[S19] Introduction to AI

Assignment 1. Report

Author: Ruslan Sabirov
Group: BS17-02
GitHub: https://github.com/russabirov1998/IAI/tree/master/Assignment%202
(for code and more examples of running)

0. Plan

1. Introduction

The work started by finishing simple tasks given on labs just to practice in simple evolutionary algorithms. After that googling process started. I tried to find the library (module) that operates with images in my favorite language Python. I found simple library PIL which can see pixels on the image and draw on it.

2. Algorithm

Basic idea

The idea is based on working with blocks of pixels. Block is an square of N*N pixels size. Usage of blocks not separated pixels allows to see results of the algorithm by human eyes and leads to speed-up the algorithm.
Algorithms start with the base image and try to transform it into some perfect image, both of them are any pictures with size 512x512.

Constants

The algorithm has 3 main features (functions): generate individual, mutation, crossover. Each has constant parameters: the number of cycles to do and block size. There are also IMAGE_SIZE (512 by default), POPULATION_SIZE (maximum number of individuals in the population), EVOLUTION_CYCLE (number of iterations: mutations, crossovers), BEST_PARENTS_COUNT (count of parents that are participating in mutation and crossover).

Fitness function

For all pixels and for all 3 color-parameters in RGB-model sums absolute difference between the analyzing picture and perfect picture.

Generating individuals

Generates individual based on the base picture.
For some number of blocks:

1. Select coordinates for the first block
2. Select coordinates for the second block
3. Swap pixels of the first and second picture

Mutation

Changes some genes in the individual and returns new child
For some number of blocks:

1. Select random coordinates as a left-top pixel of the block.
2. Calculate the best rgb for the block.
3. Draw the best generated rgb color on the block

Crossover

Gets two parents as arguments and returns child # which has the best genes of both parents.
For some number of blocks:

1. Select random coordinates as a left-top pixel of the block
2. Select the best rgb from parents
3. Draw the best rgb to the child

3. Issues Faced

Python module

It was hard to search for an appropriate module without any knowledge about working with images in the programming language. Forgetting about GA helped: easy module was found and the genetic algorithm was designed after that.

In-place drawing

Unexpectedly, find out that drawing works in-place (it does not create a new image, but changes current). Conclusion: read the documentation carefully. Solved by cloning picture and drawing on the new one.

Inheritance

I wanted to save fitness value for each picture once it calculated and do it by inheriting from one of the PIL classes and adding attribute and method for function. That was impossible because PIL uses different classes for an image file, image itself, image pixels and drawable objects. The solution was found: I created my own class which is not inherited from any other class and added two attributes (for image and for fitness value) and few methods (for loading pixels, for cloning, for creating a drawable object and so on).

Memory

First runs of the program wasted too much memory. One of them even crushed my laptop by blue death screen. Dealt by optimizing an algorithm and liquidation unnecessary cloning.

Time

As all GA programs mine works very slow. It is because of working with objects of 512*512 = 262144 pixels, each has 3 parameters (red, green, blue) for coloring. Used some asymptotical (e.g., generated not one of the 255^3 colors, but 10^3 colors) and language optimizations (e.g., added all code which is not in the function to function main()). The algorithm could be also parallelized in the future.

4. Conclusion

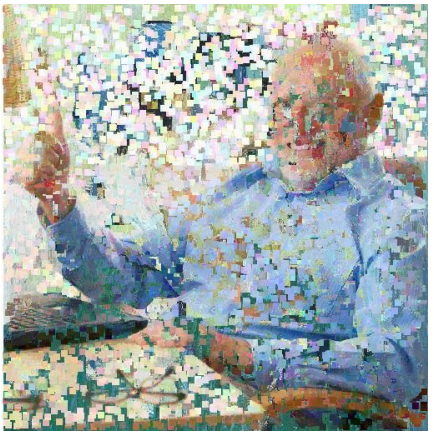
It was a good idea to give students such kind of practical assignment. It is really cool that we can create art on a computer. Genetic algorithms are very useful in the case when the solution by typical algorithms (step-by-step) is very hard to implement or design or does not exist at all.

