# Lecture 1

* 컴퓨터 비전의 역사

1. 60년대

-Block World: 사물을 기하학적 모형으로 단순화

-Summer Vision project

2. 70년대

-David Marr: 이미지를 최종 full 3D 표현으로 만들기 위한 단계지정(Input Image -> Primal sketch -> 2.5D sketch -> 3D Model)

-Generalized Cylinder: 원통모양 조합

-Pictorial Structure: 주요부위와 관절 조합

3. 80년대

-David Lowe: 면도기 인식 위해 선, 경계, 직선을 사용

* 컴퓨터비전으로 어떤 일을 할 수 있을까 고민한 시대

4. 객체인식이 어려우면 객체 분할(segmentation)을 먼저 해볼까?

-그래프이론 도입: 영상 분할 문제 해결

-얼굴인식: 1999-2000년대 AdaBoost 를 이용한 실시간 얼굴인식 성공

5. 어떻게 객체를 잘 인식할것인가?

-SIFT feature: 객체에서 중요한 특징들을 찾아 그 특징들을 다른 객체에 매칭

-Spatial Pyramid Matching: 특징을 잘 뽑아낼 수 있다면 그 특징들의 단서를 제공 가능

* 컴퓨터비전이 앞으로 풀어야 할 문제가 무엇인지 정의를 내림(객체인식)

6. Benchmark Datasets 모으기

-PASCAL Visual Object Challenge

7.기계학습 알고리즘들이 트레이닝과정에서 과적합하는 문제, 세상 모든 객체들을 인식할 수 있는가?

-ImageNet프로젝트 (당시 AI분야에서 만든 가장 큰 데이터셋) : Benchmark에 어떻게 사용하지? -> 2010에 대회개최를 통해 알고리즘을 테스트 -> CNN모델을 사용한 2012의 모델은 사람보다 오류율이 낮은 결과를 보임

* CS231n overview

1.중점적으로 다룰 내용

-Image Classification

-Object detection: 해당하는 부분이 어디에 있는지 네모상자를 그림

-Image captioning: 적절한 caption 생성

2. ImageNet 대회 우승 알고리즘

-2011년: 아직 계층적 알고리즘을 사용

-2012년(CNN의 시대 돌입): Supervision Neural network

-이후 레이어가 더 많은 network가 우승함

3. CNN의 개발은 과거에 이루어졌다.

-1998: 이미지를 입력으로 받아 숫자와 문자를 인식하는 CNN만듦, 구조가 2012의 Alexnet과 비슷

(구조: raw pixel -> convolution layer -> subsampling -> fullyconnected layer)

-왜 최근에 유명해졌는가?

1) 컴퓨터의 계산속도 , GPU의 발전 -> 연산량 증가

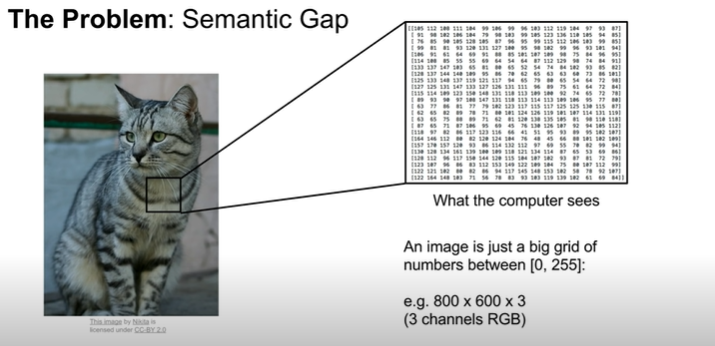
2) 규모가 큰 레이블매겨진 이미지 데이터셋의 등장(CNN이 동작을 잘 하게 한다.)

4. 사람 처럼 볼 수 있는 기계 만들기

- 인간은 이미지를 짧은 시간 봐도 배경지식과 상상력을 사용해 풍부하게 묘사할 수 있다.

# Lecture 2

-Image Classification: 컴퓨터가 입력이미지를 받아 미리 정해둔 카테고리 집합중 어느 카테고리에 속하는지 고름

-컴퓨터가 보는 고양이 이미지는 다음과 같다. 각 픽셀은 3개의 숫자로 표현된다. 즉 거대한 숫자집합에서 고양이를 인식하는것은 어려움(Sementic Gap)

-카메라를 조금만 옆으로 옮기거나 조명을 바꾸면, 고양이가 자세를 바꾸거나 장애물에 의해 가려진다면 픽셀값이 달라지는데 우리 알고리즘은 이를 견뎌야 한다.

