目录

[Image Classification 2](#_Toc56540854)

[Introduction 2](#_Toc56540855)

[Motivation 2](#_Toc56540856)

[Example 2](#_Toc56540857)

[Challenges 2](#_Toc56540858)

[The image classification pipeline 3](#_Toc56540859)

[Nearest Neighbor Classifier 4](#_Toc56540860)

[Knowledge 4](#_Toc56540861)

[Implementation in code 4](#_Toc56540862)

[Load data 4](#_Toc56540863)

[Train and evaluate a classifier 5](#_Toc56540864)

[Template 5](#_Toc56540865)

[L2 distance(optional) 6](#_Toc56540866)

[Knowledge 7](#_Toc56540867)

[Implementation in code 7](#_Toc56540868)

[k-Nearest Neighbor Classifier 6](#_Toc56540869)

[knowledge 6](#_Toc56540870)

[Hyperparameters 6](#_Toc56540871)

[Validation sets for Hyperparameter tuning 7](#_Toc56540872)

[Cross-validation 7](#_Toc56540873)

# Image Classification

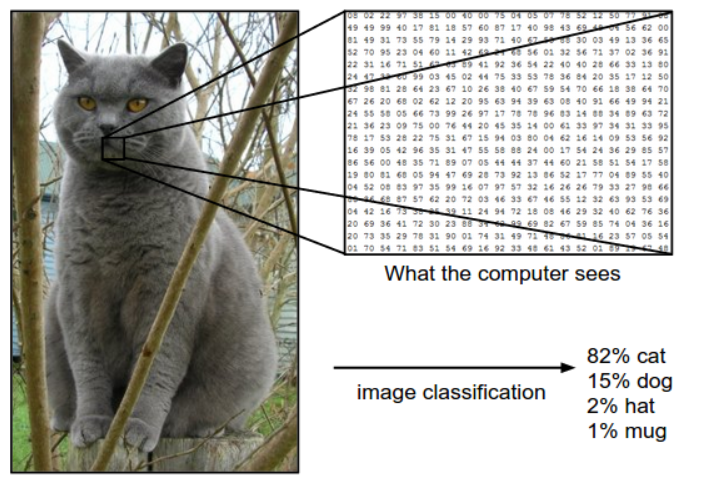
## Introduction

### Motivation

Assign an input image one label from a fixed set of categories--one of the core problems in Computer Vision.

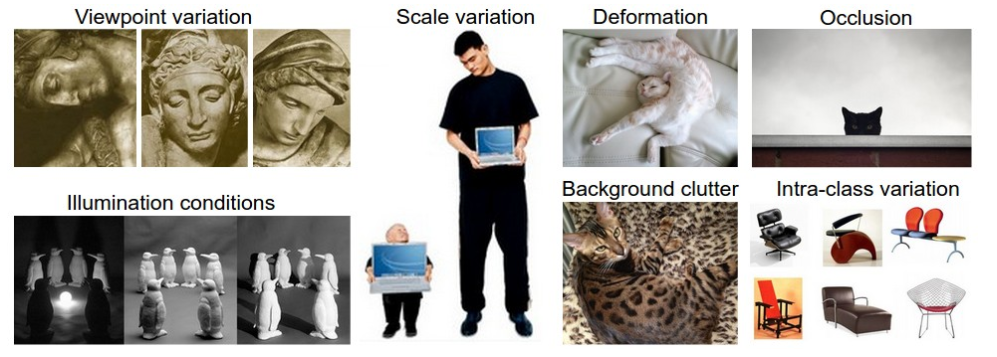
### Example

Predict a single label (or a distribution over labels as shown here to indicate our confidence) for a given image. Images are 3-dimensional arrays of integers from 0 to 255, of size Width x Height x 3. The 3 represents the three color channels Red, Green, Blue.(or RGB for short).



### Challenges

Image classification problem has a lot of challenges like illumination and viewpoints.



### ****The image classification pipeline****

**Input:** Our input consists of a set of *N* images, each labeled with one of *K* different classes. We refer to this data as the *training set*.

**Learning:** Our task is to use the training set to learn what every one of the classes looks like. We refer to this step as *training a classifier*, or *learning a model*.

**Evaluation:** In the end, we evaluate the quality of the classifier by asking it to predict labels for a new set of images that it has never seen before. We will then compare the true labels of these images to the ones predicted by the classifier. Intuitively, we’re hoping that a lot of the predictions match up with the true answers (which we call the *ground truth*).

