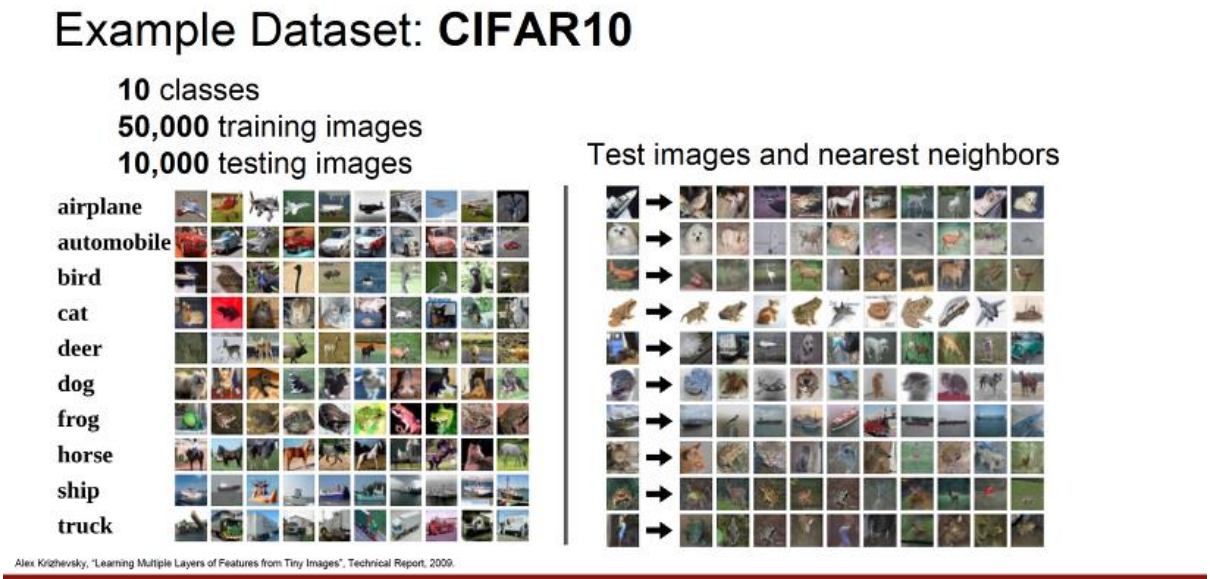
Lec 2. Image Classification pipeline



해당 강의에서 이용한 dataset: CIFAR10

Image Classification: A core task in Computer Vision

간단한 작동 방식: input image -> 고정된 category label 중 하나를 할당
인간은 시각을 통해 사고할 수 있기에 이미지를 구분하는 것은 쉬운 문제이지만 기계에게는 결코 쉬운 문제가 아님.

Problem
- Semantic Gap: 이미지 내의 개체의 의미와 컴퓨터가 실제로 인식하는 픽셀 값 사이의 큰 차이(에서 발생하는 문제). 예를 들어, 머신은 고양이 이미지의 전체적인 모습을 인식하지 못하고 [0, 255] 사이의 숫자 격자들로 이미지를 인식함.

아래는 Semantic Gap 으로 인해 발생하는 Challenges

1. Viewpoint variation: 다른 시각에서의 동일한 개체를 찍은 사진이더라도 다른 이미지라고 인식함. 즉, 동일한 것을 담은 이미지라도 픽셀값이 다르면 같은 이미지라는 것을 머신은 인식하지 못함. 예를 들어, 고양이 사진을 찍었을 때, 어떠한 변동도 없는 같은 고양이를 다른 시점에서 찍었을 때 픽셀값이 완전히 바뀌기 때문에 머신은 동일한 고양이임을 인지할 수 없음.
2. Illumination: 밝기, 빛이 개체를 비추는 위치에 따라 이미지가 달라짐.
3. Deformation: 같은 class 여도 개체에 따라서 이미지가 달라짐. 여러 종류의 고양이, 여러 모습을 한 고양이들이 있을 테고, 다양한 포즈를 취하는 고양이들이 있을 텐데, 이들을 인식할 수 있는 알고리즘을 만들어내야 함.
4. Occlusion: 인식하려는 타깃이 다른 어떠한 것에 의해 일부가 가려지거나 일부만 노출될 수도 있음. 고양이의 일부만 보이더라도 인간은 쉽게 고양이임을 파악할 수 있는데, 알고리즘이 이를 커버할 수 있어야 함.
5. Background Clutter: 배경이 타깃과 잘 구분되지 않는 경우. 보호색처럼 고양이가 배경과 비슷한 색을 가지고 있더라도 이를 파악할 수 있게 해야함.

