

# 20180411图像预处理中的特征提取

- 1- 图像预处理，可以对图像进行Otsu 最小二乘，二值化，转成向量（如1024 纬度）
- 2- 颜色特征提取：
  - 1- 灰度直方图，直接打印灰度的统计结果
  - 2- HOG 特征：先灰度化，Gamma矫正（调整亮度），梯度方向分为9 个区间，每个单元在各个方向进行直方图统计【计算各个方向的梯度 和 梯度方向】（一个9 维的特征向量），每相邻4个单元（每个单元3x3）构成一个块【一个块有  $9 \times 4 = 36$  维特征向量】，然后再移动一个3x3 的单元。最后把 所有特征都连起来。
  - 3- LBP 特征：先把图像转为灰度图，把图像分成3\*3的块，通过中心相对的高低来计算对应的纹理凹凸值，这样就能把一块块的纹理给计算出来了。【主要是计算纹理的】
  - 4- Haar-like（边缘特征检测）：白色块的所有像素之和 - 黑色块的所有像素之和。和AdaBoost 一起用。
  - 5- 角点特征和轮廓特征：计算机图形学中有重要地位。

原文链接：[https://blog.csdn.net/qq\\_31622541/article/details/78227358](https://blog.csdn.net/qq_31622541/article/details/78227358)

我们知道，在机器学习的算法实现之前，我们首先要提取图像的特征，将图像变成一个个向量，只有这样，图像才能被计算机学习，准确说，只有从我们人类概念下的图像变成离散型的变量，然后给离散型的变量赋予具体的含义，这样才能够借助计算机来实现接下来的机器学习之类的工作，为了提高学习效率，我们依然可以学习一些图形学内处理的方式，这样就可以大大的提升开发效率，可以说，现在大部分的重复性工作在于前人故意的埋坑，这里就不具体说明了。

接下来，我们借助第一个例子来说明图像预处理：

Kaggle竞赛入门 Digital recognition：

这里采用的预处理方式相当暴力，适用方式也是十分的小，因为这里连基本的数据归一化都不用做了，直接暴力）01就好了。

首先，我们直接将文件改成.txt的格式，完全读取后直接二值化，最后转成同一个大小的变量，接着就可以直接识别了。这里我们也讲下，.csv文件的读取。

<http://blog.csdn.net/u012162613/article/details/41768407>

代码如下：这里会简单的使用KNN来实现暴力搜索，时间复杂度和空间复杂度都高的一匹，具体怎么使用自行百度，这里没用什么函数，**就是自己拿一张图设为固定点，接着直接把其他的東西帶入暴力匹配就結了**，主要这个K值可能设的比较蛋疼（C++代码就不贴了，写着卵疼）。

## 1- 图像Otsu 最小二乘法（二值化）

我们先来实现一个最简单的OpenCV图片直接二值化为1024维向量的过程：

MATLAB代码：

```

img = imread('6.jpg');
img = imresize(img, [32, 32]);
subplot(121);
imshow(img);
img2 = rgb2gray(img);
%graythresh: 自动确定最佳阈值, OpenCV也有一样的功能
thresh = graythresh(img2);
%对图像二值化处理
img2 = im2bw(img2, thresh);
subplot(122);
imshow(img2);
vec = imresize(img2, [1, 1024]);

```

其中二值化算法一般借助于**Otsu最小二乘法**，一般这种情况下的这种拟合是最好的，平均拟合一般是最差的选择，选择了这个以后，我们相当于暴力提取特征，虽然如果是无差别训练，而非“对症下药”，这种做法可能会莫名的好用，尤其是我们做神经网络的时候，我估计slam也是直接暴力提取的，接着我们在这里使用暴力KNN计算（这节主要讲的是图像预处理，提取特征的方式，所以学习和分类算法会比较简单的带过，晚上还要学习和打比赛呢.....）。

```

def classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
    diffMat = tile(inX, (dataSetSize,1)) - dataSet
    sqDiffMat = diffMat**2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances**0.5
    sortedDistIndicies = distances.argsort()
    classCount={}
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
    sortedClassCount = sorted(classCount.iteritems(), key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]

