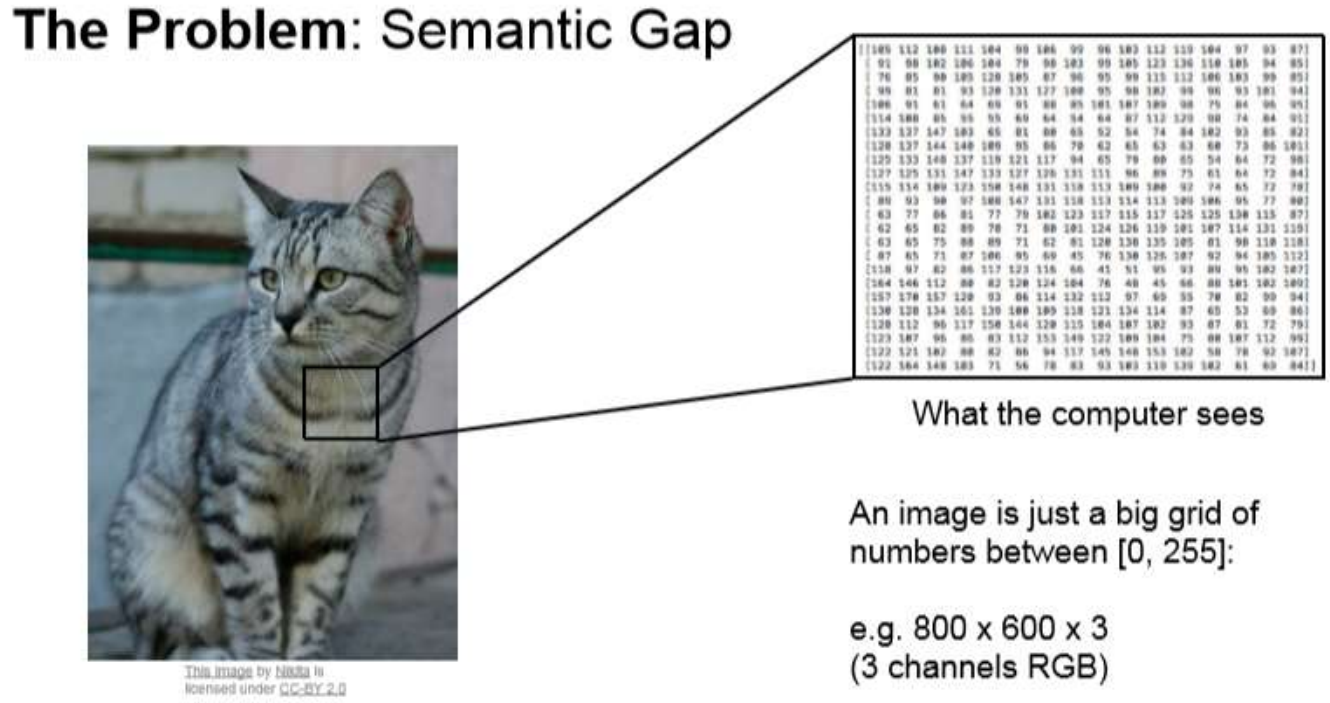
Lecture 2 Image Classification pipeline

Classification's Challenges

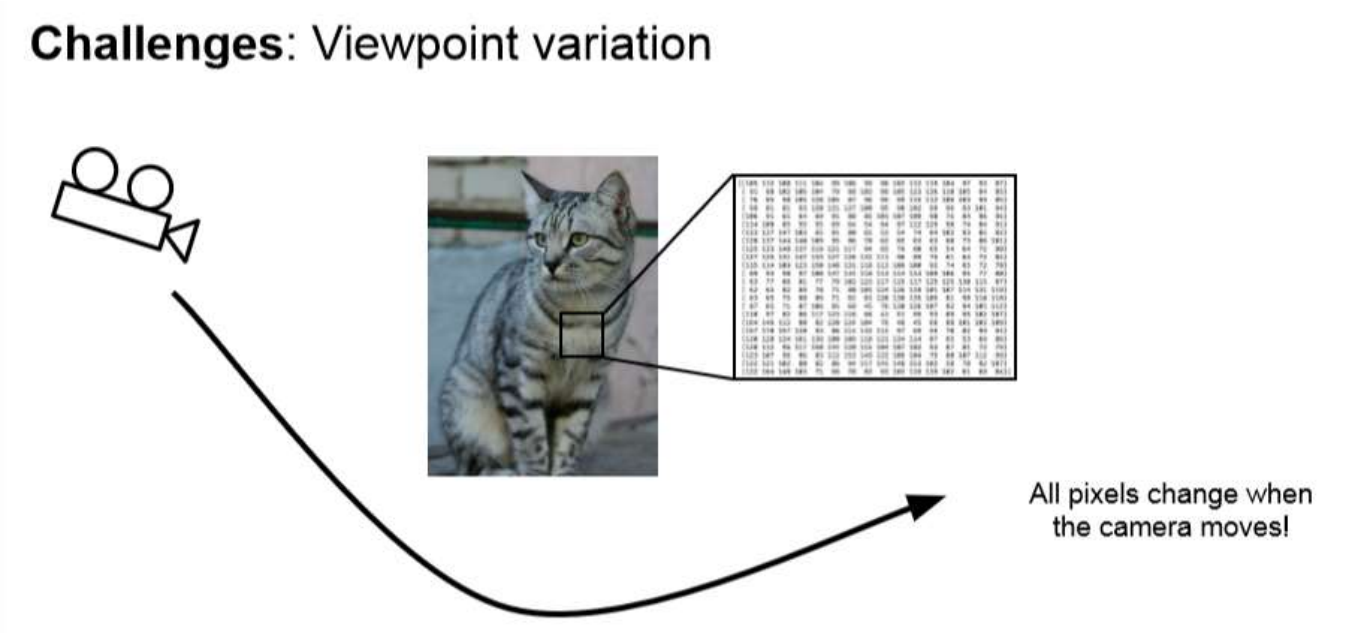
The Problem: Semantic Gap



每个图片都是由很多像素点组成的。就像主讲人说的，图片对于计算机来说就是**a gigantic grid of numbers**

所以Sematic Gap的意义就是我们给图片标的语义标签与计算机实际看到的像素值之间有巨大的差距。

Viewpoint variation



同一物体的不同角度得到的像素值序列是不一样的，但是是表示同样的物体，所以我们的算法就必须对**视角这样的问题有较强的鲁棒性rubust**

Illumination

不同的光照条件也会影响，所以我们的算法也需要对这样的情况就要较强的鲁棒性。

Deformation(变形)

Our algorithms should be robust to these different kinds of transforms.

Occlusion(遮挡)

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image by Jonsson is licensed under CC-BY 2.0

Background Clutter

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

猫身上的纹理或者猫本身的颜色与背景非常相似，那么我们的算法也需要对这种情况作出正确的判断。

Intraclass variation

This image is CC0 1.0 public domain

类内的差距，比如说同样都是猫，但是不同的猫具有不一样的颜色或者纹理什么的。

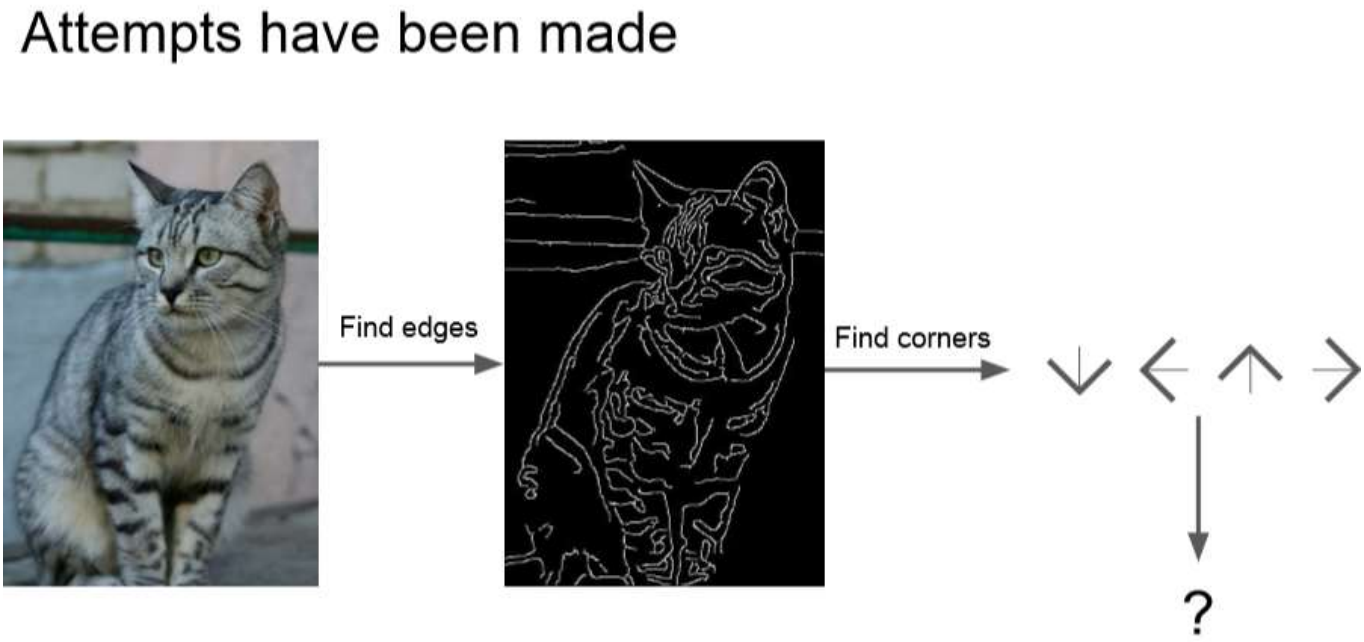
处理过程

An image classifier

图像分类的时候可以做一个这样的API接口用于接收图像并返回图像的标签。

同样，并没有现成的函数或者算法可以做到直接去classification

Attempts have been made

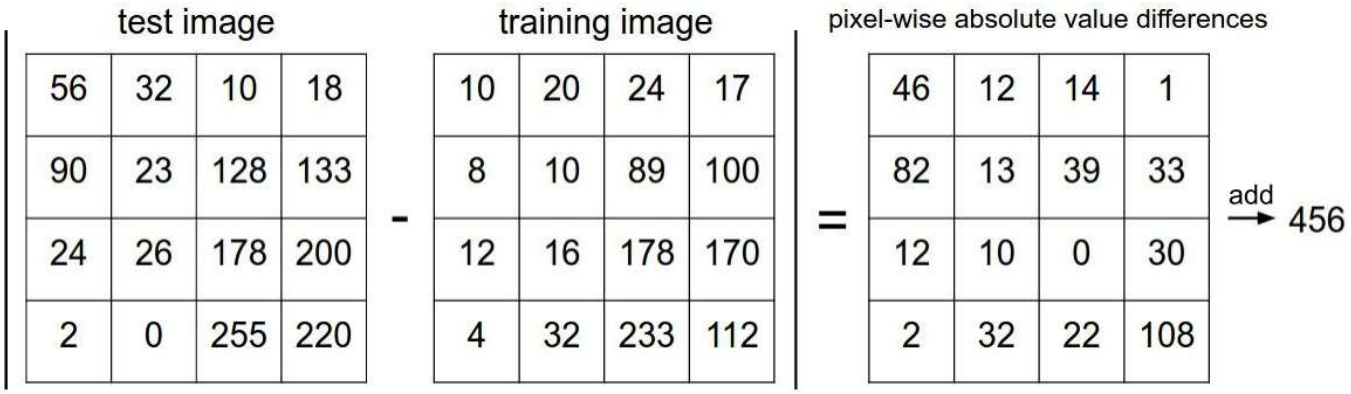


一些图像预处理的方法，比如就是想图中做的，先用函数算出边缘，然后得到某些线或者点的交汇方式并作出标签，但是这样的方法并不是一个特别好的方法。

曼哈顿距离和欧几里得距离

L1 distance

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



这里的曼哈顿距离主要在意的是测试图片的每个像素与训练图片的对应位置像素的差距。这是一种简单的最临近分类器。

```
import numpy as np

class NearestNeighbor:
  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 xrange(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
```

上图展示了最临近分类器的计算操作，基于的是曼哈顿距离的理论，训练过程主要是记录训练图片每个像素的值，而测试过程是将测试图片的每个像素与训练的model做比较。**这样的结果就是，训练的时间非常短，而测试的时间非常长，这样其实在应用中是一种很难推广的方法。**

两者的理论定义

欧几里得距离，欧氏距离，也就是我们熟知的距离，可扩展至m维

2维：d=sqrt((x1-x2)2+(y1-y2)2)

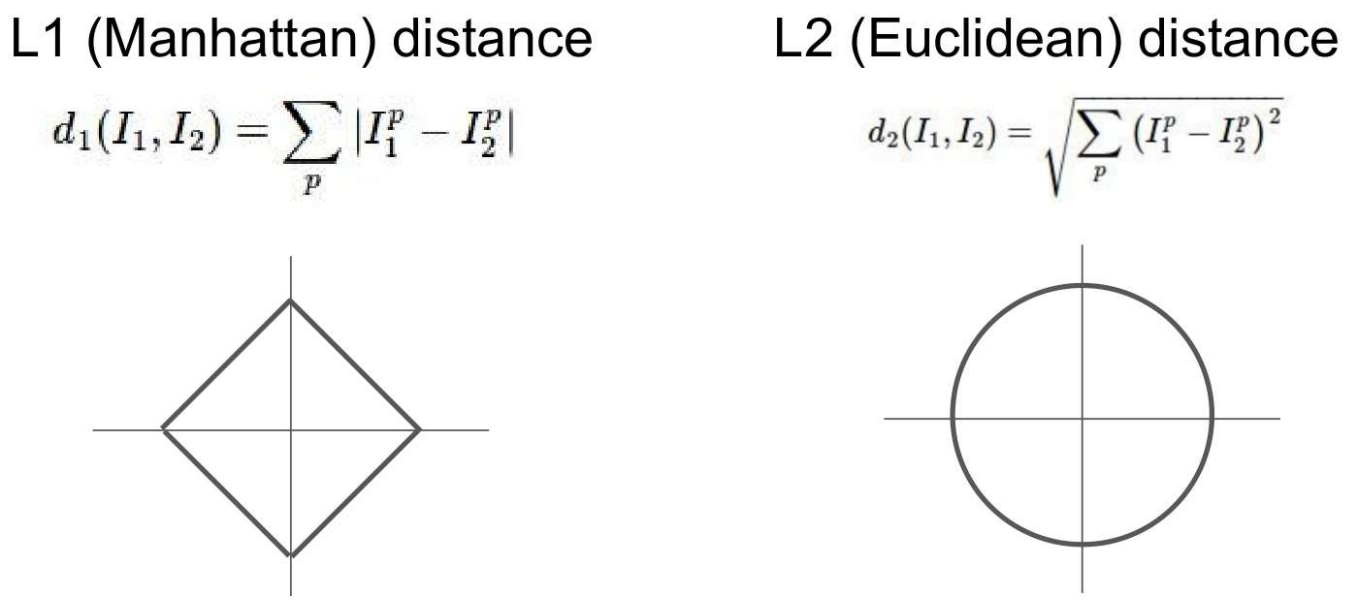
3维：d=sqrt((x1-x2)2+(y1-y2)2+(z1-z2)2)

m维：d=sqrt(∑(xi,1-xi,2)2)

