Introduction to Artificial Intelligence

Assignment II

Daniil Manakovskiy
d.manakovskiy@innopolis.university
B18-02

Table of contents

Introduction

The main objective of an algorithm described below is given an image simulate it is being drawn by lines using a pencil. The goal is achieved using 2 genetic algorithms: the first one given average line length chooses position, length and rotation of a new line, then runs the second one, to determine the best possible color of the line.

An idea is to start from an image filled with some color and add lines over the image mixing line color with the image itself with weight utils.OPACITY.

Parameters

There are several parameters that can be tweaked in order to tweak the result or improve convergence speed.

Result affecting parameters

1) **main.LINE_LENGTH** - determines the average line length. Empirically it was determined that 64 (pixels) is the best length to draw a big object at the image.
2) **utils.LINE_LENGTH_OFFSET** - final line length is randomly chosen from a uniform distribution on a range **main.LINE_LENGTH ± utils.LINE_LENGTH_OFFSET** but not less than 15 pixels.
3) **utils.OPACITY** ∈ [0, 1] - determines the weight of line color in the final image i.e. $final\_color[y,x] = line\_color * \mathbf{utils.OPACITY} + image\_color[y,x] * (1 - \mathbf{utils.OPACITY})$

Performance affecting parameters

1) **main.POPULATION_SIZE** - the size of the population of the GA responsible for line placement.
2) **main.BREEDING_INDIVIDUALS** - the number of individuals to be selected as parents in the GA responsible for line placement.
3) **main.BACKGROUND_COLOR** ∈ [0, 255] - determines what color all individuals of the initial population will be filled with. In order to speed up the convergence, set to the most common background color.
4) **color.COLOR_POPULATION_SIZE** - the size of the population of the GA responsible for color determining
5) **color.COLOR_PARENTS** - the number of individuals to be selected as parents in the GA responsible for color determining.
6) **color.COLOR_BEST_TO_NEW_POPULATION** - the number of the best-colored individuals to be directly transferred to the next population.
7) **color.COLOR_MAX_EVOLUTION_STEPS** - number of evolution steps to be performed before finding the best color match.

Note: Please be sure that :
COLOR_POPULATION_SIZE ≥ COLOR_PARENTS + COLOR_BEST_TO_NEW_POPULATION

How to run the code

1) Install python >=3.6.9
2) Install dependencies:
    a) Install opencv for python: `sudo apt-get install python-opencv` or follow the official guide.
    b) `pip3 install -r requirements.txt`
3) Change variable **main.IMAGE_FILE** to the new image path
4) Tweak parameters described in the section above. (I suggest you to tweak **main.BACKGROUND_COLOR**)
5) Run `python3 main.py`

Algorithm explanation

Representation

Line placement

Inside the memory, an individual is a 2-D array of dimensions (<image y-size>, <image x-size>) containing unsigned integers of length 8 (values 0-255) standing for the brightness of a particular image in grayscale mode.

Color determining

An individual is represented as 1-D np.array of length equal to the number of points lying in a discrete representation of a given line. Each value stands for the intensity of the pixel after mixing the line and individual.

Fitness function

The fitness function for both GAs is the same. It would be better to call it 'error function' since it returns how much an individual diverges from the target by intensity:
$error = sum(abs(target - individual))$

Image manipulation

The algorithm converts an image to a GrayScale mode during the pre-processing via OpenCV.cvtColor, then draw lines on it pixel-by-pixel during the evolution execution. An algorithm to draw a line is as follows:

1) Randomly choose a point on a plane to start the line (x_start, y_start)
2) Randomly choose an angle from interval $[0, 360°]$ and convert to radians
3) Select line length from the interval described by **main.LINE_LENGTH** and **utils.LINE_LENGTH_OFFSET** (see Parameters section).
4) Convert line properties to the final point (x_end, y_end) using the following formula:

$$\begin{cases} x\_end = int(line\_length * np.cos(angle) + x\_start) \\ y\_end = int(line\_length * np.sin(angle) + y\_start) \end{cases}$$

5) Shrink each coordinate at each axis by taking $coord = min(coord, image[axis])$ and then $coord = max(coord, 0)$
6) Discretize the line by code found here.
7) Feed the line discretization along with the target line to the second GA to determine the best color.
8) Draw the result of the second GA.

