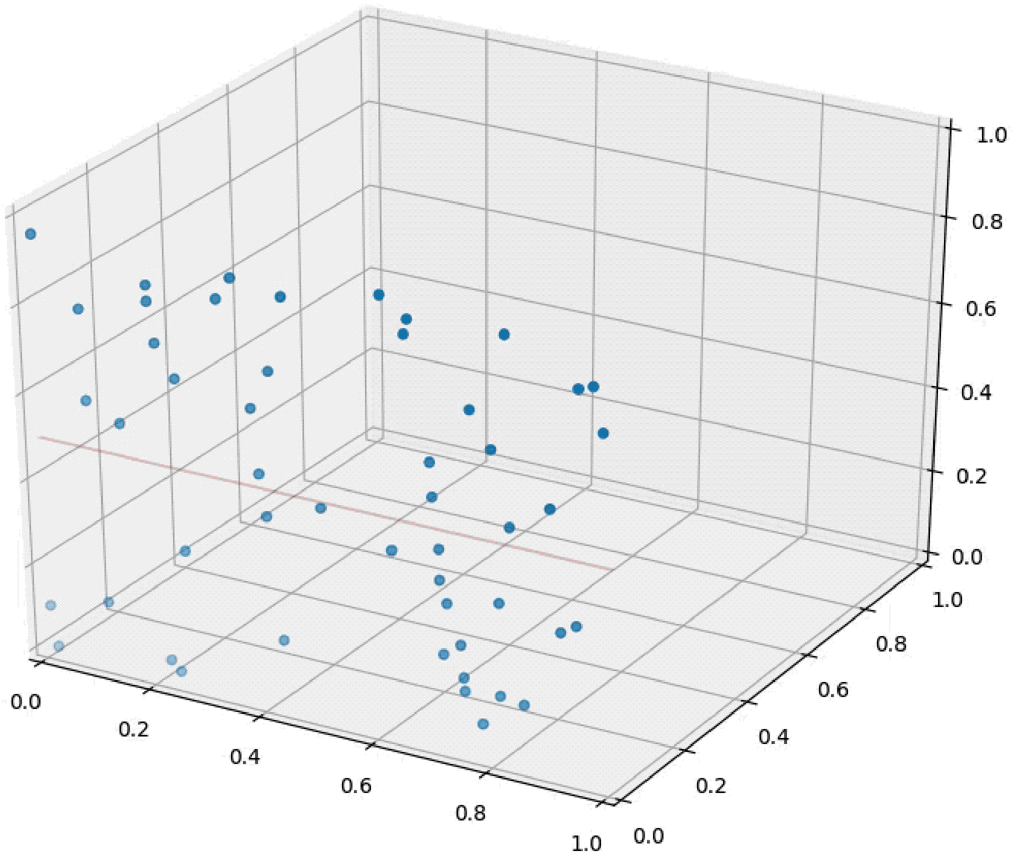
- terrible performance at test time
- distance metrics on level of whole images can be very unintuitive



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

Problems with KNN: The Curse of Dimensionality

- As the number of dimensions increases, the same amount of data becomes more sparse.
- Amount of data we need ends up being exponential in the number of dimensions



