**K-means算法实现实验报告**

201250070 郁博文

1. 数据预处理

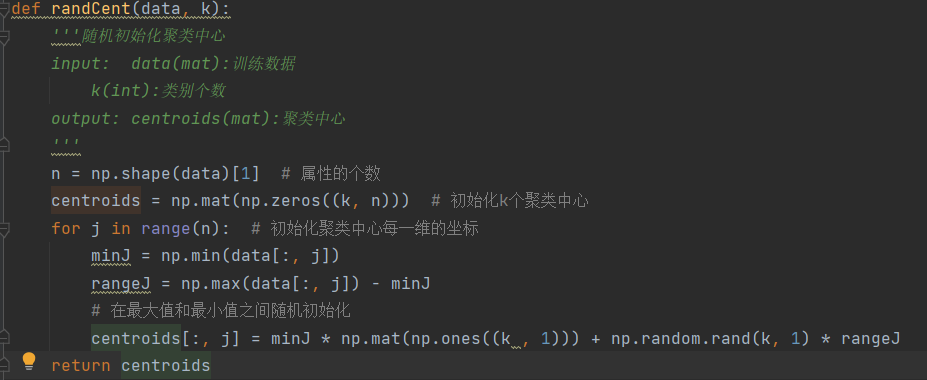
首先在网上随机找一张图片。我选用的是1600\*900的尺寸。为了方便后续计算，将其裁剪为256\*256大小。

由于图片采用RGB三通道，所以先把图片中每个像素点的通道值归约到[0,1].，结果以numpy的矩阵存储。

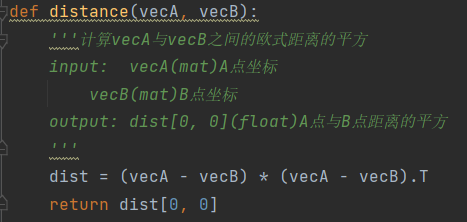


1. K-means算法原理

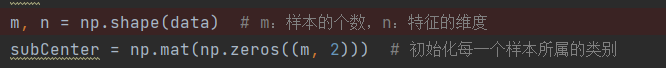
由于我们实现的是K-means算法而不是K-means++，所以在第一步我们先随机初始化中心点。使用np.random方法在通道每一维的最大值和最小值之间随机生成。



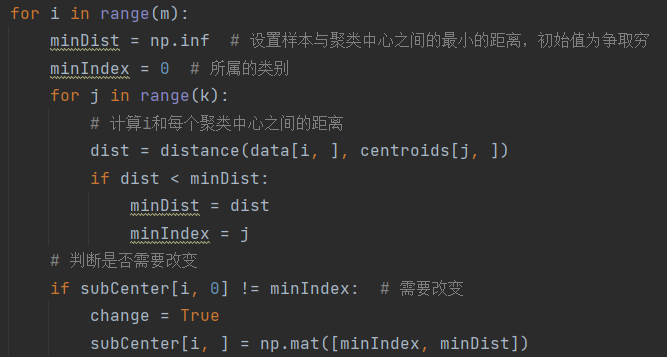
我们采用欧式距离作为评判距离的标准。具体实现如下：将两个向量做差后乘上他们的转置向量就得到欧氏距离的平方。

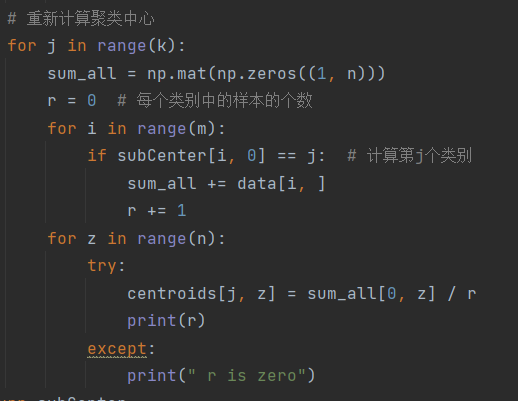


接下来就是K-means的具体实现。首先我们初始化一个矩阵，行代表像素点，第一列代表该像素点当前所属的类，第二列代表该像素点与那个类对应的中心的距离。



然后就是循环遍历中心，计算每个像素点到中心的距离然后取距离最小的那个中心所在类为像素点所在类。

在完成了对所有像素点的遍历后，我们需要计算新的中心点。这里采用了简单的取平均方式，遍历每个类别中每个像素点的三维坐标然后对每一维分别取平均。



循环退出的条件是在经历了类别判断的循环后，每个像素点的类别是否发生变化，如果有一个像素点的类别发生变化就说明整个算法还没有收敛，需要进一步计算。

1. 图像生成

在完成了K-means算法的计算之后，我们已经获取了最后形成的k个聚类的像素点的三维坐标，不过这个坐标在第一部分中被归约到了[0,1]，所以需要再\*256以回到原本通道值。然后我们取出每个像素值对应的类别，根据类别将它们的RGB通道值改为对应中心的RGB通道值，最后用PIL把这些像素点拼成新的图片。这样就完成了K-means算法。

以下是结果展示：

原图：



K=2:



K=5:

