**[深度学习与计算机视觉系列(2)\_图像分类与KNN](https://blog.csdn.net/yaoqiang2011/article/details/49949535)**

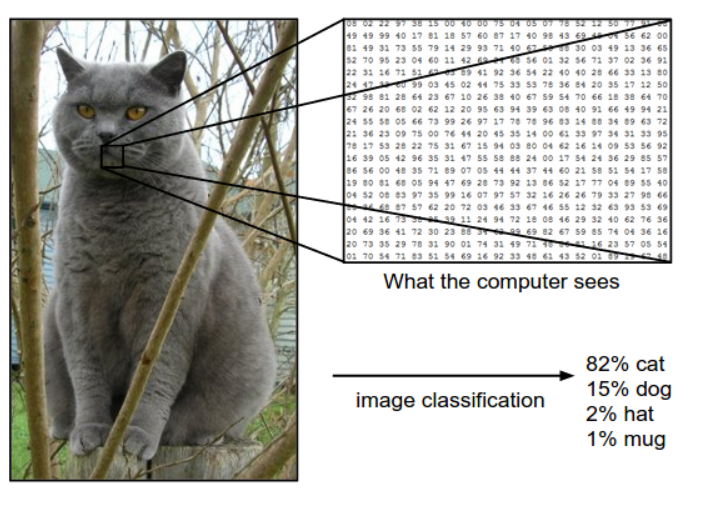
作者： [寒小阳](http://blog.csdn.net/han_xiaoyang?viewmode=contents)   
时间：2015年11月。   
出处：[http://blog.csdn.net/han\_xiaoyang/article/details/49949535](http://blog.csdn.net/han_xiaoyang/article/details/49949535" \t "_blank)   
声明：版权所有，转载请注明出处，谢谢

## 1. 图像分类问题

这是人每天自然而然会做的事情，普通到大部分时候，我们都感知不到我们在完成一个个这样的任务。早晨起床洗漱，你要看看洗漱台一堆东西中哪个是杯子，哪个是你的牙刷；吃早餐的时候你要分辨食物和碗碟…   
抽象一下，对于一张输入的图片，要判定它属于给定的一些**标签/类别**中的哪一个。看似很简单的一个问题，这么多年却一直是计算机视觉的一个核心问题，应用场景也很多。它的重要性还体现在，其实其他的一些计算机视觉的问题(比如说物体定位和识别、图像内容分割等)都可以基于它去完成。

咱们举个例子从机器学习的角度描述一下这个问题^\_^

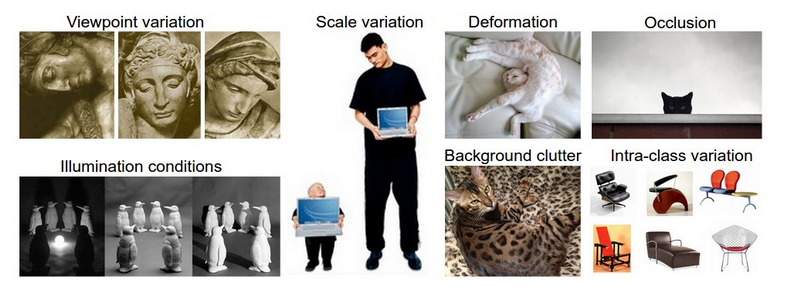
计算机拿到一张图片(如下图所示)，然后需要给出它对应{猫，狗，帽子，杯子}4类的概率。人类是灰常牛逼的生物，我们一瞥就知道这货是猫。然而对计算机而言，他们是没办法像人一样『看』到整张图片的。对它而言，这是一个3维的大矩阵，包含248\*400个像素点，每个像素点又有红绿蓝(RGB)3个颜色通道的值，每个值在0(黑)-255(白)之间，计算机就需要根据这248\*400\*3=297600个数值去判定这货是『猫』