Selection

Line placement

I suppose that if each line on the image corresponds to the best possible line, the whole image is approximated as good as possible thus for line placement I just select some number of best individuals by fitness.

Color determining

For color determining the assumption made before is false thus the selection procedure isn't that straight forward. First of all, I check whether there is an individual with 0 error (sometimes it happens), if so - immediately stop the GA and return the best individual. Otherwise, I sum up fitness (error) of all individuals and then divide the sum by fitness of a particular individual. Let call obtained array P:

$P = \begin{bmatrix} \frac{err_{sum}}{err_1} & \frac{err_{sum}}{err_2} & \cdots & \frac{err_{sum}}{err_n} \end{bmatrix}$

Array P contains values that we would like to maximize, so the bigger P_i - the better. We can use that fact to construct P' as a cumulative sum of P and then normalize it by division on the biggest element of P. After all, we have an array P' where all values are in increasing order in an interval (0, 1] and the smaller err_i the greater $P_{i+1} - P_i$. So then with probability p uniformly distributed between 0 and 1 the first individual such that $p <= P'_i$ goes to a new population. So we have a fitness-based selection where every individual has a chance to survive.

Crossover function

Since an individual is represented as an image (or its part), both crossover functions in a different manner recreate an image from existing ones. Due to such nature, both crossover functions are relatively simple.

Line placement

Since drawn lines are not stored, I decided to do a 1-point crossover over the y-axis: an image is cut by half, and a child has the first half from parent 1, the second - from another.

Color determining

The crossover function is a simple math average between 2 parents.

Mutation criteria

A mutation is a main driven force of my algorithm since that is the only way for individuals to get new lines.

Each individual created after the crossover mutates with probability of 90%. Such a high probability is being balanced by relatively small changes mutation brings to the individual. Each mutation draws from 1 to 5 new lines on an individual with equal probability.

Perception

What is art

I consider any representation of thoughts, views, dreams to be some piece of art, because it shows the creator's perception on the life situation, some object or just random thoughts.
Let’s take a look at some examples of art:

- Vincent van Gogh - The Starry Night (1889)
  According to Naifeh & Smith 2011, p. 747, the painting represents the view from the east-facing window of the author's asylum room at Saint-Rémy-de-Provence, just before sunrise, with the addition of an ideal village. Several sources clearly state the visual the painting is based on. As a final result we see that visual mixed with a unique perception of the author.

- Wassily Kandinsky (“Father of Abstract Art”) - Composition VII (1913)
  Kandinsky often called his creations “improvisations”, thus it might be a big discussion what is illustrated in the painting, but what isn’t arguable is that the painting represents some visuality from

the creator’s mind.

- Unknown author - WHITE PLASTIC CHAIR – WOOD/METAL FRAME.
  One may argue that this chair isn’t a piece of art but a usual chair. I would disagree with it, because the author mixed some visuals and created something unique (even if it was manufactured to millions of copies). People devalue such pieces of art, because they become very common in their lives.

