

Lecture 2: Image Classification pipeline

Administrative: Piazza

커뮤니케이션 수단이라네.. ㅋㅋ

For questions about midterm, poster session, projects,
use Piazza instead of staff list!

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

Administrative: Assignment 1

Out tonight, due 4/18 11:59pm

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

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

클래스 구글 클라우드 쿠폰도 나눠주네

Google Cloud Tutorial

For the class project and assignments, we offer an option to use Google Compute Engine for developing and testing your implementations. This tutorial lists the necessary steps of working on the assignments using Google Cloud. For each assignment, we will provide you with an image containing the starter code and all dependencies that you need to complete the assignment. This tutorial goes through how to set up your own Google Compute Engine (GCE)

<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

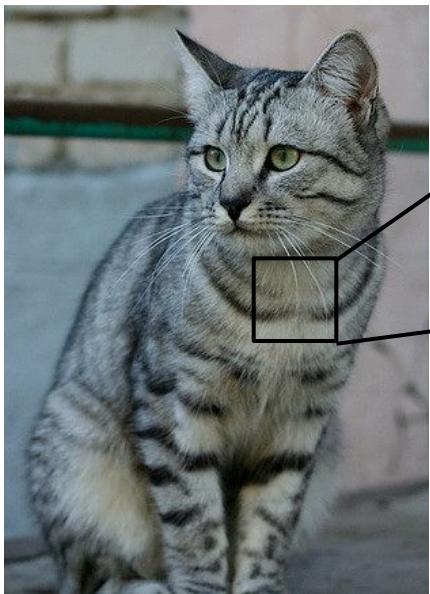
(assume given set of discrete labels)
{dog, cat, truck, plane, ...}



cat

컴퓨터가 보고 어떤 카테고리에 속할지 정해야한다.

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 90 105 128 105 87 96 95 99 115 112 106 103 99 85]
[99 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]
[115 114 109 123 150 148 131 118 113 109 100 92 74 65 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 81 128 138 135 105 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 88 82 120 124 104 76 48 45 66 88 101 102 109]
[157 170 157 128 93 86 114 132 112 97 69 55 70 62 99 94]
[130 128 134 161 139 100 109 118 121 134 114 87 65 53 69 86]
[128 112 96 117 150 144 128 115 104 107 102 93 87 81 72 79]
[123 107 96 86 83 112 153 149 122 109 104 75 80 107 112 99]
[122 121 102 88 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]]

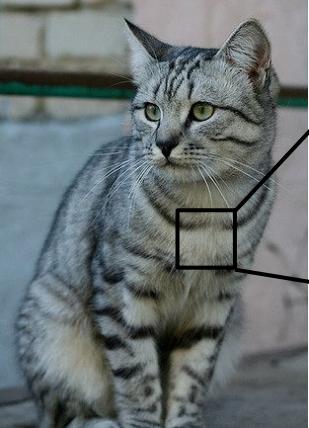
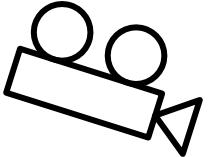
What the computer sees

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

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

가로 세로 픽셀 * 레드 그린 블루 RGB!!!

Challenges: Viewpoint variation



```
[1185 112 188 111 184 99 186 99 96 183 112 119 184 97 93 87]  
[ 91 98 182 106 104 79 98 103 99 105 123 136 118 105 94 85]  
[ 76 85 98 105 128 105 87 96 95 99 115 112 106 103 99 85]  
[ 99 81 98 105 128 105 127 108 98 99 115 112 106 103 99 85]  
[104 91 86 84 69 91 68 85 101 102 109 98 75 84 96 95]  
[114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 94 91]  
[133 137 147 103 65 81 80 65 52 54 74 84 102 93 85 82]  
[128 137 144 148 105 95 86 78 62 65 63 63 68 73 86 101]  
[102 125 131 147 133 127 116 131 111 98 89 75 61 64 72 84]  
[127 125 131 147 133 127 116 131 111 98 89 75 61 64 72 84]  
[115 111 189 123 150 148 131 118 113 109 108 92 74 65 72 78]  
[ 89 93 98 97 108 147 131 118 113 114 113 108 106 95 77 80]  
[ 63 77 86 81 77 79 182 123 137 115 111 125 125 130 115 87]  
[ 62 85 88 89 73 62 81 128 138 135 105 81 98 118 118]  
[ 63 65 75 88 89 73 62 81 128 138 135 105 81 98 118 118]  
[ 87 65 71 87 100 95 69 45 76 138 126 107 92 94 105 112]  
[118 97 82 86 117 123 116 66 41 51 95 93 89 89 95 102 107]  
[164 140 112 88 80 102 126 184 78 48 66 66 66 101 102 106]  
[137 98 104 113 92 86 104 121 112 109 95 79 84 84 84]  
[138 128 134 161 139 180 109 118 121 134 114 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 83 112 153 149 122 189 104 75 88 107 112 99]  
[122 121 102 80 82 86 94 117 145 148 153 105 58 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!

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

Challenges: Illumination



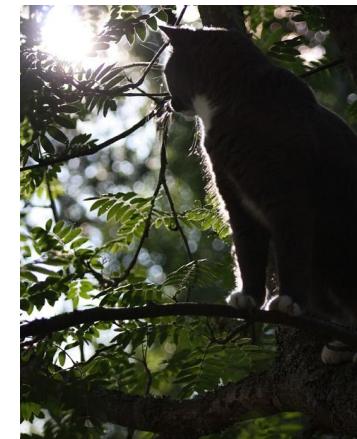
This image is CC0 1.0 public domain



This image is CC0 1.0 public domain



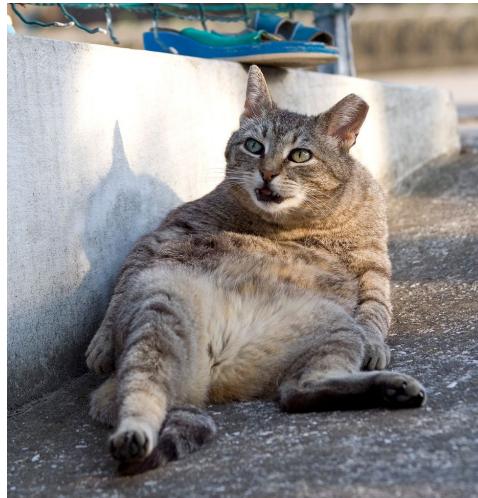
This image is CC0 1.0 public domain



This image is CC0 1.0 public domain

고양이는 밝기에 상관없이 고양이다

Challenges: Deformation



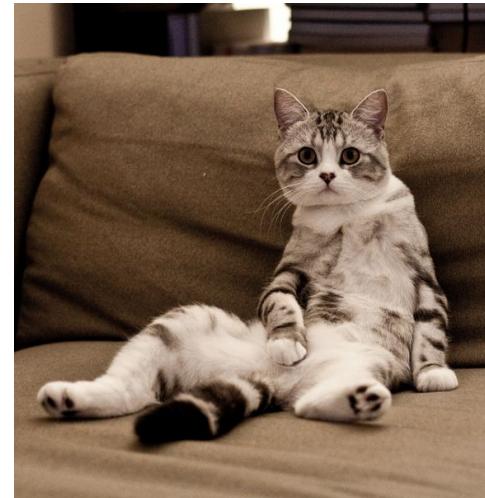
[This image by Umberto Salvagnin](#)
is licensed under CC-BY 2.0



[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

형태에 관계 없이 고양이어야 한다.

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](#)

사람은 바로 알아본다 고양이 인것을

Challenges: Background Clutter



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



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

백그라운드가 비슷하게 생겼을 수도 있다.

Challenges: Intraclass variation



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

이런 다양성에도 고양이를 알아 볼 수 있어야겠다

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

이미지에서 엣지를 검사한다.
명시적인 규칙 집합을 써내려간다

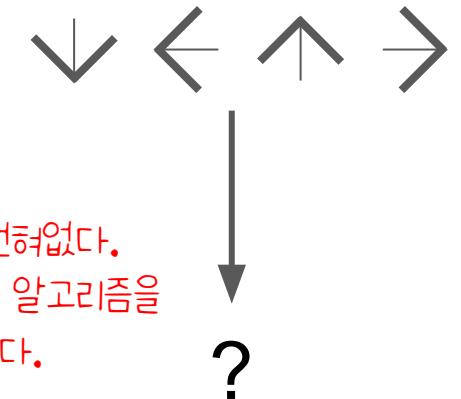


Find edges



Find corners

다른 객체를 인식해야 한다면
트럭
개 별도로 만들어야 한다
처음부터



확장성이 전혀 없다.
확장성있는 알고리즘을
만들어야 한다.

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

Data-Driven Approach

고양 물고기 각각 정의와 규칙을 정해주는것 대신
인터넷에서 고양이 비행기 사슴 데이터를 수집 / 구글 서치에서

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
```



