Genetic algorithms for art generation

Gleb Petrakov
g.petrakov@innopolis.ru

Innopolis University, 2019
Spring 2019, Introduction to Artificial Intelligence

Tools and packages

For making a genetic processor, several tools and packages were used:

- **Python 3** – as the main programming language
- **Jupyter Notebook** – as the main development environment
- **Scipy Numpy** – as a tool to work with matrices
- **Matplotlib** – as a tool to show graphs

These tools are under legal copyright notices of their creators.
None of the code of these tools was modified.

Genetic algorithm abstract

For creating art from the base image the genetic algorithm is used. Genetic algorithm (GA – for short) is a type of algorithm that is similar to one evolutionary from mother nature. In nature (and similar in GA) the goal to improve result with each new population.

The population is a set of similar individuals (also called chromosomes) that have a common build structure. Every individual has its own genes – the representation of one's properties. Making steps towards improvement should lead to an ability to choose the best from a population, the way of choosing is usually being represented as the error function, that is defined with respect to the ideal we want to reach. When the best are chosen, they may be mutated – their genes are being changed in any way possible with any probability. Genes may also be crossed (crossing over) within the best of the population.

Step by step on every iteration the best of the iteration is better and better compared to the ideal. We can stop, when we are close enough or when we reached some population changes limit.

Genetic algorithm for creating art

Given the abstract, let me define some constraints for creating art with genetic algorithms.

First of all, we should have some ideal image, we want to achieve (note that it may be not only the base image). Mostly we are working with raster graphics in RGB palette, so let me define error function as a basic image and processed image mean squared error for mapped pixel colors. For this purpose, let me consider an ideal image as a base image.

The initial population can consist of single-colored images. The fill color may be defined as a mean value of colors from the base image, so we could reduce the initial error. Let the population individual be a single image since we have to choose the best one for the art. Let the individual genes be some basic shapes with some colors.

The mutation will be considered as adding some number of these random shapes to every population member. The color of a shape will be chosen from any point of a base image to increase error function drop through evolution. The next generation will be produced as copies of only one best image from the current population.

To sum up, these are the steps:

1. Create a base population as a single-colored image
   with the mean color of the base image fill
2. Mutate the population by adding random shapes with colors from the base image
3. Choose one best of the new population
4. Create copies of the one best as a new population
5. Repeat from step 2

Technical details

The algorithm was implemented using Python 3 programming language within Jupyter Notebook environment. Numpy is used to manipulate images as matrices, OpenCV to manipulate image files and Matplotlib to draw plots. Please ensure that you have everything installed before running.

The base image and its path are being stored in a static class `Input` as properties `Input.IMAGE` and `Input.IMAGE_PATH` respectively and are being accessed during the execution.

One individual is being represented as a **`Chromosome`** class.

Can be initialized from any image.

Important properties:

- `self.image` – the main representation of a chromosome's genetic code, can be directly interpreted as an image.
- `self.shapes_generated` – the number of shapes that will be generated during mutation
- `self.shape_size` – maximum shape size (minimal is zero)
- `self.points` and `self.colors` – points and colors of the shapes inside a chromosome

Methods:

- `self.init_image` – initialize image as a mean color from the base image
- `self.generate_colors` and `self.generate_points` – generate random colors and points
- `self.generate_image` – generate an image based on colors and points

Class **`GeneticGenerator`** manages the population and performs mutations with a selection.

Important properties:

- `self.chr_shape_count` and `self.chr_shape_size` – the number of shapes that will be generated during mutation and maximum shape size (minimal is zero) to be passed to newly created chromosomes
- `self.population` – population
- `self.population_size` – population size
- `self.error` – error of the current population
- `self.error_history` – errors of all populations ever processed

Methods:

- `self.generate_population` – generate a new population from given params
- `self.population_error` – calculate population error
- `self.select_best_chromosome` – select the best in the population
- `self.perform_selection` – iterate selection process

Some misc utilities and main iteration process are conducted with the **`Main`** class.

