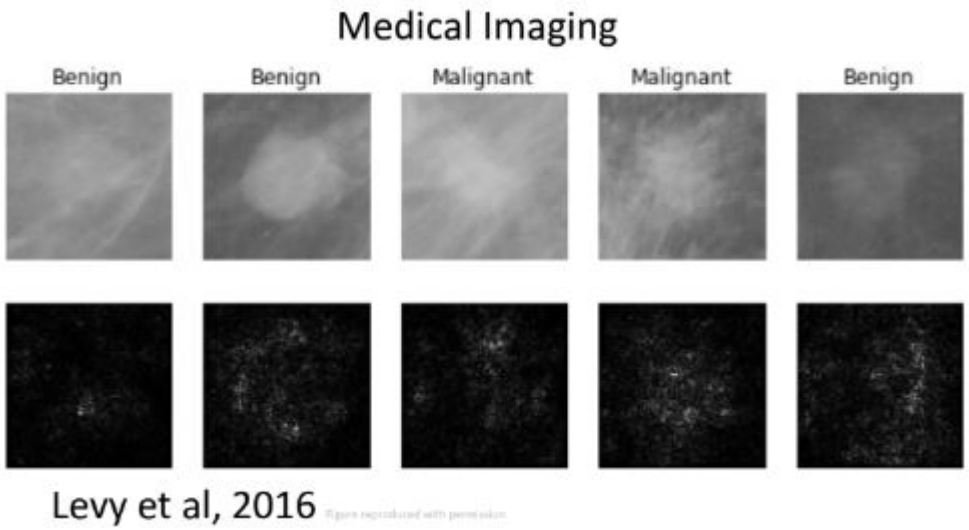
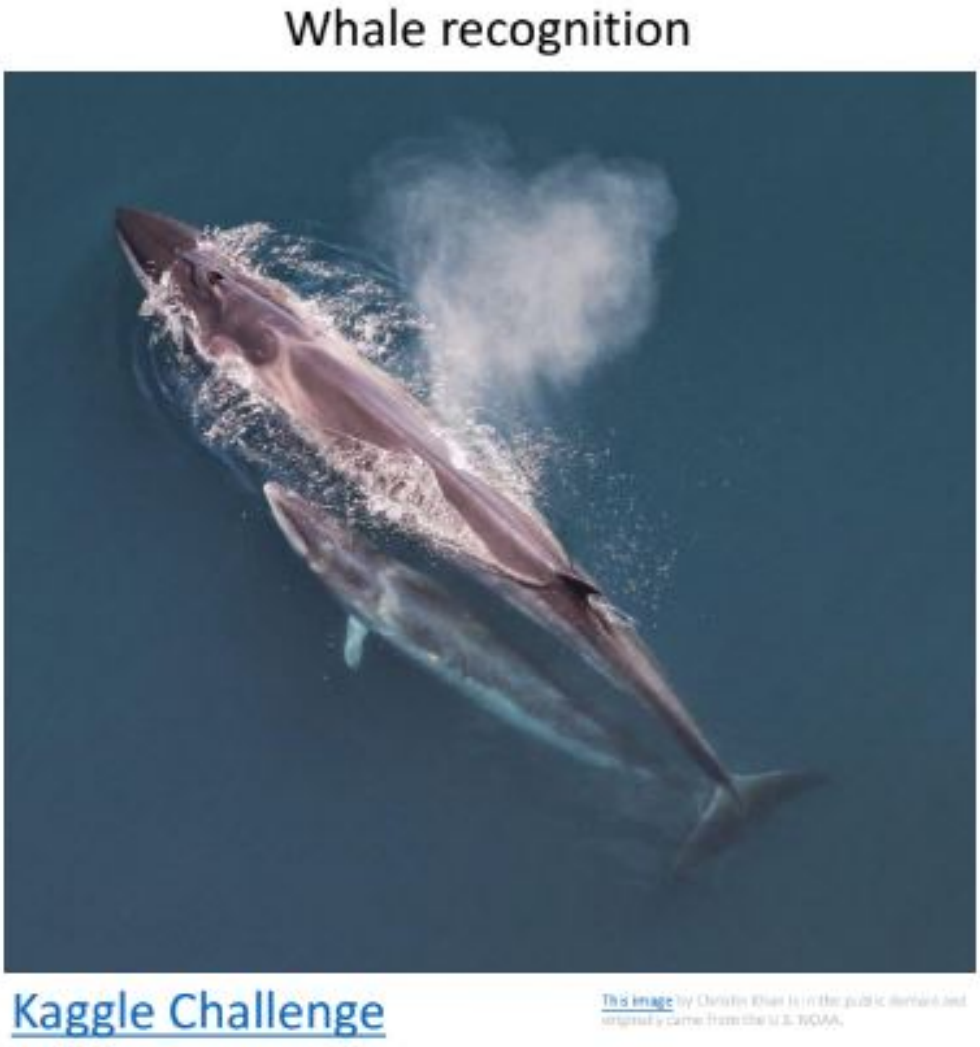
Lecture2 - Image Classification

▼ Image Classification

- **Computer Vision**에서 **Image Classification**은 매우 핵심적이고 근본적인 작업임.
- 그러나 컴퓨터는 모든 것을 숫자로 인식하기 때문에 사람이 직관적으로 인식하는 것과 차이가 있는데, 이를 **Semantic Gap** 이라함.
- 컴퓨터에서 이미지는 기본적으로 0 ~ 255 사이의 pixel로 표현되며, 3개의 channel(RGB)의 matrix형태로 표현됩니다.

