

Lecture 2:

Image Classification pipeline

Administrative: Piazza

For questions about midterm, poster session, projects, etc, use Piazza!

SCPD students: Use your @stanford.edu address to register for Piazza; contact scpd-customerservice@stanford.edu for help.

Administrative: Assignment 1

Out yesterday, due 4/18 11:59pm

- K-Nearest Neighbor
- Linear classifiers: SVM, Softmax
- Two-layer neural network
- Image features

Administrative: Friday Discussion Sections

Fridays 12:30pm - 1:20pm in Skilling Auditorium

Hands-on tutorials, with more practical detail than main lecture

Check course website for schedule:

<http://cs231n.stanford.edu/syllabus.html>

This Friday: Python / numpy / Google Cloud setup

Administrative: Python + Numpy

CS231n Convolutional Neural Networks for Visual Recognition

Python Numpy Tutorial

This tutorial was contributed by [Justin Johnson](#).

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

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

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

Administrative: Google Cloud

We will be using Google Cloud in this class

We will be distributing coupons coupons to all enrolled students

See our tutorial here for walking through Google Cloud setup:

<http://cs231n.github.io/gce-tutorial/>

Image Classification: A core task in Computer Vision



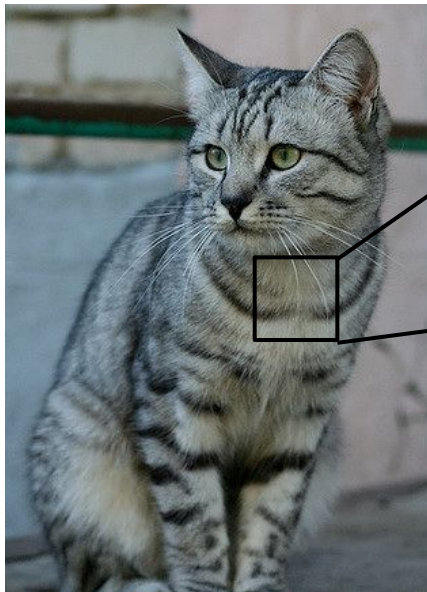
This image by Nikita is
licensed under [CC-BY 2.0](#)

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



cat

The Problem: Semantic Gap



This image by Nikita is
licensed under [CC-BY 2.0](#)

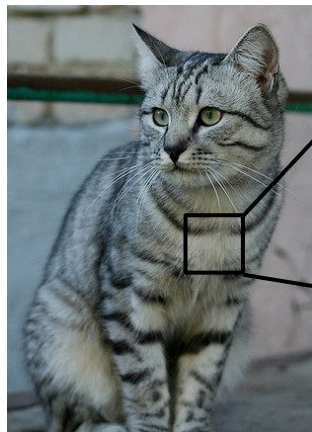
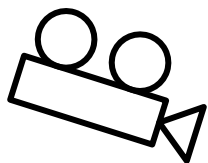
```
[[105 112 108 111 104 99 106 99 96 103 112 119 104 97 93 87]  
[ 91 98 102 106 104 79 98 103 99 105 123 136 110 105 94 85]  
[ 76 85 90 105 120 105 87 96 95 99 115 112 106 103 99 85]  
[ 99 81 81 93 120 131 127 100 95 98 102 99 96 93 101 94]  
[106 91 61 64 69 91 88 85 101 107 109 98 75 84 96 95]  
[114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]  
[133 137 147 103 65 81 80 65 52 54 74 84 102 93 85 82]  
[128 137 144 140 109 95 86 70 62 65 63 63 60 73 86 101]  
[125 133 148 137 119 121 117 94 65 79 80 65 54 64 72 98]  
[127 125 131 147 133 127 126 131 111 96 89 75 61 64 72 84]  
[115 114 109 123 150 148 131 110 113 109 100 92 74 65 72 78]  
[ 89 93 90 97 108 147 131 118 113 114 113 109 106 95 77 80]  
[ 63 77 86 81 77 79 102 123 117 115 117 125 125 130 115 87]  
[ 62 65 82 89 78 71 80 101 124 126 119 101 107 114 131 119]  
[ 63 65 75 88 89 71 62 81 120 138 135 105 81 98 110 118]  
[ 87 65 71 87 106 95 69 45 76 130 126 107 92 94 105 112]  
[118 97 82 86 117 123 116 66 41 51 95 93 89 95 102 107]  
[164 146 112 80 82 120 124 104 76 48 45 66 88 101 102 109]  
[157 170 157 120 93 86 114 132 112 97 69 55 70 82 99 94]  
[130 128 134 161 139 100 109 118 121 134 114 87 65 53 69 86]  
[128 112 96 117 150 144 120 115 104 107 102 93 87 81 72 79]  
[123 107 96 86 83 112 153 149 122 109 104 75 80 107 112 99]  
[122 121 102 80 82 86 94 117 145 148 153 102 58 78 92 107]  
[122 164 148 103 71 56 78 83 93 103 119 139 102 61 69 84]]
```

What the computer sees

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

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

Challenges: Viewpoint variation



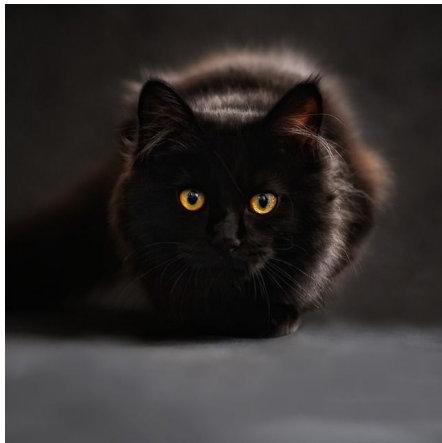
[	105	112	108	111	104	99	106	99	96	103	112	119	104	97	93	87]
[	91	98	102	106	104	79	98	103	99	105	123	136	110	105	94	85]
[	76	85	90	105	128	105	87	96	95	90	115	112	106	103	99	85]
[	99	81	81	93	120	131	127	100	95	98	102	99	96	93	101	94]
[	106	91	61	64	69	91	88	85	101	107	109	90	75	84	96	95]
[	114	100	85	55	55	69	64	54	64	87	112	129	98	74	84	91]
[	133	137	147	103	65	81	80	65	52	54	74	84	102	93	85	82]
[	128	137	144	140	109	95	86	70	62	65	63	63	60	73	86	101]
[	125	133	140	137	119	121	117	94	65	79	80	65	54	64	72	90]
[	127	125	131	147	133	127	126	131	111	96	89	75	61	64	72	84]
[	115	114	109	123	150	140	131	118	113	109	100	92	74	65	72	78]
[	89	93	90	97	100	147	131	118	113	114	113	100	106	95	77	80]
[	63	77	86	81	77	79	102	123	117	115	117	125	125	130	115	87]
[	62	65	82	80	78	71	80	101	124	126	119	101	107	114	131	119]
[	63	65	75	80	69	71	62	81	120	130	135	105	61	90	110	110]
[	87	65	71	87	106	95	69	45	76	130	126	107	92	94	105	112]
[	118	97	82	86	117	123	116	66	41	51	95	93	89	95	102	107]
[	164	146	112	80	82	120	124	104	76	40	45	66	88	101	102	109]
[	157	170	157	120	63	86	114	132	112	97	69	55	70	82	99	94]
[	130	120	134	161	139	100	109	118	121	134	114	87	65	53	69	86]
[	120	112	96	117	150	144	120	115	104	107	102	93	87	81	72	79]
[	123	107	96	86	83	112	153	149	122	109	104	75	80	107	112	99]
[	122	121	102	80	82	86	94	117	145	140	153	102	50	78	92	107]
[	122	164	148	103	71	56	78	83	93	103	119	139	102	61	69	84]

All pixels change when
the camera moves!

