Nature of Code

Creating Genetic Algorithms to Produce Art

These notebooks and this summary were inspired by the chapter titled “Evolution of Code” in Daniel Shiffman’s book *Nature of Code*. I decided to attempt to use genetic algorithms to produce interesting or meaningful images, as the intersection of art and AI models is something that I’d never previously approached. The use of GAN’s constituent networks freed from the zero sum game limitation has produced some interesting images, namely in style transfer and segmentation maps turned lifelike image a la NVIDIA’s GauGAN, but genetic algorithms are novel to me, albeit their recent neglect in the wake of the advent of CNNs.

In the following sections I detail the three approaches taken using a genetic algorithm framework to produce visually compelling imagery, their results, and my concluding remarks.

The genetic framework I chose was largely based on the array of GA applications detailed in *Nature of Code*. A high-level description of these problem’s framework is as follows:

A population is initialized, where each instance contains a set of genes. This population is then evaluated for fitness as calculated by a fitness function, and a mating pool is filled using these fitnesses as indicators for potentially good mates. Two instances are sampled from this mating pool and crossed over. The crossover between the two selected instances models reproduction in that they produce a “child” instance, and depending on the crossover method, some combination of genes from both parents are passed down. Then, this child instance replaces a population member. This process is repeated for each population member such that the entire original population is replaced. This is a single generation of the instances.

Three unique approaches were made for use in the genetic framework. Each approach shares some functionality with others, but each attempts to achieve largely different outputs.

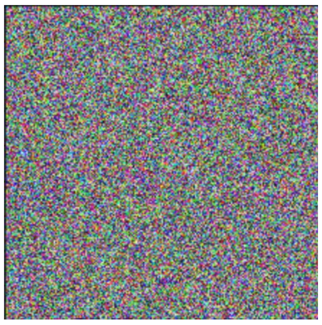
Approach 1: Reproducing a target image from random pixels

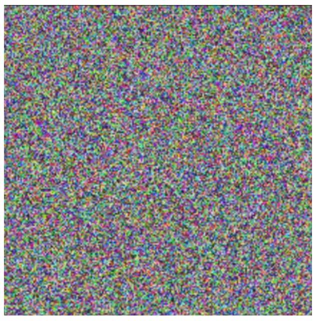
The first approach used a pixel space DNA where each member of the population was initialized with random matrices of pixel values (n x n x 3). These 3-channel images had their fitness evaluated using the Frobenius norm of the difference between a square target image (n x n x 3) and the population member's pixel matrix. These fitness values, normalized into a probability distribution, formed the mating pool for sampling without replacement. Previous implementations used a fitness function which identified the number of matching pixels in both the color and pixel space between the population member and the target, however this fitness function resulted in very low fitnesses for early generations, failing to "bootstrap" the genetic representation towards the target image.

Once two parents had been sampled, their crossover was determined using a random array of ones and zeros (n x n), which selected which pixels from each parent to pass down. Another approach here included using a pivot value to determine the size of the slice of the flattened first parent matrix to overwrite the second parent's matrix, resulting in the child matrix. This approach ended up filling the mating pool with genetic representations which were fairly similar, requiring a high mutation rate which was undesirable as images in the pool would learn some meaningful pixels from the representation and subsequently mutate and lose them.

The resulting matrix (n x n x 3) was passed to the child. Mutation evaluated a matrix (n x n) of random continuous values on the interval [0, 1) against the mutation rate, determining which pixels in the image to mutate. If mutated the pixel was replaced with random RGB values.

target


first generation

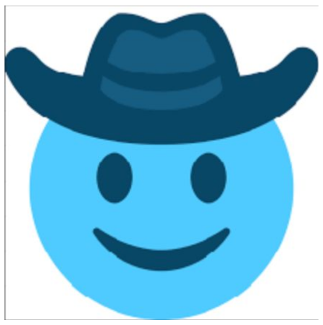

50th generation

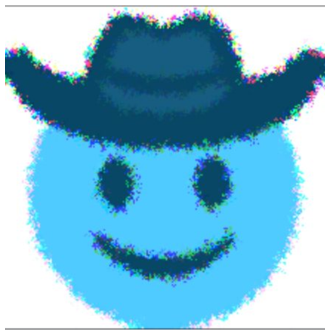
Approach 2: Modifying a given image using an arbitrary fitness function and interesting mutations

Like the first approach, the second approach also used a pixel representation for its genes. However instead of attempting to initialize every member of the population's genes with a random matrix of pixels, this approach initializes the first generation with a provided image. This image, mutated and selected via some thousands of generations within the genetic framework would hopefully produce something visually compelling.

The first generation's genes, initialized with an image, are evaluated for fitness merely by their RGB values. If the instance's genes' pixel RGB values were below a particular threshold, the instance would score higher. This encouraged genes within the pool to be bright. Once two members were sampled using a fitness-based probability distribution they were crossed over using the same method detailed in the first approach.

Mutations of the members consisted of a similar mutation evaluation method as detailed in the first approach, however once pixels were determined if they should mutate, they would spread their RGB value to one of pixels incident to them.


first generation

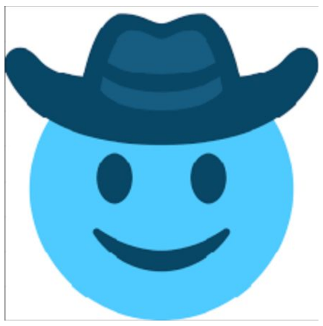

200th generation

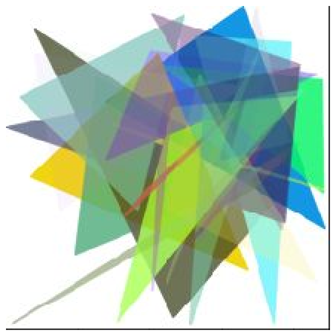
Approach 3: Representing a target image using triangles

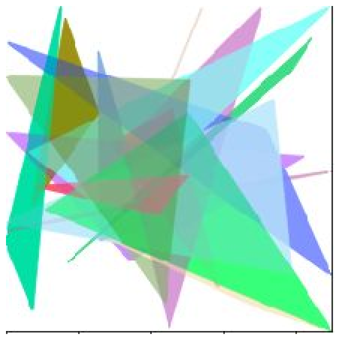
The third approach takes a novel approach to the genetic representation, moving from pixels to triangles. Each member of the first generation is initialized with a random number of triangle objects, each containing three random coordinates and a random RGB value. Fitness of each member is evaluated using the absolute difference of a pixel representation of the triangles and a target image.

The population is sampled using the fitness probability distribution. Crossover between two instances considers the ‘fitness’ of each triangle between the two, and again probabilistically samples them to produce child genes. This evaluation of fitness for triangles in the crossover method was novel; deemed necessary due to the difficulty of converging random triangles to a target image. The fitness of a triangle is considered by drawing the triangle on the currently most fit member of the population’s pixel representation, considering this new representation’s distance from the target image. The “elitist” method was also tried instead of probabilistically sampling triangles, where the most fit triangles would be passed down. However this did not improve results and reduced variation in the gene pool.

Mutations consisted of randomly moving triangle coordinates within a given interval, testing eligibility for mutation individually for each triangle of the member’s genes.


target


first generation


100th generation

Ultimately I intended this work to speak for itself in that the result would be a single, efficient algorithm modeling natural selection and producing interesting representation of pixels, however due to each approach failing to produce much outside of randomness, I have included all attempts and a report.

For the first and second approaches, the main failings were working with pixel representations for genes. Using a HPC improved some of the computation times, however there are surprisingly quite a few hyperparameters that require tweaking, and the notebook interface serves a much better environment for visually examining gene representations.

The second approach clearly could use more interesting mutations and fitnesses. The populations provided from this approach, albeit simple, already started to create interesting images.

The third approach holds the most promise-- triangles are computationally much less expensive than individual pixels, and OpenCV's libraries for drawing convex polygons are relatively speedy. The introduction of alpha blending added more computational complexity, however alpha blending solved problems with triangles overlapping.

Finally, the obvious should be noted, Python is not particularly well suited for this task. If perhaps it had been built in a different, more efficient language, I would have more interesting results.