Interclass variation: class 내의 다양성.

가능한 알고리즘
 - 이미지의 테두리를 계산하여 corner, boundary 를 찾은 후 명시적인 규칙을 통해 해당 이미지가 나타내는 것을 알 수 있음.
 -> BUT, it's super bittle. 또, 다른 카테고리를 다룰 때마다 새롭게 위의 방식을 다시 실시해야함.
 - 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
아래는 Data-Driven Approach 의 함수 구성

```
def train(images, labels):

    # Machine Learning!

    return model

def predict(model, test_images):

    #Use model to predict labels

    return test_labels
```

First classifier: Nearest Neighbor

1. train: input images and labels, and then output a model (Memorize all data and labels)

```
def train(images, labels):

    # Machine Learning!

    return model
```

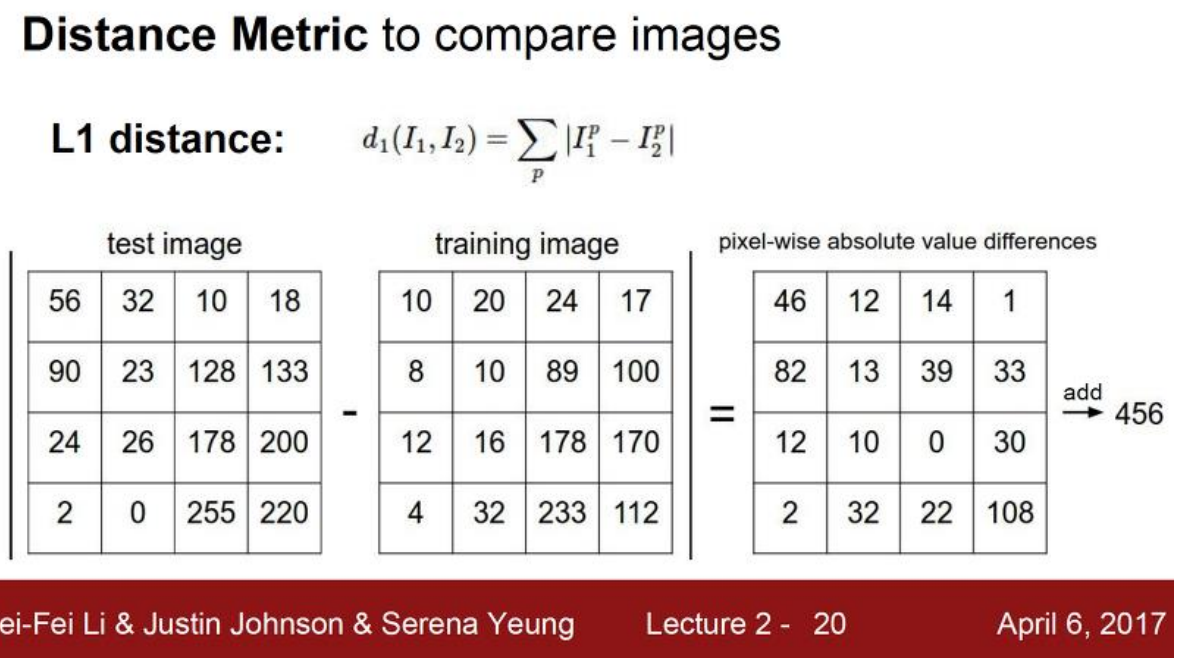
2.predict: input model and then make predictions for images (Predict the label of the most similar training image)

```
def predict(model, test_images):

    #Use model to predict labels

    return test_labels
```

Distance Metric to compare images



- L1 distance(=Manhattan distance): 각 이미지의 픽셀 간의 차이를 기준으로 함.
- 예측 방식: test data 를 입력 받으면 각 데이터 하나에 대한 모든 training data 와의 L1 distance 를 구하고 이 값이 가장 작은 사진을 정답 이미지로 선택할 것.

```
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 range(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
```

(해당 코드는 Numpy 가 제공하는 벡터화된 연산을 많이 이용했기 때문에 짧고 간결함)
- train: Memorize training data
- predict 안의 for loop: For each test image, find closest train image, predict label of nearest image
- N examples 를 가지고 할 때 빠르기는 Train 은 O(1), Predict 는 O(N)임.
- 이 방식은 좋은 방식이 아님. -> training 에서 느리더라도 predict 에서 빠른 classifier 를 원함

K-Nearest Neighbors: 데이터로부터 거리가 가까운 'k'개의 다른 데이터의 레이블을 참조하여 분류하는 알고리즘

What does this look like?

