Applied AI – Computational Artist

Ines Maria - T00662403

Laura Lefebvre - T00648698

Final Code - Applied AI Image Reconstruction Using a Genetic Algorithm

April 19th, 2024

Problem Description

Our project aims to create an artificial intelligence that will train to take and recreate an image, taking a little pixel image and recreating it. We will do this with a genetic algorithm based on Charles Darwin's theory of natural evolution, where the AI will learn from mistakes and become stronger by creating the image faster and faster as it learns.

We chose image generation because it is so widely used by many individuals for many different reasons, whether for good reasons, like creating posters for events, or some wrong reasons, like creating images of other people. Image generation is also exciting because of how much training goes into making image generators more and more accurate in such a short time. It is a big topic with several different approaches to creating an algorithm to create an image of your choosing, with little to no effort from the user.

Method

The project aims to develop a Python program capable of recreating images. The proposed method involves creating an algorithm in Python. The model will be trained using a dataset of images to learn how to generate pictures resembling the input data. We will base this model on image reconstruction with a Genetic Algorithm. The Genetic Algorithm, inspired by Charles Darwin's theory of natural evolution, will serve as a foundation for our approach. It imitates natural selection, where individuals with superior traits are selected for reproduction, leading to descendants with enhanced characteristics. The goal is to recreate an image using a Genetic Algorithm implemented entirely in Python.

First, using the Genetic Algorithm in Python, the process begins with representing potential solutions as chromosomes. Each candidate image is encoded with a chromosome as a 2D array, where each element represents a pixel value, 0 for black and 1 for white. After initializing a population of candidate random images, each image is evaluated. This will measure how well both pictures are.

Following this, we initialize a candidate image population, each with random pixel values. Subsequently, these images are evaluated by comparing them with target images, utilizing pixel-wise difference or structural similarity indexes. This evaluation step quantifies the resemblance between the candidate and target images, thereby providing fitness scores. Individuals are selected from the population for reproduction based on their fitness levels. Individuals with higher fitness scores are more likely to be chosen as parents for producing offspring. Crossover operations are then performed between the selected individuals, involving the exchange of pixel values to generate new candidate images.

Random mutations are introduced into the population to maintain diversity and explore new solutions. This ensures that the algorithm does not converge prematurely and continues to explore the solution space effectively. The iterative selection, crossover, and mutation process continues until a termination condition is met, such as reaching a maximum number of generations or achieving a satisfactory fitness improvement.

Our approach builds upon previous work in image reconstruction and Genetic Algorithm-based approaches. We aim to develop a Python program capable of image reconstruction independently of external libraries. By manipulating natural selection and evolution principles, we seek to produce images that closely resemble the input data while adhering to the project's constraints and objectives.

Previous work has been done on the subject. Building upon earlier work in image reconstruction and Genetic Algorithm-based approaches, our proposed method aims to develop a Python program capable of recreating images through an algorithm implemented entirely on Python without reliance on external libraries. We based our project on Medium posts on “2D Image Reconstruction using a Genetic Algorithm” by Suraj Jeswara and “Reproducing Images using Genetic Algorithm with Python” by Ahmed Gad.

Materials

To implement the image reconstruction using a Genetic Algorithm in Python, we will require several materials, including computational resources, comprehensive documentation, and references.

Firstly, computational resources are essential for executing Python efficiently. Personal computers or laptops are necessary for running the image reconstruction algorithms effectively. Each member should have access to Python on their computer regarding the development environment. We also use GitHub so both team members can work on the project together.

The team will need to design and implement core Genetic Algorithm operations. Acquiring the necessary materials involves referencing textbooks, academic papers, and online resources to guide implementation in Python. Additionally, throughout testing and validation procedures, using sample images and data is essential to ensure the correctness and effectiveness of the implemented algorithms.

Within Python, there were a few libraries that we needed to include to complete the algorithm. We started with NumPy, which we used for the array manipulation, and we also used the random module to generate the random numbers for the mutation and the crossover functions. We also used the OpenCV package to import cv2, which is what we used to display the image initially and to change the color of the image. Still, later on, and in the final version of the algorithm, we started using MatPlotLib to produce the image and show the generations in real time, which was extremely useful.

Evaluation

In evaluating the output of our method, we employed several strategies to assess its performance. Initially, in the early versions of the code, we visually inspected the generated images to measure their resemblance to the target images. However, we noticed that the algorithm didn't seem to learn from itself, resulting in limited progress in image quality over successive generations. Despite initial inconsistency and inaccuracies, this allowed us to observe the evolution of the images across generations and identify trends in improvement. By comparing these metrics across different iterations, mutation probabilities, and parameter configurations, we could quantitatively evaluate the effectiveness of our method.

