به نام خدا

پروژه درس هوش مصنوعی

قطعه‌بندی تصاویر سلول‌های سرطان خون به کمک الگوریتم ژنتیک

دریا فریور – ۶۱۰۳۹۷۱۴۱

تاریخ تحویل: ۲۰ بهمن ۱۴۰۰

Artificial Intelligence Project

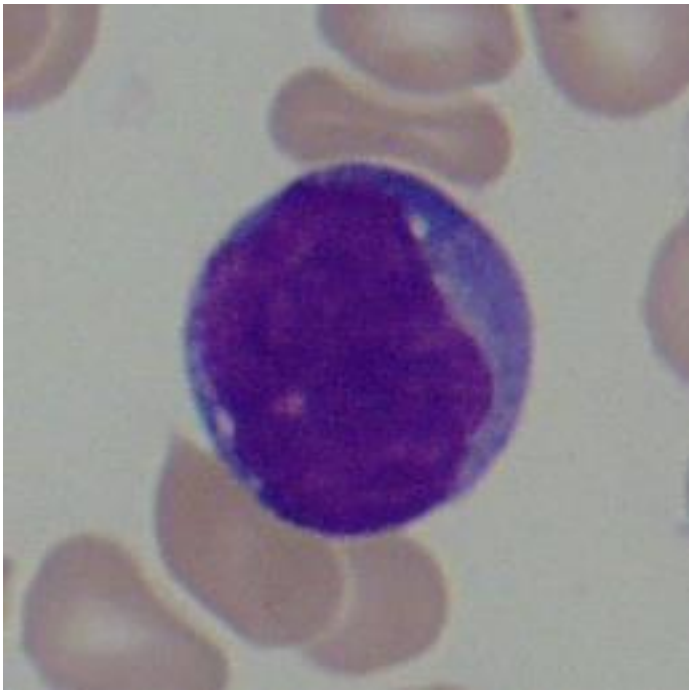
Image Fragmentation of Leukemia Cells Using Genetic Algorithm

چکیده

در این پروژه ما قصد قطعه‌بندی و رنگ‌آمیزی تصاویر سلول‌های سرطانی خون با 3 و 4 و 5 رنگ را داریم. و این کار به وسیله ماتریس‌های RGB انجام شده و از شاخص سنجش PSNR استفاده کردیم. در نهایت برای رسیدن به PSNR مطلوب از الگوریتم ژنتیک که یک الگوریتم سرچ و بهینه‌سازی می‌باشد استفاده کرده‌ایم. کدهای استفاده شده در این پروژه غالبا ابداعی بوده و برای درک بهتر الگوریتم ژنتیک از دیتافریم‌های کتابخانه‌ی pandas استفاده شده است.

شرح پروژه: مقدمه

این پروژه در خصوص قطعه‌بندی تصاویر سلول‌های سرطانی خون است. و ما در اینجا قصد داریم که تصاویری مانند «شکل 1» را با 3، 4 و یا 5 رنگ، رنگ‌آمیزی کنیم و با توابع ارزیابی، خروجی حاصل شده را بسنجیم. و سپس با استفاده از الگوریتم ژنتیک به جواب بهینه‌تری دستیابی پیدا کرده و جواب حاصل شده را نیز ارزشیابی کنیم و در نهایت برنامه‌ی نوشته شده را بر روی دیتای در نظر گرفته شده، که شامل 260 تصویر سلول سرطانی خون است اعمال کنیم.



«شکل 1»

در ابتدا برای شروع و درک اینکه چرا برای این پروژه نیازمند الگوریتم ژنتیک هستیم، کد پایتون زیر نوشته شده است، در واقع می‌خواهیم تنها برای درک بهتر مساله با سه رنگ رندوم، که از پیکسل‌های «شکل 1» گزینش شده اند، آن را رنگ آمیزی کنیم. و خروجی‌های گوناگون آن را مشاهده کنیم.

طبیعی است که سه رنگ قاعدتا خروجی بدتری از 4 و 5 دارد. در نتیجه گزینه‌ی بهتری است که مشاهده کنیم از کد ما در بدترین حالت چه خروجی‌ای حاصل می‌شود.

به این منظور با استفاده از کتابخانه‌های مقدماتی پایتون، ابتدا تصویر را تبدیل به ماتریس RGB می‌کنیم که برای شکل 1، یک آرا یه‌ی نامپای به ابعاد (257, 257, 3) حاصل می‌شود.

```
array([[[139, 120, 122],
        [141, 121, 123],
        [143, 123, 125],
        ...,
        [150, 131, 137],
        [147, 131, 132],
        [144, 128, 129]],

       [[142, 124, 124],
        [141, 121, 122],
        [140, 120, 121],
        ...,
        [147, 131, 134],
        [145, 131, 131],
        [142, 128, 127]],

       [[131, 121, 120],
        [132, 122, 121],
        [134, 122, 122],
        ...,
        [149, 130, 134],
        [149, 130, 132],
        [149, 129, 131]],
```