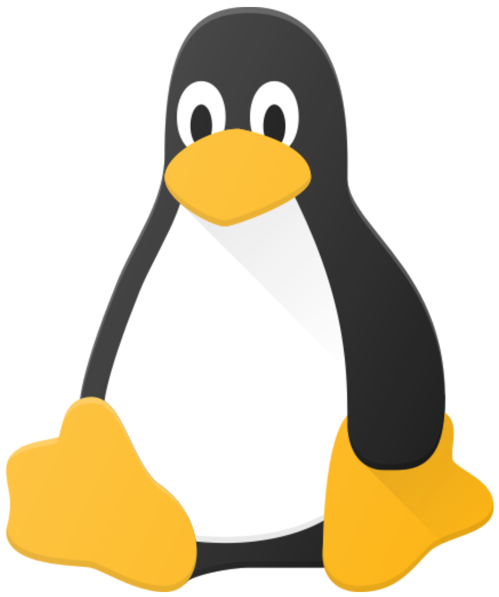
Why the output image is a piece of art

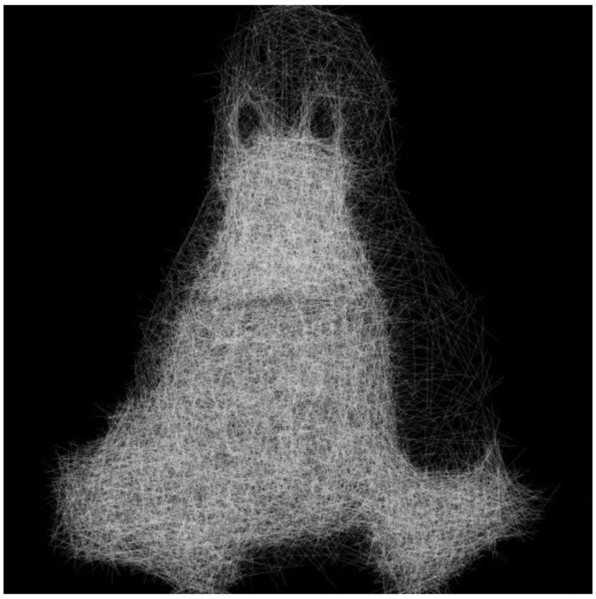
Similarly to previous examples, my algorithm produces the result based on a visual image that is being uploaded to the memory combined with the unique perception based on a pseudo-random number generator and the result of a particular iteration (provided by a fitness function).

Examples

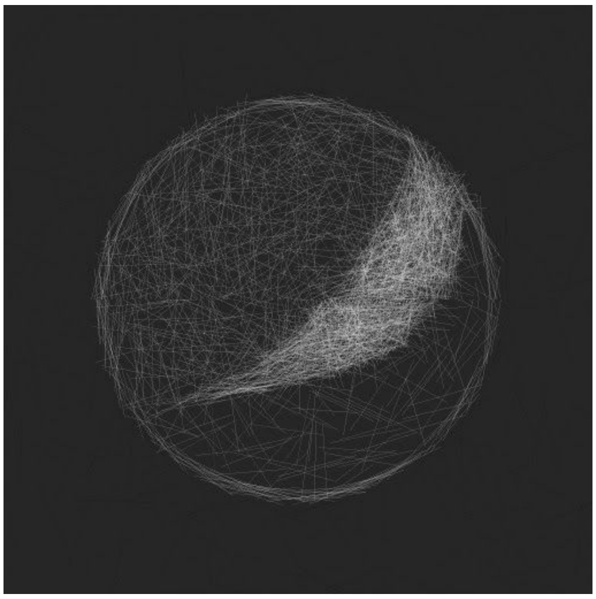
Note: Images presented below and gifs of their convergence are available at example/

1) Linux logo (initial intensity = 0, since originally image has transparent background, iteration = 8000). In grayscale mode, the initial image has undetectable edges, but the algorithm managed to restore them, although details in the penguin are indistinguishable.

