

**“数字图像处理基础”实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 题目 | 实验十：基于机器学习的图像分割 | | |
| 姓名 | 钟军凯 | 学号 | 22211374 |
| 指导教师 | 彭亚辉 | | |
| 日期 | 2025年5月4日 | | |

**电子信息工程学院**

目 录

[1. 实验目的分析 3](#_Toc191473896)

[2. 实验方案设计 3](#_Toc191473897)

[3. 实验结果 3](#_Toc191473898)

[4. 分析与讨论 7](#_Toc191473899)

[5. 问题发现与探究 7](#_Toc191473900)

[6. 实验总结 9](#_Toc191473901)

[指导老师评价 1](#_Toc191473902)6

1. 实验目的分析

本实验旨在通过学习K-means聚类算法，理解其基本原理与算法流程，并掌握其在灰度图像和彩色图像分割中的应用方法。通过使用Python实现K-means图像分割，提升对聚类算法在图像处理领域中实际应用的理解与实践能力。

2. 实验方案设计

以下是我实现k均值算法的思路：

首先，通过numpy.random.randint，在图像中随机选取k个像素点，使用它们的灰度值或者(R, G, B)值作为类中心。然后初始化last\_labels和labels两个数组，前者用于记录上一次的聚类情况，后者用于记录当前的聚类情况，通过比较两个数组是否相同，来决定算法是否结束，也就是说，如果last\_labels和labels相同，则结束算法，使用numpy.array\_equal()来判断两个数组是否相同。在每一步迭代中，先计算每个像素点的特征到每一个类中心的距离，对于灰度图像来说，特征就是灰度值；对于RGB图像来说，特征就是(R, G, B)三个通道的值。计算完距离，找到与每个像素点的特征距离最近的类中心，得到对应的类别，完成对labels数组的设置。然后根据设置好的labels数组，找出图像中类别相同的像素点，取出它们的特征，求均值，然后更新类中心。然后，进行下一次迭代，直到满足迭代停止条件。

GUI设计方面，k均值算法的参数只有一个k值，所以我考虑采用滑动条的方式来调整k值。

1. 实验结果

**此处做一个声明**：分割后的图像中，颜色只表示**类别**，不表示前景或者背景，也就是说同一个类别的像素点会是同一种颜色，在灰度图像中采用不同程度的灰度，根据类别数对灰度值范围进行线性的分割，在RGB图像中，随机生成颜色来填充相同类别的像素点。所以，对于灰度图像，可能会出现，原图像中原本是背景的部分，在分割后的图像中变成白色的情况；对于彩色图像，会出现，原图像和分割后的图像之间的色彩差异较大的情况。

图1到图5为使用k均值算法对灰度图像进行分割的结果。

|  |  |  |
| --- | --- | --- |
|  |  |  |

图 1：灰度图像k均值分割，k=2。左图：原图；中间：自己编写的k均值分割的结果；右图：sklearn的KMeans的结果

|  |  |  |
| --- | --- | --- |
|  |  |  |

图 2：灰度图像k均值分割，k=3。左图：原图；中间：自己编写的k均值分割的结果；右图：sklearn的KMeans的结果

|  |  |  |
| --- | --- | --- |
|  |  |  |

图 3：灰度图像k均值分割，k=4。左图：原图；中间：自己编写的k均值分割的结果；右图：sklearn的KMeans的结果

|  |  |  |
| --- | --- | --- |
|  |  |  |

图 4：灰度图像k均值分割，k=5。左图：原图；中间：自己编写的k均值分割的结果；右图：sklearn的KMeans的结果

|  |  |  |
| --- | --- | --- |
|  |  |  |

图 5：灰度图像k均值分割，k=6。左图：原图；中间：自己编写的k均值分割的结果；右图：sklearn的KMeans的结果

图6到图10是使用k均值算法对RGB图像进行分割的结果。

|  |  |  |
| --- | --- | --- |
|  |  |  |

图 6：RGB图像k均值分割，k=2。左图：原图；中间：自己编写的k均值分割的结果；右图：sklearn的KMeans的结果

|  |  |  |
| --- | --- | --- |
|  |  |  |

图 7：RGB图像k均值分割，k=6。左图：原图；中间：自己编写的k均值分割的结果；右图：sklearn的KMeans的结果

|  |  |  |
| --- | --- | --- |
|  |  |  |

图 8：RGB图像k均值分割，k=10。左图：原图；中间：自己编写的k均值分割的结果；右图：sklearn的KMeans的结果

|  |  |  |
| --- | --- | --- |
|  |  |  |

图 9：RGB图像k均值分割，k=14。左图：原图；中间：自己编写的k均值分割的结果；右图：sklearn的KMeans的结果

|  |  |  |
| --- | --- | --- |
|  |  |  |

图 10：RGB图像k均值分割，k=20。左图：原图；中间：自己编写的k均值分割的结果；右图：sklearn的KMeans的结果

4. 分析与讨论

通过对比自己编写的k均值分割的结果和sklearn的KMeans的结果，二者的分割结果大致相同。

在代码运行速度方面，在k值较大的时候，灰度图像的分割要明显快于RGB图像，无论是自己编写的代码还是sklearn中的KMeans。对比不同代码之间的速度，从直接的感觉上来说，自己编写的代码和使用sklearn两种情形下的速度没有显著差异，包括在k值较大的时候，分割RGB图像的情形。

5. 问题发现与探究

**5.1 问题一**

**5.1.1 问题描述**

出现类中心在几组值中循环的情况，导致迭代停止不了。也就是如果有两个类中心，两个类中心会在[a，b]，[c，d]，[e，f]等几组数值之间循环。

**5.1.2 问题解决**

我在原来的代码中，对类中心数组label\_centers是采取了数据类型转换的，也就是label\_centers = label\_centers.astype(np.uint8)。我将类中心限制在了无符号8位整数中，但是这会导致数值丢失，比如说在更新类中心的时候，如果限制为uint8类型，那么小数部分就会丢失，所以原本不相同的数字，此时就可能相同了，所以我将np.uint8改为了np.float64，让label\_centers为小数数组，此时，不能退出迭代的问题就解决了。

**5.2 问题二**

**5.2.1 问题描述**

在进行测试程序的时候，我发现每当我回到某一个参数的情形下的k均值算法的时候，分割结果可能会有不同。如图11和图12所示。而且，我在程序中使用了numpy.random.seed来设置随机数生成器的状态，想要使得每次生成的随机数相同。

|  |  |
| --- | --- |
|  |  |

图 11：k均值分割，k=4

|  |  |
| --- | --- |
|  |  |

图 12：k均值分割，k=5

**5.2.2 问题探究**

通过查阅资料发现，当初始点不同的时候，即使参数相同，k均值算法的结果也有可能会不同，因为初始点不同会导致迭代过程不同。而且，k均值算法的目标是最小化簇内元素的特征之间的距离，这是一个局部的值，所以k均值算法很难达到全局最优。

关于numpy.random.seed的问题，numpy.random.seed能够保证不同程序之间或者同一份程序在不同时刻的运行，生成的随机数是相同的，但是如果是在一份程序的运行期间，生成的随机数还会不同，所以出现了回到某一个参数，分割结果和上一次可能不同的情况。

6. 实验总结

通过这次实验，我对K-means聚类算法有了更加直观的认识，不仅了解了它的基本原理和流程，还实际动手实现了基于K-means的图像分割。无论是灰度图像还是彩色图像，通过聚类都能把图像有效地划分成不同区域。整个过程让我体会到算法在图像处理中的实际效果，同时也锻炼了我用Python编程解决实际问题的能力。

附录：

【仿真程序】

|  |
| --- |
| import tkinter as tk  from tkinter import ttk  from tkinter import filedialog as fd  from PIL import Image  from PIL import ImageTk  import numpy as np  from matplotlib import pyplot as plt  from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk  from matplotlib.figure import Figure  import struct  import math  from numpy.lib.stride\_tricks import sliding\_window\_view  np.random.seed(100)  '''  @brief 获取图像数组  @param image: PIL的Image类对象  @return height: 图像的高（行数）  @return width: 图像的宽（列数）  @return image\_array: 图像对应的数组  '''  def get\_image\_data (image):  (width, height) = image.size # 获取图像的宽和高  image\_array = np.array(image) # 获取图像的像素数据并转化为数组  return height, width, image\_array  '''  @brief 读取raw文件  @param file\_name: raw文件路径  @return raw\_array: raw文件图像数组  '''  def read\_raw (file\_name):  # 获取raw文件表示的图片的数据  raw\_file = open(file\_name, "rb") # 打开文件，以只读、二进制的方式打开  raw\_width = struct.unpack("i", raw\_file.read(4))[0] # 获取raw文件表示的图片的宽度  raw\_height = struct.unpack("i", raw\_file.read(4))[0] # 获取raw文件表示的图片的高度  raw\_data = struct.unpack(f"{raw\_width \* raw\_height}B", raw\_file.read()) # 获取raw文件表示的图片的数组  raw\_file.close() # 关闭文件  # 将获取到的数组转换为二维数组  raw\_array = np.array(raw\_data).reshape((raw\_height, raw\_width))  return raw\_array  '''  @brief 显示转换后的图像  @param image\_array: 图像数组  '''  def show\_transform\_image (image\_array):  global transform\_image\_tk  # 转换  image = Image.fromarray(np.uint8(image\_array))  transform\_image\_tk = ImageTk.PhotoImage(image)  # 显示分割后的图片  transform\_image\_label.config(image = transform\_image\_tk)  def kmeans\_segmentation (image\_array, k):  # 得到图像行数和列数  rows = image\_array.shape[0]  columns = image\_array.shape[1]    # 在图像中随机选取k个点，作为最初的类中心  label\_centers\_row = np.random.randint(0, rows, size = k)  label\_centers\_column = np.random.randint(0, columns, size = k)  label\_centers = image\_array[label\_centers\_row, label\_centers\_column]  label\_centers = label\_centers.astype(np.float64)  # 初始化类别数组  last\_labels = np.full((rows, columns), -1) # 记录上一时刻的类别  labels = np.zeros((rows, columns))  while not np.array\_equal(last\_labels, labels):  distance\_list = []    # 更新last\_labels  last\_labels = np.copy(labels)  # 计算每个像素点到类中心的距离  for label\_center in label\_centers:  if image\_array.ndim == 2: # 灰度图像  distance = np.sqrt((image\_array - label\_center) \*\* 2)  elif image\_array.ndim == 3: # RGB图像  distance = np.sqrt(np.sum((image\_array - label\_center) \*\* 2, axis = -1))  distance\_list.append(distance)  # 计算类别  distances = np.stack(distance\_list, axis = 0)  labels = np.argmin(distances, axis = 0)  labels = labels + 1  # 更新类中心  for label in range(1, k + 1):  values = image\_array[labels == label]  if values.size == 0:  if image\_array.ndim == 2:  label\_centers[label - 1] = np.random.randint(0, 256)  elif image\_array.ndim == 3:  label\_centers[label - 1] = np.random.randint(0, 256, size = 3)  else:  if image\_array.ndim == 2: # 灰度图像  label\_centers[label - 1] = values.mean()  elif image\_array.ndim == 3: # RGB图像  label\_centers[label - 1] = values.mean(axis = 0)  return labels  def image\_kmeans\_segmentation ():  global k  result = np.copy(image\_array)  # k均值  labels = kmeans\_segmentation(image\_array, k)  for i in range(k):  if image\_array.ndim == 2: # 灰度图像  grayscale = int(i / (k - 1) \* 255)  result[labels == i + 1] = grayscale  elif image\_array.ndim == 3: # RGB图像  rgb\_value = np.random.randint(0, 256, size = 3)  result[labels == i + 1] = rgb\_value  show\_transform\_image(result)  transform\_image\_tip.config(text = "k均值分割")  scale\_frame.grid(row = 3, column = 1)  def update\_kmeans (k\_):  global k  k = int(k\_)  image\_kmeans\_segmentation()  def file\_operation ():  global origin\_image\_tk, image\_array  # 获取文件路径  file\_path = fd.askopenfilename()  # 获取文件格式  file\_format = []  for i in reversed(range(len(file\_path))):  if file\_path[i] == '.': # 由于只需要文件格式，所以遇到'.'