▼ Image Classification application





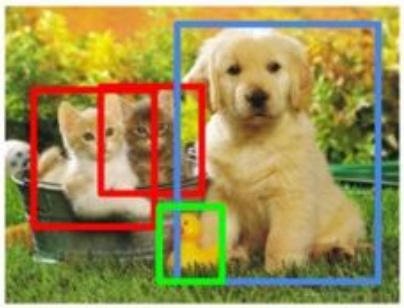
▼ Object detection

- Object detection에서는 **이미지를 classification (분류)** 하는 것 뿐만 아니라 객체라고 판단되는 곳에 **직사각형을 그려주는 localization**을 해주는 작업

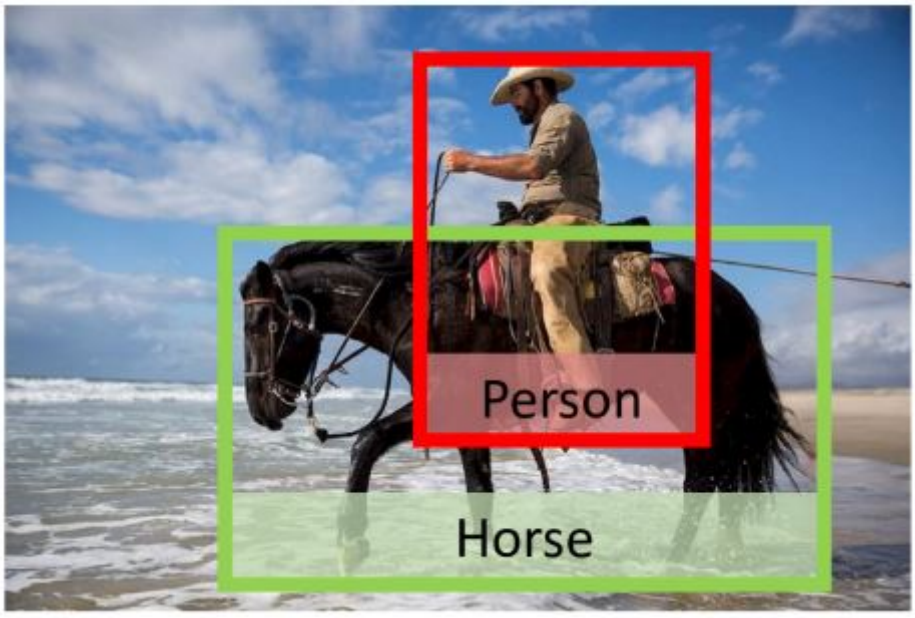
Classification

CAT

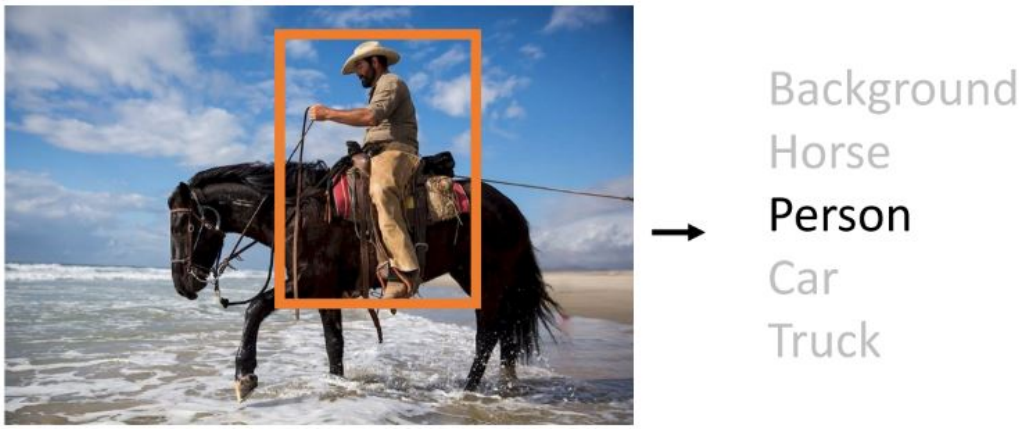
Object Detection



CAT, DOG, DUCK

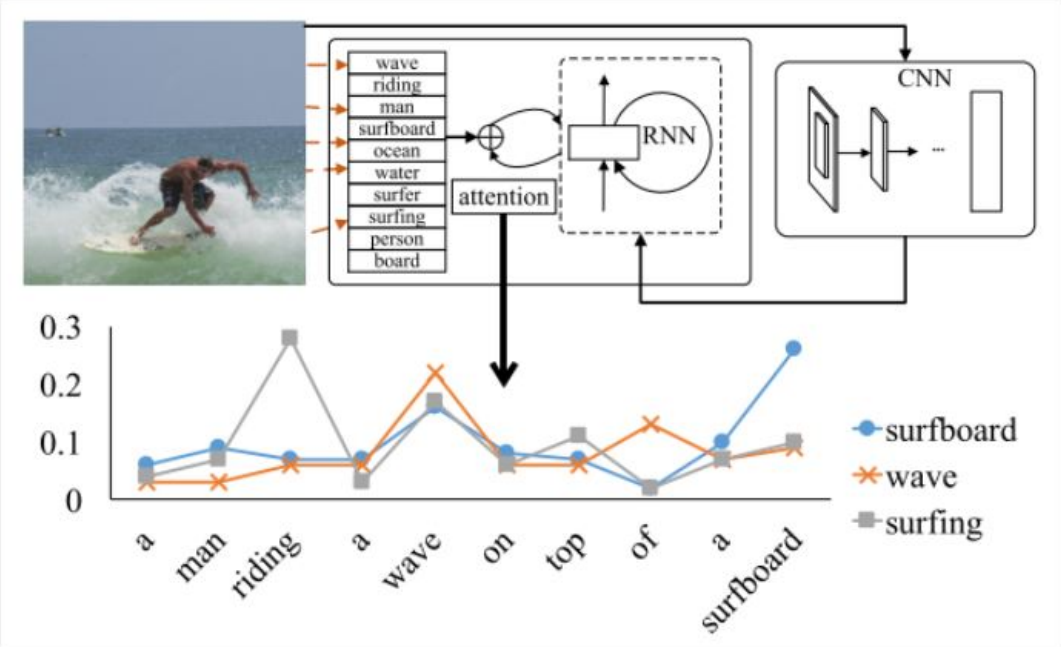


- 방법 → classifiy different **subregion**



▼ Image captioning

- Image Captioning은 말 그대로 이미지의 캡션을 달아주는 일, 즉 **이미지를 보고 어떤 이미지인지 언어로 설명**하는 작업임.

