Image Segmentation Based on Genetic Algorithm

Introduction to Artificial Intelligence

Brief description:

In this experiment, the optimal threshold of image segmentation is determined by genetic algorithm and Otsu threshold segmentation method, so that the image is binarized.

1. Otsu threshold segmentation method

In computer vision and image processing, Otsu binarization is used to automatically binarize cluster-based images.

Or, a grayscale image is degraded into a binary image.

The algorithm assumes that the image contains two types of pixels according to the dual-mode histogram (foreground pixels and background pixels), so it calculates the optimal threshold that can separate the two classes, making their intra-class variance the smallest; since the two-squared distance is constant So they have the largest variance between classes. Therefore, the Otsu binarization method is roughly a discrete simulation of one-dimensional Fisher discriminant analysis.

2. Genetic algorithm

Genetic algorithm (GA) is a search algorithm used in computational mathematics to solve optimization. It is a kind of evolutionary algorithm. Evolutionary algorithms were originally developed with reference to some phenomena in evolutionary biology, including genetics, mutation, natural selection, and hybridization.

The general implementation of genetic algorithms is a computer simulation. For an optimization problem, a certain number of candidate solutions (called individuals) can be abstractly represented as chromosomes, allowing the population to evolve toward a better solution. Traditionally, binary representations (ie strings of 0 and 1) have been used, but other representations can be used. Evolution begins with a population of completely random individuals and then occurs from generation to generation. Evaluate the fitness of the entire population in each generation, randomly select multiple individuals from the current population (based on their fitness), and generate new life populations through natural selection and mutation, which becomes the next iteration of the algorithm. Current population.

algorithm:

Select initial life population

cycle

Evaluating individual fitness in a population

The next population (roulette wheel selection, tournament selection, and Rank Based Wheel Selection) is selected based on the principle of proportionality (higher probability of high scores). The reason not only to pick the highest score is that it may converge to the local optimum, not the whole.

Change the population (cross and variation)

Until the conditions for stopping the loop are met

The simplest genetic algorithm represents a chromosome as a digit string, and a numeric variable can also be represented as an integer, or a real number (floating point number). Hybridization and mutation in the algorithm are performed on a byte string, so the so-called integer or real representation must also be converted to a digital form. For example, the form of a variable is a real number, the range is 0 to 1, and the required precision is 0.001, then it can be represented by 10 digits: 0000000000 means 0, and 1111111111 means 1. Then 0110001110 represents 0.398.

In genetic algorithms, elite selection is a very successful strategy for generating new individuals. It is to bring the best individuals as elites directly into the next generation without any change.

There are generally two kinds of genetic algorithms implemented by parallel computing. One is the so-called rough parallel genetic algorithm, that is, one computing unit contains one population, and the other is a so-called fine parallel genetic algorithm, in which each computing unit processes one chromosome individual.

Genetic algorithms sometimes introduce other variables. For example, in real-time optimization problems, time correlation and interference can be introduced in the fitness function.

3. Experimental ideas

3.1 General idea:

[01]: calculating an image gray histogram of the image to be segmented;

[02]: encoding the gray value of the image, randomly generating M initial populations;

[03]: calculating the fitness value of each individual according to the OTSU algorithm;

[04]: Establish a population and perform a number of genetic operations, including sequential selection, crossover, and mutation operations.

[05]: Select operation (natural selection)

The individuals in the contemporary population are selected from the top M individuals according to the fitness value from large to small, and they are copied to the next generation population. At the same time, using a random approach, keep a certain percentage of individuals behind, let them survive, and copy them into the next generation of population.

[06] : Crossover operation (reproduction)

Randomly select the parent and the mother, perform cross operations, cross the analogous chromosomes, and randomly select the intersections. Cross over

The new individual is added to the next generation of the population until the population is sufficient.

[07] : Mutational Operations (Gene Variation)

The mutation rate P of the individual in the population resulting from the above crossover operation is changed, such as mutation of a certain gene of an individual.

[08] : Perform multiple iterations until the population has evolved a certain number of algebras and select its highest priority individual.

[09]: Excellent individuals are converted into segmentation thresholds, processed images, and displayed.

3.2 coding thinking:

How to establish a connection between grayscale and chromosomes:

In the grayscale image, the gray value of each pixel is from 0 to 256, and the binary is 8 bits. I regard eight bits as one chromosome, and each bit is a gene point.

How to consider the choice of next generation population:

First, calculate the fitness for each individual in the population. With the OTSU algorithm, the minimum variance within the class can be obtained. Through this, the fitness can be obtained. The degree of fitness can indicate the superiority and inferiority of the individual. But you can't just choose good individuals. Some bad individuals may produce good quality next-generation individuals. Therefore, when choosing, consider choosing the best individuals with the previous percentage a. In the remaining individuals, randomly select the proportion b. The individual survives and the rest is eliminated.

How to consider variation:

Considering the possibility of variation in the real environment is relatively small, so the probability of mutation is set to control the possibility of mutation, while ensuring the possibility of mutation of each gene is the same, here also consider the combined threshold If the variation is high, it may lead to a big difference, but the final experimental result is no different. On the contrary, it is easier to treat all genes equally.

4. Experiment code

---- main.py ----

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

From PIL import Image

Import numpy as np

From genetic import GA

Filename = 'images/mikaso.jpeg'

Defthreshold(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')

Defmain():

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)]

        #Select an adaptable chromosome

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

        Parents = graded[:retain\_length]

        #Select a non-strong, but surviving chromosome

        For chromosome in graded[retain\_length:]:

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

Parents.append(chromosome)

        Return parents

Deffitness(self, chromosome):

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

        Return fitness

Defcrossover(self, parents):

        Children = []

        # The amount of children who need to breed

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:

                # Randomly select intersections

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

                #Generate mask, convenient bit operation

                Mask = 0

                For i in xrange(cross\_pos):

                    Mask |= (1 <<i)

                Male = parents[male]

                Female = parents[female]

                #孩子 will get the gene of the father before the intersection and the gene of the mother after the intersection (including the intersection)

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

Children.append(child)

        # After breeding, the number of children and parents is equal to the original population, where the population can be updated.

Self.population = parents + children

Defmutation(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), chrom