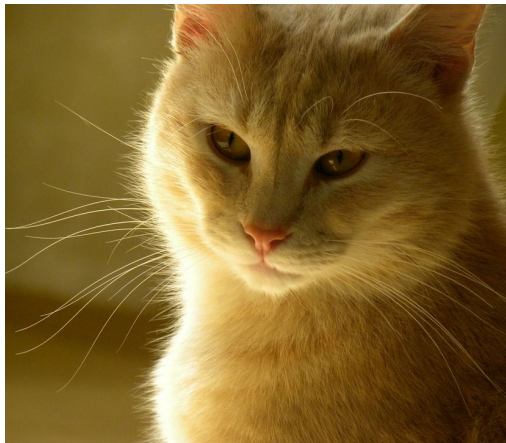
Challenges: Illumination



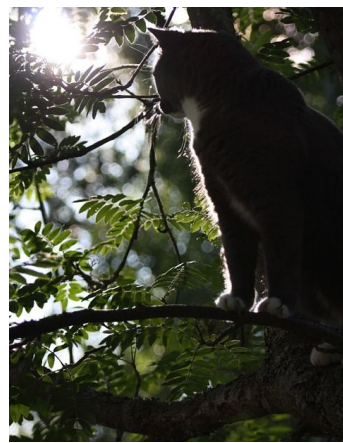
[This image](#) is [CC0 1.0](#) public domain



[This image](#) is [CC0 1.0](#) public domain

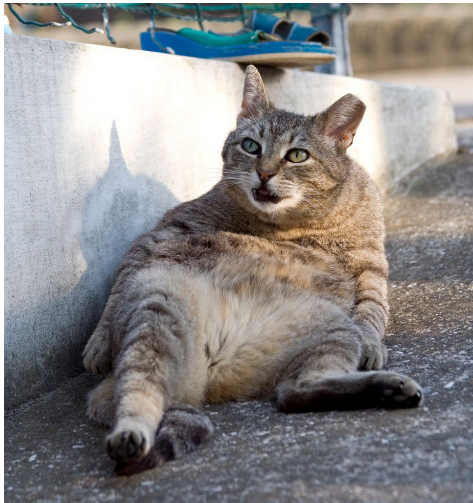


[This image](#) is [CC0 1.0](#) public domain



[This image](#) is [CC0 1.0](#) public domain

Challenges: Deformation



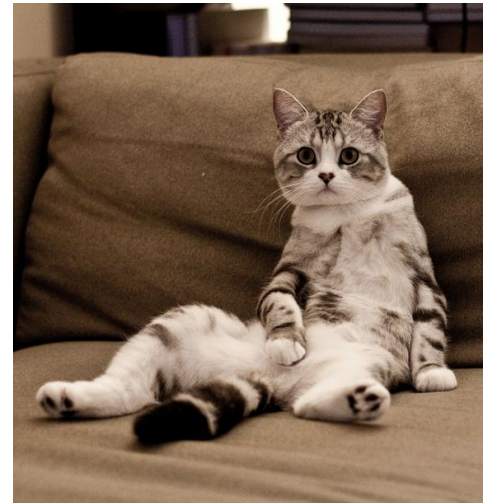
This image by [Umberto Salvagnin](#)
is licensed under [CC-BY 2.0](#)



This image by [Umberto Salvagnin](#)
is licensed under [CC-BY 2.0](#)



This image by [sare bear](#) is
licensed under [CC-BY 2.0](#)



This image by [Tom Thai](#) is
licensed under [CC-BY 2.0](#)

Challenges: Occlusion



This image is [CC0 1.0](#) public domain



This image is [CC0 1.0](#) public domain



This image by [jonsson](#) is licensed under [CC-BY 2.0](#)

Challenges: Background Clutter



[This image](#) is [CC0 1.0](#) public domain



[This image](#) is [CC0 1.0](#) public domain

Challenges: Intraclass variation



[This image](#) is [CC0 1.0](#) public domain

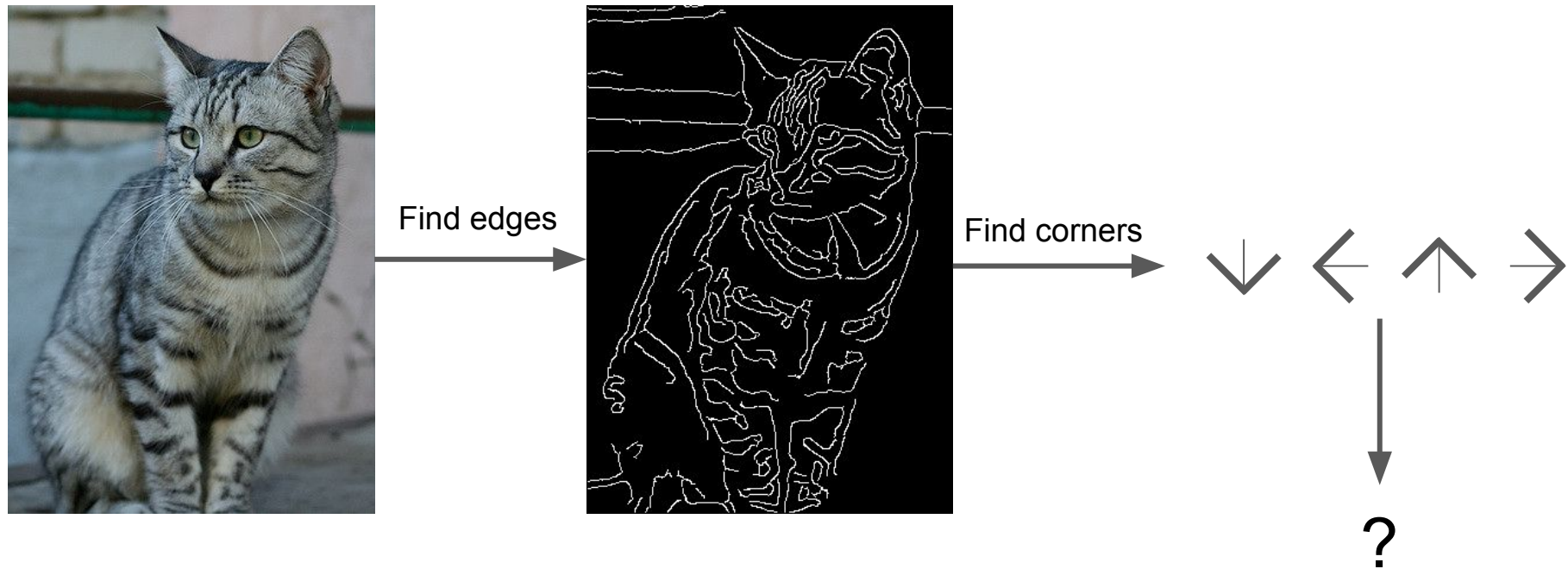
An image classifier

```
def classify_image(image):  
    # Some magic here?  
    return class_label
```

Unlike e.g. sorting a list of numbers,

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

Attempts have been made



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

Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

Example training set

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

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

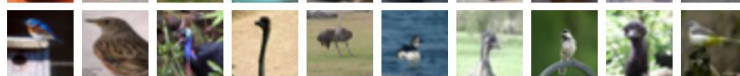
airplane



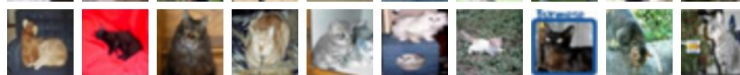
automobile



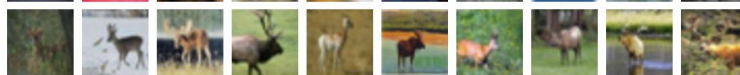
bird



cat



deer



First classifier: **Nearest Neighbor**

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



Memorize all
data and labels

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



Predict the label
of the most similar
training image

Example Dataset: CIFAR10

10 classes

50,000 training images

10,000 testing images

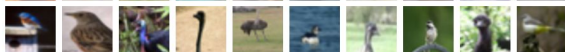
airplane



automobile



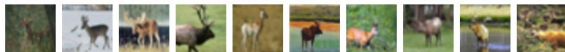
bird



cat



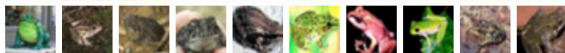
deer



dog



frog



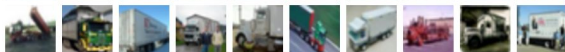
horse



ship



truck



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

Example Dataset: CIFAR10

10 classes

50,000 training images

10,000 testing images



Test images and nearest neighbors



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

Distance Metric to compare images

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	→ 456 add
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	

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
```

Memorize 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 each test image:
Find closest train image
Predict label of nearest 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: With N examples, how fast are training and prediction?