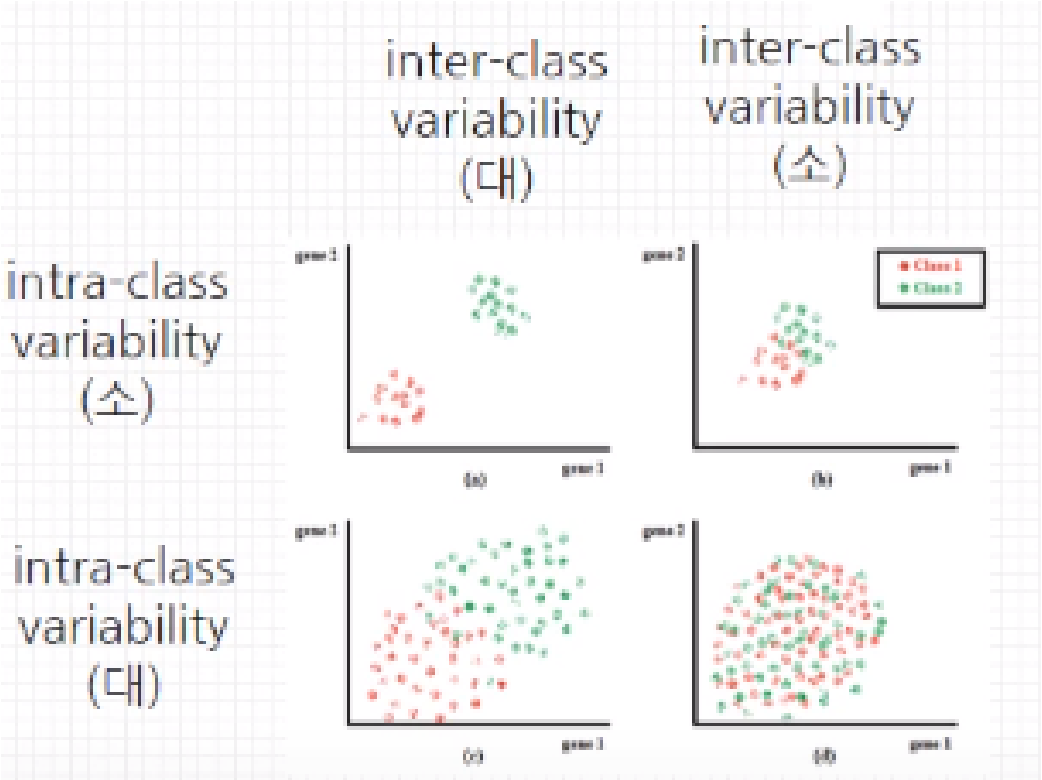


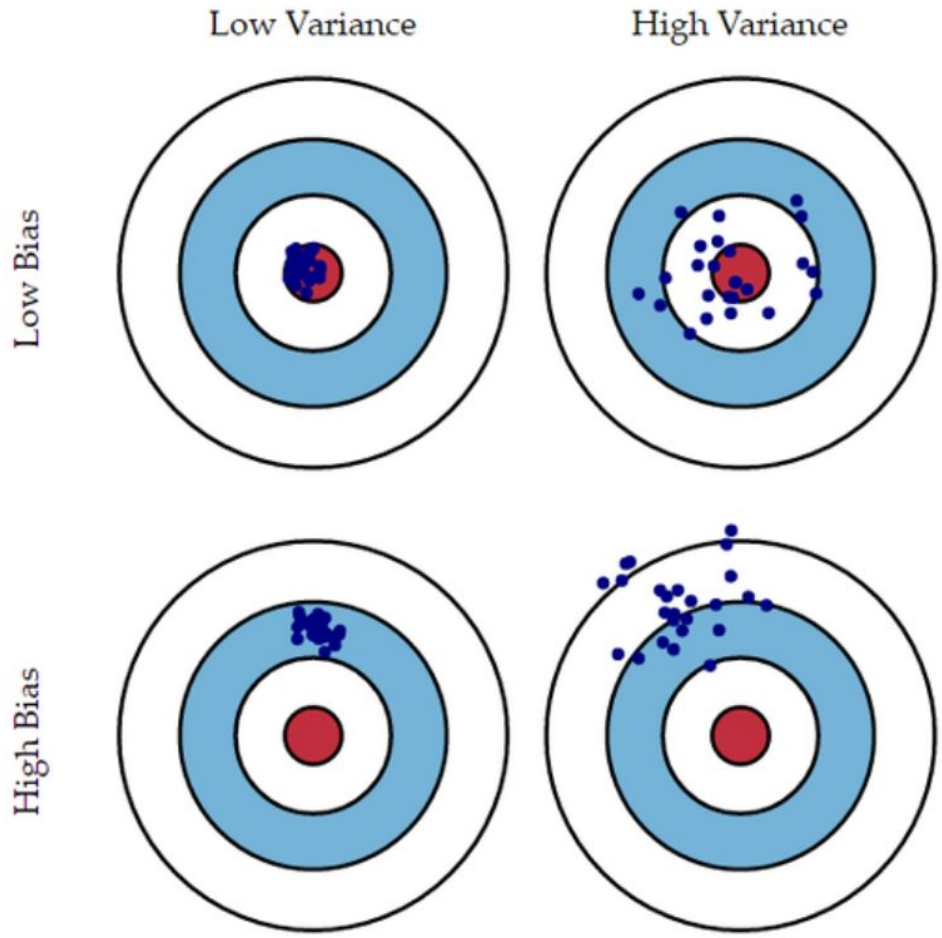
- **Image Captioning**의 접근 방식은 크게 **‘Top-Down Approach’**와 **‘Bottom-Up Approach’**로 구분된다.
- **Top-Down Approach**에서는 이미지를 통째로 시스템에 통과 시켜서 얻은 ‘요점’을 언어로 변환하는 반면 **Bottom-Up Approach**에서는 이미지의 다양한 부분들로부터 단어들을 도출해내고, 이를 결합하여 문장을 얻어냄.
- 현재 가장 많이 쓰이고 있는 접근 방식은 **Top-Down Approach**인데, 그 이유는 **Recurrent Neural Network(RNN)**를 이용하여 각 Parameter들을 Train Data로부터 학습시킬 수 있으며, 이 방식의 성능이 가장 좋다고 평가 받기 때문임.
- 하지만, 이러한 **Top-Down Approach**은 **이미지의 디테일한 부분들에 집중하는 것이 상대적으로 어렵다**는 단점이 있으며 **Bottom-Up Approach**는 이미지의 모든 부분으로부터 하나씩 뽑아낸 단어들을 조합하기 때문에 디테일에까지 신경을 써줄 수 있음.

▼ Challenges

- **ViewPoint Variation**
  - 카메라가 움직일 때마다 pixels이 변화하는 문제

- **Intraclass Variation**
  - **intra-/inter-class variaion**
    - **Intra-class variaion** : class 내부의 분산이 어떠한가
    - **Inter-class variaion :** class 간 분산이 어떠한가

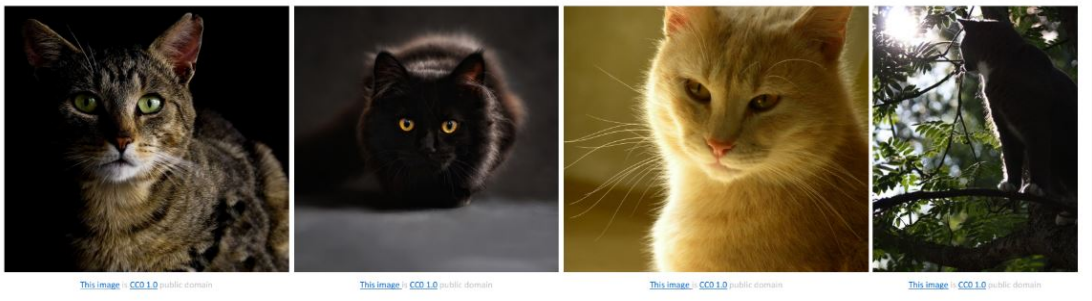




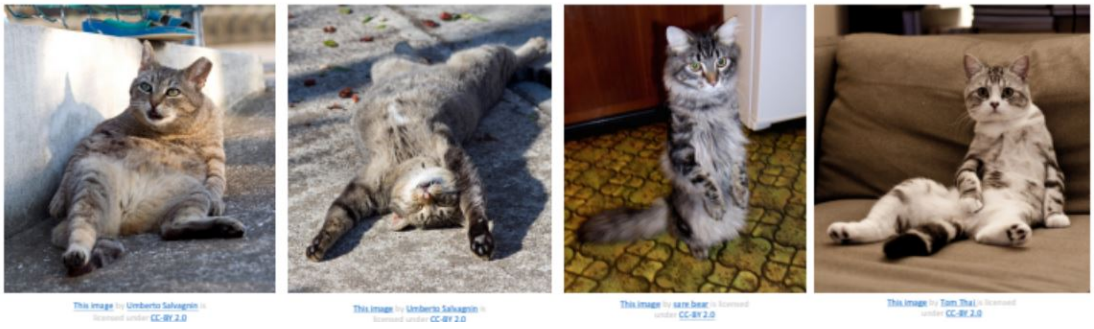
- **Fine-Grained Categories**
  - 비슷한 class 사이에서 classification을 해야함

