基于遗传算法的图像分割

人工智能导论课

任课老师：孔万增

学号：150501648

姓名：朱宏林

简 述：

本实验采用遗传算法和大津阈值分割法确定图像分割的最佳阈值，从而对图像进行二值化分割。

1. 大津阈值分割法

在计算机视觉和图像处理中，大津二值化法用来自动对基于聚类的图像进行二值化

或者说，将一个灰度图像退化为二值图像。

算法假定该图像根据双模直方图（前景像素和背景像素）把包含两类像素，于是它要计算能将两类分开的最佳阈值，使得它们的类内方差最小；由于两两平方距离恒定，所以即它们的类间方差最大。因此，大津二值化法粗略的来说就是一维Fisher判别分析的离散化模拟。

1. 遗传算法

遗传算法（英语：genetic algorithm (GA) ）是计算数学中用于解决最佳化的搜索算法，是进化算法的一种。进化算法最初是借鉴了进化生物学中的一些现象而发展起来的，这些现象包括遗传、突变、自然选择以及杂交等。

遗传算法通常实现方式为一种计算机模拟。对于一个最优化问题，一定数量的候选解（称为个体）可抽象表示为染色体，使种群向更好的解进化。传统上，解用二进制表示（即0和1的串），但也可以用其他表示方法。进化从完全随机个体的种群开始，之后一代一代发生。在每一代中评价整个种群的适应度，从当前种群中随机地选择多个个体（基于它们的适应度），通过自然选择和突变产生新的生命种群，该种群在算法的下一次迭代中成为当前种群。

算法:

选择初始生命种群

循环

评价种群中的个体适应度

以比例原则（分数高的挑中机率也较高）选择产生下一个种群（轮盘法（roulette wheel selection）、竞争法（tournament selection）及等级轮盘法（Rank Based Wheel Selection））。不仅仅挑分数最高的的原因是这么做可能收敛到局部的最佳点，而非整体的。

改变该种群（交叉和变异）

直到停止循环的条件满足

最简单的遗传算法将染色体表示为一个数位串，数值变量也可以表示成整数，或者实数（浮点数）。算法中的杂交和突变都是在字节串上进行的，所以所谓的整数或者实数表示也一定要转化为数位形式。例如一个变量的形式是实数，其范围是0～1，而要求的精度是0.001，那么可以用10个数位表示：0000000000表示0，1111111111表示1。那么0110001110就代表0.398。

在遗传算法里，精英选择是一种非常成功的产生新个体的策略，它是把最好的若干个个体作为精英直接带入下一代个体中，而不经过任何改变。

通过并行计算实现遗传算法一般有两种，一种是所谓粗糙并行遗传算法，即一个计算单元包含一个种群；而另一种是所谓精细并行遗传算法，每一个计算单元处理一个染色体个体。

遗传算法有时候还引入其他变量，例如在实时优化问题中，可以在适应度函数中引入时间相关性和干扰。

1. 实验思路

**3.1 大致思路：**

[01]：计算待分割图像的图像灰度直方图；

[02]：对图像的灰度值进行编码，随机产生M个初始种群；

[03]：根据OTSU算法计算每个个体的适应度值；

[04]：建立种群，进行一定数量的遗传操作， 包括顺次执行的选择操作、 交叉操作和变异操作。

[05]：选择操作（自然选择）

将当代种群中的个体按照适应度值由大到小选择前M个个体，将他们复制到下一代种群中。同时使用随机的方式，保留一定比例落后的个体，让其存活，将他们复制到下一代种群中。

[06] ：交叉操作（繁殖）

随机选出父体和母体，进行交叉操作，类比染色体的交叉，随机选择交叉点。交叉完

成的新个体补充到下一代种群中，直到种群数量足够为止。

[07] ：变异操作（基因变异）

将上述交叉操作产生的种群中的个体的变异率P进行变化，如某一个体的某一个基因进行突变。

[08] ：进行多次迭代，直到种群进化了一定数量的代数以后，选择其最优先的个体。

[09] ：优秀个体转换成分割阈值，处理图像，显示效果。

* 1. **编码思考：**

如何建立灰度图和染色体的联系：

灰度图中，每个像素点的灰度值从0到256，二进制是8比特位，我将八个比特位看成一个染色体，每一个比特位就是一个基因点。

如何考虑选择下一代种群：

首先先把种群中的每一个个体计算适应度，借助OTSU算法，可以得出类内方差最小的值，通过此得出适应度，适应度的大小即可表示个体的优劣程度。但不能只选出色的个体，有些不好的个体可能会产生优质的下一代个体，因此在选择的时候，考虑首先选择前百分比a的优秀个体，剩下的个体中，随机选取比例b的个体幸存，其余淘汰。