Methods:

- `self.init_figure` – initialize Matplotlib figure with statistics
- `self.update_figure` – update Matplotlib figure with new data
- `self.change_parameters` – change needed parameters
- `self.iterate` – do n iterations of aging
- `self.autorun` – do all the stuff

Executing

The program takes some arguments to define GA properties:

- `--shape-count` – the number of shapes to generate in one chromosome (default: 5)
- `--shape-size` – maximal shape size (default: 20)
- `--population-size` – population size (default: 35)
- `--epochs` – how many evolutions there will be (default: 2000)
- `--splits` – how many intermediate outputs will be generated (default: 10)
- `--image` – [**required**] path to the base image

If you are running the program within some notebook, please be sure to specify arguments in the options parameter in the main block a list like it would be in the command line: e.g. `['--image', 'assets/house.png']`

Please, be sure to satisfy all dependencies. There may be major issues in importing if you are using some cloud environments. Needed libraries provided commented for the apt package manager.

Result

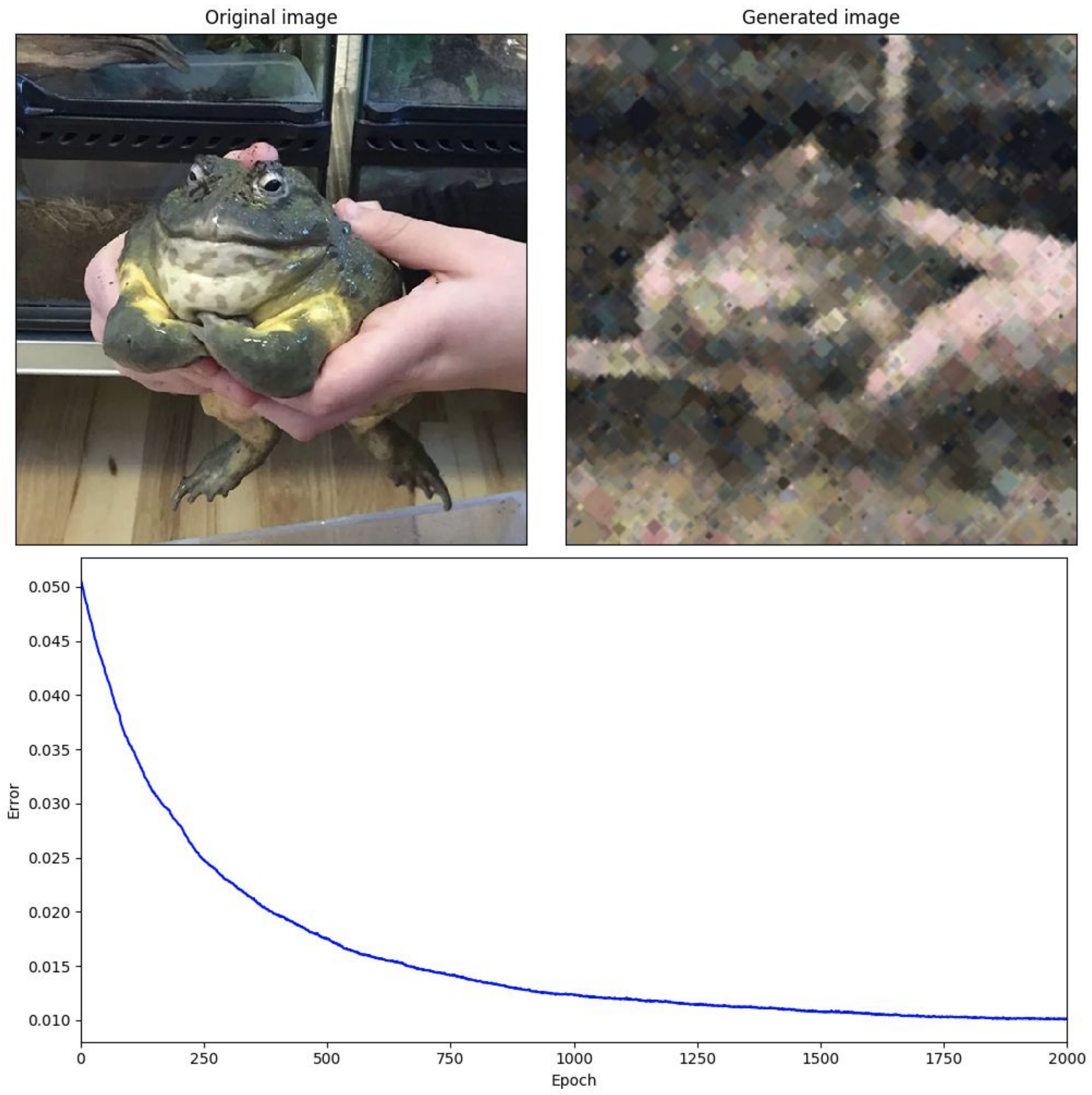
This genetic algorithm provides us some nice picture, which may be considered as an art of fauvism movement done with big strokes of oil paint with expected certainty. As a provider of close to the ideal image art, this GA may be considered good, but in creating something new and inspirational it lacks creativity for sure. Examples of the works you may find below.