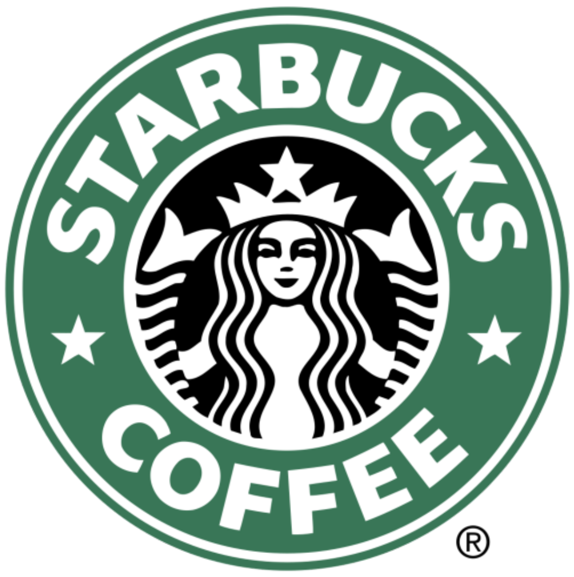


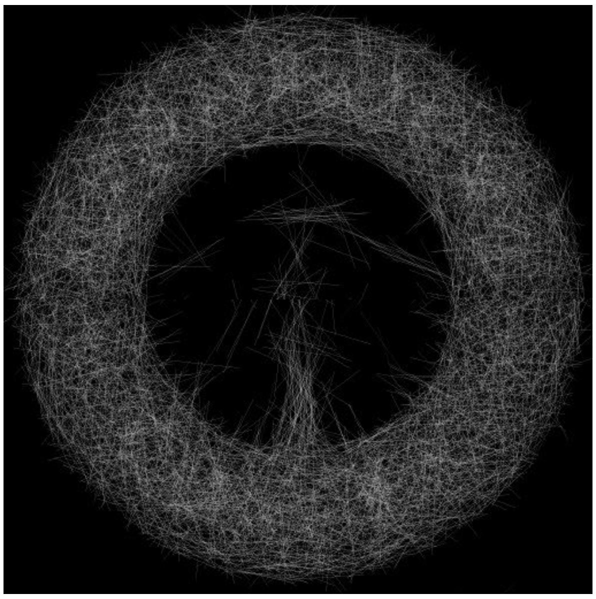


2) Pepsi logo (initial intensity = 38, iteration = 2500). The logo more looks like a hand-drawing: it has a bit inaccurate strokes, color differentiate by different stroke density and intensity.

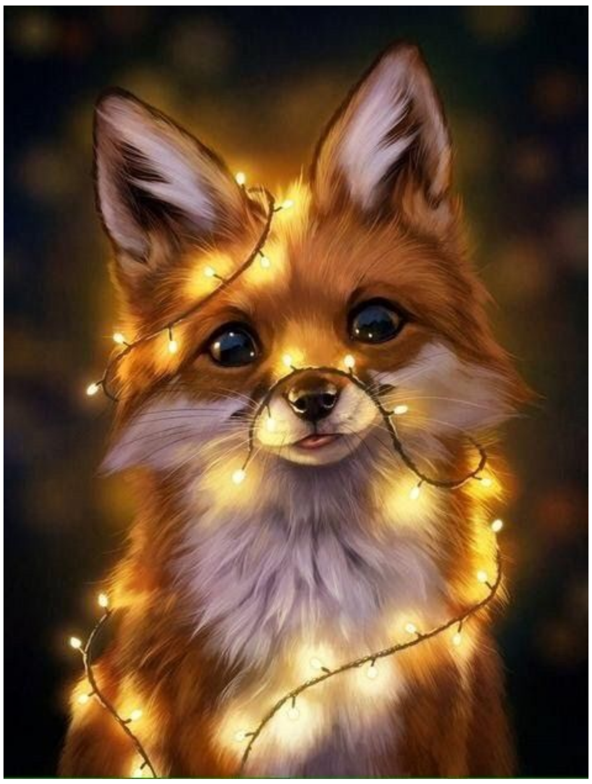


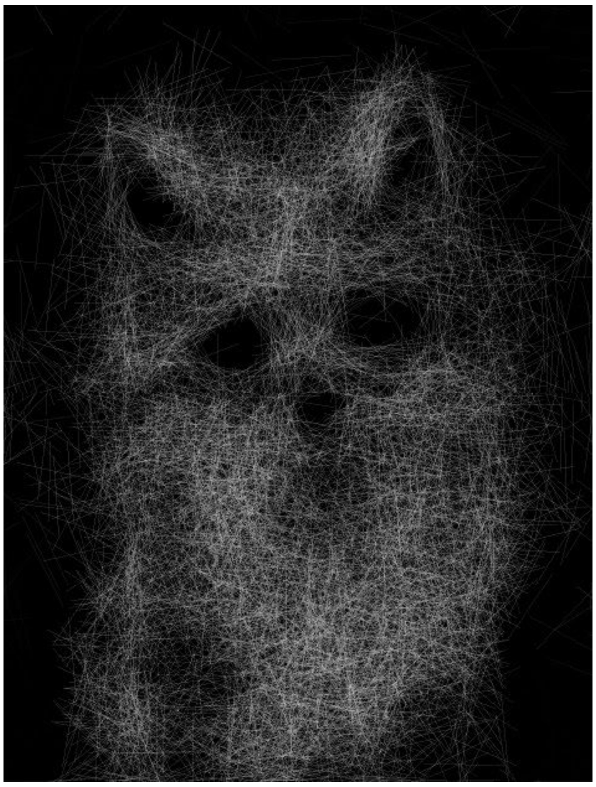
3) Starbucks logo (Initial intensity = 0, iteration = 3000).
The algorithm is bad in recreating small details. But the circle looks like a nice donut.