### 1.1 图像识别的难点

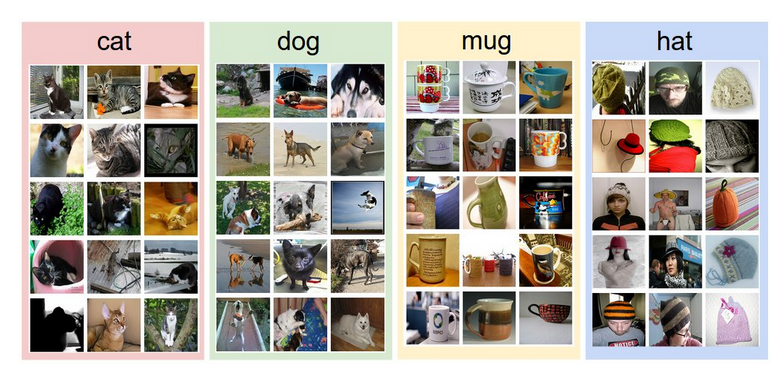
图像识别看似很直接。但实际上包含很多挑战，人类可是经过数亿年的进化才获得如此强大的大脑，对于各种物体有着精准的视觉理解力。总体而言，我们想『教』会计算机去认识一类图，会有下面这样一些困难：

* **视角不同**，每个事物旋转或者侧视最后的构图都完全不同
* **尺寸大小不统一**，相同内容的图片也可大可小
* **变形**，很多东西处于特殊的情形下，会有特殊的摆放和形状
* **光影等干扰/幻象**
* **背景干扰**
* **同类内的差异(比如椅子有靠椅/吧椅/餐椅/躺椅…)**



### 1.2 识别的途径

首先，大家想想就知道，这个算法并不像『对一个数组排序』或者『求有向图的最短路径』，我们没办法提前制定一个流程和规则去解决。定义『猫』这种动物本身就是一件很难的事情了，更不要说去定义一只猫在图像上的固定表现形式。所以我们寄希望于机器学习，使用『Data-driven approach/数据驱动法』来做做尝试。简单说来，就是对于每个类别，我们都找一定量的图片数据，『喂』给计算机，让它自己去『学习和总结』每一类的图片的特点。对了，这个过程和小盆友学习新鲜事物是一样一样的。『喂』给计算机学习的图片数据就和下图的猫/狗/杯子/帽子一样：



### 1.3 机器学习解决图像分类的流程/Pipeline

整体的流程和普通机器学习完全一致，简单说来，也就下面三步：

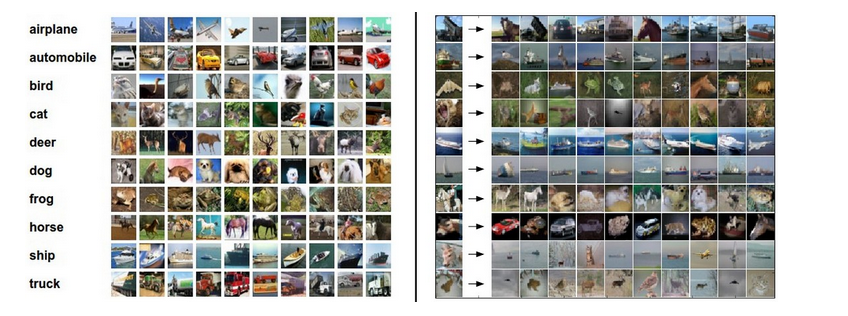
* **输入**：我们的给定K个类别的N张图片，作为计算机学习的训练集
* **学习**：让计算机逐张图片地『观察』和『学习』
* **评估**：就像我们上学学了东西要考试检测一样，我们也得考考计算机学得如何，于是我们给定一些计算机不知道类别的图片让它判别，然后再比对我们已知的正确答案。

## 2. 最近邻分类器(Nearest Neighbor Classifier)

先从简单的方法开始说，先提一提**最近邻分类器/Nearest Neighbor Classifier**，不过事先申明，它和深度学习中的卷积神经网/Convolutional Neural Networks其实一点关系都没有，我们只是从基础到前沿一点一点推进，最近邻是图像识别一个相对简单和基础的实现方式。

### 2.1 CIFAR-10

[CIFAR-10](http://www.cs.toronto.edu/~kriz/cifar.html)是一个非常常用的图像分类数据集。数据集包含60000张32\*32像素的小图片，每张图片都有一个类别标注(总共有10类)，分成了50000张的训练集和10000张的测试集。如下是一些图片示例：



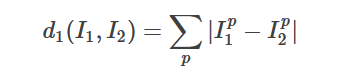
上图中左边是十个类别和对应的一些示例图片，右边是给定一张图片后，根据像素距离计算出来的，最近的10张图片。

