AI assignment 2 Report

Rafik Hachana

BS19-04

r.hachana@innopolis.university

Introduction

This report discusses the problem of making an artistic rendition of a given image using the technique of genetic algorithms. The task needs a lot of adaptation of the toolbox of genetic algorithms for the sake of a good convergence rate towards the desired result, and in order to have a decent practical performance of the algorithm. The output of the algorithm is an image that resembles the input image (meaning that it is recognizable for the observer) that is painted using overlapping circles all over the picture canvas. The use of circles makes the output image look like a painting or a mosaic of the original reference image. The exact nature of the final "artistic effect" depends on the chosen parameters of the algorithm (e.g. circle size and opacity), these parameters will be further discussed in the corresponding section of the report.

The report is comprised of 4 main sections: The use of a genetic algorithm (population, crossover, ...), the implementation details (e.g. the choice of parameters, libraries used, ...),the output of the algorithm (e.g. comparison of input and output images,...) and argumentation on whether the generated output can be considered as art. We conclude the report with a reflection on all the sections, with the strengths and weaknesses of the algorithm, and what can be improved.

The genetic algorithm

Population and chromosomes

At first, one might go for the most straightforward choice of population and use several images as a population sample. But considering the practical performance of that choice, it is more efficient to consider the circles (which are the brushes used to paint the picture) as the individuals of our population.

Genotype

Each circle is stored as a tuple (x,y,(b,g,r),diameter).

The first 2 elements x and y are the position of the circle's center on the canvas (which is of the same size as the input image). Therefore they have to be within the intervals [diameter, h - diameter -1] and [diameter, w - diameter -1] respectively.

The tuple (b,g,r) represents the color of the circle in RGB (the order is changed from RGB to BGR because of the image library used to load and show the images).

The last value *diameter* represents the diameter of the circle in pixels.

Fitness function

The fitness function is calculated for each circle in the population. We define it using the error value of a circle, which is the arithmetic mean of the L2 error (Euclidean distance) for each pixel of the circle, compared with the corresponding pixel in the input image. The fitness of an individual is higher if their error is lower. So we use a certain tolerance of error and pass all the individuals with a lower error value. The formula for the error is as follows:

$$error(circle(x, y, (b, g, r)) = \frac{\sum_{(i,j)\in circle} \sqrt{(b - b_{ref,i,j})^2 + (g - g_{ref,i,j})^2 + (r - r_{ref,i,j})^2}}{circleArea}$$

Fitness threshold

In order to select the fit individuals that will survive till the next generation, we used a threshold value of the fitness function, this threshold value has an initial value that is quite tolerant, and then it gets less and less tolerant the more the algorithm progresses. This technique is used to tune the current solution and get it as close as possible to the desired image.

Auxiliary fitness function

The auxiliary fitness function is not used in the selection of chromosomes. It is instead used to calculate the overall fitness of the image, and therefore assess the progress of the algorithm. It is used to calculate the varying mutation rate.

Making a new generation

After selecting the fit individuals from the previous generation, we pass the fittest individuals directly to the next generation. And we used both mutations and crossovers in order to fill the population till the desired population size.

Crossover

The crossover is performed as follows:

1. Sort the survivors of the previous generation according to their (x,y) coordinate. These sorted survivors are the parents to be used in the crossover.
2. Choose a parent circle at random from the list of parents.
3. Assuming that the parent chosen in the previous step has index i, we choose a second parent from the parents in the interval [i+1,i+1000]. This way we ensure that the 2 chosen parents are close to each other on the image. This is guaranteed because of the sorting operation we did in step 1. The diameter is passed automatically as it is constant.
4. We compute the arithmetic mean of both positions and colors and assign it to the offspring.

With the above operation, we are able to generate an offspring. And the method used makes it easier for the next offsprings to fill gaps in the image. The offspring circles are between the parent circles in terms of position and color. This will make the algorithm faster especially for pictures with gradients of color.

Mutation

For simplicity, the mutation is chosen to be a complete mutation. Which means that the offspring circle is a completely random circle in terms of position and color.

The mutation rate starts high (close to 1) and gets decreased as the generated image gets better. This is due to the fact that mutations are very helpful in the start. In the start of the algorithm, we don't have so many fit circles to crossover, so crossover is not very useful in the few first iterations. The way that the mutation rate

is calculated is discussed in the next section of the report, about implementation details.

Implementation details

Variables of the algorithm

To make the algorithm more efficient, we tuned the variables such as mutation rate, population size and the fitness algorithm.

Variable mutation rate

In genetic algorithms, crossovers are more useful the more we are closed to the desired solution. That's why it is better to start at a high mutation rate and decrease it gradually as the result gets closer to the input image. The mutation rate for the current generation is calculated as follows:

$$mutationRate = min(50, overallFitness - 50)^{generationNumber/50}$$

Variable fitness threshold

By observation, we found that having a high tolerance for unfit individuals in the start of the algorithm accelerates the initial approximation of the image. The tolerance is slightly decreased with each generation using the following formula.

$$newTolerance = tolerance \times 0.9975$$

Variable population size

The more we have fit individuals, the better it is to generate more and more new individuals using crossover and mutation. The population size is updated with each generation by adding a small increment, in the following way:

$$newPopulationSize = populationSize + populationIncrement$$

Halting criteria

The algorithm halts in any of the following cases:

1. We have reached the desired overall fitness of the image (using the auxiliary fitness function).
2. We have reached the defined maximum number of generations. (can be entered by the user to limit the execution time).
3. We have the maximum predefined population size (700000 circles)

Flow of the algorithm

At a high level the algorithm works as follows:

1. Load the input image
2. Downscale the input image if needed
3. Generate a random population of circles to fill the empty canvas
4. While we haven't reached any halting criteria
    a. Select the fit individuals from the current generation using the fitness function.

    b. Make a new generation by crossover, mutation, and directly passing the fittest circles from the previous generation.
    c. Calculate the mutation rate, population size and fitness threshold for the next iteration, using the overall fitness of the current generated image.
5. Save the image.

Libraries used

The code of the solution was written using Python 3. The following libraries were used in the way explained below.

OpenCV

Opencv was used to load, show and save images, as well as stamping circles on the image with a certain opacity.

NumPy

NumPy was used to store the image while processing it, it is needed since it is fundamental for OpenCV.

Numba

Numba is a library that accelerates Python code using JIT. It was used solely for optimization purposes.

Guide to running the code

1. Check the requirements.txt file for the required libraries.
2. Run the file ‘main.py’
3. You will be prompted to enter the path of the input image. You can type null in order to use the image “sample.jpg” included in the folder.
4. For further experimentation, you can open the Python file and change the values of the brush size, brush opacity and the algorithm variables in the start of the file.

For further details you can check the “readme.txt” file included in the submission folder.

Results

In the next few pages we present input images that were tested on the algorithm, along with the corresponding output image, the last image includes a plot of how the fitness function evolves with each generation.

Input 1

Output 1 - Brush size 4px, opacity 0.5

Output 1 - Brush size 20px, Opacity 0.7

Input 2

Output 1 - Brush size 4px at 0.5 opacity

Note: The result looks almost identical because of the high image resolution and the low brush size.

Input 3

Output 3 - Brush size 4 at 0.5 opacity

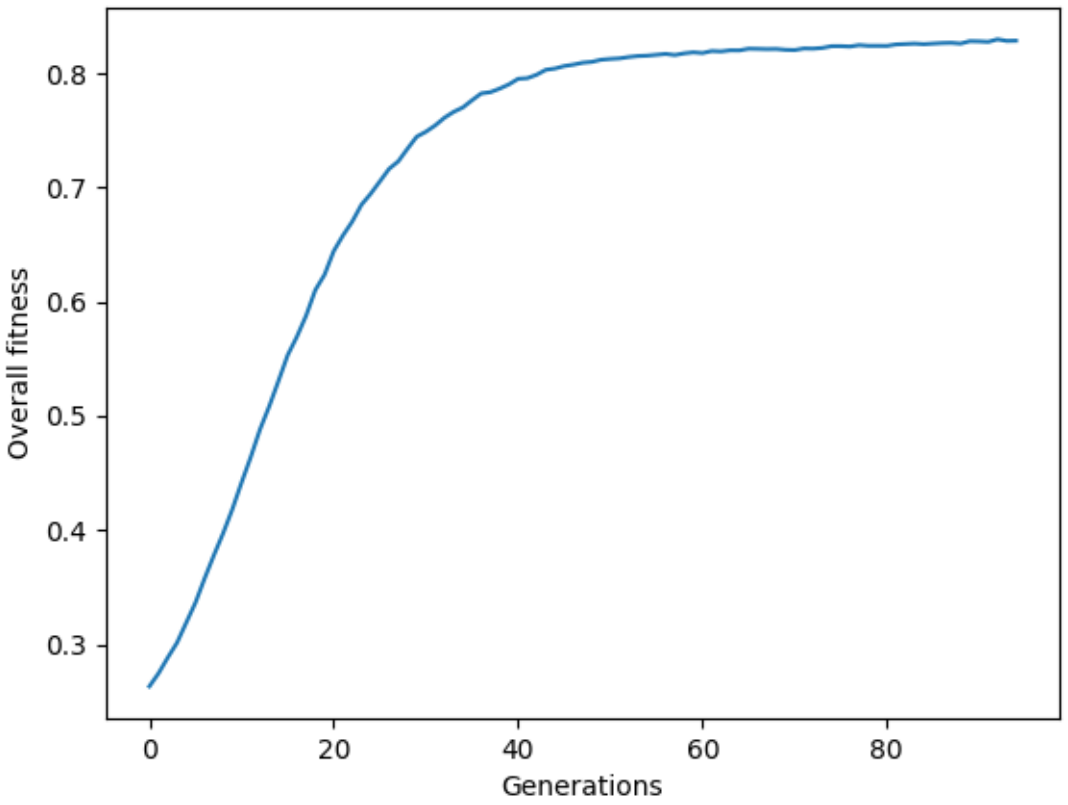
Input 4

Output after 57 generations - Brush size 6px at 0.5 opacity

Input 5

Output after 100 generations - Brush size 8px with 0.5 opacity

Evolution of the fitness value over the generation for the input 5



Did we just make art?

Since the start of the artistic trend in AI, the debate on whether we should consider the computer generated art as "real art" has been going. Usually one debating side would say that the creations done by AI are real art, and the other side would argue that it just looks like art and therefore is not real art.

I personally think that AI on its own cannot fully make art. Because the main purpose of the art is not the material work itself, but the meaning. Humans like art because of the emotions it makes them feel, and the emotions of an art don't come from the combination of color itself, but by the process of interpreting the art and giving it meaning. The meaning doesn't necessarily need to be a symbol of something, it can be just the thought that the image was created by a human as an art. When we consume art, and therefore consume its meaning, we keep in mind the fact that the art was created by another human, and we all do that subconsciously. The meaning in the art is assumed to be embedded in the artistic work by another human being, the artist.

This elaborates the question of the debate from "Can AI create art?" to "Does AI generated art have meaning?". And here the answer would depend. It depends on the intention of the creator. We can answer this by saying that AI is a tool for art creation, and this tool can be used by humans to make art and embed it with meaning.

If we consider this Python code as a tool for making art, and this tool has been used by a human to create meaningful art, then we have indeed created art. However, it is the human who chose to create the art in that way and to use the AI for that purpose, the AI by itself didn't create art, it just created the image representation of the art.

Reflection and conclusion

The algorithm created here is an interesting application of genetic algorithms, it can be used as a tool for art creation. However, it is just a tool, it cannot make creative decisions on its own. The algorithm also has some technical weaknesses, it can be further optimized for faster and more optimized results. It can give a decent result (a recognizable image) within 1 minute. A good result would usually take around 15 to 30 minutes depending on the image and the algorithm variables.