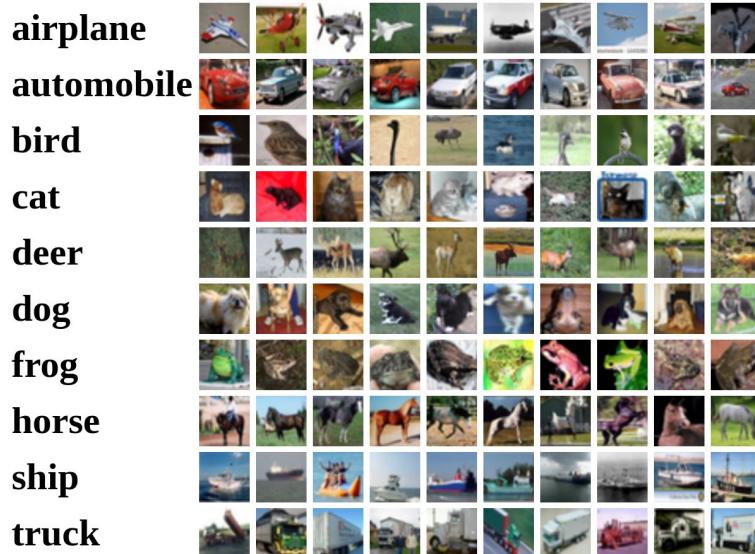
Predict the label
of the most similar
training image

Example Dataset: CIFAR10

10 classes

50,000 training images

10,000 testing images



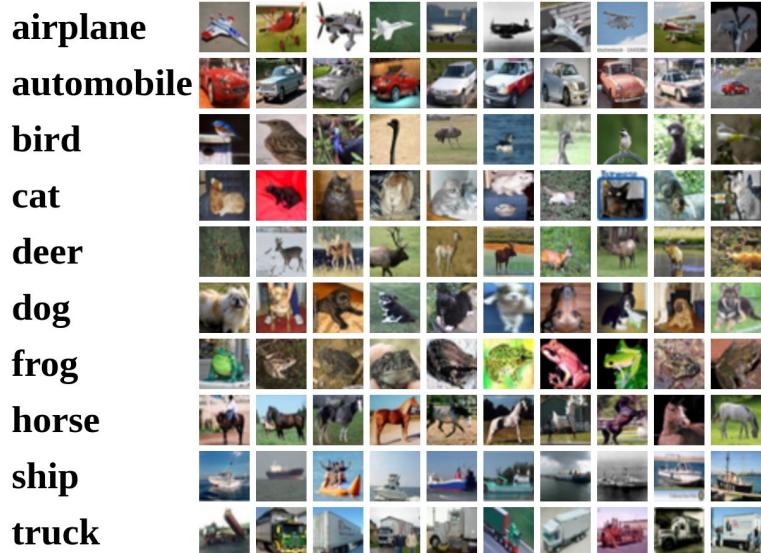
Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Technical Report, 2009.

Example Dataset: CIFAR10

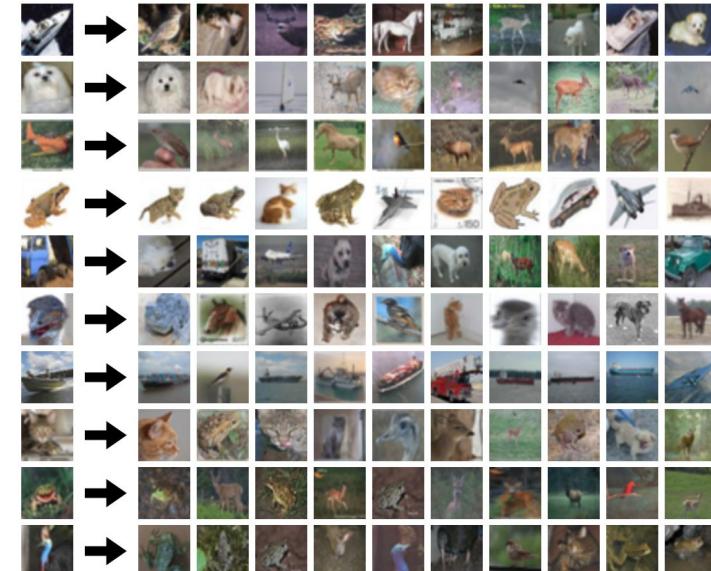
10 classes

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			
56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

training image			
10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

add
→ 456

```

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

NNC는 매우 간단하다