- **Background Variation**
  - 배경으로 인해 object detection이 어려움

- **illumination Variation**
  - 빛과 그림자의 영향으로 classification이 어려움



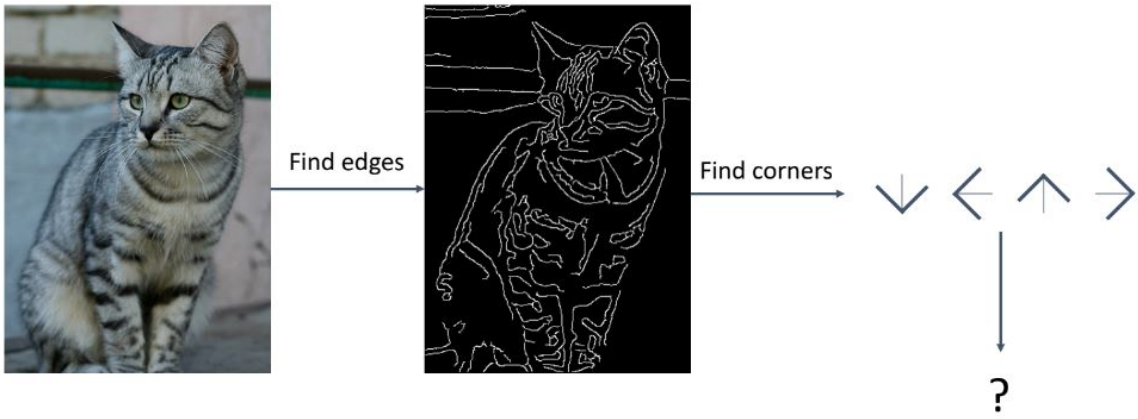
- **Deformation**
  - 같은 Category 라도 different pose or position으로 인해 classification이 어려움



- **Occulsion**
  - 우리가 classify 하려는 object가 숨어있음

▼ Data-driven approach

- 이런 숫자로부터 우리는 **'이 사진이 고양이다'라는 의미를 추출해 내고 싶은 게 목적**이지만 **빛(illumination Variation)**, **변형(Deformation)**, **보호색(Background Variation)** 등 많은 Hurdle이 존재하기에, 명백한 방법이 없었음.



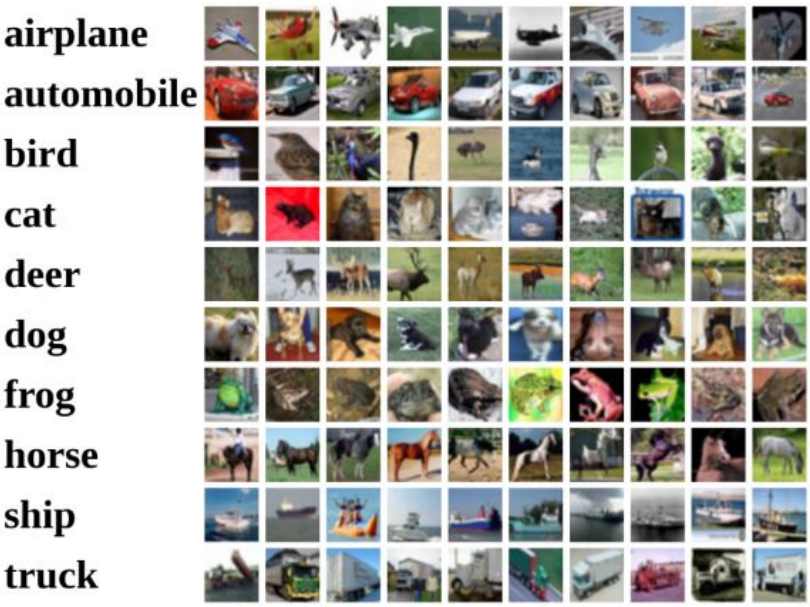
- 가장자리 모서리를 따라 outline을 만들어내며 추출하는 시도들이 있었지만 쉽지 않음. 그래서 고안된 방법이 **Data에 기반한 접근법(Data-driven approach)**임.
- **Data-driven approach**

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

- **MNIST Data**

  - Drosophila of computer vision
    - 연구자들이 초파리로 처음 연구를 해보는 것을 의미
    - 너무 간단한 data set이라 좋은 성능이 나오므로 결과를 맹신하지 말고 아이디어의 검증 용도로만 사용 해야함
    - 10 classes & 50k training images & 10k test images

- **CIFAR10**



- 10 classes & 50k training images & 10k test images
- + 32 x 32 RGB Images

- ImageNet

- **Gold Standard**
- 높은 정확도를 보이는 상위 5개 알고리즘이 예측한 5개 labels 중 하나로 classify
- 1000 classes & 1.3M training images & 50k validation images (50 per class) & 100k test images (100 per class)

    - Images have variable size, but often resized to 256x256 for training
- MIT Places

    - **focus on scene category**
    - 365 classes of different scene types & 8M training images & 18.25k validation images (50 per class) & 328.5k test images (900 per class)
    - Images have variable size, but often resized to 256x256 for training
- Number of Training Pixels
    - ImageNet Data set is **qualitively different and convincing** data set but **computationally expensive**
    - CIFAR 100 is **middle ground**
    - Data set size trend is **getting bigger**