Animation from https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html

Linear Classifiers

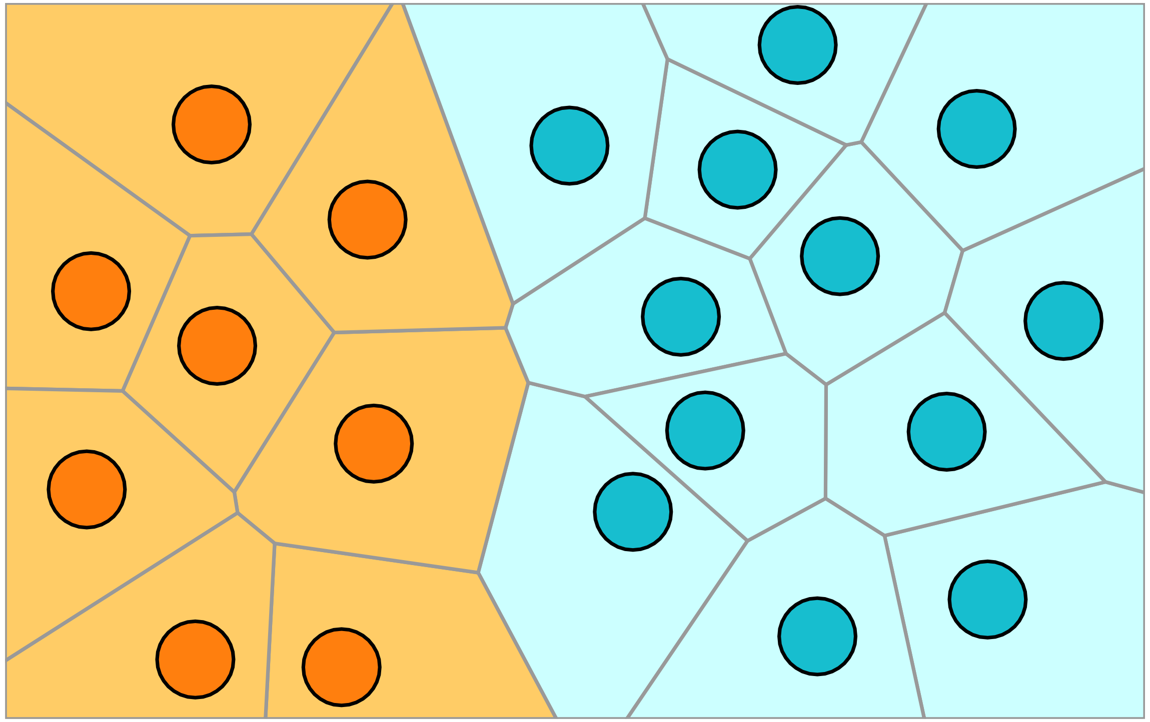
Neural Network

Linear classifiers

This image is CC0 1.0 public domain

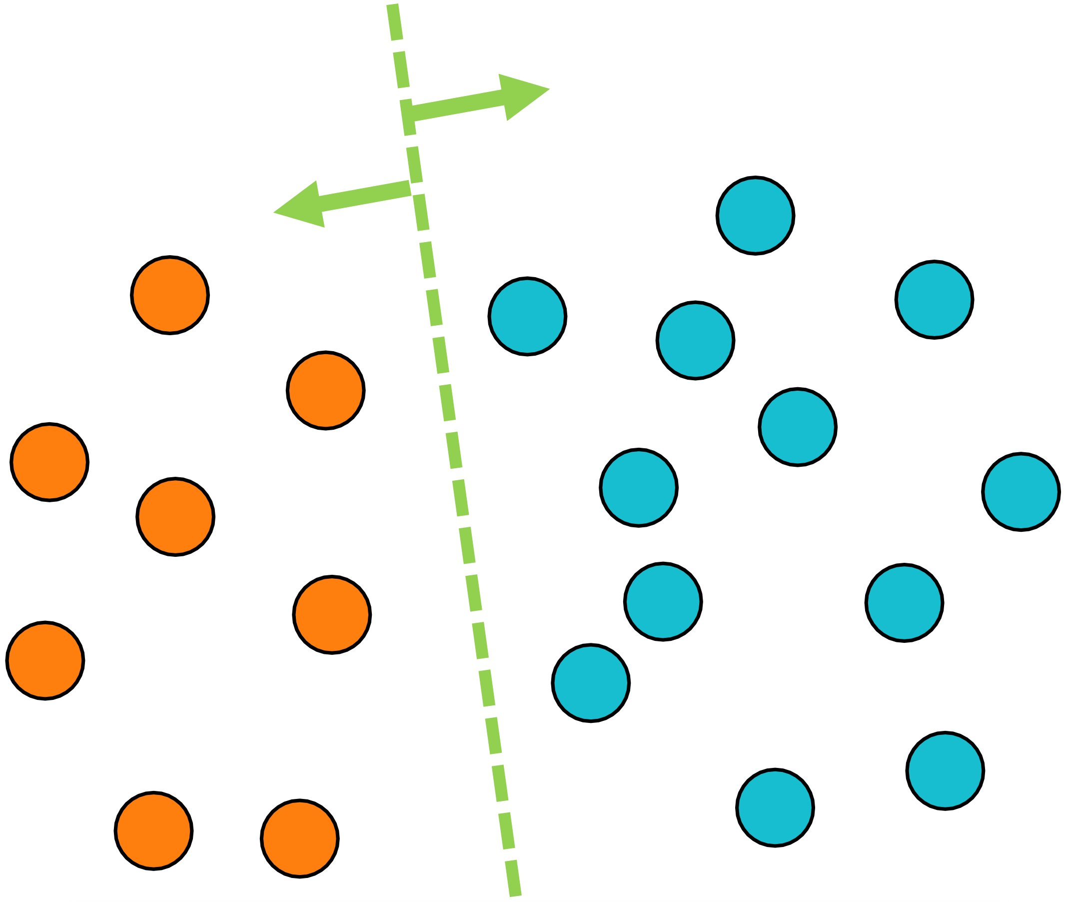
Linear Classification vs. Nearest Neighbors

- Nearest Neighbors
  - Store every image
  - Find nearest neighbors at test time, and assign same class



Linear Classification vs. Nearest Neighbors

- Nearest Neighbors
  - Store every image
  - Find nearest neighbors at test time, and assign same class
- Linear Classifier
  - Store hyperplanes that best separate different classes
  - We can compute continuous class score by calculating (signed) distance from hyperplane



We can interpret this as a linear "score function" for each class.

Score functions

class scores

Slide adapted from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Parametric Approach

image parameters

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

10 numbers, indicating class scores

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

Slide adapted from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Parametric Approach: Linear Classifier

$$\underbrace{\boxed{f(x,W)}}_{\text{10x1}} = \underbrace{\boxed{W}}_{\text{10x3072}}\overset{\text{3072x1}}{\boxed{x}}$$

