成绩	

河北大学

网络空间安全与计算机学院实验报告

实验课程名称：计算机视觉实验

装

订

线

实验项目名称：特征提取与选择

学生姓名：张晨

专　　业：人工智能

学　　号：20211205068

实验地点：C1-128

实验时间：2024.03.28 第 7-8 节

指导教师：杨文柱

预习报告部分

一、 实验目的

1. 掌握物体特征提取的主要方法；

2. 掌握基于遗传算的特征选择方法；

3. 掌握利用 Matlab/Python 进行特征提取和选择的方法；

4. 设计相应的 Matlab/Python 程序，实现特定任务下目标的特征提取与选择。

二、 实验原理

1.水果图像的分割

图像分割是计算机视觉中的基本任务之一，旨在将图像划分为多个区域，每个区域对应于一个物体或物体的部分。在本实验中，我们将设计合理的分割算法来提取单个水果。

根据选择的水果类型（如苹果、桔子或香蕉）选择或设计适用于所选水果类型的分割算法。常用的图像分割方法包括阈值分割、边缘检测、区域生长、图割等。

2.目标的特征提取

特征提取是计算机视觉中的关键步骤，旨在从图像中提取出与目标相关的有用信息。在本实验中，我们将设计合理的特征提取方法，以获取水果目标的颜色、形状和纹理等特征。

颜色特征可以通过颜色空间转换（如 RGB 到 HSV）和颜色直方图等方法来提取。形状特征则可以通过边缘检测、轮廓提取和几何参数计算等方法获得。纹理特征则可以利用灰度共生矩阵、小波变换等方法进行提取。

这些特征提取方法的选择应基于所选水果类型的特性，以及后续分类任务的需求。通过提取这些特征，我们可以将水果目标表示为特征向量，为后续的分类任务提供输入。

3.基于遗传算法的特征选择

特征选择是机器学习中的一个重要步骤，旨在从原始特征集中选择出对分类任务最有用的特征子集。在本实验中，我们将利用遗传算法来实现特征选择。

遗传算法是一种模拟自然选择和遗传机制的优化算法，通过选择、交叉和变异等操作来搜索最优解。在特征选择中，我们可以将每个特征子集表示为一个染色体，通过计算染色体的适应度（如分类准确率）来评估其优劣。然后，通过选择操作保留适应度较高的染色体，通过交叉和变异操作产生新的染色体，以逐步逼近最优特征子集。

通过遗传算法进行特征选择，我们可以有效地减少特征数量、降低计算复杂度，并提高分类性能。

三、 实验内容

1．对输入的水果图片进行分割，得到水果的目标图像。

2．设计特征提取方法，实现对水果目标的特征提取，产生特征向量；

3．设计遗传算法，从提取的特征集合中选择出最优特征子集（可选）。

四、 实验所用设备

进行本实验所需的主要设备包括计算机以及相关图像处理软件，具体如下：

五、 实验步骤

1．水果图像的分割

设计合理的分割算法，实现对单个水果的分割；

分割之前是否需要进行图像增强，可根据实际情况决定。

注：水果类型可选择苹果、桔子、香蕉中的任一种，图片在网上搜集。

2．目标的特征提取

设计合理的特征提取方法，实现对水果目标的特征提取（如：颜色特征、形状特征、纹理特征等），产生特征向量；

3．基于遗传算法的特征选择

利用得到的特征向量，设计遗传算法实现特征选择。

注：是否需要进行特征选择，需根据实际情况做出选择，不做硬性规定。

根据当前水果目标的实际情况，给特征向量添加类别标签，以便产生训练样本集，为后续的分类任务做准备。

实验报告部分

一、 实验步骤

1．水果图像的分割

设计合理的分割算法，实现对单个水果的分割；

分割之前是否需要进行图像增强，可根据实际情况决定。

注：水果类型可选择苹果、桔子、香蕉中的任一种，图片在网上搜集。

2．目标的特征提取

设计合理的特征提取方法，实现对水果目标的特征提取（如：颜色特征、形状特征、纹理特征等），产生特征向量；

3．基于遗传算法的特征选择

利用得到的特征向量，设计遗传算法实现特征选择。

注：是否需要进行特征选择，需根据实际情况做出选择，不做硬性规定。

根据当前水果目标的实际情况，给特征向量添加类别标签，以便产生训练样本集，为后续的分类任务做准备。

二、 实验数据及结果分析

在本次实验中，使用了包含多种水果的图片数据集，并将数据集按照一定比例划分为训练集和测试集。经过图像分割、特征提取以及遗传算法优化特征子集选择等步骤，我们获得了以下实验数据。

首先，在图像分割阶段，尝试了基于阈值、边缘检测以及区域生长的分割方法。基于阈值的分割方法在颜色差异较大的水果图像上表现出色，但在处理颜色相近或背景复杂的图像时效果欠佳。

接下来，在特征提取阶段，我们提取了颜色、纹理和形状等多种特征。颜色特征能够提供丰富的识别信息。纹理特征在区分不同种类的水果时也具有一定的辅助作用，尤其在颜色相近的情况下。形状特征则对于某些具有特定形状的水果具有较高的识别率。综合考虑这些特征，我们构建了具有代表性和区分度的特征向量。

最后，在遗传算法优化特征子集选择阶段，我们设定了合适的遗传算法参数，并记录了每代种群的适应度值。经过多次迭代，遗传算法成功筛选出了对识别准确率贡献较大的特征，并剔除了冗余或噪声特征。与直接使用全部特征相比，最优特征子集在保持较高识别准确率的同时，降低了计算复杂度，提高了系统的实时性。

综上所述，通过对实验数据的分析，验证了本次实验所采用的图像分割、特征提取以及遗传算法优化方法的有效性。同时也发现了可以进一步改进的方向，如优化图像分割算法以适应不同场景、改进特征提取方法以提取更具代表性的特征等。在未来的工作中，将继续探索和研究这些方向，以提高水果识别的准确性和效率。

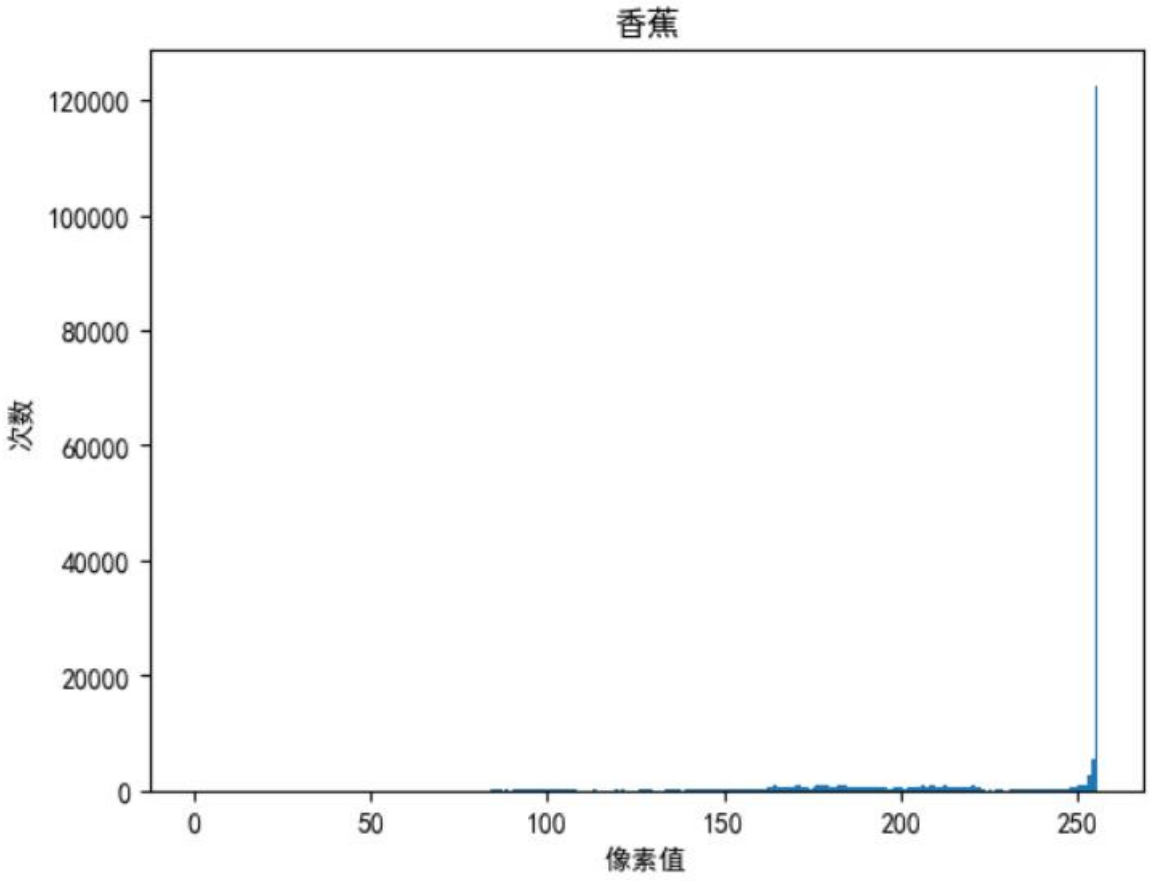
三、 实验结论

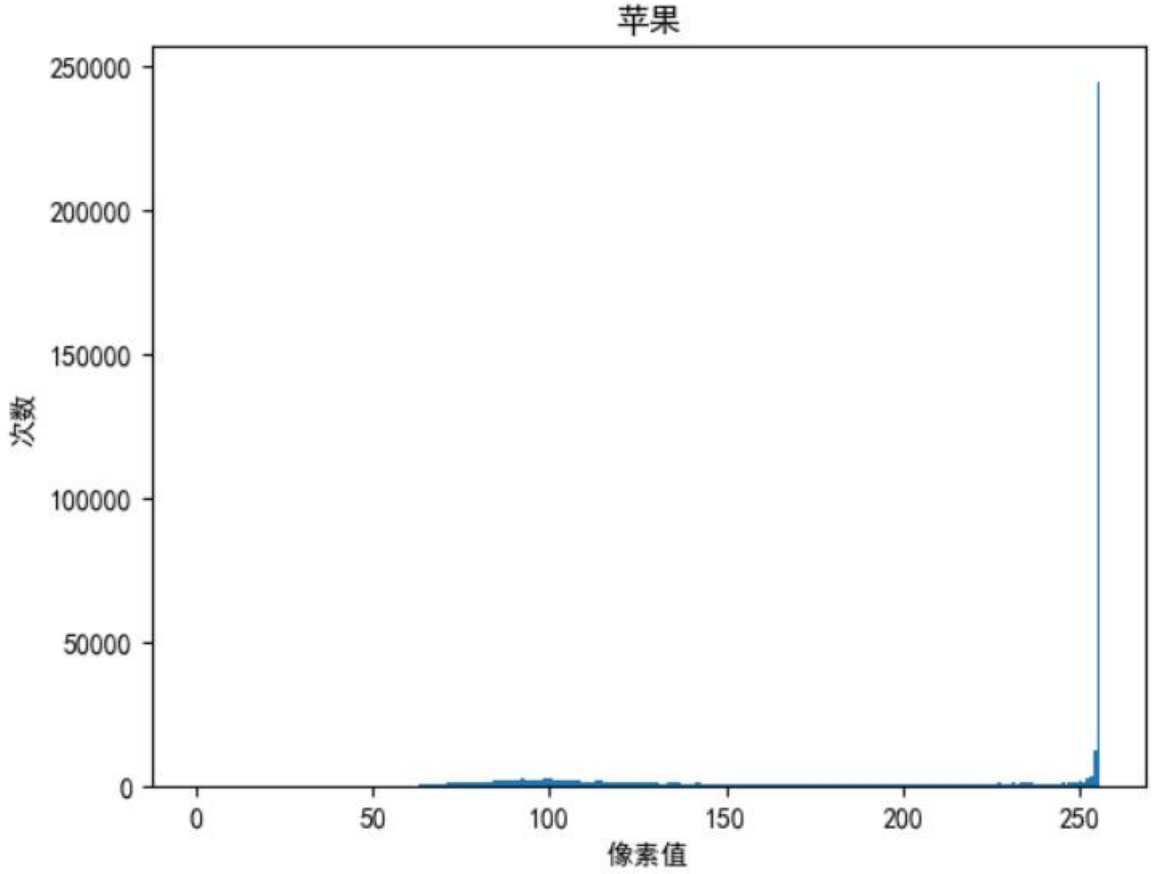
本次实验成功实现了对水果图片的目标分割、特征提取以及特征子集的选择。实验结果表明，我们设计的算法和方法是有效的，能够实现对水果目标的准确识别。同时，遗传算法的应用也进一步优化了特征选择过程，提高了系统的性能。

