

1. 文件结构介绍

参考：

[基于Kmeans聚类算法实现图像分割（从原理开始实现） k均值聚类分割颜色的原理-CSDN博客](#)



[K-Means \(daya-jin.github.io\)](#)






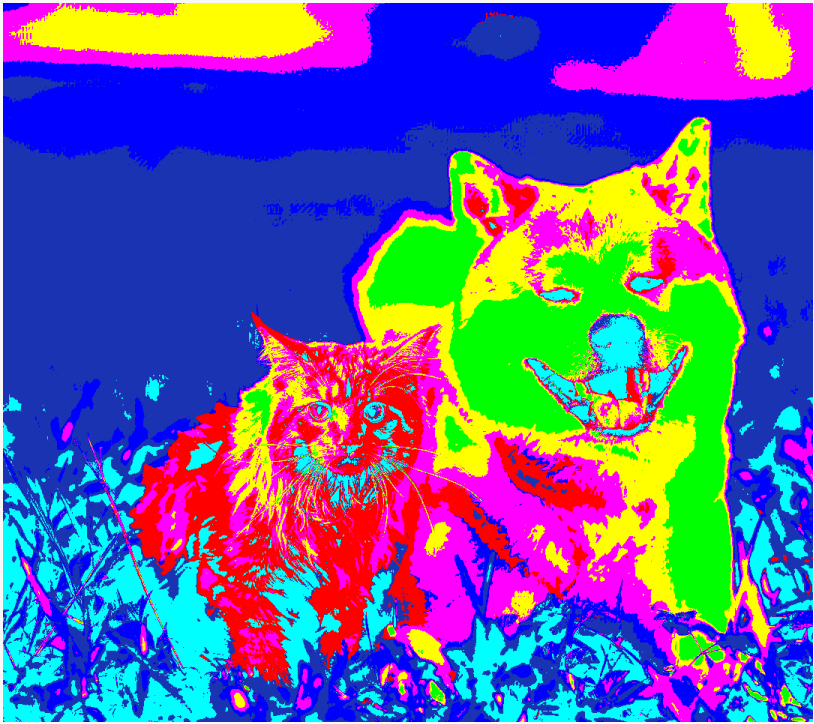
peiqi.png和dogcat.png是两张原始图片，都尝试了下发现peiqi.png比较简单很容易分出来，最后展示的是dogcat.png的结果，原始图片如下：




2. 实验结果列表

n_clusters_	epoch	output
2	16	
3	40	

n_clusters_	epoch	output
4	33	
5	34	

n_clusters_	epoch	output
6	48	
7	70	

n_clusters_	epoch	output
8	60	

3. 算法原理

3.1. 初始化

- 选择K个初始聚类中心：从数据集中随机选择K个样本作为初始的聚类中心（质心）。

3.2. 分配样本到最近的聚类中心

- 计算样本与聚类中心的距离：对数据集中的每个样本，计算它与每个聚类中心的距离（通常使用欧氏距离）。
- 将样本分配到最近的聚类中心：将每个样本分配给离它最近的聚类中心所代表的簇。

3.3. 更新聚类中心

- 计算每个簇的新聚类中心：对于每个簇，计算该簇内所有样本的平均值，得到新的聚类中心。
- 更新聚类中心：将每个簇的聚类中心更新为新计算出的聚类中心。

3.4. 重复迭代直至收敛

- 重复步骤2和3：将样本重新分配到最近的聚类中心，并更新聚类中心，直到满足停止条件。
- 停止条件：通常为达到最大迭代次数或聚类中心不再发生显著变化。

3.5. 聚类结果

- 得到最终的聚类结果：当停止条件满足时，每个样本被分配到了一个簇，形成了最终的聚类结果。

在实际的代码中，则是将图片的每个像素点的RGB作为属性进行聚类分析，最后用不同的颜色绘制图片的像素点，使其相同的簇用一个颜色，从而获得最终的可视化结果

4. 代码说明

基本遵循了以上的步骤进行复现，一些实验过程发现的问题和解决在第五部分会进行说明

```
1  import cv2
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5
6  class MyKMeans:
7      # n_clusters: 聚类中心个数， 默认为 8
8      # max_iter: 最大迭代次数， 默认为 400
9      def __init__(self, n_clusters: int = 8, max_iter: int = 400,
10 init_gen_center_way: str = 'kmeans'):
11         self.n_clusters = n_clusters
12         self.max_iter = max_iter
13         # self.tolerance = tolerance 可以考虑用，相当于一个聚类中心移动量的阈值
14         self.init_gen_center_way = init_gen_center_way
15
16         self.cluster_center_list = None
17         self.sample_label_list = None
18         self.sample_label_list_prev = None
19         self.dist = None
20
21     def __gen_init_k_center(self, x_train):
22         n_sample = x_train.shape[0] # 样本个数
23         n_feature = x_train.shape[1] # 特征个数，即RGB三个通道，每个样本的特征
24
25         if self.init_gen_center_way == 'kmeans':
26             # 使用Kmeans的方式(random)初始化聚类中心
27             # 为了在数据范围内产生随机k个聚类中心，首先计算各特征的统计量
28             # 否则可能超出范围
29             f_mean = np.mean(x_train, axis=0) # 每一列(r/g/b)的均值
30             f_std = np.std(x_train, axis=0)
31
32             print("f_mean: ", f_mean)
33             print("f_std: ", f_std)
34
35             # np.random.randn(x, y)生成x行y列的随机数，服从标准正态分布
36             self.cluster_center_list = f_mean +
37             np.random.randn(self.n_clusters, n_feature) * f_std
38
39             # elif self.init_gen_center_way == 'kmeans++':
40             #     # 使用随机的方式初始化聚类中心
41             #     # 第一个质心随机选
42             #     idx = np.random.randint(0, n_sample)
43             #     first_random_center = [x_train[idx, :]]
44             #
45             #     # 按算法选出后面k-1个质心
46             #     for i in range(1, self.n_clusters):
```

```

45
46     def fit(self, x_train):
47         n_sample = x_train.shape[0]
48
49         # 生成初始聚类中心
50         self.__gen_init_k_center(x_train)
51
52         # 初始化样本到聚类中心的距离为0
53         # 样本数 * 聚类中心数, 每个样本到每个聚类中心的距离
54         self.dist = np.zeros((n_sample, self.n_clusters))
55         # 初始化样本的标签为-1
56         self.sample_label_list = np.zeros(n_sample) - 1
57
58         epoch = 0
59         # epoch 小于最大迭代次数, 且聚类中心移动距离大于阈值时, 继续迭代
60         while epoch < self.max_iter:
61
62             # 计算每个样本到每个聚类中心的距离
63             for i in range(self.n_clusters):
64                 # x_train是n_sample * n_feature的矩阵, 每一行是一个样本
65                 # axis=1, 按行求范数, 消去n_feature, 只剩下n_sample, 即每个样本到聚
类中心的距离
66                 # dist[:, i]表示第i个聚类中心到所有样本的距离, 左右都是列向量
67                 self.dist[:, i] = np.linalg.norm(x_train -
self.cluster_center_list[i], axis=1)
68
69             # 遍历dist每一行, 根据距离分类
70             for i in range(n_sample):
71                 # 样本对应的类别为距离最近的聚类中心
72                 self.sample_label_list[i] = np.argmin(self.dist[i])
73
74             # 打印一些信息, 如epoch, 聚类中心
75             print('epoch: ', epoch)
76             print('cluster_center_list: ', self.cluster_center_list)
77             print()
78
79             # 比较分类是否发生变化, 如果不变则退出
80             if np.array_equal(self.sample_label_list,
self.sample_label_list_prev):
81                 break
82
83             # 保存下现在的标签情况
84             self.sample_label_list_prev = np.copy(self.sample_label_list)
85
86             # 计算新的聚类中心
87             for i in range(self.n_clusters):
88                 # 按列求均值, 即求每个聚类中心的新位置
89                 # axis=0, 按列求均值, 消去n_sample, 只剩下n_feature, 即每个聚类中心
的新位置
90                 self.cluster_center_list[i] =
np.mean(x_train[self.sample_label_list == i], axis=0, keepdims=True)
91                 # 注意这里可能有空簇, 即某个聚类中心没有样本, 这时候均值为nan, 需要处
理
92                 # 一种考虑将其删除 (不能删? 删了还算kmeans吗?)
93                 # 一种是为其重新生成一个随机聚类中心
94                 # 这里采用第二种方法

```

```

95         if np.isnan(self.cluster_center_list[i]).any():
96             # 为了在数据范围内产生随机k个聚类中心，首先计算各特征的统计量
97             # 否则可能超出范围
98             f_mean = np.mean(x_train, axis=0)
99             f_std = np.std(x_train, axis=0)
100             # 生成新的聚类中心: f_mean + k*f_std, 其中 k 是一堆服从标准正
           态分布的随机变量
101             self.cluster_center_list[i] = f_mean +
           np.random.randn(*f_mean.shape) * f_std
102
103         epoch += 1
104
105         # 输入测试集，输出预测的聚类标签
106         # 图像分割用不到这个函数
107         # def predict(self, x_test):
108         #     n_sample = x_test.shape[0]
109         #     dist_test = np.zeros((n_sample, self.n_clusters))
110         #
111         #     for i in range(self.n_clusters):
112         #         dist_test[:, i] = np.linalg.norm(x_test -
           self.cluster_center_list[i], axis=1)
113         #     clus_pred = np.argmin(dist_test, axis=1)
114         #
115         #     return clus_pred
116
117
118     if __name__ == '__main__':
119         # 读取图片， 转换为RGB格式
120         img_name = 'peiqi'
121         img = cv2.imread(img_name + '.png')
122         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
123
124         # 显示图片
125         # plt.figure(figsize=(10, 10))
126         # plt.imshow(img)
127         # plt.axis('off') # 不显示坐标轴
128         # plt.show()
129
130         # 获取图片的高和宽
131         w, h = img.shape[0], img.shape[1]
132
133         # 打印图片的高和宽
134         print('w: ', w)
135         print('h: ', h)
136
137         # img本来是w * h * 3的三维矩阵，现在转换为w * h行，3列的二维矩阵
138         img = img.reshape((w * h, 3))
139
140         n_clusters_ = 9
141         max_iter_ = 200 # 因为后面聚类有判断如果是聚类中心不变了就停止，所以设大一点无所谓
142
143         # 聚类
144         kmeans = MyKMeans(n_clusters_, max_iter_, init_gen_center_way='kmeans')
145         kmeans.fit(img)
146
147         # 给不同类别准备的颜色，一共10种颜色

```



```

148     colors = [
149         [0, 0, 1.], # 蓝色
150         [1., 0., 0.], # 红色
151         [0., 1., 0.], # 绿色
152         [1., 1., 0.], # 黄色
153         [0., 1., 1.], # 青色
154         [1., 0., 1.], # 品红色
155         [0.7, 0.2, 0.1], # 橙色
156         [0.2, 0.7, 0.1], # 橄榄绿
157         [0.9, 0.5, 0.2], # 棕色
158         [0.5, 0.5, 0.5] # 灰色
159     ]
160
161     # 把每个像素点的颜色换成聚类中心的颜色
162     output = np.zeros((w, h, 3))
163     for i_ in range(w):
164         for j_ in range(h):
165             output[i_, j_, :] = colors[int(kmeans.sample_label_list[i_ * h
+ j_])]
166     output = output * 255
167
168     # 显示图片
169     # plt.figure(figsize=(10, 10)) # 设置窗口大小
170     # plt.imshow(output.astype(np.uint8))
171     # # plt.axis('off') # 不显示坐标轴
172     # plt.show()
173     # 保存图片到dogcat目录下
174     cv2.imwrite(img_name + '/' + img_name + '_kmeans_' + str(n_clusters_) +
'.png', output)

```

5. 补充说明

5.1. 支持Kmeans++

如果设置init_gen_center_way为kmeans++则使用kmeans++的方式进行初始化

5.2. 计算聚类中心出现NaN

```

# 注意这里可能有空簇，即某个聚类中心没有样本，这时候均值为nan，需要处理
# 一种是考虑将其删除（不能删？删了还算kmeans吗？）
# 一种是为其重新生成一个随机聚类中心
# 这里采用第二种方法
if np.isnan(self.cluster_center_list[i]).any():
    # 为了在数据范围内产生随机k个聚类中心，首先计算各特征的统计量
    # 否则可能超出范围
    f_mean = np.mean(x_train, axis=0)
    f_std = np.std(x_train, axis=0)
    # 生成新的聚类中心：f_mean + k*f_std，其中 k 是一堆服从标准正态分布的随机变量
    self.cluster_center_list[i] = f_mean + np.random.randn(*f_mean.shape) * f_std

```

解决方案是重新生成一个随机的聚类中心

