Reproducing Images Using Genetic Algorithm

Ivan Hladkyi, Denis Kolomyets, Tejas Anil Shah[1]

[1]Institute of Computer Science, University of Tartu

Introduction

Genetic Algorithms (GAs) are a class of algorithms which are based on the natural process of evolution and draw on the ideas of natural selection, survival of the fittest, and mutation.
Speaking broadly, these algorithms are generally applied to optimization problems where we need some form heuristic that can guide the algorithm to move towards better and better solutions.

Within Genetic Algorithms there are three broad classes.

1. **Traditional GA:** This class of GAs are based on Darwinian evolution. They can be applied to problems where the search space is too large for exhaustive search algorithms to work efficiently. The Travelling Salesman Problem is a good problem to apply the this class of algorithms to. More details about this in the next section.
2. **Interactive Selection:** For applications where the user's choice is to be taken into account, we can adapt traditional GAs to have an element of interaction. The user, or the "Intelligent Designer" can select some of the population to survive, and that makes the next generation of the population to be more similar to what the user picked.
3. **Ecosystem Simulation:** This approach is the holy grail and also a hot topic of philosophical debates. In this approach, the idea is to simulate an ecosystem and basically build a simulated world in a computer and let the world evolve according to laws of that simulation. Although the debates centre around Creationism vs. Evolution, we are more interested in the whole computational challenge and building algorithms that can efficiently model the world around us. We do this in hopes of understanding how the notion of balance in an ecosystem can be achieved.

As an application of the Traditional GA, we chose the following problem statement:
Given an input image the algorithm must reproduce the image with a given set of geometric shapes.

Design & Concepts

There are three major working components when it comes to a Traditional Genetic algorithm. These are essentially the pillars on which the Darwinian Theory of Evolution is built.

1. **Heredity:** There must be a process of transfer of traits from one generation to the next. If an organism survives long enough and reproduces, it's traits are passed on and these traits are the ones which are more successful.
2. **Variation:** This is more commonly known as mutation. In nature this occurs due to some mistakes in DNA replication, but in our algorithms we need to introduce some mechanism due to which the traits expressed change with some probability. If there were no mutations there wouldn't be a change in the population. All the generations would be identical, and therefore this would defeat the entire point of these algorithms.
3. **Selection:** This process, commonly known as **Survival of the Fittest**, is the core of the algorithm. In the algorithm there should be a mechanism due to which some individuals have the opportunity to reproduce and pass down their genes, which the rest don't. This process changes from problem to problem. In most problems and in nature, the object of evolution isn't always clear. Fortunately, in our problem we have a clear goal of using different shapes and make the image resemble a target image. So this makes it a little easier for us to design a function which calculates the fitness of the population, and selects the best individuals.

Additionally, in genetics and biology there are the concepts of *genotypes* and *phenotypes*. Essentially, genotypes are the coded information in the genes, while phenotypes are the physical expression of the genotypes.
So for instance, in our example we could think of the genotypes to be the matrices which encode the colors of pixels, while the phenotypes would be the actual pixels themselves. These correspondences change from applications to application.

As a broad overview, the algorithm works in three stages:

1. Initialize population
2. Evaluate the fitness of the individuals
3. Breed the next generation from the fittest
   (a) Kill the unfit population
   (b) Add variations/mutations to the genotype
   (c) Replace the old fit population with the new one
4. Repeat evaluation and breed steps

Fine Points

When it comes to getting genetic algorithms to work, getting the parameters right is quite tricky. For instance, in the natural world if there were too many predators and too few prey, the predators would first hunt the prey to extinction and then slowly they too end up going extinct. Finding that balance is something that is quite puzzling both when it comes to ecosystems and also our own genetic algorithms.
Clearly, the variable of the populations numbers plays an important role in how evolution pans out.
Additionally, depending on how we evaluate the fitness we can see different convergence rates. For instance, we could make the fitness function exponential in evaluating certain traits, this would make the algorithm favor these traits more. This would have the effect of the algorithm converging to the max fitness for these traits quicker when compared to fitness functions which aren't exponential.
The last component of mutation also plays a crucial role in the algorithm. As an example, if there is no mutation the algorithm will never give a population that evolves. In contrast if there is too much mutation, the population will keep changing so much and will lead to the population numbers being too unstable to sustain.
In our project we experimented with different fitness functions. We also attempted gradient descent to optimize these factors. Although we weren't very successful in implementing gradient descent, our regular genetic algorithm was able to give us good results.

Implementation Details and Results

Our implementation is in Python and mainly relies on the *opencv, numpy* libraries for image and matrix manipulations.
Since we implemented the traditional genetic algorithm, the main components of the algorithm are implemented as follows:

1. Initial population is randomly generated *(Fig 1: Right)*
2. We calculate the fitness for the population:
   We used two different fitness functions to calculate the fitness of the population. They're both distance based metrics. The general idea is that we calculate how far away the individual is from the original image. Both our fitness functions gave us similar results.
3. We select the fittest individuals
4. For the next generation we add a few mutations in the form of additional shapes to the fit individuals. We achieve this by picking shapes from our set (circles, rectangles and lines) and we place them at random locations on the image.
5. The old generation is killed
6. We continue the process for many generations *(Fig 2)*

The progression is best visualized in an animated form. So please see the visualizations folder in our repository.

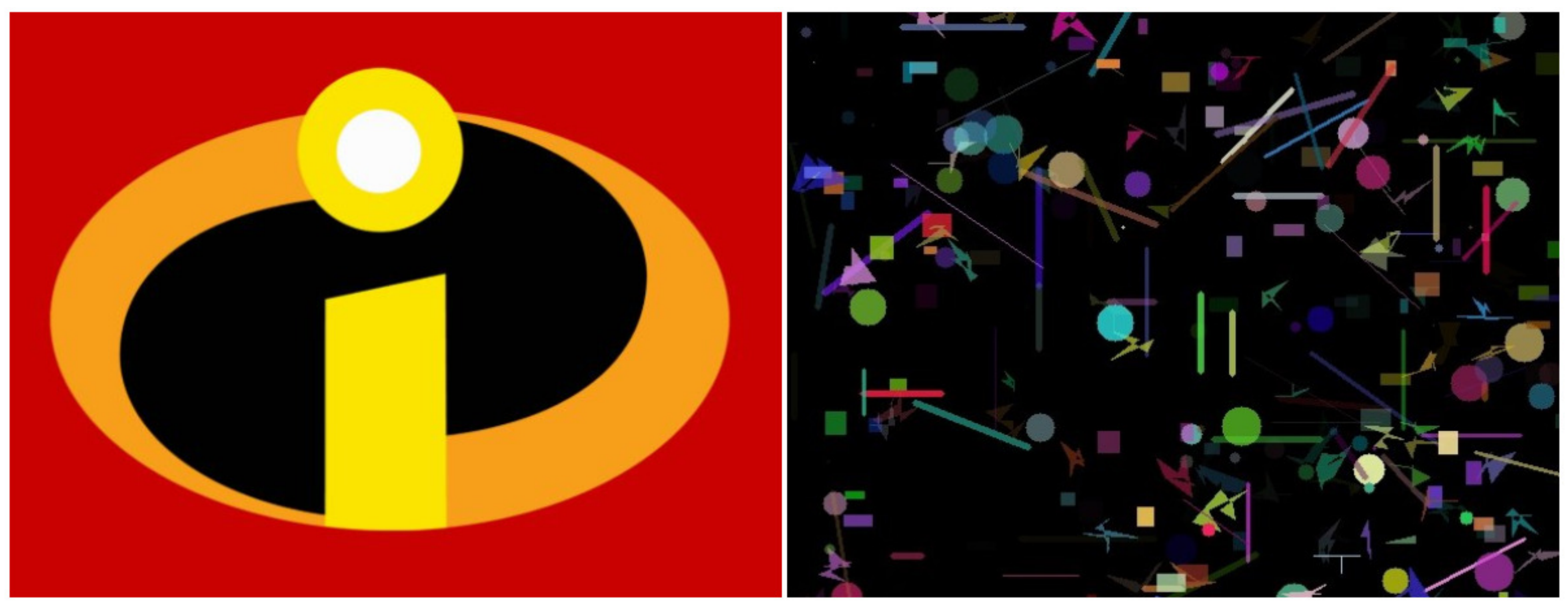
Fig 1: Original image (Left), and Generation #1 (Right)

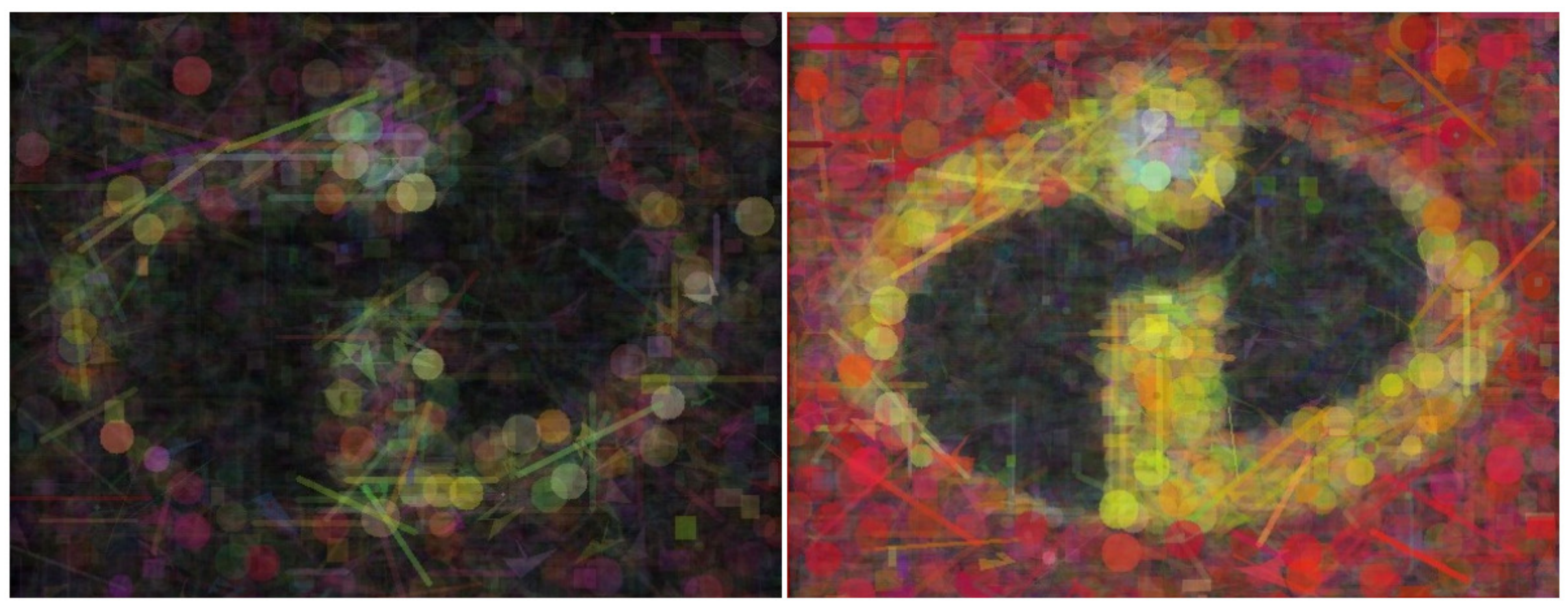
Fig 2: Generation #4500 (Left), and Generation #10000 (Right)

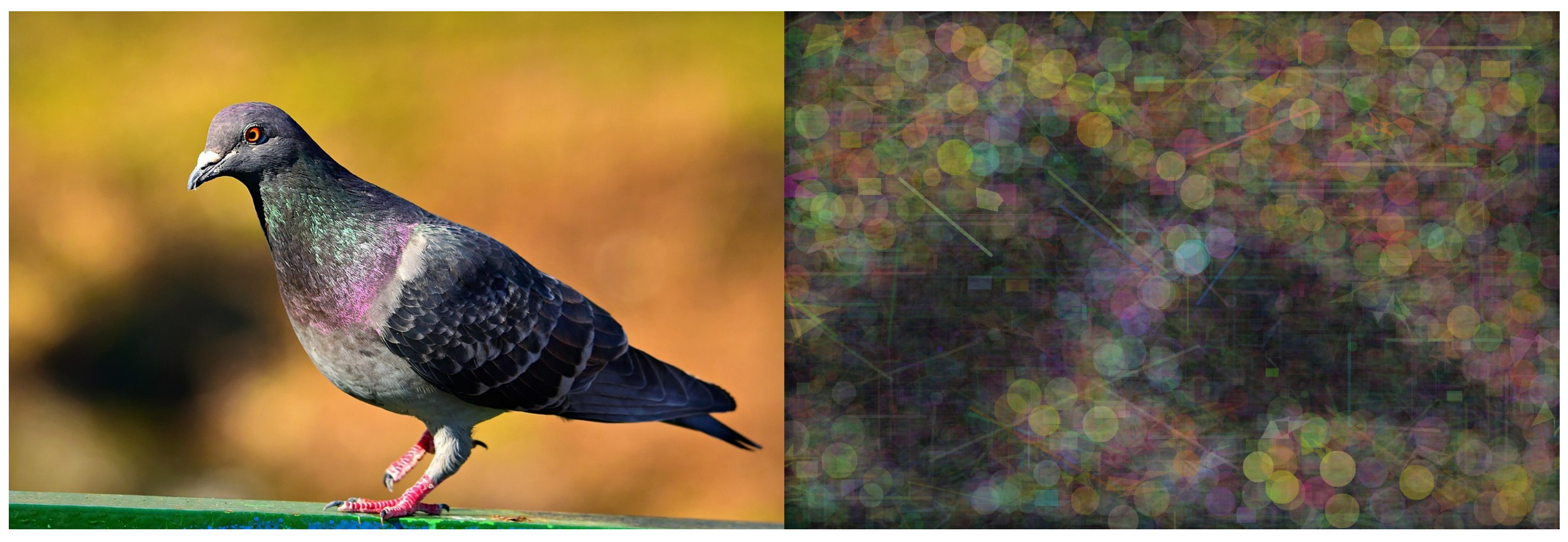
Fig 3: Original image (Left), and Generation #4500 (Right)

Conclusions

We were able to successfully implement the algorithm and understand how genetic algorithms work.

From our trials of reproducing images, we noticed that it takes a very long time (in the order of multiple hours) for complicated images like that of the pigeon *(Fig:3)* to converge to something that is recognizable. Where as for simpler images without gradients *(Fig:1)* the algorithm moves pretty quick comparatively.

We tried to build a gradient descent routine that could optimize the parameters such as the initial population, mutation rates etc, but ran into trouble with implementation.

References

1. Theory and concepts from The Nature of Code by Daniel Shiffman
2. Based on code by Azad Yasar