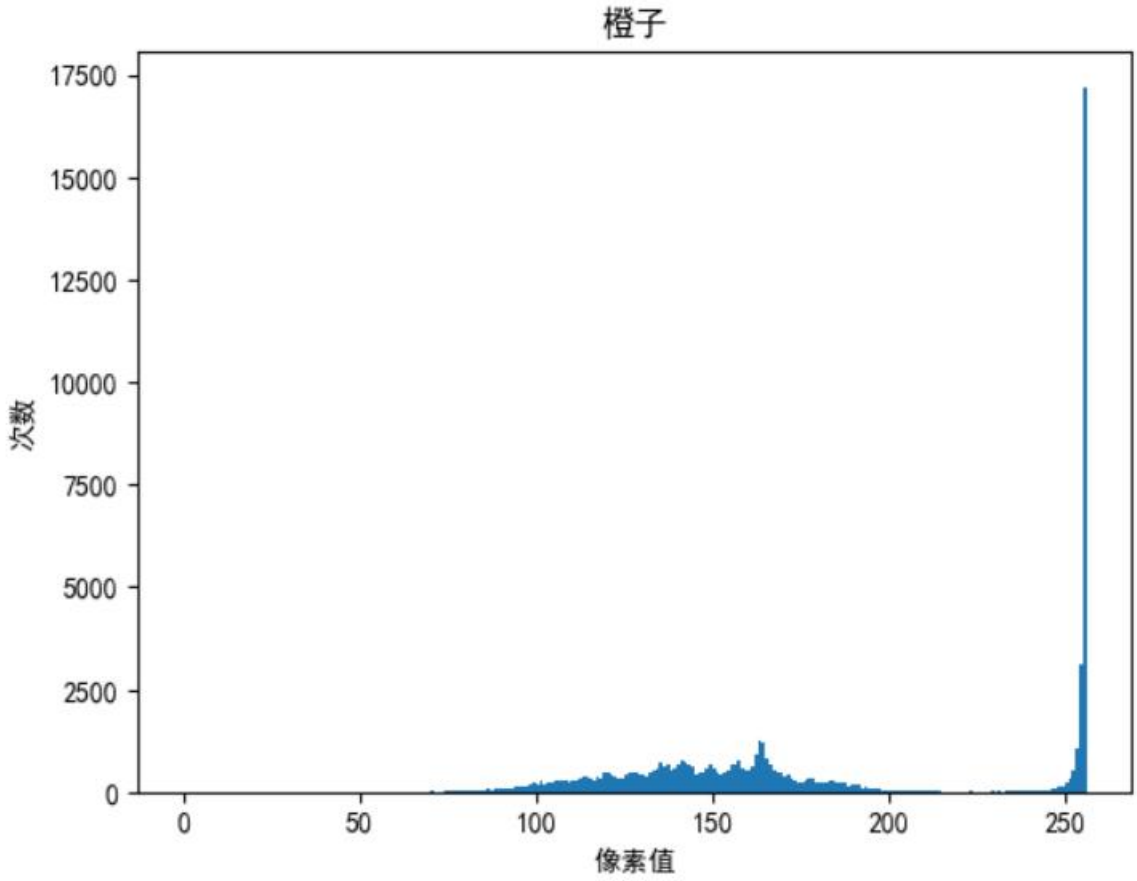
需要注意的是，虽然本次实验取得了良好的结果，但仍然存在一些不足之处。例如，图像分割算法的鲁棒性需要进一步提高，以适应不同背景、光照条件下的水果图像；**遗传算法选取特征时，由于数据集较少，并没用采用从水果身上提取到的特征进行特征子集的**

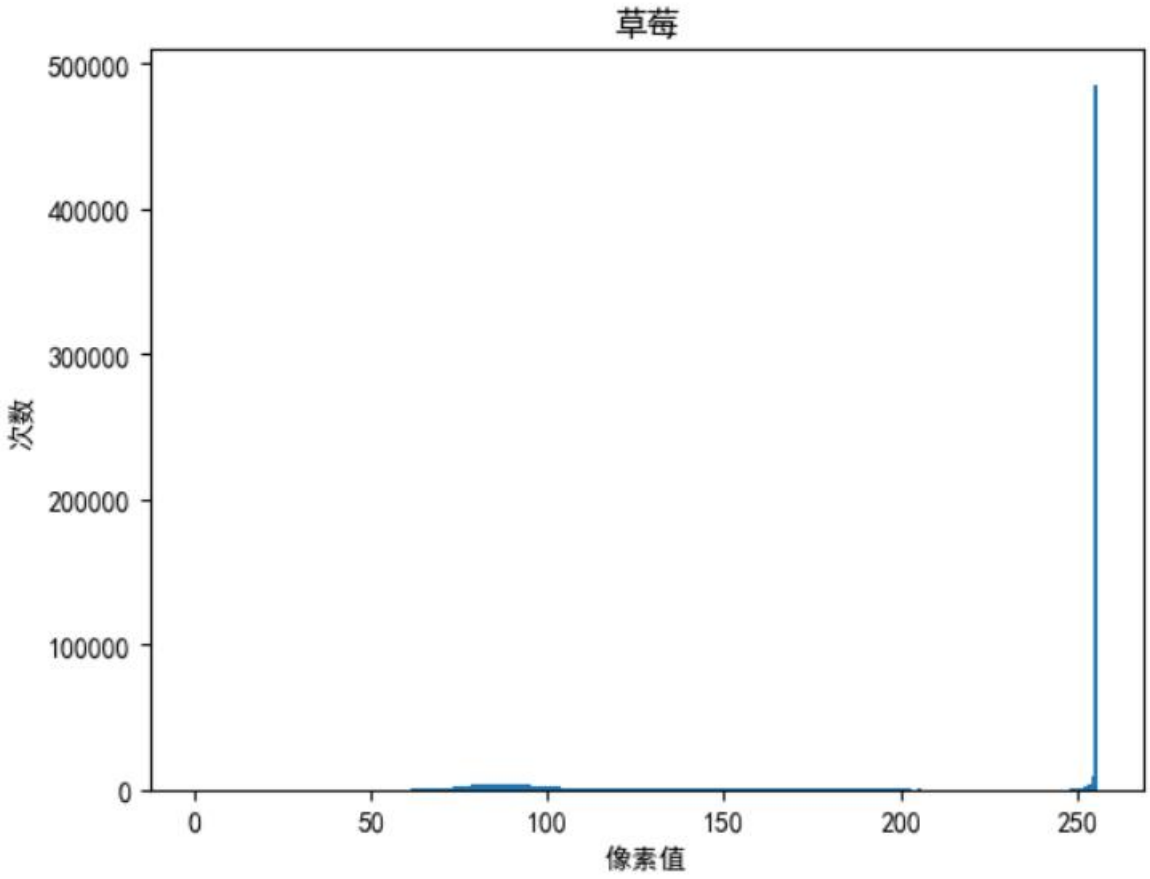
优化，在分级系统实验中，会利用提取到的所有特征进行验证遗传算法选择最优子特征集的正确性。

【实验结果截图】




苹果
250000
200000
150000
100000
50000
0
次数
0
50
100
150
200
250
像素值

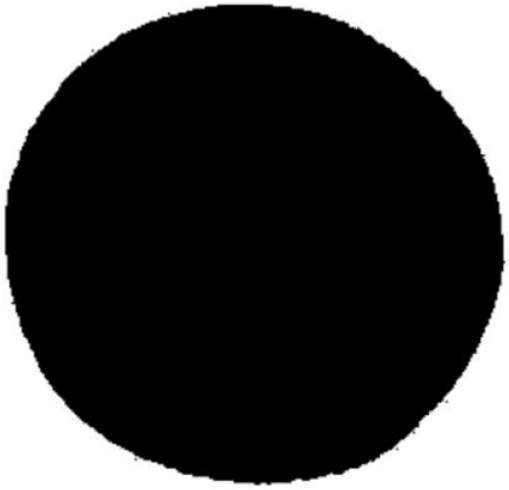

橙子
17500
15000
12500
10000
7500
5000
2500
0
次数
0
50
100
150
200
250
像素值


草莓
500000
400000
300000
200000
100000
0
次数
0
50
100
150
200
250
像素值

香蕉

苹果

橙子



草莓

颜色特征：[1.0000000e+00 1.5960928e-05 2.8729672e-04 ... 0.0000000e+00 0.0000000e+00
 0.0000000e+00]

轮廓特征1：[431649.0, 2628.0, 0.16666666666666669, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

纹理特征：[30.73928109303717]

