重 庆 大 学

学 生 实 验 报 告

实验课程名称 人工智能导论

开课实验室 DS1502

学 院 软件学院 年级 2021 专业班 软工X班

学 生 姓 名 XXX 学 号 2021XXXX

开 课 时 间 2023 至 2024 学年第 1 学期

总 成 绩	
教师签名	

大数据与软件学院制

《人工智能导论》实验报告

开课实验室：DS1502 **2023 年 11 月 25 日**

学院	大数据与软件学院	年级、专业、班	21 软件工程 X 班	姓名	XXX	成绩	
课程名称	人工智能导论	实验项目名　称	基于遗传算法的图像二值化		指导教师	XX	
教师评语	教师签名： 2023 年　月　日						

一、 实验目的

本实验采用遗传算法和大津算法确定图像二值化的最佳阈值，从而对图像进行二值化分割。

二、实验内容

① 计算图像灰度直方图： 对待分割的图像进行预处理，计算其灰度直方图，用于后续遗传算法的适应度计算；

② 初始化种群： 随机生成M个个体，并对图像的灰度值进行编码，形成初始种群；

③ OTSU算法计算适应度值： 使用OTSU算法计算每个个体的适应度值，即将图像根据个体编码的阈值进行二值化，并评估二值化结果的质量；

④ 选择操作（自然选择）： 根据适应度值由大到小选择前M个个体，将它们复制到下一代种群中。同时，随机选择一定比例的较差个体，保留并复制到下一代，以维持种群的多样性；

⑤ 交叉操作（繁殖）： 随机选取父体和母体，进行交叉操作，模拟染色体的交叉。通过随机选择交叉点，生成新个体，并将其加入下一代种群，直到种群数量足够；

⑥ 变异操作（基因变异）： 对上一步产生的种群中的个体进行基因变异，即以一定概率P对个体的某个基因进行突变；

⑦ 迭代操作： 重复进行选择、交叉和变异操作，形成新一代种群。进行多次迭代，直到种群进化了一定数量的代数；

⑧ 选择最优个体： 从最终的种群中选择适应度最优的个体，将其转换成图像分割的阈值，然后应用该阈值对图像进行二值化分割；

⑨ 显示效果： 对于选定的最优阈值，处理图像并显示分割效果，评估算法在该图像上的性能。

三、使用仪器、材料

1. 操作系统：Windows 11
2. 开发设备：Lenovo Legion R9000P2021H
3. 开发平台：PyCharm 2023.1

四、实验过程原始记录(数据、图表、计算等)：

（一）源代码

```python
import numpy as np
from PIL import Image

1 个用法
class GeneticAlgorithm:
    def __init__(self, image, population_size):
        # 初始化遗传算法对象
        self.image = image  # 输入图像
        self.population_size = population_size  # 种群大小
        self.gene_length = 8  # 染色体基因长度
        self.chromosomes = np.random.randint(0, 256, self.population_size, dtype=np.uint8)  # 初始化种群的染色体
        self.selection_rate = 0.5  # 选择父代个体的概率
        self.strong_rate = 0.2  # 直接保留适应性强的染色体的比例
        self.mutation_rate = 0.05  # 染色体变异的概率

    2 用法
    def evaluate_fitness(self, threshold):
        # 评估染色体的适应度，调用OTSU算法
        return OTSU().otsu(self.image, threshold)

    1 个用法
    def select_parents(self):
        # 选择父代个体
        fitness = [(self.evaluate_fitness(chromosome), chromosome) for chromosome in self.chromosomes]
        sorted_fitness = sorted(fitness, reverse=True)
        parents = [sorted_fitness[i][1] for i in range(int(len(sorted_fitness) * self.strong_rate))]
        parents += [chromosome[1] for chromosome in sorted_fitness[int(len(sorted_fitness) * self.strong_rate):] if np.random.random() < self.selection_rate]
        return parents
```

```python
    def crossover(self, parents):
        # 个体交叉操作
        children = []
        child_count = len(self.chromosomes) - len(parents)
        while len(children) < child_count:
            father, mother = np.random.choice(parents, size=2, replace=False)
            position = np.random.randint(0, self.gene_length)
            mask = (1 << position) - 1
            child = (father & mask) | (mother & ~mask)
            children = np.append(children, child)
        # 更新种群，确保数据类型为 uint8
        self.chromosomes = np.append(parents, children).astype(np.uint8)

    1 个用法
    def mutate(self):
        # 个体变异操作
        for i in range(len(self.chromosomes)):
            if np.random.random() < self.mutation_rate:
                j = np.random.randint(0, self.gene_length)
                # 在染色体中随机翻转一个比特，确保数据类型为 uint8
                self.chromosomes[i] ^= (1 << j)

    1 个用法
    def evolve(self):
        # 进化操作，包括选择父代、交叉和变异
        parents = self.select_parents()
        self.crossover(parents)
        self.mutate()
```

```
    1 个用法
    def get_best_threshold(self):
        # 从最终种群中获取最佳阈值
        fitness = [(self.evaluate_fitness(chromosome), chromosome) for chromosome in self.chromosomes]
        sorted_fitness = sorted(fitness, reverse=True)
        return sorted_fitness[0][1]

1 个用法
class OTSU:
    1 个用法
    def otsu(self, image, threshold):
        # 实现OTSU算法进行适应度评估
        image = np.asarray(image).T
        size = image.shape[0] * image.shape[1]
        bin_image = image < threshold
        summ = np.sum(image)
        w0 = np.sum(bin_image)
        sum0 = np.sum(bin_image * image)
        w1 = size - w0
        if w1 == 0:
            return 0
        sum1 = summ - sum0
        mean0 = sum0 / w0
        mean1 = sum1 / w1
        fit = w0 / size * w1 / size * (mean0 - mean1) ** 2
        return fit
```

```
1 个用法
def apply_threshold_and_display(threshold, image):
    # 应用阈值并显示处理后的图像
    temp = np.asarray(image)
    print("灰度值矩阵为: ")
    print(temp)
    array = np.where(temp < threshold, 0, 255).reshape(-1)
    image.putdata(array)
    image.show()
    image.save('output.jpg')

1 个用法
def main():
    file_path = 'example.jpg'
    image = Image.open(file_path)
    image.show()

    gray_image = image.convert('L')
    ga = GeneticAlgorithm(gray_image, 16)

    for iteration in range(200):
        ga.evolve()
        if (iteration + 1) % 10 == 0:  # 每隔10次迭代输出一次
            print(f"迭代 {iteration + 1}: {ga.chromosomes}")

    best_threshold = ga.get_best_threshold()
    print("最佳阈值:", best_threshold)

    apply_threshold_and_display(best_threshold, gray_image)

if __name__ == "__main__":
    main()
```

