Genetic Programming for Non-Photorealistic Image Generation

Sebastian Charmot and **Daniel Cowan**
{secharmot,dacowan}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

In this paper, we investigate the effectiveness of using a genetic algorithm to generate non-photorealistic images using randomly-generated, randomly-colored shapes on a blank canvas. Contrary to previous work the same problem that uses a single individual and only mutation, our experiments use multiple individuals and crossover functions. Our findings indicate that the methods we implemented were fairly effective, but the color in some portions of the generated image significantly differed from the target image. We attributed this error to our fitness function which used the difference in RGB values as opposed to a metric that more accurately models a perceived difference in color by the human eye.

1 Introduction

A genetic algorithm is comprised of several general steps that resemble genetic reproduction in the biological sense. First, it creates a random population. Then, it selects members using natural selection to reproduce and place their offspring in the future population. This process is then done in an iterative manner with the expectation that the individuals of each generation are improving. The goal of this paper is to create a genetic algorithm that generates a non-photo-realistic version of a target image.

In this paper, we followed many traditional genetic programming techniques including: crossover, mutation, and tournament selection (Oliveira et al. 2015). Since a genetic algorithm has several parameters to fine tune, we experimented with a variety of combinations and gathered data to measure how effective each variation of parameters was. These experiments allowed us to quantitatively evaluate which individual parameters had the most success in generating a non-photo-realistic version of an image.

The rest of the paper will be structured in the following format. In the next section, background, we will describe the relevant terminology and parameters of our genetic algorithm. Then, we will describe how we tested and evaluated the different combinations of parameters in the experiments section. From there we analyze the results of the experiments, and end on the conclusions that can be drawn from our results.

2 Background

Representation of Individuals

The first part of creating a genetic algorithm is determining a representation of the individuals in each population. In our approach, each individual is represented as an image. In practice, this means that each individual is an array of RGB values. It is notable that this is different from what we found when doing our literature search on academic papers that solved a similar problem. Many researchers approached the problem by starting with a single blank 'canvas' and iteratively adding pieces onto it. For example, one researcher tested different painting strokes in their genetic algorithm to create a non-photo-realistic image (Wu 2018). Another researcher used the same blank canvas model and iteratively added triangles to the canvas in order to generate a non-photo-realistic image of the Mona Lisa (Johansson 2008). Johansson referred to this process as single individual genetic evolution where improvements in fitness happen through mutation only. This means that those researchers used only a single individual and did not use a crossover function in their algorithms. We wanted to test if our approach of using complete images as our individuals and performing crossover on two parents could create results comparable to those of the researchers whose genetic algorithm only improved upon a single individual.

We used two methods to generate our random individuals. The first method involved choosing a random color as the background and then adding a random number of polygons of random colors onto it. We refer to this method as Generate Random Individual 1. An example of what this looks like can be seen below in Figure 1.

Figure 1: Example of Generate Random Individual 1

The second method we used to generate random individ-

uals was by generating a random RGB value for each individual pixel in an individual. The inspiration for the second method of generating random individuals came from a researcher who was able to reproduce an image using this method (Gad 2019). Interestingly, the final results of Gad resembled the original image but with heavy noise. An example of what this kind of individual looks like can be seen below in Figure 2.

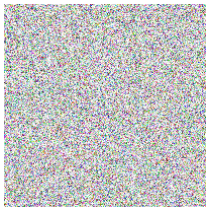


Figure 2: Example of Generate Random Individual 2

The method that we use to generate the random individuals of our first population is important as it can help the algorithm converge faster. In a genetic algorithm, all future populations are derived from the first. Therefore, a poor method of generating random individuals could provide a weak foundation for the future generations to improve off of.

Fitness

Each individual is assigned a fitness measure which determines how similar it is to the target image. This metric determines how likely it is for an individual to crossover to the next generation. Each individual's fitness is measured as the sum of the absolute differences in RGB values between an individual and the target image. Note that this choice of fitness function means that lower fitness values are more desirable. Since there are about 50,000 pixels in the images we used for training, we expect our fitness values to be quite large.

Tournaments

To select individuals for crossover, we run tournaments with a sample size of eight individuals from our current population and select a single 'winners' from each tournament. Once we have randomly selected individuals for a tournament, we sort them by their fitness values such that fitter individuals are at the front of the list. A weighted probability list is then used to determine two winners for the tournament. This weighted probability list is weighted by the formula $p * (1 - p)^i$ where p is the starting tournament probability and i is the index of the individual in the list. It is not always in the best interest to select the most fit individuals as this can decrease the diversity of the individuals in a population. Therefore, the goal of our weighted probability list is to select the better individuals most of the time. In theory this should allow us to both improve the fitness values of individuals and maintain a diverse future population. To perform crossover, we select two parents by calling the tournament function twice.

Crossover

The goal of crossover is to create an individual with a better fitness than its two parents. For our genetic algorithm, we actually implemented three different kinds of crossover functions. To illustrate each crossover function, we will show the results on what would happen if the parents were the side by side images seperated by white space below in Figure 3.

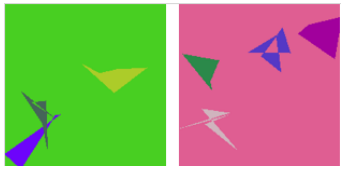


Figure 3: Parents for Crossover Function Demonstrations

1. For Crossover 1, we convert the opacity of each image to 0.5 and then overlay them. This can also be thought of as averaging the two parents across their RGB values to produce their offspring. We can see the result of Crossover 1 on the two parents from Figure 3 in Figure 4 below.

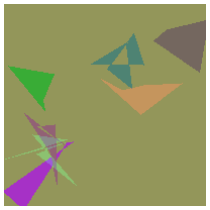


Figure 4: Child after Crossover 1

2. For Crossover 2, we select a random row of the child image where everything prior to that row is from the first parent and everything after is from the second parent. This leaves large portions of both parents relatively untouched in the child. We can see the result of Crossover 2 on the two parents from Figure 3 in Figure 5 below.

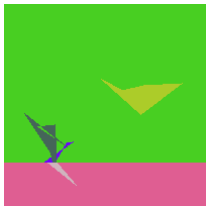


Figure 5: Child after Crossover 2

3. For Crossover 3, every pixel in the child is chosen to either be from parent 1 or parent 2 at random. As opposed to the previous two crossover functions, Crossover 3 is not as 'smooth'. We can see the result of Crossover 3 on the two parents from Figure 3 in Figure 6 below.

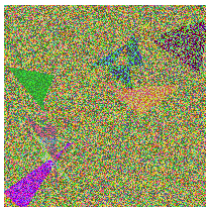


Figure 6: Child after Crossover 3

Mutations

We use mutations to increase diversity within each generation's population. A mutation could theoretically generate enough noise to push an individual out of a local minima when the fitness values of the population are not improving. It is important to consider the magnitude of a mutation operation on an individual. This is because a large perturbation has the potential to greatly improve or reduce the fitness of an individual whereas a smaller perturbation has less variance but less potential for large improvements. As a result, we created two different mutation functions for our genetic algorithm where one has a larger disturbance than the other.

1. For Mutation 1, we add a single polygon of random color onto an individual. In Figure 7 below, we can see a side by side comparison of the effect of Mutation 1. On the left is the original individual and on the right is the individual after a Mutation 1 operation.

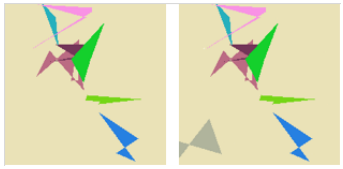


Figure 7: Side by Side Comparison of Mutate 1

It can be seen that Mutation 1 has quite a discernible effect on the individual, which is likely to change its fitness by a noticeable amount.

2. For Mutation 2, we select ten of the pixels at random from an individual and increment one of their RGB values by ten. In Figure 8 below, we can see a side by side comparison of the effect of Mutation 1. On the left is the original individual and on the right is the individual after a Mutation 1 operation.

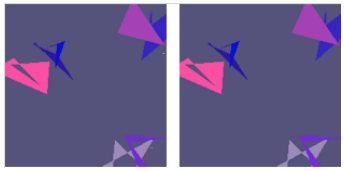


Figure 8: Side by Side Comparison of Mutate 2

The effect of Mutation 2 is much more subtle than Mutation 1. This means that Mutate 2 does not impact the Individual as much as Mutate 1, which would cause smaller variations in fitness.