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 2 - 30

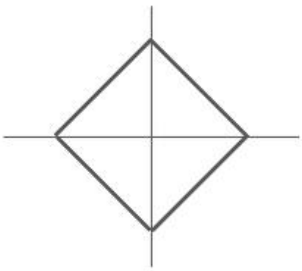
KNN 을 예시 데이터셋에 적용한 결과

K=10 으로 했고, 왼쪽은 test data, 오른쪽은 predict 결과. 초록색은 정답, 빨간색은 오답. 오답의 개수가 정답보다 많음을 알 수 있음. KNN 은 그리 좋은 알고리즘은 아님.

그럼에도 L2 distance 를 이용한 KNN 을 살펴보겠다.

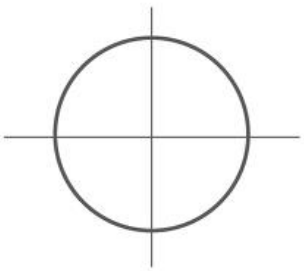
L1 (Manhattan) distance

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



L2 (Euclidean) distance

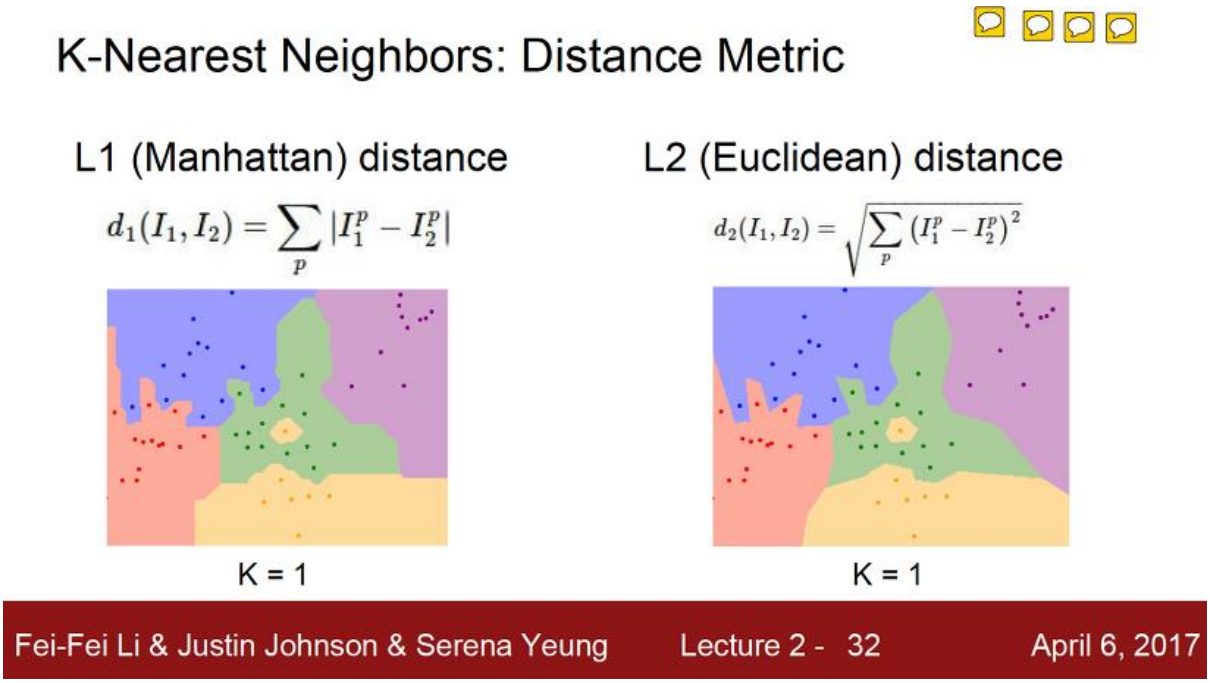
$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 2 - 31 April 6, 2017

- L2 distance(=Euclidean distance)
- L1 distance 는 L2 distance 보다 입력 데이터 모양에 더 큰 영향을 받음.

- input data 의 feature 가 중요한 의미를 가질 때에는 L1 distance, general 한 결과를 얻고 싶을 때에는 L2 를 사용하는 것을 권장. 그래도 둘 다 직접 시도해보고 더 나은 방식을 선택하는 것이 가장 좋음.



각각을 적용한 결과

- L1 의 경우, decision boundary 가 좌표축을 따라가는 모습을 보임. (좌표계에 종속적이기 때문)
- L2 의 경우, 좌표계에 큰 영향을 받지 않음.

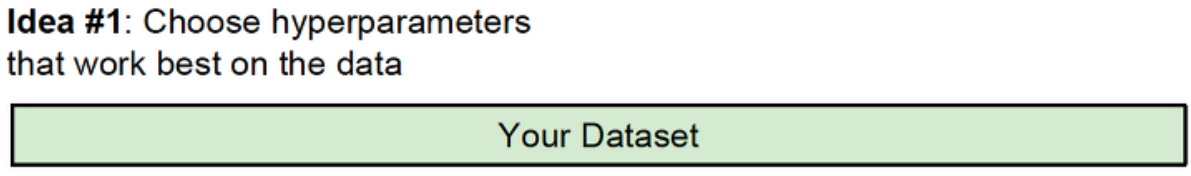
Hyperparameters

: 적합한 K 값과 metric 을 판단하는 문제

- problem-dependent 하므로 최적의 하이퍼파라미터는 문제마다 천차만별

하이퍼파라미터를 결정하는 방법

1. 가진 training dataset 에 가장 좋은 결과를 보여주는 파라미터를 선택



머신을 training data 에 overfitting 하게 만들면 오히려 새로운 데이터를 보게 했을 때 좋은 결과를 내놓지 않을 수도 있음. 정말 안 좋은 아이디어이므로 사용해서는 안됨.

2. 가진 data 를 train set, test set 으로 나누고 test set 에 대해서 가장 좋은 성능을 보여주는 파라미터 선택

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

idea #1 을 개선한 것처럼 보이지만 사실상 같은 결과를 불러일으킴. (test data 에 overfitting)

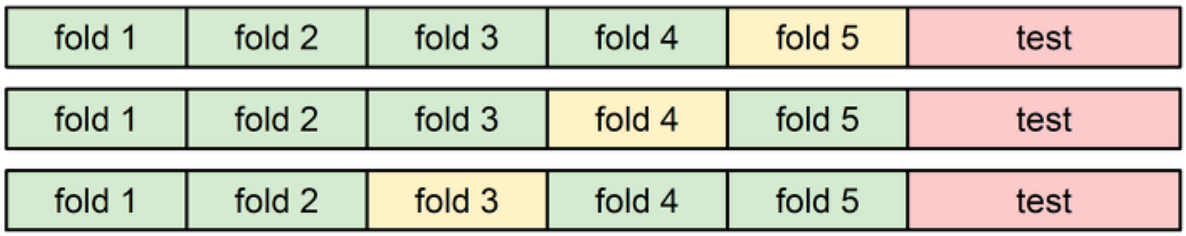
3. train, validation, test set 으로 나눔. validation set 으로 하이퍼파리미터를 결정하고 이를 test 에서 평가

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

완벽한 방법은 아니나 위의 두 방식보다 괜찮음. validation set 을 이용해 하이퍼파라미터를 조절하고 이 결과를 test set 에서 평가하는 것. test set 은 평가 시 딱 한 번만 사용해야함.
**training 과 validation 의 차이: training 에서는 알고리즘이 label 을 볼 수 있음. validation 에서는 label 에 바로 접근할 수 없고, 알고리즘이 잘 작동하는지 확인할 때 접근할 수 있음.*

4. training set 을 여러개의 fold 로 나누고 각 fold 가 돌아가면서 validation set 역할을 함. 이를 통해 나온 결과들의 평균을 이용해 좋은 파라미터를 찾아가는 방식

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



이 방식은 data set 의 크기가 작을 때 유용하게 쓰임. (적은 data 에도 불구하고 generalize 하게 적용될 수 있는 파라미터를 찾기 위한 방식) 따라서 딥러닝에는 그리 많이 쓰이지 않음.

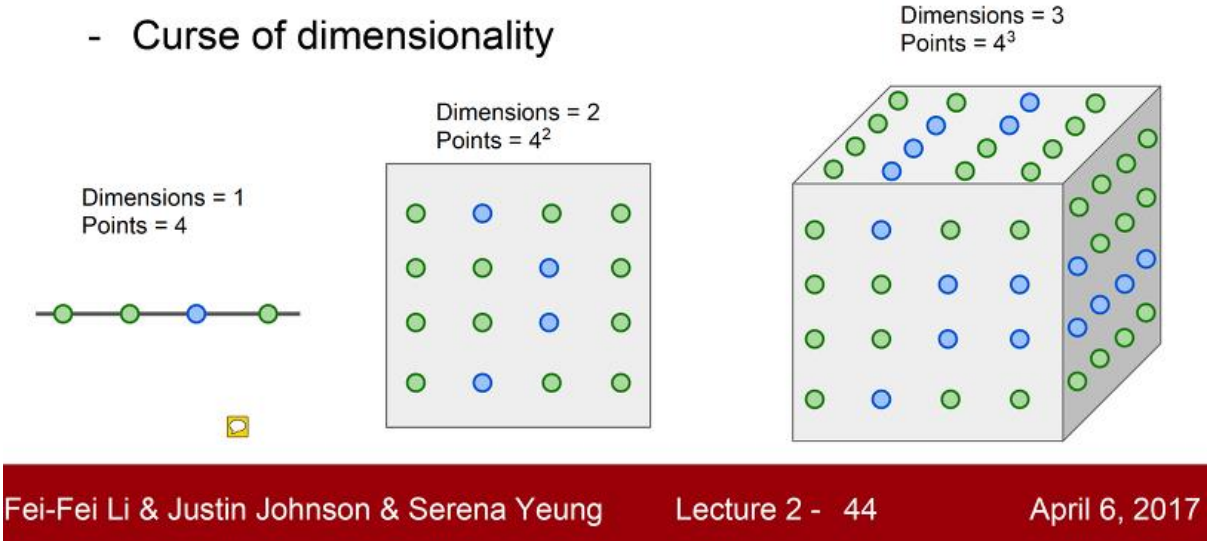
BUT KNN 알고리즘은 image classification 분야에서 절대 쓰이지 않음.

- prediction 이 매우 느림.
- 픽셀 간의 차가 이미지의 유사성/차이를 잘 나타내지 못함.

Original image is CC0 public domain

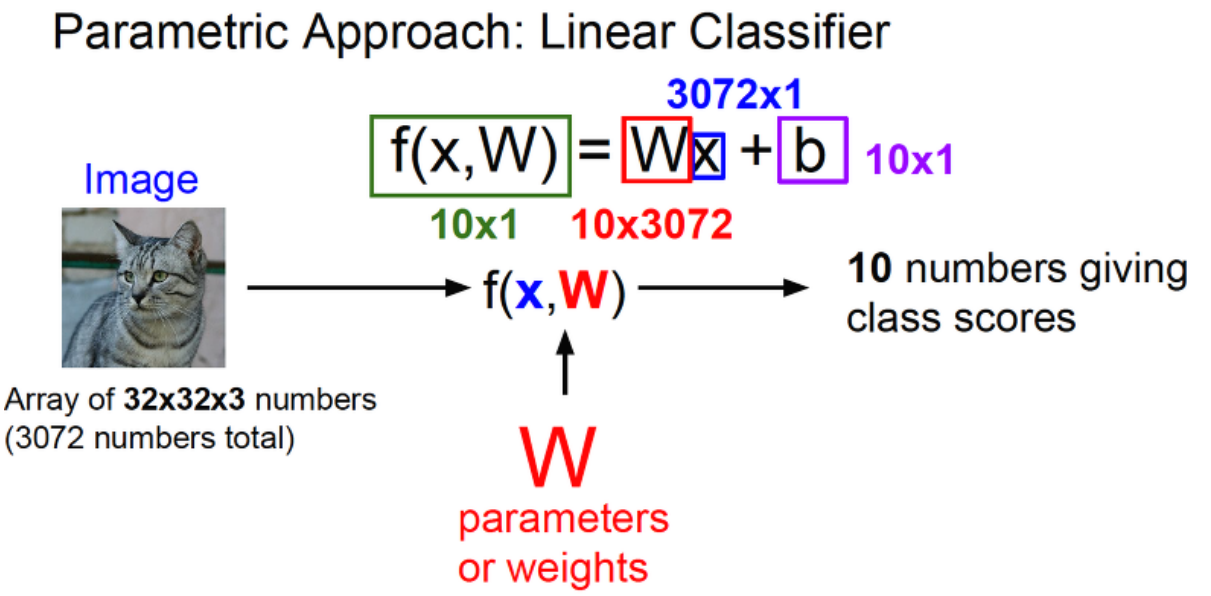
(all 3 images have same L2 distance to the one on the left)