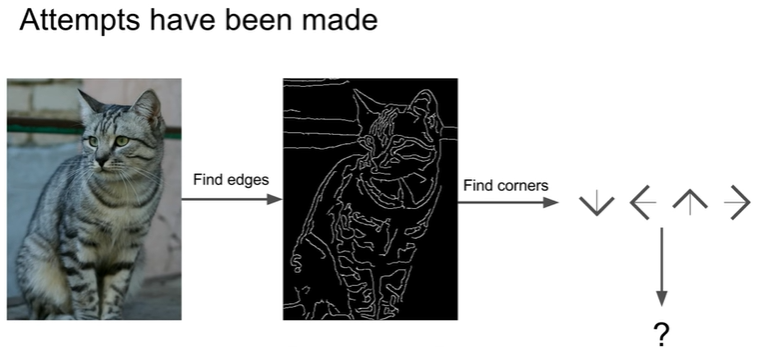
-또 다른 문제: 배경과 비슷한 경우, 하나의 클래스 안에 다양성이 존재함

-이미지 분류 API를 작성해 본다면,

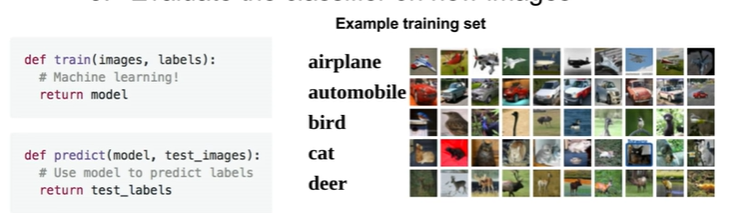


어떻게 풀어야 할까?

사물 인식의 경우 객체를 인식하는 직관적이고 명시적인 알고리즘은 존재하지 않는다.

그간의 시도: 이미지에서 경계와 코너를 카테고리로 분류하여 명시적인 규칙 집합을 만듦 -> 잘 동작하지 않음, 확장성이 없는 방법

-가능한 insight: 데이터 중심 접근방법

1. 많은 사진 데이터셋을 수집한다.

2. 이 데이터셋들을 활용해 머신러닝classifier을 학습시킨다.

3. 학습 모델로 새로운 이미지를 테스트해보면 잘 인식한다.

-> API가 변경됨 train함수(입력: 이미지, 레이블/ 출력: 모델), predict함수(입력: 모델/ 출력: 이미지 예측값)

-간단한 Classifier: Nearest Neighbor

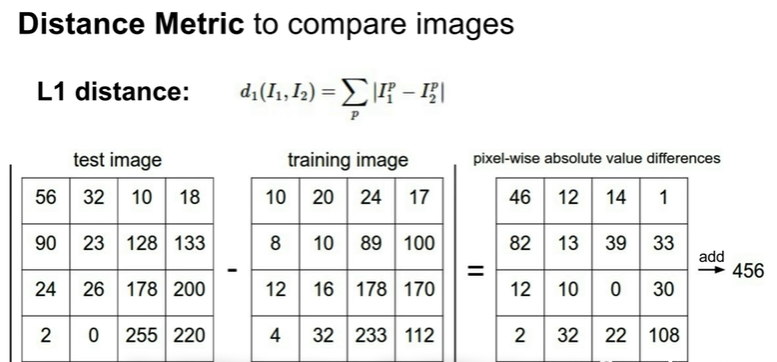
1. Train step: 모든 학습 데이터를 기억한다

2. Prediction step: 새로운 이미지가 들어오면 기존 중 가장 유사한 이미지로 레이블을 예측한다.

-예시데이터셋: CIFAR10

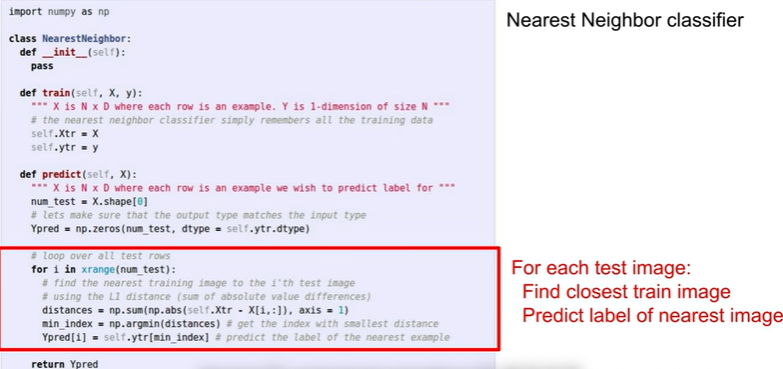
왼쪽부터 가장 가까운 이미지

-이미지 쌍이 있을때 어떻게 비교할까? 어떤 비교함수를 사용하느냐에 달림. 위 예제에는 L1 Distance를 사용했다.



