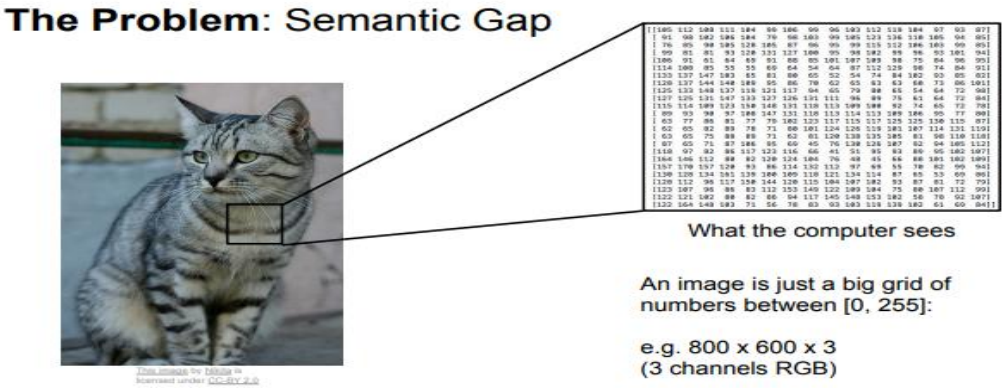
Lecture 2 Image Classification

Image classification : 입력 이미지를 받아 시스템에 미리 정해 놓은 카테고리 집합 중 해당하는 것 고르기

우리의 시각 체계는 이러한 객체 인식 task에 고도화되어 있고 익숙해져 있어 쉬워보이지만 컴퓨터 즉, 기계에게는 정말 어려운 일이다.



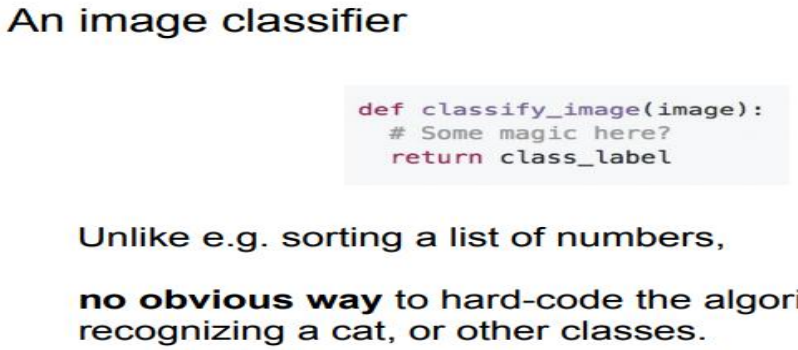
컴퓨터에게 이미지는 각 픽셀마다 3개의 수(RGB)로 표현됨 (red, green, blue) [0,255]

800x600 사이즈 이미지라면 800x600x3 개의 숫자가 있는 것 (3 channels RGB)

➔ 이걸 보고 어떻게 고양이인줄 알까? 이게 바로 의미론적인 차이(고양이라는 레이블을 이 이미지 붙인 것)

우리의 알고리즘은 조명, 각도, 고양이의 자세 같은 다양한 변형에 강인해야한다 -> 어려움

(occasion(다른 사물에 의해 고양이의 일부가 가려짐), background clutter(배경과 객체가 비슷), intraclass variation(하나의 고양이 클래스에도 다양성이 존재))



그래서 image classifier는 단순한 코딩으로 바로 답을 얻을 수 없다. 그래서 여러 시도들이 있었음

여러 가지 시도가 있었다

이미지에서 경계 값과 모서리를 계산 -> 이런 경계와 모서리를 가지면 고양이구나!라고 판단 -> 하지만 이 알고리즘은 각 이미지마다 다른 알고리즘을 작성해야하므로 확장성이 전혀 없다