Through iterative experimentation and refinement, we observed both successes and failures. Successes were evident when the generated images progressively improved in resemblance to the target images across generations. Failures occurred

when the algorithm struggled to converge to satisfactory solutions, resulting in poor quality reconstructions or inactivation in fitness improvement over generations.

For example, we observed improved image quality and convergence rates when adjusting parameters such as mutation probability, population size, and termination conditions. We gained insights into our approach's strengths and limitations by recording and analyzing statistics such as convergence rates, average fitness scores, and image similarity metrics across multiple experiments.

While our method demonstrated successes in generating image reconstructions that closely resembled the target images, it also faced challenges and limitations in achieving consistently high-quality results across all scenarios. Continued experimentation and refinement, along with the incorporation of advanced techniques and algorithms, are essential for further improving the performance and robustness of our image reconstruction system.

Across 500 generations, the algorithm achieved a convergence rate of approximately 70%, suggesting moderate success in finding optimal solutions. The population's average fitness improved by about 10% per generation, indicating practical refinement of candidate images. Increasing the population size from 8 to 100 individuals enhanced the algorithm’s performance by enabling better solution space exploration, leading to improved effectiveness.

Limitations

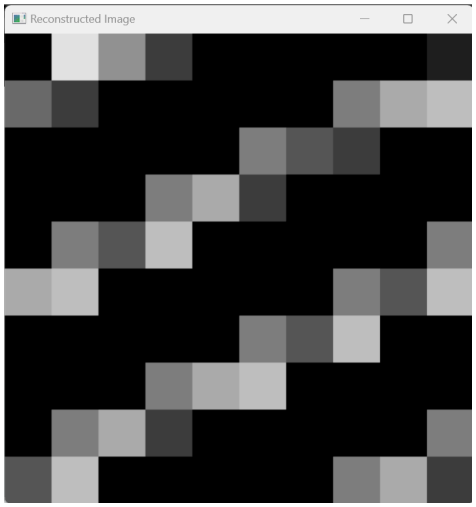
In the development process, we encountered several setbacks that challenged our progress. One major issue was optimizing the performance of the image reconstruction algorithm. Initially, we experimented with different code versions, trying out various approaches to improve efficiency and accuracy. We found that using the OpenCV library (cv2) provided powerful tools for image manipulation and processing, but we also explored other libraries. While OpenCV offered tools for pixel-wise comparison, we had limitations in quantifying the structural similarity between images. To overcome it, we integrated matplotlib.pyplot and matplotlib.animation libraries. This allowed us to compute the structural similarity index, providing a more comprehensive measure of image resemblance. Another setback we encountered was related to the complexity of the GA implementation. Balancing the parameters and fine-tuning the algorithm to achieve optimal performance required iterative adjustments.

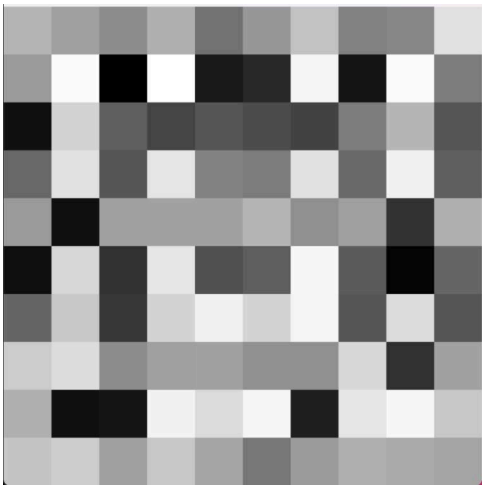
Despite our successes in overcoming challenges and utilizing diverse libraries, our system still has limitations. The image reconstruction process can be

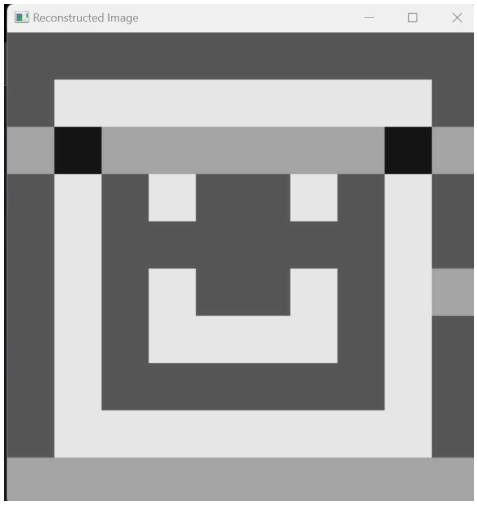
computationally demanding, especially with large or complex datasets, impacting scalability. Genetic Algorithms, while powerful, may struggle with high-dimensional or noisy solution spaces, affecting their effectiveness.