Sources

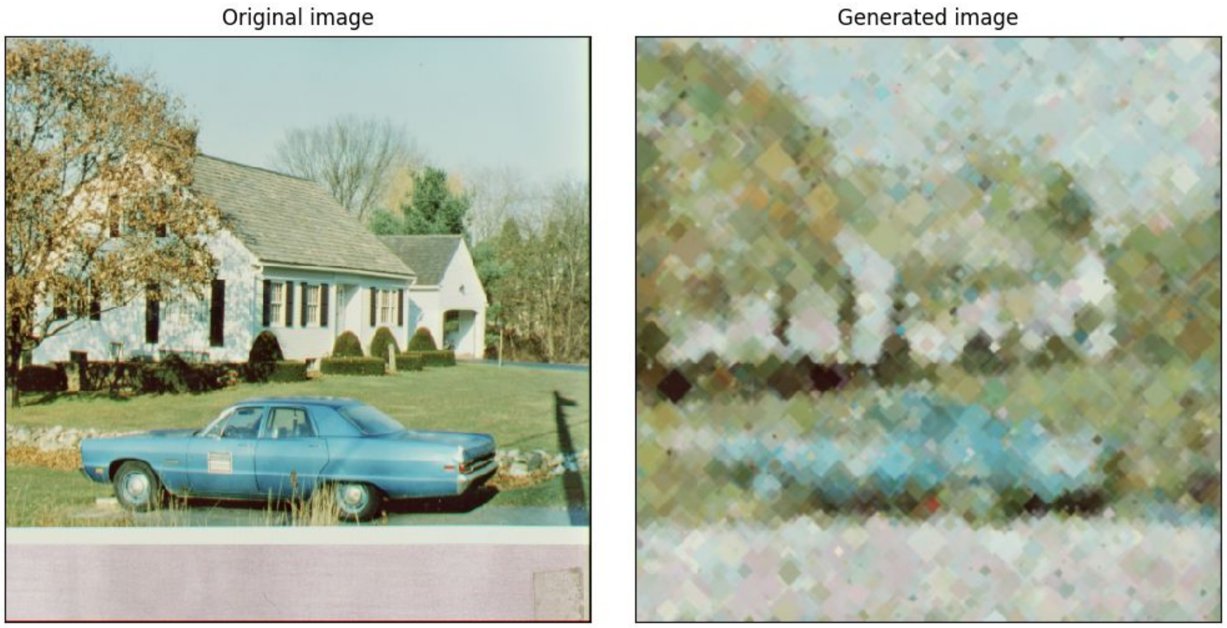
This report and source code, base images and results with splits can be found in the GitHub repository of Gleb Petrakov.