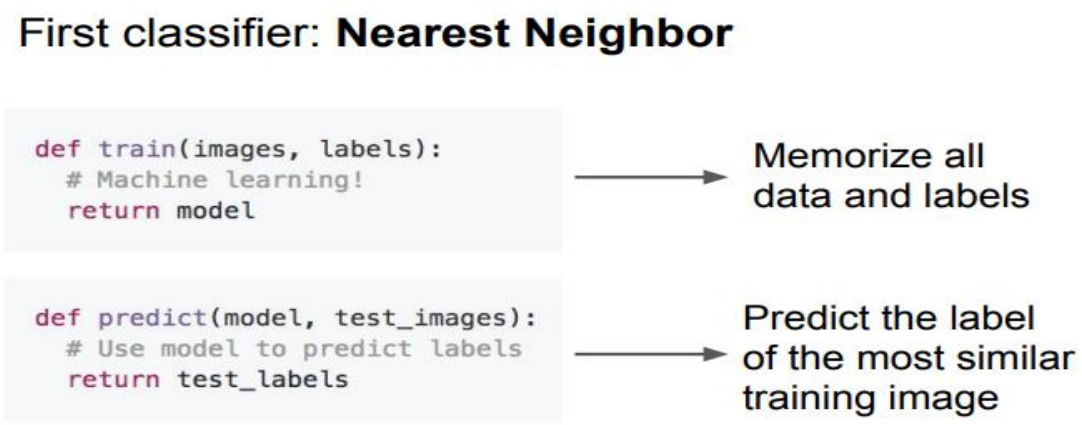
데이터 중심 접근 방법

1. 각 카테고리 별로 인터넷에서 많은 양의 image data 수집
2. Classifier를 훈련시키기 위해 machine learning 이용
3. 새로운 이미지로 classifier 테스트

➔ 우리는 그래서 두 개의 함수 필요(train, predict 함수)

다양한 고양이 사진들을 보여주면서 아이들에게 학습시키면, 아이들은 훨씬 더 빠르게 어떤 것이 고양이인지 알아내는 것과 같은 원리

첫 번째 분류기 : Nearest Neighbor



말 그대로 사진들을 모두 외워서 다른 사진들을 봤을 때 지금까지 알고 있던 사진들과 비교하여 가장 비슷하게 생긴 것을 찾아내는 것이다.

입력 이미지에 NN 알고리즘을 적용하면 트레이닝 셋에서 가장 가까운 샘플을 찾음 -> 그 샘플의 레이블을 반환

CIFAR-10은 machine learning에서 자주 쓰는 연습용 data set

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.

생각해야 할 것! 이미지 쌍이 있을 때 어떤 비교 함수를 사용할 것인가

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	add → 456
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	

L1 distance는 이미지를 pixel-wise로 비교한다.

이 4x4 이미지에서는 두 이미지가 456만큼 차이난다. 숫자가 작을수록 비슷한 이미지라는 뜻

NN classifier 구현 python 코드는 단순하다.

```python
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


```python
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

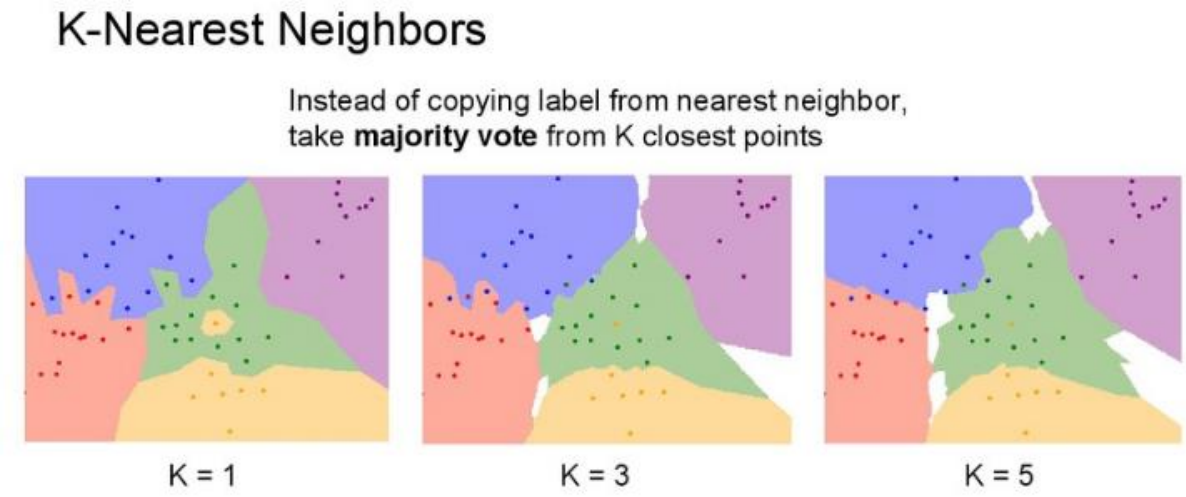
Q: With N examples, how fast are training and prediction?