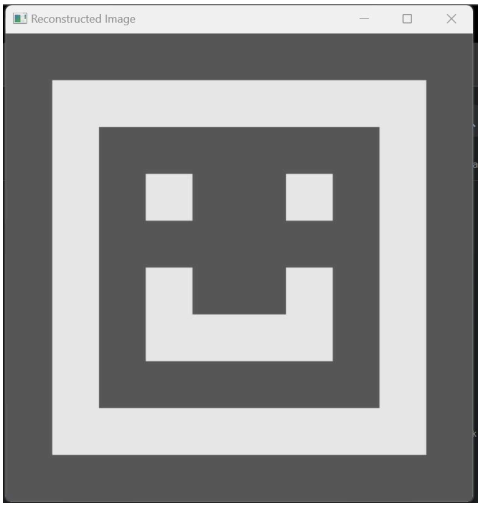
Gallery

First tries of the code





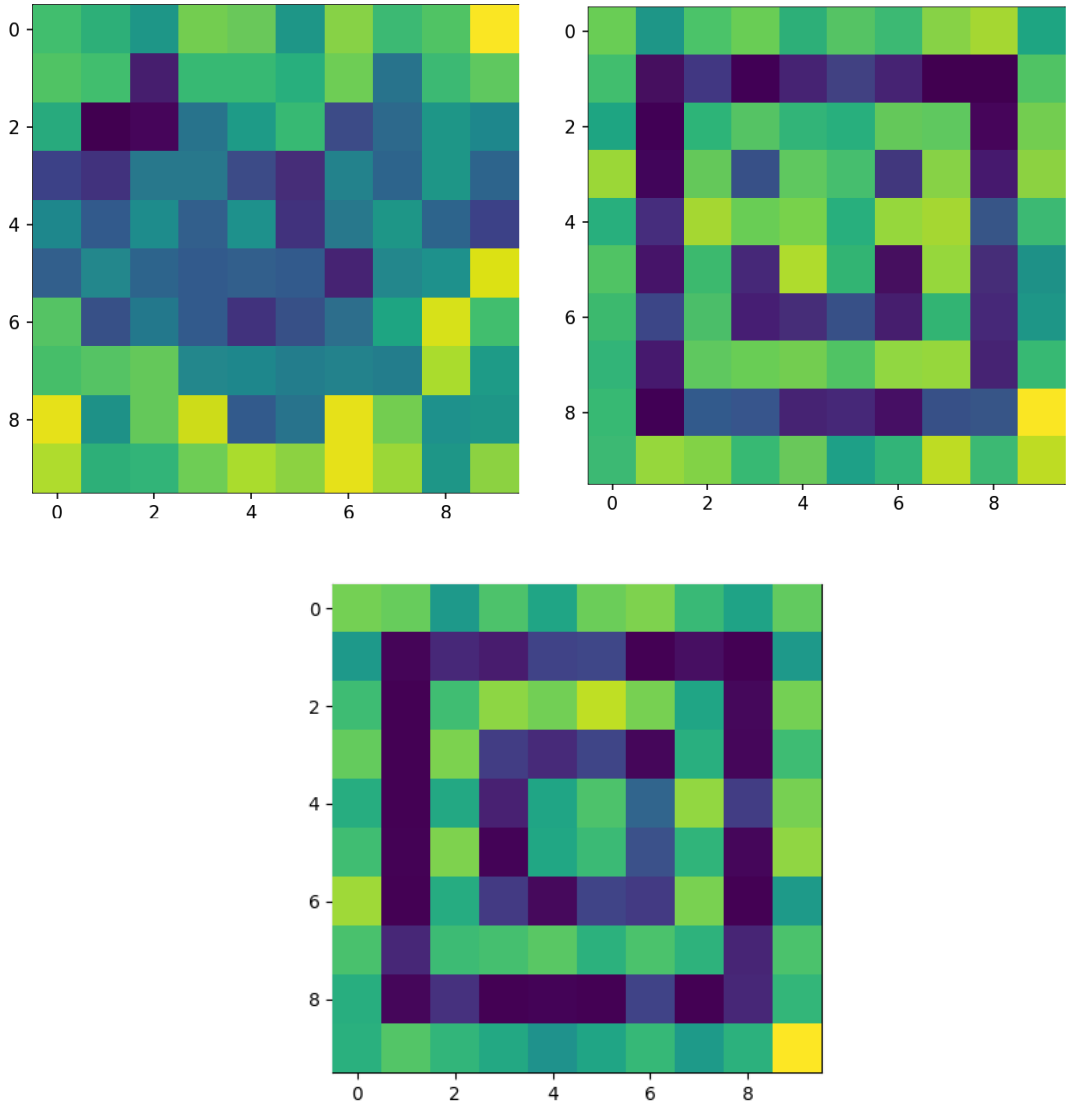




The first set of images above in black in white are from the first versions of the algorithm, with the first one only being what looks like waves, and then the second image looked a little more similar to a smiley face, which is what the target image was. Then, the other two black and white images were almost perfect copies of the target

image, and we were getting those right away, which led us to realize that the algorithm was not learning from itself anymore.