## Nearest Neighbor Classifier

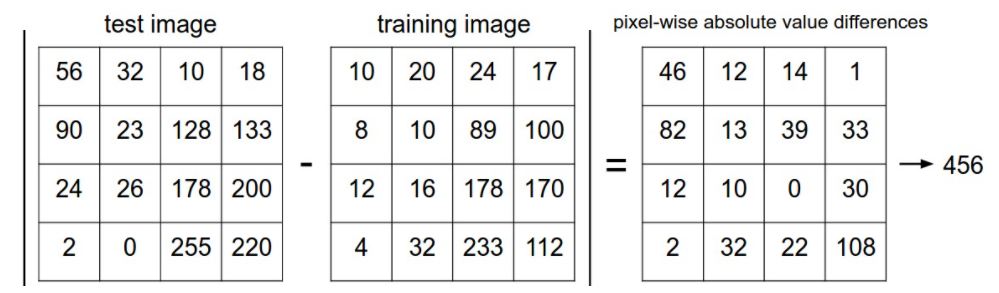
### ****Knowledge****

Have nothing to do with Convolutional Neural Networks and it is very rarely used in practice, but it will allow us to get an idea about the basic approach to an image classification problem.

Given two images and representing them as vectors

Compute L1 distance:

Visualization procedure:



### Implementation in code

#### Load data

First, let’s load the CIFAR-10 data into memory as 4 arrays: the training data/labels and the test data/labels. In the code below, Xtr (of size 50,000 x 32 x 32 x 3) holds all the images in the training set, and a corresponding 1-dimensional array Ytr (of length 50,000) holds the training labels (from 0 to 9):

Xtr, Ytr, Xte, Yte **=** load\_CIFAR10('data/cifar10/') *# a magic function we provide*

*# flatten out all images to be one-dimensional*

Xtr\_rows **=** Xtr.reshape(Xtr.shape[0], 32 **\*** 32 **\*** 3) *# Xtr\_rows becomes 50000 x 3072*

Xte\_rows **=** Xte.reshape(Xte.shape[0], 32 **\*** 32 **\*** 3) *# Xte\_rows becomes 10000 x 3072*

#### Train and evaluate a classifier

nn **=** NearestNeighbor() *# create a Nearest Neighbor classifier class*

nn.train(Xtr\_rows, Ytr) *# train the classifier on the training images and labels*

Yte\_predict **=** nn.predict(Xte\_rows) *# predict labels on the test images*

*# and now print the classification accuracy, which is the average number*

*# of examples that are correctly predicted (i.e. label matches)*

**print** 'accuracy: %f' **%** ( np.mean(Yte\_predict **==** Yte) )

#### Template

import numpy **as** np

**class** **NearestNeighbor**(object):

**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)*

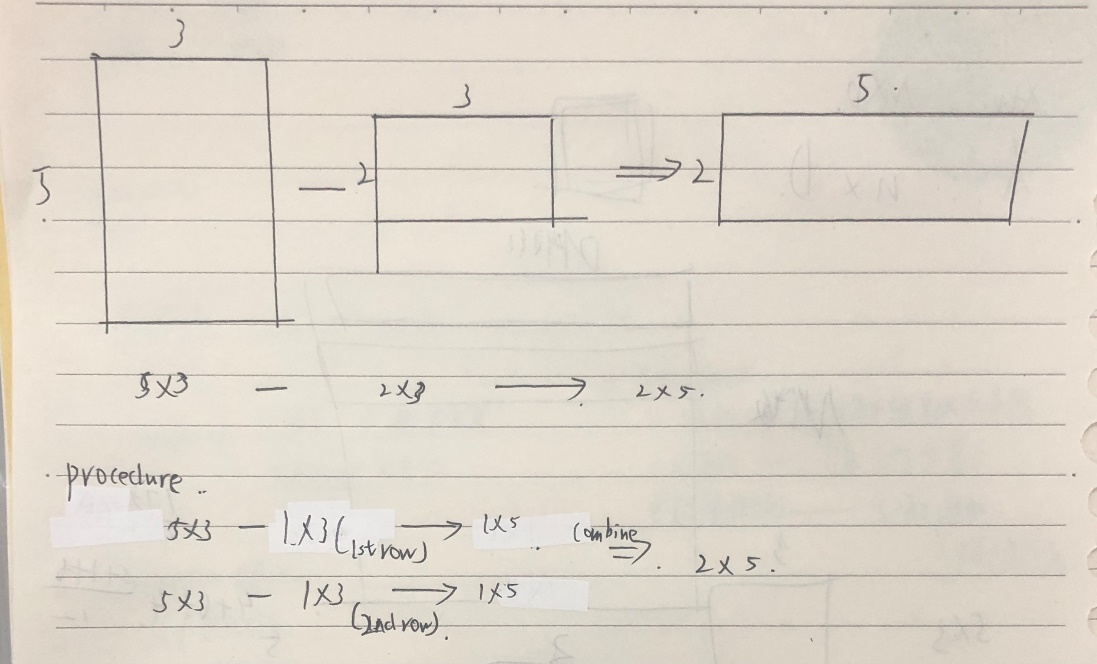
distances **=** np.sum(np.abs(self.Xtr **-** X[i,:]), axis **=** 1)

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

refer **test\_distance.py** to help understand the array



### L2 distance(optional)

#### Knowledge

#### Implementation in code

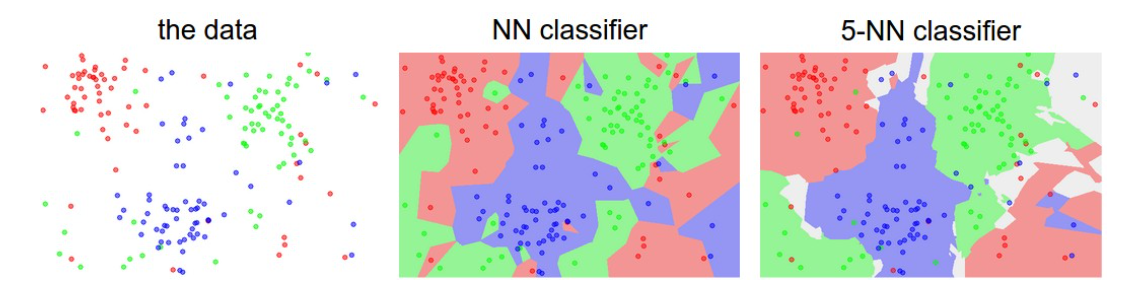
distances **=** np.sqrt(np.sum(np.square(self.Xtr **-** X[i,:]), axis **=** 1))

## k-Nearest Neighbor Classifier

### knowledge

Find the top **k** closest images, and have them vote on the label of the test image.

In particular, when *k = 1*, we recover the Nearest Neighbor classifier. Intuitively, higher values of **k** have a smoothing effect that makes the classifier more resistant to outliers



### Hyperparameters

k and the distance measure

## Validation sets for Hyperparameter tuning

**we cannot use the test set for the purpose of tweaking hyperparameters**.

Split our training set in two: a slightly smaller training set, and what we call a **validation set**.

*# assume we have Xtr\_rows, Ytr, Xte\_rows, Yte as before*

*# recall Xtr\_rows is 50,000 x 3072 matrix*

Xval\_rows **=** Xtr\_rows[:1000, :] *# take first 1000 for validation*

Yval **=** Ytr[:1000]

Xtr\_rows **=** Xtr\_rows[1000:, :] *# keep last 49,000 for train*

Ytr **=** Ytr[1000:]

*# find hyperparameters that work best on the validation set*

validation\_accuracies **=** []

**for** k **in** [1, 3, 5, 10, 20, 50, 100]:

*# use a particular value of k and evaluation on validation data*

nn **=** NearestNeighbor()

nn.train(Xtr\_rows, Ytr)

*# here we assume a modified NearestNeighbor class that can take a k as input*

Yval\_predict **=** nn.predict(Xval\_rows, k **=** k)

acc **=** np.mean(Yval\_predict **==** Yval)

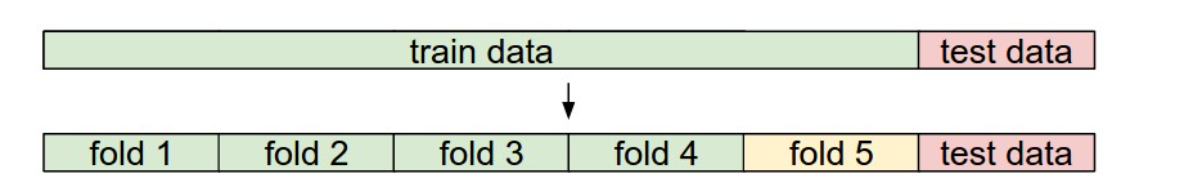
**print** 'accuracy: %f' **%** (acc,)

*# keep track of what works on the validation set*

validation\_accuracies.append((k, acc))

### **Cross-validation**

The size of training data (and therefore also the validation data) might be small.



Green-training set

Yellow-Validation fold

Iterate 5 times