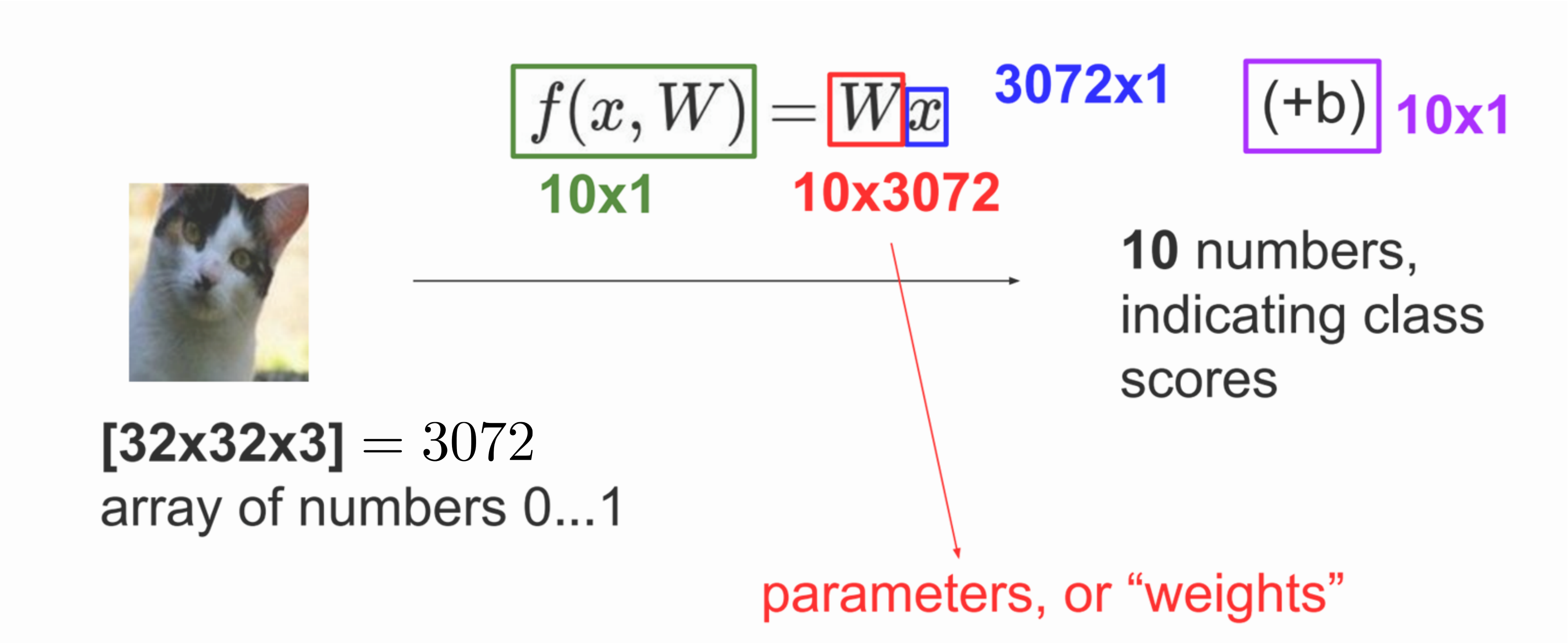
[32x32x3] $= 3072$
array of numbers 0...1

10 numbers,
indicating class
scores

parameters, or "weights"

Slide adapted from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Parametric Approach: Linear Classifier



$$\boxed{f(x,W)} = \boxed{W}\boxed{x} \quad \text{3072x1} \qquad \boxed{\text{(+b)}} \; \text{10x1}$$