就退出  break  file\_format.append(file\_path[i])  file\_format.reverse() # 列表反转  file\_format\_str = "".join(file\_format) # 转换为字符串  # 判断文件格式并得到Image对象和图像数组  if file\_format\_str == "raw":  image\_array = read\_raw(file\_path)  image = Image.fromarray(image\_array)  else:  image = Image.open(file\_path)  \_, \_, image\_array = get\_image\_data(image)  # 转换为tkinter能解析的PhotoImage对象  origin\_image\_tk = ImageTk.PhotoImage(image)  # 显示图像  origin\_image\_label.config(image = origin\_image\_tk)  origin\_image\_tip.config(text = "原图像")  transform\_image\_label.config(image = "")  transform\_image\_tip.config(text = "")  scale\_frame.grid\_forget()  if \_\_name\_\_ == "\_\_main\_\_":  # 初始化  k = 2  # 创建基本界面  root = tk.Tk()  root.title("数字图像处理实验十: 基于机器学习的图像分割") # 设置界面标题  root.grid()  root.grid\_columnconfigure(0, weight = 1) # root的第一列会适应界面大小的改变  root.grid\_rowconfigure(0, weight = 1) # root的第一行会适应界面大小的改变  # 创建一个画布，画布可以被滚动条控制  canvas = tk.Canvas(root)  canvas.grid(row = 0, column = 0, sticky = "nsew") # canvas会填充整个界面  # 创建滚动条  scrollbar\_1 = ttk.Scrollbar(root, orient = "vertical", command = canvas.yview)  scrollbar\_1.grid(row = 0, column = 1, sticky = "ns") # 纵轴方向填充  scrollbar\_2 = ttk.Scrollbar(root, orient = "horizontal", command = canvas.xview)  scrollbar\_2.grid(row = 1, column = 0, sticky = "ew") # 横轴方向填充  # canvas与滚动条关联  canvas.config(yscrollcommand = scrollbar\_1.set)  canvas.config(xscrollcommand = scrollbar\_2.set)  # 创建Frame容器，用于存放各种子容器  frame = ttk.Frame(canvas, padding = 10)  frame.grid(row = 0, column = 0)  canvas.create\_window((0, 0), window = frame, anchor = "nw") # 将frame嵌入canvas  frame.bind("<Configure>", lambda event: canvas.configure(scrollregion = canvas.bbox("all"))) # 使滚动条适应frame  # 创建存放按键的Frame容器  button\_frame = ttk.Frame(frame, padding = 10)  button\_frame.grid(row = 0, column = 0)  # 创建“打开文件”按键  open\_file\_button = ttk.Button(button\_frame, text = "打开文件", command = file\_operation)  open\_file\_button.grid(row = 0, column = 0)  # 创建功能按键  kmeans\_button = ttk.Button(button\_frame, text = "k均值分割", command = image\_kmeans\_segmentation)  kmeans\_button.grid(row = 1, column = 0)  # 创建“退出”按键  quit\_button = ttk.Button(button\_frame, text = "退出", command = root.destroy)  quit\_button.grid(row = 2, column = 0, columnspan = 4)  # 创建图像标签  origin\_image\_label = ttk.Label(frame)  origin\_image\_label.grid(row = 1, column = 0)  transform\_image\_label = ttk.Label(frame)  transform\_image\_label.grid(row = 1, column = 1)  # 创建图像标注  origin\_image\_tip = ttk.Label(frame)  origin\_image\_tip.grid(row = 2, column = 0)  transform\_image\_tip = ttk.Label(frame)  transform\_image\_tip.grid(row = 2, column = 1)  # 创建存放滑动条的Frame容器  scale\_frame = ttk.Frame(frame, padding = 10)  # scale\_frame.grid(row = 3, column = 1)  # 创建滑动条  kmeas\_scale = tk.Scale(scale\_frame, from\_ = 2, to = 30, orient = "horizontal", length = 150, command = update\_kmeans)  kmeas\_scale.grid(row = 0, column = 0)  kmeas\_scale.set(k)  root.mainloop() |

指导老师评价

|  |  |  |
| --- | --- | --- |
|  | | |
| 仿真实验报告成绩 | 交流展示成绩 | 总评成绩 |
|  |  |  |

指导教师：

年 月 日

**实验报告提交说明:**

文件名 **姓名\_学号\_实验x.docx**

文件格式 Microsoft Word