Introduction to AI (IAI)

Assignment 2. Innopolis University, Spring 2021

Name: Kerim Kochekov
Group Number: 5th group (BS18-DS-02)
Professor: Joseph Alexander Brown
TA: Anton Egorov
Github link: https://github.com/KerimKochekov/IAI_Assignment2

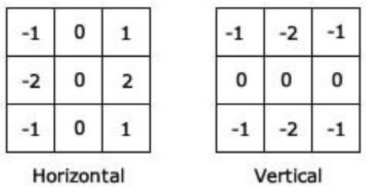
Part 1: Description and Clarification

- Task Statement: The reproduce an image similar to input image using artistic way with Genetic Algorithm.
- Task Constraints:
    - Input image: 512x512 pixels image
    - Output image: 512x512 pixels image
    - Algorithm based on Evolutionary algorithm "Genetic algorithm"
    - No usage for external libraries that related to Evolutionary algorithm
- Brief solution: I like idea of machines to draw white-black picture without using any color similar to human drawings, where there is no rule of straightness of lines or colors. Besides it looks beautiful and artistic. The drawing by machine based on edge detection technique called "SobelEdgeDetection". With the help of this mathematical model, we can evaluate our generation results using fitness function to check how far from desired image. Our fitness function basically finding difference value using Mean-Squared error. The algorithm works pretty much like classic genetic algorithm starting with initial population and start to generate new populations for 2000-3000 times(generations). For each generation we do crossover between all pairs of current population and apply mutation for some of them. Laterward compute all their fitness value and choose the best ones for next generation, which we call it selection part. Mutation in our algorithms follow: randomly fix circle (random radius and random center) inside picture and change the pixels of image inside circle according to a normal distribution random value.
- Another my idea was producing image from small pictures dataset, which we also call it also "Mosaic". But later I thought, lot of students will apply this technique, so I decided to do somehow different thing.
- Structure and details of project:
    - Project implemented in C++
    - Implemented "util.h" for reading and writing jpeg using help of "jpeglib" library
    - All algorithm implemented in single "art.cpp" file, here is structure of file:
        - Used ready "SobelEdgeDetection" algorithm for edge detection
        - Used "artImage" class for keeping picture in a "char" format and make flatten. In case of usage end, delete from memory for optimization
        - Implemented classical Genetic algorithm, so you can configure mutation and crossover probabilities as you want. Also, you can freely change population size and number of generations for getting better performance.
    - /output directory contains all evolution of process of algorithm(fixed 10 to not spam lot of images). It will be cleared before each run, so if you want to not lose this evolution, save to different directory before running once more. Do not delete this directory for sake of algorithm.
    - Population size is 5 (5*4/2=10 candidate images in each generation)

Part 2: Description and Clarification

Chromosome description

First RBG 512x512 image transformed to grayscale image that contains only one channel(3 channel to 1 channel) and applied Sobel edge detection algorithm in both vertical and horizontal way using Sobel kernel(3x3 filter matrices). The idea of algorithm is to detect edges where color values change dramatically between the sides of the edge in grayscale image (only with values 0…255 we can detect) and it can be applied for colored images too. More formally this algorithm is a gradient based method. It works with first order derivatives, calculates the first derivatives of the image separately for the X and Y axes (horizontal and vertical). The derivatives are only approximations (because the images are not continuous). To approximate them, the following kernels are used for convolution:



-1	0	1
-2	0	2
-1	0	1

Horizontal

-1	-2	-1
0	0	0
-1	-2	-1

Vertical

From these applied kernels we will get two values G_x and G_y for each pixel and calculate angles to round one of the nearest four possible angles for representing vertical, horizontal and two diagonal directions. Gradient direction is always perpendicular to edges:

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. So, it is checking if it is a local maximum in its neighborhood in the direction of gradient. After all it end we will endup with **sobel(x,y)** values for each pixel so we can compute our fitness function and represent our chromosome.
Our chromosome representation is flattened char array with size of 262144(512x512) values in range [0,255] (formal grayscale image representation)

Fitness function
Because of our calculated ready desired **sobel(x,y)** values, we can apply classical MSE(Mean Squared Error) to determine how far from intendent result. How much smaller fitness value, much better representation in our task.

$$MSE(current) = \frac{1}{512 \cdot 512} \sum_{x=0}^{512} \sum_{y=0}^{512} (sobel(x,y)\ -\ current(x,y)\ )^2$$

