متین فاضل 9825583

الگوریتم ژنتیک از روند تکاملی زیستی الهام گرفته شده است و به طور عمده برای انتخاب ویژگی بهینه در مسائل مختلف استفاده می‌شود. الگوریتم ژنتیک به طور کلی از روش‌های هیوریستیک برای یافتن بهترین راه‌حل ممکن نزدیک به راه‌حل بهینه پیروی می‌کند. این الگوریتم در بسیاری از زمینه‌ها کاربرد دارد، بیشتر در مسائل بهینه‌سازی. همچنین می‌توان از آن برای بازسازی تصاویر به منظور به دست آوردن تصاویر در قالب اصلی آنها استفاده کرد.

الگوریتم‌های ژنتیک در ابتدا بر روی یک مجموعه اولیه از بهترین راه‌حل‌های ممکن، به نام جمعیت اولیه، عمل می‌کنند. هر یک از راه‌حل‌ها در این مجموعه بر اساس پارامترهای خاصی دسته‌بندی می‌شوند و در الگوریتم‌های ژنتیک به عنوان ژن‌ها شناخته می‌شوند. ژن‌ها در الگوریتم‌های ژنتیک بر اساس پارامترهای تعیین شده رتبه‌بندی می‌شوند. ژن‌هایی که تمام توانایی‌ها را دارند، به یک تابع برازندگی منتقل می‌شوند تا برازندگی بهترین راه‌حل‌های انتخاب شده از مجموعه راه‌حل‌های کاندید به مسئله را تعیین کند. اگر بهترین راه‌حل این توانایی را برای ارث بردن از تمامیت تابع سلامتی داشته باشد، این راه‌حل به نسل بعدی منتقل می‌شود. عملکرد کلی الگوریتم‌های ژنتیک در سه سطح selection، Crossover و Mutation اتفاق می‌افتد.

حال به بررسی کد زده شده میپردازیم:

ابتدا در سلول زیر تمامی کتابخانه ها ایمپورت میشوند:

```
import cv2 as opencv
import random
from skimage.metrics import peak_signal_noise_ratio as psns
import cv2
import matplotlib.pyplot as plt
import numpy as np
```

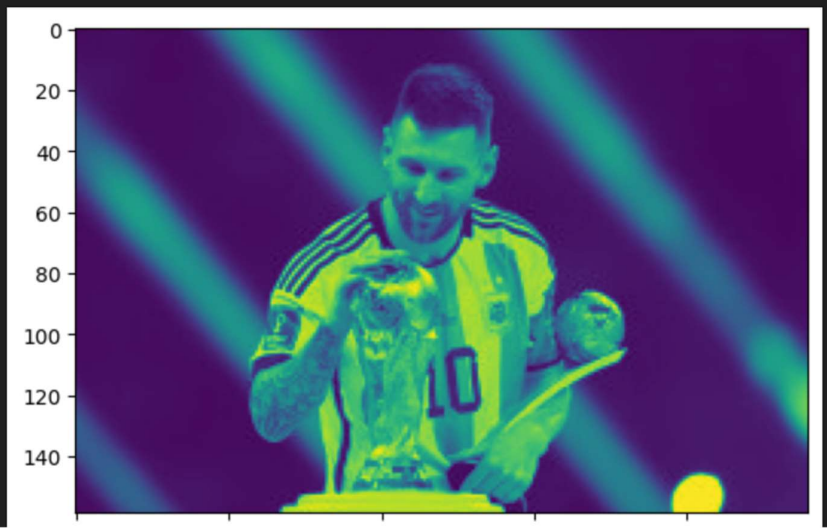
✓ 0.0s

در سلول زیر با استفاده از کتابخانه opencv تصویر بارگذاری شده است و به صورت gray scale درآمده است و در آخر هم خروجی عکس نمایش داده شده است:

```
main_image = opencv.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
image_width, image_height  = main_image.shape
print(main_image.shape)
plt.imshow(main_image)
plt.show()
```

✓ 0.2s

```
(159, 240)
```



در این قسمت هم به صورت کاملا تصادفی یکسری دایره بر روی عکس ها قرار میگیرد تا عملیات mutation انجام گیرد(برای این کار میشد برای مثال از خطوط و یا نویز های رندوم استفاده کرد)

```
def make_circles(image):

    x = np.random.randint(0,image_width-20)
    y = np.random.randint(0,image_height-20)
    radius = np.random.randint(1,min(image_width, image_height)/2)
    center = (x,y)
    outline_width = np.random.randint(1,3)
    color = (random.randint(0,255))

    cv2.circle(image, center, radius, color, thickness=outline_width)
```