- Curse of dimensionality: 알고리즘이 잘 동작하도록 만들기 위해서는 우리가 다룰 data 의 차원제곱만큼의 학습데이터가 필요함.



Linear Classification

: Neural Network 의 기본이 되는 구조 중 하나



x: 입력 데이터

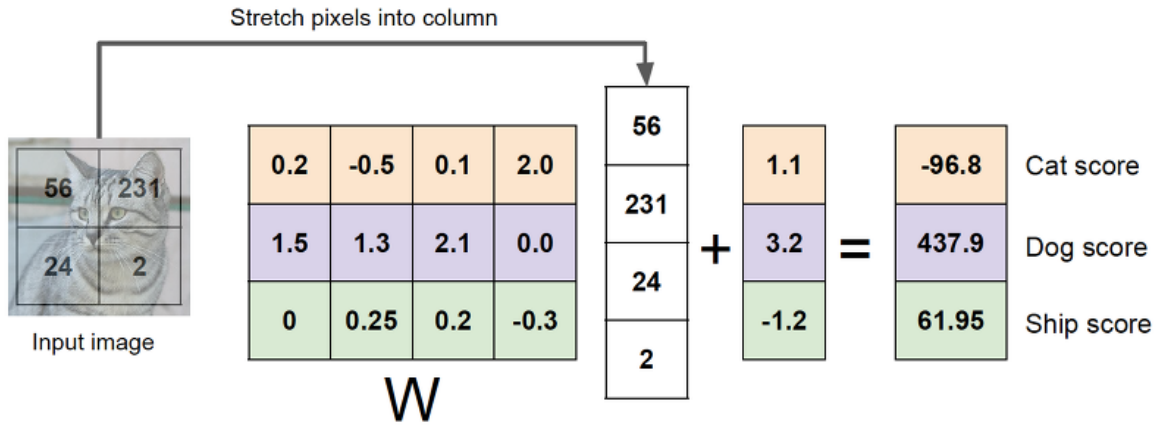
W: 파라미터/weight -> 각 class 의 classifier 를 학습

b: bias (output 과 같은 모양을 갖는 상수) -> 한번도 보지 못한 데이터를 분류할 수 있도록 도와주는 수.

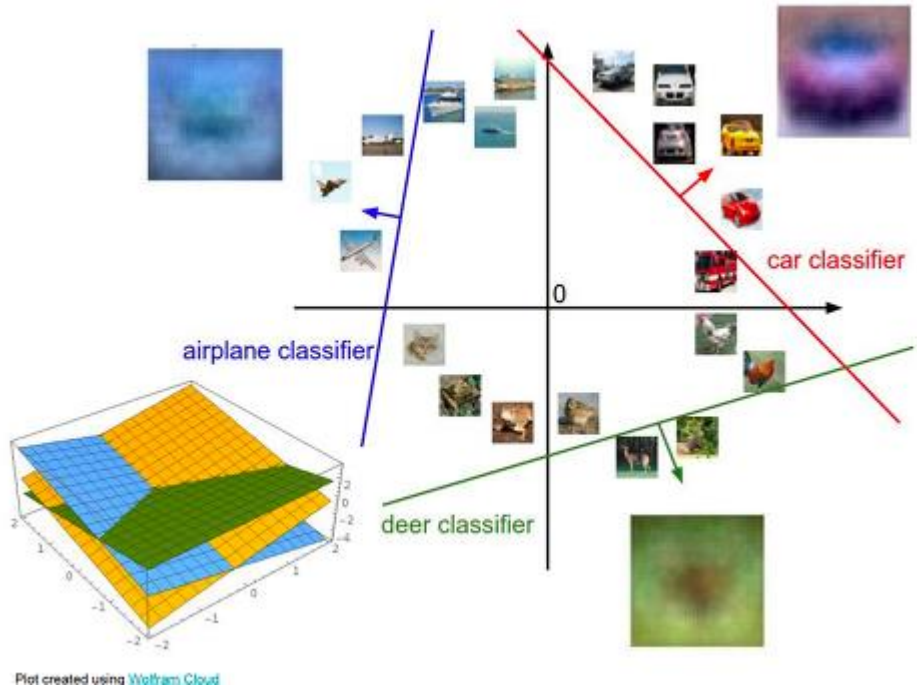
CIFAR10 의 카테고리 수가 10 개이므로 10 개의 숫자를 결과로 받을 수 있어야 함. 여기서 행렬의 곱셈을 사용함.

CIFAR10 의 이미지 사이즈는 [32x32x3]임. 이를 한 줄로 펴면 [3072x1]의 행렬을 얻을 수 있음. 여기서 내적을 이용해 x 와 W 를 곱해야 함. 이 때 output 이 [10x1]이어야 하므로 Wx 순서로 곱해줘야 함.

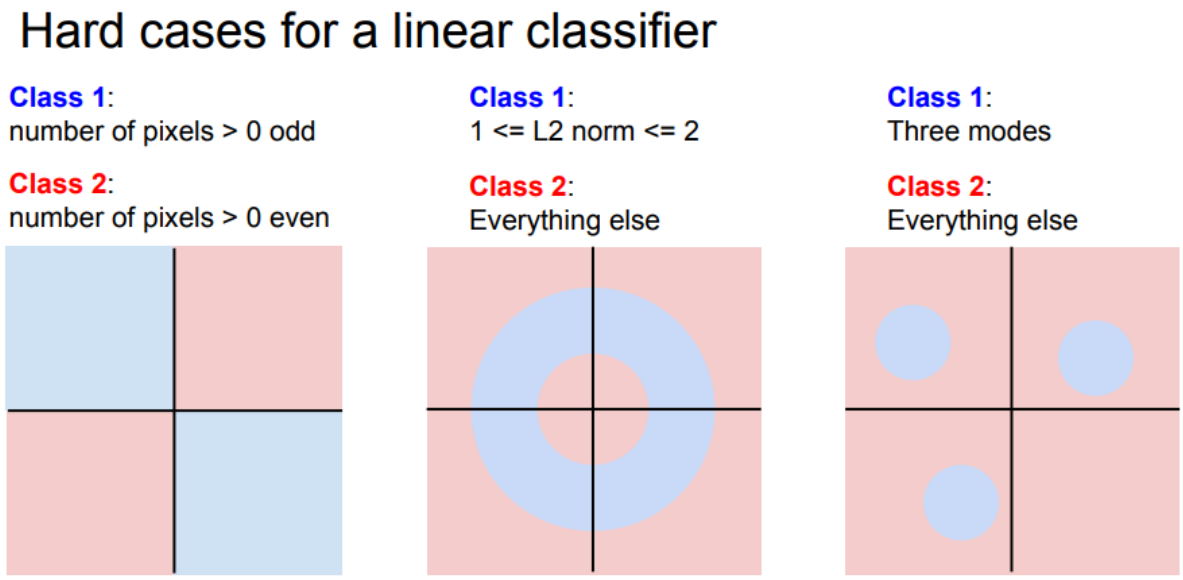
[10x3072] * [3072x1] = [10x1]



해당 슬라이드를 보면 Dog score 가 높으므로 dog 라고 예측이됨. (넣은 이미지는 고양이임. 따라서 잘못된 판단을 한 것.) 따라서 이를 개선시키기 위해 W 의 값을 개선시켜나가야 함. 아래의 사진은 linear classifier 의 예시임.



다음은 linear classifier 가 잘 동작하지 않는 dataset 임



1. 사분면의 반대 위치에 decision boundary 가 있는 경우
2. 원형의 decision boundary 가 있는 경우
3. 여러개의 독립적인 decision boundary 가 있는 경우

->위의 세 경우 모두 하나의 선으로 두 class 를 분류하는 것은 어려움.