**KDE 大作业**

**一、编程平台及环境配置**

编程使用的软件：pycharm（Anaconda下的Python 3.7 64-bit环境）

使用的库函数有：cv2、math、numpy、Pillow、mpl\_toolkits、matplotlib

1. **编程思路（使用EP核函数进行核密度估计）**

**第一步**、我们获得20个样本集，然后按照训练样本90%，测试样本10%的原则，将图片分为18张训练数据集与2张测试数据集。（其中每一个训练数据集的图片含1920\*1030个像素点，为RGB三通道模式。）

**第二步**、对测试集图片上的每个像素点使用EP核函数公式，公式如下：



选取带宽h=100，计算出其概率估计。

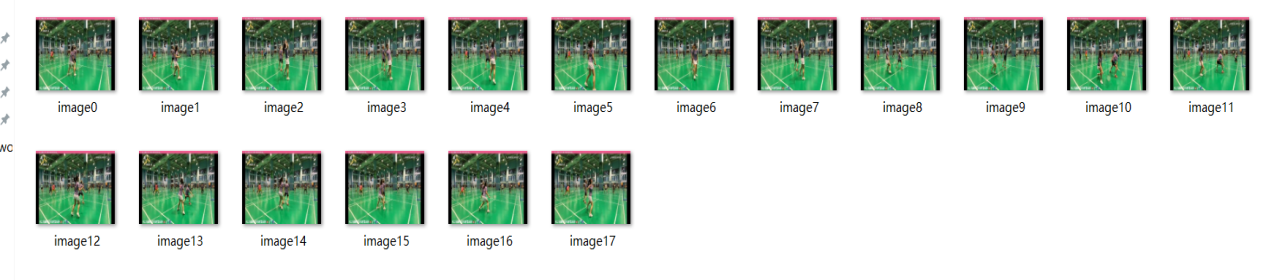
**第三步**、根据选择的参数，我们先观察EP函数的常数部分的量级3\*10^(-8)，并且如果测试图片与训练图片若完全相似，则将部分作为常数1看待，这样我们可以根据公式得到了一个大概的EP函数的阈值为：1x10^(-7)这一个量级的数字。

**第四步**、由阈值得到的图像显示结果，调整阈值的范围，直到达到最优的二值化图像，并且得到对应的EP三维彩色图像估计图。

**第五步**、问题的发现以及解决方法。

**三、数据源的选取**

训练数据和测试数据我选取了网上b站上的固定摄像头下拍摄的羽毛球训练情况，共18组训练数据集，2个测试数据集，



**图3-1 样本中的图片**



**图3-2 两张测试集图片**

**四、实现的Python代码和注释**

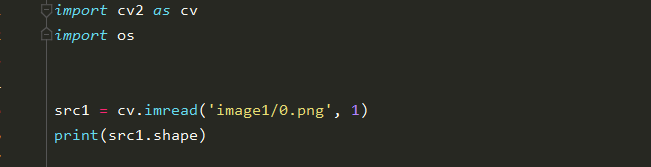
1. **打开pycharm，并且做好相应的环境配置**
2. **导入完成程序所需要的库函数。**



**图4-1 库函数代码**

**3、按照自动化思想编写一个循环遍历打开训练集的函数，通过读取源图片来确定创建数组的大小格式并且通过cv库读取图片得到图片的样式大小为(1030, 1920, 3)。**

代码如下：





**图4-2 图片样式**

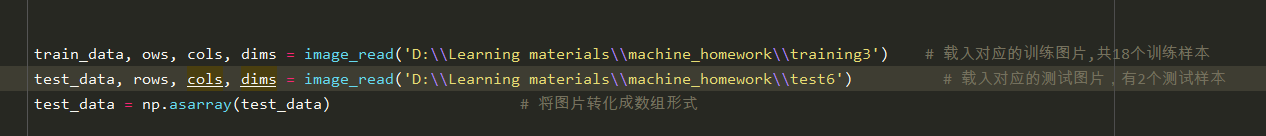
因此我们知道应该创建的数组是1920\*1030的三通道RGB图像。



**图4-3 图片加载函数**

**4、载入数据集与测试集。**

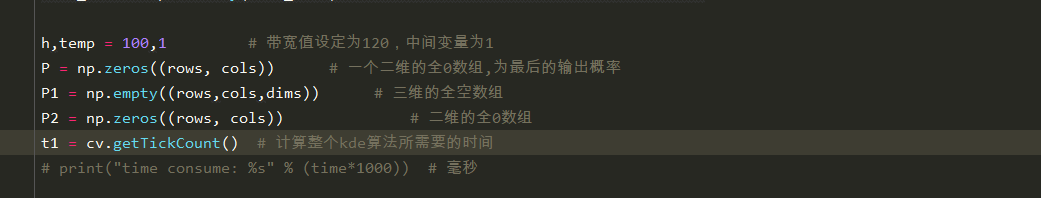
代码如下：



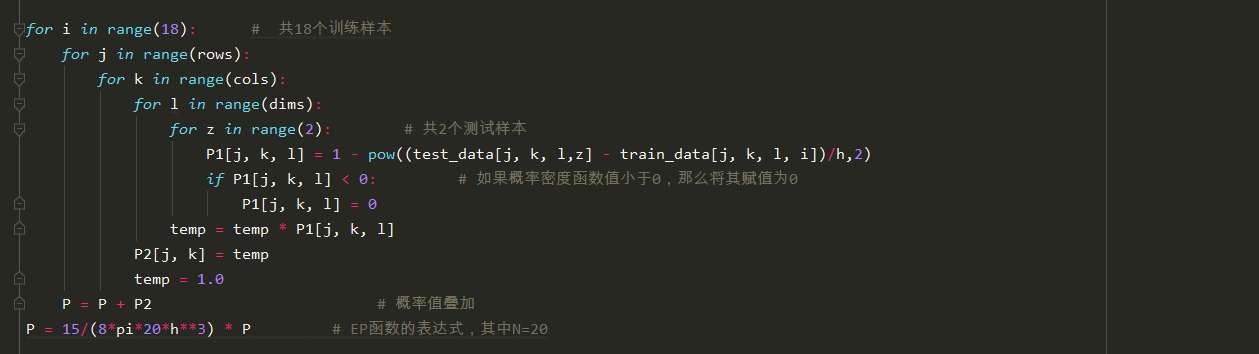
**图4-4 载入图片代码**

1. **编写核密度估计函数Epanechnikov。**

代码如下：



**图4-5 参数准备代码**

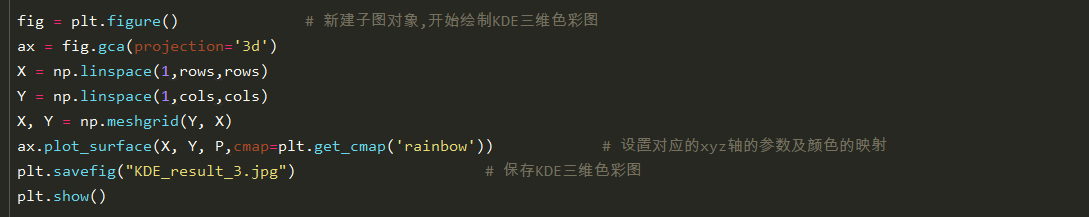


**图4-6 Epanechnikov函数实现代码**

**6、绘制出对应的3维概率密度函数图片及二值图像**

(1) 绘制3维图片部分的程序及注释。

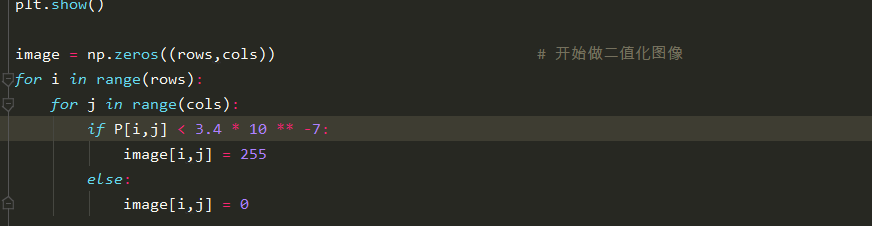
代码如下：



**图4-7 Ep三维图代码**

1. 根据EP函数的输出大小，如果小于阈值，则像素为白色，说明图像相似度低；反之则像素为黑色，说明图像相似度高。转化为二值图像并绘制的程序及注释。

代码如下：



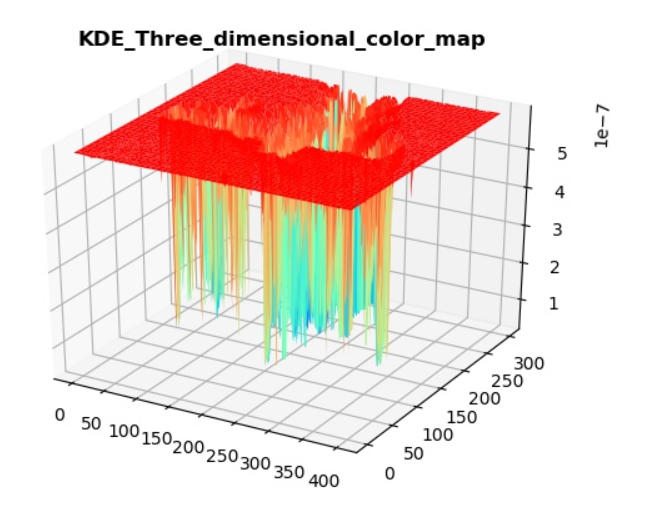
**图4-8根据阈值做二值化准备**



**图4-9 二值化图像显示代码**

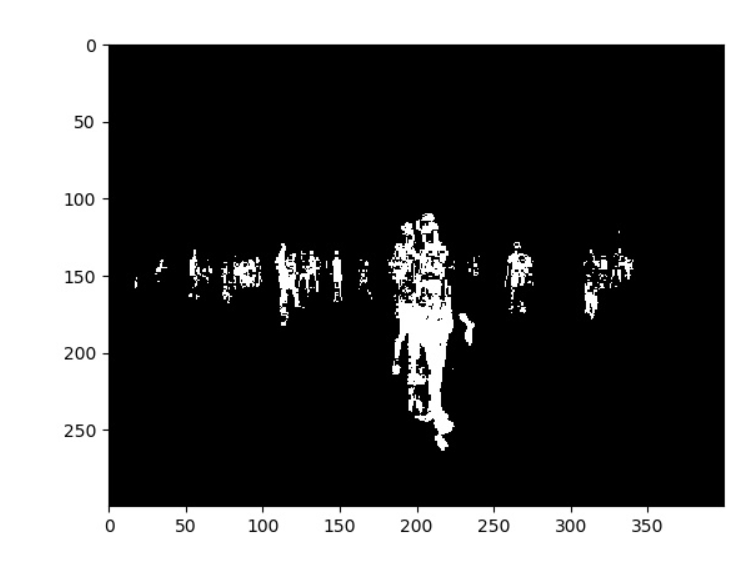
**五、实验结果**

1、KDE三维图：



**图5-1 KDE三维图**

2、二值化得到的图像：

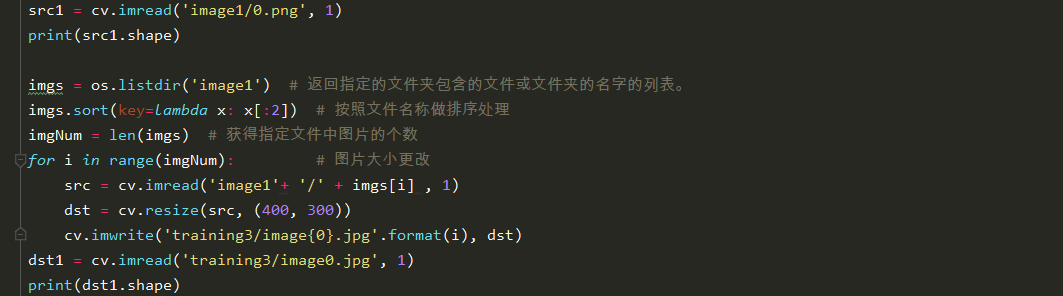