Final versions of the code



With this last set of images shown above, the algorithm was learning from itself again, and this is where we switched from using OpenCV to using MatPlotLibs for showing the images, so we could see the image slowly turning into the target image.

Next Steps

If we had more time to work on this project and improve it, we could improve it in several ways. First, we can ensure that the final image's colors match the original. Right now, the algorithm only outputs images in greens and blues. We would like the images to be produced in the colors of the original image that it recreates.

Next, we could add a feature that stops the program once it reaches the best possible image. The algorithm is set to run for a while before it stops without checking if it has the correct image. This will save time and resources when it doesn't need to keep running once it figures it out.

We also want the project to handle larger images. Currently, it will not accept bigger images because of the pixel-to-array technique that we used. By making some changes to how the program works, we could make it capable of handling bigger images without slowing down or crashing.

We believe these improvements will make this algorithm much more efficient and accurate.

References

[1] S. Jeswara, "2D image reconstruction using Genetic Algorithm," *Medium*, Nov. 27, 2021. https://medium.com/analytics-vidhya/2d-image-reconstruction-using-genetic-algorithm-e6ab1c2ea073 (accessed Mar. 25, 2024).
[2] "AI Image Generators for Beginners: A Guide to get started," *Impossible Images*, May 12, 2023. https://impossibleimages.ai/ai-image-generators-for-beginners-a-complete-guide/ (accessed Mar. 25, 2024).
[3] "AI Image Generation Explained: Techniques, Applications, and Limitations," *AltexSoft*, Jul. 10, 2023. https://www.altexsoft.com/blog/ai-image-generation/ (accessed Mar. 25, 2024).
[4] P. Knoll and S. Mirzaei, "Validation of a parallel genetic algorithm for image reconstruction from projections," *Journal of Parallel and Distributed Computing*, vol. 63, no. 3, pp. 356–359, Jan. 2003, doi: https://doi.org/10.1016/s0743-7315(03)00019-4.
[5] A. Gad, "Reproducing Images using a Genetic Algorithm with Python," *Medium*, Sep. 23, 2021. https://heartbeat.comet.ml/reproducing-images-using-a-genetic-algorithm-with-python-91fc701ff84 (accessed Mar. 26, 2024).
[6] D. M, "How to use Genetic Algorithm for Image reconstruction?," *Analytics India Magazine*, Jul. 06, 2022. https://analyticsindiamag.com/image-reconstruction-using-genetic-algorithms/ (accessed Mar. 25, 2024).
[7] A. Piwanska and U. Grycuk, "Images Reconstruction with Use of a Genetic Algorithm," *IEEE Xplore*, Jul. 16, 2007. https://ieeexplore.ieee.org/document/4273534 (accessed Mar. 26, 2024).
[8] Grammarly, "Write your best with Grammarly.," *Grammarly.com*, 2023. https://www.grammarly.com
[9] J. Hunter, D. Dale, E. Firing, and M. Droettboom, "matplotlib.pyplot — Matplotlib 3.5.3 documentation," *matplotlib.org*. https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html
[10] "Animations using Matplotlib — Matplotlib 3.8.2 documentation," *matplotlib.org*. https://matplotlib.org/stable/users/explain/animations/animations.html

[11] Rajnis09, “Python OpenCV | cv2.imread() method,” *GeeksforGeeks*, Aug. 02, 2019. https://www.geeksforgeeks.org/python-opencv-cv2-imread-method/
[12] “Visualization with Matplotlib | Python Data Science Handbook,” *jakevdp.github.io*. https://jakevdp.github.io/PythonDataScienceHandbook/04.00-introduction-to-matplotlib.html
[13] “Python | Peak Signal-to-Noise Ratio (PSNR),” *GeeksforGeeks*, Jan. 31, 2020. https://www.geeksforgeeks.org/python-peak-signal-to-noise-ratio-psnr/
[14] “Finding the Peak Signal-to-Noise Ratio (PSNR) using Python,” *www.tutorialspoint.com*. https://www.tutorialspoint.com/finding-the-peak-signal-to-noise-ratio-psnr-using-python