픽셀 차이 값을 빼고 모두 더하는 방식

예시에서는 두 차이가 456만큼 난다.

구현 파이썬 코드

-NN알고리즘은 train 시간은 빠르지만 pridict 시간은 길다. (CNN과는 반대)

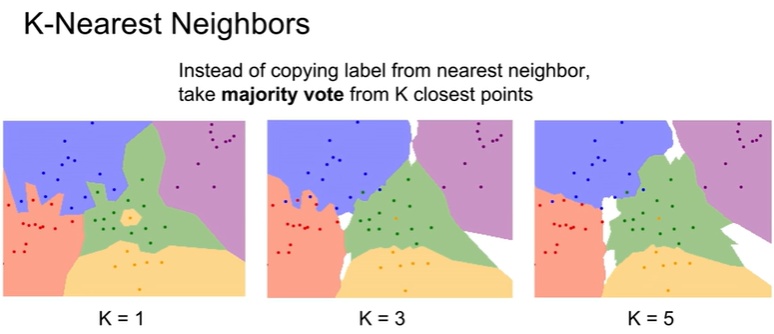
-NN을 적용해보면 어떻게 생겼을까? Decision region 그림

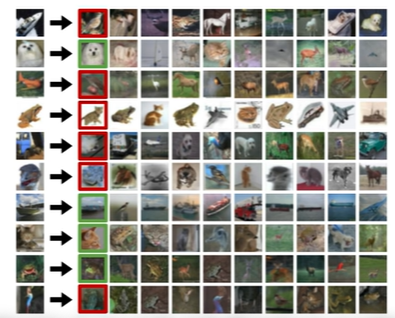


-NN분류기는 공간을 나눠서 각 레이블로 분류한다.

-문제: 초록영역 가운데 노랑영역 생김, 초록영역이 파랑영역 침범

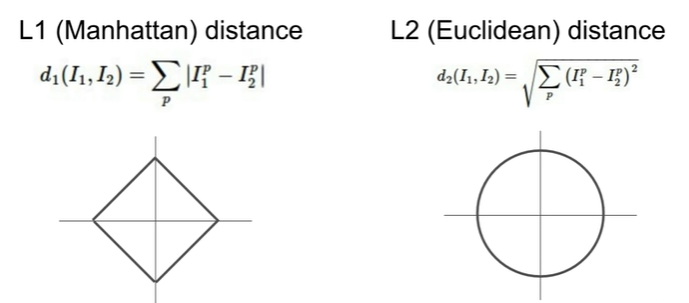
-개선: KNN

Distance metric을 이용해 가까운 이웃을 k개만큼 찾고 이웃끼리 투표하는 방법 (k>1 로 주로 사용)



-> KNN사용: 결과가 안좋음

-Distance Metrics

L1: 기존 좌표계를 회전하면 거리가 바뀐다. 각 성분이 개별적인 의미를 가지면 어울림(키, 몸무게)