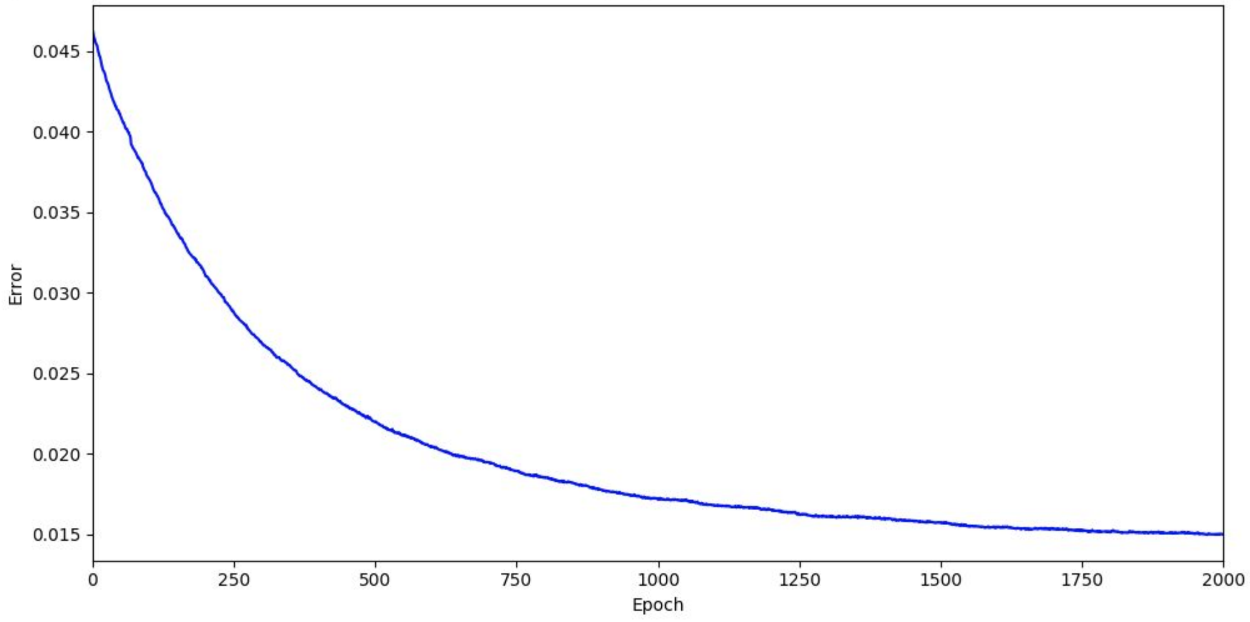
https://github.com/imajou

Example 1 – Boss Toad

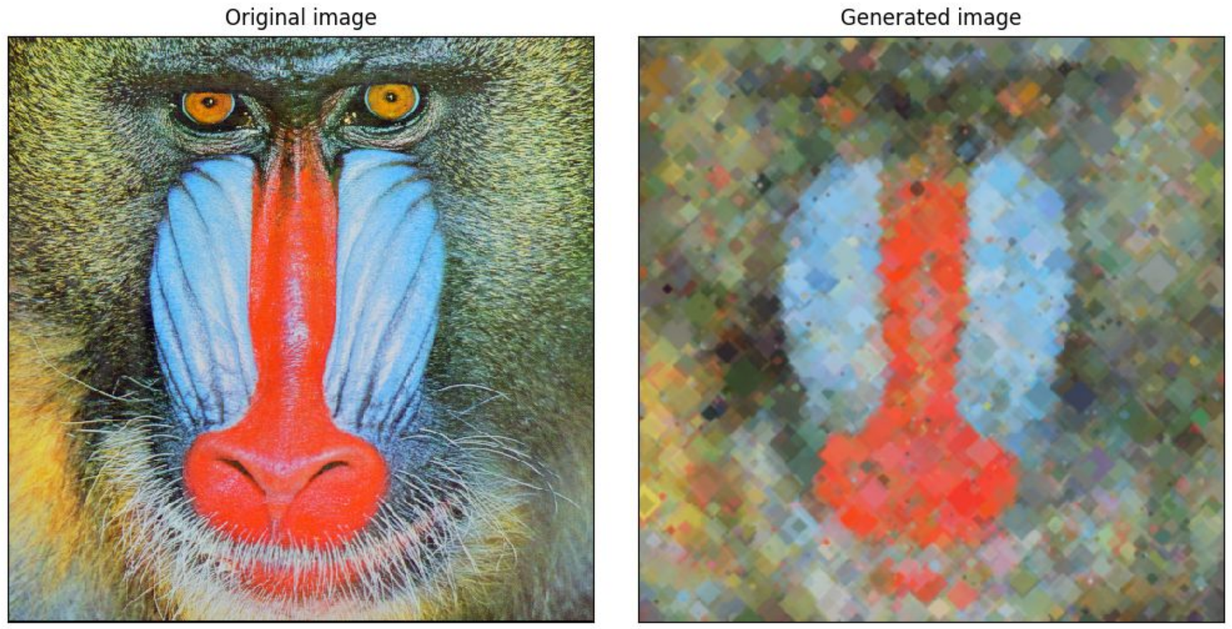