```

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

Memorize 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

For each test image:
 Find closest train image
 Predict label of nearest image

테스트 함수에서는 이미지를 입력받고 L1 Distance
 로 비교

```

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

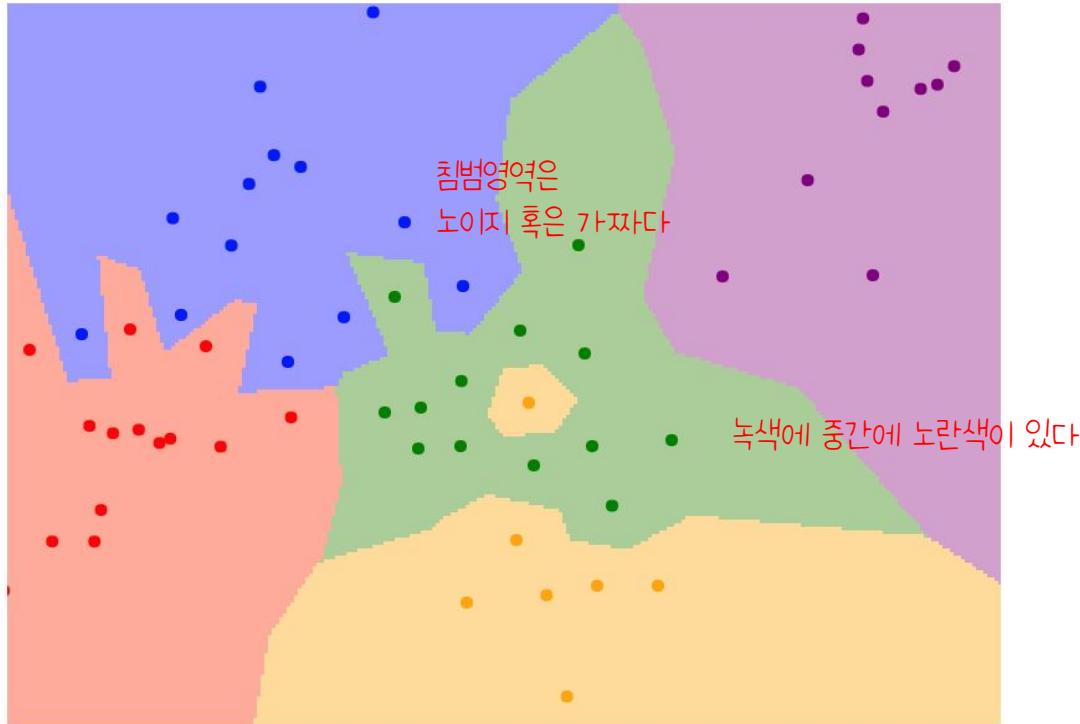
Q: With N examples,
how fast are training
and prediction?

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?

NN알고리즘

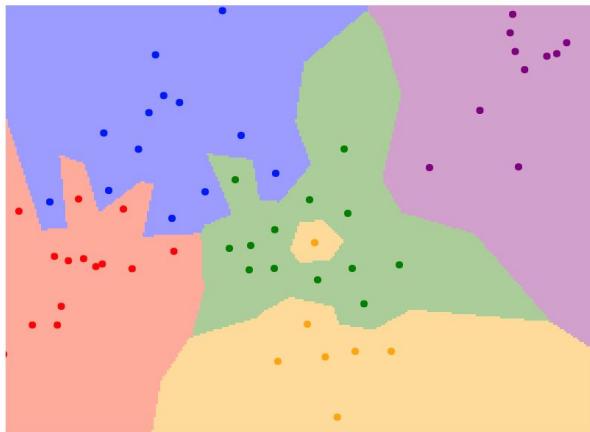


K-Nearest Neighbors

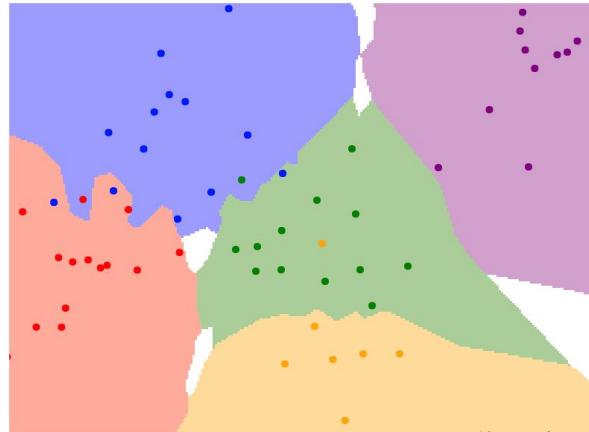
단순하게 가까운 이웃만 찾기보다는 조금 더 고급진 방법을 도입

거리별 가중치를 도입

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

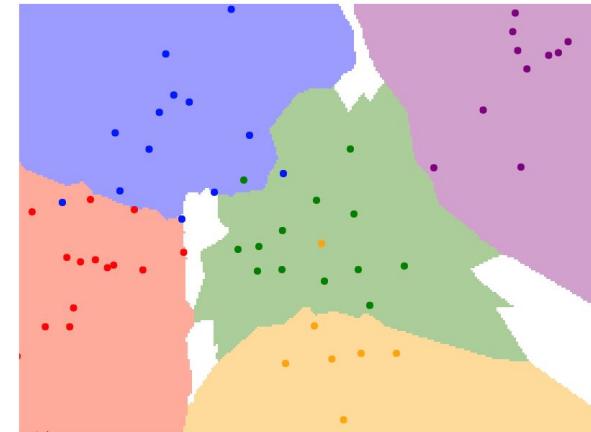


$K = 1$



$K = 3$

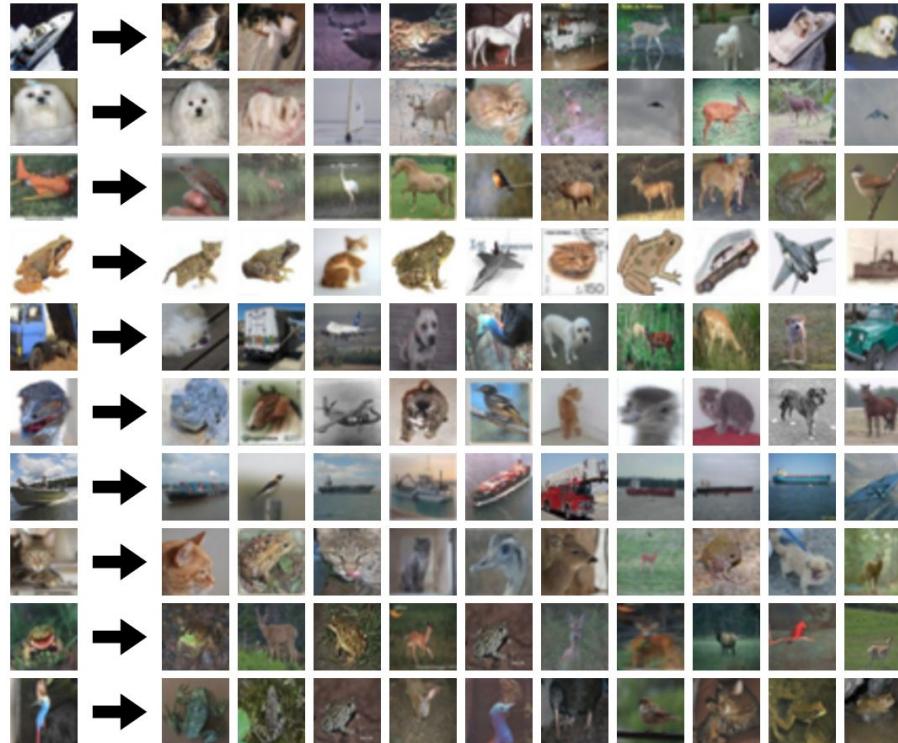
흰색은 결정할 수 없는 영역



$K = 5$

녹색안에 노란영역 사라짐

What does this look like?



What does this look like?

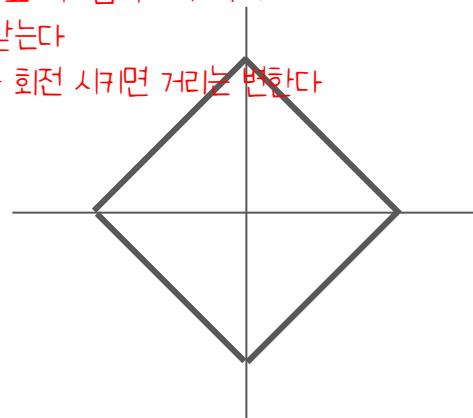


K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

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

L1은 어떤 좌표 시스템이냐에 따라
많은 영향을 받는다
기준 좌표계를 회전 시키면 거리는 변한다

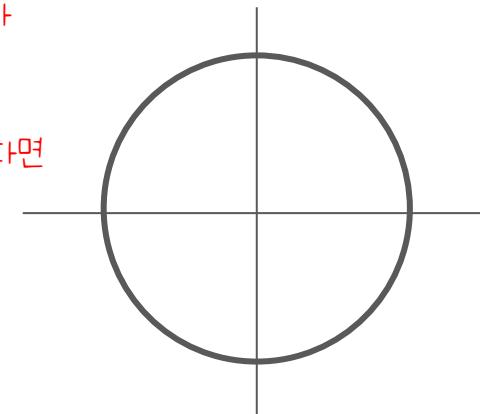


L2 (Euclidean) distance

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

모든 거리가 같아서
좌표계와 아무 연관이 없다

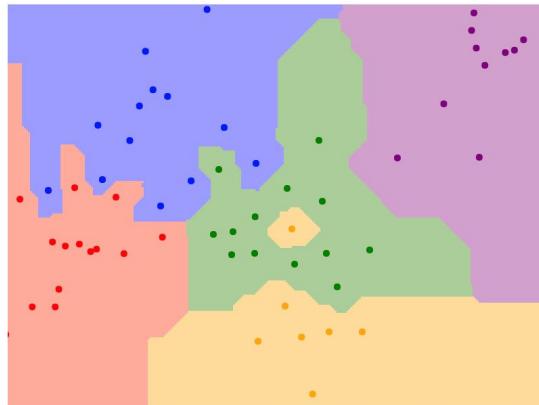
만약 벡터의 각각요소들이
개별적인 의미를 가지고 있다면
L1 이 더 잘어울림



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

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



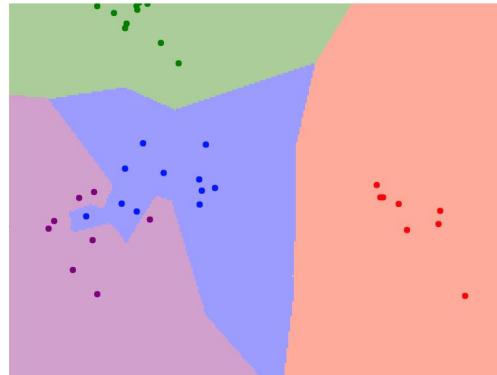
$K = 1$

K-Nearest Neighbors: Demo Time

K-Nearest Neighbors Demo

This interactive demo lets you explore the K-Nearest Neighbors algorithm for classification. Each point in the plane is colored with the class that would be assigned to it using the K-Nearest Neighbors algorithm. Points for which the K-Nearest Neighbor algorithm results in a tie are colored white.

