Aim

The aim of this project is to simulate the evolution of a population of images to match a given target color using a genetic algorithm. This involves evolving a population of randomly generated images over several generations to minimize the difference between the images and the target color.

Objective

1. To understand the process of natural selection and genetic evolution.
2. To apply genetic algorithms to solve an optimization problem.
3. To visualize the evolutionary process and the resulting evolved images.
4. To develop an intuition for genetic operators like selection, crossover, and mutation.

Summary

This project demonstrates the use of a genetic algorithm to evolve a population of images towards a target color. The algorithm starts with a population of random images and iteratively improves them by selecting the best images (elites), breeding them, and introducing mutations. The fitness of each image is measured by the mean absolute difference between its RGB values and the target RGB values. Over multiple generations, the population converges towards the target color.

Tools and Libraries Used

- **Python**: Programming language used for implementation.
- **NumPy**: Library for numerical operations on arrays.
- **Matplotlib**: Library for plotting and visualizing images.
- **Google Colab**: Platform for running the code on a T4 GPU.

Procedure

1. **Initialization**: Create an initial population of random images.
2. **Fitness Calculation**: Compute the fitness of each image based on its similarity to the target color.
3. **Selection**: Select a subset of the best images (elites) based on their fitness.
4. **Crossover**: Generate new images by combining parts of two parent images.
5. **Mutation**: Introduce random changes to some of the new images.
6. **Evolution**: Replace the old population with the new one and repeat the process for a specified number of generations.
7. **Visualization**: Display the target color and the best evolved image after the final generation.

Explanation of the Code

1. **Fitness Calculation**:

CODE:

```
def calc_fitness(chromosome, target_rgb):
    return np.mean(np.abs(chromosome - target_rgb))
```

2. **Population Evolution**:

CODE:

```
def evolve_population(population, target_rgb, mutation_rate, elite_ratio):
    fitness_values = [calc_fitness(c, target_rgb) for c in population]
    elite_index = np.argsort(fitness_values)[:int(len(population) * elite_ratio)]
    elites = [population[i] for i in elite_index]
    new_population = []

    while len(new_population) < len(population) - len(elites):
        parent1, parent2 = random.choices(elites, k=2)
        child = np.zeros(parent1.shape)

        for i in range(len(parent1)):
            if random.random() < 0.5:
                child[i] = parent1[i]
            else:
                child[i] = parent2[i]

            if random.random() < mutation_rate:
                child[i] = np.random.randint(0, 256)

        new_population.append(child)

    return elites + new_population
```

3. **Simulation**:

CODE:

```
def simulate(size, target_rgb, generations, mutation_rate, elite_ratio):
    population = [np.random.randint(0, 256, size=(3,)) for _ in range(size)]

    for i in range(generations):
        population = evolve_population(population, target_rgb, mutation_rate, elite_ratio)
        best_fit = min(population, key=lambda c: calc_fitness(c, target_rgb))
        print(f'Generation {i}: Best fitness {calc_fitness(best_fit, target_rgb)}, RGB values {best_fit}')

    return min(population, key=lambda c: calc_fitness(c, target_rgb))
```

4. **2D Image Extension**:

CODE:

```
def calc_fitness(chromosome, target_grid):
    return np.mean(np.abs(chromosome - target_grid))

def evolve_population(population, target_grid, mutation_rate, elite_ratio):
    fitness_values = [calc_fitness(c, target_grid) for c in population]
    elite_index = np.argsort(fitness_values)[:int(len(population) * elite_ratio)]
    elites = [population[i] for i in elite_index]
    new_population = []

    while len(new_population) < len(population) - len(elites):
        parent1, parent2 = random.choices(elites, k=2)
        child = np.zeros(parent1.shape)

        for i in range(len(parent1)):
            for j in range(len(parent1[i])):
                for k in range(len(parent1[i][j])):
                    if random.random() < 0.5:
                        child[i][j][k] = parent1[i][j][k]
            else:
                child[i][j][k] = parent2[i][j][k]

            if random.random() < mutation_rate:
                child[i][j][k] = np.random.randint(0, 256)

        new_population.append(child)

    return elites + new_population

def simulate(size, target_grid, generations, mutation_rate, elite_ratio):
    grid_size = (16, 16, 3)
    population = [np.random.randint(0, 256, size=grid_size) for _ in range(size)]
    images = []

    for i in range(generations):
        population = evolve_population(population, target_grid, mutation_rate, elite_ratio)
        best_fit = min(population, key=lambda c: calc_fitness(c, target_grid))
        print(f'Generation {i}: Best fitness {calc_fitness(best_fit, target_grid)}')
        images.append(best_fit.astype(np.uint8))

    return min(population, key=lambda c: calc_fitness(c, target_grid)), images
```

Highlights

- **Genetic Algorithm**: The project demonstrates the use of a genetic algorithm, including selection, crossover, and mutation, to evolve a population towards a target solution.
- **Visualization**: The evolution of the population is visualized, showing how the images become more similar to the target color over generations.
- **Extensibility**: The project is extended from evolving single RGB values to evolving 2D images, demonstrating the flexibility of genetic algorithms in handling more complex problems.

Conclusion

This project successfully demonstrates the application of genetic algorithms for optimizing a population of images to match a target color. Through iterative evolution, the population converges towards the target, showcasing the effectiveness of natural selection principles in solving optimization problems. The visualization of the evolutionary process provides a clear understanding of how genetic algorithms work and their potential in various applications.