GENETIC ALGORITHM

Generating image

Things to know : the main code is generating an image with size of 32*32 by it self for simpler calculations and having low maintenance on laptop! .

- **First population** : the first populations is 50 and the image size is 32 by 32.
- **Chromosome : color image** : we use **numpy** to draw random integers in range of (0 , 256) for each pixel channel which is (R , G , B) and the shape is (height , width , 3) so every pixels get three values between (0 to 255) and also each channel value is stored as an 8-bit unsigned integer (standard for images)

  Grayscale : same as before but now the array is 2D and again giving values of (0 , 255) to each pixels .

  Binary image : now we using float numbers uniformly in (0.0 , 1.0) and comparing the pixels with conditions of being greater or be smaller than 0.5

- **Calculate fitness :** we converting both images to avoid the overflow then for each pixels we compute the squared difference and next we calculate the average over all pixels (mse) . now we have to turn that error into fitness The worst possible squared error for a single channel is $(0-255)^2 = 255^2$. By subtracting the MSE from this maximum we flip it so that **higher** values mean **better** matches. Whenever the algorithm evaluates a population it calls this calculate_fitness on each chromosome against the target, and then uses those fitness scores to guide selection, crossover, and mutation toward ever better approximations of the original image.

- **Tournament selection :** randomly picks a subset of chromosomes then evaluate each sample chromosome’s fitness against the target (original image pixels) then we select the best one with highest fitness from it. Note : we gave the better chance to the higher fitness individuals but making it not always picking the absolute best one.

- **Crossover :** for recombining two parents to produce two new children that inherit pixels from both . **how it works : chance to skip** : with probability 1 – crossover_rate Simply clone's the parents (no mixing)

  Mask generation : creates a Boolean mask of the same shape as the images where each position is True (~50% of the time)

  Recombine : in child 1 whenever mask == true take pixels from parent 2 ; elsewhere keep parent1 and in child 2 it is the opposite.

  Why : allows building blocks (pixels patterns) from two good parents to combine into potentially better offspring.

- **Mutate :** for introducing random variation so the population can explore new regions of the solution space .

  How it works: generates a mask marking each pixel for mutation with probability **mutation_rate** then reassign those marked pixels to new random values **:**

  For color/grayscale : picking a uniform from (0 , 255)

  For binary : randomly choose black (0) or white (255)

- **Evolve image :** for preserving the best ever found

  Why it's there : the code selects two parents via tournament selection then crossover them to get two children , next mutate each child and replace the old population . it is like a factory for gathering information and giving the new products to us .