سه پیکسل را به صورت تصادفی انتخاب کرده، و بعد از آن طبیعتا باید پیکسل‌هایی که نزدیک‌ترین مقدار به هر کدام از سه پیکسل ما دارند را با آن رنگ کنیم. طبیعتا این سنجش به وسیله‌ی میزان قرمزی، سبزی و آبی بودن امکان پذیر نمی‌باشد، و در نتیجه باید از این سه مقدار، میزان سفیدی/سیاهی استخراج شود که به صورت زیر محاسبه می‌شود.

$$0.299 \cdot Red + 0.587 \cdot Green + 0.114 \cdot Blue$$

در نهایت ماتریس تازه‌ای تشکیل می‌دهیم و خروجی را برای «شکل 1» مشاهده می‌کنیم.

```
import numpy as np
import PIL
from PIL import Image
import random
```

```
im = Image.open('Im001_1.tif')
A = np.squeeze(im, axis=None)
print(np.shape(A))
```

```
(257, 257, 3)
```

```
E1 = random.choice(A)
F1 = random.choice(E1)
E2 = random.choice(A)
F2 = random.choice(E2)
E3 = random.choice(A)
F3 = random.choice(E3)
```

```
Gray1 = F1[0]*0.299 + F1[1]*0.587 + F1[2]*0.114
Gray2 = F2[0]*0.299 + F2[1]*0.587 + F2[2]*0.114
Gray3 = F3[0]*0.299 + F3[1]*0.587 + F3[2]*0.114
U = [round(Gray1, 2), round(Gray2, 2), round(Gray3, 2)]
```

```
def find_nearest(array, value):
    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()
    return array[idx]
```

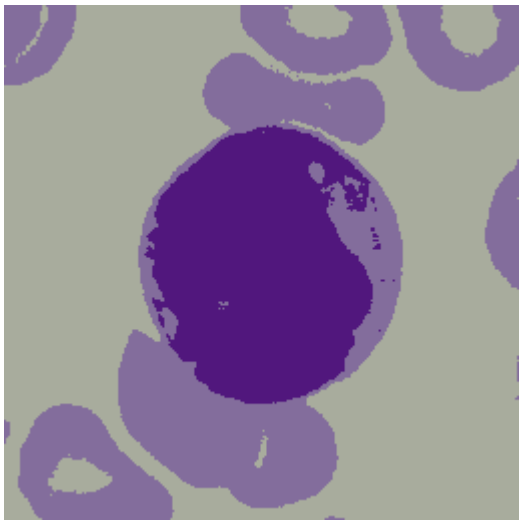
```
count = 0
B = []
for i in A:
    for j in i:
        jGray = j[0]*0.299 + j[1]*0.587 + j[2]*0.114
        o = find_nearest(U, jGray)
        if o == round(Gray1, 2):
            B.append([F1[0],F1[1],F1[2]])
            print('f1')
            count = count + 1
        elif o == round(Gray2, 2):
            B.append([F2[0],F2[1],F2[2]])
            print('f2')
            count = count + 1
        elif o == round(Gray3, 2):
            B.append([F3[0],F3[1],F3[2]])
            print('f3')
            count = count + 1
```

```
C = np.array(B)
D = np.reshape(C,(257, 257, 3))
```

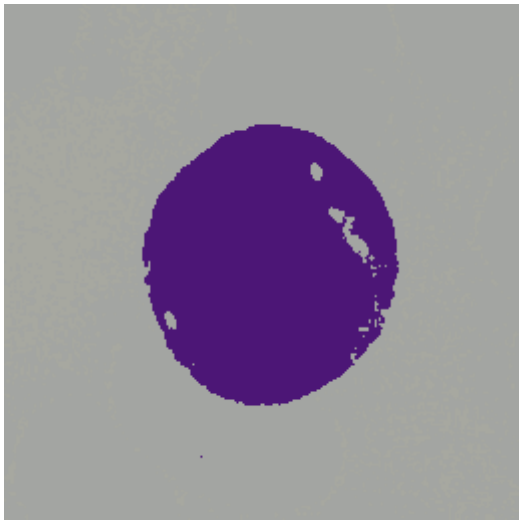
```
img1 = Image.fromarray(D, 'RGB')
img1.show()
img1.save('my.png')
```

به ازای مقادیر تصادفی مختلف خروجی‌های زیر برای «شکل 1» به دست می‌آید:

«شکل 2»



«شکل 3»



«شکل 4»

«شکل 5»

چنانچه مشاهد می‌شود خروجی‌ها، از کیفیت‌های بسیار متفاوتی برخوردارند. شکل 2 از رنگ‌بندی مناسبی برخوردار نمی‌باشد و شکل 4 و 5 نیز در تشخیص Edge ها ناتوان مانده است. شکل 3 خروجی قابل قبول‌تری است، هرچند که با 3 رنگ می‌توانیم نتیجه‌ای به مراتب بهتر از شکل 3 هم داشته باشیم.

اینجاست که بحث Optimization (بهینه‌سازی) مطرح می‌شود و ما به سراغ الگوریتم ژنتیک که یک الگوریتم آپتیمایزیشن محسوب می‌شود، می‌رویم.