L2: 회전에도 거리가 변하지 않으며 일반적인 벡터. 요소들간 실질적인 의미가 없는 경우 사용

->둘다 써보고 성능이 더 좋은 것 사용하자

-Hyperparameter(초모수): KNN 에서 k, Distance Matrix 같은 값

-Hyperparameter 설정 방법

* Idea #1: 모든 데이터를 Train 으로 사용하는 방법
  + 우리는 주어진 데이터에서 좋은 성능을 가지는 것보다 새로운 데이터 들에 대해 높은 정확도 를 가져야 합니다.
  + 모든 데이터를 사용하는 경우 KNN에서 K=1인 경우 Train set에 대해 가장 좋은 결과를 가지지만, 새로운 데이터에 대해서는 그렇지 않습니다.
* Idea #2: 데이터를 Train과 Test 로 나누는 방법
  + 그럼 Train과 Test로 나누어서 사용해볼까요?
  + Train 데이터를 이용해서 Test 데이터가 정답인지 아닌지 확인합시다.
  + 이제 K값을 바꾸어 가면서 이중에 가장 높은 성능을 가지는 K값을 찾으면 되겠습니다!
  + 하지만 이 방법은 이 Test 데이터에 대해서만 좋은 결과 값을 가질 수 도 있습니다.
  + 우리는 실제로 보지 않은 데이터에 대해서 좋은 성능을 평가 해야 합니다.
* Idea #3: 데이터를 Train, Validation(Dev), Test 로 나누는 방법
  + 이제 우리는 validation를 추가로 사용할 것입니다.
  + 이제 Train를 한 뒤에 이 Validation를 이용하여 좋은 성능을 가지는 Hyperparameter를 찾을 것 입니다.
  + 이 Validation에서 좋은 성능을 가지는 Hyperparameter를 찾은 뒤 에 마지막에 딱 한번 Test 데이터를 테스트 하여, 이 데이터를 최대한 "Unseen Data"로 활용하는 방법 입니다.
  + 위 방법들 중 이 방법이 제일 올바른 방법

-Cross-Validation

Dataset를 Fold 단위로 자르고, 이 fold 중 하나를 validation으로 선택하고 나머지를 train 데이터로 사용 합니다.

위 방법으로 Validation으로 사용할 fold를 바꿔가면서 반복 하고, 이중에 가장 좋은 성능을 가지는 Hyperparameter를 찾아내는 방법 입니다.

이 방법은 validation의 데이터가 편향되는 현상을 방지할 수 있습니다.

기존에 방식보다 많은 학습시간 을 요구하며 이 방법은 거의 사용되지 않고 데이터가 적은 상황에서 유용한 장점을 가집니다.

-Curse of dimension(차원의 저주): KNN이 잘 작동하기 위해서는 전체 공간을 조밀하게 커버할 수 있을 정도의 데이터가 필요하다 는 이야기입니다.

여기서 Curse of dimension 이란 아래 그림과 같습니다.

* 1차원 에서는 4개의 데이터가 필요 했다면,
* 2차원 에서는 4 \* 4 = 16개의 데이터가,
* 3차원 에서는 4 \* 4 \* 4 = 64개의 데이터가 필요하다.

이와 같이 고차원으로 갈 수록 기하급수 적인 데이터가 필요 합니다.

이 필요한 데이터가 4개라면 괜찮지만 10000개가 필요하다고 하면 실감할 수 있을 것입니다.

-Linear Classification: 단순하지만 이후에 배우게 되는 Neural Network와 CNN의 기반이 되는 알고리즘입니다. 즉 아래 그림과 같이 기본 블럭이 되는 것입니다.

#### - Parametric Approach

Linear Classification은 "Parametric model" 의 가장 기본적인 형태입니다.

Parametric model에는 두가지 성분이 있습니다.

1. 가중치 값 (Weight Parameter) : W
2. 편향 값 (Bias) : b

위에 Parametric model 값을 단순하게 "Wx + b" 와 같이 선형적으로 사용하는 것을 바로 Linear Classification 이라고 합니다.

아래 그림에서는 기본적인 Linear Classification 예시를 보여줍니다.

1. X: 입력 Image
   * X의 크기 : 32x32x3 = 3072x1 크기를 가지는 고양이 사진을 사용합니다.
2. W: 가중치 값
   * 만약 분류할 동물의 클래스 수가 10개 라고 합시다.
   * Dimension 크기를 맞춰주기 위해서 W의 크기 : 10x3072를 가집니다.
3. b: Bias 값
   * Bias(편향치)는 x와 W의 곱한 결과에 이 값을 더해줍니다.
   * Bias 크기: 10x1를 가지는 것을 볼 수 있습니다. (Class 갯수)
   * 이 Bias는 데이터와 무관하게 특정 클래스에 "우선권"을 부여합니다.
   * 예) 데이터셋이 불균등한 상황 : 고양이 사진이 개 사진보다 많은 경우

- 결국 Linear Classification은 단순히 행렬과 벡터의 곱 의 형태라는 것을 알았고, 템플릿 매칭 과 관련이 있습니다.

이러한 관점에서 각 카테고리에 대해 하나의 템플릿을 학습 한다는 것도 배웠습니다.

그리고 가중치 행렬 W를 학습시키고 나면 새로운 학습 데이터에도 스코어를 매길 수 있습니다.

이러한 관점에서 Train 시간 은 O(N) 만큼 필요하지만 Test 시간 은 O(1)으로 우리가 원하던 Test 시간을 가집니다.