

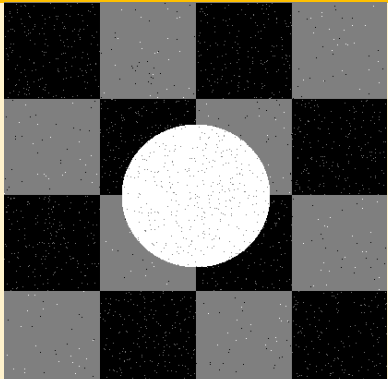
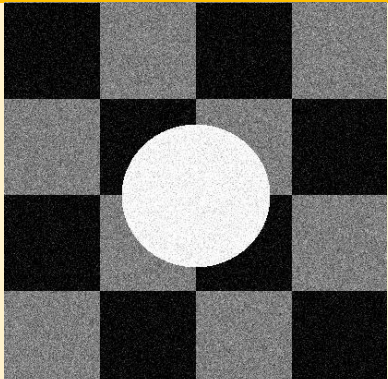
تمرین سری اول ، درس PGM		
dalirani.1373@gmail.com	۹۶۱۳۱۱۲۵	فرهاد دلیرانی

\*در گزارش، تصویرها را با اندازه‌ی کوچک‌تری قرار داده‌ام. برای دیدن تصویرها در **اندازه‌ی اصلی** به پوشه‌ی **output** مراجعه فرمایید.

## بخش اول

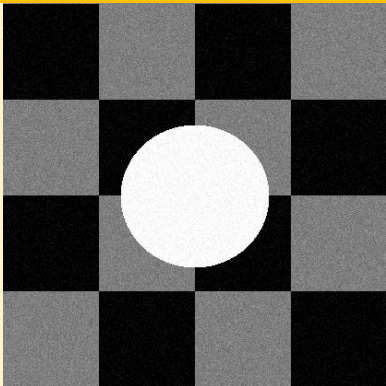
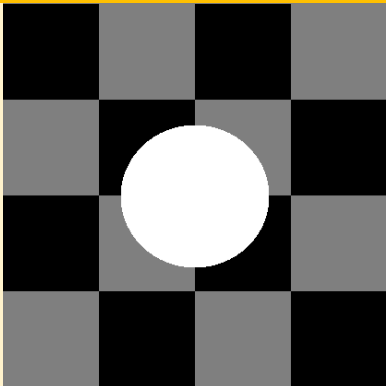
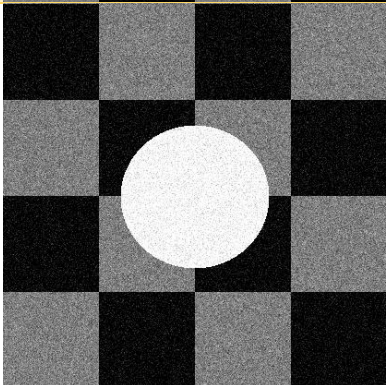
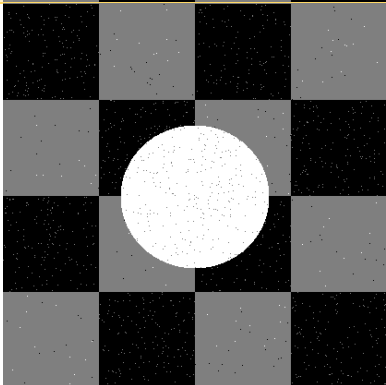
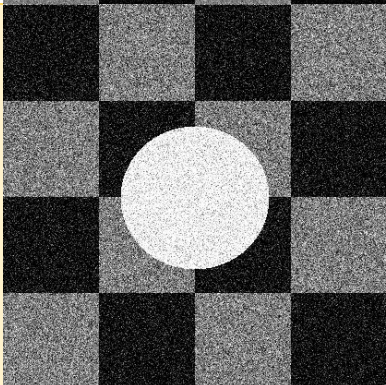
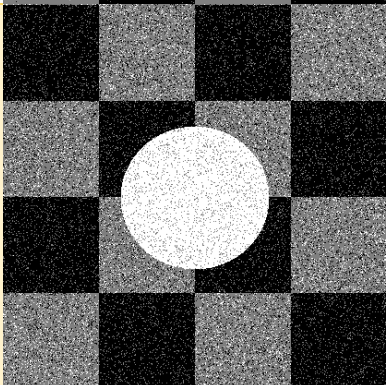
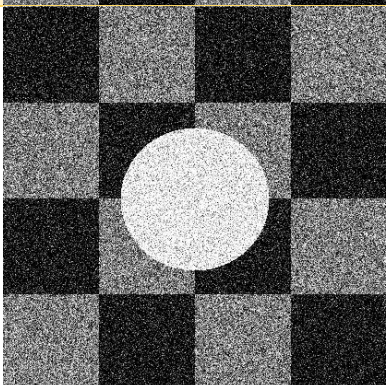
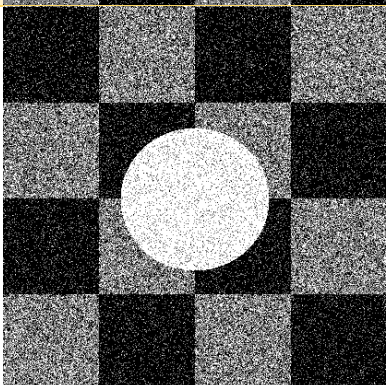
### قسمت الف)