如何考虑变异：

考虑到真实的环境中变异的可能性比较小，所以设定了一个变异的概率来控制变异的可能性，同时保证每个基因变异的可能性是一样的，这里也结合考虑过结合阈值来说的话，变异高位就可能导致相差太大，但最后的实验结果是没什么差别，反而对所有基因一视同仁思路更加简单。

1. 实验代码

---- main.py ----

# -\*- coding: utf-8 -\*-

from PIL import Image

import numpy as np

from genetic import GA

filename = 'images/mikaso.jpeg'

def threshold(t, image):

image\_tmp = np.asarray(image)

intensity\_array = list(np.where(image\_tmp<t, 0, 255).reshape(-1))

image.putdata(intensity\_array)

image.show()

image.save('images/output.png')

def main():

im = Image.open(filename)

im.load()

im.show()

im\_gray = im.convert('L') # translate to gray map

ga = GA(im\_gray)

for x in xrange(50):

ga.evolve()

best\_threshold = ga.result()

print best\_threshold

threshold(best\_threshold, im\_gray)

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

main()

---- genetic.py ----

# -\*- coding: utf-8 -\*-

import numpy as np

from otsu import otsu, fast\_ostu

np.random.seed(8)

class GA:

def \_\_init\_\_(self, image, N = 10):

"""

genetic algorithm

:param image: image feature

:param N: num of population

:param population: N population

"""

self.image = image

self.N = N

self.population = np.random.randint(0, 256, self.N)

self.retain\_rate = 0.2

self.random\_select\_rate = 0.5

self.mutation\_rate = 0.1

self.length = 8

def evolve(self):

parents = self.selection()

self.crossover(parents)

self.mutation(self.mutation\_rate)

def selection(self):

graded = [(self.fitness(chromosome), chromosome) for chromosome in self.population]

graded = [x[1] for x in sorted(graded, reverse=True)]

# 选出适应性强的染色体

retain\_length = int(len(graded) \* self.retain\_rate)

parents = graded[:retain\_length]

# 选出适应性不强，但是幸存的染色体

for chromosome in graded[retain\_length:]:

if np.random.random() < self.random\_select\_rate:

parents.append(chromosome)

return parents

def fitness(self, chromosome):

fitness = fast\_ostu(self.image, chromosome)

return fitness

def crossover(self, parents):

children = []

# 需要繁殖的孩子的量

target\_count = len(self.population) - len(parents)

while len(children) < target\_count:

male = np.random.randint(0, len(parents)-1)

female = np.random.randint(0, len(parents)-1)

if male != female:

# 随机选取交叉点

cross\_pos = np.random.randint(0, self.length)

# 生成掩码，方便位操作

mask = 0

for i in xrange(cross\_pos):

mask |= (1 << i)

male = parents[male]

female = parents[female]

# 孩子将获得父亲在交叉点前的基因和母亲在交叉点后（包括交叉点）的基因

child = ((male & mask) | (female & ~mask)) & ((1 << self.length) - 1)

children.append(child)

# 经过繁殖后，孩子和父母的数量与原始种群数量相等，在这里可以更新种群。

self.population = parents + children

def mutation(self, rate):

for i in xrange(len(self.population)):

if np.random.random() < rate:

j = np.random.randint(0, self.length - 1)

self.population[i] ^= 1 << j

def result(self):

graded = [(self.fitness(chromosome), chromosome) for chromosome in self.population]

graded = [x[1] for x in sorted(graded, reverse=True)]

return graded[0]

---- otsu.py ----

# -\*- coding: utf-8 -\*-

import numpy as np

def total\_pix(image):

size = image.shape[0] \* image.shape[1]

return size

def fast\_ostu(image, threshold):

image = np.transpose(np.asarray(image))

total = total\_pix(image)

bin\_image = image < threshold

sumT = np.sum(image)

w0 = np.sum(bin\_image)

sum0 = np.sum(bin\_image \* image)

w1 = total - w0

if w1 == 0:

return 0

sum1 = sumT - sum0

mean0 = sum0 / (w0 \* 1.0)

mean1 = sum1 / (w1 \* 1.0)

varBetween = w0 / (total \* 1.0) \* w1 / (total \* 1.0) \* (mean0 - mean1) \* (mean0 - mean1)

return varBetween

1. 实验结果

原图与结果图对比

