

گزارش پروژه درس هوش مصنوعی

ریحانه درفشی - ۶۱۰۳۹۶۰۹۸

صورت مسئله:

در این پروژه به پیاده‌سازی روشی برای قطعه‌بندی تصاویر سلول‌های سرطانی خون می‌پردازیم. تفاوت این روش قطعه‌بندی با سایر روش‌ها، جستجو بر روی رنگ‌ها به جای روش‌های معمول «پیدا کردن مرزها» یا روش‌های یادگیری ماشین است که مستقیماً بر روی تصویر کار می‌کنند. در این روش، هر مجموعه رنگ انتخاب‌شده‌ای دارای مقداری برازشی هستند که میزان «خوب بودن» آن مجموعه رنگ را مشخص می‌کند.

الگوریتم:

برای رسیدن به مجموعه رنگ‌های مناسب برای هر تصویر، از الگوریتم ممیتیک استفاده کردم. تفاوت الگوریتم ممیتیک با الگوریتم ژنتیک در محلی حاصل می‌شود که برای هر کروموزوم در الگوریتم ممیتیک، مقدار برازش، بهترین جواب ممکن در همسایگی آن کروموزوم است. این کار باعث بالا رفتن سرعت پردازش می‌شود، به گونه‌ای که با جمعیتی ۵۰ عضوی، تنها با پردازش ۱۰ نسل توانستم به جوابهای قابل قبولی برسم.

```
def memetic(linear_image, clusters):  
    population = [  
        Individual([[random.randint(0, 255) for _ in range(3)] for __ in  
range(clusters)])  
        for _ in range(POPULATION_SIZE)  
    ]  
    for individual in population:  
        individual.calculate_fitness(linear_image)  
  
    for iteration in range(ITERATIONS):
```

```

print(iteration, ": ", population[0].fitness, end="\r")
new_generation = []
for i in range(POPULATION_SIZE):
    for j in range(i + 1, POPULATION_SIZE):
        if random.random() < Crossover_Probability:
            first_child, second_child = crossover(population[i].best_neighbour,
population[j].best_neighbour)
            new_generation.append(Individual(first_child))
            new_generation.append(Individual(second_child))
    for individual in new_generation:
        individual.calculate_fitness(linear_image)

population.extend(new_generation)
population.sort(key=lambda x: x.fitness)
population = population[:POPULATION_SIZE]

return population[0].best_neighbour, population[0].fitness

```

تابع برازش:

یکی از توابع برازش قابل استفاده در این روش، مجموع فاصله اقلیدسی رنگ نقاط تصویر با رنگ‌های نزدیک به خود است. بدین صورت که پس از اعمال مجموعه رنگ‌ها روی تصویر (اینگونه که هر پیکسل به نزدیکترین رنگ تخصیص داده می‌شود)، مجموع فواصل اقلیدسی نقاط را حساب می‌کنیم و هر چه این مقدار کمتر باشد، نتیجه مطلوب‌تر است. اما در این روش مشکلی نیز وجود دارد؛ در حالتی که از سه رنگ فقط دو رنگ مورد استفاده قرار گیرد نیز ممکن است نتیجه مطلوبی داشته باشیم، در صورتی که این نتیجه مطلوب ما نیست. پس می‌توان با ضرب کردن تعداد رنگ‌های استفاده نشده به اضافه یک، در مقدار برازش باعث زیاد شدن برازش نمونه‌های ناکامل شد.

