CS231n——Computer Vision

1. Course Introduction

Visual Data: Dark matter of the Internet

The field of computer vision is truly an interdisciplinary field, and it touches on many different areas of science and engineer and technology.

The history of vision

1600s the Renaissance period of time　camera obscura

1959　Hubel & Wiesel　the visual processing mechanism

1970　David Marr　Vision

- Input images: Perceived intensities
- Primal Sketch: Zero crossings, blobs, edges, bars, ends, virtual lines, groups, curves boundaries
- 2.5D Sketch: Local surface orientation and discontinuities in depth and in surface orientation
- 3-D Model Representation: 3-D models hierarchically organized in terms of surface and volumetric primitives

1979　Brooks & Binford　Generalized Cylinder

1973　Fischler and Elschlager　Pictorial Structure

- Object s composed of simple geometric primitives.

1987　David Lowe　Try to recognize razors by constructing lines and edges.

1997　Shi & Malik　Normalized Cut

1999　David Lowe　“SIFT” & Object Recognition

2001　Viola & Jones　Face Detection

2006　Lazebnik, Schmid & Ponce　Spatial Pyramid Matching

2005　Dalal& Trigas　Histogram of Gradients

2009　Felzenswalb, McAllester, Ramanan　Deformable Part Model

CS231-n overview

CS231-n focuses on one of the most important problems of visual recognition - image classification. There is a number of visual recognition problems that are related to image classification, such as object detection, image captioning.

Convolutional Neural Networks (CNN) have become an important tool for object recognition.

2.Image Classification

Image Classification: A core task in Computer Vision

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

The problem and challenge

- Semantic gap
- Viewpoint variation
- Illumination
- Deformation
- Occlusion
- Background Clutter
- Interclass variation

An image classifier

```
def classify_image(image):
    #Some magic here?
    return class_label
```

There is no obvious way to hard-code the algorithm for recognizing a cat or other classes.

We want to come up with some algorithm or some method for these recognition tasks, which scales much more naturally to all the variety of objects in the world.

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

```
def train(image, labels):
    #MAchine learning!
    return model

def predict(model, test_images):
    #Use model to predect labels
    return test_labels
```

First classifier: Nearest Neighbor

1. During the training, it just need to memorize all the data and labels.

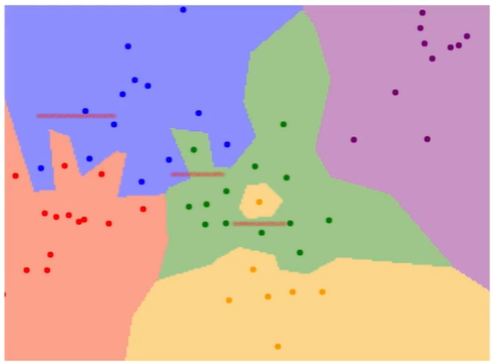
2. And then predict the label of the most similar training image.

Use the distance metric to compare images.

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

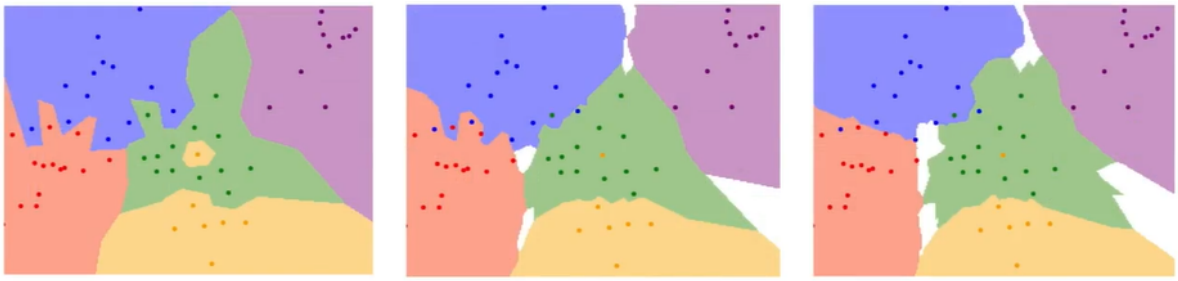
With N examples, the training is $O(1)$ and predicting is $O(N)$. This is bad, because we want classifiers that are fast at prediction; slow for training is ok.

the decision region



K-nearest neighbors

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



The K can smooth our boundaries and lead to better results.

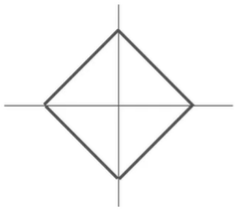
Distance Metric:

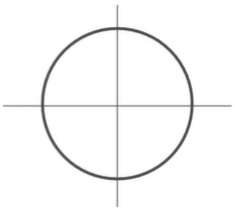
L1(Manhattan) distance: $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$

L2(Euclidean) distance: $d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$

Different distance metrics make different assumptions about the underlying geometry or topology that you'd expect in the space.

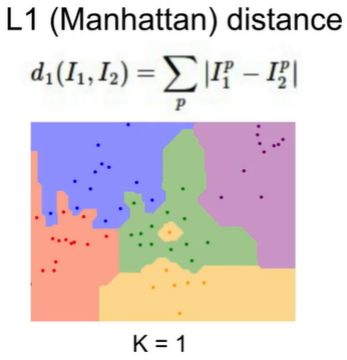
Each points in the two figures have the same distances to the original point. The first one use L1 distance and the second one using L2 distance.

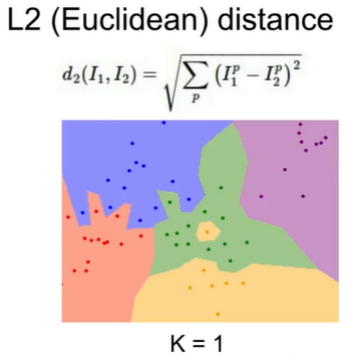




The L1 distance depends on the choice of your coordinate frame. So if you rotate the coordinate frame that would actually change the L1 distance between the points. Whereas change the frame that in the L2 distance doesn't matter.

You can use this algorithm on many different types of data.





Hyperparameters

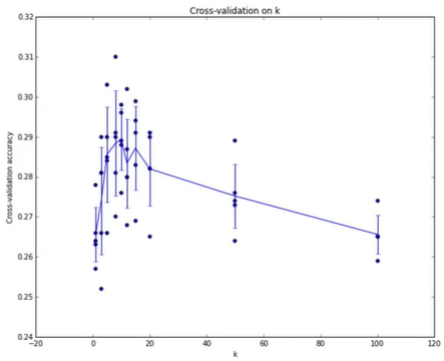
What is the best value of k to use? What is the best distance to use? These are hyperparameters: choices about the algorithm that we set rather than learn. This is very problem-dependent because we must try them all out and see what works best.

Setting hyperparameters

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

Split data into folds, try each fold as validation and average the results. This is useful for small datasets, but not used too frequently in deep learning.

Here is an example of 5-fold cross-validation for the value of K.



Each point is a single outcome, the line goes through the mean, and the bars indicated the standard deviation.

k-Nearest Neighbor on images never used.

- Very slow at test time.
- Distance metrics on pixels are not informative.
- Curse of dimensionality

Linear Classification

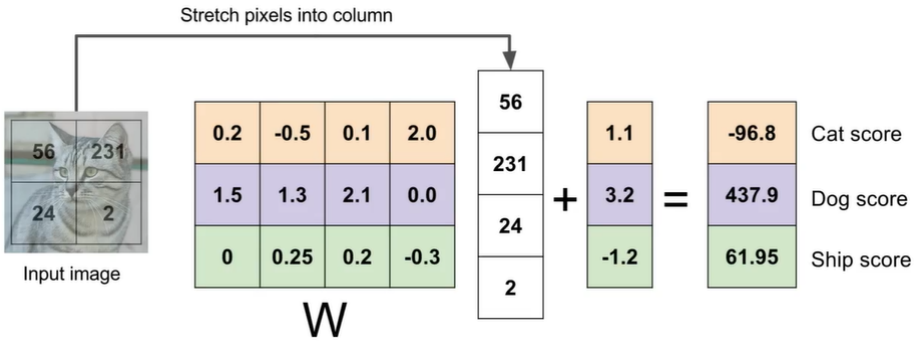
Parametric Approach: Linear Classifier

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

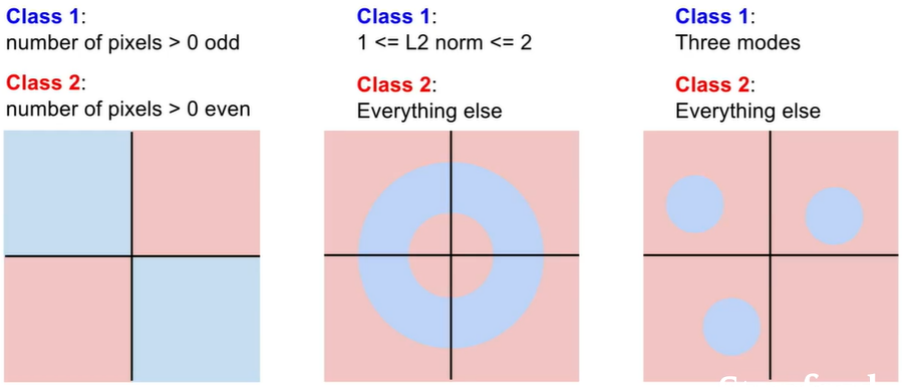
X is the image input, such as an array of $32 \times 32 \times 3$ numbers (3072 in total)

W are parameters or weights

Output is 10 numbers giving class scores.



Hard cases for a linear classifier



So how to choose the right W?

3. Loss Functions and Optimization