ما قصد داریم با کمک الگوریتم ژنتیک بهترین یا حداقل یکی از بهترین جواب‌های ممکن را به دست بیاوریم. درک الگوریتم ژنتیک و نقشی که می‌تواند در پروژه‌ی ما داشته باشد، انتخاب توابع و پارامترهای مناسب و پیاده‌سازی صحیح، سهم عمده‌ای در کیفیت جواب نهایی دارند.

قبل از اینکه به سراغ الگوریتم ژنتیک برویم لازم است تابع برازش یا ارزیابی را بر روی یکی پاسخ اعمال کنیم تا در مرحله‌ی Fitness الگوریتم ژنتیک دچار مشکل نشویم.

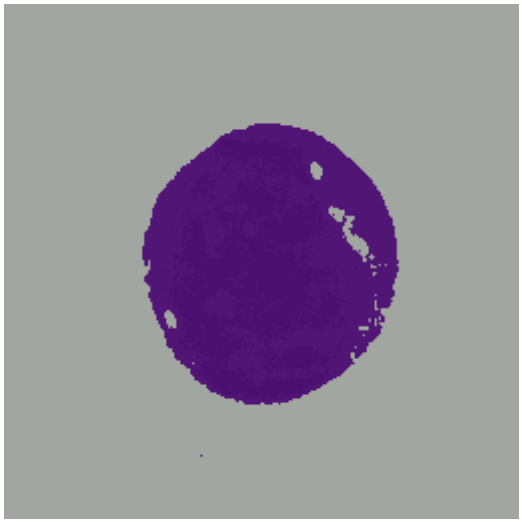
ما سنجش PSNR را انتخاب می‌کنیم.

$$
\begin{aligned}
PSNR &= 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \\
&= 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \\
&= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE).
\end{aligned}
$$

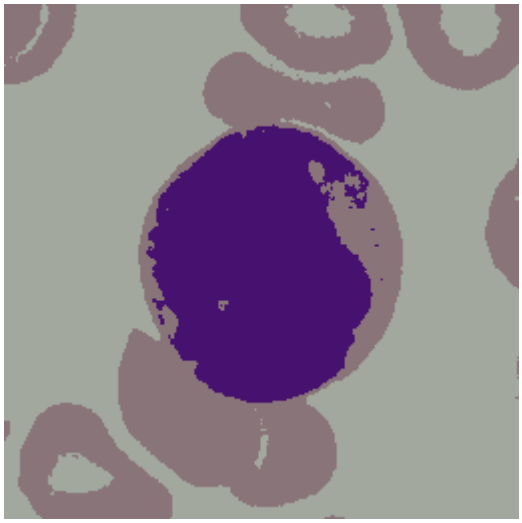
برای PSNR در cv2 دستوری وجود دارد و ما از آن استفاده می‌کنیم. و چندین بار امتحان می‌کنیم. نتایج به شکل زیر حاصل می‌شود.

```
img_a = cv2.imread('Im001_1.tif')
img_b = cv2.imread('my.tif')
cv2.PSNR(img_a, img_b)
```