10x1

10x3072

[32x32x3] $= 3072$

array of numbers 0...1

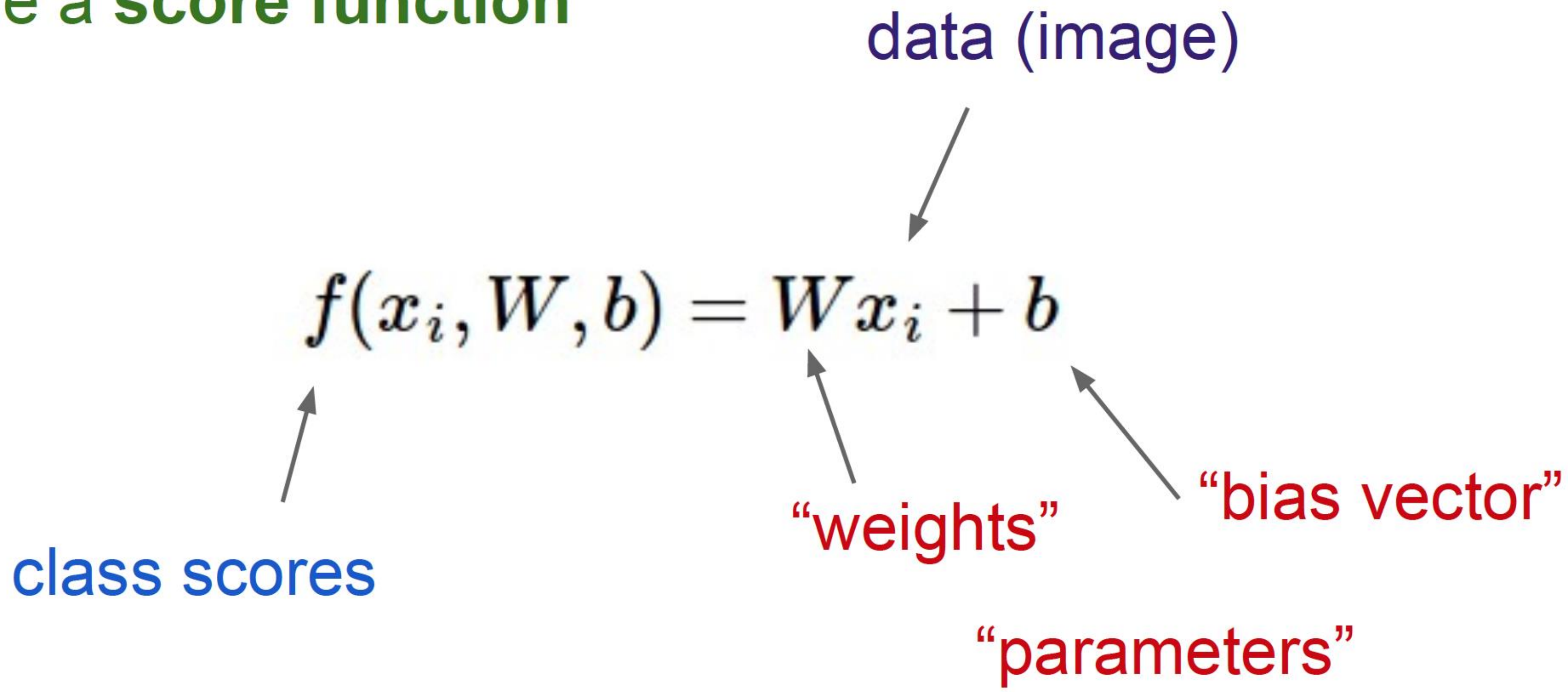
10 numbers, indicating class scores

parameters, or “weights”

Slide adapted from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Linear Classifier

define a **score function**



data (image)

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

class scores

“weights”

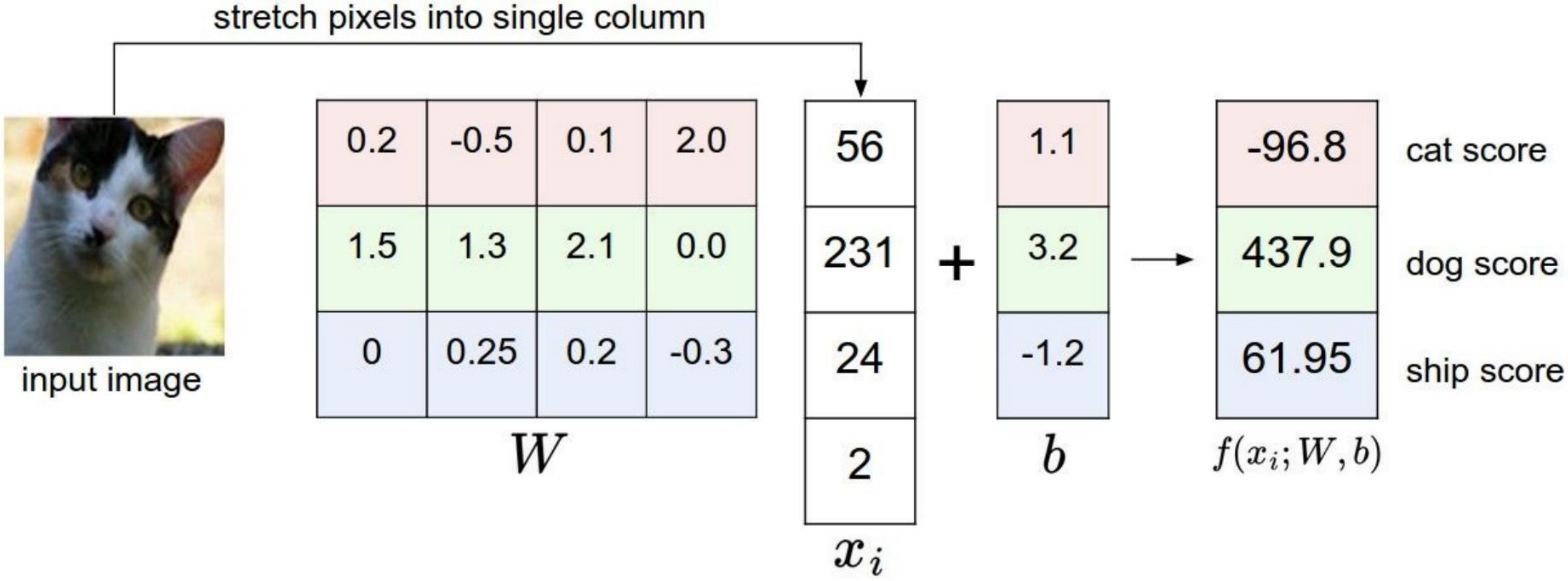
“bias vector”

“parameters”

Slide adapted from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Interpretation: Algebraic

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



stretch pixels into single column

input image

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

Slide adapted from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Interpretation: Geometric

- Parameters define a hyperplane for each class:

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

- We can think of each class score as defining a distribution that is proportional to distance from the corresponding hyperplane

Interpretation: Template matching

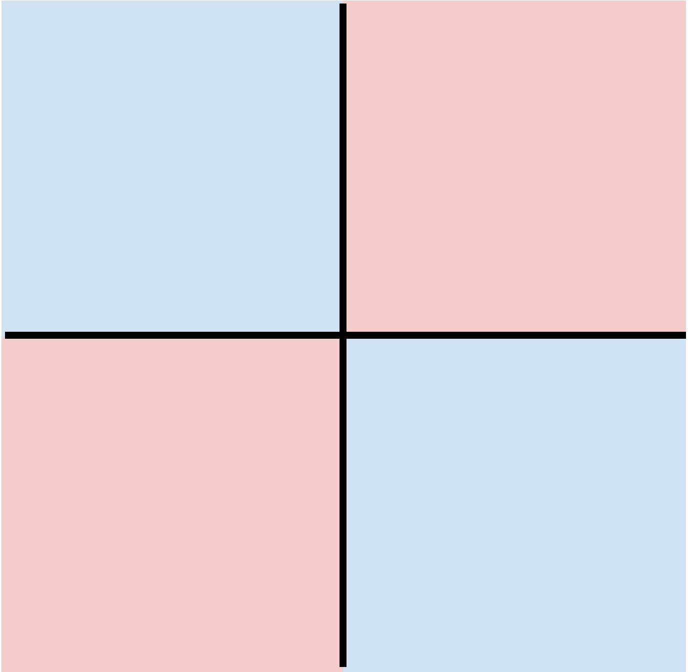
- We can think of the rows in W as templates for each class