(برای مثال اگر از ۵ رنگ فقط ۴ رنگ استفاده شود، مقدار برآزش را ضرب در ۲ می‌کنیم و اگر همه رنگ‌ها استفاده شوند با ضرب کردن ۱ در جواب تغییری در مقدار برآزش نخواهیم داشت)

```
def fitness(target, colors):
    res = 0
    used_colors = [0 for _ in colors]
    for pixel in target:
        color_index = np.argmin([np.linalg.norm(pixel - color) for color in colors])
        res += np.linalg.norm(pixel - colors[color_index])
        used_colors[color_index] = 1
    mult_factor = (len(colors) - sum(used_colors) + 1)
    return res * mult_factor
```

تولید مثل:

برای تولید مثل کافی است با هر کروموزوم مانند آرایه‌ای از اشیاء برخورد کنیم و دو فرزند را با جابجا کردن تعدادی از رنگ‌های پدر و مادر تولید کنیم. (به جای خود پدر و مادر از بهینه محلی همسایگی پدر و مادر استفاده می‌کنیم). بدین ترتیب، هم رنگ‌ها به مرور زمان در همسایگی خود تغییر می‌کنند، هم با ترکیب کردن رنگ‌هایی که فاصله کمی با نقاط داشته‌اند مجموعه‌هایی با برآزش کمتر تولید می‌شود.

همسایگی کروموزوم‌ها:

یکی از تصمیم‌های اصلی چگونگی انتخاب همسایگی کروموزوم‌ها بود. برای مثال اگر مجموعه‌ای با سه رنگ داشته باشیم، عملاً در فضایی ۹ بعدی در بازه ۰ تا ۲۵۵ قرار داریم. حال اگر همسایگی هر بعد را بازه به شعاع یک آن نیز می‌دیدیم، هر کروموزوم به طور میانگین ۳ به توان ۹ همسایه داشت که عددی بسیار بزرگ و عملاً غیر قابل محاسبه

برای برآزش است. پس به جای گرفتن همه همسایه‌ها، اقدام به انتخاب ۱۰ همسایه تصادفی در بازه‌ای به شعاع مشخص دور کروموزوم کردم و نتیجه مطلوبی هم از نظر سرعت و هم از نظر دقت گرفتم.

```
def get_neighbour_color(color, range):
```

```
    blue = max(0, min(255, color[0] + random.randint(-range, range)))
```

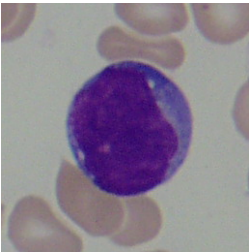
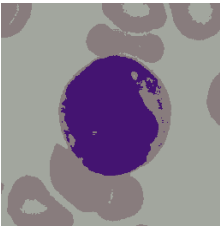
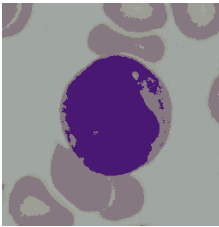
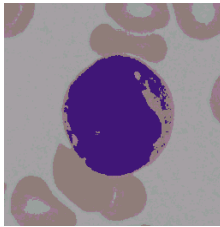
```
    green = max(0, min(255, color[1] + random.randint(-range, range)))
```

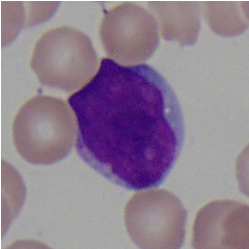
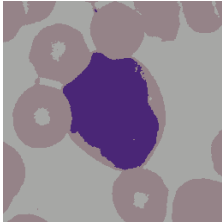
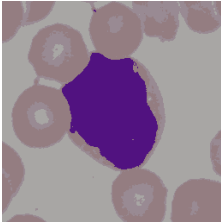
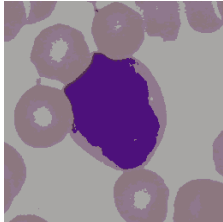
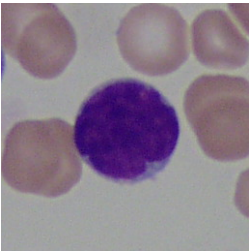
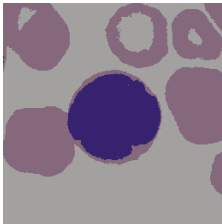
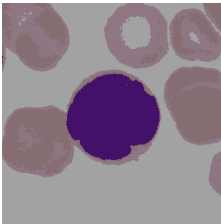
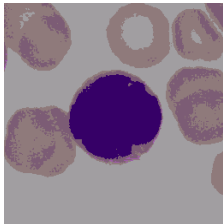
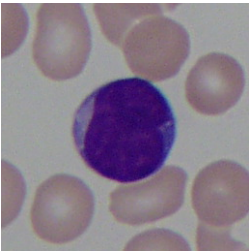
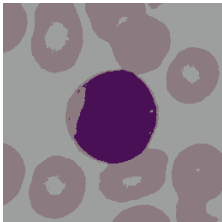
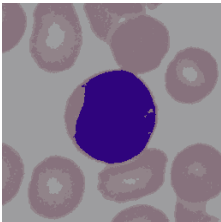
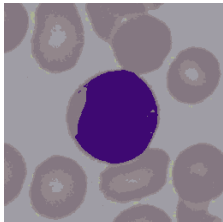
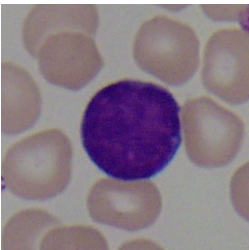
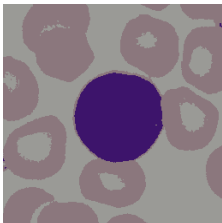
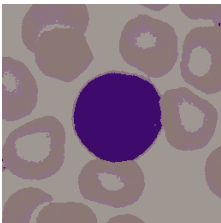
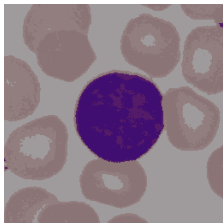
```
    red = max(0, min(255, color[2] + random.randint(-range, range)))
```

```
    return np.array([blue, green, red], dtype=np.uint8)
```

```
def get_neighbour(colors, range=10):
```

```
    return np.array(list(map(lambda x: get_neighbour_color(x, range), colors)))
```

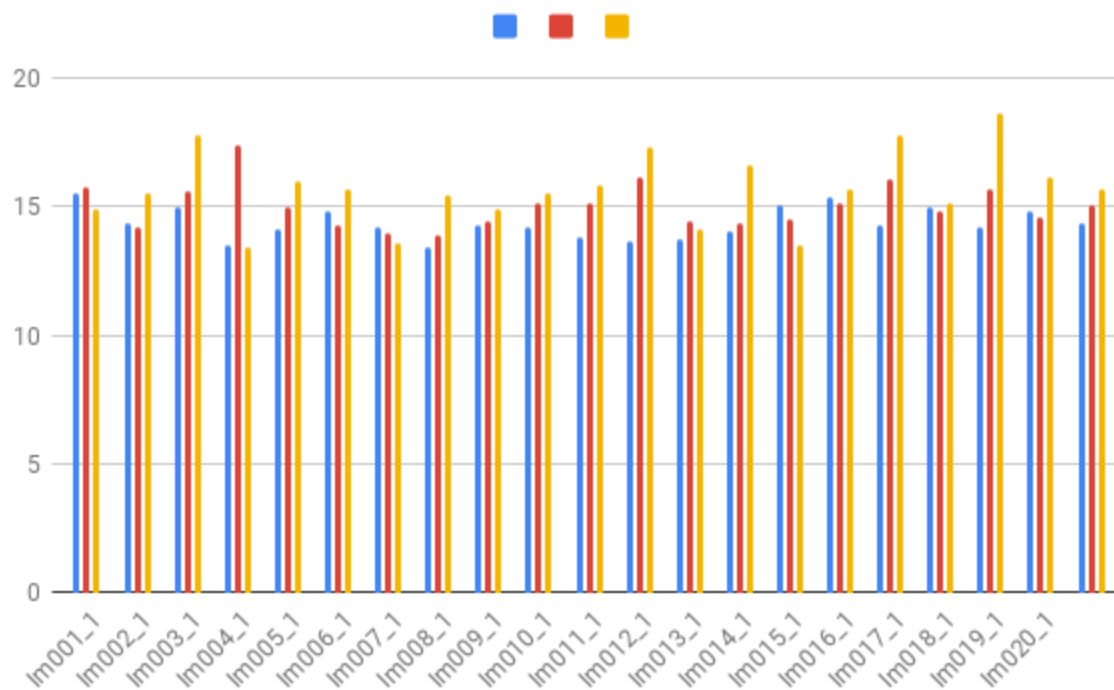
Photo number	original	Segmented with 3 colors	Segmented with 4 colors	Segmented with 5
1				

2				
3				
4				
5				

نتایج:

الگوریتم پیاده‌سازی شده را برای ۲۰ تصویر اجرا کردم و برای ۵ تصویر خروجی تصویرهای تولید شده در جدول بالا قابل مشاهده است. همچنین کل نتیجه، اعم از مقدار تابع برازش و مقدار PSNR در جداول زیر آمده‌اند.

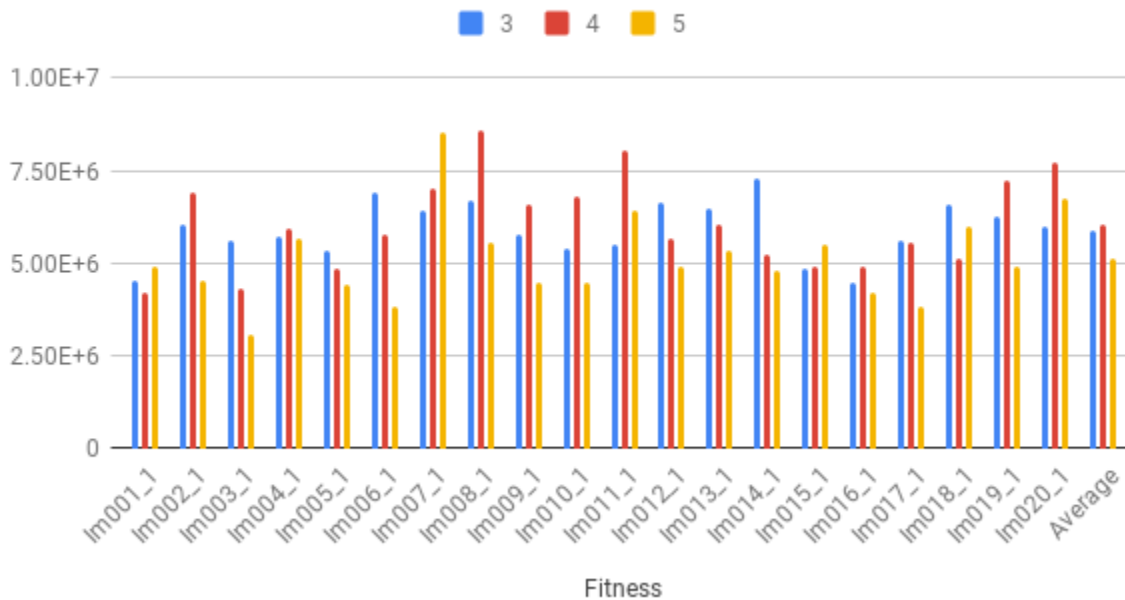
PSNR	3	4	5
Im001_1	15.54140594	15.7640645	14.94164327
Im002_1	14.35728901	14.21911197	15.5018028
Im003_1	14.95128637	15.6267213	17.78460728
Im004_1	13.5380716	17.38259943	13.41301753
Im005_1	14.11128483	14.98609102	15.9553282
Im006_1	14.83384497	14.26964998	15.70497979
Im007_1	14.21607897	13.94263191	13.60728109
Im008_1	13.40559972	13.92848738	15.41662646
Im009_1	14.30571264	14.4716288	14.90465056
Im010_1	14.2336613	15.13118648	15.50421284
Im011_1	13.78005379	15.16367854	15.82942604
Im012_1	13.62865708	16.17811872	17.27741264
Im013_1	13.73046618	14.42753057	14.14628729
Im014_1	14.0859432	14.39353808	16.62273549
Im015_1	15.0636308	14.49708348	13.48433702
Im016_1	15.34247703	15.14955605	15.66427666
Im017_1	14.26206749	16.07681534	17.78766325
Im018_1	14.97514227	14.81246656	15.104062
Im019_1	14.22765067	15.64951005	18.59975571
Im020_1	14.84025134	14.61880391	16.12794443
Average	14.37152876	15.0344637	15.66890252



Fitness	3	4	5
Im001_1	4523736.749	4220825.916	4884060.786
Im002_1	6039769.681	6899875.798	4546603.705
Im003_1	5623537.518	4334518.995	3048561.682
Im004_1	5730002.894	5949896.915	5667668.885
Im005_1	5326286.434	4857714.567	4449977.093
Im006_1	6930846.991	5778658.315	3826175.525
Im007_1	6420013.984	6998965.647	8519535.458
Im008_1	6670249.056	8596551.057	5579161.972
Im009_1	5754364.681	6584463.133	4471123.103
Im010_1	5418432.756	6796828.79	4473512.358
Im011_1	5514175.058	8052108.964	6425470.475
Im012_1	6663937.686	5651018.193	4895122.088

Im013_1	6483250.25	6019765.971	5325430.195
Im014_1	7287932.081	5209025.419	4784757.306
Im015_1	4856579.825	4931277.709	5504187.702
Im016_1	4483849.876	4892267.878	4205956.52
Im017_1	5620413.614	5556044.363	3813099.822
Im018_1	6594267.519	5117359.594	5992058.846
Im019_1	6244270.283	7241090.853	4931183.204
Im020_1	5992811.136	7700402.467	6755566.678
Average	5908936.404	6069433.027	5104960.67

3, 4 and 5



colors	3	4	5
Im001_1	125 124 136 113 20 68 158 166 162	128 120 134 147 158 156 161 166 160 114 24 72	119 22 64 164 160 165 157 160 153 121 125 143 143 121 155
Im002_1	165 167 166 136 131 149 118 38 74	147 138 153 164 166 168 128 18 81 129 125 148	84 46 83 164 166 167 123 17 77 127 119 140 141 120 143
Im003_1	110 34 54 124 104 134 160 161 162	163 162 163 114 109 134 103 17 67 131 119 142	103 0 54 117 92 124 123 122 143 169 90 158 161 158 161
Im004_1	86 16 73 127 122 139 158 159 157	137 131 145 125 114 135 125 4 47 159 157 157	148 178 171 114 9 62 165 157 159 139 134 146 129 119 132
Im005_1	125 119 139 150 154 154	109 11 62 132 115 141	118 116 138 104 27 83

	107 20 61	146 151 159 116 119 139	113 13 64 156 155 159 127 126 147
--	-----------	----------------------------	---

رنگ‌های تولید شده برای عکس‌های موجود در جدول اول به فرمت BGR

References

- Das, Swagatam, and Amit Konar. "Automatic image pixel clustering with an improved differential evolution." *Applied Soft Computing*, vol. 9, no. 1, January 2009, pp. 226-236.
- Falco, De, et al. "Facing classification problems with Particle Swarm Optimization." *Applied Soft Computing*, vol. 7, no. 3, June 2007, pp. 652-658.
- Krishna Gopal Dhal, et al. "Acute lymphoblastic leukemia image segmentation driven by stochastic fractal search." 13 January 2020.