```

这些代码在后续KNN那里会详细讲解，但是这里，就直接跳过了。

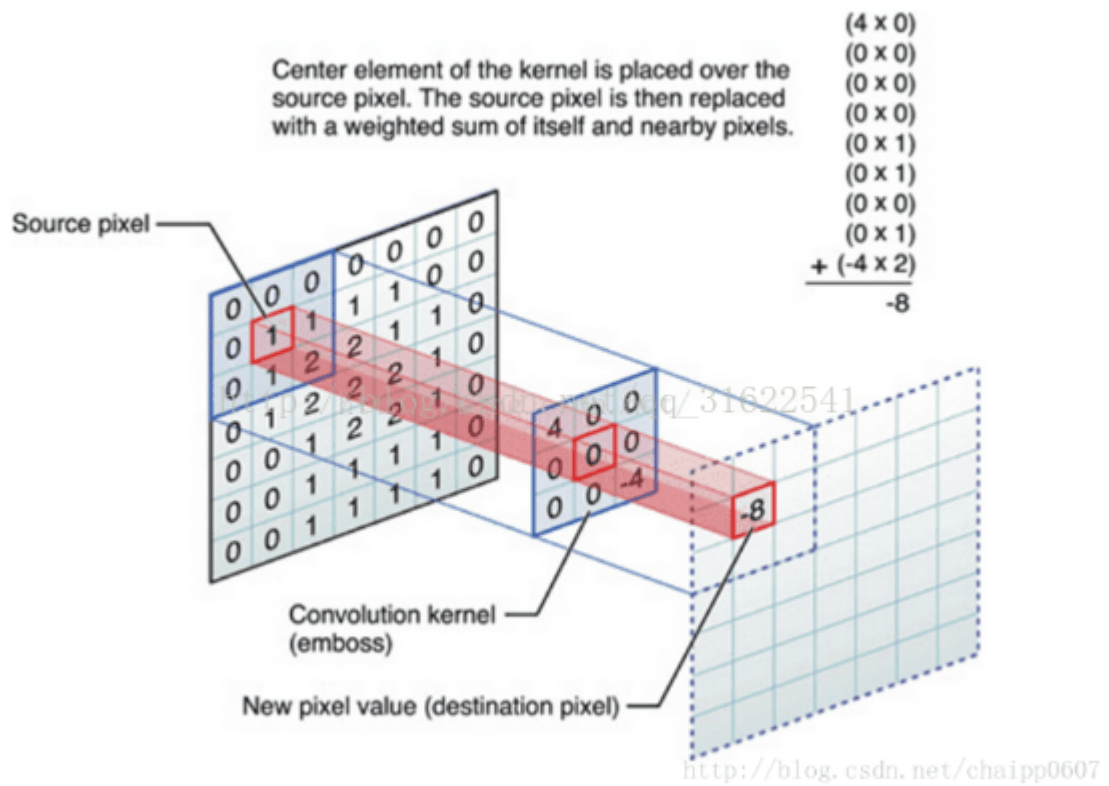
接着我们来讲讲第二种提取方式，那就是我们在CNN和RNN里面预处理的方式，这种方式相对于原本的方法比较高级了点，不过这里准确说是强化特征，而特征使用的还是整体特征，没有专门钻出一个来作为真实特征。

首先拿CNN（卷积神经网络）来看：这里对图像的操作是卷积，我们首先简单了解下那个卷积是什么东西，从信号系统来看，卷积就是信号1经过信号2的时候发生的变化，是信号的累加过程。

$$y(n) = \sum_{i=-\infty}^{\infty} x(i)h(n-i) = x(n) * h(n)$$

在信号处理过程中，我们认为影响信号无穷大，但图像过程中我们把正无穷看成高斯卷积核的大小就好了，实际上，我们不可能无限放大信号影响域，这时候，我们只要简单的使用小的一个域就好了，为了保证中心效应，我们一般使用的是奇数，最终累加相乘的内积就是了，采用不同的核最终得到的结果的代表意义也是不一样的。

卷积核的计算：



参考网站：

<http://blog.csdn.net/chaipp0607/article/details/72236892?locationNum=9&fps=1>

直接使用OpenCv程序 ( C++ )：

```

#include <iostream>
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
using namespace std;
using namespace cv;

int main()
{
    Mat srcImage = imread("1.jpg");
    namedWindow("srcImage", WINDOW_AUTOSIZE);
    imshow("原图", srcImage);
    Mat kernel = (Mat_<double>(3,3) <<
        -1, 0 ,1,
        -2, 0, 2,
        -1, 0, 1);
    Mat dstImage;
    filter2D(srcImage, dstImage, srcImage.depth(), kernel);
    namedWindow("dstImage", WINDOW_AUTOSIZE);
    imshow("卷积图", dstImage);
    waitKey(0);
    return 0;
}

```

MATLAB 程序：

```

function [ fiter ] = fiter( Kernal,srcImg )
    [w,h] = size(srcImg);
    [wi,hi] = size(Kernal);
    if wi ~= hi
        warning('Kernal Message mistake');
        return;
    end
    if mod(wi,2) == 1
        warning('Kernal Message mistake');
        return;
    end
    len = (wi - 1) / 2;
    for i = 1:1:w
        for j = 1:1:h
            if i <= len || i > w - len || j <= len || j > w - len
                fiter(i,j) = srcImg(i,j);
            else
                fiter(i,j) = calc( srcImg(i-len:i+len,j-len:j+len) , Kernal , wi);
            end
        end
    end
end

function [res] = calc(base,Kernal,n)
    res = 0;
    for i = 1:1:n
        for j = 1:1:n
            res = res + base(i,j) * Kernal(i,j);
        end
    end
end