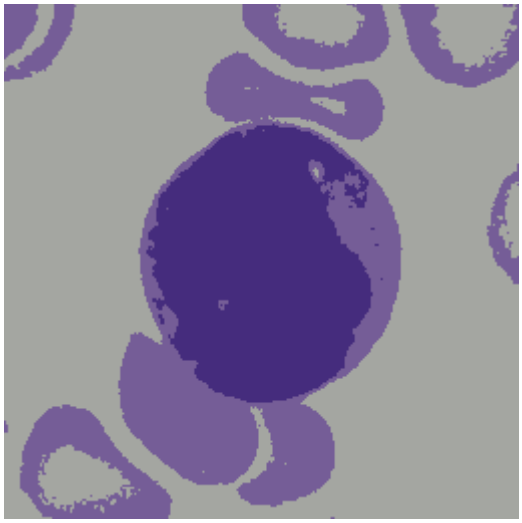
```
15.174868138712467
```



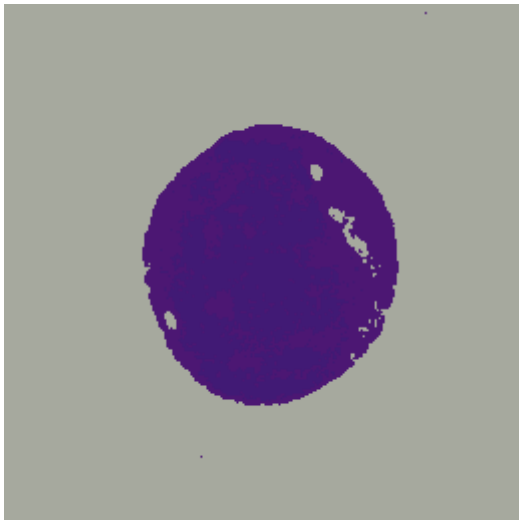
22.51145504436041



27.985572954240496



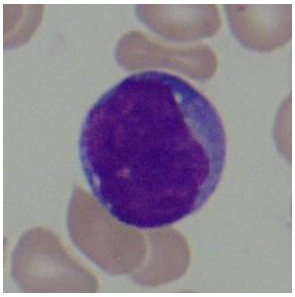
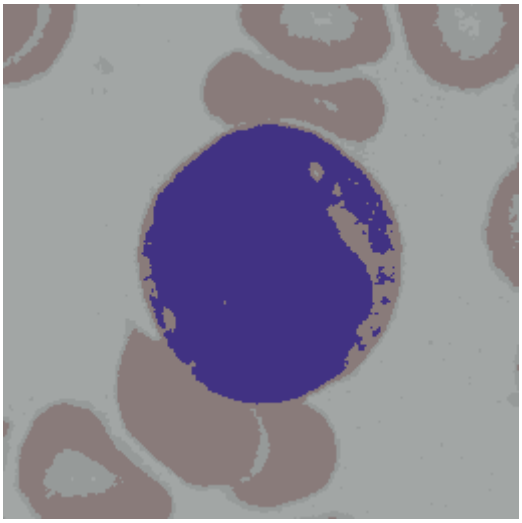
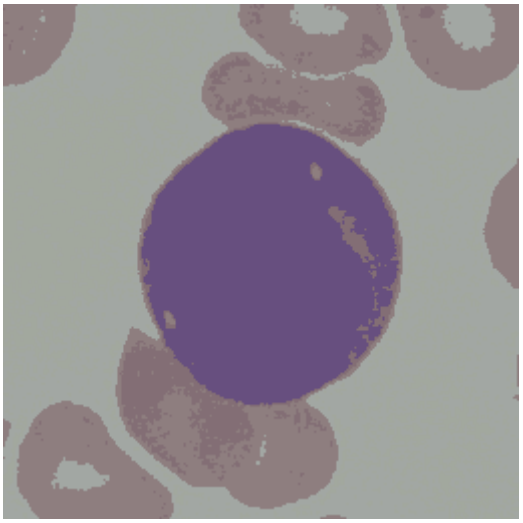
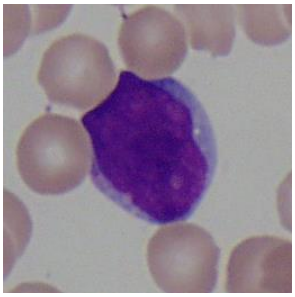
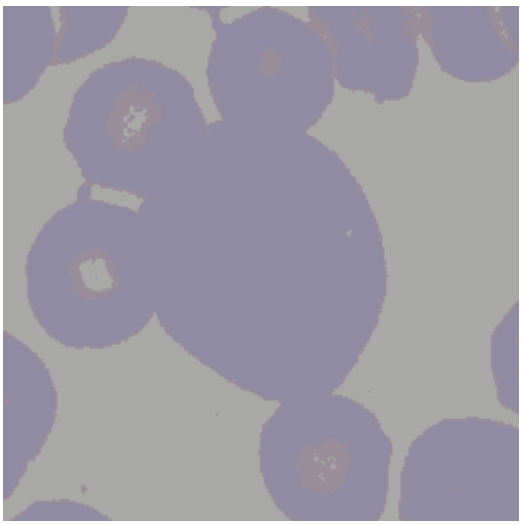
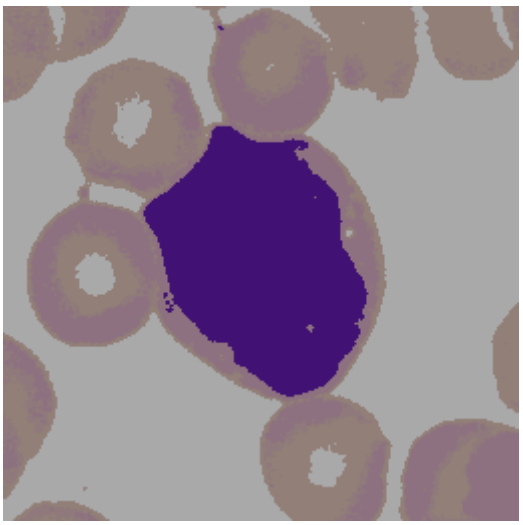
24.553332976551353

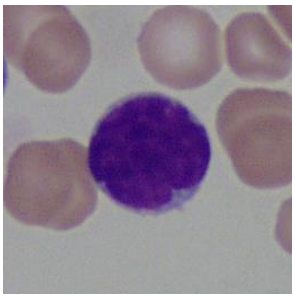


22.068654021240693

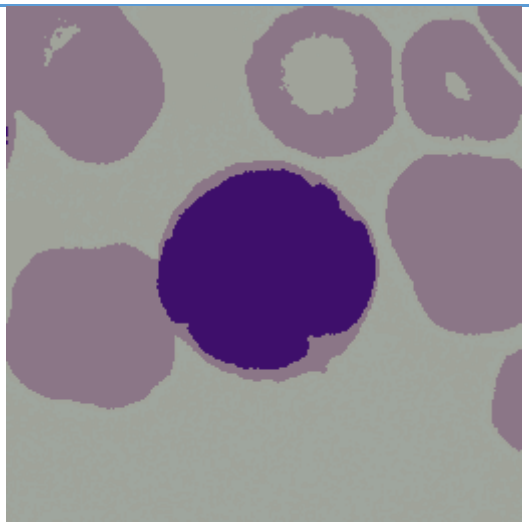
همچنین با 4 و 5 رنگ کاملا تصادفی برای تصاویر مختلف به نتایج زیر رسیده‌ایم.

کدهای پایتون که مشابه کد قسمت قبلی است، در قسمت پیوست آورده شده است.

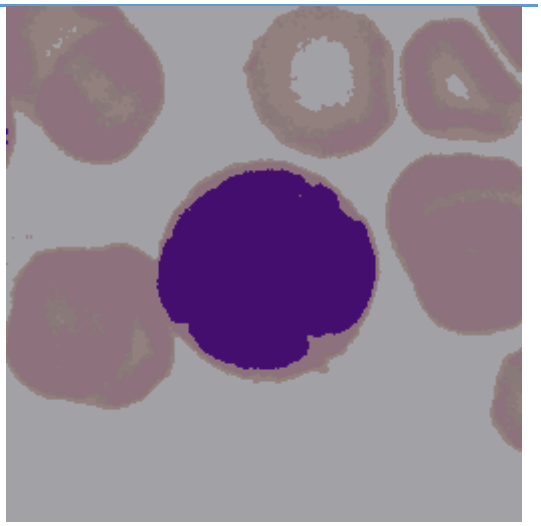
Original Image	With 4 Colors	With 5 Colors
 Im001_1.tif	 PSNR = 27.167762449485092	 PSNR = 25.030551760058085
 Im002_1.tif	 PSNR = 17.311125759941305	 PSNR = 28.132474723468526



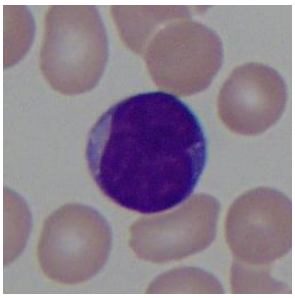
Im003_1.tif



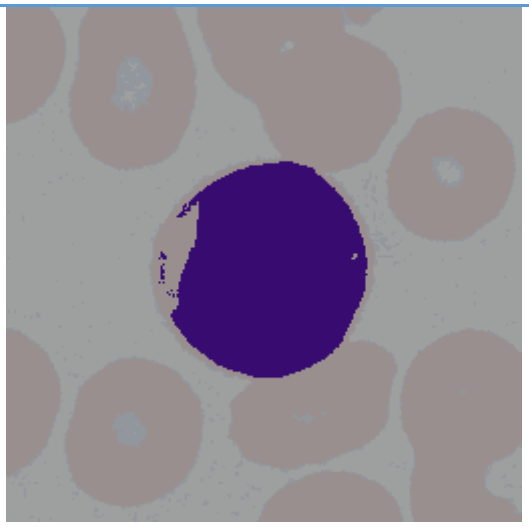
PSNR = 28.67486136871159



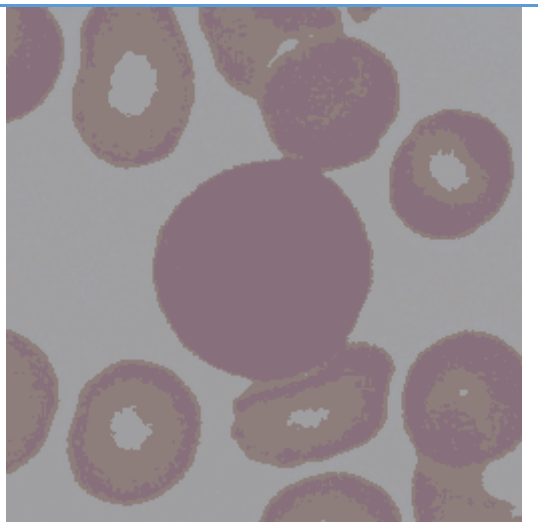
PSNR = 29.889787423509638



Im004_1.tif



PSNR = 28.67486136871159

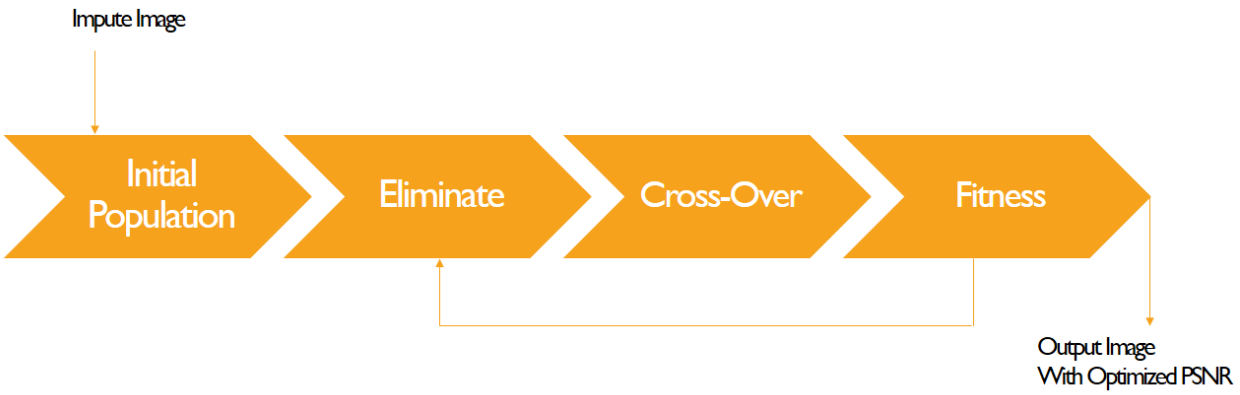


PSNR = 20.014787060793466

شرح پروژه: الگوریتم ژنتیک

الگوریتم ژنتیک

الگوریتم ژنتیک GA روش جستجوی احتمالاتی فراگیر است که از فرآیند تکامل زیست شناختی طبیعی پیروی می‌کند. الگوریتم ژنتیک بر جمعیت جواب‌های بالقوه عمل می‌کند و اصول تنازع بقا را در تولید تقریب‌های بهتر و بهتر جواب مساله به کار می‌گیرد. در هر نسل مجموعه‌ی جدیدی از تقریب‌ها با فرآیند اتخاب بهتر عضو براساس میزان برازش Fitness ساخته می‌شود. این فرآیند نهایتا به تکامل جمعیتی از اعضا ختم می‌شود که نسبت به اعضای اولیه ک در واقع والدین اصلی آنهاست با محیط سازگاری بهتری دارند.



الگوریتم ژنتیک برای 3 رنگ

قصد داریم برای حالتی که با سه رنگ تصاویر را رنگ‌آمیزی می‌کنیم الگوریتم ژنتیک طراحی کنیم:

فعال‌سازی جمعیت اولیه:

جمعیت اولیه را به صورت زیر طراحی می‌کنیم:

	1-R	1-B	1-G	2-R	2-B	2-G	3-R	3-B	3-G	PSNR
2	161	169	158	139	120	126	67	17	116	28.278851
6	158	164	160	63	16	108	138	118	127	27.947832
5	131	115	118	77	21	110	162	161	157	27.747291
1	161	168	161	73	23	118	153	144	139	26.437601
0	96	76	135	67	30	126	160	166	156	23.882020
4	165	166	161	162	167	163	87	31	120	22.185472
7	80	68	140	163	166	159	160	162	157	22.039804
3	144	124	149	145	129	130	156	164	153	17.927882

در هر ردیف سه رنگ و شاخص PSNR آنها مشخص شده است.

```
def initial_population(IMAGE):
    population = pd.DataFrame(0, index=np.arange(8), columns=['1-R','1-
B','1-G','2-R','2-B','2-G','3-R','3-B','3-G','PSNR'])
    for z in population.index:
        im = Image.open(IMAGE)
        A = np.squeeze(im, axis=None)
        E1 = random.choice(A)
        F1 = random.choice(E1)
        E2 = random.choice(A)
        F2 = random.choice(E2)
        E3 = random.choice(A)
        F3 = random.choice(E3)
        Gray1 = F1[0]*0.299 + F1[1]*0.587 + F1[2]*0.114
        Gray2 = F2[0]*0.299 + F2[1]*0.587 + F2[2]*0.114
        Gray3 = F3[0]*0.299 + F3[1]*0.587 + F3[2]*0.114
        B = []
        for i in A:
            for j in i:
                jGray = j[0]*0.299 + j[1]*0.587 + j[2]*0.114
                U = [round(Gray1, 2), round(Gray2, 2), round(Gray3, 2)]
                o = find_nearest(U, jGray)
                if o == round(Gray1, 2):
                    B.append([F1[0],F1[1],F1[2]])
                elif o == round(Gray2, 2):
                    B.append([F2[0],F2[1],F2[2]])
                elif o == round(Gray3, 2):
                    B.append([F3[0],F3[1],F3[2]])
        C = np.array(B)
        D = np.reshape(C,np.shape(A))
        img1 = Image.fromarray(D, 'RGB')
        img1.save('my.tif')
        population['1-R'].iloc[z] = F1[0]
        population['1-B'].iloc[z] = F1[1]
        population['1-G'].iloc[z] = F1[2]
        population['2-R'].iloc[z] = F2[0]
        population['2-B'].iloc[z] = F2[1]
        population['2-G'].iloc[z] = F2[2]
        population['3-R'].iloc[z] = F3[0]
        population['3-B'].iloc[z] = F3[1]
        population['3-G'].iloc[z] = F3[2]
        img_a = cv2.imread(IMAGE)
        img_b = cv2.imread('my.tif')
        PSNR_score = cv2.PSNR(img_a, img_b)
        population['PSNR'].iloc[z] = PSNR_score
        population = population.sort_values(by=['PSNR'], ascending=False)
    return population
```

مرحله Eliminate:

دو ترکیب برتر را جدا می‌کنیم.