```

import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

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

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

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

        return Ypred

```

Nearest Neighbor classifier

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

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

```

import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

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

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

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

        return Ypred

```

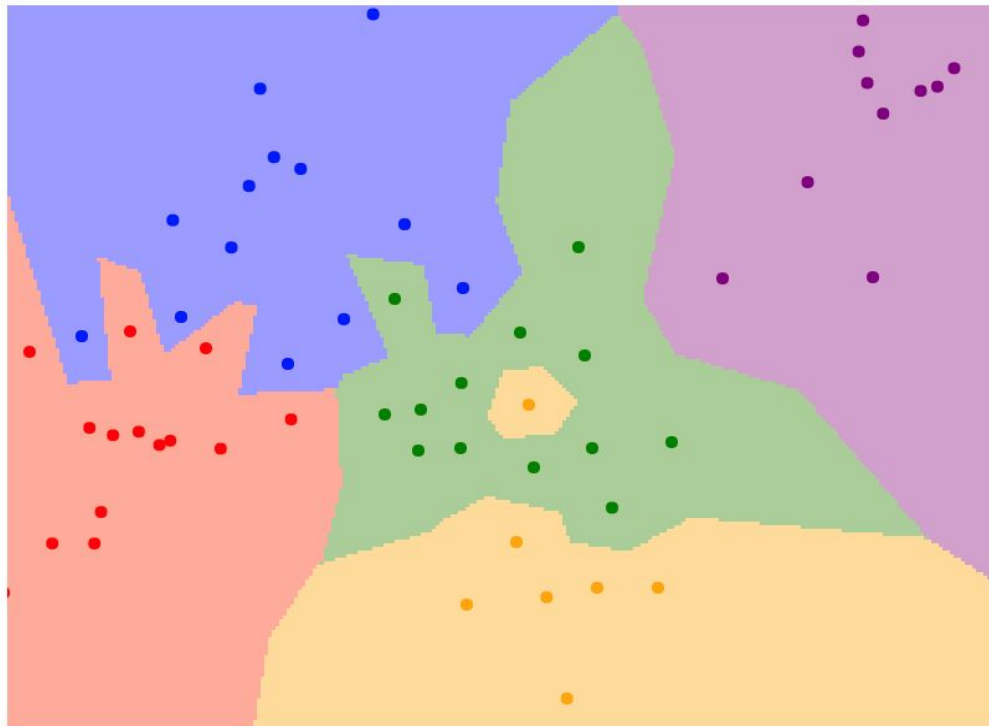
Nearest Neighbor classifier

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

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

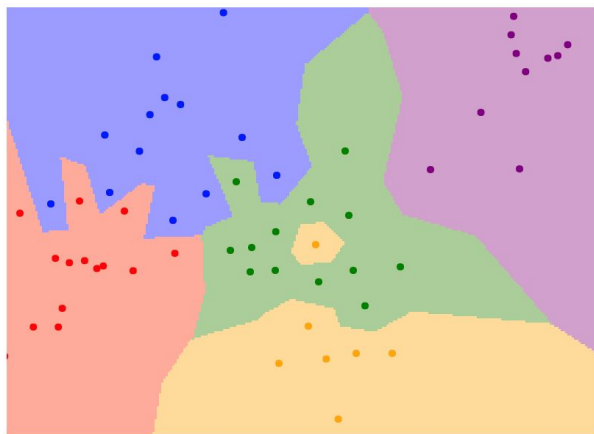
This is bad: we want classifiers that are **fast** at prediction; **slow** for training is ok

What does this look like?

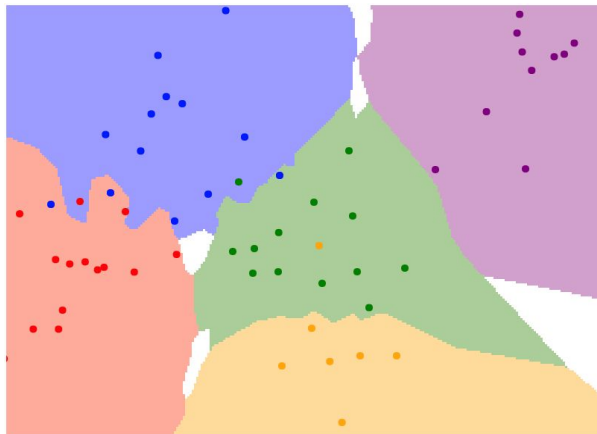


K-Nearest Neighbors

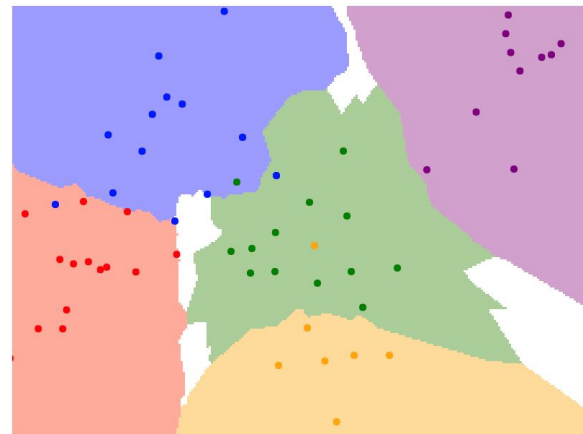
Instead of copying label from nearest neighbor,
take **majority vote** from K closest points



$K = 1$



$K = 3$



$K = 5$

What does this look like?