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

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

보통은 test time이 train time 보다 빨라야한다 -> 안 좋은 classifier

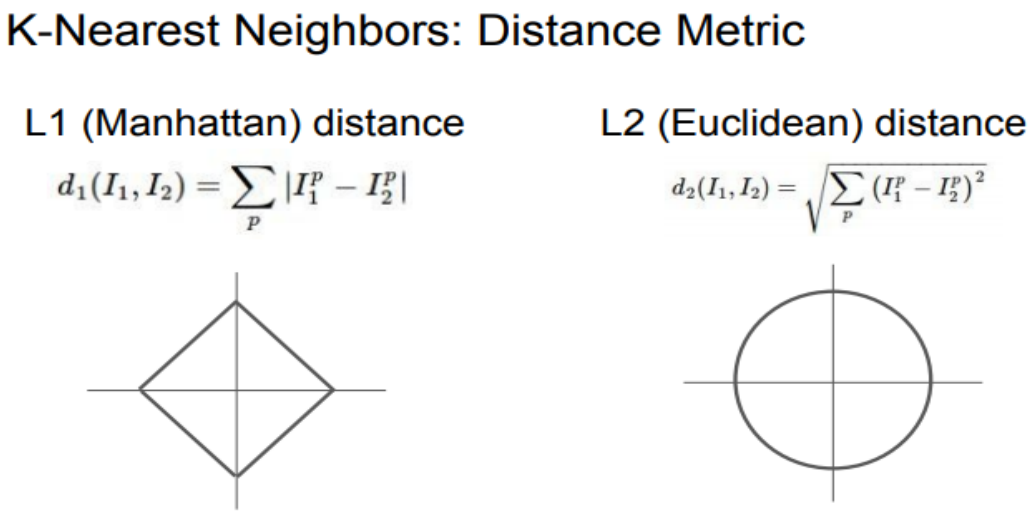
CNN 같은 경우는 train이 오래 걸려도 test는 엄청 빠름



K = 1인 것은 nearest neighbor 방식으로 했을 때 그린 그림이다. 색깔은 개나 고양이 같은 사진의 종류를 말하고, 점들은 사진 자체를 의미한다. 가운데 노란색 부분을 보면 저 노란색 주변에 있는 점들은 초록색이어야할 것 같은데 저 점 하나 때문에 그 부분만 노란색 부분이 된다는 것은 무리가 있어 보인다.

그래서 고안된 것이 KNN이다. 가장 가까운 사진을 찾는건 그대로지만 주변 K개를 봤을 때 가장 비슷한 것이 바로 정답일 것일거라는 생각이다.

➔ Knn도 성능이 좋은 편은 아니다. 그렇지만 k가 클수록 결정력이 상승되는 것은 맞음



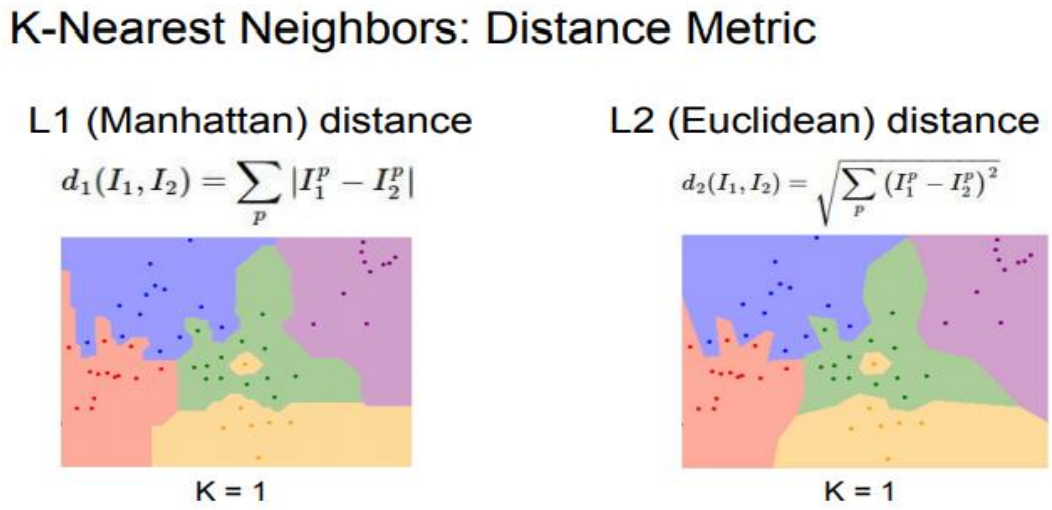
KNN의 거리 척도

L1 : 각 특징 벡터의 각각 요소들이 개별적인 의미를 가진 경우 적합 ex.키, 몸무게

L2: 일반적인 벡터의 경우 더 적합

KNN 분류기로 이미지, 벡터 외에 다양한 종류 데이터를 다룰 수 있다.