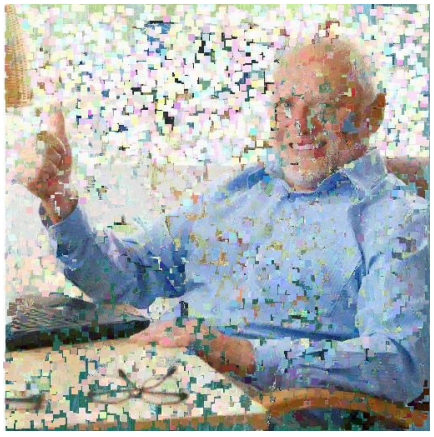
For very good result 100-150 iterations are not enough, need to increase it.

For me output images are art, they do not look like regular images (there are some holes with background) and that makes it extraordinary. Is not that new style in the art?


base image


iter 20

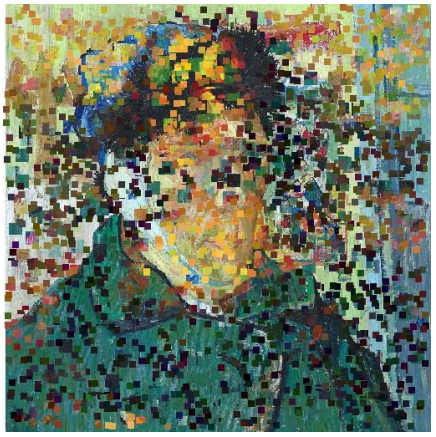

iter 40

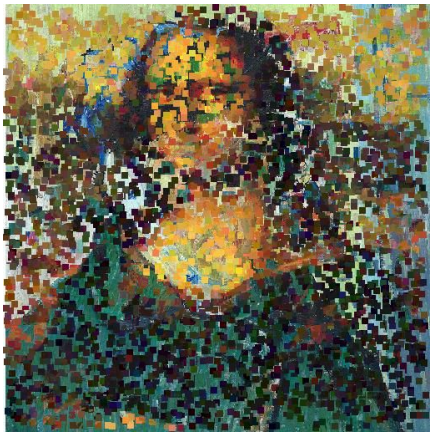

iter 60

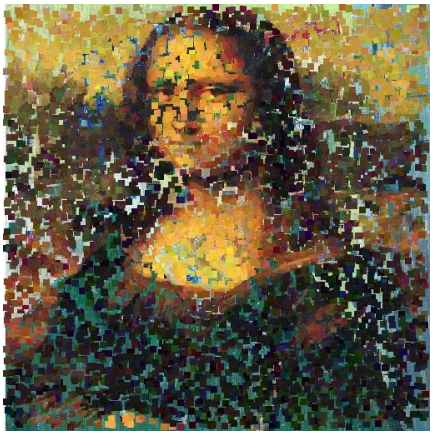

iter 80

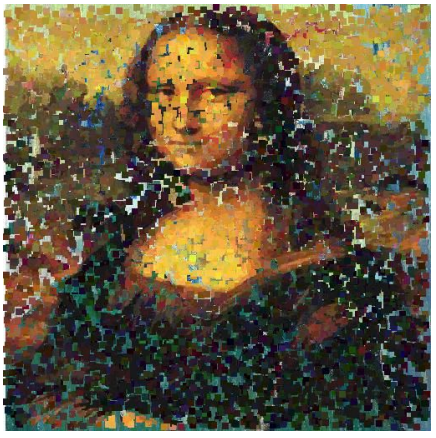

iter 100