（二）实现效果

1. 原图

example.jpg

2. 实现效果

output.jpg

3. 迭代过程

```
迭代 10: [173 173 173 173 173 173 172 143 140 236 173 173 137 172 237 173]
迭代 20: [173 173 173 173 173 173 172 165 164  45 173  47 172 172 173  45]
迭代 30: [171 175 175 175 173 173 173 172 175 173 175 173 175 173 175 173]
迭代 40: [175 175 175 175 173 173 173  47  45  45  47  47  45 175 173 175]
迭代 50: [175 175 175 175 175 175 175 173 173 167  37 191 175 175 173 175]
迭代 60: [175 175 175 175 175 175 175 175 175 175 239  47 175 175 175 175]
迭代 70: [175 175 175 175 175 175 175 143 175 171 175 175 175 175 171 175]
迭代 80: [175 175 175 175 175 175 175 175  47  47  47  47  47 175 175 175]
迭代 90: [191 175 175 175 171 173 175 191 175 175 175 175 175 175 175 175]
迭代 100: [175 175 175 175 174 175 175 143 175 175 175 239 143 175 175 175]
迭代 110: [175 175 175 175 175 175 167 175 167 167 175 175 175 175 175 175]
迭代 120: [175 175 175 175 175 175 167 143 175 175 175 175 175 174 175 175]
迭代 130: [175 175 175 175 175 175 173 173 191 175 175 175 173 175 175 175]
迭代 140: [175 175 175 175 175 191 175 175 173 175 175 175 239 175 175 175]
迭代 150: [175 175 175 175 175 175 175 173 173 173 173 175 175 173 175 175]
迭代 160: [175 175 175 175 175 175 175 175 171  47 175 175 175 175  47 175]
迭代 170: [175 175 175 175 175 175 143 175 175 175 175 143 175 191 175 175]
迭代 180: [175 175 175 175 175 175 175 175 175 175 175 175 175 175 175 175]
迭代 190: [175 175 175 175 175 175 171 167 239 175 239 167 175 239 171 175]
迭代 200: [175 175 173 175 175 175 175 175 175 175 175 167 175 175 175 175]
最佳阈值: 175
灰度值矩阵为:
[[175 174 173 ... 183 190 162]
 [197 196 195 ... 186 193 165]
 [219 218 217 ... 186 193 166]
 ...
 [124 131 149 ... 194 194 193]
 [126 139 156 ... 195 194 194]
 [127 146 163 ... 196 195 195]]
```

输出的最佳阈值为 175。

4. 总结

本实验旨在通过应用遗传算法实现图像阈值处理，利用自适应优化策略找到最佳阈值，以提升图像二值化效果。以下是主要实现的功能：

（1） 遗传算法类 (GeneticAlgorithm)：

① 初始化： 通过__init__方法，初始化遗传算法对象，包括输入图像、种群大小、染色体基因长度、染色体初始化、选择率、保留率、变异率等；

② 适应度评估 (evaluate_fitness)：利用 OTSU 算法对染色体进行适应度评估，评价染色体的图像二值化效果；

③ 选择父代 (select_parents)：根据适应度选择父代染色体，一部分来自适应性强的染色体，一部分以一定概率选择适应性较弱的染色体；

④ 交叉 (crossover)：通过交叉操作生成新一代染色体，增加遗传算法的多样性；

⑤ 变异 (mutate)：对染色体进行变异操作，引入随机性，提升遗传算法的探索能力；

⑥ 进化 (evolve)：包括选择父代、交叉和变异等操作，推动种群朝着更优秀的方向演化；

⑦ 获取最佳阈值 (get_best_threshold)：从最终种群中获取适应性最强的染色体作为最佳阈值。

（2） OTSU 类 (OTSU)：

OTSU 算法 (otsu)：根据输入图像和阈值，实现 OTSU 算法进行适应度评估，用于遗传算法中的染色体适应度评估。

（3） 应用阈值并显示 (apply_threshold_and_display):

应用阈值 (apply_threshold_and_display)：根据最佳阈值将图像进行二值化，并显示处理后的图像。

（4）主函数 (main)：

① 初始化图像和遗传算法:：打开图像文件，将其转换为灰度图，并初始化遗传算法对象；

② 遗传算法迭代：迭代遗传算法，每 10 次输出一次染色体信息，推动遗传算法不断优化图像二值化效果；

③ 获取最佳阈值：获取经过遗传算法优化后的最佳阈值；

④ 应用最佳阈值并显示：根据最佳阈值对图像进行处理，并显示处理后的图像。

通过这个遗传算法实验，成功利用自适应优化的策略，提高了图像的二值化效果，使得算法能够自动寻找最佳阈值，进一步提升了图像处理的质量。