**图 5-2 binary后的二值化图**

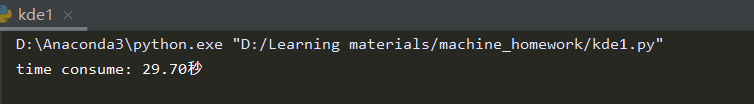
**六、问题以及解决措施**

1.从网上获得的图片因为初始形状是(1030, 1920, 3)，可能图片比较大，导致运行的程序时间过长，这里我们通过opencv库的resize重新调整图片的大小，改成(400,300,3)这样的样式，使得原来需要运行5分钟以上的程序现在30秒左右就可以得到结果。

代码如下：



**图6-1图像样式大小改变的代码**

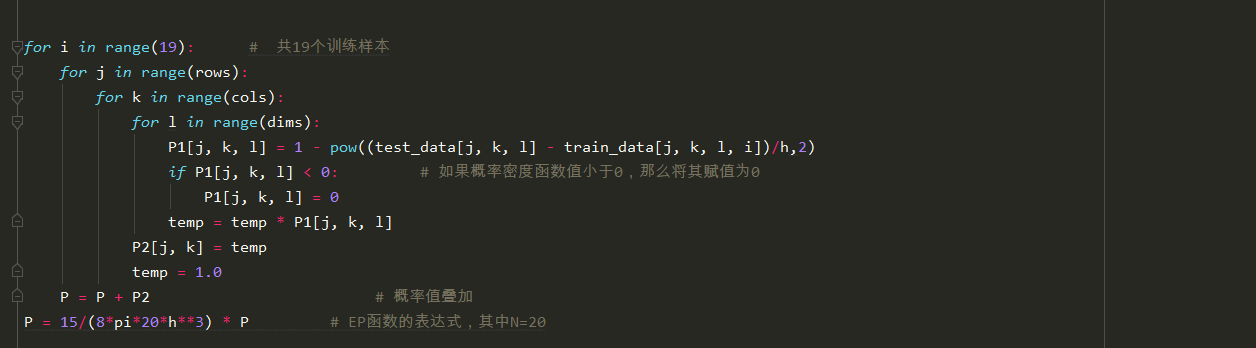


**图6-2 运行程序所需时间**

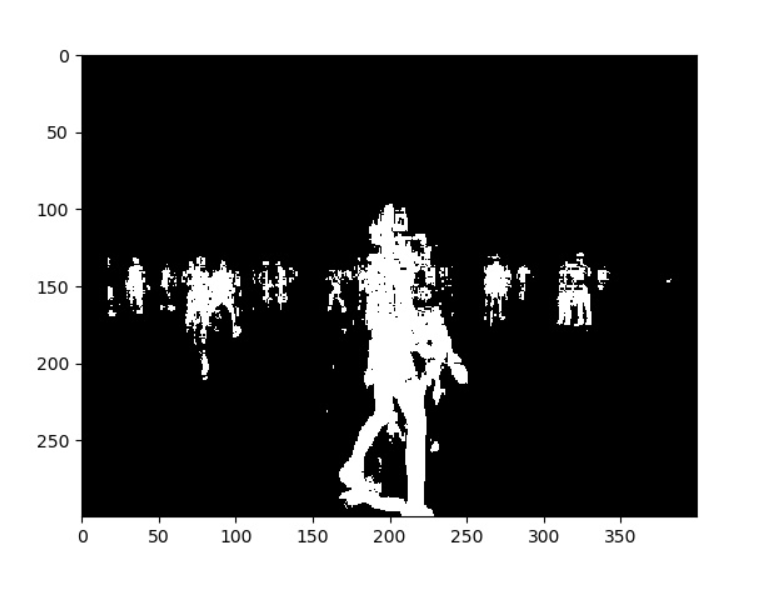
2、带宽h的选择问题：若h选取值太小，KDE曲线会更加陡峭，二值化后会出现很多的白点，为防止二值化图像中白点过多，所以选取了一个较大值100.

3. 因为总的样本数不是很多，并且测试集有两个，可能得到的图片中人的精度不是很高，我们将训练集换为19个，测试集换为1个，进而对EP核函数主体进行更改。

代码如下：



**图6-3 更改训练集与测试集后的代码**

获得的二值化图像如下：

**图6-4 binary后的二值化图像**

从二值化图像可以见到与原来的2个测试集的图片相比少了很多人的重影部分，并且轮廓更加清晰。

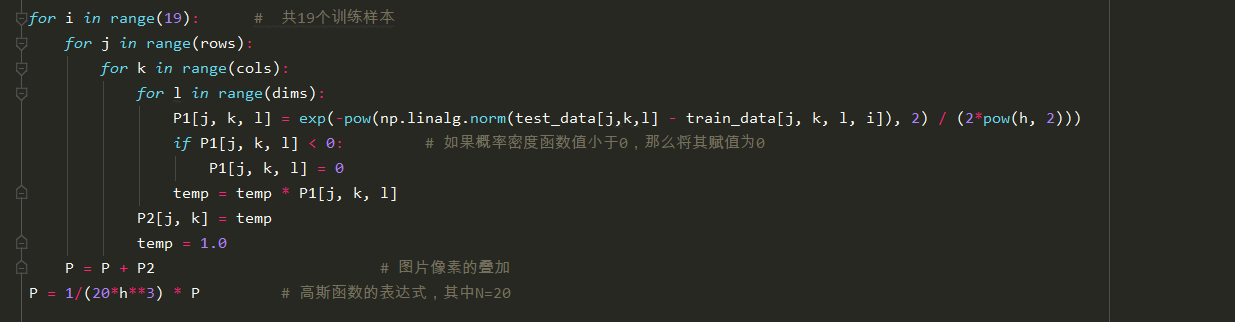
**七、大作业内容4：使用高斯核函数进行分析**

1、给出的高斯核函数公式如下：



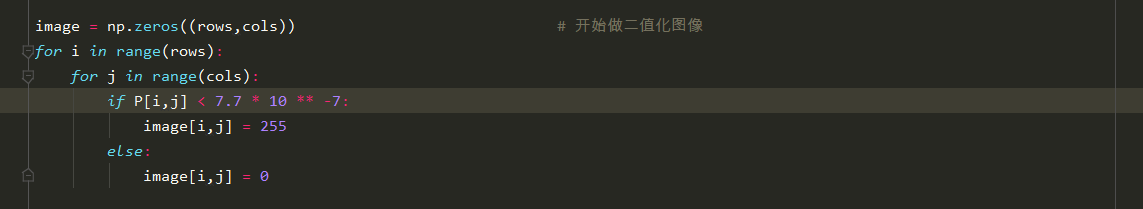
2、改动的程序部分如下图所示：

（1）函数表达式部分，包括循环体中的表达式和循环体外得出结果的表达式：



**图 7-1 Gaussian核函数实现的代码**

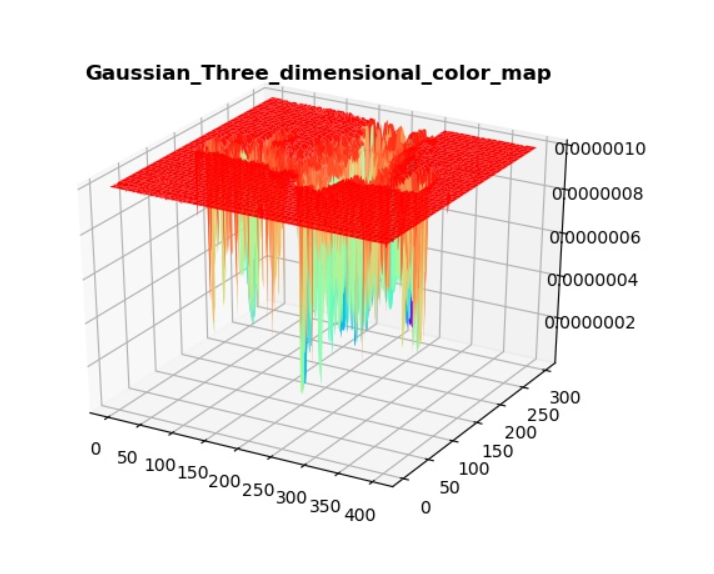
（2）阈值部分：经实验，在使用高斯核密度函数后， 阈值要适当地调高，经过调试之后将阈值改为7.7 x10^(-7)，代码如下：



**图7-2 调整后的阈值代码**

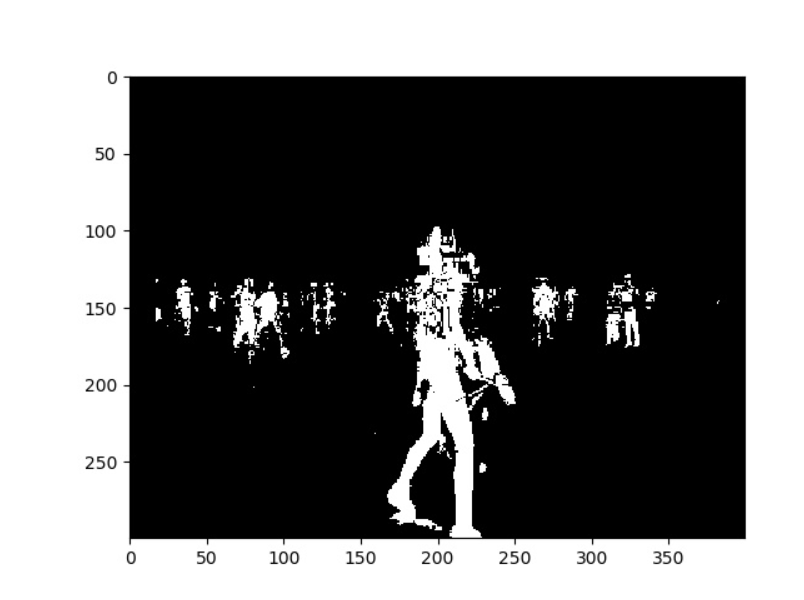
（3）其他部分程序不改动，可参照上述EP核函数实现的代码程序截图。

3、Gaussian三维图如下图所示：



**图7-3 KDE三维彩色图**

4、二值化后图像如下图所示：



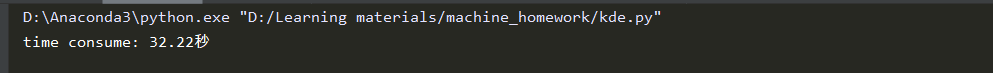
**图7-4 binary后的二值化图像**

5、实验分析

使用Gaussian核函数进行核密度估计的结果相较EP核函数估计结果来看：总体来说图片的质量差不多。

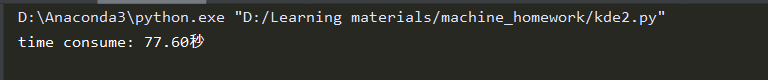
首先，在实验过程中，由于要在循环体中进行大量exp()函数进行计算，导致程序运行过程明显减慢，需要更长的时间才能得到结果。为了有一个更直观的反应，在相同的情况下我们引入cv库中的geTickCount()函数计时。