### 2.2 基于最近邻的简单图像类别判定

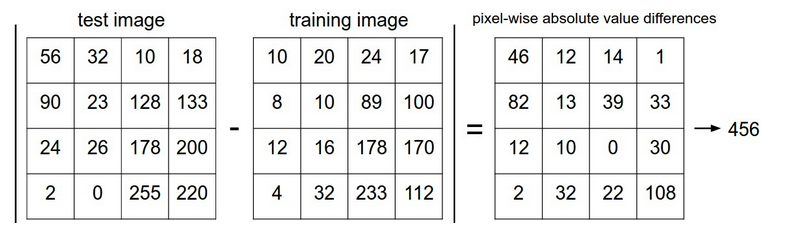
假如现在用CIFAR-10数据集做训练集，判断一张未知的图片属于CIFAR-10中的哪一类，应该怎么做呢。一个很直观的想法就是，既然我们现在有每个像素点的值，那我们就根据输入图片的这些值，计算和训练集中的图片距离，找最近的图片的类别，作为它的类别，不就行了吗。

恩，想法很直接，这就是『最近邻』的思想。偷偷说一句，这种直接的做法在图像识别中，其实效果并不是特别好。比如上图是按照这个思想找的最近邻，其实只有3个图片的最近邻是正确的类目。

即使这样，作为最基础的方法，还是得掌握，我们来简单实现一下吧。我们需要一个图像距离评定准则，比如最简单的方式就是，比对两个图像像素向量之间的l1距离(也叫曼哈顿距离/cityblock距离)，公式如下：



其实就是计算了所有像素点之间的差值，然后做了加法，直观的理解如下图：



我们先把数据集读进内存：

1. #! /usr/bin/env python
2. #coding=utf-8
3. **import** os
4. **import** sys
5. **import** numpy as np
7. **def** load\_CIFAR\_batch(filename):
8. """
9. cifar-10数据集是分batch存储的，这是载入单个batch
11. @参数 filename: cifar文件名
12. @r返回值: X, Y: cifar batch中的 data 和 labels
13. """
15. with open(filename, 'r') as f:
16. datadict=pickle.load(f)
18. X=datadict['data']
19. Y=datadict['labels']
21. X=X.reshape(10000, 3, 32, 32).transpose(0,2,3,1).astype("float")
22. Y=np.array(Y)
24. **return** X, Y

27. **def** load\_CIFAR10(ROOT):
28. """
29. 读取载入整个 CIFAR-10 数据集
31. @参数 ROOT: 根目录名
32. @return: X\_train, Y\_train: 训练集 data 和 labels
33. X\_test, Y\_test: 测试集 data 和 labels
34. """
36. xs=[]
37. ys=[]
39. **for** b **in** range(1,6):
40. f=os.path.join(ROOT, "data\_batch\_%d" % (b, ))
41. X, Y=load\_CIFAR\_batch(f)
42. xs.append(X)
43. ys.append(Y)
45. X\_train=np.concatenate(xs)
46. Y\_train=np.concatenate(ys)
48. **del** X, Y
50. X\_test, Y\_test=load\_CIFAR\_batch(os.path.join(ROOT, "test\_batch"))
52. **return** X\_train, Y\_train, X\_test, Y\_test
54. # 载入训练和测试数据集
55. X\_train, Y\_train, X\_test, Y\_test = load\_CIFAR10('data/cifar10/')
56. # 把32\*32\*3的多维数组展平
57. Xtr\_rows = X\_train.reshape(X\_train.shape[0], 32 \* 32 \* 3) # Xtr\_rows : 50000 x 3072
58. Xte\_rows = X\_test.reshape(X\_test.shape[0], 32 \* 32 \* 3) # Xte\_rows : 10000 x 3072

