Introduction to AI Assignment 2

Hussein Younes @husseinyounes
April 27, 2020

Contents

1 Problem representation and image manipulation **2**
1.1 Problem representation and overview . . . 2
1.2 Image manipulation . . . 2

2 Genetic Algorithm implementation **3**
2.1 Algorithm explanation . . . 3
2.2 Chromosomes representation . . . 3
2.3 Fitness function . . . 3
2.4 Crossover . . . 4
2.5 Mutation . . . 4
2.6 Optimizations . . . 4

3 The artistic aspect **5**

4 Sample images **6**

1 Problem representation and image manipulation

1.1 Problem representation and overview

- An evolutionary algorithm was designed to reproduce and generate images in an artistic way. The algorithm is given a reference image and then tries to regenerate this image using polygons and different shapes. The algorithm was designed and implemented in python which has some benefits in terms of the available tools for image manipulation but also has some drawbacks in term of run-time speed in pixel rendering and canvas allocation.
- The main idea of the algorithm is to try approaching the original image by drawing shapes on the canvas and try to improve these drawings generation after another depending on the value of the fitness function. The algorithm starts from random N polygons with some transparency ratio. In each optimization step it randomly modifies one parameter (like color components or polygon vertices) and check whether such new variant looks more like the original image. If it is, we keep it, and continue to mutate this one instead. The final image is represented as a collection of overlapping polygons of various colors and transparencies to get an artistic view of the source image. (more implementation details in 2).

1.2 Image manipulation

- Firstly, the pre-built OpenCV packages for Python was used for reading the images, writing on them, and drawing the polygons. However, as openCV requires copying the image and adding an overlay to change the color density parameter (alpha), rendering the pixels becomes very costly when the population size increases. Therefore, I used Python Imaging Library for changing the alpha parameter rather than copying the image at every iteration.
- **NOTE:** the algorithm works with any image rather than just 512x512
- Here are the main functions used for image manipulation:
  - **Reading** the image file

```
182  def make_art():
183      filename = 'mona_lisa_crop.jpg'
184      img = cv2.imread(filename, 1)
```

Figure 1: reading an image using openCV

  - **Writing/Saving** the image file

```
name = img_file_name + str(generation_num) + ".jpg"
if generation_num % 500 == 0:
    cv2.imwrite(name, img)
    cv2.waitKey(30)
```

Figure 2: saving an image using openCV

  - **Drawing** a list of polygons on an image

```
drw = ImageDraw.Draw(image, 'RGBA')
for i in range(indv.shape[0]):
    drw.polygon(indv[i][0], tuple(indv[i][1]))
image = numpy.array(image).astype(numpy.uint8)
```

Figure 3: drawing a polygon using PIL

– **NOTE:** for testing and optimization, I implemented a resize function that resizes the source image to a smaller image to reduce the number of pixels and the computational complexity for high quality images.

2 Genetic Algorithm implementation

2.1 Algorithm explanation

Let's start with the general scheme of the algorithm:

1. **Initial population generation:** first of all, the algorithm starts by generating the initial population randomly. The initial population contains N individuals each of which is a list of polygons that are randomly generated and ready to mutate and propagate through the next generations.

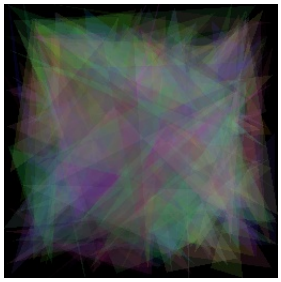


Figure 4: Individual sample from the initial population

NOTE: All the initial polygons are drawn with an initial alpha, which mean that not all of them are completely visible at the beginning, and only the good ones will start to appear and and contribute to the image. After a while, almost all the polygons will be contributing and overlapping to produce the artistic image.

2. **Selection Mechanism:** After having the initial population, we select the fittest M individuals as candidates for the crossover process, to select the best parents we first calculate the qualities of the population individuals and then we sort the qualities to select the best candidates.
3. **Fitness function:** to get the fitness (quality) of an individual, we calculate the pixel difference between the individual and the source image. The lower the difference the fittest the individual and therefore closer to the goal image and better looking.
4. **Crossover:** after calculating the fitness for the population and selecting the parents, we take pairs of parents to cross them over, which is taking half of the polygon from the first parent and the other half from the second parent and put the resulting chromosome in a child. We continue taking such parents pairs until we satisfy the number of needed children.
5. **Mutation:** for the population to improve its genes, we mutate the children to introduce new changes to the already existing individuals hoping that these changes will increase the fitness function of the population in the following generations.

2.2 Chromosomes representation

- A chromosome is represented by transforming an image (e.g 512x512x3) to a 1D array of length (e.g 786432 = 512*512*3), this representation is easier to perform mutations and crossover. After that we can easily fold the chromosome to get back the 512x512x3 image to draw it.

2.3 Fitness function

- the fitness function is simply the pixel difference between the current individual and the source image, the fitness function is crucial for improving the current looking of the image as it tells the population how good its chromosomes are. After calculating the pixel difference we can get a percentage of how similar our image

is to the optimal image. This percentage can be easily obtained as follows:

$$100 * \left(1 - \frac{dif}{(255 * shape[0] * shape[1] * 3)}\right)$$

Where $shape$ is the shape of the source image. Please note that the higher the percentage, the better the image quality.

2.4 Crossover

- After having the fitness of each individual in the population, we select the fittest M parents and we start the crossover process and the children generation. Each child takes half of its genes from a parent, and the other half from the other parent as described earlier. Accordingly, I tried several crossover functions like getting genes randomly from the parents into the child. However, experimentally, this has introduced more bad children as it wastes most of the good genes from the parents. Therefore, the first method was adopted for my implementation.

2.5 Mutation

- After the crossover process, for each child in the new population we introduce a mutation with a certain rate. A mutation is one of the following operations with a corresponding probability for each:
  1. selecting a random vertex of a random polygon and changing it by delta normally distributed near the current position. Such a mutation is called Gaussian mutation which makes the polygon vertices converge quickly to a better position rather than setting them completely randomly.
  2. changing one the of the color parameter R,G,B to a random value between 0,255
  3. changing the alpha parameter is not trivial as it may let polygons with higher areas to have denser colors and therefore preventing some polygons from actually appearing in the image, such cases can be avoided in time of the execution by the mutation itself but it may take longer to get rid of such solidly colored polygons. Therefore, setting alpha to low initial value solves the problem.

2.6 Optimizations

- There are few things to point out for optimizing the time complexity of the used genetic algorithm, first I implemented the naive solution and then I started tuning and optimizing the following issues:
  1. Clearly, most of the execution time was wasted during the canvas allocation and image copying to add the color density overlays in openCV. Therefore, I used the Python Imaging Library to get rid of the frequent image copying. However, at every iteration we need to draw all the polygons of an individual which is a drawback that might be solved by cashing the canvas as at each iteration only few mutations are performed or by decreasing the size of population.
  2. As described in the algorithm, we have many parameters to play with such as the population size, mutation rate, number of vertices of the polygons, etc. Accordingly, bigger population sizes introduce more improvements after each generation and lead to a higher fitness because of the diversity of the population's genes. However, there are wasted cycles and unnecessary mutations and crossovers that are done many time at each iteration, moreover, we need more calculations for a bigger population size. Therefor, setting the population size to a fairly small number have made more improvements faster than having a big population size.
  3. Introducing the Gaussian mutation was a great improvement in terms of execution speed. Because it allows the points to converge to convenient positions faster.

3 The artistic aspect

- Humans define art as the expression or application of human creative skill and imagination, typically in a visual form such as painting or sculpture, producing works to be appreciated primarily for their beauty or emotional power. Such an emotional power differs from a human to another, because we are different in nature, and what may seem beautiful to someone might not seem so to the other. And this is why judges may have different opinions while they are ranking our work and images. For me, art is what makes me happy just to look at or listen to. For example, image evolution for me is an art and I wouldn't mind trying my algorithm on images and watching them improving for hours.
- What makes the computer unique in his drawing approach is that it gives you another perspective of the image at each run. Unlike what most people do first when drawing, the computer will not draw the faces or shapes outline, it will start from unique positions that diverges to the image from another point of view at each program run.
- For example, I consider the following image as a piece of art because it shows the power of the overlapping polygons, and how the cooperation between these shapes contributes in performing some details like making curvy lines. Moreover, the alpha parameter along with the overlapping polygons adds a depth to the image, especially to the dark areas. Moreover, approaching a piece of art from different perspective, adds an artistic dimension to it.

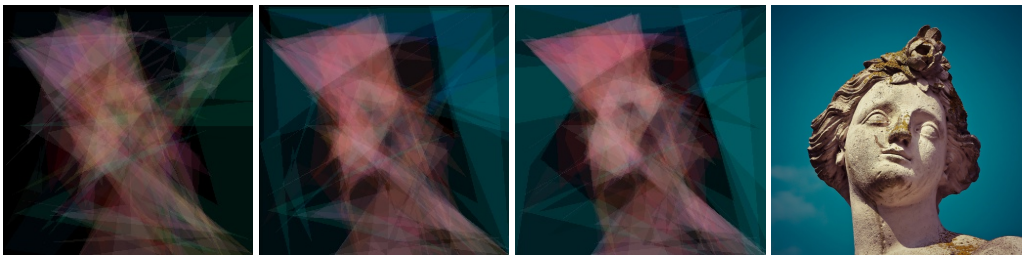
Figure 5: MonaLise generated by the genetic algorithm

4 Sample images

Here are two samples for image evolution:

(a) generation 1000 (b) generation 10000 (c) generation 80000 (d) Optimal image

Figure 6: Evolution of the MonaLisa using 100 polygon



(a) generation 1500 (b) generation 6000 (c) generation 70000 (d) Optimal image

Figure 7: Image evolution using 50 polygons