

数字图像处理编程报告

K-means 算法实现基于颜色的图像分割

2021/11/7

实验环境

1. 操作系统: Ubuntu 18.04 x64
2. IDE: Code::Blocks
3. 编译器: GCC

实验原理

K-means 算法是科学和工业应用中用于解决聚类问题的标准解决方案。

给定数字图像和一个整数 K ，可以使用 **K-means** 聚类算法将图像像素划分为 K 个组，使得同一组中的像素在颜色上相似。

本程序读取以常见格式 (JPEG、PNG、BMP、GIF...) 保存的数字图像和整数 K ，能生成初始的分段版本图像，其中 K 个具有相似颜色像素的不同区域是可区分的。

算法流程

1. 初始化：从初始图像中随机选取 K 个像素并设置为聚类中心
2. 分配像素：每个像素被分配到它最近的聚类，即欧氏距离最小
3. 更新聚类中心：通过计算每个聚类的像素的平均值来更新集群中心
4. 图像更新：将初始图像的每个像素的 RGB 值替换为其聚类中心的值

主要算法实现

1. 通过检测输入图像获得的颜色值，存储在内存中，构成进行分割的初始数据集。矩阵的每一行包含图像每个像素的 0 到 255 个 RGB 值。给定宽度为 W 和高度为 H 的数字图像，初始数据集是由 $N = W \times H$ 行和三列RGB组成的矩阵。
2. **集群中心的初始化**。从数据集中随机选取 K 个像素并设置为初始聚类中心。在内存中分配了一个 K 行矩阵centers来存储聚类中心的值。

```
void init_centers(byte_t *data, double *centers, int n_px,
                  int n_ch, int n_clus)
{
    int k, ch, rnd;

    for (k = 0; k < n_clus; k++) {
        rnd = rand() % n_px;

        for (ch = 0; ch < n_ch; ch++) {
            centers[k * n_ch + ch] = data[rnd * n_ch + ch];
        }
    }
}
```

3. **将每个像素分配给最近的集群。**对于数据集的每个像素，计算与所有聚类中心的距离。找到距离值最小的聚类中心，并将像素分配给该聚类。一个像素与聚类中心之间的距离定义为 RGB 颜色分量值的平方差之和。为了在算法的每次迭代中记录每个像素所属的聚类，定义一个大小为 N 的数组 label 并存储在内存里，另一个由 N 个元素组成的数组 dist 用于保存每个像素到其所属聚类中心的距离。

```
void assign_pixels(byte_t *data, double *centers, int *labels,
double *dists, int *changes, int n_px,
int n_ch, int n_clus)
{
    int px, ch, k;
    int min_k, tmp_changes = 0;
    double dist, min_dist, tmp;

    for (px = 0; px < n_px; px++) {
        min_dist = DBL_MAX;

        for (k = 0; k < n_clus; k++) {
            dist = 0;

            for (ch = 0; ch < n_ch; ch++) {
                tmp = (double)(data[px * n_ch + ch] -
                    centers[k * n_ch + ch]);
                dist += tmp * tmp;
            }

            if (dist < min_dist) {
                min_dist = dist;
                min_k = k;
            }
        }

        dists[px] = min_dist;

        if (labels[px] != min_k) {
            labels[px] = min_k;
            tmp_changes = 1;
        }
    }

    *changes = tmp_changes;
}
```

3. **更新聚类中心。**对于每个聚类，通过计算属于该聚类的所有像素的平均值来重新计算中心值。在聚类中心矩阵 centers 中，每个中心都被更新，使得每个颜色分量都是属于该聚类的像素的各个颜色分量的平均值。在聚类为空的情况下，将其新中心设置为与当前集群中心距离较远的数据集像素。

```
void update_centers(byte_t *data, double *centers, int *labels,
double *dists, int n_px, int n_ch, int n_clus)
{
    int px, ch, k;
    int *counts;
    int min_k, far_px;
    double max_dist;
```

```

counts = malloc(n_clus * sizeof(int));

// 重设聚类中心矩阵

for (k = 0; k < n_clus; k++) {
    for (ch = 0; ch < n_ch; ch++) {
        centers[k * n_ch + ch] = 0;
    }

    counts[k] = 0;
}

// 计算每个聚类的颜色分量的平均值
for (px = 0; px < n_px; px++) {
    min_k = labels[px];

    for (ch = 0; ch < n_ch; ch++) {
        centers[min_k * n_ch + ch] += data[px * n_ch + ch];
    }

    counts[min_k]++;
}

for (k = 0; k < n_clus; k++) {
    if (counts[k]) {
        for (ch = 0; ch < n_ch; ch++) {
            centers[k * n_ch + ch] /= counts[k];
        }
    } else {
        // 若聚类为空，则选择较远的像素点作为集群中心

        max_dist = 0;

        for (px = 0; px < n_px; px++) {
            if (dists[px] > max_dist) {
                max_dist = dists[px];
                far_px = px;
            }
        }

        for (ch = 0; ch < n_ch; ch++) {
            centers[k * n_ch + ch] = data[far_px * n_ch + ch];
        }

        dists[far_px] = 0;
    }
}
free(counts);
}

```

4. 重复执行步骤三，直到聚类分配不再改变，说明算法收敛；或达到规定的最大迭代次数后，停止运行。
5. **更新图像数据。**图像中每个像素的 RGB 值被替换为该像素所属的聚类中心的 RGB 值。数据矩阵 data 被更新，用作生成最终分割图像的 RGB 值。

```
void update_data(byte_t *data, double *centers, int *labels, int
n_px, int n_ch)
{
    int px, ch, min_k;

    for (px = 0; px < n_px; px++) {
        min_k = labels[px];

        for (ch = 0; ch < n_ch; ch++) {
            data[px * n_ch + ch] = (byte_t)round(centers[min_k * n_ch +
ch]);
        }
    }
}
```

运行结果



原测试图片



K = 4 时的分割图像



K = 8 时的分割图像



K = 16 时的分割图像



K = 32 时的分割图像

分析:

K值影响了生成图像的分割精度。K = 4 分割时，场景的主要元素（背景中的马、草、山和天空的形状）能被清晰识别。当 K 值增大时，分割后的图像开始与原始图像相似。