예를 들어 문장 분류면 두 문장 간의 거리를 측정할 거리 척도만 정해주면 어떤 데이터라도 가능



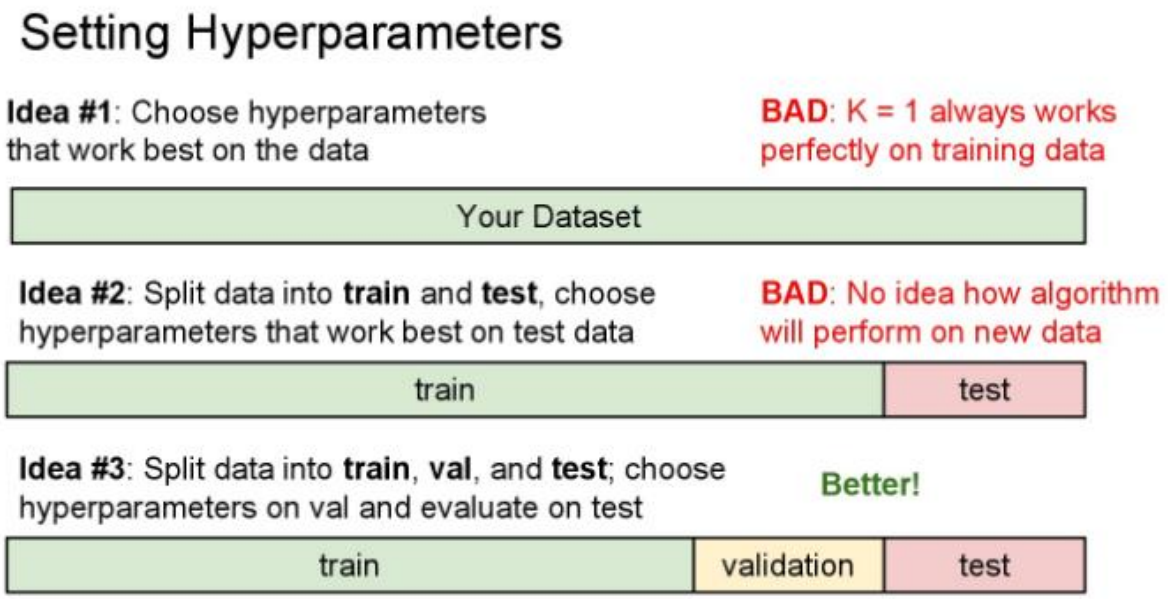
거리 척도에 따라 결과가 다르다.

L1은 좌표 시스템의 영향을 받기 때문에 결정 경계가 축에 영향을 받고

L2는 좌표 시스템의 영향을 받지 않기 때문에 경계가 자연스럽다.

KNN을 사용하기 위해 반드시 선택해야 하는 항목 : 하이퍼파라미터(k값, 거리 척도)

적절한 하이퍼파라미터는 해결해야하는 문제마다 다르다. 여러 시도를 해보고 좋은 것을 선택해야 한다.



아이디어 1 : training data의 정확도와 성능을 최대화하는 hyperparameter 선택

➔ Terrible idea (K=1이 학습 데이터를 가장 완벽하게 분류)

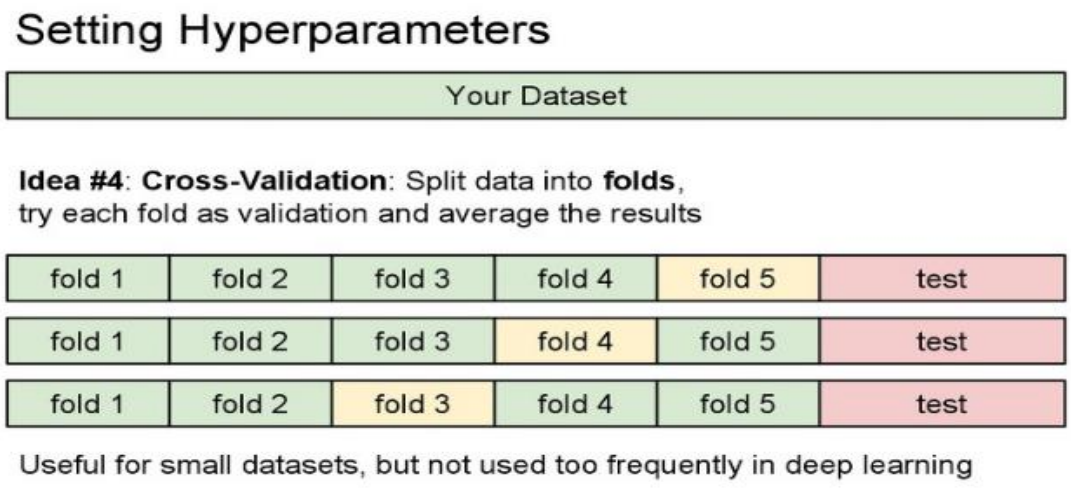
➔ 근데 k가 1보다 커야 학습 데이터에서는 몇 개 잘 못 분류해도 학습 데이터에 없는 새로운 데이터에 대해서는 좋은 성능을 보임(기계학습에서는 이게 중요)

아이디어 2: 전체 dataset을 train과 test로 나눈 후에 test data에서 잘 돌아가는 hyperparameter 선택

➔ Terrible idea (train에서 훈련된 것을 test에서 확인한 후에, test data에 적절한 k 또는 거리 척도를 바꾸어 주더라도, 결국엔 test 셋에서만 잘 되는 것일 수도 있고, 아예 새로운 데이터가 들어왔을 때 어떻게 반응할지 모름

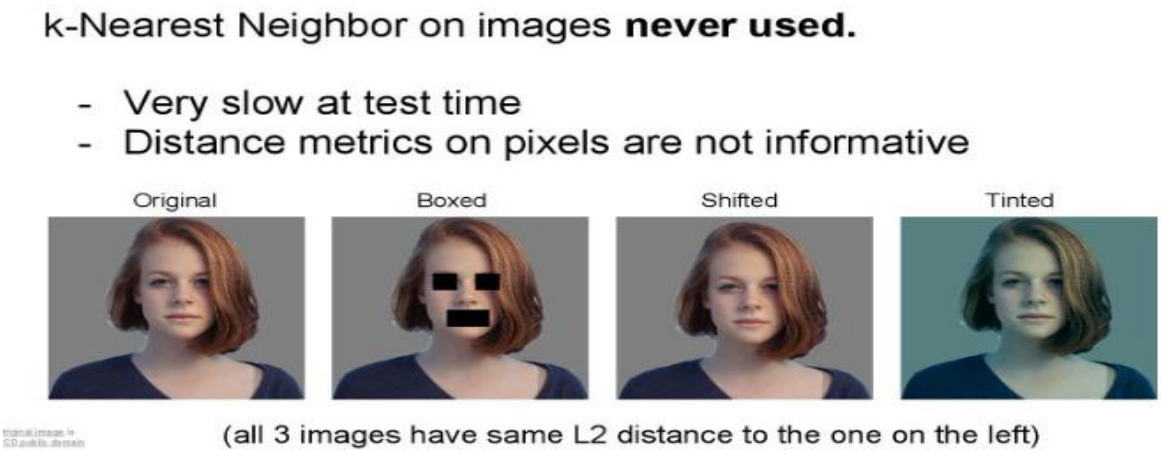
아이디어 3: dataset을 train, validation, test set으로 분류

➔ Train 데이터셋에서 열심히 훈련된 값을 validation 셋으로 검증한다. 여기서 가장 좋았던 parameter를 선택해서 이를 가지고 test set으로 오직 한 번만 수행! 그렇게 되면 우리가 진짜 원하던 새로운 데이터에서의 테스트를 해볼 수 있게 되는 것이다.