```

复杂度高的一匹，但确实是这么算的，如果加入了浮点运算直接算崩掉，一点办法都没有。

接下来，我们来讲下其他特征提取方式，其中主要有颜色特征，形状特征，区域特征。

## 2- 颜色特征提取：

### 2.1 灰度直方图（按灰度分布生成向量）

代码如下：

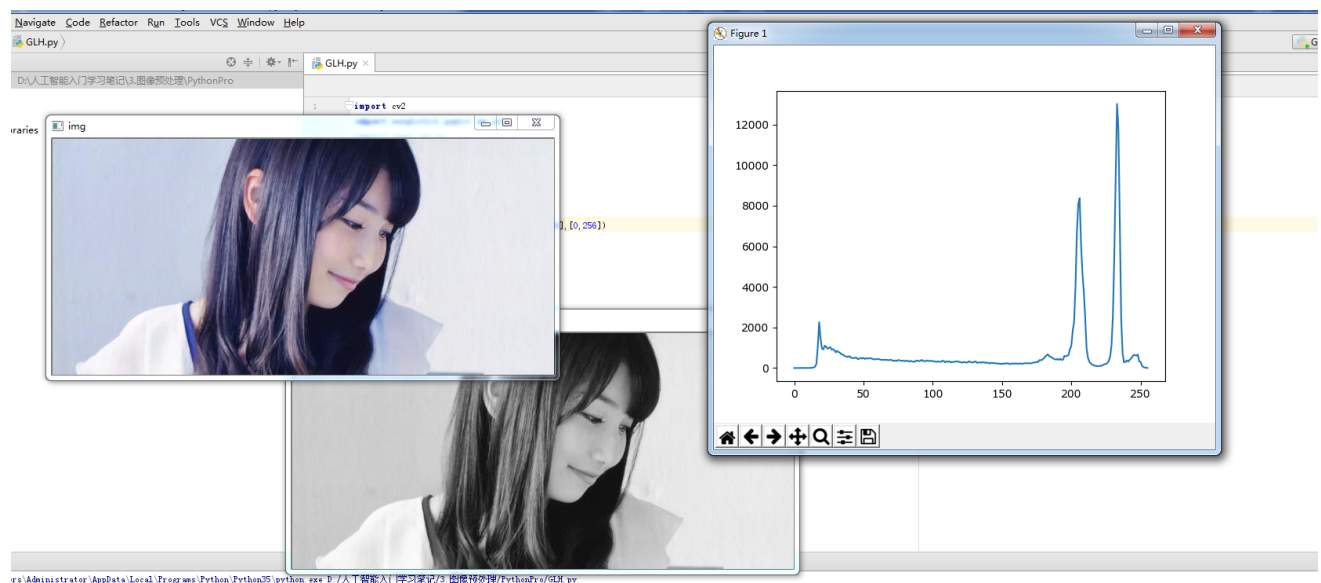
```

import cv2
import matplotlib.pyplot as plt
import numpy as np

img = cv2.imread('1.jpg')
img2 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow('img', img)
cv2.imshow('gray', img2)
hist_cv = cv2.calcHist([img2], [0], None, [256], [0, 256])
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(hist_cv)
plt.show()
cv2.waitKey(0)

```

效果如下：



我们直接把灰度分布打印出来就是特征向量，结果如下：

```
[ 3.03000000e+02]
[ 3.60000000e+02]
[ 3.45000000e+02]
[ 2.82000000e+02]
[ 3.28000000e+02]
[ 3.07000000e+02]
[ 3.30000000e+02]
[ 2.78000000e+02]
[ 2.99000000e+02]
[ 3.00000000e+02]
[ 3.09000000e+02]
[ 3.26000000e+02]
[ 3.23000000e+02]
[ 3.03000000e+02]
[ 2.90000000e+02]
```

结果就是上面显示的一个列向量，简单训练的时候直接带进去就好了，基本都不用考虑好坏的，因为神经网络不就是那么无脑的东西吗，但是，单靠他是达不到我们梦想的高度的，一定要知道这一点，不然就陷入了大误区。

## 2.2 HOG特征（进行物体检测描述算子，图像梯度加梯度方向）

特征获取过程：

1. 灰度化（将图像变成一个x,y,z（灰度）的三维图像）。
2. 采用Gamma校正法对输入图像进行颜色空间的标准化（归一化），调节图像的对比度，降低图像局部阴影和光照变化所造成的影响，同时可以抑制噪声（主要因为局部表层的曝光贡献比重比较大，所以这种压缩处理能够有效降低图像局部阴影和光照变化，因为颜色信息作用不大，所以通常先转为灰度图），但归一化不是什么时候都能用的，在曝光不严重的时候是会失真的（详见计算机图形学）。

$$I(x, y) = I(x, y)^{\text{gamma}}$$

公式如下：

（减缓曝光效果，转为人眼适应的值）

3. 计算每个像素的梯度（包括大小和方向），捕获轮廓信息的同时，弱化光照的干扰。

$$G_x(x, y) = H(x+1, y) - H(x-1, y)$$

$$G_y(x, y) = H(x, y+1) - H(x, y-1)$$

式中  $G_x(x, y)$ ,  $G_y(x, y)$ ,  $H(x, y)$  分别表示输入图像中像素点(x,y)处的水平方向梯度、垂直方向梯度和像素值。像素点(x,y)处的梯度幅值和梯度方向分别为：

$$G(x, y) = \sqrt{G_x(x, y)^2 + G_y(x, y)^2}$$

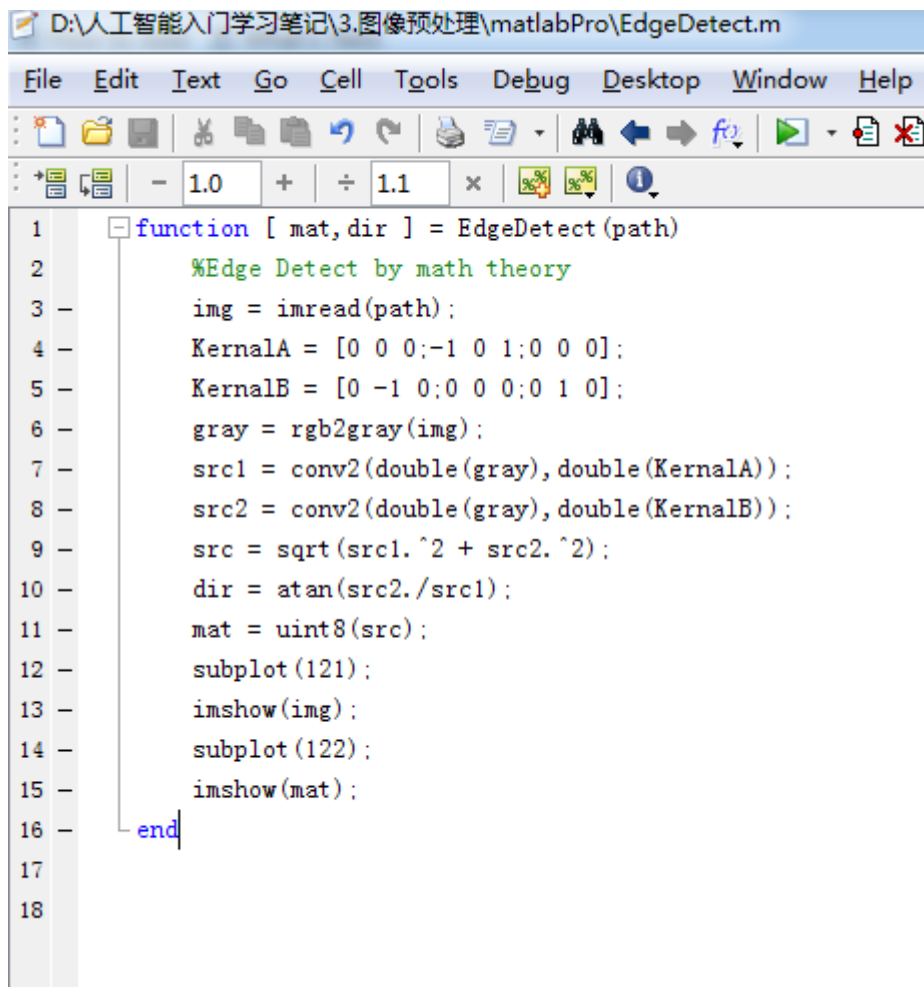
$$\alpha(x, y) = \tan^{-1}\left(\frac{G_y(x, y)}{G_x(x, y)}\right)$$

4. 将图像分为小cell（例如6\*6像素/cell）
5. 统计每个cell的直方图（不同梯度的个数），**一般统计梯度方向**，分为8个区间统计就可以了，每个直方图统计出来就是每个cell的descriptor。
6. 将每几个cell组成一个block，一个block中所有的cell特征的descriptor串联起来便得到该block的HOG特征的descriptor。
7. 将图像中所有的block的HOG特征的descriptor**串联起来**后就可以得到该image的HOG特征的descriptor了，这就是我们最终使用的特征向量了，**用这个向量再去计算SVM就可以实现一些简单的行人检测或者再识别的东西了。**
8. 特征数计算方法（后面你就会知道为什么会算那么慢了，结合**滑动窗口统计**之后，我们会发现，每次迭代的量都是上万的，这样不慢才怪）。

把样本图像分割为若干个像素的单元（cell），把梯度方向平均划分为9个区间（bin），在每个单元里面对所有像素的**梯度方向在各个方向区间进行直方图统计，得到一个9维的特征向量，每相邻的4个单元构成一个块（block），把一个块内的特征向量联起来得到36维的特征向量**，用块对样本图像进行扫描，**扫描步长为一个单元。最后将所有块的特征串联起来，就得到了人体的特征。**例如，对于64 x 128的图像而言，每16 x 16的像素组成一个cell，每2 x 2个cell组成一个块，因为每个cell有9个特征，所以每个块内有4 x 9=36个特征，以8个像素为步长，那么，水平方向将有7个扫描窗口，垂直方向将有15个扫描窗口。也就是说，64 x 128的图片，总共有36\*15=3780个特征。

代码如下：

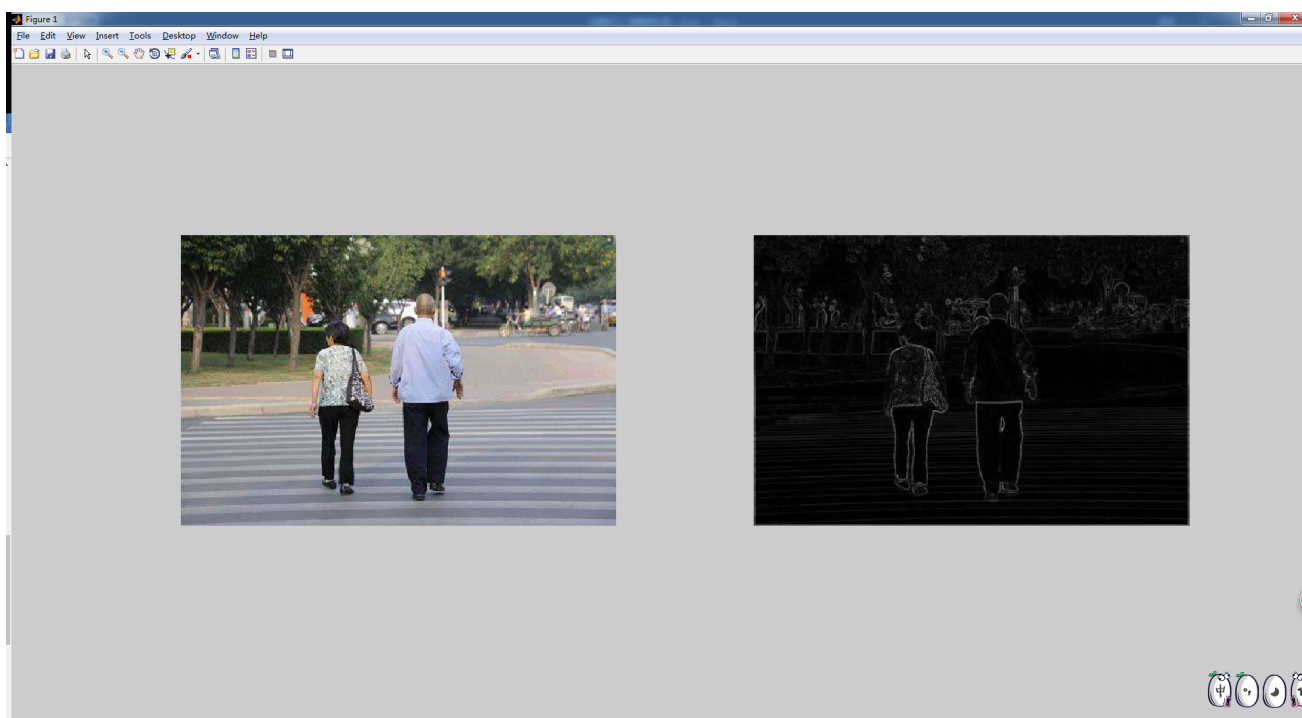
#### 1.边缘特征提取以及图像归一化处理



```
1 function [ mat, dir ] = EdgeDetect(path)
2     %Edge Detect by math theory
3     img = imread(path);
4     KernalA = [0 0 0;-1 0 1;0 0 0];
5     KernalB = [0 -1 0;0 0 0;0 1 0];
6     gray = rgb2gray(img);
7     src1 = conv2(double(gray),double(KernalA));
8     src2 = conv2(double(gray),double(KernalB));
9     src = sqrt(src1.^2 + src2.^2);
10    dir = atan(src2./src1);
11    mat = uint8(src);
12    subplot(121);
13    imshow(img);
14    subplot(122);
15    imshow(mat);
16    end
17
18
```



效果如下：

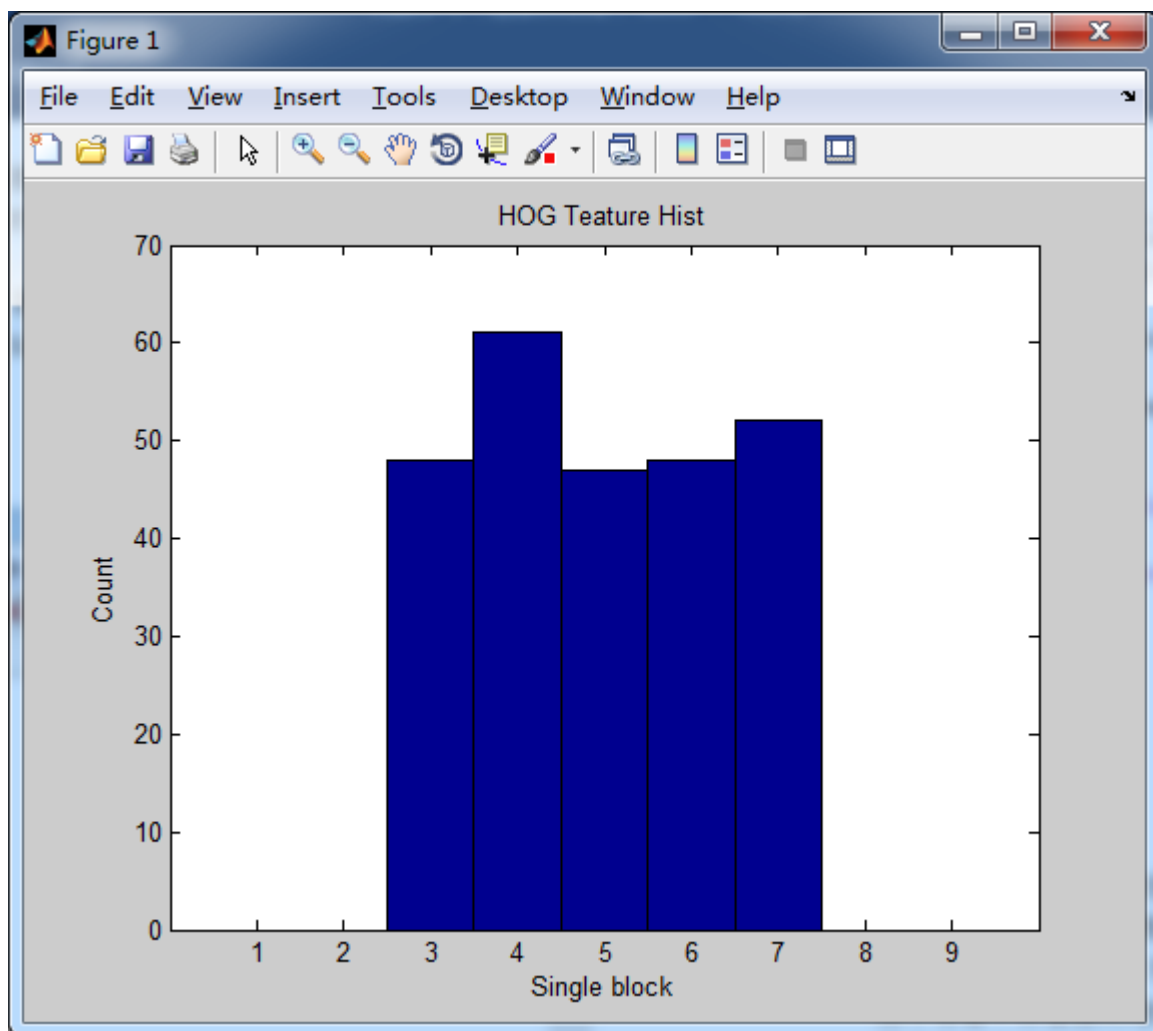


接下来，我们来计算一段区域提取出来的直方图。

程序源码：

```
function [ Hist ] = cellHist( cell )
    [x,y] = size(cell);
    Hist = 1:9;
    cell = reshape(cell, [1,x*y]);
    for i = 1:1:9
        Hist(i) = length(find(cell > 2*pi/9*(i-1)-pi & cell <= 2*pi/9*i-pi));
    end
    bar(Hist,1);
    xlabel('Single block');
    ylabel('Count');
    title('HOG Feature Hist');
end
```

实现效果：



接着就十分的简单了，我们只需要简单的把之前的东西计算到blocks里面后再重新计算到一个向量里面了，接下来我就不接着写下去了，因为接下来的东西就涉及到学习算法了，脱离了这里的主题了（好吧，其实是我懒了，接着直接教大家怎么调包吧）。

Python直接调包：

```

import cv2
import matplotlib.pyplot as plt
image = cv2.imread("1.jpg", 0)
winSize = (64, 64)
blockSize = (16, 16)
blockStride = (8, 8)
cellSize = (8, 8)
nbins = 9
derivAperture = 1
winSigma = 4.
histogramNormType = 0
L2HysThreshold = 2.0000000000000001e-01
gammaCorrection = 0
nlevels = 64
hog = cv2.HOGDescriptor(winSize, blockSize, blockStride, cellSize, nbins, derivAperture, winSigma,
                        histogramNormType, L2HysThreshold, gammaCorrection, nlevels)
#compute(img[, winStride[, padding[, locations]]]) -> descriptors
winStride = (8, 8)
padding = (8, 8)
locations = ((10, 20),)
hog.compute(image, winStride, padding, locations)
plt.plot()

```

C++直接调包：

```

HOGDescriptor *hog=new HOGDescriptor(cvSize(3,3),cvSize(3,3),cvSize(5,10),cvSize(3,3),9); //具
本意见参考文章1,2
vector<float>descriptors;//结果数组
hog->compute(trainImg, descriptors,Size(1,1), Size(0,0)); //调用计算函数开始计算

```

数据格式：

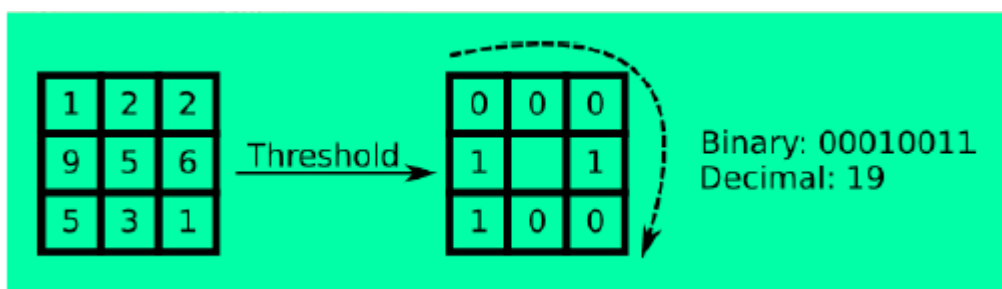
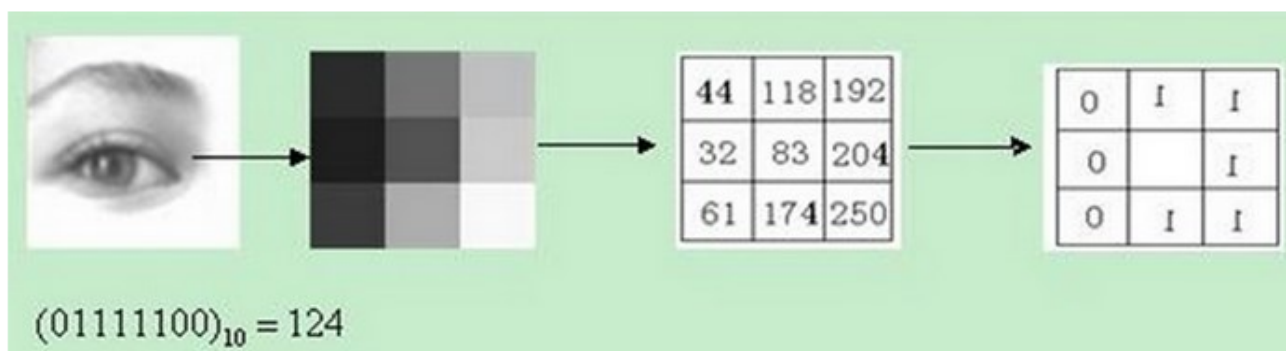
```

[[ 0.20307139]
 [ 0.09652878]
 [ 0.11739929]
 ...,
 [ 0.06741033]
 [ 0.02252736]
 [ 0.31531346]]

```

## 2.3 LBP特征（描述局部纹理特征）

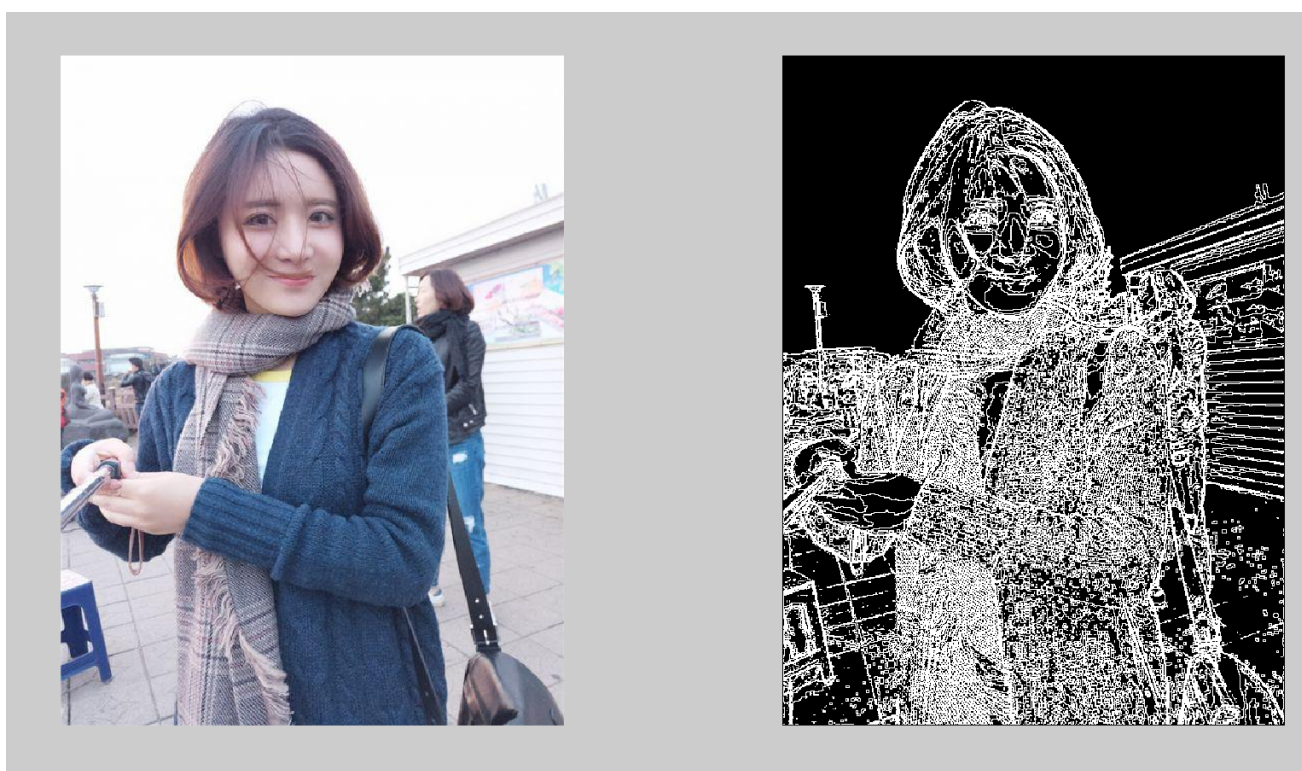
原始LBP特征的提取：先把图像转为灰度图，把图像分成3\*3的块，通过中心相对的高低来计算对应的纹理凹凸值，这样就能把一块块的纹理给计算出来了，我们先写个简单的计算程序来显示下最简单的特征样式，表现在图像上。



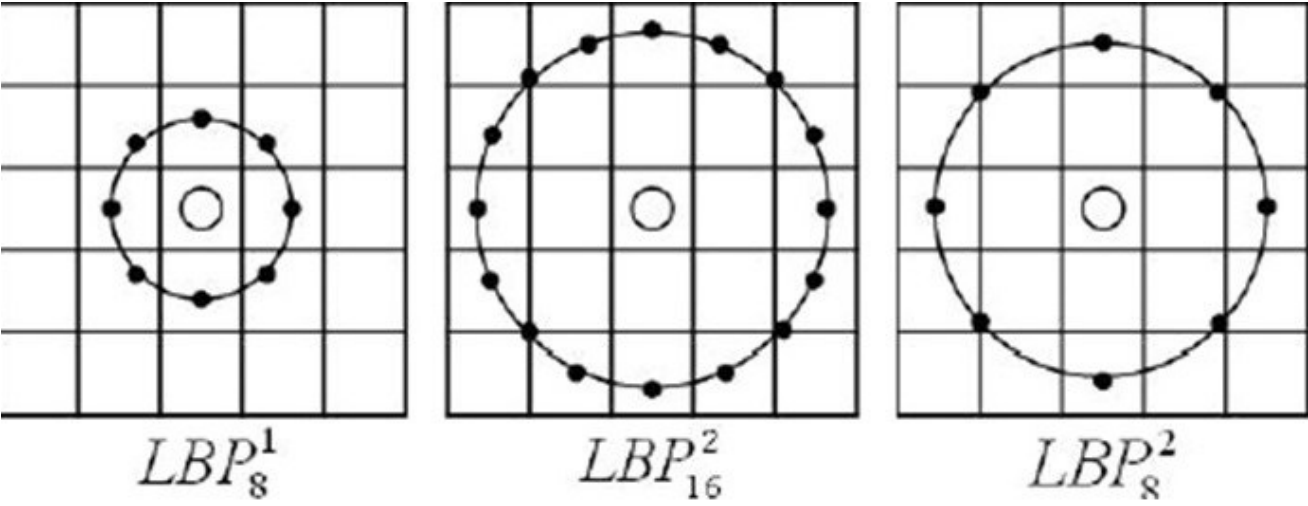
简单纹理特征提取（MATLAB代码）：

```
function [ result ] = midCompare(mat)
    res = zeros(3,3);
    for i = 1:1:3
        for j = 1:1:3
            if mat(i,j) > mat(2,2)
                res(i,j) = 1;
            end
        end
    end
    result = res(1,1) * 128 + res(1,2) * 64 + res(1,3) * 32 + res(2,3) * 16 + res(3,3) * 8 + res(3,2) * 4 + res(3,1) * 2 + res(2,1);
end
```

提取出来后的结果：

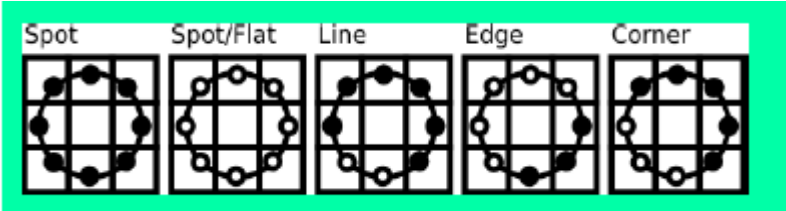


用了我们可爱的发妈妈的美照（我不会被打吧）。。。。不过上面的程序的复杂度高一匹，所以怎么说呢，这个程序还不是那么的尽善尽美的，所以大家可以努力一起搞下，把复杂度降低一点，这样就可以了。



为了提高程序的鲁棒性，我们基本会采用这种按圆形获取整体形状，然后把几个不同圆半径的图像进行编码，**最终实现比较稳定的纹理特征的提取**，而不会因为一些特殊原因导致奇怪的问题，对了，还有一点就是要去除过度曝光。

公式和大致图形过程如下：



对于给定中心点 $(x_c, y_c)$ ，其邻域像素位置为 $(x_p, y_p)$ ， $p \in P$ ，其采样点 $(x_p, y_p)$ 用如下公式计算：

$$x_p =$$
$$y_p =$$

$$x_c + R \cos(\frac{2\pi p}{P})$$
$$y_c - R \sin(\frac{2\pi p}{P})$$

R是采样半径，p是第p个采样点，P是采样数目。由于计算的值可能不是整数，即计算出来的点不在图像上，我们使用计算出来的点的插值点。目的的插值方法有很多，Opencv使用的是双线性插值，双线性插值的公式如下：

$$f(x, y) \approx$$

$$\begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix}$$

## 2.4 Haar-like特征（边缘检测特征）

矩形特征的值，是指图像上两个或多个形状大小相同的矩形内部所有像素灰度值之和差值，即使用**白色矩形区域所有像素灰度值之和减去黑色矩形区域所有像素灰度值之和**（这个可以说是相当的简单，看下东西就好了，这种方法一般都是和AdaBoost方法一起使用的，而基本的特征提取简单的一匹）。

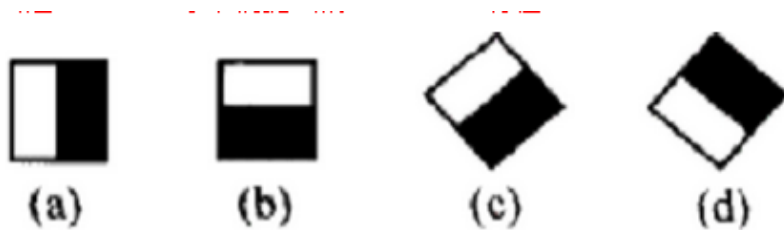


图 2 边界特征

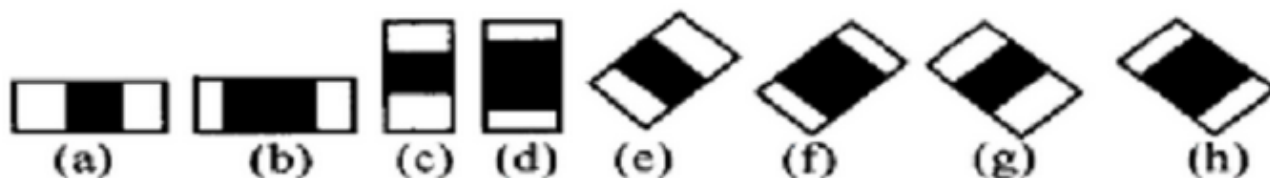


图 3 线特征

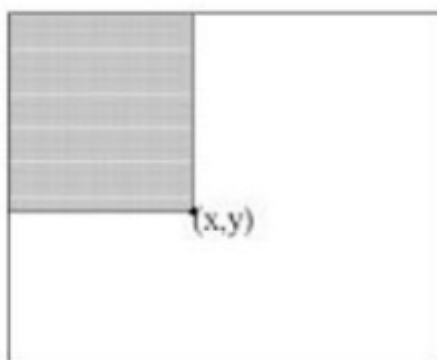


图 4 中心特征

公式：

$$\text{Sat}(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

如图2 所示，点  $(x, y)$  处的积分图像值等于图中阴影部分所有像素灰度值的和。



## 2.5 角点特征和轮廓特征

（机器学习无关，但是在某些匹配算法和计算机图形学中的填充算法中有着十分重要的地位，比如Harris，检测出来角点和边缘之后在计算机图形学中的重要程度简直就高的有点可怕，不过这里就不重点讲了，详见Opencv—python turtotials）。

然后介绍完所有的特征和写好所有的特征提取代码之后，我想说的一件事是不要遇到什么特征就去抓，不然你的每一波训练都是麻瓜，你需要先去把特征都找出来之后，然后去对比每个特征相对的权重，Andrew Ng老师的课程里面虽然说了那些无效特征到最后都会自己在神经网络之类的训练模型里面变成皮皮虾，但在svm里面，不是直接带值，这样一来，特征的选择就更加重要了，按前人的分类选择特征，然后对应选择合适的方法，SVM还是

DeepLearning，还是暴力KNN或者稍微带点脑子但是慢的一匹的K-means，所以选好特征再在原本基础上使用最小二乘法实现最小值获取，这样就可以开始搞事情了。