✓ 0.0s

در تابع create_random_population به صورت تصادفی جمعیت اولیه را مقدار دهی میکنیم به این صورت که یک سری تصاویر خالی به فرم تصویر اصلی ایجاد میکنیم و بر روی هر یک از آنها یک سری دایره به صورت رندوم میکشیم و به جمعیت اضافه میکنیم و در تابع evaluate_fitness هم با استفاده از peak_signal_noise_ratio میزان شباهت عکس تولید شده را با عکس اصلی را پیدا میکنیم:

```
def create_random_population(size):
    population = []
    for I in range(0, size):
        blank_image = np.zeros((image_width, image_height), dtype=np.uint8)
        make_circles(blank_image)
        population.append(blank_image)
    return population
```

✓ 0.0s

```
def evaluate_fitness(image):
    return psns(image, main_image)
```

✓ 0.0s

در این تابع crossover دو عکس به عنوان والد داده میشود و با انتخاب یک قسمت به عنوان crossover_point به عنوان قسمت جدا شدن استفاده میکنیم و در نهایت با ترکیب قسمت اول والد اول و قسمت دوم والد دوم مقدار offspring را بدست می آوریم:

```
def crossover(image1, image2):
    offspring = np.zeros(image1.shape,dtype=np.uint8)
    crossover_point = np.random.randint(0, image_width)
    offspring[:, :crossover_point] = image1[:, :crossover_point]
    offspring[:, crossover_point:] = image2[:, crossover_point:]
    return offspring
```

✓ 0.0s

در این قسمت که همان جهش نامیده میشود بر روی هر یک از عکس ها به صورت تصادفی یک دایره ایجاد میکنیم:

```
def mutate(image):
    image_filled = image.copy()
    make_circles(image_filled)

    return image_filled
```

✓ 0.0s

در این قسمت با توجه به مقادیر برازندگی هر یک از والد ها را به صورت رندوم انتخاب میکنیم و لیست نهایی را بر میگردانیم:

```python
def get_parents(local_population, local_fitnesses):
    local_parents_list = []
    for _ in range(0, len(local_population)):
        parents = random.choices(local_population, weights=local_fitnesses, k=2)
        local_parents_list.append(parents)
    return local_parents_list
```

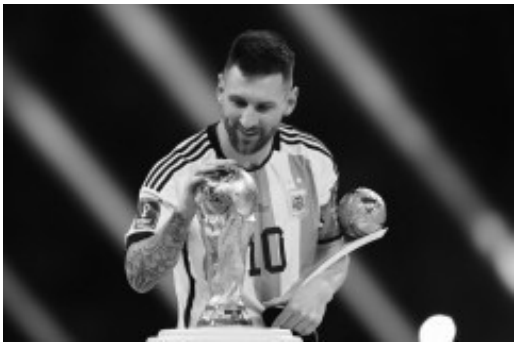
✓ 0.0s

در این قسمت سلول نهایی به اندازه تعداد نسل انتخاب شده (28000) این الگوریتم را اجرا میکنیم و با توجه به جمعیت تولید شده در قسمت قبل ابتدا مقدار برازندگی را محاسبه میکنیم و 5 کروموزم برتر را انتخاب میکنیم و سپس والد های آنها را بر میگزینیم با توجه به مقادیر برازندگی و ابتدا عملیات crossover و سپس mutation با توجه به شانس ایجاد جهش و سپس در نسل بعدی new_population هم عکس های برتر را قرار میدهیم و به ازای هر 1000 نسل میزان برازندگی و عکس را ذخیره میکنیم:

```
for generation in range(0, 28000):

    fitnesses = []
    for index,img in enumerate(population):
        fitness_value = evaluate_fitness(img)
        fitnesses.append(fitness_value)

    top_population_ids = np.argsort(fitnesses)[-5:]

    new_population = []

    parents_list = get_parents(population, fitnesses)

    for i in range(0, population_size):
        first_parent = parents_list[i][0]
        second_parent = parents_list[i][1]
        new_img = crossover(first_parent, second_parent)

        if random.uniform(0.0, 1.0) < chance_mutation:
            new_img = mutate(new_img)
        new_population.append(new_img)

    for ids in top_population_ids:
        new_population.append(population[ids])

    if generation % 1000 == 0:
        print(generation)
        print('The fitness value of the best in the population : ',fitnesses[top_population_ids[0]])

    open_cv_image = np.array(population[top_population_ids[0]])

    population = new_population

    if (generation % 1000 == 0):
        cv2.imwrite(f"results/Iter{generation}.jpg", open_cv_image)

cv2.imwrite("output.jpg", open_cv_image)
```

2m 31.6s

عکس اولیه:



عکس نهایی تولید شده: