**武汉理工大学**

|  |
| --- |
| **实** |
| **验** |
| **报** |
| **告** |

|  |  |
| --- | --- |
| 课程名称： | 数字图像处理A |
| 学 期： | 2022-2023学年第2学期 |
| 小组成员： | 何江洋 0122111371113  程明超 0122110870608 |
|  | 戴嘉讯 0122114670916  刘琪 0122114671216  黎宇航 0122110870710 |
| 实验名称： | 图像聚类实验 |

2023 年 5 月 20 日

**目 录**

1. 小组成员和分工情况…………………………………………….3
2. 实验环境………………………………………………………….3
3. 实验内容………………………………………………………….3
4. 实验原理分析…………………………………………………….3
5. 实验结果………………………………………………………….6
6. 实验总结………………………………………………………….15
7. 代码附录………………………………………………………….15
8. 小组成员和分工情况：

何江洋：主要负责完成利用K-means++算法对cifar10数据集进行聚类分割。

刘琪：主要负责完成利用K-means算法对单张图片的聚类分割。

程明超：辅助完成K-means++算法实现，主要负责分类结果的分析计算，撰写实验报告。

戴嘉讯：辅助完成K-means算法，辅助完成实验报告。

黎宇航：辅助完成K-means++算法，辅助完成实验报告。

2．实验环境

本次实验主要完成K-means算法，为了更加清楚的表达K-means算法的整体机制，此次实验使用python语言实现。

在实验的过程中，引入了cv2和torch等相关的包，对图像的处理非常方便。

3.实验内容

3.1利用K-means算法实现对单张图像的聚类分割。

3.2利用K-means++算法实现对cifar10上数据集的分类。

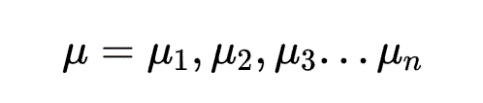
4.实验原理分析

4.1K-means算法原理分析如下：

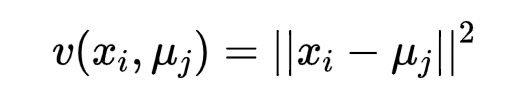
K-means算法又称K均值算法。其基本步骤如下：

1. 从样本集中随机选取K个样本作为簇中心
2. 计算每个样本点到K个聚类中心的相应距离。
3. 选择与中心距离最小的点作为该中心簇的样本点。
4. 对每个簇的聚类中心按照该簇的样本点进行调整。

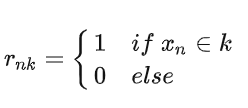
我们对K-means聚类算法的收敛的数学证明如下：

首先指定样本分为n个簇，那么可以初始化一组每个簇对应的中心。

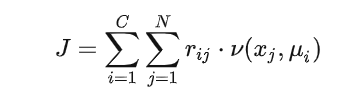
其次，我们计算每个样本点与这n个簇样本中心之间的距离。



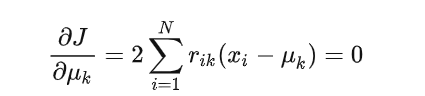
然后对于每个样本点选取距离某个中心**ai**的最短距离，作为该簇的样本点。那么样本点与中心簇之间存在以下关系：



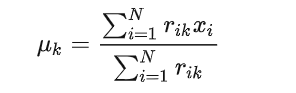
此时便可以构造出整个K-means算法的损失函数：



此时我们的任务转化为，求出损失函数值最小的情况下每个中心簇的样本簇内距离最小。我们对损失函数求导，并令其为0。



那么可以得到第k个中心点为：



由此可以看出，K-means算法和EM算法具有异曲同工之妙，都是先初始化中心点然后进行估计后再对原来初始化的中心点进行更新操作，不断迭代下去，直到中心的变化不大为止。

4.2.K-means算法的优缺点

优点：

（1）容易理解，聚类效果能够接受。

（2）处理大数据集时，该算法仍然能够保持较好的伸缩性。

（3）当簇近似为高斯分布时，聚类效果非常出色。

（4）算法的复杂度相对其他算法较低。

缺点：

1. K值需要认为首先进行初始化，存在随机性，不同K值，对应最终聚类的结果会截然不同。
2. 对初始的中心簇的初始化非常敏感，初始化的好坏很大程度上会影响算法的迭代次数。
3. 对异常值十分敏感，受异常值影响较大。
4. 不适合数据集太离散，样本类别不平衡，非凸分类等。

4.3.K-means算法的关键点。

4.3.1 数据预处理

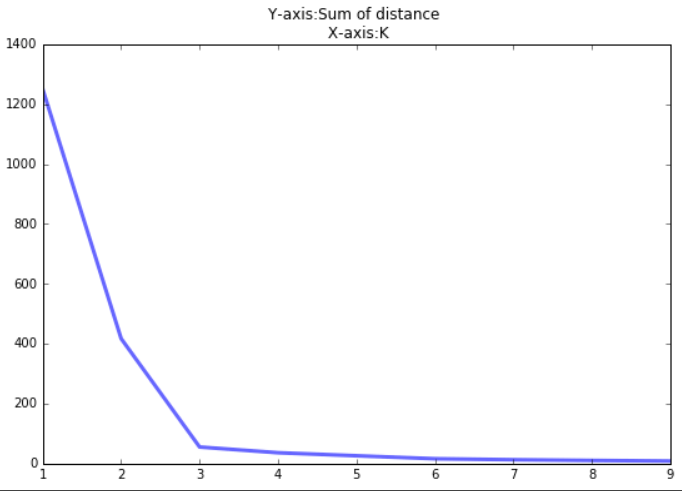
K-means算法本质上是基于欧氏距离的数据划分算法均值和方差大的维度将对数据的聚类产生决定性影响。所以未做归一化处理和统一单位的数据是无法直接参与运算和比较的。常见的数据预处理方式有：数据归一化，数据标准化。

此外，离群点或者噪声数据会对均值产生较大的影响，导致中心偏移，因此我们还需要对数据进行异常点检测。

4.3.2 K值的选取

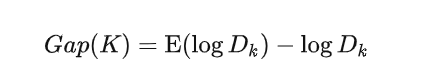
常见的K值选取法有手肘法和Gap statistic 方法。

手肘法是指选取K值使得聚类情况不再急剧变化时，即为合理的K值。



在此图中，K=3后曲线趋于平缓，聚类不再发生剧烈变化，因此K=3即为一个较为合理的K值。但是手肘法，需要人对K值得合理性进行判断。

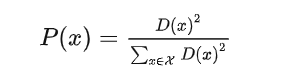
Gap statistic 方法能够解决手肘法的缺点。



其中 DK为损失函数，这里E(logDk)指的是logDk的期望。这个数值通常通过蒙特卡洛模拟产生，我们在样本里所在的区域中按照均匀分布随机产生和原始样本数一样多的随机样本，并对这个随机样本K-Means，从而得到一个Dk。如此往复多次，通常20次，我们可以得到20logDk。对这2个数值求平均值，就得到E（logDk）的近似值。最终可以计算 Gap Statisitc。而 Gap statistic 取得最大值所对应的 K 就是最佳的 K。

4.2 K-means++算法分析如下

K-means++算法是在K-means算法上的改进，仍然是首先初始化选出n个聚类中心，不同点在于，我们需要把每个样本点距离每个聚类中心的最短距离记为D（X），计算出下一个聚类中心的概率P（X），其中：



选取概率最大的样本点作为下一个聚类中心，也就是说，我们使得每个类簇两两之间的距离尽可能大。每确定一个聚类中心后，重新计算D（X）重复上述步骤，直到全部找出n个聚类中心。

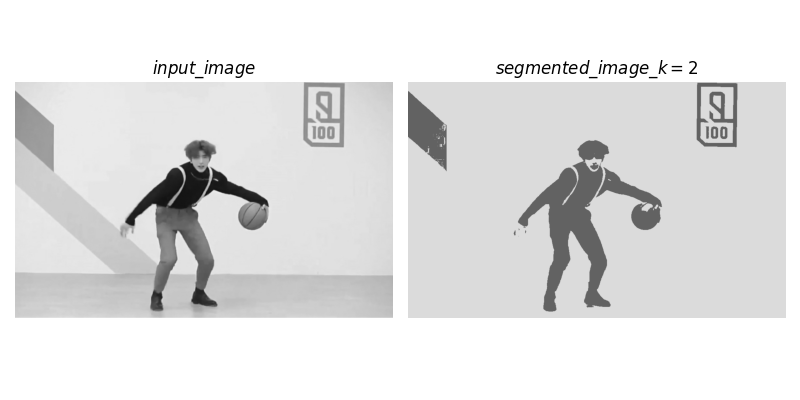
K-means++是对K-means在初始化每个聚类中心上做出了优化，由于K-means算法对聚类中心的初始化异常敏感，因此K-means++在此做出了优化，有效地解决了这一问题。

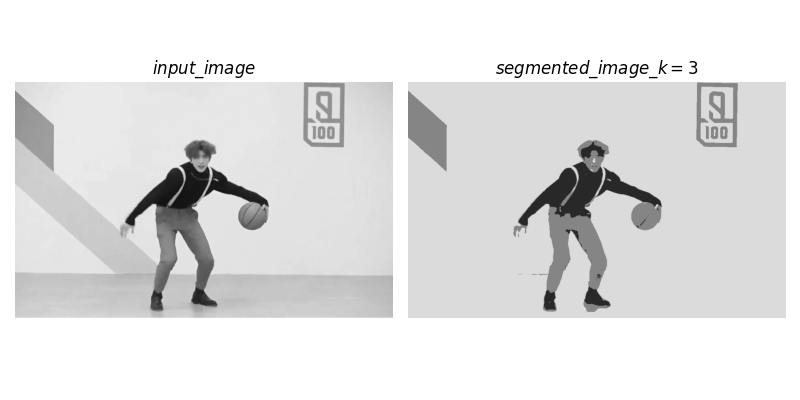
5．实验结果

利用K-means对单张图像进行分割。

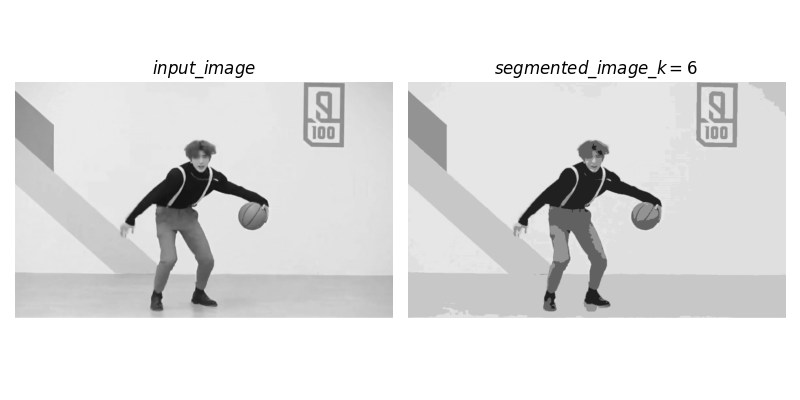
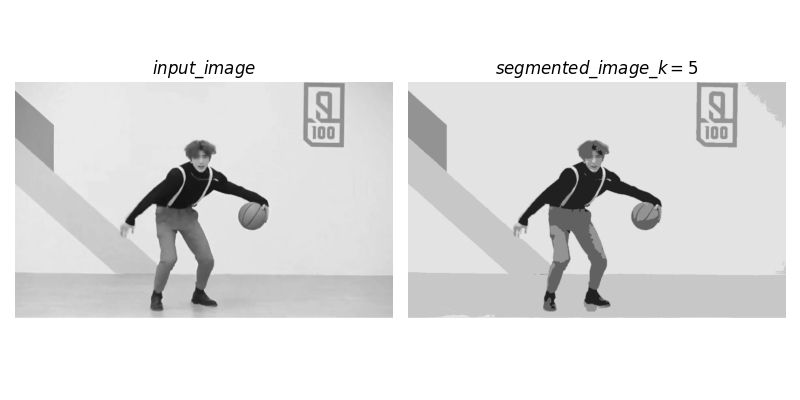
原图如下：

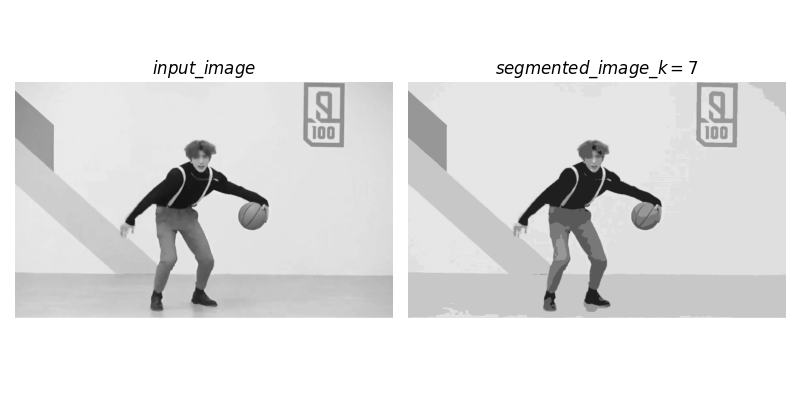






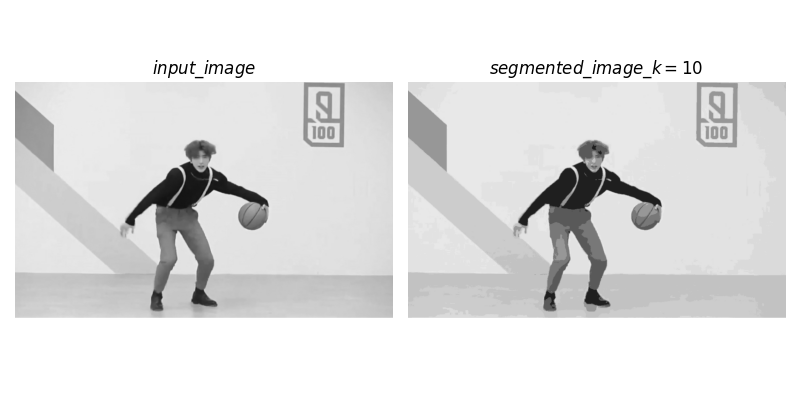










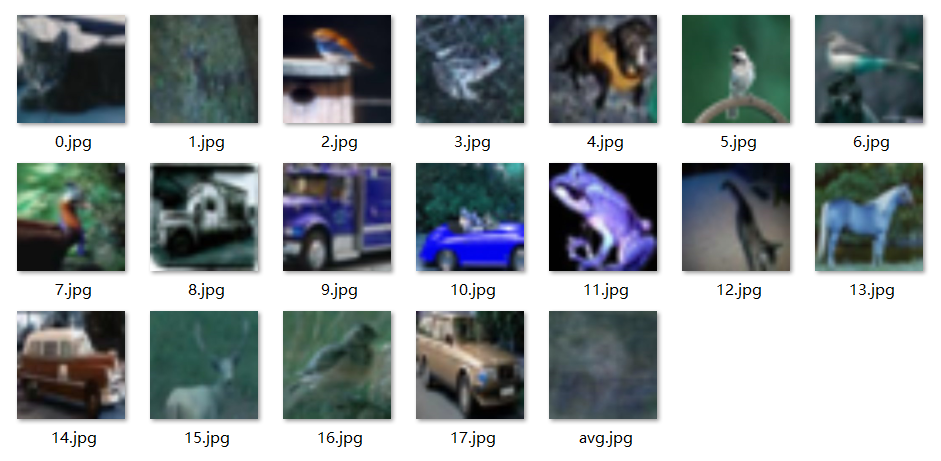


以上是不同K值下对应的图像分割效果，从图像分割效果来看，我们选择的K值越大，聚类得个数越多，图像中的细节表现得也就越好。

利用K-means++实现对Cifar10的聚类结果如下：

声明：数据集在不改变数据类数的前提下使用了数据集的子集。

聚类得到的第一类结果如下：



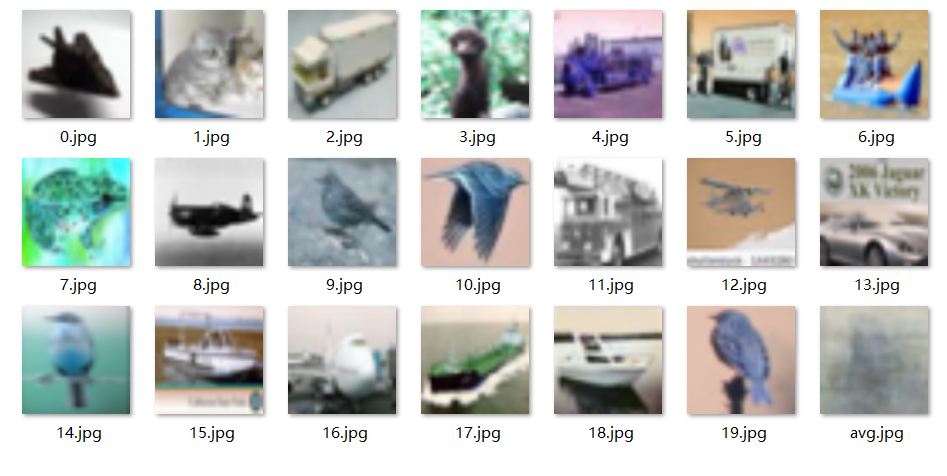
第二类：



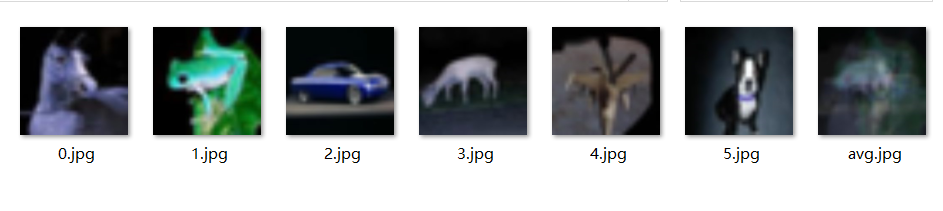
第三类：



第四类：



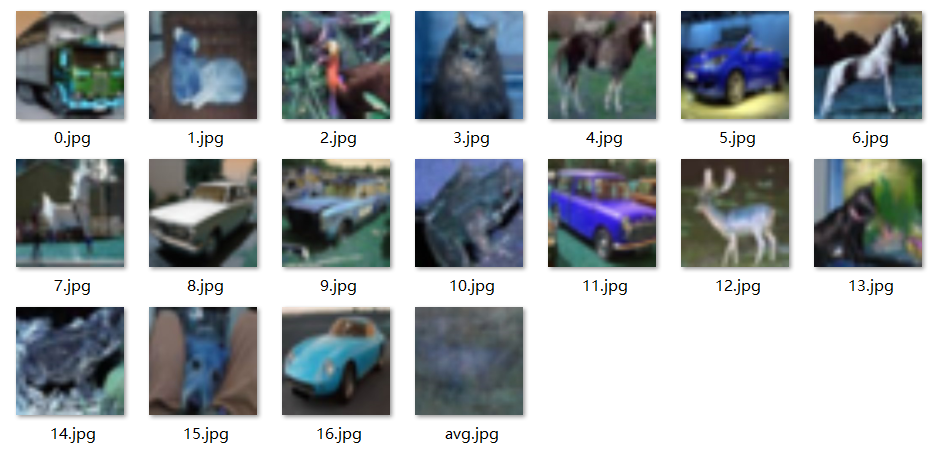
第五类：



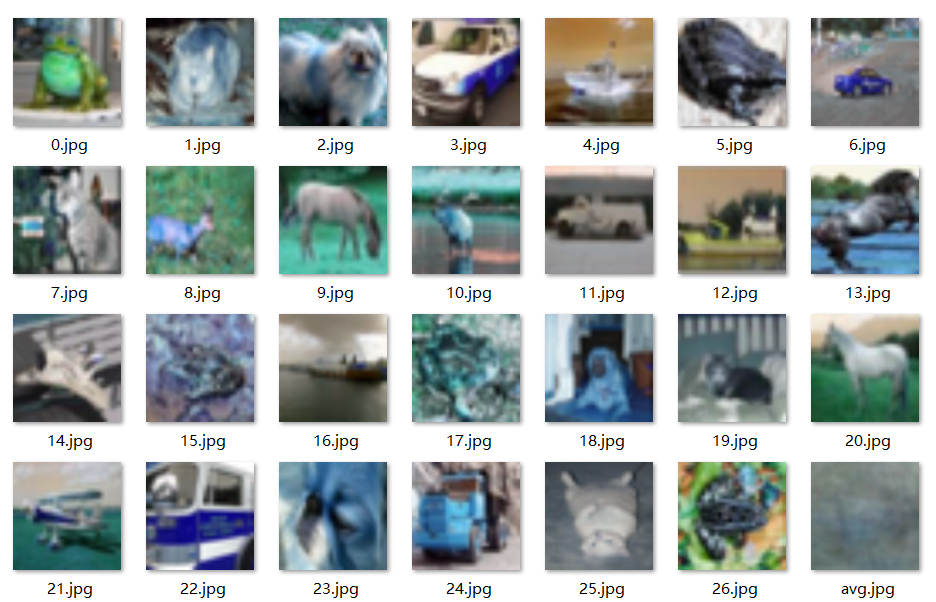
第六类：



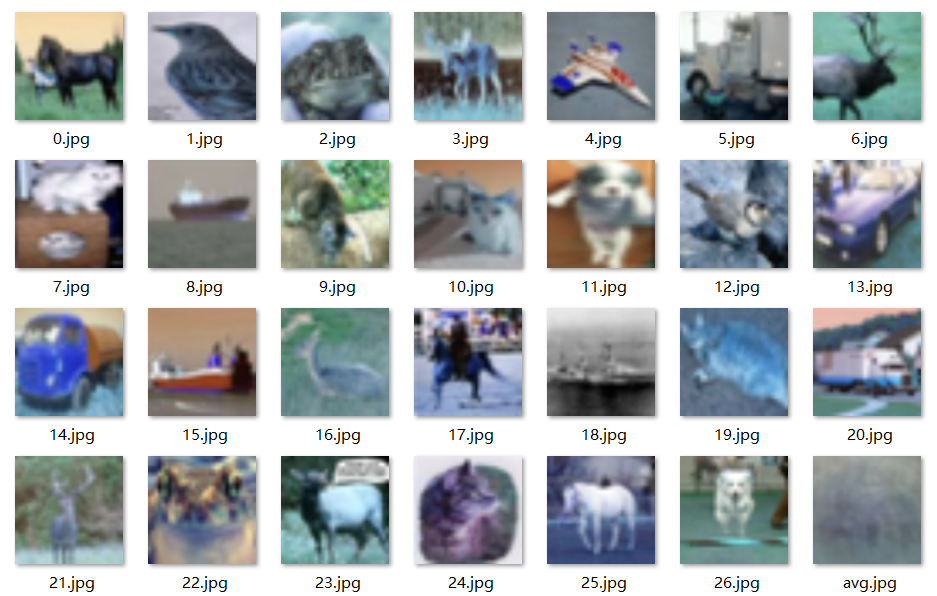
第七类：



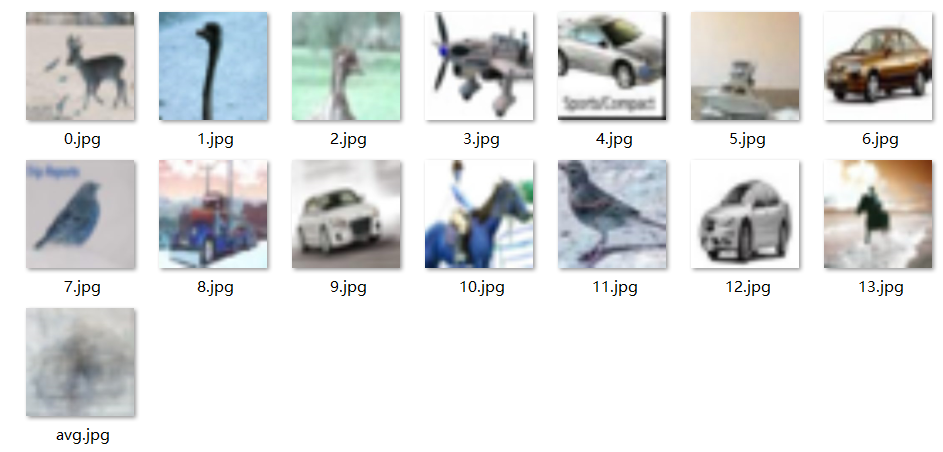
第八类：



第九类：



第十类：



在上述实验结果中，每个类中的类别标签应当从avg图像中得出，我们能够看出聚类的效果差强人意。每个类别得到正确聚类的图像基本在50%左右。这是因为K-means++算法仍然存在一些比较明显的缺点。该过程是通过比对所有数据集中的图片来完成的，所以必须将所有图片读取在内存中，**容易造成内存爆炸**；其次，对一幅图像进行判断类别，需要比对所有训练集的图像，**识别的过程消耗计算量巨大。**

6.实验总结

本次实验利用python语言实现了图像的分割和聚类，熟练掌握了K-means算法，并且大胆尝试了K-means++算法。我们从对数据集的处理到对算法的整体把握都有了较大的进步。在实验的过程中，我们不仅掌握了数字图像处理的知识，更在机器学习算法方面取得了较大的进步。同时，在配置算法运行的环境中，遇到了一些问题，对每个程序包的选择查阅了相关的资料。最终，我们成功使代码执行出了我们期待的结果。总体来说，本次实验的难度适中，我们能够克服困难成功实现，这个过程也使我们对这些算法产生了浓厚的兴趣。

7．代码附录

K-means单张图像分割：

import numpy as np

import cv2 as cv

import matplotlib.pyplot as plt

def test\_KMeans(image\_path):

    image = cv.imread(image\_path, cv.IMREAD\_GRAYSCALE)

    image = cv.cvtColor(image, cv.COLOR\_BGR2RGB)

    pixel\_value = np.float32(image.reshape((-1, 3)))

    # 终止条件

    criteria = (cv.TERM\_CRITERIA\_EPS + cv.TERM\_CRITERIA\_MAX\_ITER, 200, 0.1)

    # 起始的中心选择

    flags = cv.KMEANS\_RANDOM\_CENTERS

    # 定义簇的数量

    K = 10

    \_, labels, center = cv.kmeans(pixel\_value, K, None, criteria, 10, flags)

    center = np.uint8(center)

    # 将所有像素转换为质心的颜色

    segmented\_image = center[labels.flatten()]

    # 重塑回原始图像尺寸

    segmented\_image = segmented\_image.reshape((image.shape))

    plt.figure(figsize=(8, 4))

    plt.subplot(121)

    plt.imshow(image)

    plt.axis("off")

    plt.title(f"$input\\_image$")

    plt.subplot(122)

    plt.imshow(segmented\_image)

    plt.axis("off")

    plt.title(f"$segmented\\_image\\_" + "k=" + str(K) + "$")

    plt.tight\_layout()

    plt.savefig("segmented\_result\_" + "k=" + str(K) + ".png")

    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    test\_KMeans("1.jpg")

K-means++对cifar10聚类：

import torchvision

import torch

import numpy as np

from torch.utils.data import DataLoader

import os

import cv2

import random

def load\_data():

    transform = torchvision.transforms.Compose(

        [torchvision.transforms.ToTensor()]

    )

    train = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)

    train\_loader = torch.utils.data.DataLoader(train, batch\_size=1)

    for i, (img, \_) in enumerate(train\_loader):

        images.append(np.transpose(img[0].numpy(), (1, 2, 0)))

        if i > 200:

            break

def distance(img, centroids):

    return np.array([np.sum(centroid - img) \*\* 2 for centroid in centroids])

def kmeans\_plus(k):

    centroids = []

    idx = random.randint(0, len(images) - 1)

    centroids.append(images.pop(idx))

    for \_ in range(k - 1):

        sum = 0

        dx = np.zeros((len(images),))

        for i, img in enumerate(images):

            dx[i] = np.min([(img - centroid) \*\* 2 for centroid in centroids])

            sum += dx.sum()

        p = np.array(dx) / sum

        max\_idx = np.argmax(p)

        centroids.append(images.pop(max\_idx))

    print("finish")

    return centroids

def kmeans(k=10):

    centroids = kmeans\_plus(k)

    clu = dict()

    for epoch in range(100):

        for i in range(k):

            clu[i] = []

        for img in images:

            index = distance(img, centroids).argmin()

            clu[index].append(img)

        for i in range(k):

            sum = np.zeros\_like(img)

            for img in clu[i]:

                sum += img

            mean = sum / len(clu[i])

            centroids[i] = mean

        print(epoch)

    return clu, centroids

if \_\_name\_\_ == '\_\_main\_\_':

    images = []

    load\_data()

    clu, centroids = kmeans()

    for i in range(len(centroids)):

        os.mkdir(f"./{i}")

        ims = clu[i]

        sum = np.zeros\_like(images[0].shape)

        for idx, im in enumerate(ims):

            cv2.imwrite(f"./{i}/{idx}.jpg", cv2.resize(im \* 255, (320, 320)))

            sum = sum + im

        sum = sum / (idx + 1)

        cv2.imwrite(f"./{i}/avg.jpg", cv2.resize((sum \* 255), (320, 320)))

数据集采用Cifar10，参考网络。