Crossover
After each generation we apply crossover for each possible pair of current population with probability 0.5. I tried to simulate real human-gene production. For each gene it either inherits from father or mother with equal probability 0.5. In more detail, for each crossover for given 2 parents, we take near half of pixels from one parent and rest from another parent in random way. I thought that this idea may be interesting in my application, and I think it affects the evolution mechanism of the final picture in a positive manner.

```
const int crosoverProb = 2;
artImage* crossover(artImage *father , artImage *mother){
    artImage *ret = new artImage(father);
    for(int j = 0 ; j < father->H ; j++)
        for(int i = 0 ; i < father->W ; i++)
            if(rand() % crosoverProb == 0)
                swap(father->pixels[j][i] , mother->pixels[j][i]);
    return ret;
}
```

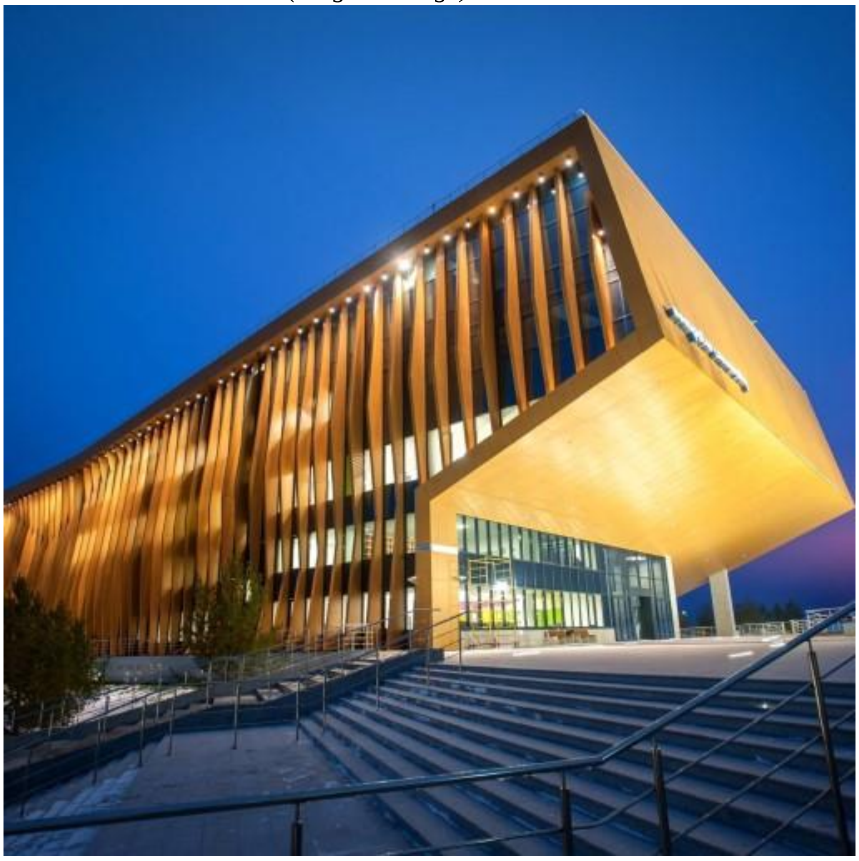
Mutation
Personally, I believe most important part of algorithm. Without mutation population will stuck on some local minimum and do not move from there. I decided to use random circle approaching for this part as I talked brief in introduction. In $\frac{1}{3}$ probability we will apply mutation for population for discovering new parts with new circles. Basically, we will fix some random radius and center on the image and do excavation work for explorations. For each pixel inside fixed random circle set new value using normal distribution value(with alpha, it reduced 10% after each 300th generation). The parameters of distribution vary for each possible pixel. The best part of this mutation is that we can discard this mutation if exploration is unsuccessful, where new fitness value more than old one.

Selection
After each mutation process for some of genes, we have to apply selection process for next generation. For this part I decided to use straight-forward selection process for this task. We basically sort new population by their fitness value and take prefix of fixed population size as next population(the best 5 genes in our case)