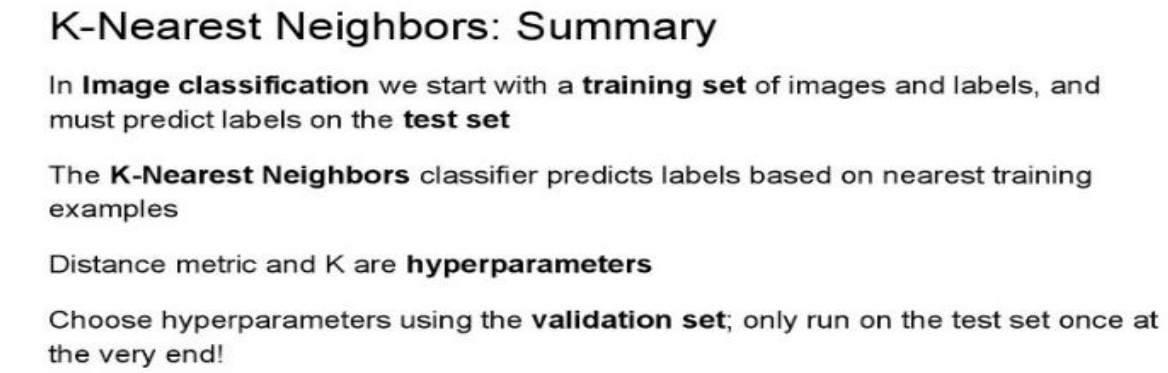
아이디어 4: cross-validation

➔ 일단 큰 거 하나와, 작은거 하나 총 두개로 나눈다. 그 중 작은 것은 test 셋으로 설정하고 큰 거 하나를 놓고 다시 여러 개로 나누어 첫 번째 부분만 val로 놓고 나머지를 train으로 놓은 뒤에 hyperparameter를 설정하고, 다시 두 번째 부분만 val로 놓고 나머지를 모두 train으로 놓은 뒤에 hyperparameter를 설정하고... 이런 식으로 해서 그 결과의 평균값을 hyperparameter로 한다. 근데 이것은 계산량이 많아 deep learning에서는 잘 쓰이지 않는다. (다음 그림은 5-fold cross validation)

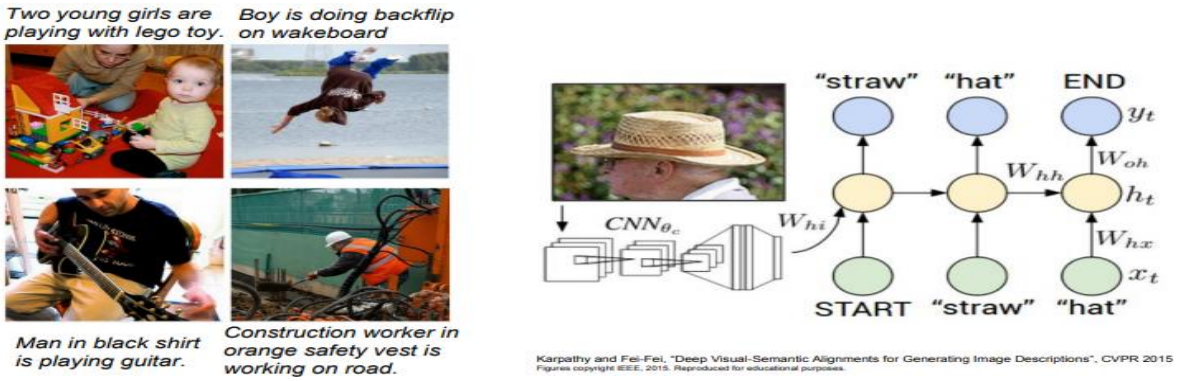


아쉽게도 근데 이미지 분류에 KNN은 쓰이지 않는다.

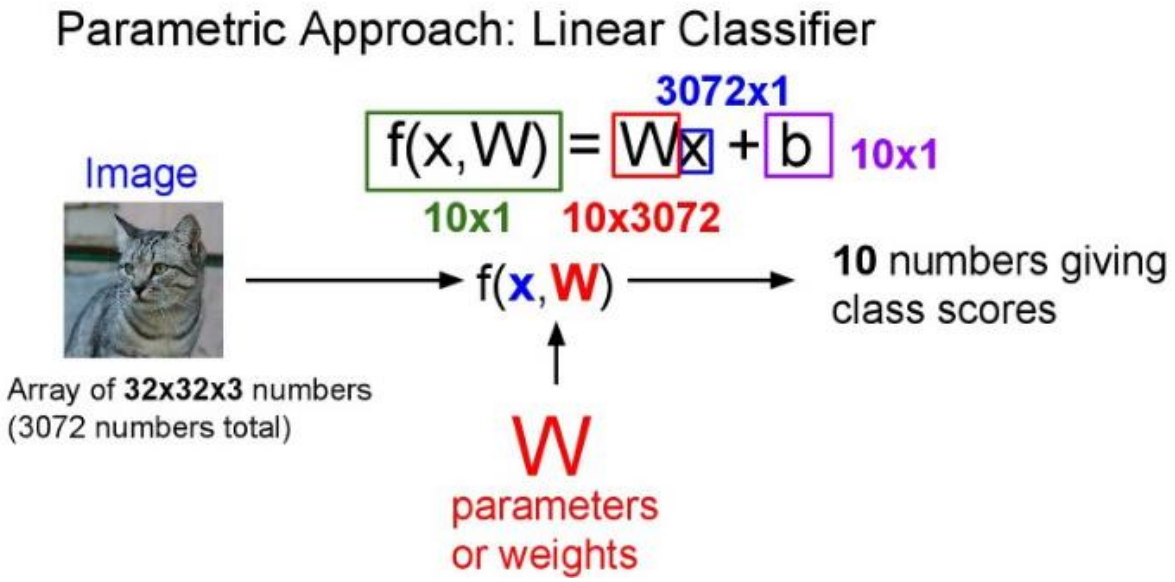
Predict 하는데 시간이 굉장히 오래 걸리기도 하고, 이미지 사이의 distance는 사실 의미있는 값이 아니기 때문이다. 오른쪽 세 개의 사진은 분명 왼쪽 사진과 다름에도 불구하고 knn으로 분류하면 같은 L2 distance 값을 가지기 때문에 같은 사진이라 분류한다.



<Linear Classification>



이미지 인식을 위해 필요한 CNN + 언어 인식을 위해 필요한 RNN을 레고 블록처럼 붙이고 한번에 학습 시킨다.



Linear classifier는 parametric approach!

입력 이미지를 x라 하고 가중치를 w라고 한다.

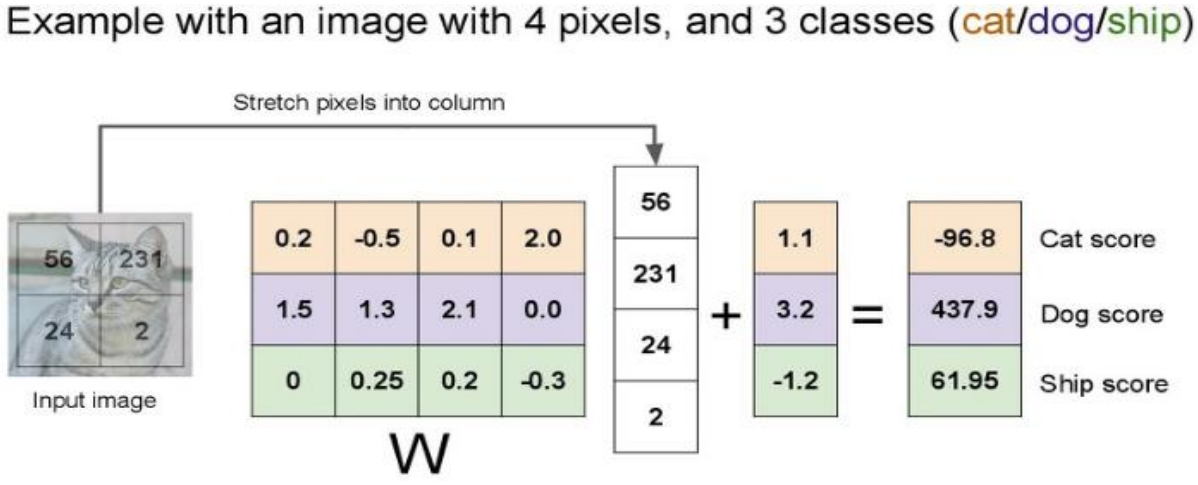
이미지가 32x32x3개의 숫자로 이루어져있다고 하면(CIFAR-10의 이미지 사이즈가 32X32), 그 이미지와 w라는 변수를 함수에 넣으면 10개 클래스에 대한 점수가 나오게 하는 것이다.

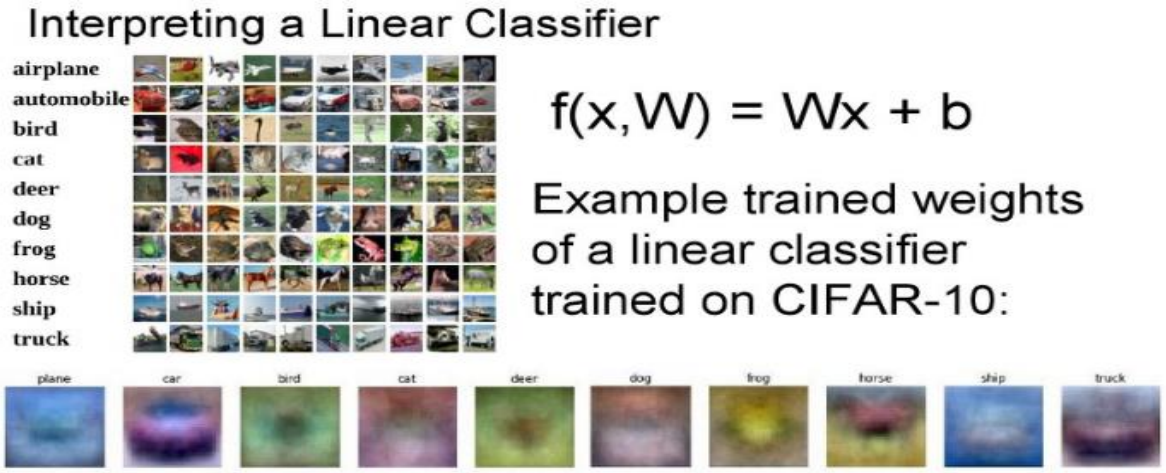
이것이 바로 linear classification의 매개변수적 접근 방식이다.

KNN은 w라는 parameter 없었고, 그냥 모든 training set을 test time에 사용했다. 근데 parametric approach에서는 트레이닝 데이터 셋의 정보를 요약해 그 정보를 w에 담기 때문에 test time에서 더 이상 training set이 필요하지 않아 시간이 절약된다.

F(x.W) = Wx+b (여기서 b는 bias term : 데이터 셋이 불균형할 때 추가)

Ex) 데이터 셋에 고양이만 엄청 많고 강아지가 적다면 측정할 이미지가 고양이일 확률이 크니까 bias 값을 고양이에 크게 점수를 부여해주는 것이다.



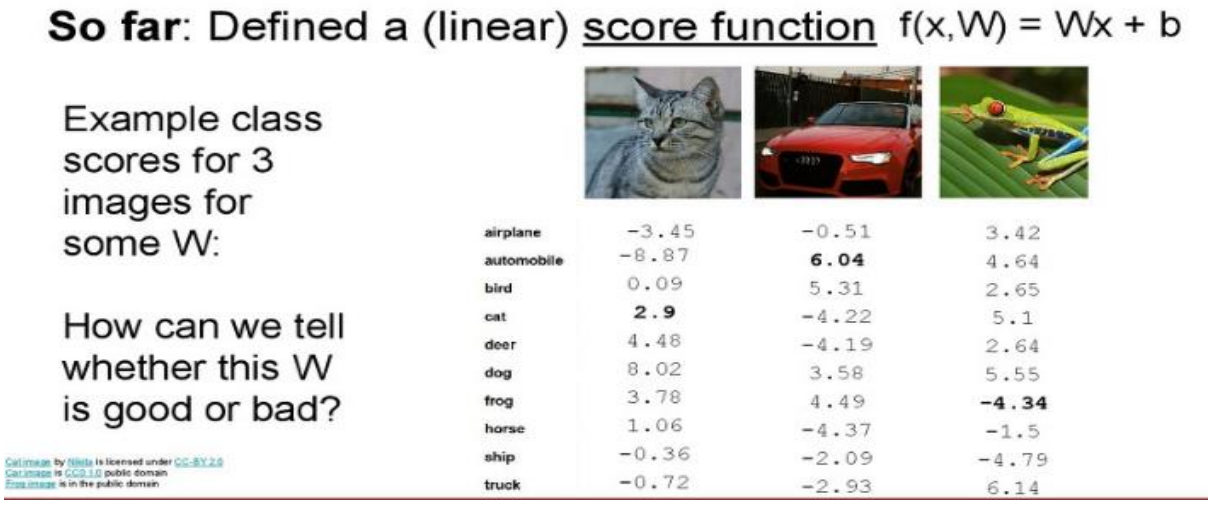


Linear classification은 템플릿 매칭과 유사하다.

가중치 행렬 w의 각 행은 각 이미지 클래스에 대한 템플릿으로 볼 수 있고 그 행 벡터와 이미지 열벡터 간의 내적 계산을 하면 된다.

W는 학습된 결과이다.

문제점 : 각 클래스에 대해서 단 하나의 템플릿만을 학습가능하다. 위 사진들은 CIFAR-10 데이터셋으로 훈련된 weight값을 사진으로 표현한 것이다. 각 카테고리에 가까운 방향으로 사진이 나온 것 같긴 하지만, horse는 머리가 양 쪽에 두개 달린 것처럼 이상하다. 이것이 바로 linear classifier의 문제점 중 하나인데, 왼쪽을 보고 있는 말과 오른쪽을 보고 있는 말 같은 다양한 사진을 평균을 내버린다는 것이다.



다음 linear function은 제 기능을 못한다. 고양이의 경우 강아지가 가장 점수가 높다는 등....

좋은 w값을 찾는 방법은 다음 시간에 계속된다.