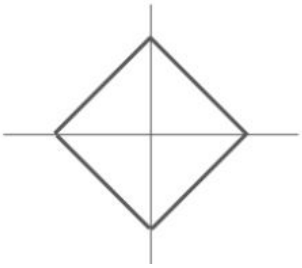
▼ Nearest Neighbor

- Implement ML algorithm, we need to implement **train & predict function**
    - Train func : Memorize all data and label
    - Predict fun : Predict the label of the most similar training image
        - 2 개 이미지의 유사도를 계산하는 함수가 필요

▼ L1 distance vs L2 distance

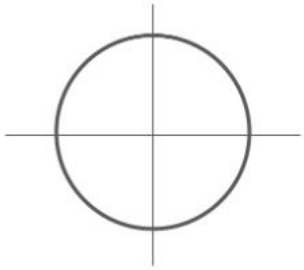
L1 (Manhattan) distance

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



L2 (Euclidean) distance

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



- L1 distance : 각 pixel값의 차이를 구한 후 결괏값을 합산하는 방법
  - 이는 Least Absolute Deviations(LAD), Least Absolute Errors(LAE), Least Absolute Value(LAV), Least Absolute Residual(LAR) 등으로도 불림.
  - L1 Loss는 L2 Loss에 비해 **이상치(Outlier)의 영향을 덜 받는, Robust한 특성**을 가집니만, **0에서 미분이 불가능**함.
- L2 distance : 각 pixel값의 차이를 제곱한 후 root를 씌운 후 결괏값을 합산하는 방법
  - 이는 Least Squares Error(LSE, 최소자승법)로도 불림.
  - 이는 두 개 값의 절대값을 계산하던 L1 Loss와는 달리 L2 Loss는 제곱을 취하기에, 이상치가 들어오면 오차가 제곱이 되어 이상치에 더 영향을 받습니다. 때문에 이상치가 있는 경우에는 적용하기 힘든 방법론임.
  - Outlier point가 실제 데이터와 비교적 비슷한 위치에 존재할 때, 영향을 L1 Loss에 비해 덜 받는 **일관적인 예측을 할 수 있는 Stable한 특성을 가짐.**

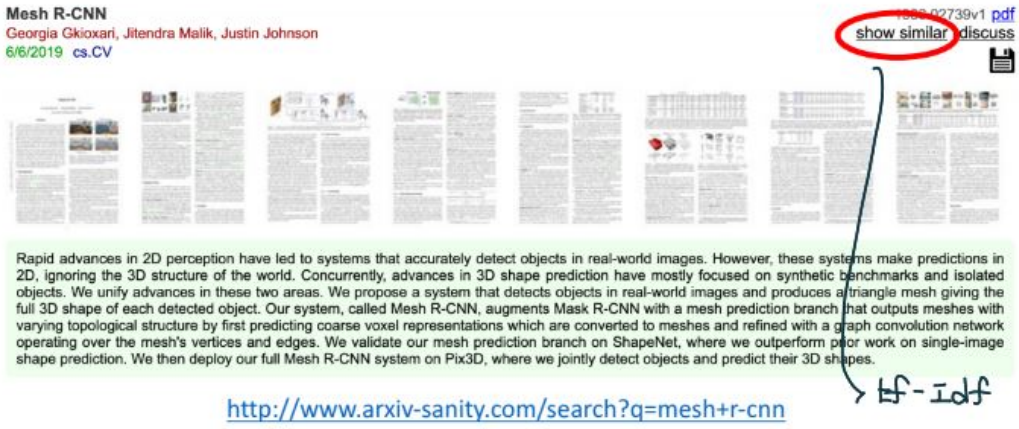
▼ **Nearest Neighbor Classifier**

```
class NearestNeibor :

  def __init__(self) :

    pass

  ## Memorize training data
  def train(self , X , y) :
    # X is N x D where each row is an examples. y is label which is 1-dim
    # of size N

    self.Xtr = X
    self.ytr = y
```

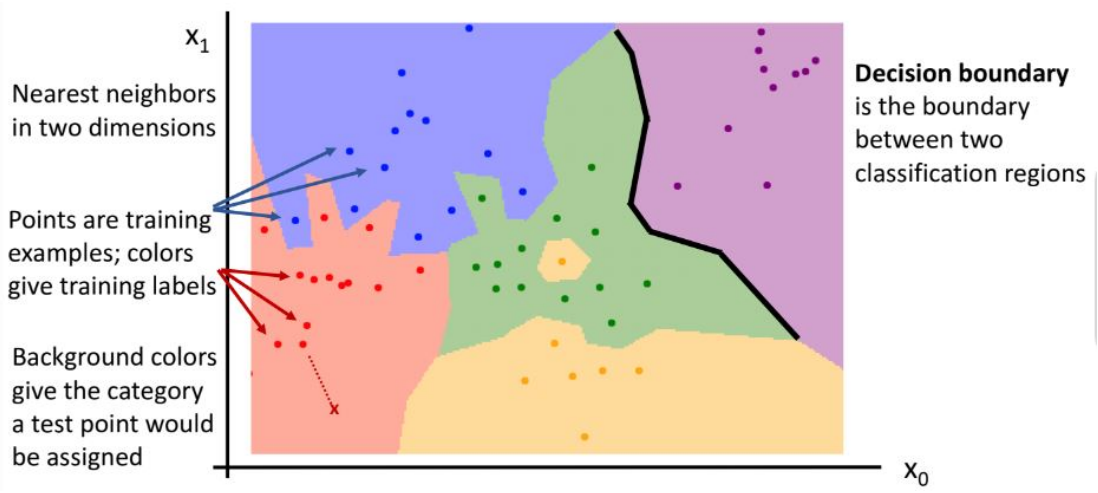
```
def predict(self , X) :

  # X is N x D where each row is an example we wish to predict label for
  num_test = X.shape(0)
  Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

## For each test image
## : Find nearest training image & Return label of nearest image
for i in xrange(num_test)
  # find the nearest training images to the ith test image
  # using the L1 distance (sum of absolute value difference)
  distances = np.sum(np.abs(sel.Xtr - X[i,:]) , axis = 1)

  # get the index with smallest distance
  min_index = np.argmin(distances)

  # predict the label of nearest example
  Ypred[i] = self.ytr[min_index]
```