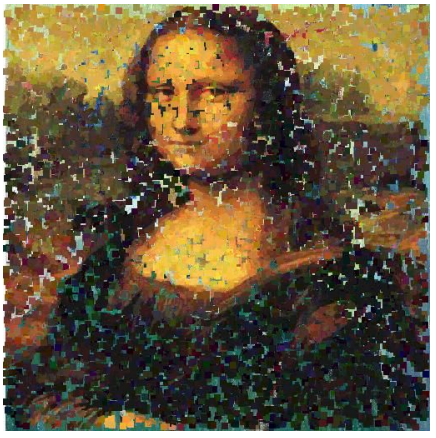

base image


iter 20


iter 40

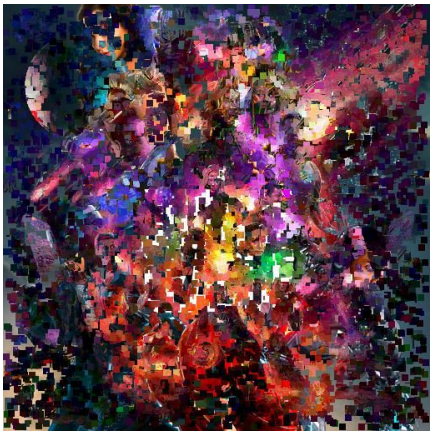

iter 60

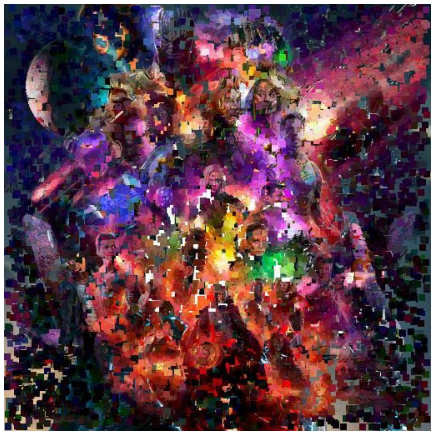

iter 80

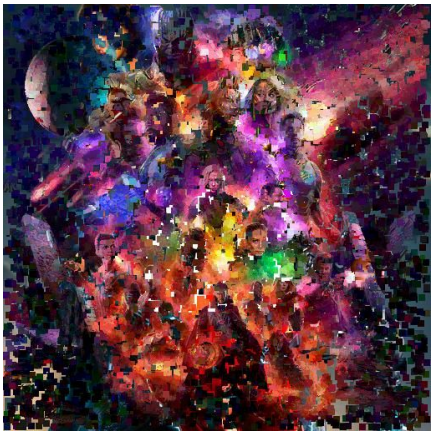

iter 100

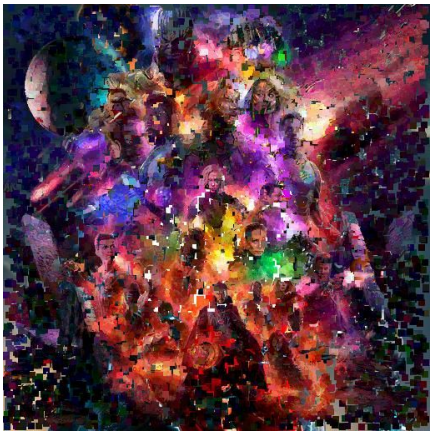
initial image


iter 30

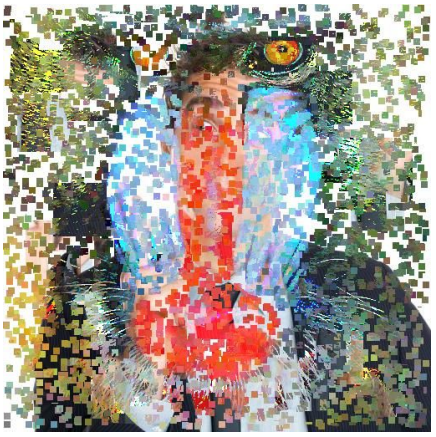

iter 60

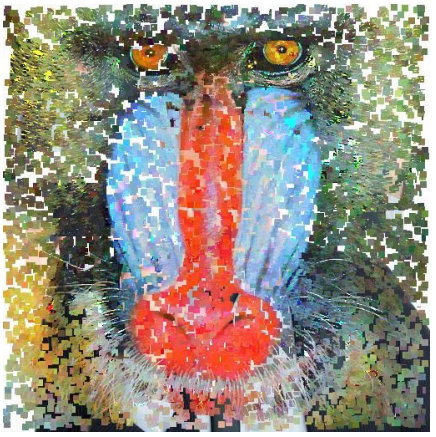

iter 90

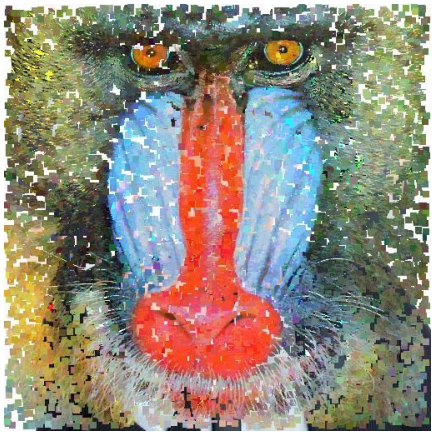

iter 120


iter 150


base image


iter 30


iter 60


iter 90


iter 120


iter 150