下面我们实现最近邻的思路：

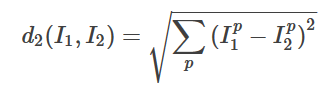
1. **class** NearestNeighbor:
2. **def** \_\_init\_\_(self):
3. **pass**
5. **def** train(self, X, y):
6. """
7. 这个地方的训练其实就是把所有的已有图片读取进来 -\_-||
8. """
9. # the nearest neighbor classifier simply remembers all the training data
10. self.Xtr = X
11. self.ytr = y
13. **def** predict(self, X):
14. """
15. 所谓的预测过程其实就是扫描所有训练集中的图片，计算距离，取最小的距离对应图片的类目
16. """
17. num\_test = X.shape[0]
18. # 要保证维度一致哦
19. Ypred = np.zeros(num\_test, dtype = self.ytr.dtype)
21. # 把训练集扫一遍 -\_-||
22. **for** i **in** xrange(num\_test):
23. # 计算l1距离，并找到最近的图片
24. distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
25. min\_index = np.argmin(distances) # 取最近图片的下标
26. Ypred[i] = self.ytr[min\_index] # 记录下label
28. **return** Ypred
30. nn = NearestNeighbor() # 初始化一个最近邻对象
31. nn.train(Xtr\_rows, Y\_train) # 训练...其实就是读取训练集
32. Yte\_predict = nn.predict(Xte\_rows) # 预测
33. # 比对标准答案，计算准确率
34. **print** 'accuracy: %f' % ( np.mean(Yte\_predict == Y\_test) )

最近邻的思想在CIFAR上得到的准确度为**38.6%**，我们知道10各类别，我们随机猜测的话准确率差不多是1/10=10%，所以说还是有识别效果的，但是显然这距离人的识别准确率(94%)实在是低太多了，不那么实用。

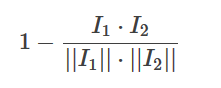
### 2.3 关于最近邻的距离准则

我们这里用的距离准则是l1距离，实际上除掉l1距离，我们还有很多其他的距离准则。

* 比如说l2距离(也就是大家熟知的欧氏距离)的计算准则如下：



* 比如余弦距离计算准则如下：



更多的距离准则可以参见[scipy相关计算页面](http://docs.scipy.org/doc/scipy-0.16.0/reference/generated/scipy.spatial.distance.pdist.html" \t "_blank).

## 3. K最近邻分类器(K Nearest Neighbor Classifier)

这是对最近邻的思想的一个调整。其实我们在使用最近邻分类器分类，扫描CIFAR训练集的时候，会发现，有时候不一定距离最近的和当前图片是同类，但是最近的一些里有很多和当前图片是同类。所以我们自然而然想到，把最近邻扩展为最近的N个临近点，然后统计一下这些点的类目分布，取最多的那个类目作为自己的类别。

恩，这就是KNN的思想。

KNN其实是一种特别常用的分类算法。但是有个问题，我们的K值应该取多少呢。换句话说，我们找多少邻居来投票，比较靠谱呢？

### 3.1 交叉验证与参数选择

在现在的场景下，假如我们确定使用KNN来完成图片类别识别问题。我们发现有一些参数是肯定会影响最后的识别结果的，比如：

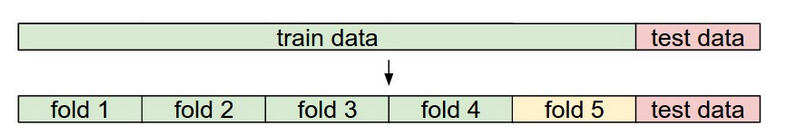
* 距离的选择(l1,l2,cos等等)
* 近邻个数K的取值。

每组参数下其实都能产生一个新的model，所以这可以视为一个模型选择/model selection问题。而对于模型选择问题，最常用的办法就是在[交叉验证](https://en.wikipedia.org/wiki/Cross-validation_%28statistics%29" \t "_blank)集上实验。

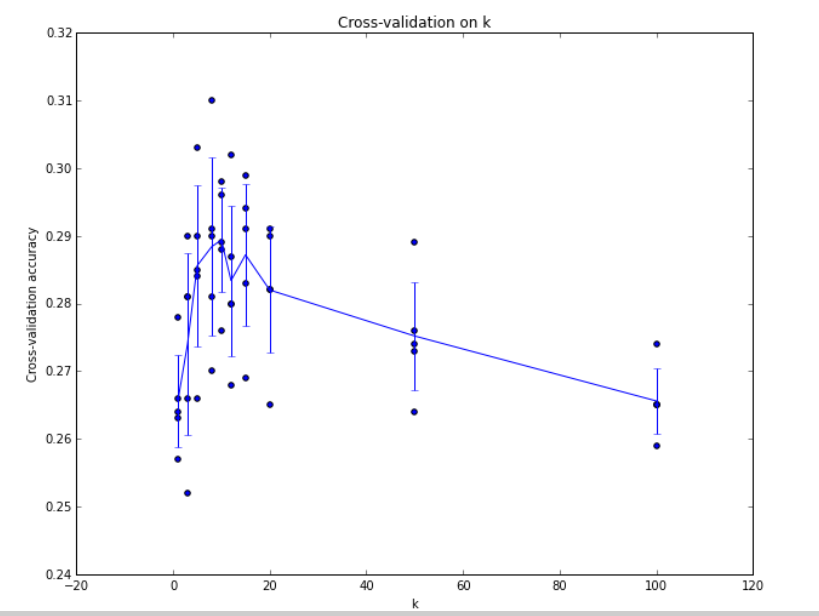
数据总量就那么多，如果我们在test data上做模型参数选择，又用它做效果评估，显然不是那么合理（因为我们的模型参数很有可能是在test data上过拟合的，不能很公正地评估结果）。所以我们通常会把训练数据分为两个部分，一大部分作为训练用，另外一部分就是所谓的cross validation数据集，用来进行模型参数选择的。比如说我们有50000训练图片，我们可以把它分为49000的训练集和1000的交叉验证集。