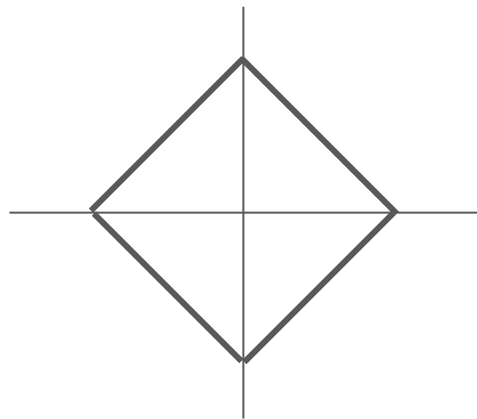
What does this look like?



K-Nearest Neighbors: Distance Metric

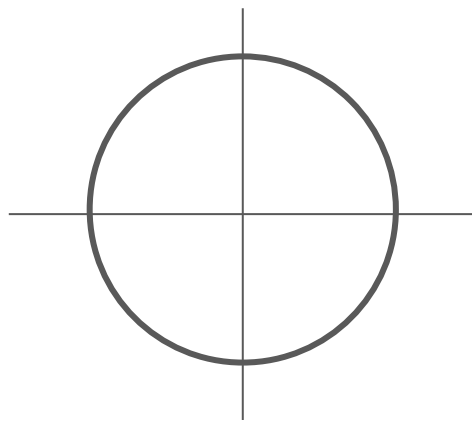
L1 (Manhattan) distance

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



L2 (Euclidean) distance

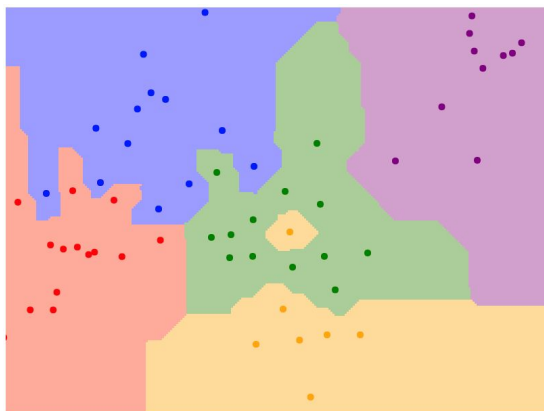
$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

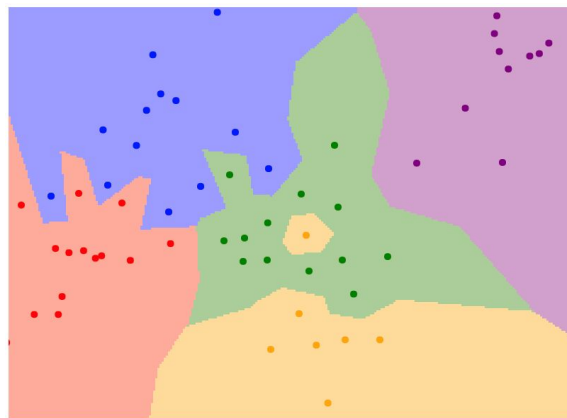
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



K = 1

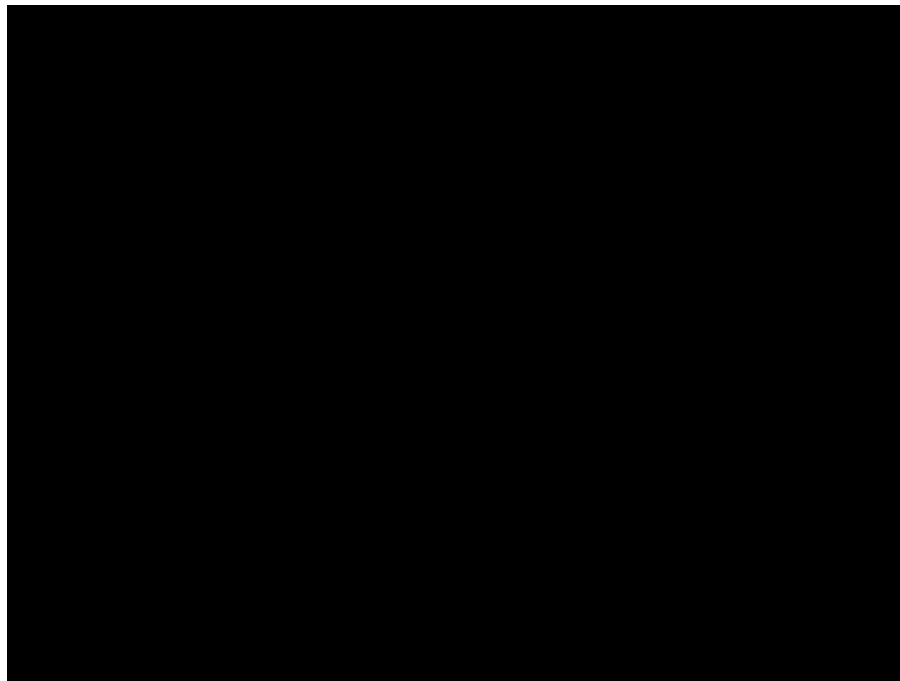
L2 (Euclidean) distance

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



K = 1

K-Nearest Neighbors: Demo Time



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

Hyperparameters

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

What is the best **distance** to use?

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

Hyperparameters

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

What is the best **distance** to use?

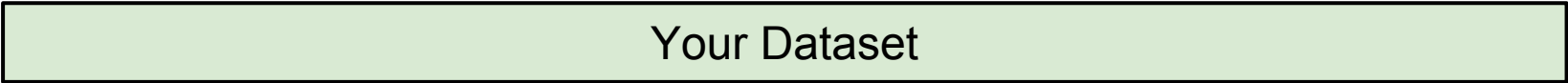
These are **hyperparameters**: choices about the algorithm that we set rather than learn

Very problem-dependent.

Must try them all out and see what works best.

Setting Hyperparameters

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