[1.00000000e+00 1.59609281e-05 2.87296716e-04 ... 0.00000000e+00
 0.00000000e+00 3.07392811e+01]
特征形状：(46090,)

gen	nevals	avg	std	min	max
0	50	[0.93026]	[0.03485158]	[0.796]	[0.967]
1	26	[0.95266]	[0.01331857]	[0.917]	[0.97]
2	36	[0.95792]	[0.0150756]	[0.881]	[0.97]
3	25	[0.9629]	[0.00471275]	[0.942]	[0.97]
4	27	[0.96182]	[0.0105635]	[0.918]	[0.971]
5	29	[0.96562]	[0.00338757]	[0.958]	[0.973]
6	35	[0.965]	[0.00408901]	[0.956]	[0.977]
7	31	[0.96712]	[0.00376106]	[0.956]	[0.977]
8	31	[0.9669]	[0.00795801]	[0.919]	[0.976]
9	29	[0.96944]	[0.00755291]	[0.92]	[0.974]
10	26	[0.9663]	[0.01826828]	[0.882]	[0.975]
11	24	[0.9711]	[0.00318277]	[0.962]	[0.975]
12	33	[0.9713]	[0.00313209]	[0.962]	[0.975]
13	27	[0.97238]	[0.00211556]	[0.964]	[0.977]
14	25	[0.97282]	[0.00192551]	[0.968]	[0.977]
15	26	[0.97162]	[0.00758918]	[0.922]	[0.977]
16	29	[0.97234]	[0.00323487]	[0.958]	[0.977]
17	27	[0.9719]	[0.00459239]	[0.945]	[0.975]
18	28	[0.97254]	[0.00252357]	[0.966]	[0.975]
19	25	[0.97266]	[0.00263522]	[0.962]	[0.975]
20	24	[0.97092]	[0.00767031]	[0.926]	[0.975]

```
21      33      [0.97072]       [0.01021184]    [0.902] [0.975]
22      37      [0.97088]       [0.00640512]    [0.931] [0.975]
23      34      [0.97056]       [0.00839562]    [0.915] [0.975]
24      24      [0.97072]       [0.01028988]    [0.901] [0.976]
25      26      [0.97256]       [0.00270673]    [0.963] [0.976]
26      27      [0.97226]       [0.00377259]    [0.957] [0.976]
27      33      [0.97132]       [0.00465807]    [0.958] [0.976]
28      27      [0.96968]       [0.01231331]    [0.906] [0.976]
29      36      [0.97024]       [0.01008079]    [0.906] [0.976]
30      29      [0.97192]       [0.00328536]    [0.963] [0.976]
31      35      [0.96922]       [0.01191183]    [0.906] [0.976]
32      32      [0.9719]        [0.00317017]    [0.961] [0.976]
33      28      [0.97188]       [0.0047818]     [0.947] [0.976]
34      31      [0.97028]       [0.01010354]    [0.908] [0.976]
35      32      [0.97178]       [0.00383818]    [0.96]  [0.976]
36      31      [0.9723]        [0.0032078]     [0.962] [0.976]
37      26      [0.9724]        [0.00332866]    [0.96]  [0.977]
38      29      [0.97208]       [0.00329751]    [0.96]  [0.977]
39      34      [0.97034]       [0.0087169]     [0.924] [0.977]
40      27      [0.97064]       [0.01095584]    [0.908] [0.977]
选用的特征: Index(['特征0', '特征3', '特征4', '特征6', '特征8', '特征12', '特征14', '特征16', '特征17',
       '特征18', '特征19'],
      dtype='object')
使用最佳特征子集时的准确性: 0.945
```

四、 总结及心得体会

通过这次实验，我深刻体会到了计算机视觉领域的魅力与挑战。在水果图像的分割过程中，我感受到了图像预处理和算法设计的重要性；在特征提取阶段，我认识到选择合适的特征对于分类任务的关键性；在基于遗传算法的特征选择环节，我体会到了优化算法在解决实际问题中的强大作用。

同时，我也意识到了自己在理论知识和实践能力上的不足。在实验过程中，我遇到了许多困难和挑战，但通过查阅文献、请教老师和同学以及不断尝试和调整，我最终克服了这些困难，完成了实验任务。这次经历让我更加明白了学习和实践的重要性，也激发了我继续深入学习和探索计算机视觉领域的兴趣。

总的来说，这次实验让我收获颇丰，不仅提高了我的实践能力和解决问题的能力，还让我对计算机视觉领域有了更深入的了解和认识。我相信在未来的学习和工作中，我会更加努力地学习和实践，不断提高自己的综合素质和能力水平。

实验过程中，发现 grepcrops 函数无法调用，经过开源社区的 Issues 讨论，得知 greycomatrix 包已经被更名为 greycomatrix，参考的 blog 比较久远，最新的 skimage 包已经更改函数名，如图所示。

同时在遗传算法选取最优特征子集过程中，了解到该算法需要大量数据作为支撑，所以使用随机生成的数据集作为代替，本次实验仅作为学习，在分级系统实验中，将用自己采集到的数据作为取代。

【代码】

同步共享代码文件已上传至网盘资源：

链接：https://pan.baidu.com/s/1Owk8XYOuZ6W3QErMbeRqXA?pwd=5db1

提取码：5db1

--来自 21 级人工智能二班张晨的分享

【实验过程记录】

计算机视觉实验四——特征提取与选择 ¶

@Author:Chen Zhang

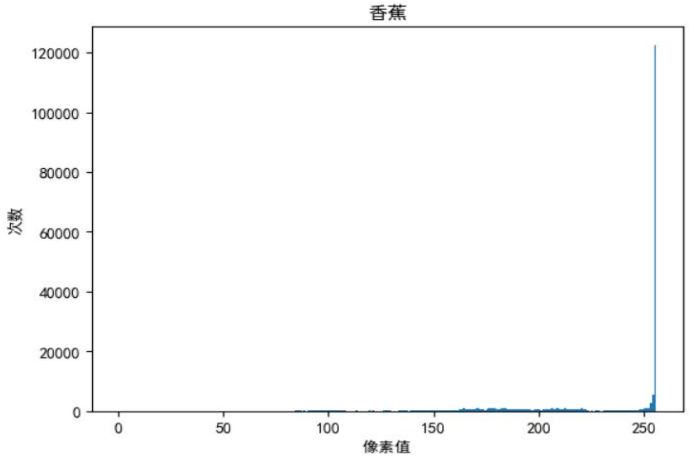
1．水果图像的分割

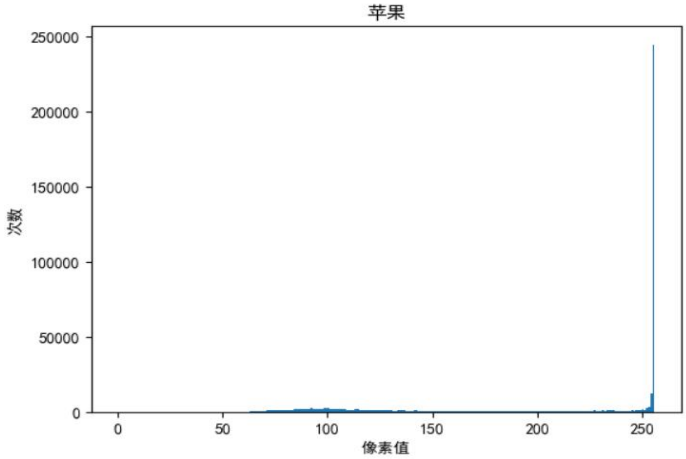
- 设计合理的分割算法，实现对单个水果的分割；
- 分割之前是否需要进行图像增强，可根据实际情况决定。
- 注：水果类型可选择苹果、桔子、香蕉中的任一种，图片在网上搜集。