1. # 假定已经有Xtr\_rows, Ytr, Xte\_rows, Yte了，其中Xtr\_rows为50000\*3072 矩阵
2. Xval\_rows = Xtr\_rows[:1000, :] # 构建1000的交叉验证集
3. Yval = Ytr[:1000]
4. Xtr\_rows = Xtr\_rows[1000:, :] # 保留49000的训练集
5. Ytr = Ytr[1000:]
7. # 设置一些k值，用于试验
8. validation\_accuracies = []
9. **for** k **in** [1, 3, 5, 7, 10, 20, 50, 100]:
11. # 初始化对象
12. nn = NearestNeighbor()
13. nn.train(Xtr\_rows, Ytr)
14. # 修改一下predict函数，接受 k 作为参数
15. Yval\_predict = nn.predict(Xval\_rows, k = k)
16. acc = np.mean(Yval\_predict == Yval)
17. **print** 'accuracy: %f' % (acc,)
19. # 输出结果
20. validation\_accuracies.append((k, acc))

这里提一个在很多地方会看到的概念，叫做k-fold cross-validation，意思其实就是把原始数据分成k份，轮流使用其中k-1份作为训练数据，而剩余的1份作为交叉验证数据(因此其实对于k-fold cross-validation我们会得到k个accuracy)。以下是5-fold cross-validation的一个示例：



以下是我们使用5-fold cross-validation，取不同的k值时，得到的accuracy曲线(补充一下，因为是5-fold cross-validation，所以在每个k值上有5个取值，我们取其均值作为此时的准确度)



可以看出大概在k=7左右有最佳的准确度。

### 3.2 最近邻方法的优缺点

K最近邻的优点大家都看出来了，思路非常简单清晰，而且完全不需要训练…不过也正因为如此，最后的predict过程非常耗时，因为要和全部训练集中的图片比对一遍。

实际应用中，我们其实更加关心实施predict所消耗的时间，如果有一个图像识别app返回结果要半小时一小时，你一定第一时间把它卸了。我们反倒不那么在乎训练时长，训练时间稍微长一点没关系，只要最后应用的时候识别速度快效果好，就很赞。后面会提到的深度神经网络就是这样，深度神经网络解决图像问题时训练是一个相对耗时间的过程，但是识别的过程非常快。

另外，不得不多说一句的是，优化计算K最近邻时间问题，实际上依旧到现在都是一个非常热门的问题。**Approximate Nearest Neighbor (ANN)**算法是牺牲掉一小部分的准确度，而提高很大程度的速度，能比较快地找到近似的K最近邻，现在已经有很多这样的库，比如说[FLANN](http://www.cs.ubc.ca/research/flann/).

最后，我们用一张图来说明一下，用图片像素级别的距离来实现图像类别识别，有其不足之处，我们用一个叫做[t-SNE](http://lvdmaaten.github.io/tsne/)的技术把CIFAR-10的所有图片按两个维度平铺出来，靠得越近的图片表示其像素级别的距离越接近。然而我们瞄一眼，发现，其实靠得最近的并不一定是同类别的。

其实观察一下，你就会发现，像素级别接近的图片，在整张图的颜色分布上，有很大的共性，然而在图像内容上，有时候也只能无奈地呵呵嗒，毕竟颜色分布相同的不同物体也是非常多的。

## 参考资料与原文

[cs231n 图像分类与KNN](http://cs231n.github.io/classification/)