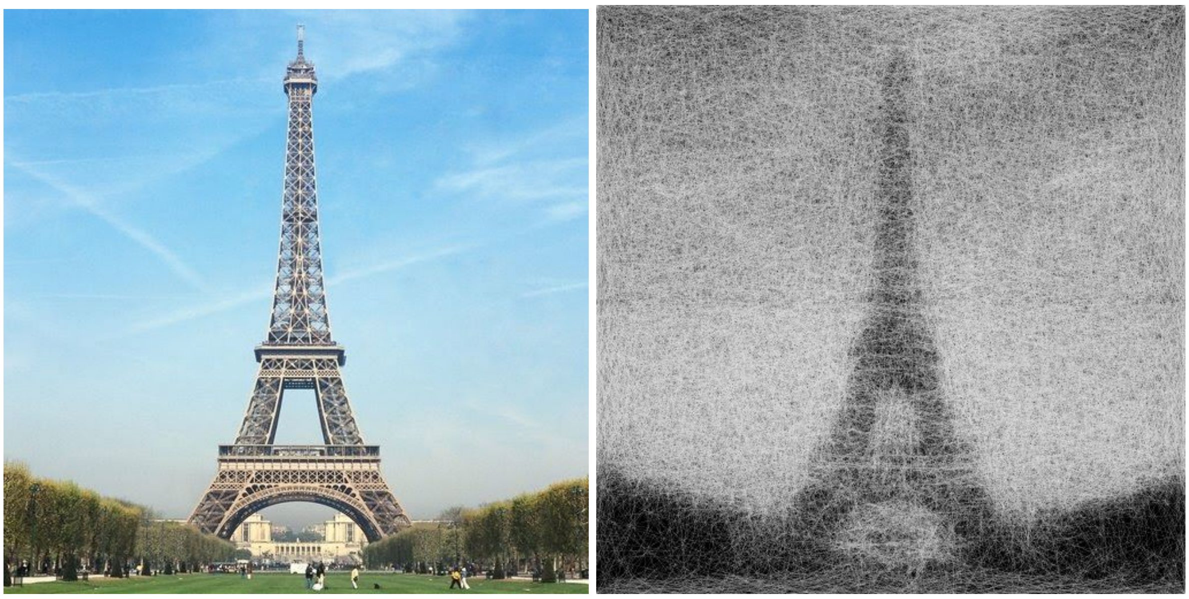


4) A fox (initial intensity = 0, iteration = 3500).
Again we see an overall good representation of the object and colors. Sharp, distinguishable edges, make the image look more like real drawing.

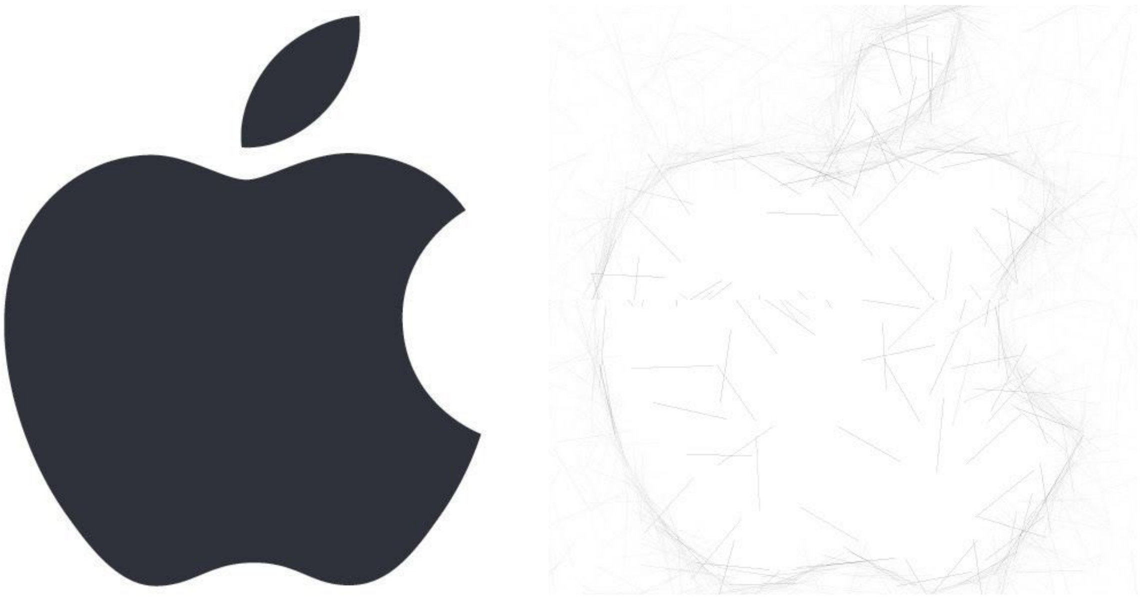




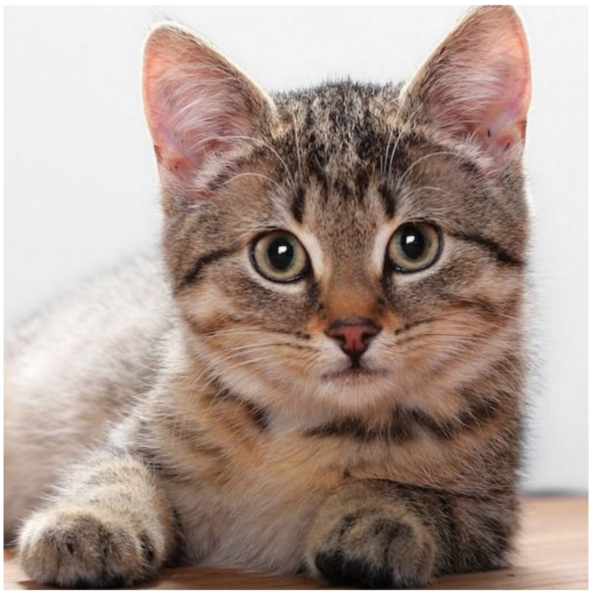
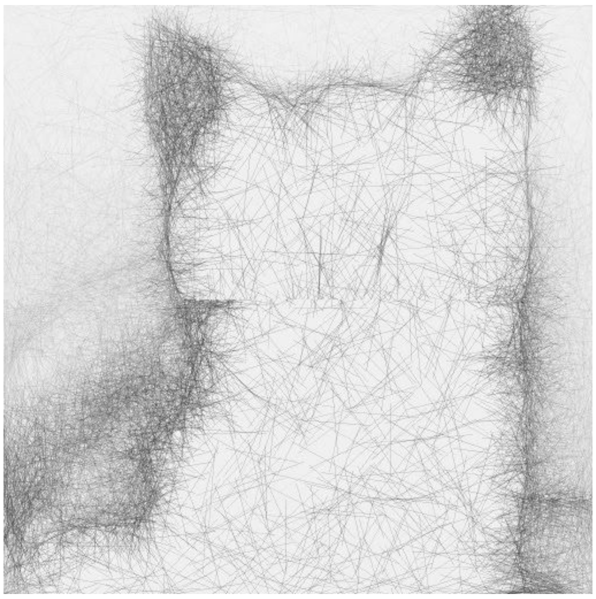
5) The Eiffel tower (initial intensity = 0, iteration = 16500).
The example shows the behavior of the algorithm if the initial background-color is completely different from the original one. The algorithm tries to recreate the background and then an outline of the image. As you can see, it took much more time compared to other examples.



6) Apple Inc. logo. (initial intensity = 255, iteration = 5000).
On big solid contrast objects, the algorithm starts outlining them from the background. Such behavior was initially unpredictable, but it looks good for me and even better fits into a model of human-artist that draws his/her sketches up to a different level of details.



7) A cat (initial intensity = 240, iteration = 8500).
Another example of an image that is being outlined.

8) Plains (initial intensity = 100, iteration = 23000).
If an input image has no distinguishable objects, the result becomes more abstract.