- Q.1 With N examples, how fast is training?
  - constant (1)
  - just store pointer
- Q.2 With N examples, how fast is testing?
  - linear time (N)
  - folding the size of the image and compuation
  - need to compart it to each of the n training example
  - 실제에 적용하기 위해서는 training이 오래 걸려도 testing이 빨라야하므로 좋지 않은 방법임

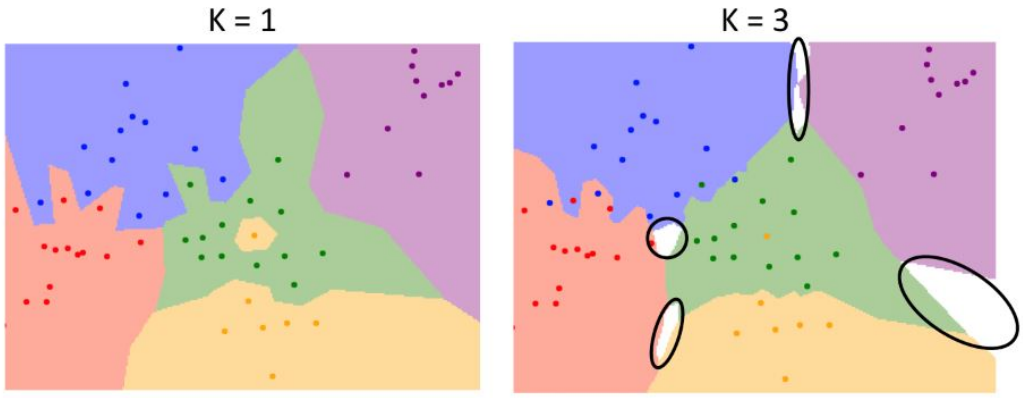
- 그러나 **Distance Metric**만 있으면 **어떠한 타입의 데이터에나 적용할 수 있음**



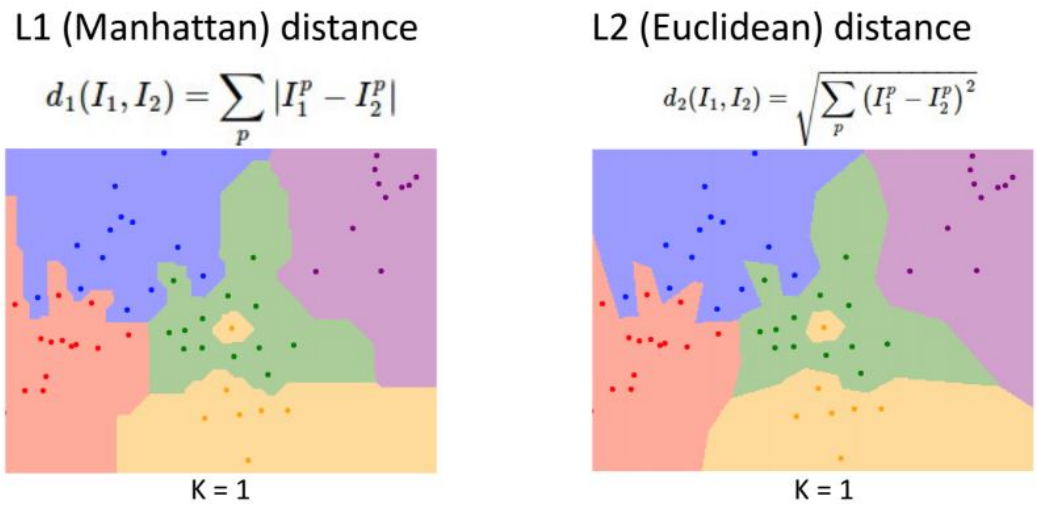
▼ Decision Boundaries



- K의 변화



- Distance Metirc의 변화



  - L1 distance는 수직, 수평, 대각과 같이 boundary의 기울기가 한정적이므로 더 잘게 나누어짐

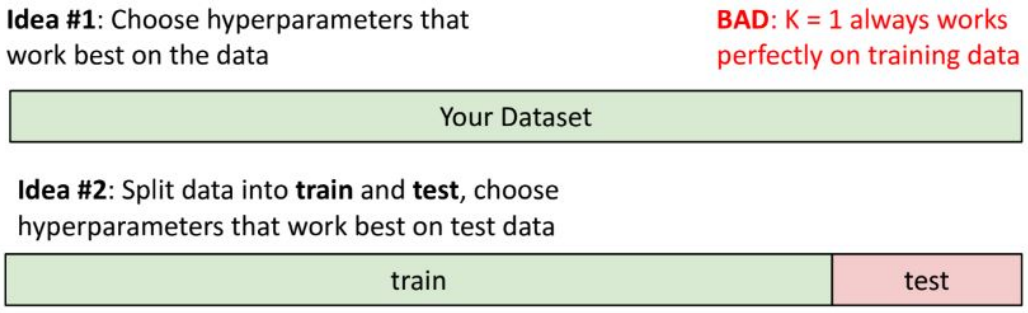
▼ **Hyperparameters**

- What is the best value of **K** to use ?
- What is the best **distance metric** to use ?