	1-R	1-B	1-G	2-R	2-B	2-G	3-R	3-B	3-G	PSNR
2	161	169	158	139	120	126	67	17	116	28.278851
6	158	164	160	63	16	108	138	118	127	27.947832
5	0	0	0	0	0	0	0	0	0	0.000000
1	0	0	0	0	0	0	0	0	0	0.000000
0	0	0	0	0	0	0	0	0	0	0.000000
4	0	0	0	0	0	0	0	0	0	0.000000
7	0	0	0	0	0	0	0	0	0	0.000000
3	0	0	0	0	0	0	0	0	0	0.000000

```
def eliminate(population):
    population = population.sort_values(by=['PSNR'], ascending=False)
    population.iloc[2:8,:] = 0
    return population
```

مرحله **Cross-Over** (تقاطع یا جفت گیری):

به صورت تصادفی دو ردیف اول را با هم دیگر ترکیب می‌کنیم.

	1-R	1-B	1-G	2-R	2-B	2-G	3-R	3-B	3-G	PSNR
2	161	169	158	139	120	126	67	17	116	28.278851
6	158	164	160	63	16	108	138	118	127	27.947832
5	67	17	116	138	118	127	161	169	158	0.000000
1	67	17	116	139	120	126	161	169	158	0.000000
0	139	120	126	63	16	108	138	118	127	0.000000
4	158	164	160	67	17	116	67	17	116	0.000000
7	63	16	108	138	118	127	161	169	158	0.000000
3	139	120	126	63	16	108	63	16	108	0.000000

```
def cross_over(population):
    Max1_1 = population.iloc[1:2,0:3].values.tolist()
    Max1_2 = population.iloc[0:1,0:3].values.tolist()
    Max1_3 = population.iloc[1:2,3:6].values.tolist()
    Max2_1 = population.iloc[0:1,3:6].values.tolist()
    Max2_2 = population.iloc[1:2,6:9].values.tolist()
    Max2_3 = population.iloc[0:1,6:9].values.tolist()
    Gens = [Max1_1, Max1_2, Max1_3, Max2_1, Max2_2, Max2_3]
    #1
    population.iloc[2:3,0:3] = random.choice(Gens)
    population.iloc[2:3,3:6] = random.choice(Gens)
    population.iloc[2:3,6:9] = random.choice(Gens)
    #2
    population.iloc[3:4,0:3] = random.choice(Gens)
    population.iloc[3:4,3:6] = random.choice(Gens)
    population.iloc[3:4,6:9] = random.choice(Gens)
    #3
    population.iloc[4:5,0:3] = random.choice(Gens)
    population.iloc[4:5,3:6] = random.choice(Gens)
    population.iloc[4:5,6:9] = random.choice(Gens)
    #4
    population.iloc[5:6,0:3] = random.choice(Gens)
    population.iloc[5:6,3:6] = random.choice(Gens)
    population.iloc[5:6,6:9] = random.choice(Gens)
    #5
    population.iloc[6:7,0:3] = random.choice(Gens)
    population.iloc[6:7,3:6] = random.choice(Gens)
    population.iloc[6:7,6:9] = random.choice(Gens)
    #6
    population.iloc[7:8,0:3] = random.choice(Gens)
    population.iloc[7:8,3:6] = random.choice(Gens)
    population.iloc[7:8,6:9] = random.choice(Gens)
    #7
    population.iloc[8:9,0:3] = random.choice(Gens)
    population.iloc[8:9,3:6] = random.choice(Gens)
    population.iloc[8:9,6:9] = random.choice(Gens)
    #8
    population.iloc[2:3,0:3] = random.choice(Gens)
    population.iloc[2:3,3:6] = random.choice(Gens)
    population.iloc[2:3,6:9] = random.choice(Gens)
    return population
```

مرحله ارزیابی یا Fitness:

در این ردیف PSNR ردیف‌های فرزند (حاصل کراس‌اور دو ردیف اول) را مشخص می‌کنیم.

```
def fitness(IMAGE, population):

    for z in population.index:

        im = Image.open(IMAGE)
        A = np.squeeze(im, axis=None)

        Gray1 = population['1-R'].iloc[z]*0.299 + population['1-
B'].iloc[z]*0.587 + population['1-G'].iloc[z]*0.114
        Gray2 = population['2-R'].iloc[z]*0.299 + population['2-
B'].iloc[z]*0.587 + population['2-G'].iloc[z]*0.114
        Gray3 = population['3-R'].iloc[z]*0.299 + population['3-
B'].iloc[z]*0.587 + population['3-G'].iloc[z]*0.114
        B = []
        U = []
        for i in A:
            for j in i:
                jGray = j[0]*0.299 + j[1]*0.587 + j[2]*0.114
                U = [round(Gray1, 2), round(Gray2, 2), round(Gray3, 2)]
                o = find_nearest(U, jGray)
                if o == round(Gray1, 2):
                    B.append([population['1-R'].iloc[z], population['1-
B'].iloc[z], population['1-G'].iloc[z]])
                elif o == round(Gray2, 2):
                    B.append([population['2-R'].iloc[z], population['2-
B'].iloc[z], population['2-G'].iloc[z]])
                elif o == round(Gray3, 2):
                    B.append([population['3-R'].iloc[z], population['3-
B'].iloc[z], population['3-G'].iloc[z]])
        C = np.array(B)
        D = np.reshape(C,np.shape(A))

        img1 = Image.fromarray(D, 'RGB')
        img1.save('my.tif')

        img_a = cv2.imread(IMAGE)
        img_b = cv2.imread('my.tif')
        PSNR_score = cv2.PSNR(img_a, img_b)
        population['PSNR'].iloc[z] = PSNR_score

    population = population.sort_values(by=['PSNR'], ascending=False)
    return population
```