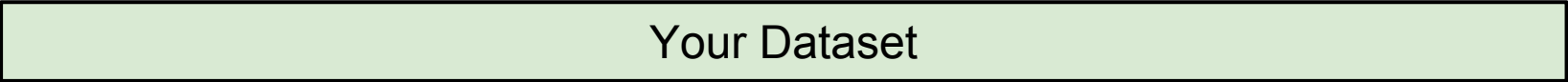


Your Dataset

Setting Hyperparameters

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

BAD: $K = 1$ always works perfectly on training data

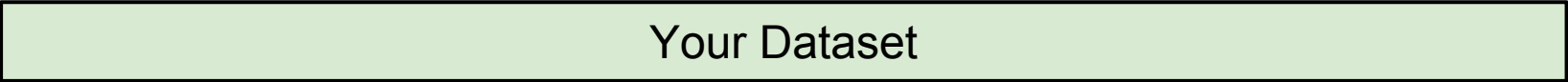


Your Dataset

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



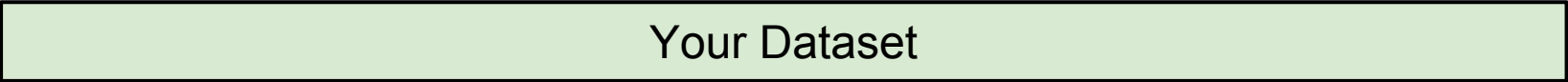
train

test

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



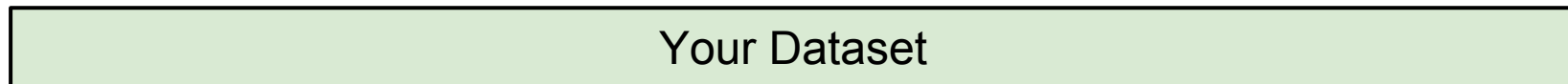
train

test

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



Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!



Setting Hyperparameters

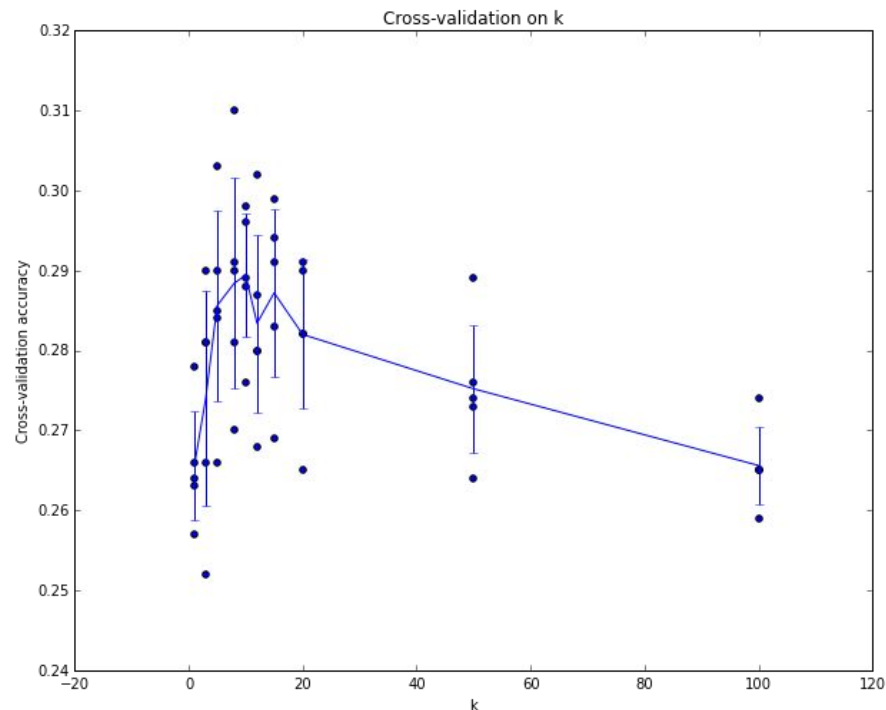
Your Dataset

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

Setting Hyperparameters



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

Each point: single
outcome.

The line goes
through the mean, bars
indicated standard
deviation

(Seems that $k \approx 7$ works best
for this data)

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

- Very slow at test time
- Distance metrics on pixels are not informative

Original



Boxed



Shifted



Tinted

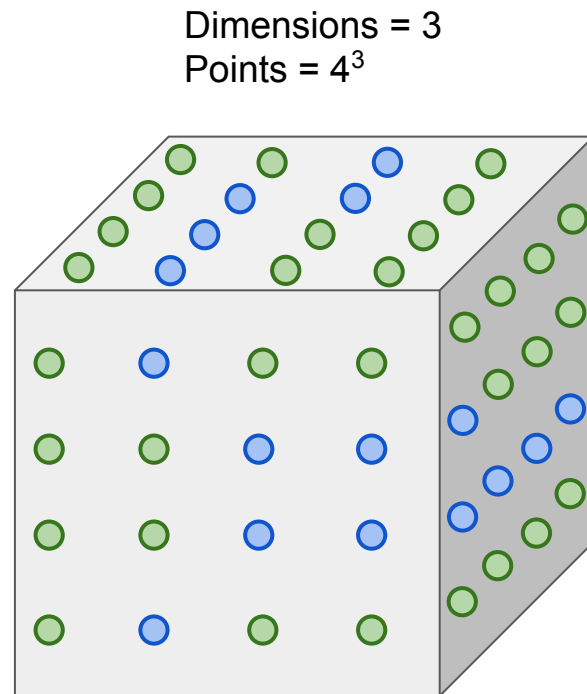
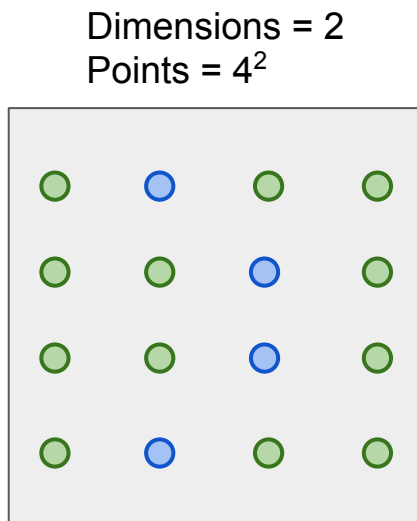
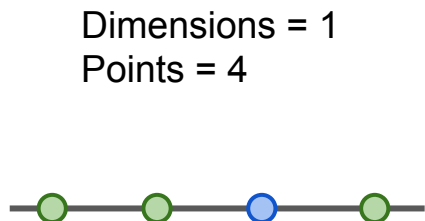


(all 3 images have same L2 distance to the one on the left)