Genetic Algorithm

With the components discussed in this section, we have the elements necessary to create a genetic algorithm. Our genetic algorithm starts by generating an initial population of random individuals and assigning a fitness value to each of them. From there, we run tournaments to find individuals to perform crossover on. In the process of creating offspring, there is also a chance mutation will be applied to a random individual and placed in the future population. This process of generating new individuals is repeated until there is a new generation of individuals that is the same size as the last population. Our genetic algorithm is set to run for a set amount of generations.

3 Experiments

The primary purpose of our experiments is to determine which combination of parameters within the genetic algorithm will create the fittest individual. To do so, we test different combinations of parameters and evaluate how well they perform. Table 1 below summarizes the first round of variations we will be testing.

Table 1: Experimental Variations

Algorithm Component	Variations
Generate Random Individual	Method 1 Method 2
Crossover Function	Method 1 Method 2 Method 3 Hybrid of all 3
Mutate Function	Method 1 Method 2

We will use a graph of the fittest individual per epoch to evaluate each of the variations. Then once we have determined the best variation, we will vary the population size and mutation rate to see if we can fine tune the genetic algorithm even further. Finally, once we have finalized the parameters, we will run the genetic algorithm on a target image and display the results for a visual test.

Each experimental run was run on the same target image for 10000 generations with a population size of 50, and the fitness of the fittest individual was recorded every 10 generations. If we had unlimited time resources, we would have let our algorithm run until zero improvement is made from generation to generation, but from initial tests, we saw our algorithm did not significantly benefit from training on more than 10000 generations. This was the reasoning behind limiting the number of generations for parameter tuning at 10000. All tests were performed on the image shown in Figure 9.

Figure 9: Target Image for Experimentation

4 Results

Random Individual Generation

Of the two methods we implemented to generate random individuals, Generate Random Individual 1 performed far better. After 10000 generations, the individuals from Generate Random Individual only had a marginal improvement in fitness even compared to the first completely random population. We believe that this is a result of the individuals having too much 'similar" randomness. Although each individual pixel is random, the individuals essentially look the same visually. It is difficult to imagine how a series of individuals from Generate Random Individual 2 could turn into the target image, therefore it makes sense that their offspring have difficulty in improving their fitness. Generate Random Individual 1, however, was able to produce better results with a noteable improvement in fitness over each generation. Therefore, we decided to only use Generate Random Individual 1 for the rest of our experiments.

Mutate Function

When comparing our two mutate functions, Mutate 1 yielded notably better results than Mutate 2. This is not what we originally expected because the change that Mutate 1 performs on an individual is much larger than Mutate 2 on average. Adding a completely new shape with a random color should create more disturbance than simply increasing the values of 10 pixels. The difference in performance between the two mutate functions was significant enough that we limited future tests to simply using Mutate 2.

Crossover Function

We independently tested the three crossover functions with all other variables constant. As a result of our previous experiments, we used Generate Random Individual 1 to create the first population of individuals. Our results indicate that Crossover 2 performed better than the other two crossover functions, as shown in Figure 10.

Crossover 2 is the only crossover function we implemented that keeps large portions of each parent intact. This could be a factor as to why it performed better than the other two crossover functions. It is interesting to note the similarity in performance of Crossover 1 and Crossover 2. This could be an indication that the effect of averaging two images versus taking a pixel at random from either parent is similar.

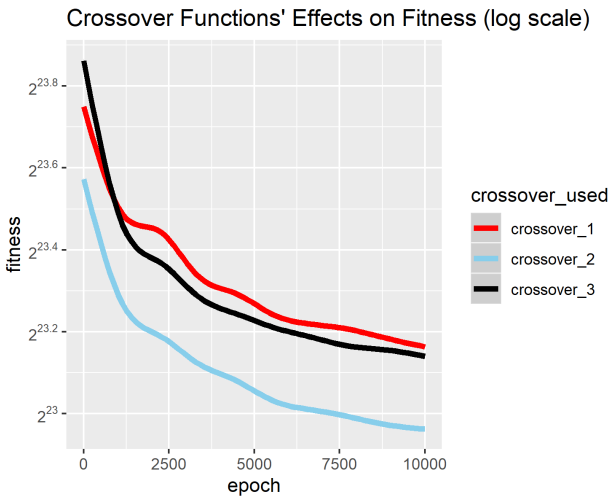


Figure 10: Performance of the three crossover functions

Population Size

As specified in the Experiments section, all previous tests were done with a population size of 50. We hypothesize that a larger population size would improve the algorithm since a larger population size creates a larger sample space of possible combinations and more diversity. As expected, the algorithm benefited from larger population sizes, which can be seen in Figure 11.

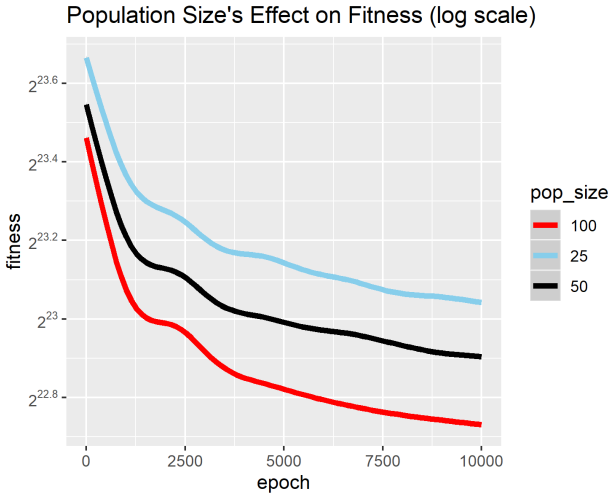


Figure 11: Larger population sizes yielded better results.

Unfortunately, population sizes larger than 50 were at the limits of our computational power and required an exponential amount of time to run. Even though our results seem to indicate a larger population size benefited our algorithm, we still chose to use 50 due to the limitations of our computational power.

Hybrid Variations

We also wanted to test whether a hybrid combination of parameters in our genetic algorithm would yield better results than a non-hybrid version. Based on our previous results, we wanted to test different weightings for the four genetic algorithm functions: Crossover 2, Crossover 3, Mutate 1, and Generate Random Individual 1. We decided on the combination of Crossover 2 and Crossover 3 specifically since Crossover 1 performed the worse. We also hypothesized that Crossover 3 could be effective in certain situations. The results of our tests are shown in Figure 12.

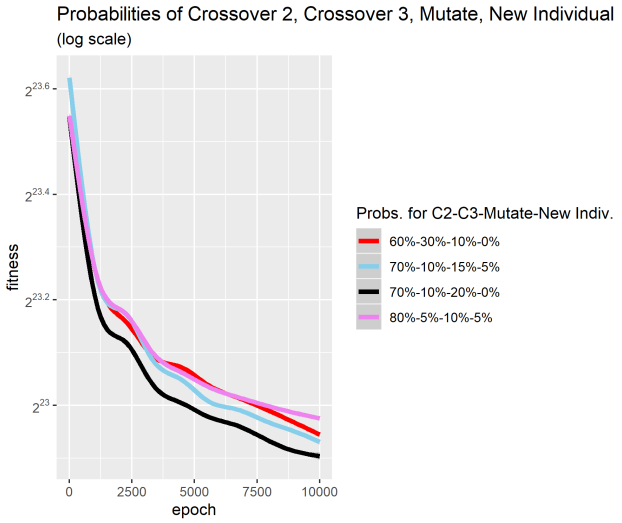
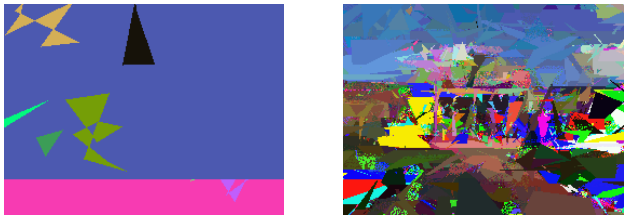


Figure 12: Results from splits of genetic functions.

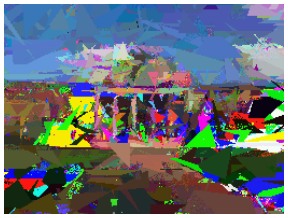
Note that the sum of the percentages within the 4 variations sum to 100%. This is because each component is a method of creating an offspring. Since we only expected Crossover 3 to be useful in a few situations, we kept the rate of use for Crossover 3 low in comparison to Crossover 2. New Individual refers to the percentage chance of calling Generate Random Individual 1 and adding a new random individual to the next population. We kept the rate for Generate Random Individual 1 low as well since we also expected it to be helpful only a small percentage of the time. It could be helpful in maintaining diversity but it is probably better to keep individuals that have been improving since the first generation in the future populations instead.

Our results indicated that the combination of: Crossover 2 (70%), Crossover 3 (10%), Mutate 1 (20%), and Generate Random Individual 1 (0%) performed the best in our hybrid tests. It is interesting that not generating new individuals after the first generation was best. This is an indication that it is best to only keep individuals that have been improving since the first generation in future populations. We can also see in Figure 12 that the best hybrid model did perform better than simply using Crossover 2 with a 5% Mutate 1 rate. This seems to support our hypothesis that Crossover 3 can be useful in certain situations even if it is not the better crossover function on its own.

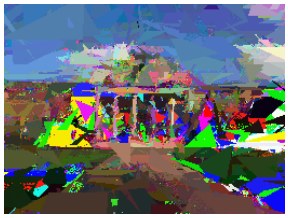


((a)) Most fit individual after 1 generation

((b)) Most fit individual after 10000 generations



((c)) Most fit individual after 20000 generations



((d)) Most fit individual after 30000 generations

Figure 13: Progression of generation of image shown in Figure 9.

Final Model Results

After all data was collected, we decided on a genetic algorithm with the following parameters:

- Crossover 2 70% of the time
- Crossover 3 10% of the time
- Mutate 1 20% of the time
- Method 1 for generating a random individual
- Population size 50

We wanted to visually evaluate how our best model would perform on the target image shown in Figure 9, which was the image we used during the parameter tuning process. The evolution process is shown in Figure 13.

First, we can see that the genetic algorithm was able to recreate several distinct parts of the target image such as the pillars, the brick path, the clouds, and the lawn. With that said, there are some areas where the algorithm performed poorly such as the neon green and white patches that are not in the target image. It seems that the algorithm had trouble learning how to accurately fill in areas with darker colors. Other than this issue, we were quite impressed with the performance of the considering it was able to recreate many of the details from the target image.

As a second test, we wanted to see how our algorithm performed on a photo of the Mona Lisa, shown in Figure 14. As stated in Background, our work was partially influenced by the work of Johansson, who tested their algorithm on the Mona Lisa. The results of our algorithm with the Mona Lisa as the target image are shown in Figure 15.

Again, we see the same problem presented in Figure 13: the algorithm has great difficulty evolving properly to darker colors. To investigate this issue, we looked into our choice of fitness function. For the experiment, we chose to measure fitness by summing the absolute differences between each

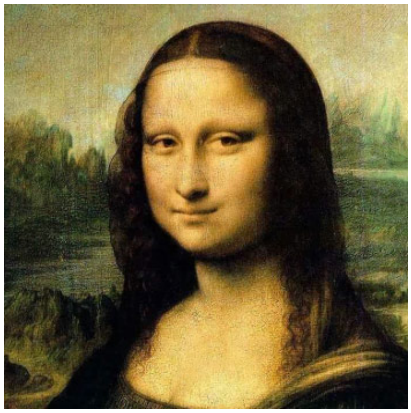


Figure 14: Mona Lisa Target Image

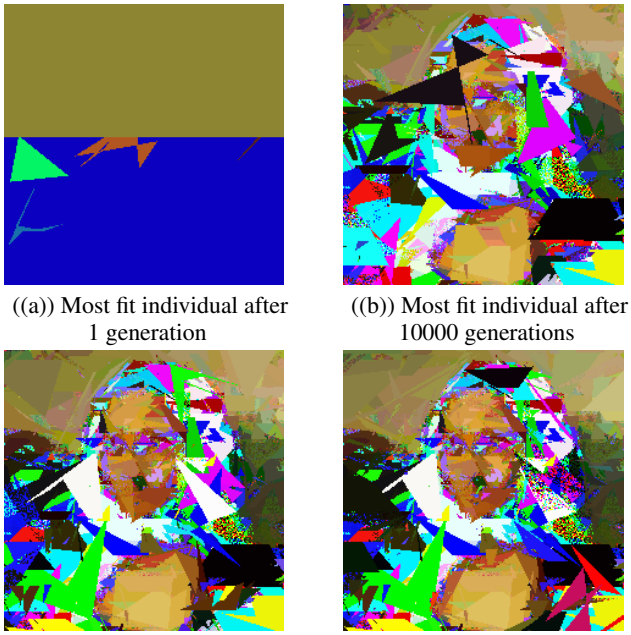
pixel's RGBA value in the target image and the current individual. The issue is that this measurement of color difference doesn't accurately represent a human's perception of color (Melgosa 1994). In hindsight, then, we could have used a fitness function that better relates distances between colors to the differences in their perception by humans (Mokrzycki and Tatol 2011). We ran a small-scale test with one of these fitness functions, specifically the CIE 1976 Delta-E function, which calculates the difference between two colors in a way that models human perception better. The test was run with Figure 14 as the target image, and results are shown in Figure 16.

It is evident that the previous issues our algorithm had with evolving properly to darker portions of the image are less prominent. However, it was unable to match the level of detail our algorithm was able to achieve with our original fitness function – especially looking at the facial features of the Mona Lisa – but we are fairly certain that this new fitness function has the potential to achieve similar detail given enough time to evolve. Clearly, then, simply using differences in RGBA values might not have been the most effective way of quantifying how close an individual was to the target image.

Regardless of the issues presented, we believe that our genetic algorithm was effective in generating non-photorealistic images from a given target image. We did not yield the optimal result, which would have been a generated images that had no significant deformities in color or shape, but our algorithm still showed impressive detail and color accuracy.

5 Conclusions

In this paper, we investigated a method for generating non-photorealistic images using a genetic algorithm. We found that our specific genetic algorithm was generally able to reproduce target images in a non-photorealistic manner; however, the algorithm had difficulty representing areas in the target image that were very dark in color. We determined this issue to be the fault of our chosen fitness function, which did not effectively model color value differences as perceived by the human eye. Regardless, it does seem that



((a)) Most fit individual after 1 generation

((b)) Most fit individual after 10000 generations

((c)) Most fit individual after 20000 generations

((d)) Most fit individual after 30000 generations

Figure 15: Progression of generation of image shown in Figure 14.

((a)) Most fit individual after 1 generation

((b)) Most fit individual after 1000 generations

((c)) Most fit individual after 3000 generations

((d)) Most fit individual after 5000 generations

Figure 16: Progression of generation of image shown in Figure 14 using CIE 1976 Delta-E as fitness function.

our method was effective in creating non-photorealistic images given a target image. Future work could reproduce this study with a new fitness function, such as the one we used in our small scale test shown in Figure 16. A hybrid version would be interesting too as it might be able to preserve both detail and dark colors. Other work could also vary the shapes used to generate the non-photorealistic images, such as using only one type of shape.

6 Contributions

Daniel Cowan (DC) wrote the skeleton for the classes that ran the genetic algorithm and helped with implementation of the genetic algorithm itself. DC also performed data collection and wrote the Results and Conclusion sections. Sebastian Charmot (SC) took the lead in implementing the genetic algorithm and also prepared the code for data collection by setting up a Jupyter Notebook. SC wrote the Introduction, Background, and Experiments sections, as well.

References

Gad, A. 2019. Reproducing images using a genetic algorithm with python. `https://heartbeat.fritz.ai/reproducing-images-using-a-genetic-algorithm-with-python-91fc701ff84`. Retrieved on Dec. 16, 2020.

Johansson, R. 2008. Genetic programming: Evolution of mona lisa. `https://rogerjohansson.blog/2008/12/07/genetic-programming-evolution-of-mona-lisa/`. Retrieved on Dec. 14, 2020.

Melgosa, M. 1994. Uniformity of color metrics tested with a color-difference tolerance dataset. *Applied Optics, Volume 33, Issue 34, December 1, 1994, pp.8069-8077* 33(34):8069–8077.

Mokrzycki, W., and Tatol, M. 2011. Color difference delta e - a survey. *Machine Graphics and Vision* 20:383–411.

Oliveira, L. O. V.; Otero, F. E.; Pappa, G. L.; and Albinati, J. 2015. Sequential symbolic regression with genetic programming. In *Genetic and Evolutionary Computation*. Springer International Publishing. 73–90.

Wu, T. 2018. Image-guided rendering with an evolutionary algorithm based on cloud model. *Computational Intelligence and Neuroscience* 2018(3).