曼哈顿距离，出租车距离，在一个横竖分布的网格上，两点之间的距离即为曼哈顿距离

也就是说只能横着走或者竖着走

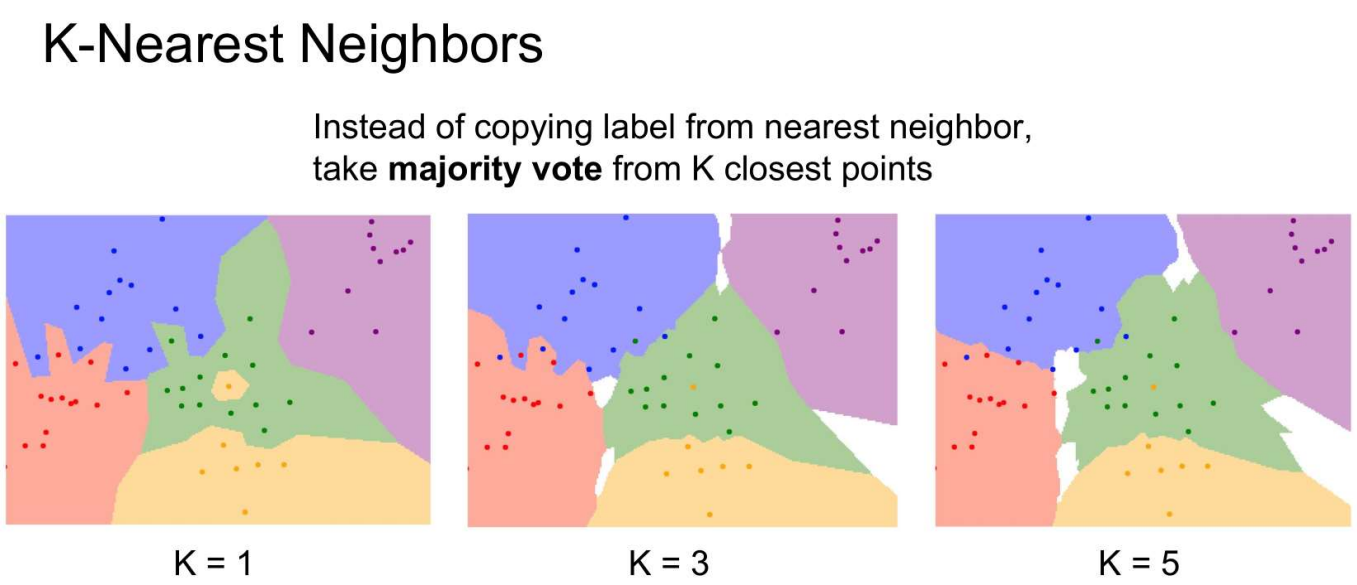
d=|x1-x2|+|y1-y2|



这里主要的问题在于，欧几里得距离对于任何的特征都没有偏向，所以如果我们在做分类的时候并没有去有特定的考虑的话，可以用欧几里得距离。

但是曼哈顿距离就完全不同了，他其实相当于有所偏向的欧几里得距离，也就是说某些特征比另外一些特征更加对距离有限制，所以，如果在分类的过程中对某些特征比较看重的话，是需要用曼哈顿距离的。

K最临近分类



这里用的方法是**majority vote**最大值投票的方法。

超参数——Hyperparameters

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

选择超参数的三种方式：

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data

Your Dataset

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

train | test

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

Better!

train | validation | test

显然第一和二种是不合理的：

- 对于第一种来说，在训练阶段去设计超参数并不会确认你的超参数适应新的数据集，更不用说你连测试都没有测试过。
- 对于第二种来说，显然会稍微好一点，但是仍存在着非常多的弊端，首先，你讲一部分的训练集转变为测试集，虽然在一定程度上加强了超参的可靠性，但是不得不承认这样与第一种方法其实没有什么区别，你的超参在test上表现的再好也不能标明你一定会在新的测试集上表现的很出色。
- 对于第三种方式，显然是最好的，首先，我们将数据集分为三个部分，训练之后首先会在validation上做验证，以保证超参的可靠性，然后在测试集中做测试，这样加一步会提高超参的可靠性。

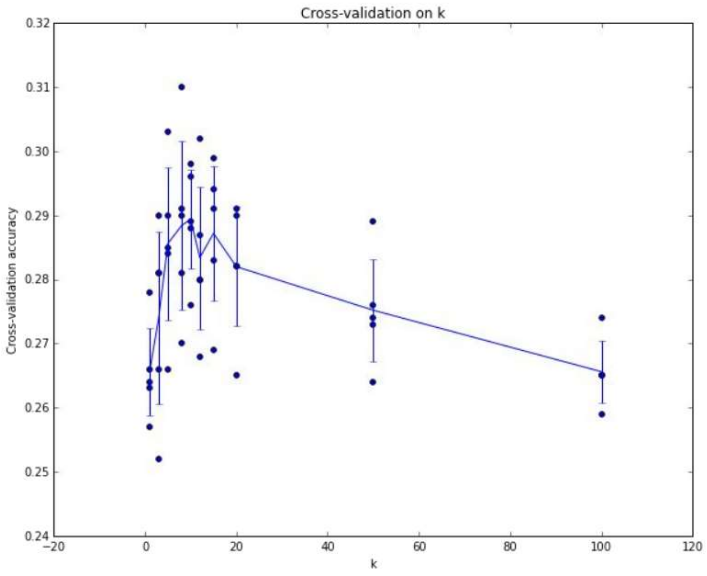
选择超参数的第四种方式——交叉验证

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

fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

交叉验证也就是将同一数据集分成多个fold，每次将不同的fold作为验证集做交叉验证，这样虽然造成大量的计算量，但是相对来说是非常棒的方法。

Setting Hyperparameters



Example of
5-fold cross-validation
for the value of **k.**

Each point: single outcome.

The line goes through the mean, bars indicated standard deviation

(Seems that k ~= 7 works best for this data)

这里展示的是交叉验证的结果，可以看出，就像图中说的，k大概是7的时候是最好的结果。每次我们用数据集做五次交叉验证，得到平均的准确度。

K-Nearest Neighbor 不常使用的原因

第一，在测试过程中花费时间太长

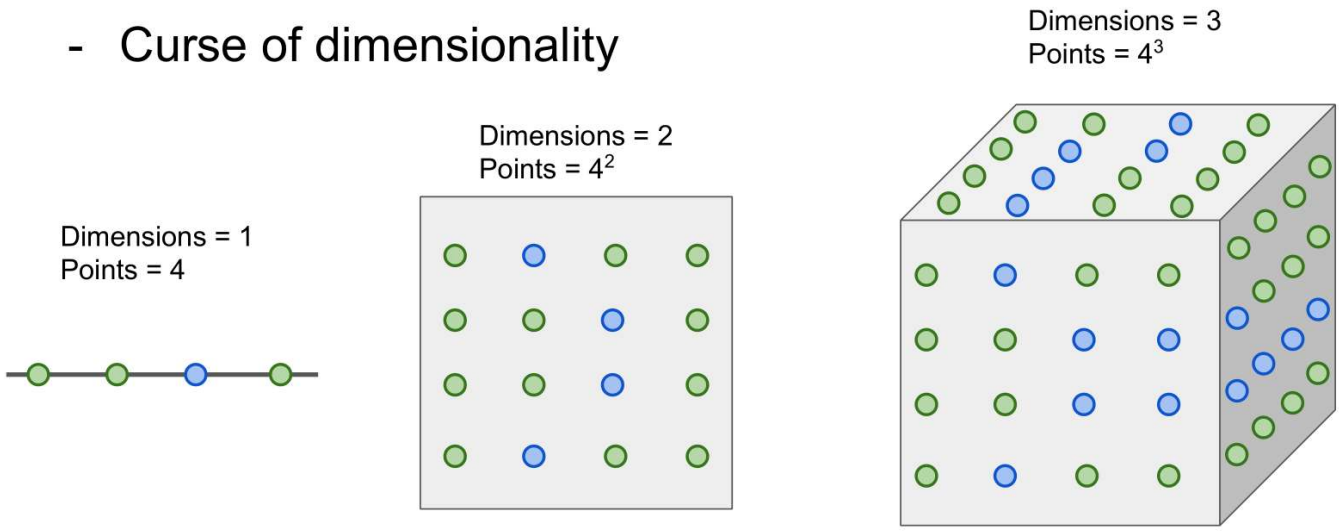
第二，像素点的距离值运算并不一定informatric——不一定有足够的信息。

mage is
ic domain

(all 3 images have same L2 distance to the one on the left)

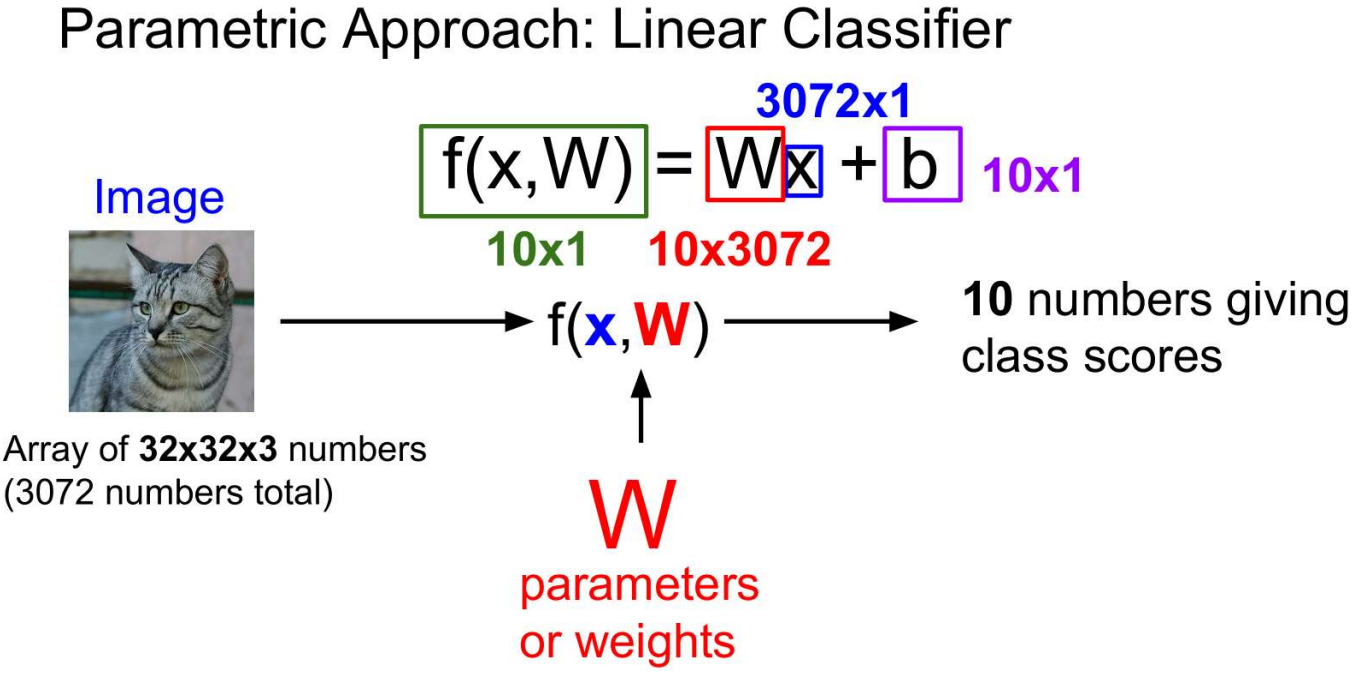
像上面的四张图，是完全不相同的四张图，但是在算L2距离的时候，也就是欧几里得距离的时候，因为不考虑坐标的问题，所以其实这四张图的L2距离是完全一样的，这样可以看出，L2距离并不是合理的图像分类的算法。

第三，K近邻算法随着维度的增加，需要的数据量是平方次增加。



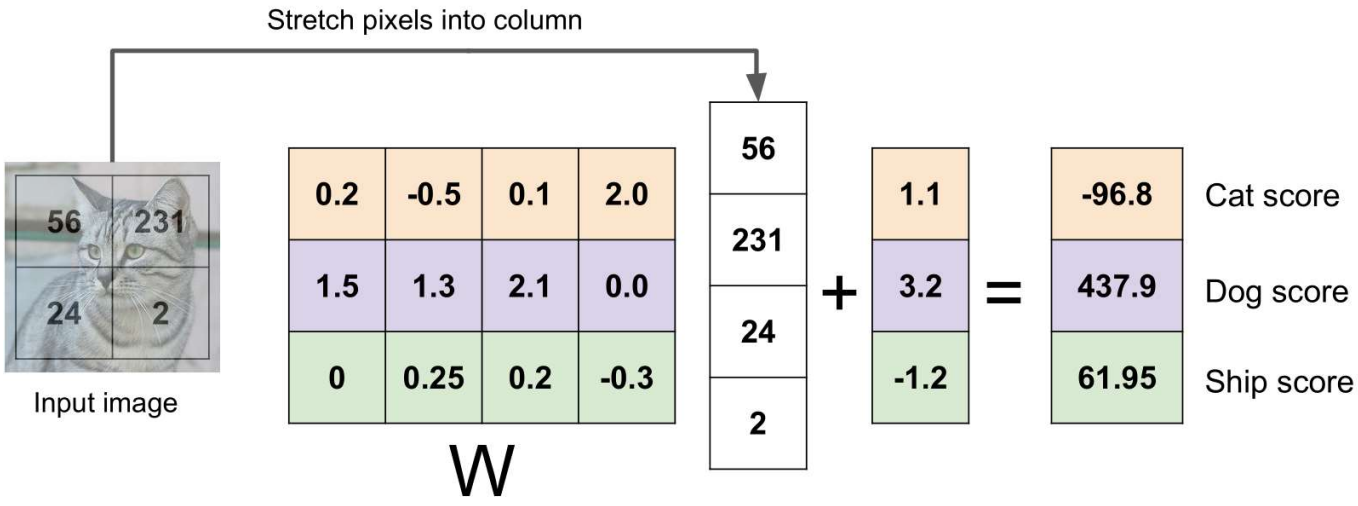
Linear Classification——线性分类

新的test的方式——权重model



也就是说，我们每次训练只需要训练出啦权重，然后训练部分就废掉了，不用管了，在测试的过程中只需要权重参数就可以解决所有问题。

另，上图猫的三个通道指的是RGB三通道。


Stretch pixels into column
56
231
24
2
Input image
0.2
-0.5
0.1
2.0
1.5
1.3
2.1
0.0
0
0.25
0.2
-0.3
W
56
231
24
2
+
1.1
3.2
-1.2
=
-96.8
437.9
61.95
Cat score
Dog score
Ship score