[Original image is](#)
[CC0 public domain](#)

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

- Curse of dimensionality



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!

Linear Classification

Neural Network

Linear
classifiers

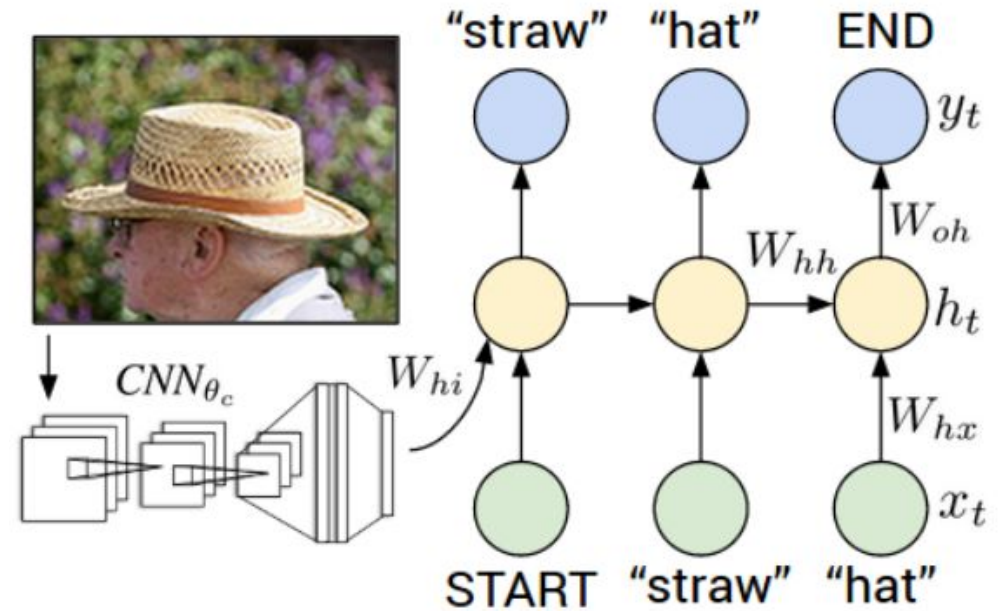


[This image](#) is [CC0 1.0](#) public domain

Two young girls are playing with lego toy. Boy is doing backflip on wakeboard



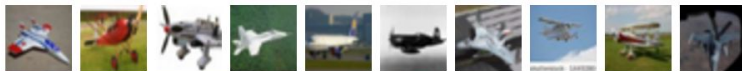
Man in black shirt is playing guitar. Construction worker in orange safety vest is working on road.



Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
 Figures copyright IEEE, 2015. Reproduced for educational purposes.

Recall CIFAR10

airplane



automobile



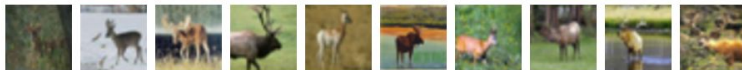
bird



cat



deer



dog



frog



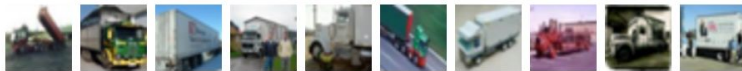
horse



ship



truck



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

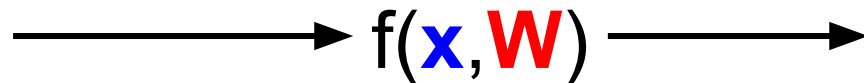
10,000 test images.

Parametric Approach

Image



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



10 numbers giving
class scores

↑
W

parameters
or weights

Parametric Approach: Linear Classifier

Image



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