Rows of *W* in $f(x_i, W, b) = Wx_i + b$

Hard Cases for a Linear Classifier

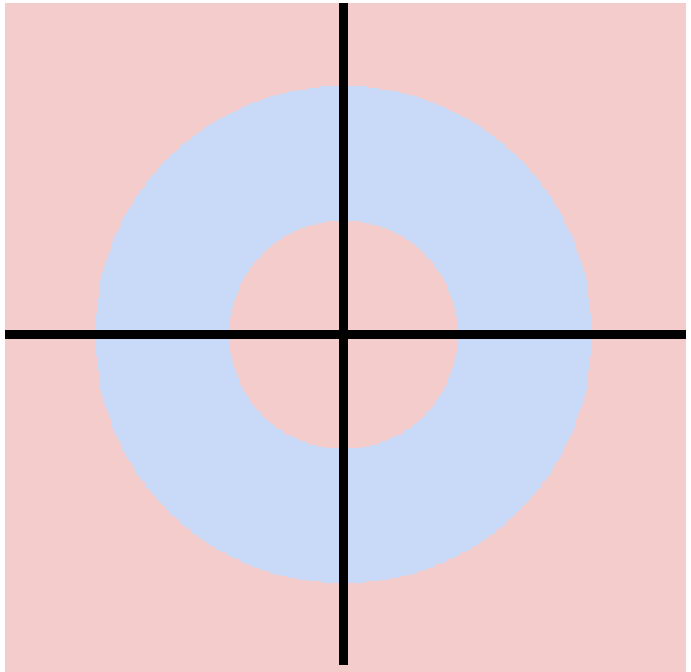
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



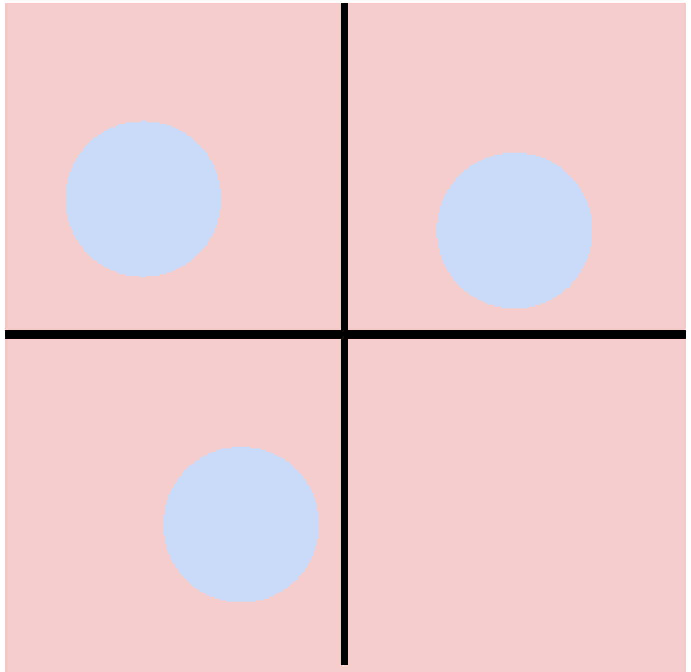
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

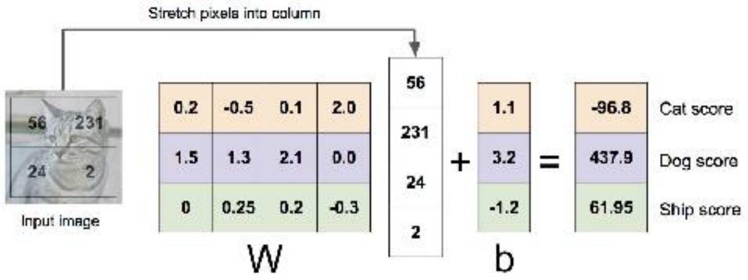
Class 2:
Everything else



Linear Classifier: Three Viewpoints

Algebraic Viewpoint

f(x,W) = Wx

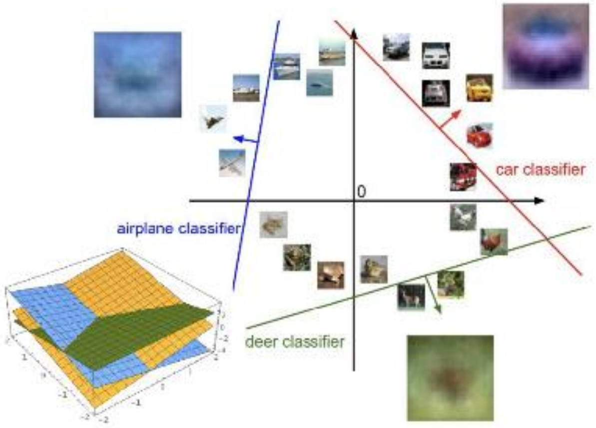


Visual Viewpoint

One template
per class

Geometric Viewpoint

Hyperplanes
cutting up space



So far: Defined a (linear) score function $f(x,W) = Wx + b$

Example class scores for 3 images for some W:

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

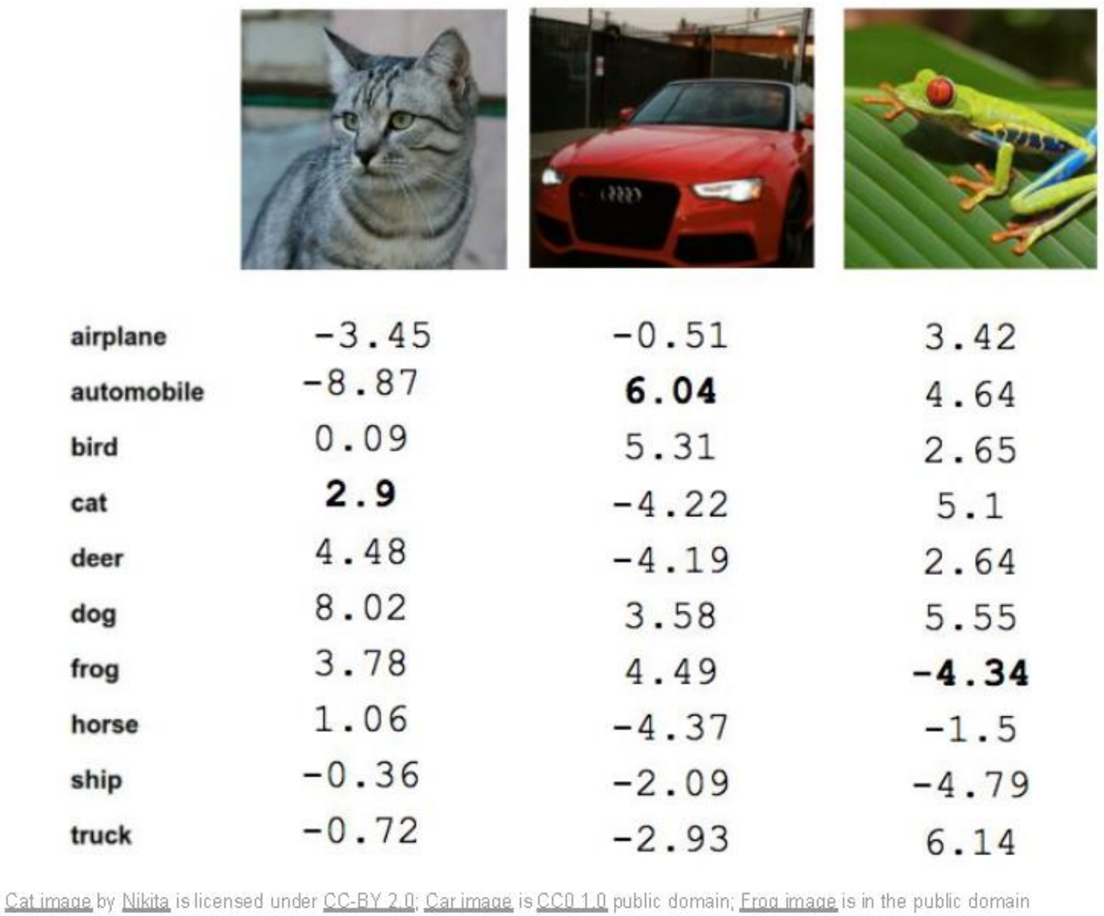
airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

Recap

- Learning methods
  - k-Nearest Neighbors
  - **Linear classification**

- Classifier outputs a **score function** giving a score to each class
- How do we define how good a classifier is based on the training data? (Spoiler: define a *loss function*)

Linear classification



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

Output scores

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)**

Loss functions

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

A **loss function** tells how good our current classifier is

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and
y_i is (integer) label

Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Loss function, cost/objective function

- Given ground truth labels (y_i), scores $f(x_i, \mathbf{W})$
  - how unhappy are we with the scores?

- Loss function or objective/cost function measures unhappiness

- During training, **want to find the parameters W that minimize the loss function**

Simpler example: binary classification

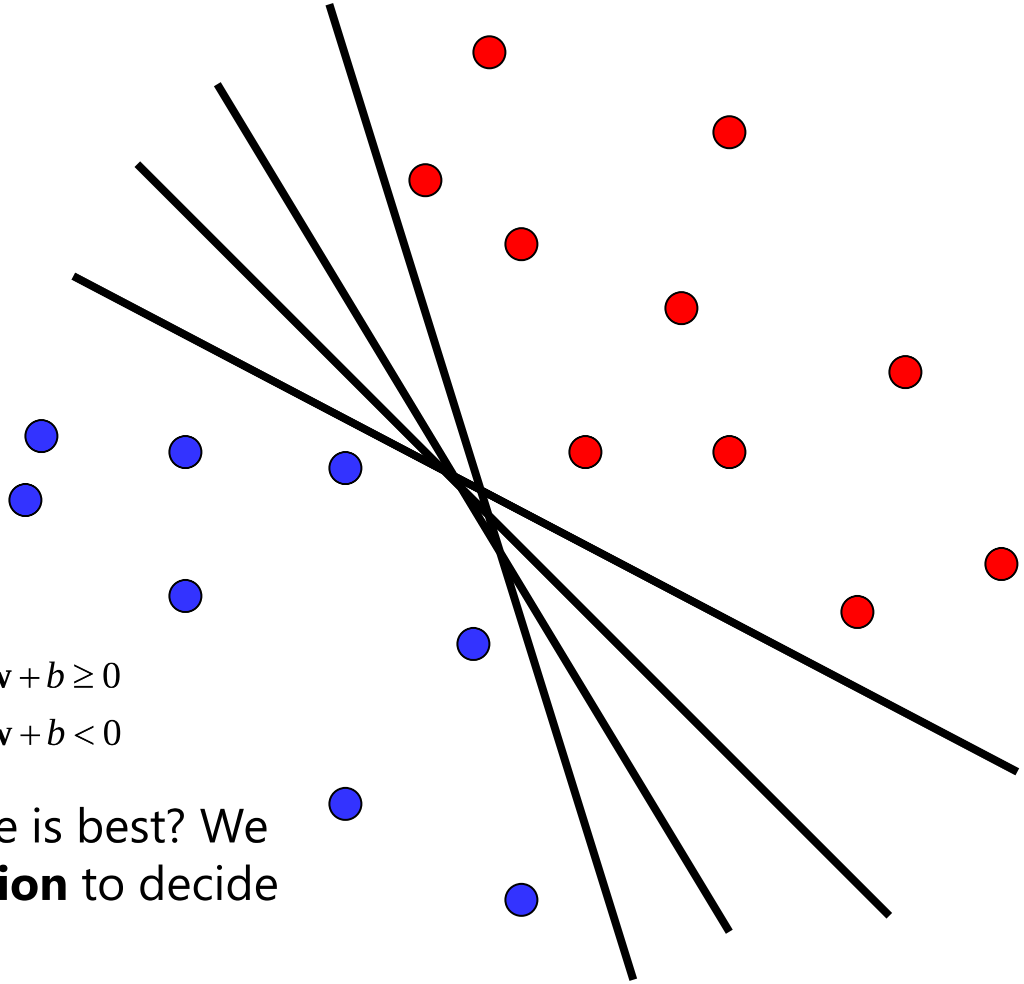
- Two classes (e.g., “cat” and “not cat”)
    - AKA “positive” and “negative” classes

cat

not cat

Linear classifiers

- Find linear function (*hyperplane*) to separate positive and negative examples



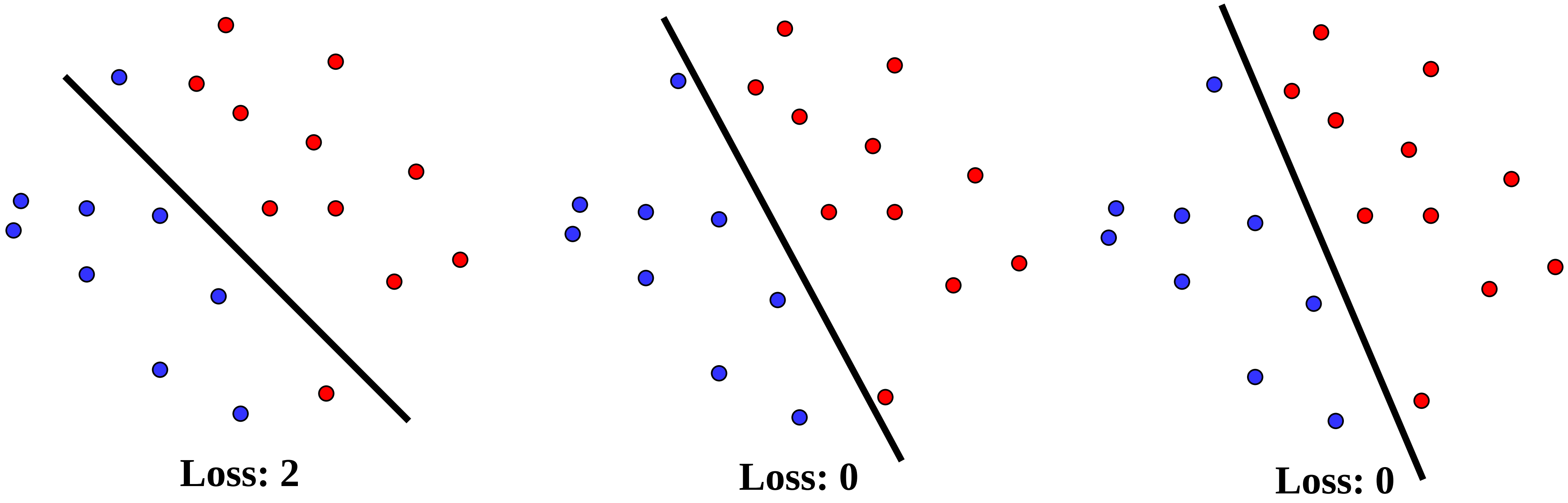
$$\mathbf{x}_i \text{ positive}: \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 0$$

$$\mathbf{x}_i \text{ negative}: \quad \mathbf{x}_i \cdot \mathbf{w} + b < 0$$

Which hyperplane is best? We need a **loss function** to decide

What is a good loss function?