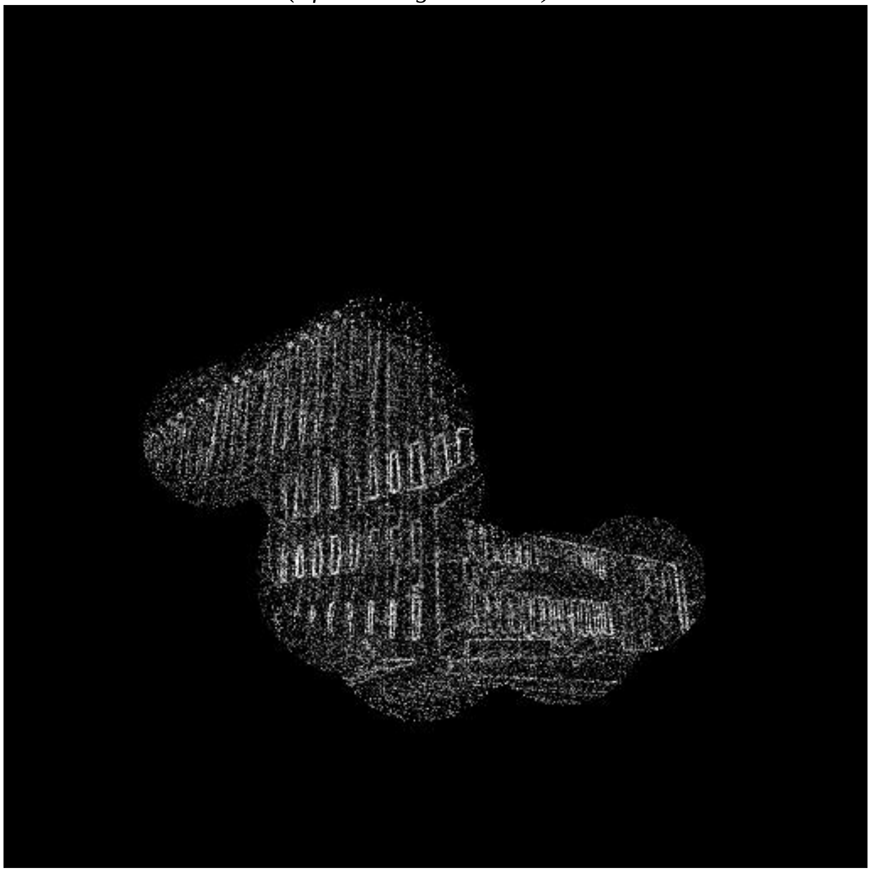
Part 3: Results

Note: Algorithm applied on Innopolis image that has been taken from internet and resized to 512x512

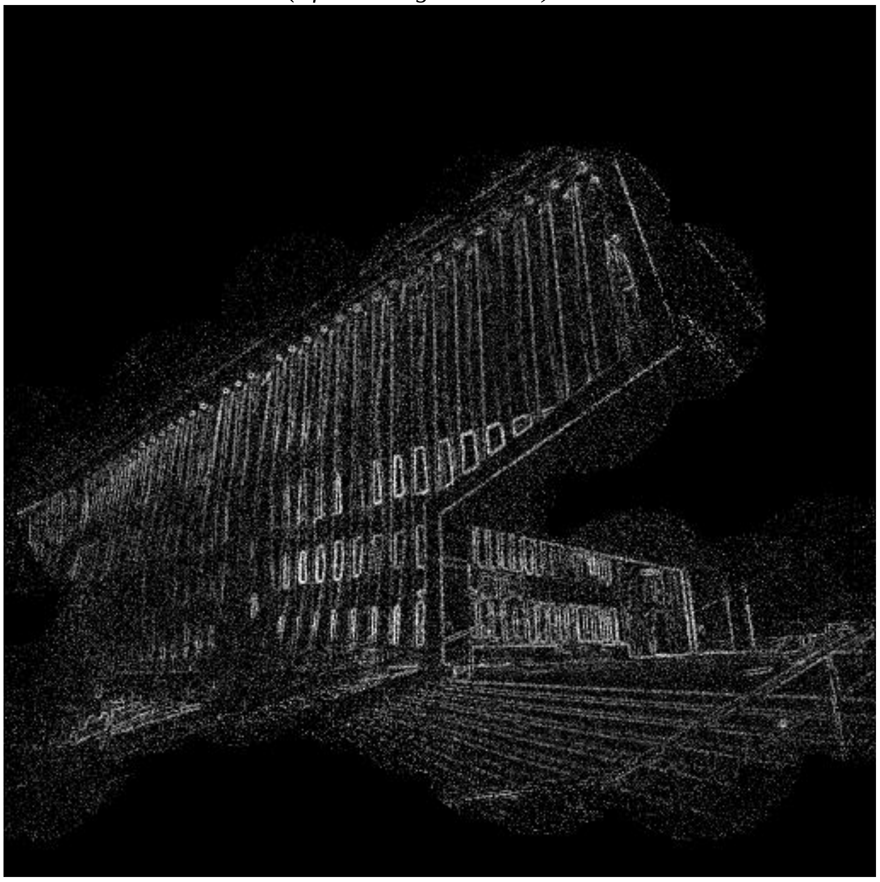
(Original image)



(After 1000 generations)



(After 1810 generations)



*(**FINAL RESULT** – After 3000 generations)*