You can move points around by clicking and dragging!



Metric

L1 L2

Num classes

2 3 4 5

Num Neighbors (K)

1 2 3 4 5 6 7

Num points

20 30 40 50 60

<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

한번도 보지 못한 데이터에서 잘 작동해야하는데 알고리즘의 성능을 측정 할 수 있어야한다
테스트데이터에서만 잘 돌아가는 모델을 만들면 안된다

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

테스트 셋은 최대한 분리한다 논문 마감 일주일 전에 쓴다함

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

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
--------	--------	--------	--------	--------	------

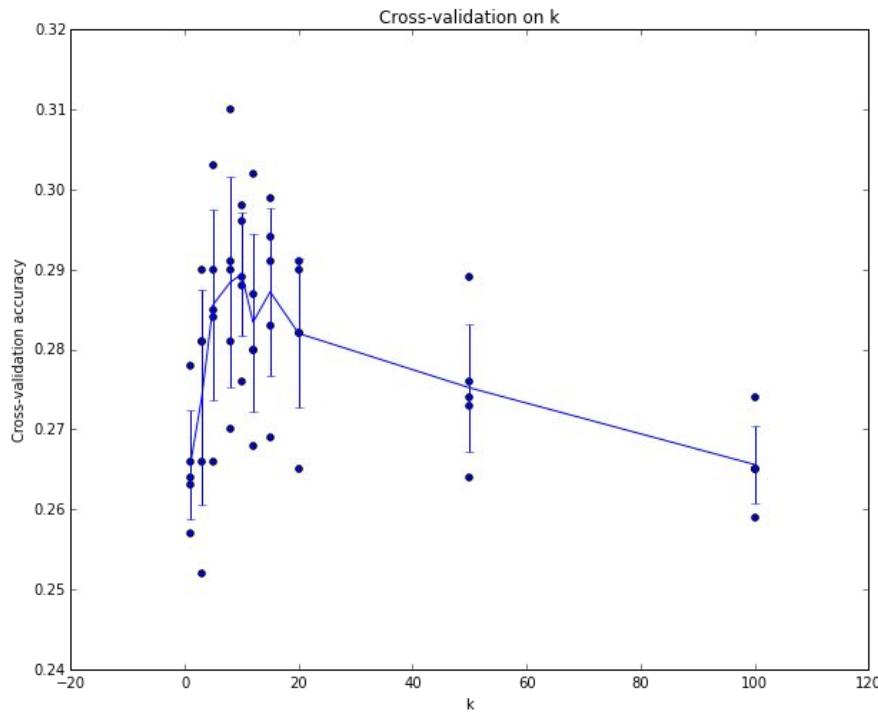
fold 1	fold 2	fold 3	fold 4	fold 5	test
--------	--------	--------	--------	--------	------

fold 1	fold 2	fold 3	fold 4	fold 5	test
--------	--------	--------	--------	--------	------

Useful for small datasets, but not used too frequently in deep learning

너무 많아서.,,, 딥러닝은 잘 안쓴다.

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 $k \sim 7$ works best
for this data)

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

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

이 이미지에는 KNN 잘 안 쓴다

오리지널과 오류 이미지들간에 동일한 L₂값으로 가진다

Original



Boxed



Shifted



Tinted



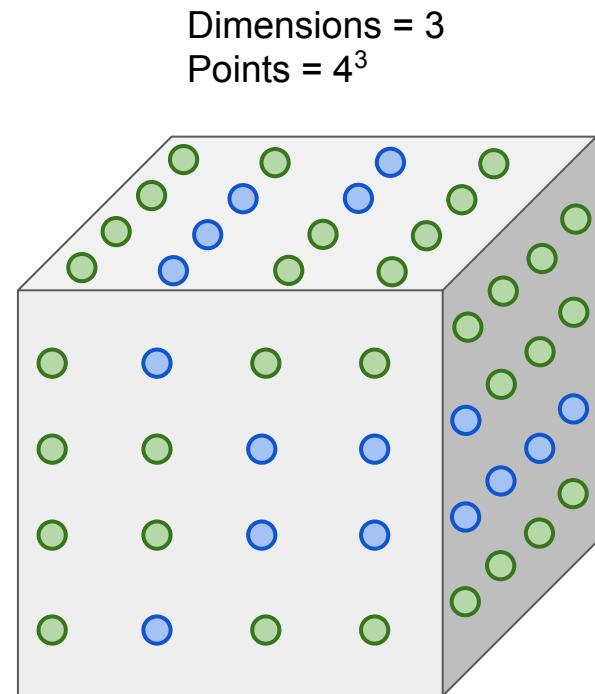
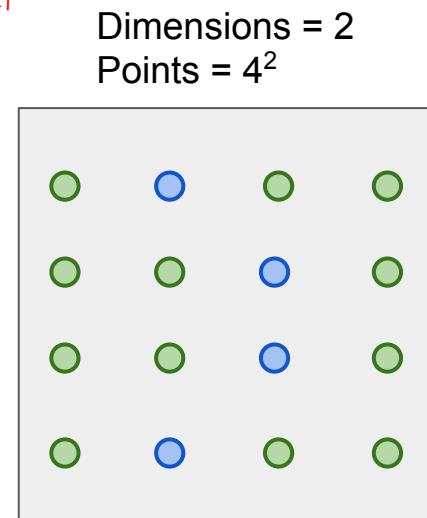
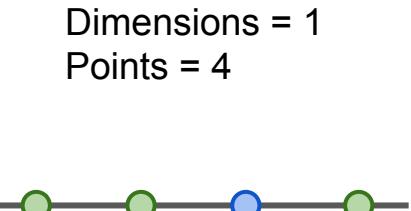
(all 3 images have same L₂ distance to the one on the left)

Original image is
CC0 public domain

k-Nearest Neighbor on images never used.

- Curse of dimensionality

각 점은 트레이닝 샘플이 속한 카테고리



KNN 알고리즘은 샘플의 manifolds를 가정하지 않는다

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

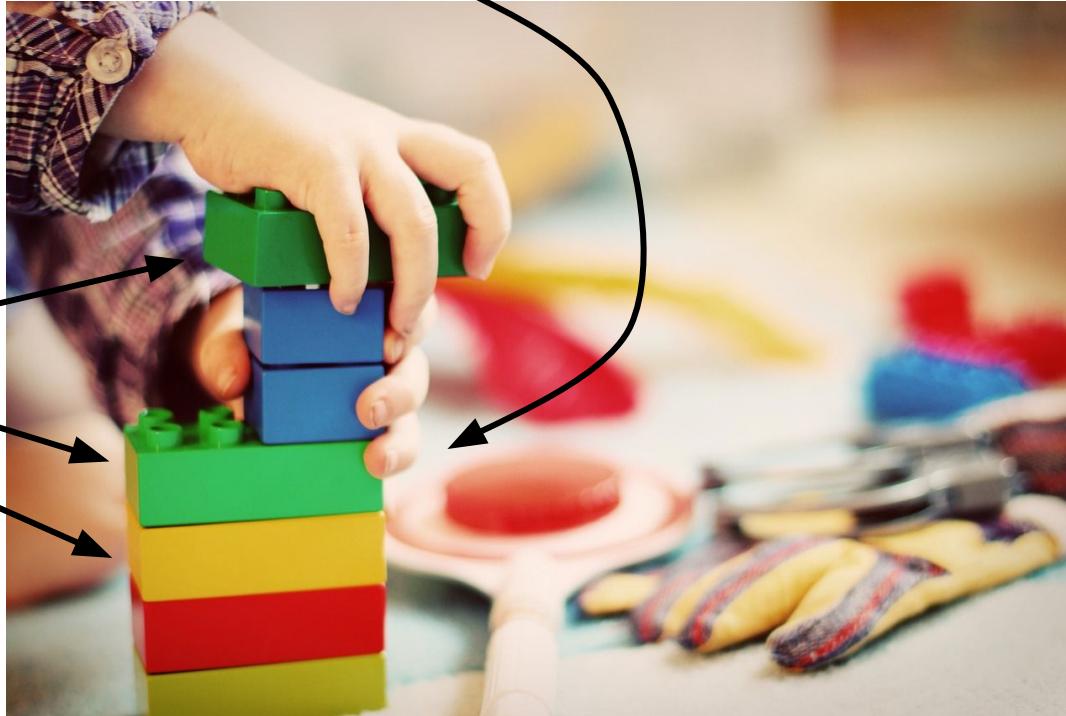
Distance metric and K are **hyperparameters**

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.

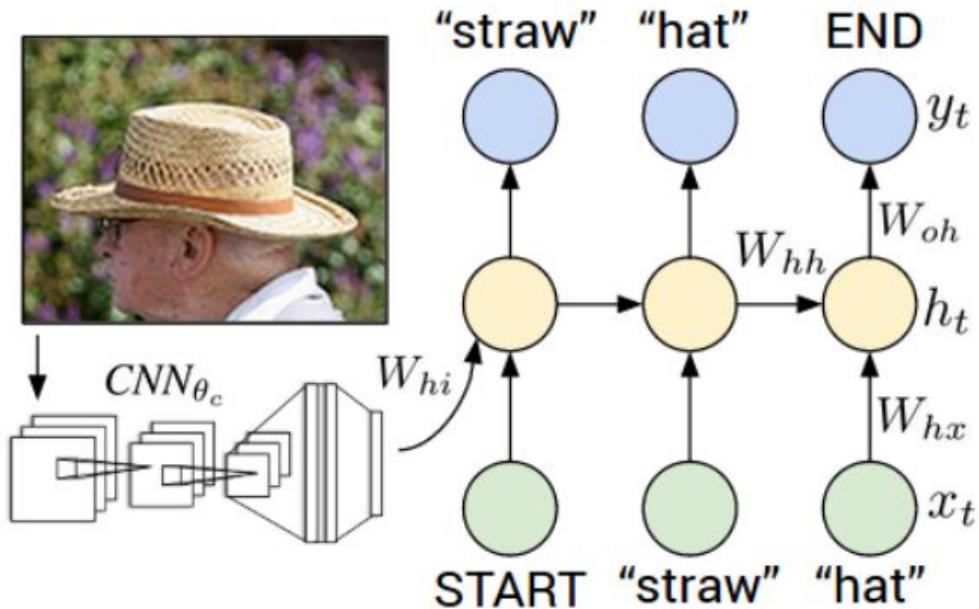
Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figures copyright IEEE, 2015. Reproduced for educational purposes.

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



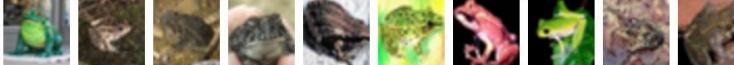
deer



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)

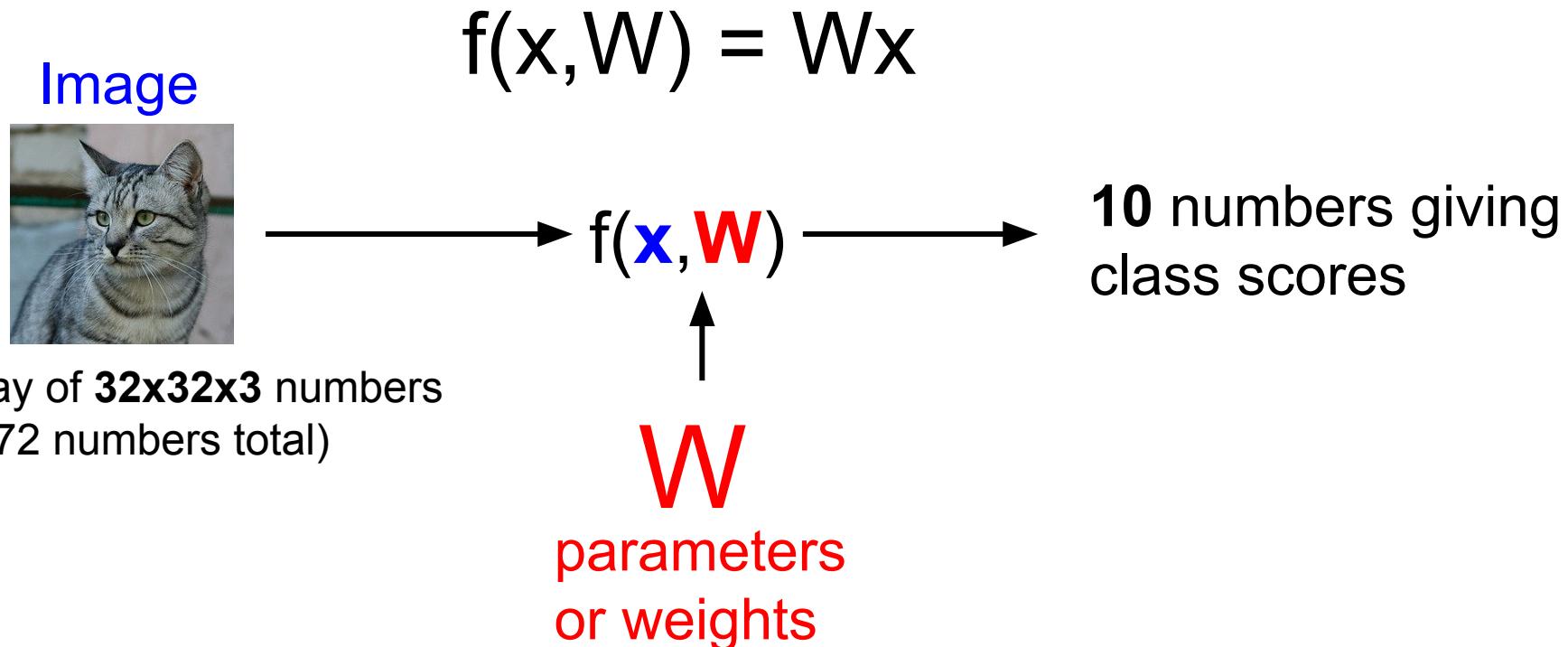
$$\xrightarrow{f(x, W)}$$

W
parameters
or weights

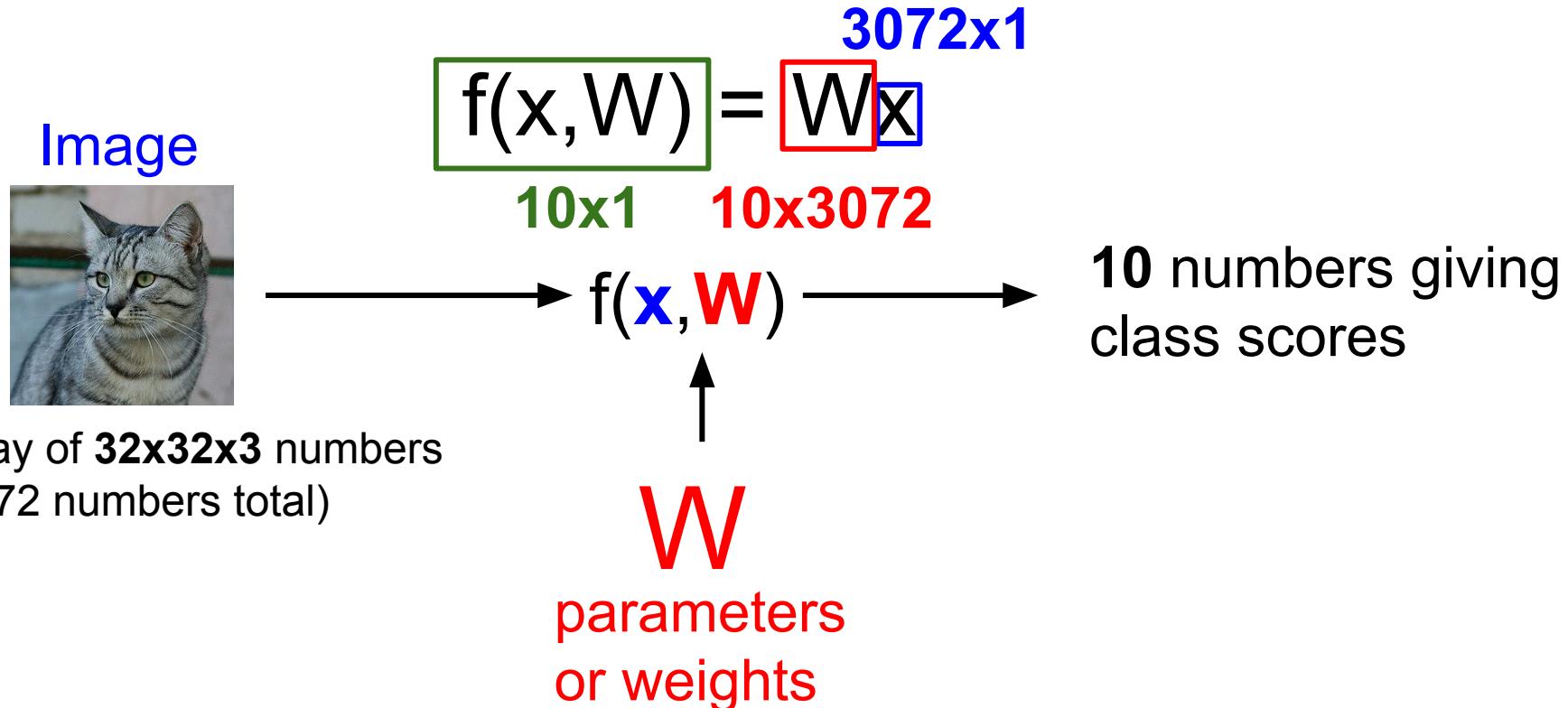
10 numbers giving
class scores

파라미터 모델에 두개의 요소가 있다.
입력 이미지 – X
W – 세타 가중치

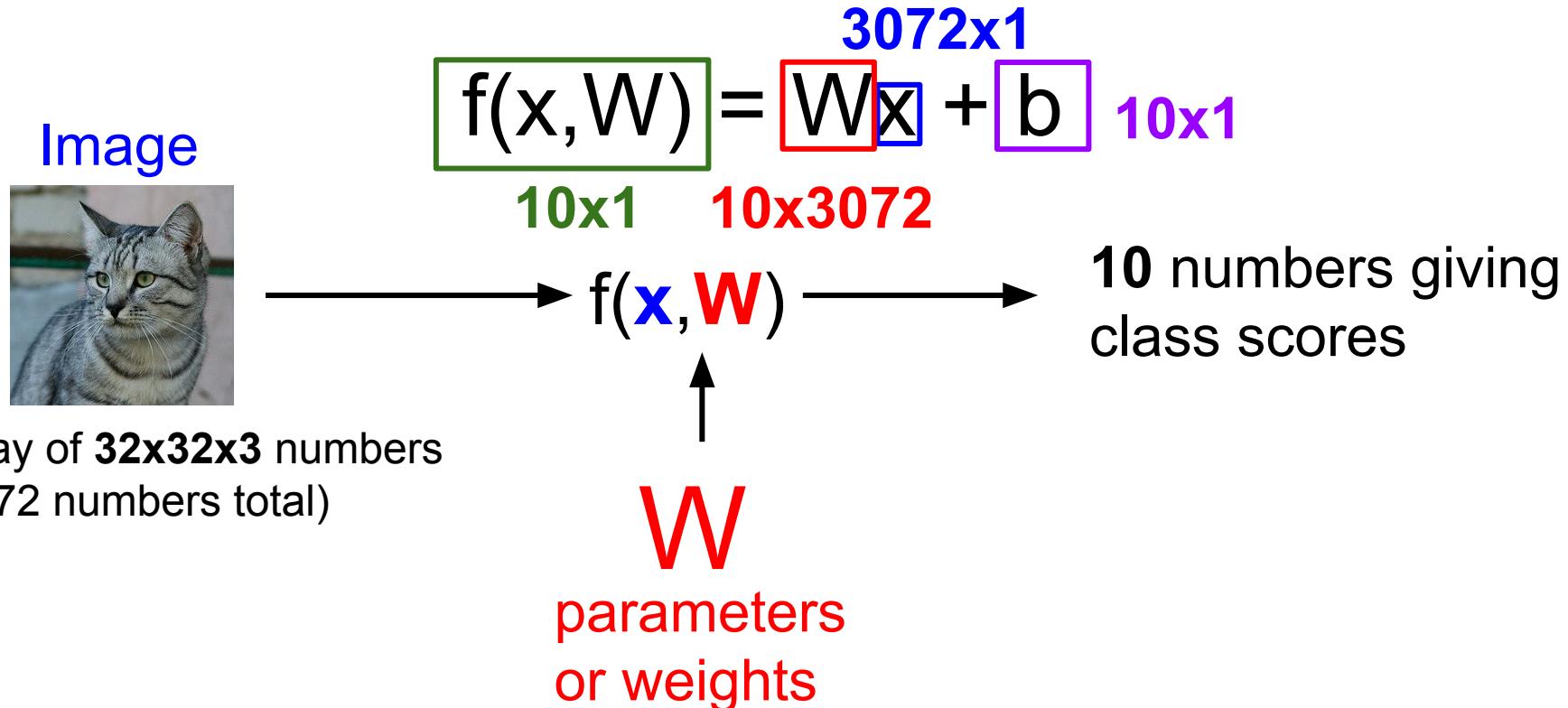
Parametric Approach: Linear Classifier



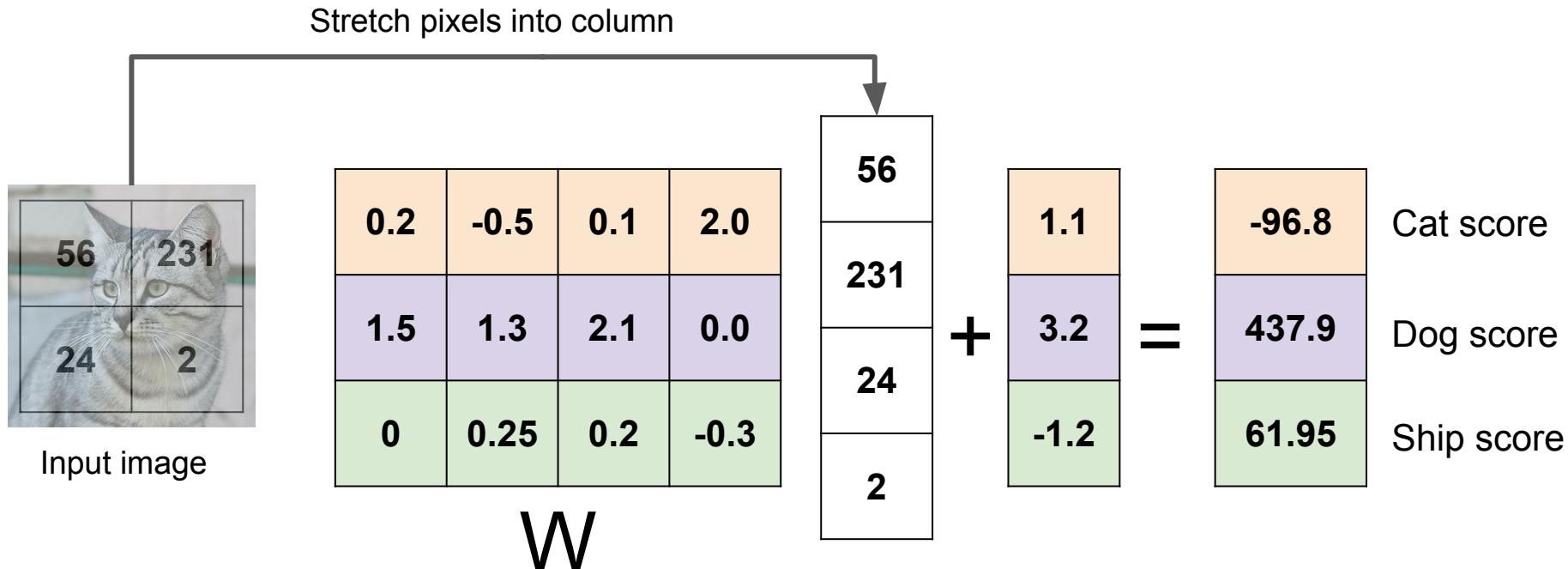
Parametric Approach: Linear Classifier



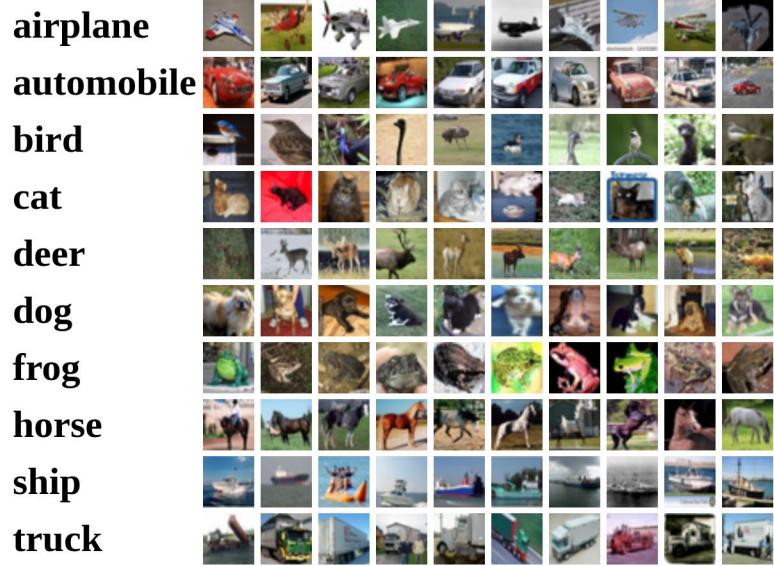
Parametric Approach: Linear Classifier



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



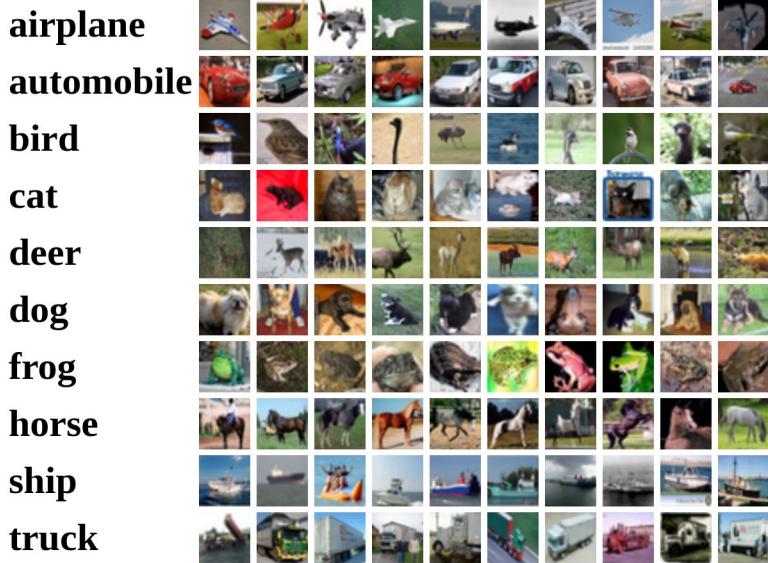
Interpreting a Linear Classifier