الگوریتم کلی:

مقدار stopping_criterion زمان توقف الگوریتم را مشخص می‌کند که هرچه عدد بالاتری باشد الگوریتم نتیجه‌ی بهتری خواهد داشت، همچنین شرطی تعریف کرده ایم که در صورت نرسیدن به جواب بهتر از 10 الگوریتم دوباره از نو اجرا شود.

```
stopping_criterion = 5
generation = initial_population('Im001_1.tif')
j = 1
while j < stopping_criterion:
    generation = eliminate(generation)
    generation = cross_over(generation)
    generation = fit('Im001_1.tif',generation)
    if j + 1 == stopping_criterion and generation['PSNR'].iloc[0] < 10:
        generation = initial_population('Im001_1.tif')
        j = 1
    j = j + 1
generation
```

پیوست: کدهای پایتون

کد 4 رنگ:

```
import numpy as np
import PIL
from PIL import Image
import random
import cv2

im = Image.open('Im004_1.tif')

A = np.squeeze(im, axis=None)

E1 = random.choice(A)
F1 = random.choice(E1)
E2 = random.choice(A)
F2 = random.choice(E2)
E3 = random.choice(A)
F3 = random.choice(E3)
E4 = random.choice(A)
F4 = random.choice(E4)

Gray1 = F1[0]*0.299 + F1[1]*0.587 + F1[2]*0.114
Gray2 = F2[0]*0.299 + F2[1]*0.587 + F2[2]*0.114
Gray3 = F3[0]*0.299 + F3[1]*0.587 + F3[2]*0.114
Gray4 = F4[0]*0.299 + F4[1]*0.587 + F4[2]*0.114

U = [round(Gray1, 2), round(Gray2, 2), round(Gray3, 2), round(Gray4, 2)]

def find_nearest(array, value):
    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()
    return array[idx]

B = []
for i in A:
    for j in i:
        jGray = j[0]*0.299 + j[1]*0.587 + j[2]*0.114
        o = find_nearest(U, jGray)
        if o == round(Gray1, 2):
            B.append([F1[0],F1[1],F1[2]])
        elif o == round(Gray2, 2):
            B.append([F2[0],F2[1],F2[2]])
        elif o == round(Gray3, 2):
            B.append([F3[0],F3[1],F3[2]])
        elif o == round(Gray4, 2):
            B.append([F4[0],F4[1],F4[2]])

C = np.array(B)
```

```
D = np.reshape(C,np.shape(A))

img1 = Image.fromarray(D, 'RGB')
img1.show()
img1.save('4-4.tif')

img_a = cv2.imread('Im004_1.tif')
img_b = cv2.imread('4-4.tif')
PSNR_score = cv2.PSNR(img_a, img_b)
PSNR_score
```

کد 5 رنگ:

```
import numpy as np
import PIL
from PIL import Image
import random
import cv2

im = Image.open('Im004_1.tif')

A = np.squeeze(im, axis=None)

E1 = random.choice(A)
F1 = random.choice(E1)
E2 = random.choice(A)
F2 = random.choice(E2)
E3 = random.choice(A)
F3 = random.choice(E3)
E4 = random.choice(A)
F4 = random.choice(E4)
E5 = random.choice(A)
F5 = random.choice(E5)

Gray1 = F1[0]*0.299 + F1[1]*0.587 + F1[2]*0.114
Gray2 = F2[0]*0.299 + F2[1]*0.587 + F2[2]*0.114
Gray3 = F3[0]*0.299 + F3[1]*0.587 + F3[2]*0.114
Gray4 = F4[0]*0.299 + F4[1]*0.587 + F4[2]*0.114
Gray5 = F5[0]*0.299 + F5[1]*0.587 + F5[2]*0.114

U = [round(Gray1, 2), round(Gray2, 2), round(Gray3, 2), round(Gray4, 2),
round(Gray5, 2)]

def find_nearest(array, value):
    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()
    return array[idx]
B = []
for i in A:
    for j in i:
        jGray = j[0]*0.299 + j[1]*0.587 + j[2]*0.114
```

```
        o = find_nearest(U, jGray)
        if o == round(Gray1, 2):
            B.append([F1[0],F1[1],F1[2]])
        elif o == round(Gray2, 2):
            B.append([F2[0],F2[1],F2[2]])
        elif o == round(Gray3, 2):
            B.append([F3[0],F3[1],F3[2]])
        elif o == round(Gray4, 2):
            B.append([F4[0],F4[1],F4[2]])
        elif o == round(Gray5, 2):
            B.append([F5[0],F5[1],F5[2]])

C = np.array(B)

D = np.reshape(C,np.shape(A))

img1 = Image.fromarray(D, 'RGB')
img1.show()
img1.save('5-4.tif')

img_a = cv2.imread('Im004_1.tif')
img_b = cv2.imread('5-4.tif')
PSNR_score = cv2.PSNR(img_a, img_b)
PSNR_score
```