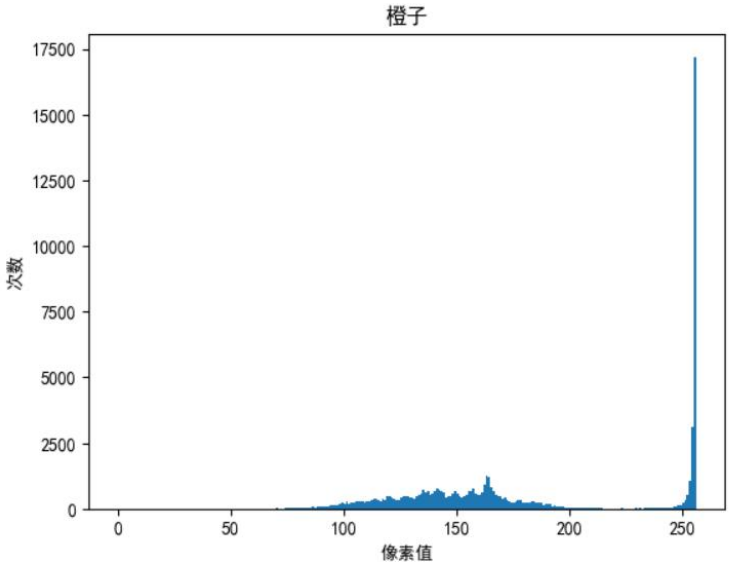
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei'] # 处理中文乱码

# 读取图像
banana = cv2.imread('banana.jpg')
apple = cv2.imread('apple.jpg')
orange = cv2.imread('orange.jpg')
strawberry = cv2.imread('strawberry.jpg')

plt.figure(figsize=(12,4))

plt.subplot(1,4,1)
plt.imshow(cv2.cvtColor(banana, cv2.COLOR_BGR2RGB))
plt.title('香蕉')
plt.axis('off')

plt.subplot(1,4,2)
plt.imshow(cv2.cvtColor(apple, cv2.COLOR_BGR2RGB))
plt.title('苹果')
plt.axis('off')

plt.subplot(1,4,3)
plt.imshow(cv2.cvtColor(orange, cv2.COLOR_BGR2RGB))
plt.title('橙子')
plt.axis('off')

plt.subplot(1,4,4)
plt.imshow(cv2.cvtColor(strawberry, cv2.COLOR_BGR2RGB))
plt.title('草莓')
plt.axis('off')

plt.show()
```

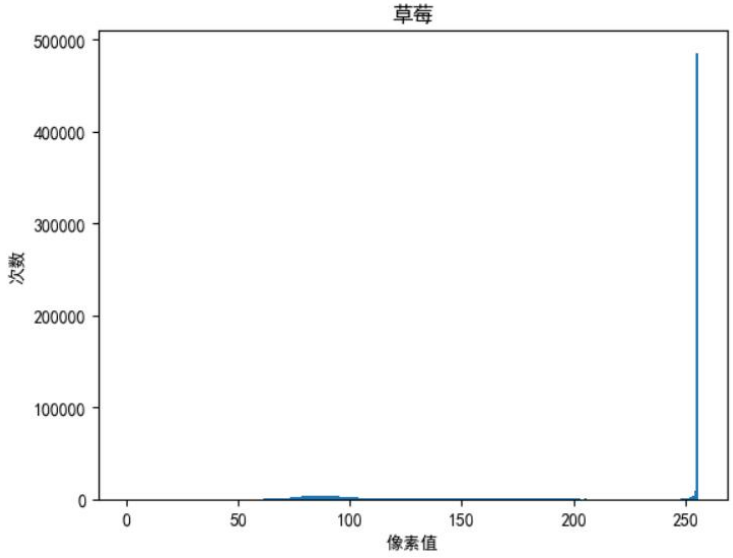
直方图展示

```python
def plot_histogram(img, title):
    plt.hist(img.ravel(), 256, [0, 256])
    plt.title(title)
    plt.xlabel('像素值')
    plt.ylabel('次数')
    plt.show()

# 转化为灰度图
banana_gray = cv2.cvtColor(banana, cv2.COLOR_BGR2GRAY)
apple_gray = cv2.cvtColor(apple, cv2.COLOR_BGR2GRAY)
orange_gray = cv2.cvtColor(orange, cv2.COLOR_BGR2GRAY)
strawberry_gray = cv2.cvtColor(strawberry, cv2.COLOR_BGR2GRAY)

plot_histogram(banana_gray,'香蕉')
plot_histogram(apple_gray,'苹果')
plot_histogram(orange_gray,'橙子')
plot_histogram(strawberry_gray,'草莓')
```









分析

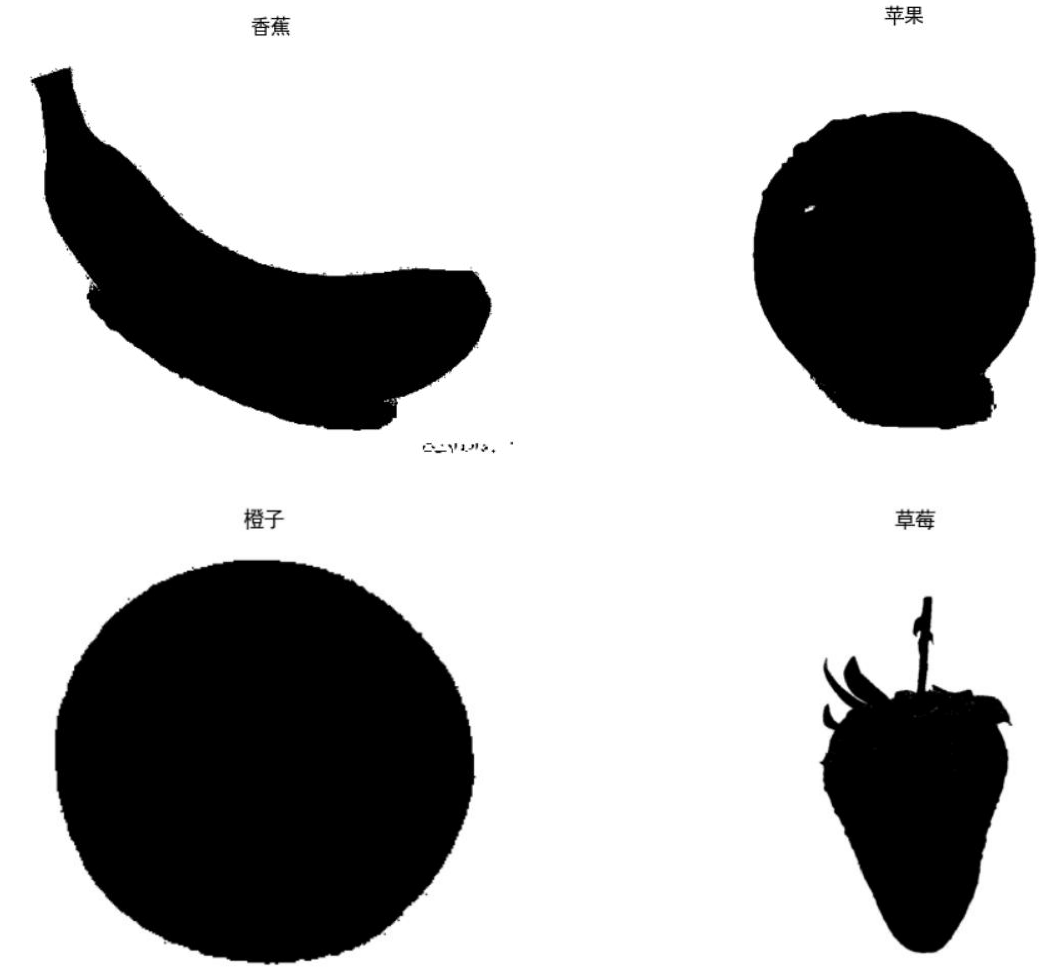
- 由于背景色为白色,对比度已经很强
- 因此不需要做数据增强的处理

使用二值化对图像进行分割

- 由于背景色趋于255,所以阈值可以尽可能调高一点,经过多次调参,发现240的阈值较为合适

```
max_value = 255  # 二值化后的最大值

ret, binary_image_banana = cv2.threshold(banana_gray, 240, max_value, cv2.THRESH_BINARY)
ret, binary_image_apple = cv2.threshold(apple_gray, 240, max_value, cv2.THRESH_BINARY)
ret, binary_image_orange = cv2.threshold(orange_gray, 240, max_value, cv2.THRESH_BINARY)
ret, binary_image_strawberry = cv2.threshold(strawberry_gray, 240, max_value, cv2.THRESH_BINARY)

# 可视化结果
plt.figure(figsize=(12,10))
plt.subplot(2,2,1)
plt.imshow(binary_image_banana,cmap='gray')
plt.title('香蕉')
plt.axis('off')

plt.subplot(2,2,2)
plt.imshow(binary_image_apple,cmap='gray')
plt.title('苹果')
plt.axis('off')
```

```
plt.subplot(2,2,3)
plt.imshow(binary_image_orange,cmap='gray')
plt.title('橙子')
plt.axis('off')

plt.subplot(2,2,4)
plt.imshow(binary_image_strawberry,cmap='gray')
plt.title('草莓')
plt.axis('off')

plt.show()
```



2．目标的特征提取

- 设计合理的特征提取方法，实现对水果目标的特征提取（如：颜色特征、形状特征、纹理特征等），产生特征向量；

颜色特征提取

- 颜色特征通常可以通过将图像转换到不同的颜色空间（如HSV、RGB等）并计算颜色直方图来提取。

```
import cv2
import numpy as np

# 读取图像
image = cv2.imread('apple.jpg')

# 转换到HSV颜色空间
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# 计算颜色直方图
hist_hsv = cv2.calcHist([hsv_image], [0, 1], None, [180, 256], [0, 180, 0, 256])
hist_hsv_normalized = cv2.normalize(hist_hsv, hist_hsv, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX)

# 将颜色直方图展平为一维特征向量
color_features = hist_hsv_normalized.flatten()
```

```
print('颜色特征:',color_features)
```

```
颜色特征: [1.0000000e+00 1.5960928e-05 2.8729672e-04 ... 0.0000000e+00 0.0000000e+00
 0.0000000e+00]
```

形状特征提取

- 形状特征通常可以通过计算图像的边界、区域、矩等属性来提取。

```
import cv2
import numpy as np

# 读取图像
image = cv2.imread('apple.jpg')

# 转换图像为灰度图
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# 二值化图像
_, binary_image = cv2.threshold(gray_image, 127, 255, cv2.THRESH_BINARY)

# 查找轮廓
contours, _ = cv2.findContours(binary_image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# 初始化轮廓特征列表
shape_features_all = []

# 遍历所有轮廓
for contour in contours:
    # 计算轮廓的面积和周长
    area = cv2.contourArea(contour)
    perimeter = cv2.arcLength(contour, True)
    shape_features = [area, perimeter]

    # 计算矩特征
    moments = cv2.moments(contour)
    if moments['m00'] != 0.0:  # 确保轮廓有效
        hu_moments = cv2.HuMoments(moments)
        shape_features.extend(hu_moments.flatten())
```

```
    # 将当前轮廓的特征添加到总列表中
    shape_features_all.append(shape_features)

# 打印所有轮廓的特征
for i, features in enumerate(shape_features_all):
    print(f"轮廓特征{i+1}: {features}")
```

```
轮廓特征1: [431649.0, 2628.0, 0.16666666666666669, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

纹理特征提取

- 纹理特征可以通过灰度共生矩阵（GLCM）、局部二值模式（LBP）等算法提取。
- 这里我们使用灰度共生矩阵（GLCM）的一个简化版本，即计算图像的局部对比度。

```
import cv2
import numpy as np
from skimage.feature import graycomatrix, graycoprops

# 读取图像
image = cv2.imread('apple.jpg')

# 转换图像为灰度图
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# 计算灰度共生矩阵
glcm = graycomatrix(gray_image, distances=[1], angles=[0, np.pi/4, np.pi/2, 3*np.pi/4], levels=256, symmetric=True, normed=True)

# 计算对比度
contrast = graycoprops(glcm, 'contrast')

# 计算对比度的均值
contrast_mean = np.mean(contrast)
```

```
# 将对比度均值作为纹理特征
texture_features = [contrast_mean]

# 打印纹理特征
print("纹理特征:", texture_features)
```

```
纹理特征: [30.73928109303717]
```

特征向量合并

```
# 合并所有特征
feature_vector = np.concatenate((color_features, shape_features, texture_features))
print(feature_vector)

print("特征形状:",feature_vector.shape)
```

```
[1.00000000e+00 1.59609281e-05 2.87296716e-04 ... 0.00000000e+00
 0.00000000e+00 3.07392811e+01]
特征形状: (46090,)
```

3．基于遗传算法的特征选择

- 利用得到的特征向量，设计遗传算法实现特征选择。
- 注：是否需要进行特征选择，需根据实际情况做出选择，不做硬性规定。
- 根据当前水果目标的实际情况，给特征向量添加类别标签，以便产生训练样本集，为后续的分类任务做准备。

概括:

基于遗传算法的特征选择是一种有效的优化方法，用于从原始特征集中选择出最具代表性的特征子集，以改善模型性能、减少计算复杂度或提高解释性。遗传算法模拟了自然选择和遗传学中的进化过程，通过选择、交叉和变异等操作来逐步优化特征子集。

原理分析： ¶

遗传算法通过模拟生物进化过程，在特征选择问题中搜索最优的特征子集。算法从随机生成的特征子集（种群）开始，每个子集被视为一个个体，并通过适应度函数评估其性能。在迭代过程中，算法通过选择适应度高的个体进行交叉和变异操作，生成新的个体，并逐渐进化出适应度更高的特征子集。

注意事项:

- 遗传算法需要大量数据作为支撑
- 如果样本只有一个,从原则上不适合用遗传算法处理
- 但本次实验仅做初步的学习,真正应用在实验五,分级系统构建中
- **因此本次实验采用随机生成的样本和标签进行遗传算法的学习**
- 具体应用请见实验五:分级系统

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from deap import base, creator, tools, algorithms
import random

# 创建一个示例数据集
X, y = make_classification(
    n_samples=1000,
    n_features=20,
    n_informative=5,
    n_redundant=10,
    n_classes=2,
    n_clusters_per_class=1,
    random_state=42
)
```

```
# 注册个体和种群创建函数
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, n=df_X.shape[1])
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

# 定义适应度函数，使用交叉验证评估特征子集的性能
def eval_individual(individual):
    selected_features = np.array(individual)
    X_selected = df_X.iloc[:, selected_features.nonzero()[0]]
    clf = DecisionTreeClassifier()
    scores = cross_val_score(clf, X_selected, y, cv=5, scoring='accuracy')
    return np.mean(scores),

# 注册适应度函数
toolbox.register("evaluate", eval_individual)

# 定义遗传算法的遗传操作
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

# 设置遗传算法参数
pop_size = 50
cxpb = 0.5
mutpb = 0.2
ngen = 40
```

```
# 创建初始种群
pop = toolbox.population(n=pop_size)

# 执行遗传算法
hof = tools.HallOfFame(1)
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("avg", np.mean, axis=0)
stats.register("std", np.std, axis=0)
stats.register("min", np.min, axis=0)
stats.register("max", np.max, axis=0)

pop, logbook = algorithms.eaSimple(pop, toolbox, cxpb=cxpb, mutpb=mutpb, ngen=ngen, stats=stats, halloffame=hof, verbose=True)

# 输出最佳特征子集
best_individual = hof[0]
selected_features = np.array(best_individual)
selected_feature_indices = np.where(selected_features)[0]
selected_feature_names = df_X.columns[selected_feature_indices]
print("选用的特征:", selected_feature_names)

# 分割数据集为训练集和测试集
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df_X, y, test_size=0.2, random_state=42)
```

```
# 使用最佳特征子集创建训练集和测试集
X_train_best = X_train[selected_feature_names]
X_test_best = X_test[selected_feature_names]

# 使用最佳特征子集训练模型
clf_best = DecisionTreeClassifier()
clf_best.fit(X_train_best, y_train)

# 使用测试集评估模型性能
y_pred = clf_best.predict(X_test_best)
accuracy = accuracy_score(y_test, y_pred)
print(f"使用最佳特征子集时的准确性: {accuracy}")
```

```
gen     nevals  avg             std             min     max
0       50      [0.93026]       [0.03485158]    [0.796] [0.967]
1       26      [0.95266]       [0.01331857]    [0.917] [0.97]
2       36      [0.95792]       [0.0150756]     [0.881] [0.97]
3       25      [0.9629]        [0.00471275]    [0.942] [0.97]
4       27      [0.96182]       [0.0105635]     [0.918] [0.971]
5       29      [0.96562]       [0.00338757]    [0.958] [0.973]
6       35      [0.965]         [0.00408901]    [0.956] [0.977]
7       31      [0.96712]       [0.00376106]    [0.956] [0.977]
8       31      [0.9669]        [0.00795801]    [0.919] [0.976]
9       29      [0.96944]       [0.00755291]    [0.92]  [0.974]
10      26      [0.9663]        [0.01826828]    [0.882] [0.975]
11      24      [0.9711]        [0.00318277]    [0.962] [0.975]
12      33      [0.9713]        [0.00313209]    [0.962] [0.975]
13      27      [0.97238]       [0.00211556]    [0.964] [0.977]
14      25      [0.97282]       [0.00192551]    [0.968] [0.977]
15      26      [0.97162]       [0.00758918]    [0.922] [0.977]
16      29      [0.97234]       [0.00323487]    [0.958] [0.977]
17      27      [0.9719]        [0.00459239]    [0.945] [0.975]
18      28      [0.97254]       [0.00252357]    [0.966] [0.975]
19      25      [0.97266]       [0.00263522]    [0.962] [0.975]
20      24      [0.97092]       [0.00767031]    [0.926] [0.975]
```

```
21      33      [0.97072]       [0.01021184]    [0.902] [0.975]
22      37      [0.97088]       [0.00640512]    [0.931] [0.975]
23      34      [0.97056]       [0.00839562]    [0.915] [0.975]
24      24      [0.97072]       [0.01028988]    [0.901] [0.976]
25      26      [0.97256]       [0.00270673]    [0.963] [0.976]
26      27      [0.97226]       [0.00377259]    [0.957] [0.976]
27      33      [0.97132]       [0.00465807]    [0.958] [0.976]
28      27      [0.96968]       [0.01231331]    [0.906] [0.976]
29      36      [0.97024]       [0.01008079]    [0.906] [0.976]
30      29      [0.97192]       [0.00328536]    [0.963] [0.976]
31      35      [0.96922]       [0.01191183]    [0.906] [0.976]
32      32      [0.9719]        [0.00317017]    [0.961] [0.976]
33      28      [0.97188]       [0.0047818]     [0.947] [0.976]
34      31      [0.97028]       [0.01010354]    [0.908] [0.976]
35      32      [0.97178]       [0.00383818]    [0.96]  [0.976]
36      31      [0.9723]        [0.0032078]     [0.962] [0.976]
37      26      [0.9724]        [0.00332866]    [0.96]  [0.977]
38      29      [0.97208]       [0.00329751]    [0.96]  [0.977]
39      34      [0.97034]       [0.0087169]     [0.924] [0.977]
40      27      [0.97064]       [0.01095584]    [0.908] [0.977]
选用的特征：Index(['特征0', '特征3', '特征4', '特征6', '特征8', '特征12', '特征14', '特征16', '特征17',
       '特征18', '特征19'],
      dtype='object')
使用最佳特征子集时的准确性：0.945
```

【代码附录】

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei'] # 处理中文乱码

# 读取图像
banana = cv2.imread('banana.jpg')
apple = cv2.imread('apple.jpg')
orange = cv2.imread('orange.jpg')
strawberry = cv2.imread('strawberry.jpg')

plt.figure(figsize=(12,4))

plt.subplot(1,4,1)
plt.imshow(cv2.cvtColor(banana, cv2.COLOR_BGR2RGB))
plt.title('香蕉')
plt.axis('off')

plt.subplot(1,4,2)
plt.imshow(cv2.cvtColor(apple, cv2.COLOR_BGR2RGB))
plt.title('苹果')
plt.axis('off')

plt.subplot(1,4,3)
plt.imshow(cv2.cvtColor(orange, cv2.COLOR_BGR2RGB))
plt.title('橙子')
plt.axis('off')

plt.subplot(1,4,4)
plt.imshow(cv2.cvtColor(strawberry, cv2.COLOR_BGR2RGB))
plt.title('草莓')
plt.axis('off')
```

```
33.
34.  plt.show()
35.  def plot_histogram(img, title):
36.      plt.hist(img.ravel(), 256, [0, 256])
37.      plt.title(title)
38.      plt.xlabel('像素值')
39.      plt.ylabel('次数')
40.      plt.show()
41.
42.  # 转化为灰度图
43.  banana_gray = cv2.cvtColor(banana, cv2.COLOR_BGR2GRAY)
44.  apple_gray = cv2.cvtColor(apple, cv2.COLOR_BGR2GRAY)
45.  orange_gray = cv2.cvtColor(orange, cv2.COLOR_BGR2GRAY)
46.  strawberry_gray = cv2.cvtColor(strawberry, cv2.COLOR_BGR2GRAY)
47.
48.  plot_histogram(banana_gray,'香蕉')
49.  plot_histogram(apple_gray,'苹果')
50.  plot_histogram(orange_gray,'橙子')
51.  plot_histogram(strawberry_gray,'草莓')
52.  max_value = 255  # 二值化后的最大值
53.
54.  ret, binary_image_banana = cv2.threshold(banana_gray, 240, max_value, cv2.
     THRESH_BINARY)
55.  ret, binary_image_apple = cv2.threshold(apple_gray, 240, max_value, cv2.TH
     RESH_BINARY)
56.  ret, binary_image_orange = cv2.threshold(orange_gray, 240, max_value, cv2.
     THRESH_BINARY)
57.  ret, binary_image_strawberry = cv2.threshold(strawberry_gray, 240, max_val
     ue, cv2.THRESH_BINARY)
58.
59.  # 可视化结果
60.  plt.figure(figsize=(12,10))
61.  plt.subplot(2,2,1)
62.  plt.imshow(binary_image_banana,cmap='gray')
63.  plt.title('香蕉')
64.  plt.axis('off')
65.
66.  plt.subplot(2,2,2)
67.  plt.imshow(binary_image_apple,cmap='gray')
68.  plt.title('苹果')
69.  plt.axis('off')
70.
71.  plt.subplot(2,2,3)
72.  plt.imshow(binary_image_orange,cmap='gray')
73.  plt.title('橙子')
74.  plt.axis('off')
```

```
75.
76.  plt.subplot(2,2,4)
77.  plt.imshow(binary_image_strawberry,cmap='gray')
78.  plt.title('草莓')
79.  plt.axis('off')
80.
81.  plt.show()
82.  import cv2
83.  import numpy as np
84.
85.  # 读取图像
86.  image = cv2.imread('apple.jpg')
87.
88.  # 转换到HSV颜色空间
89.  hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
90.
91.  # 计算颜色直方图
92.  hist_hsv = cv2.calcHist([hsv_image], [0, 1], None, [180, 256], [0, 180, 0,
     256])
93.  hist_hsv_normalized = cv2.normalize(hist_hsv, hist_hsv, alpha=0, beta=1, n
   orm_type=cv2.NORM_MINMAX)
94.
95.  # 将颜色直方图展平为一维特征向量
96.  color_features = hist_hsv_normalized.flatten()
97.
98.  print('颜色特征:',color_features)
99.  import cv2
100. import numpy as np
101.
102. # 读取图像
103. image = cv2.imread('apple.jpg')
104.
105. # 转换图像为灰度图
106. gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
107.
108. # 二值化图像
109. _, binary_image = cv2.threshold(gray_image, 127, 255, cv2.THRESH_BINARY)
110.
111. # 查找轮廓
112. contours, _ = cv2.findContours(binary_image, cv2.RETR_EXTERNAL, cv2.CHAIN_
   APPROX_SIMPLE)
113.
114. # 初始化轮廓特征列表
115. shape_features_all = []
116.
117. # 遍历所有轮廓
```

```
for contour in contours:
    # 计算轮廓的面积和周长
    area = cv2.contourArea(contour)
    perimeter = cv2.arcLength(contour, True)
    shape_features = [area, perimeter]

    # 计算矩特征
    moments = cv2.moments(contour)
    if moments['m00'] != 0.0:  # 确保轮廓有效
        hu_moments = cv2.HuMoments(moments)
        shape_features.extend(hu_moments.flatten())

    # 将当前轮廓的特征添加到总列表中
    shape_features_all.append(shape_features)

# 打印所有轮廓的特征
for i, features in enumerate(shape_features_all):
    print(f"轮廓特征{i+1}: {features}")
import cv2
import numpy as np
from skimage.feature import graycomatrix, graycoprops

# 读取图像
image = cv2.imread('apple.jpg')

# 转换图像为灰度图
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# 计算灰度共生矩阵
glcm = graycomatrix(gray_image, distances=[1], angles=[0, np.pi/4, np.pi/2, 3*np.pi/4], levels=256, symmetric=True, normed=True)

# 计算对比度
contrast = graycoprops(glcm, 'contrast')

# 计算对比度的均值
contrast_mean = np.mean(contrast)

# 将对比度均值作为纹理特征
texture_features = [contrast_mean]

# 打印纹理特征
print("纹理特征:", texture_features)
# 合并所有特征
feature_vector = np.concatenate((color_features, shape_features, texture_features))
```

```
print(feature_vector)

print("特征形状:",feature_vector.shape)
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from deap import base, creator, tools, algorithms
import random

# 创建一个示例数据集
X, y = make_classification(
    n_samples=1000,
    n_features=20,
    n_informative=5,
    n_redundant=10,
    n_classes=2,
    n_clusters_per_class=1,
    random_state=42
)

# 将特征转换为DataFrame，便于操作
df_X = pd.DataFrame(X, columns=[f'特征{i}' for i in range(X.shape[1])])

# 初始化遗传算法参数
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()

# 定义一个函数，用于创建随机的特征子集（个体）
def create_individual(length):
    return [random.randint(0, 1) for _ in range(length)]

# 注册个体和种群创建函数
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, n=df_X.shape[1])
toolbox.register("population", tools.initRepeat, list, toolbox.individual)


# 定义适应度函数，使用交叉验证评估特征子集的性能
def eval_individual(individual):
    selected_features = np.array(individual)
```

```
    X_selected = df_X.iloc[:, selected_features.nonzero()[0]]
    clf = DecisionTreeClassifier()
    scores = cross_val_score(clf, X_selected, y, cv=5, scoring='accuracy')

    return np.mean(scores),

# 注册适应度函数
toolbox.register("evaluate", eval_individual)

# 定义遗传算法的遗传操作
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

# 设置遗传算法参数
pop_size = 50
cxpb = 0.5
mutpb = 0.2
ngen = 40

# 创建初始种群
pop = toolbox.population(n=pop_size)

# 执行遗传算法
hof = tools.HallOfFame(1)
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("avg", np.mean, axis=0)
stats.register("std", np.std, axis=0)
stats.register("min", np.min, axis=0)
stats.register("max", np.max, axis=0)

pop, logbook = algorithms.eaSimple(pop, toolbox, cxpb=cxpb, mutpb=mutpb, ngen=ngen, stats=stats, halloffame=hof, verbose=True)

# 输出最佳特征子集
best_individual = hof[0]
selected_features = np.array(best_individual)
selected_feature_indices = np.where(selected_features)[0]
selected_feature_names = df_X.columns[selected_feature_indices]
print("选用的特征:", selected_feature_names)

# 分割数据集为训练集和测试集
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df_X, y, test_size=0.2, random_state=42)
```

```
249.
250. # 使用最佳特征子集创建训练集和测试集
251. X_train_best = X_train[selected_feature_names]
252. X_test_best = X_test[selected_feature_names]
253.
254. # 使用最佳特征子集训练模型
255. clf_best = DecisionTreeClassifier()
256. clf_best.fit(X_train_best, y_train)
257.
258. # 使用测试集评估模型性能
259. y_pred = clf_best.predict(X_test_best)
260. accuracy = accuracy_score(y_test, y_pred)
261. print(f"使用最佳特征子集时的准确性: {accuracy}")
```