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

What is this thing doing?

Interpreting a Linear Classifier



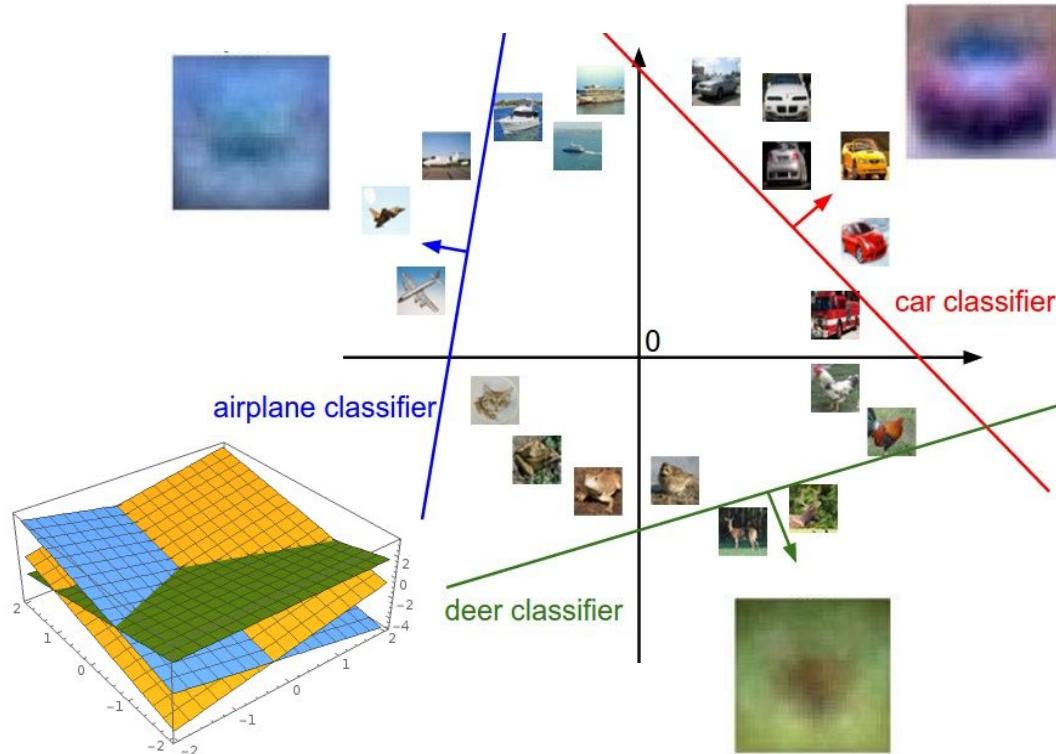
$$f(x, W) = Wx + b$$

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



Interpreting a Linear Classifier

이미지를 고차원의 한곳으로 몰아주는 방법이 있다.



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



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

Cat image by Nikita is licensed under CC-BY 2.0

Plot created using [Wolfram Cloud](#)

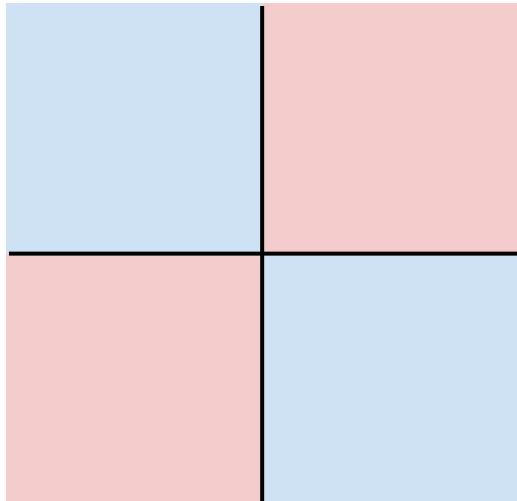
Hard cases for a linear classifier

Class 1:

number of pixels > 0 odd

Class 2:

number of pixels > 0 even

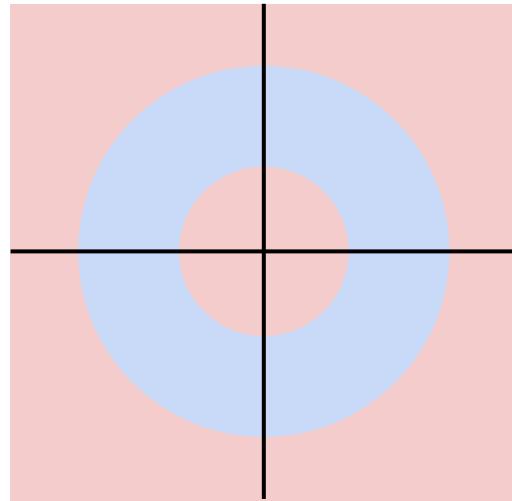


Class 1:

$1 \leq L_2 \text{ norm} \leq 2$

Class 2:

Everything else

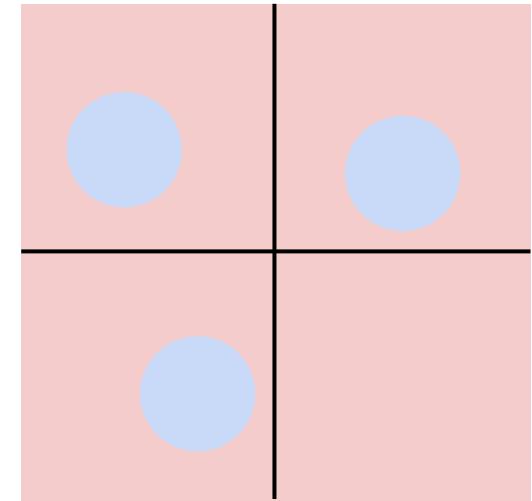


Class 1:

Three modes

Class 2:

Everything else



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)