- One possibility: Number of misclassified examples
  - Problems: discrete, can’t break ties
  - We want the loss to lead to *good generalization*
  - We want the loss to work for more than 2 classes



Softmax classifier

- Interpret Scores as unnormalized log probabilities of classes

$$f(x_i, W) = W x_i \quad \text{(score function)}$$

$$\boxed{\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}},$$

softmax function

Squashes values into *probabilities* ranging from 0 to 1

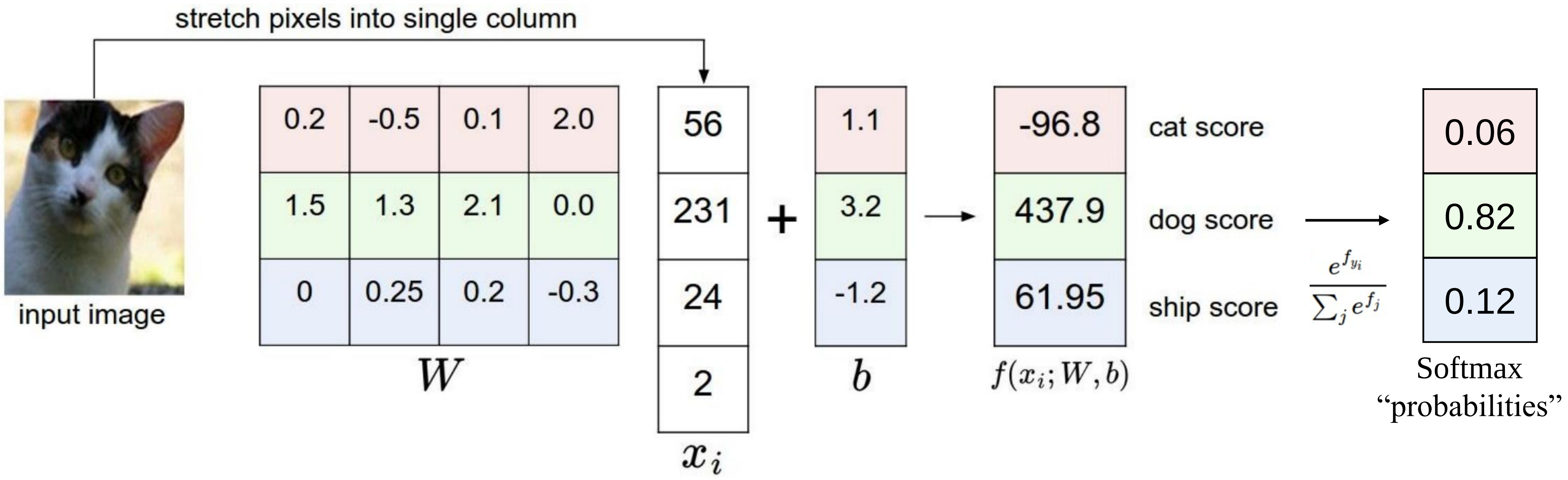
$$P(y_i \mid x_i; W)$$

Example with three classes:

$$[1, -2, 0] \rightarrow [e^1, e^{-2}, e^0] = [2.71, 0.14, 1] \rightarrow [0.7, 0.04, 0.26]$$

Softmax classifier

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



stretch pixels into single column

input image

$$\underbrace{\begin{bmatrix}0.2 & -0.5 & 0.1 & 2.0\\ 1.5 & 1.3 & 2.1 & 0.0\\ 0 & 0.25 & 0.2 & -0.3\end{bmatrix}}_{W} \underbrace{\begin{bmatrix}56\\231\\24\\2\end{bmatrix}}_{x_i} + \underbrace{\begin{bmatrix}1.1\\3.2\\-1.2\end{bmatrix}}_{b} \rightarrow \underbrace{\begin{bmatrix}-96.8\\437.9\\61.95\end{bmatrix}}_{f(x_i;W,b)} \begin{matrix}\text{cat score}\\ \text{dog score}\\ \text{ship score}\end{matrix} \xrightarrow[\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}]{} \begin{bmatrix}0.06\\0.82\\0.12\end{bmatrix}$$

Softmax
“probabilities”

Cross-entropy loss

$f(x_i, W) = Wx_i$ (score function)

Cross-entropy loss

$$f(x_i, W) = W x_i$$ (score function)

f_{y_i} : score of correct class

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

$$L_i = -f_{y_i} + \log \sum_j e^{f_j}$$

We call L_i *cross-entropy loss*

Cross-entropy loss

$f(x_i, W) = Wx_i$ (score function)

$$L_i = -\log\left(\underbrace{\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}}_{\boxed{P(y_i \mid x_i; W)}}\right) \qquad L_i = -f_{y_i} + \log\sum_j e^{f_j}$$

We call L_i *cross-entropy loss*

i.e. we're minimizing the negative log likelihood.

Losses

- Cross-entropy loss is just one possible loss function
  - One nice property is that it reinterprets scores as probabilities, which have a natural meaning

- SVM (max-margin) loss functions also used to be popular
  - But currently, cross-entropy is the most common classification loss

Summary

- Have score function and loss function
  - Currently, score function is based on linear classifier
  - Next, will generalize to convolutional neural networks
- Find W and b to minimize loss

$$L = \underbrace{\frac{1}{N}\sum_i -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)}_{\text{Average of cross-entropy loss over all training examples}} + \lambda \sum_k \sum_l W_{k,l}^2$$

Regularization term
(will talk about this later)

Questions?