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

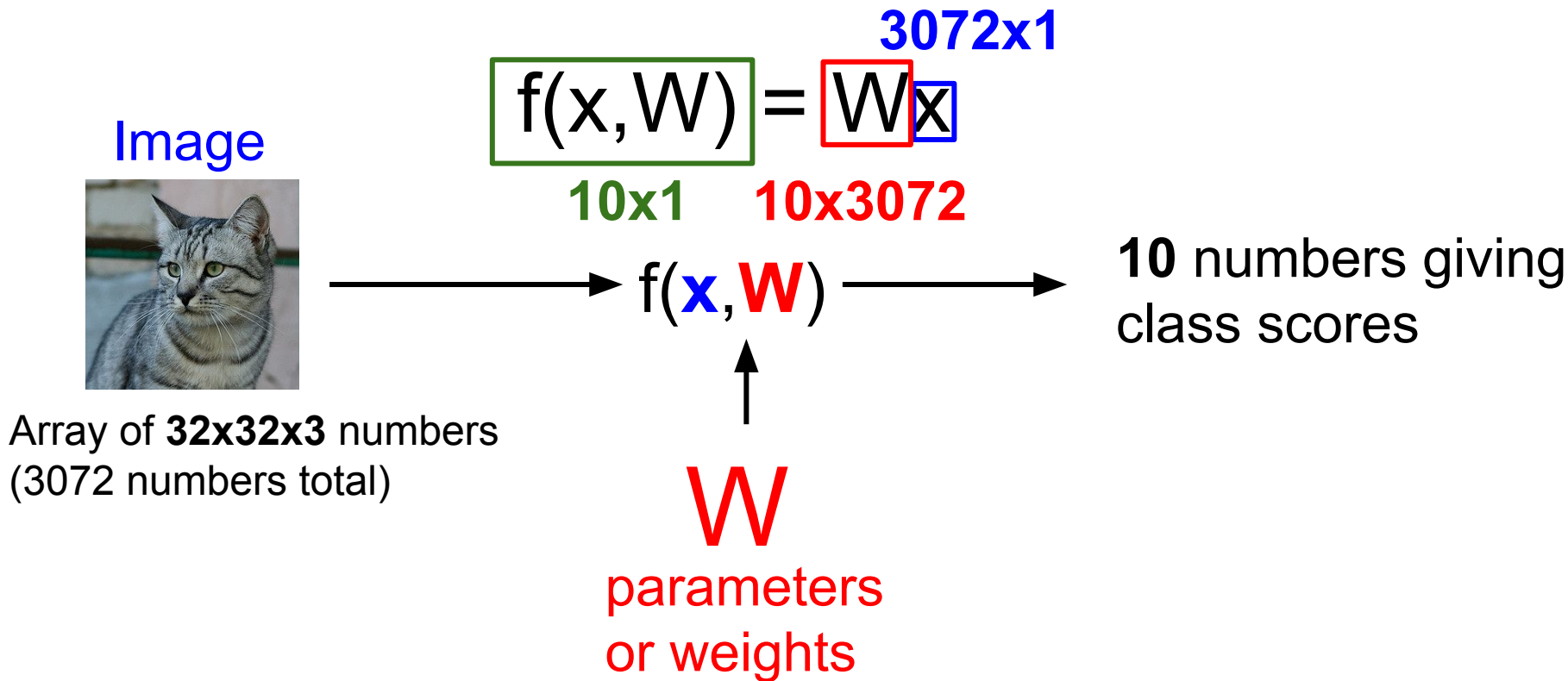
$f(\mathbf{x}, \mathbf{W})$

$\mathbf{W}$

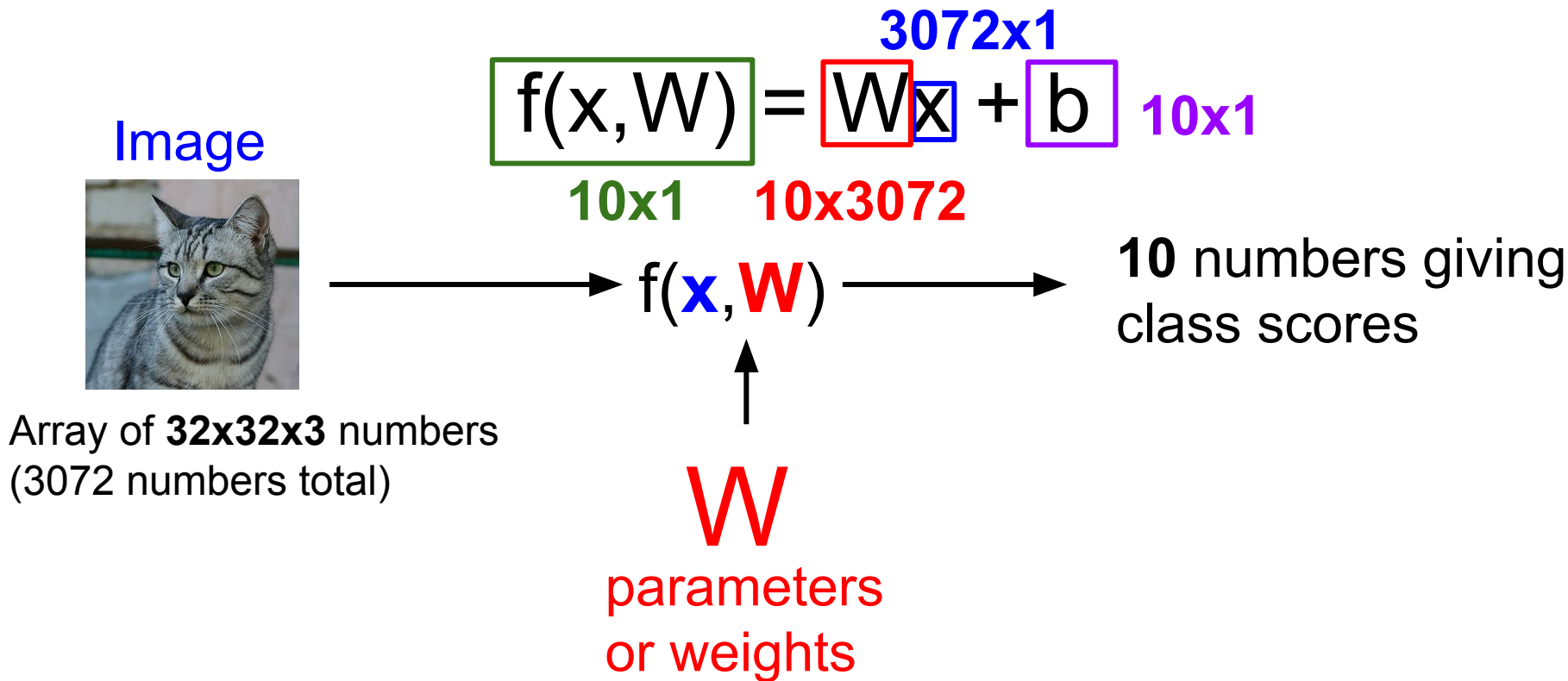
parameters
or weights

10 numbers giving
class scores

Parametric Approach: Linear Classifier

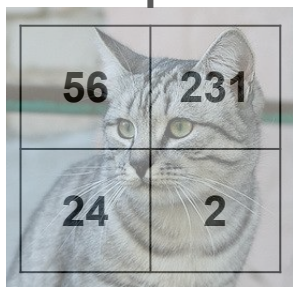


Parametric Approach: Linear Classifier

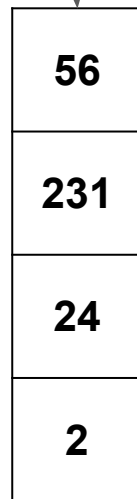


Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

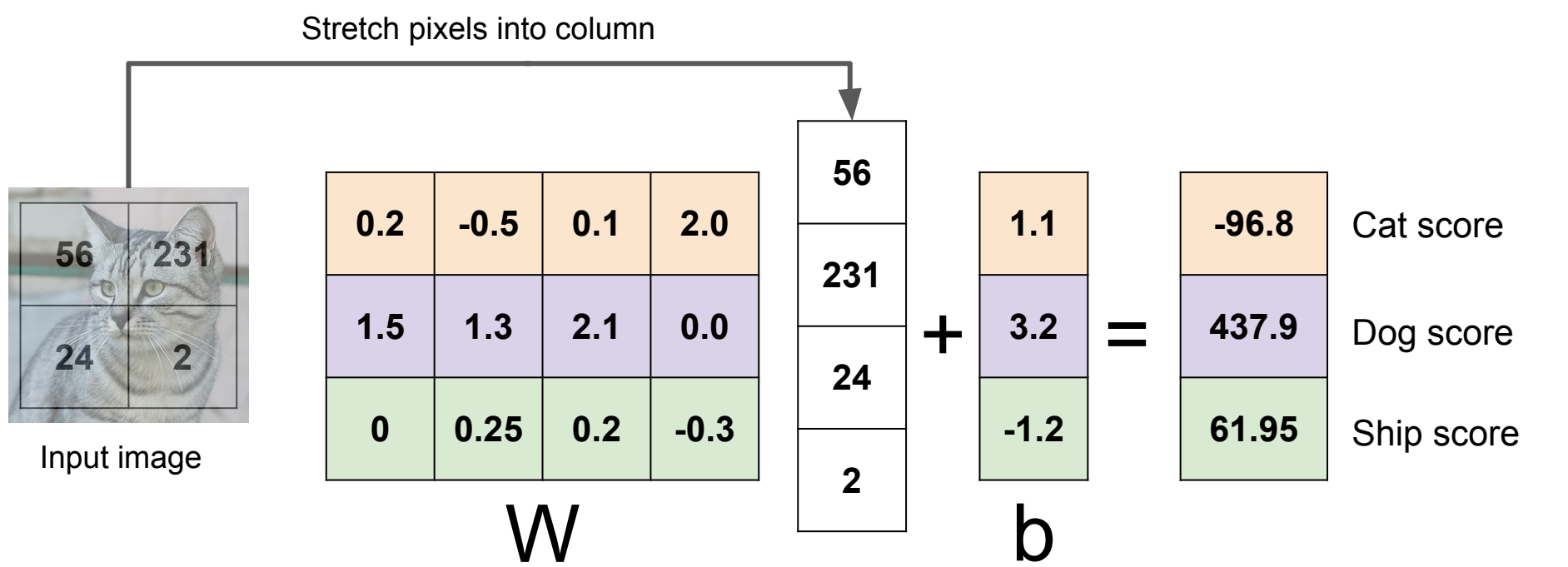
Stretch pixels into column



Input image



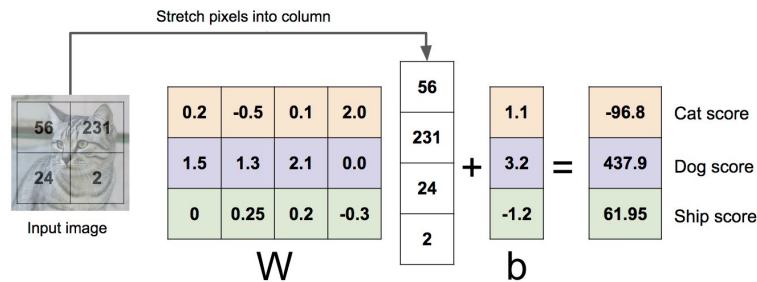
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



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

Algebraic Viewpoint

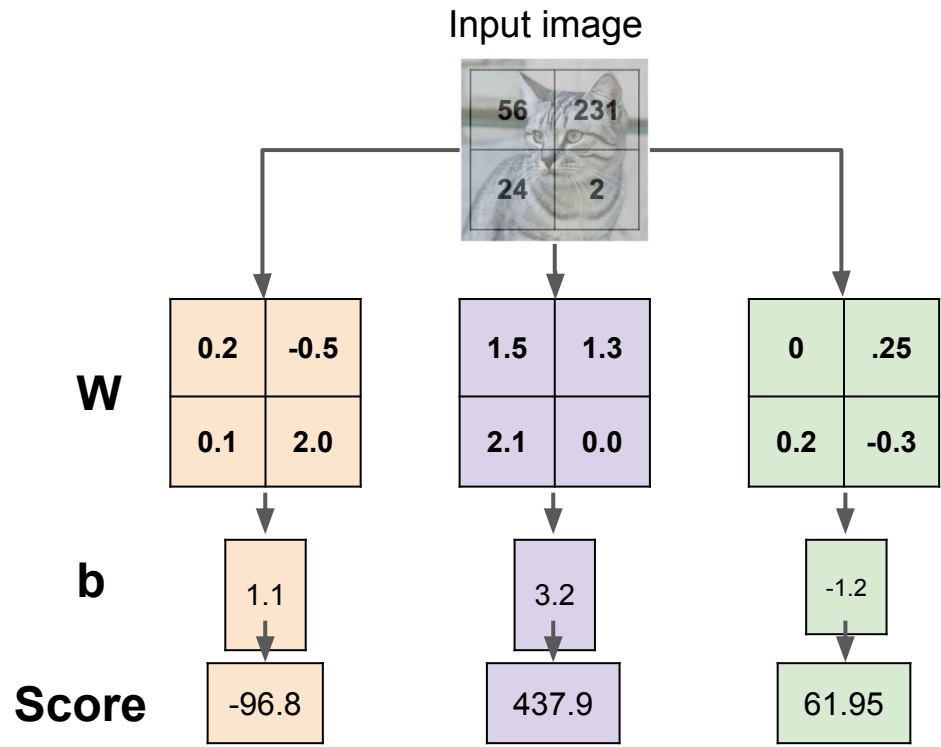
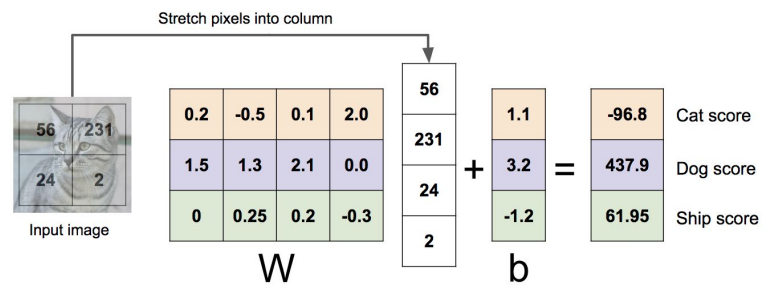
$$f(x, W) = Wx$$



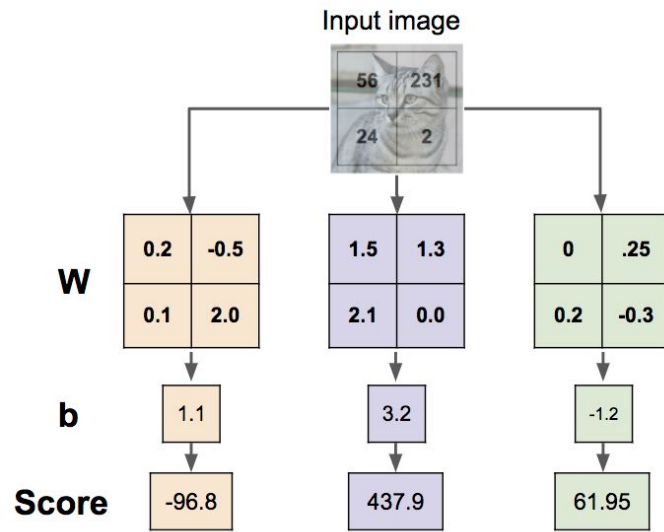
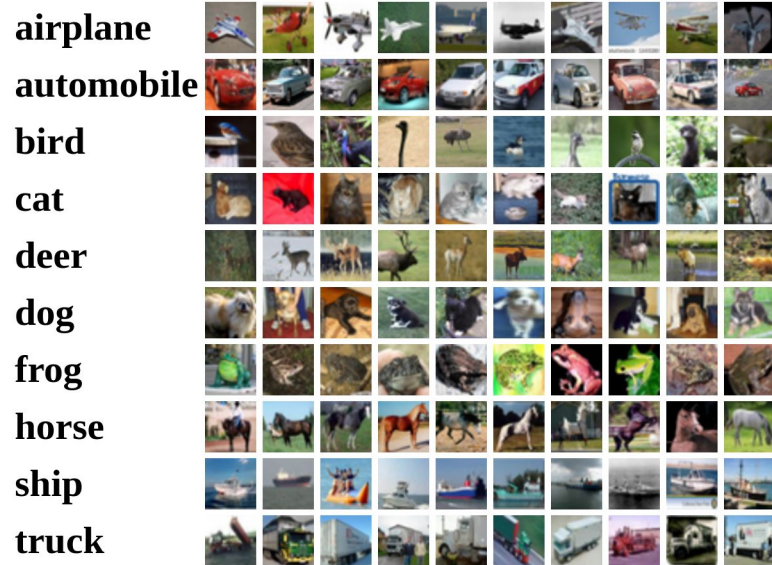
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Algebraic Viewpoint

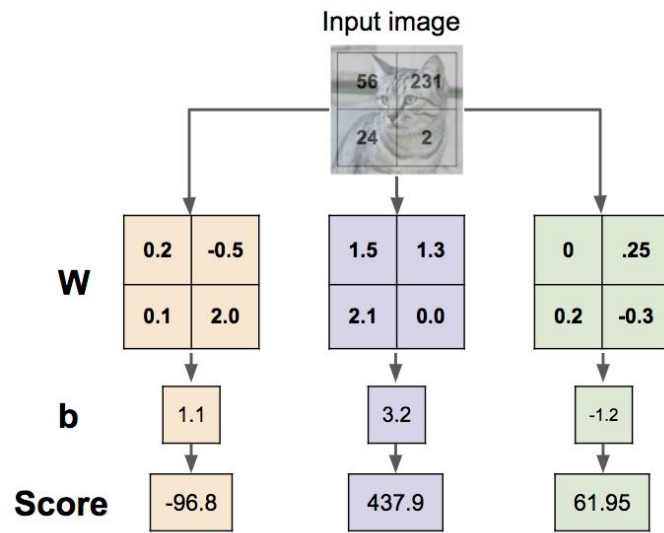
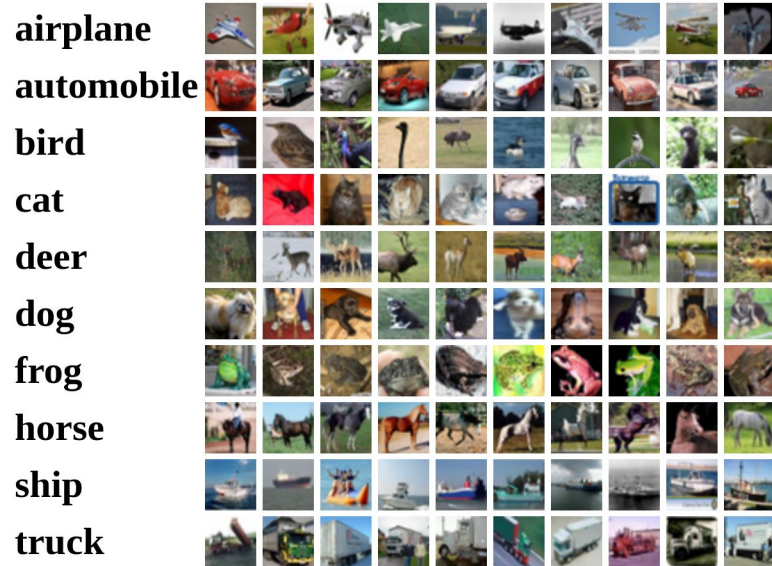
$$f(x,W) = Wx$$



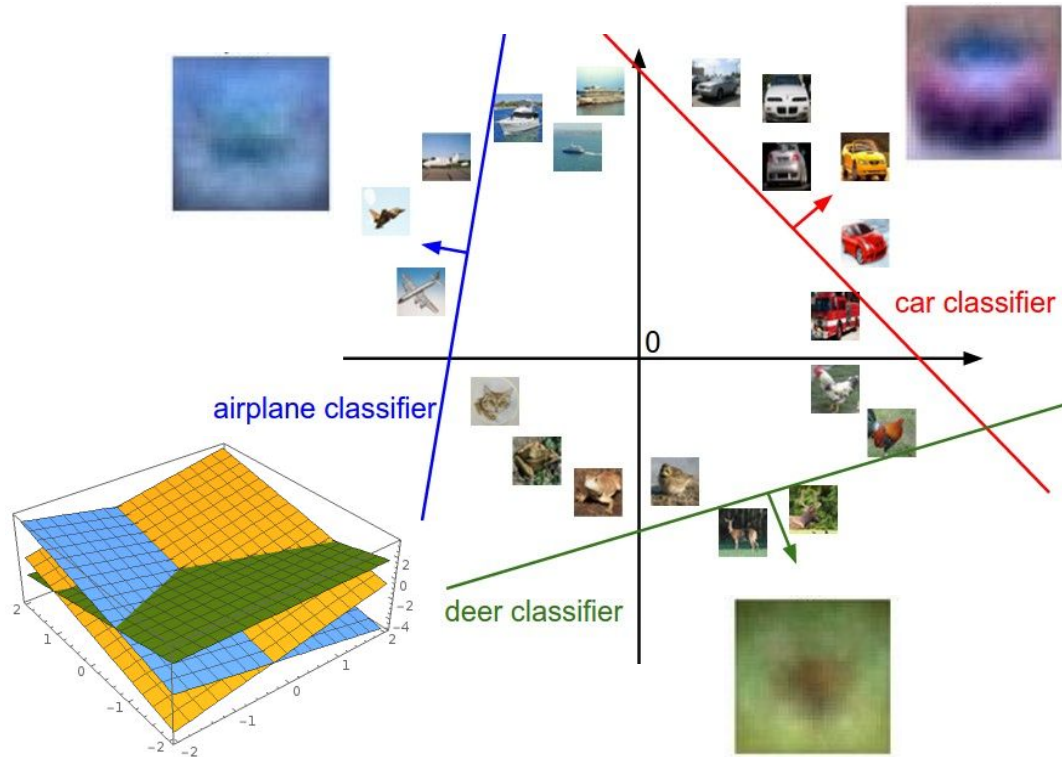
Interpreting a Linear Classifier



Interpreting a Linear Classifier: Visual Viewpoint



Interpreting a Linear Classifier: Geometric Viewpoint



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



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

Plot created using [Wolfram Cloud](#)

[Cat image](#) by [Nikita](#) is licensed under [CC-BY 2.0](#)

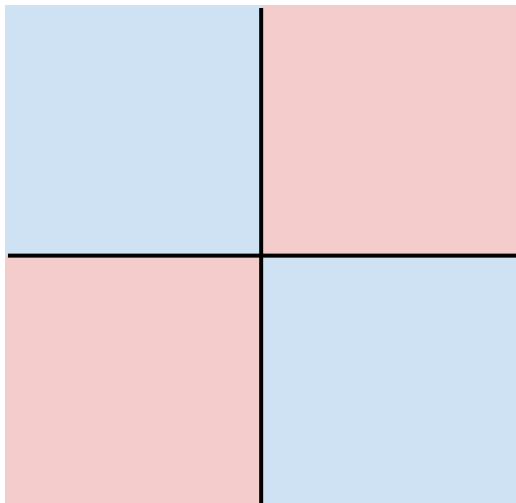
Hard cases for a linear classifier

Class 1:

First and third quadrants

Class 2:

Second and fourth quadrants

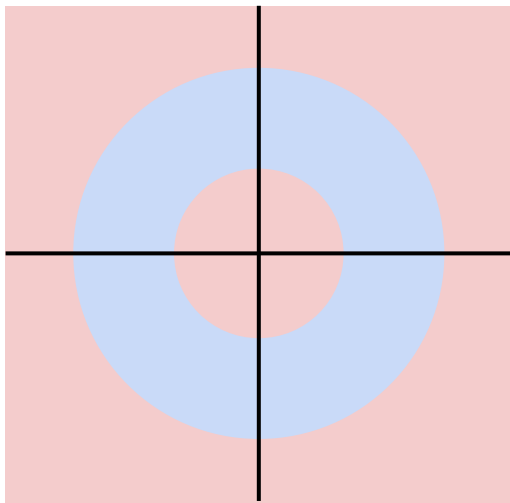


Class 1:

$1 \leq \text{L2 norm} \leq 2$

Class 2:

Everything else

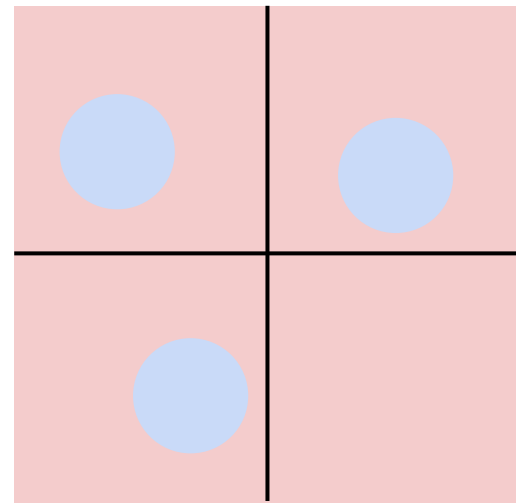


Class 1:

Three modes

Class 2:

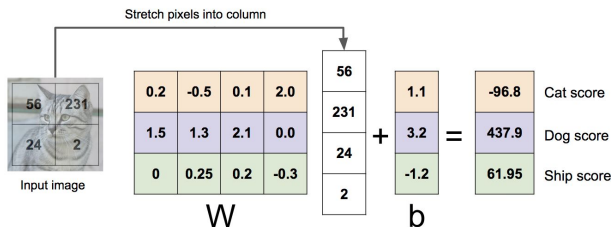
Everything else



Linear Classifier: Three Viewpoints

Algebraic Viewpoint

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



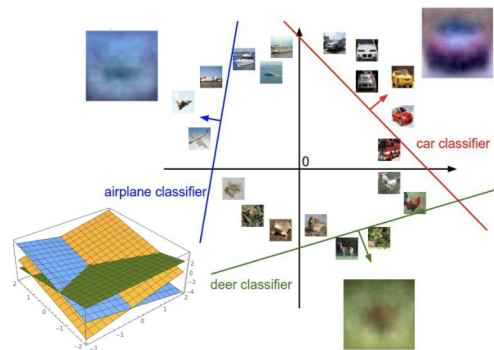
Visual Viewpoint

One template
per class



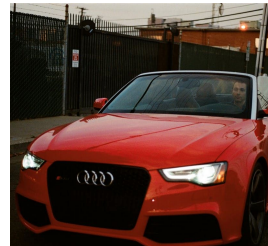
Geometric Viewpoint

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

[Cat image](#) by [Nikita](#) is licensed under [CC-BY 2.0](#)
[Car image](#) is [CC0 1.0](#) public domain
[Frog image](#) is in the public domain

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

Coming up:

- Loss function
- Optimization
- ConvNets!

(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)