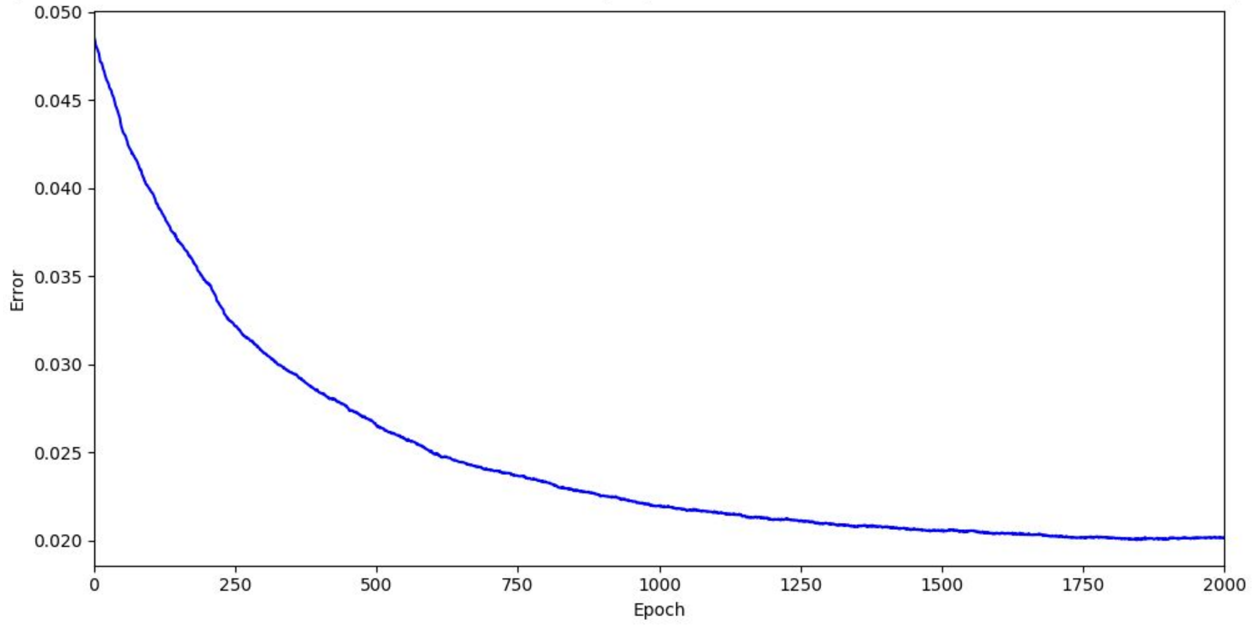
Example 2 – House





Example 3 – Primate




Error
Epoch