1. Choose Hyperparameters that work best on the data

   - K = 1로 설정했을 때 training data에 대해서는 당연히 100%를 예측하겠지만 이는 과적합 문제를 야기함

2. Split data into train and test, choose hyperparameters that work best on test data



   - test data에 알고리즘을 한번 적용한 순간, idea를 pollute하여 새로운 데이터에 어떤 알고리즘이 적합한지 알 수 없음

3. Split data into train, validation, and test, choose Hyperparameters on validation and evaluate on test

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

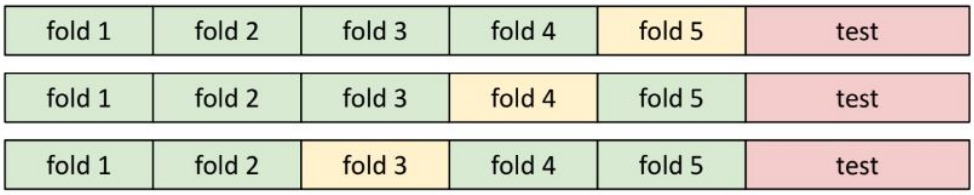
BAD: No idea how algorithm will perform on new data

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

Better!

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

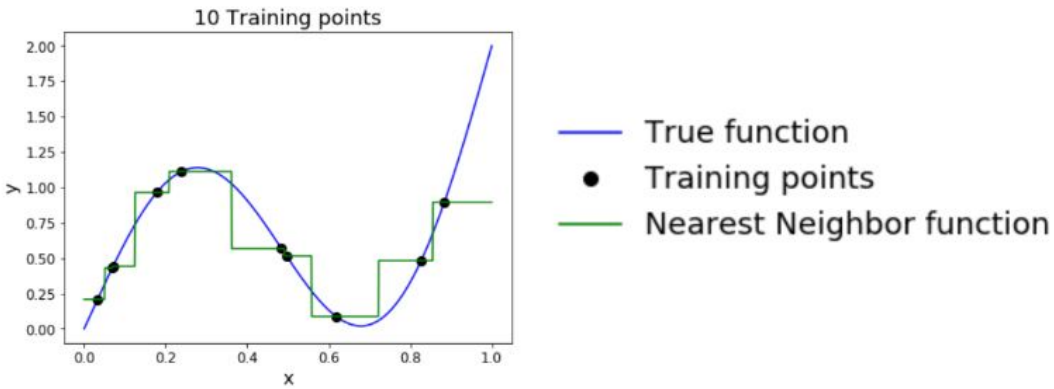
Idea #4: **Cross-Validation**: Split data into **folds**, try each fold as validation and average the results

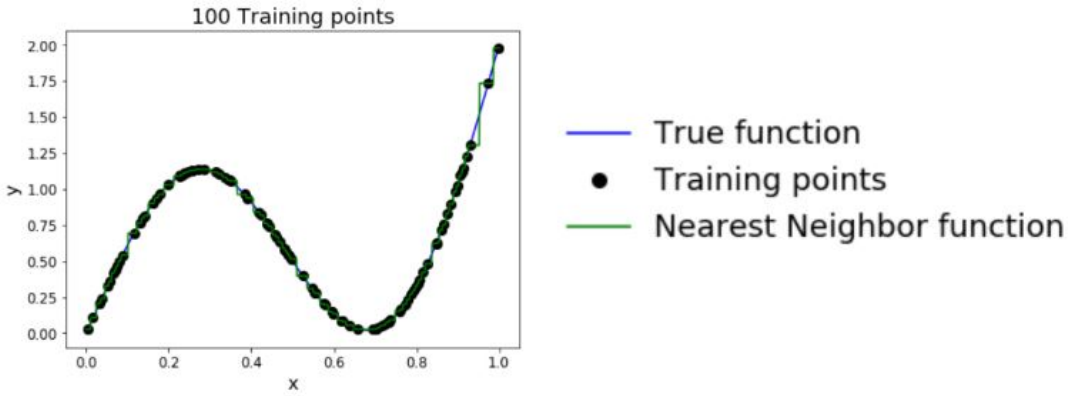


Useful for small datasets, but (unfortunately) not used too frequently in deep learning

▼ Universal Approximation

- As the number of training samples goes to infinity, nearest neighbor can represent any function





- But it causes **Curse of Dimensionality**
  - Curse of Dimensionality : For uniform coverage of space, number of training points needed grows exponentially with dimension

Number of possible 32x32 binary images:

$2^{32x32} \approx 10^{308}$

Number of elementary particles in the visible universe: (source)

$\approx 10^{97}$

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