EP核函数所需时间:



**图7-5 EP核函数运行所需时间图**

Gaussian核函数完成所需时间：



**图7-6 Gaussian核函数运行所需时间图**

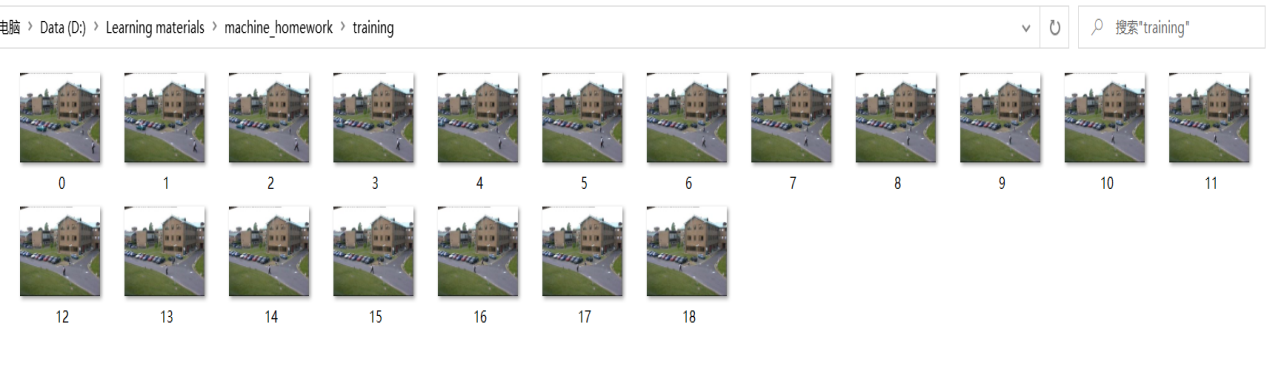
其次，通过细致观察两个图像，我们发现用Gaussian核函数得到的二值化图像在人与人之间交界处噪声点更少，所以Gaussian得到的图片质量更好。

最后，我们可以得到结论：在其他条件相同下，Gaussian核函数得到的图片质量更好，但所需的时间资源更多；同理，EP核函数图片质量更低，所需的时间更少。

**八、大作业内容5： 从网上搜集的一组数据集图片**

1. **数据集来源**

我们从网上CSDN核密度估计实验现有的资料中下载已有20张图片，由于图片的由固定镜头拍摄以及人物只有2个人，干扰较小，预计能得到不错的结果。



**图8-1 由CSDN下载的固定角度的样本**



**图8-2 一张测试集图像**

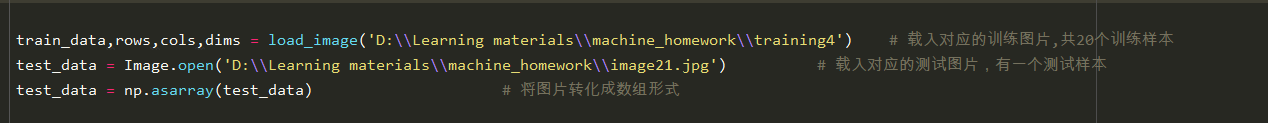
1. **程序更改**

（1）读取训练数据的函数中，改变图片大小的数据，代码如下：



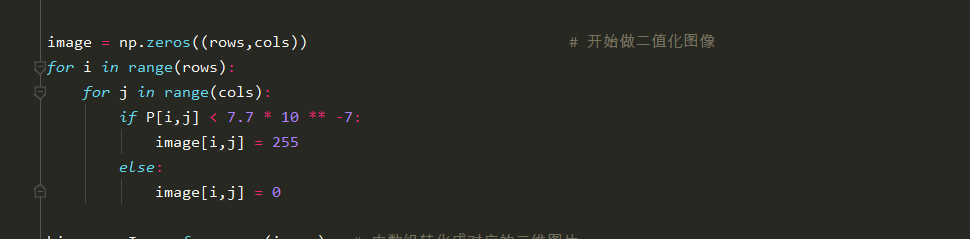
**图8-3 部分改变代码**

（2）改变读取训练数据和测试图片的文件地址，代码如下：



**图8-4 载入相应图片代码**

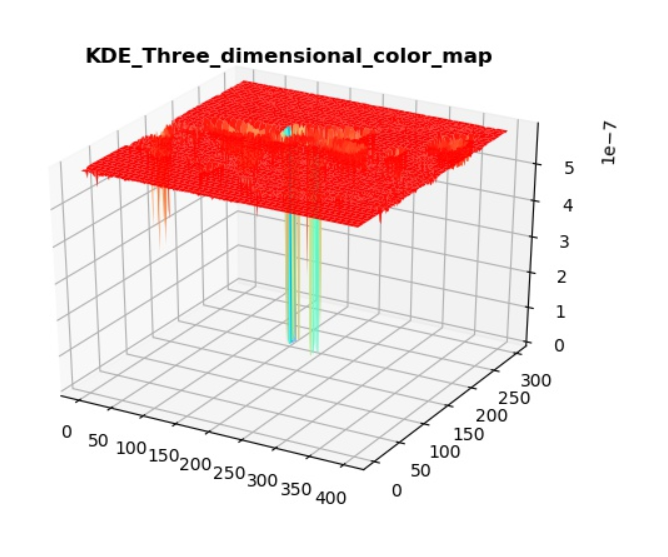
1. 通过调整合适的阈值来得到合适的二值化图片，代码如下：



**图8-5 调整阈值后的代码**

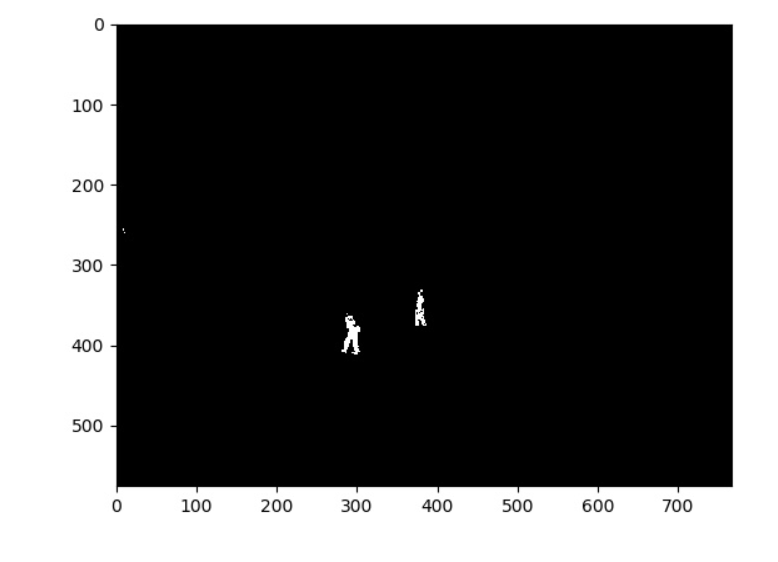
**3、实验结果**

（1）KDE函数三维图像：



**图8-6 KDE三维色彩图**

（2）二值化后的图像：



**图8-7 二值化图像**

1. 二值化图像说明：我们可以看到二值化图像检测出了行走的2个人和，较好得体现出了运动的事物。

**九、大作业心得**

通过本次实验，我更深入地理解了EP核函数与Gaussian核函数，通过实验把神奇的应用在直观而有趣的图像上，会使我终生难忘。同时，对python的库函数也有了更多的掌握；最后，我相信在今后的学习中会对机器学习这门课有更多的理解。