کد این قسمت در **p1a.py** قرار دارد. ابتدا تصویر اصلی را می‌خوانم و آن را به یک تصویر یک کاناله تبدیل می‌کنم. سپس به آن نویز گاوسی با میانگین صفر و انحراف معیار ۰,۰۱ اضافه می‌کنم. از اینجا به بعد دیگر از تصویر بدون نویز هیچ استفاده‌ای نمی‌کنم. فرض شده است برای هر کلاس  $p(\text{Intensity}|\text{Label})$  یک توزیع گاوسی دارد. در تصویر نویزی، سه ناحیه‌ی  $۵۰ \times ۵۰$  پیکسل مربوط به سه کلاس مختلف را، انتخاب می‌کنم سپس از شدت‌روشنایی‌های این سه ناحیه برای تخمین میانگین و انحراف از معیار  $p(\text{Intensity}|\text{Label})$  استفاده می‌کنم. در ادامه‌ی کد، میزان  $p(\text{Label}|\text{Intensity})$  را برای تک تک پیکسل‌های تصویر، به ازای ۳ کلاس مختلف محاسبه می‌کنم و در آخر کلاسی را به عنوان برچسب پیکسل انتخاب می‌کنم که  $p(\text{Label}|\text{Intensity})$  بزرگ‌تری دارد. در زیر خروجی‌های کد را مشاهده می‌کنید.

دقت قطعه‌بندی	تصویر نویزی بعد از قطعه‌بندی	تصویر نویزی (میانگین ۰، واریانس ۰,۰۱)
98.82%		

### قسمت ب)

کد این قسمت در **p1b.py** قرار دارد. در این قسمت از کد قسمت قبل استفاده کرده‌ام با این تفاوت که از سه مقدار مختلف واریانس برای ایجاد تصویر نویزی استفاده می‌کنم. در زیر خروجی‌های کد را مشاهده می‌کنید:

واریانس	تصویر نویزی	تصویر نویزی بعد از قطعه‌بندی	دقت قطعه‌بندی
0.002			100%
0.009			99.13%
0.04			84.51%
0.08			70.40%

هرچه میزان نویز (واریانس نویز گاوسی) افزایش می‌یابد، دقت قطعه‌بند بیزساده کاهش می‌یابد.

### قسمت ج)

کد این قسمت در `simulated_annealing.py` ، `calculate_potential.py` و `p1c.py` قرار دارد.

**Calculate\_potential.py**: در این فایل دو تابع وجود دارد که از آن‌ها براساس تابع زیر، برای محاسبه‌ی پتانسیل تصویر برچسب

زده شده استفاده می‌کنیم:

$$U(\omega) = \sum_s \left( \log \left( \sqrt{2\pi\sigma_{\omega_s}} \right) + \frac{(f_s - \mu_{\omega_s})^2}{2\sigma_{\omega_s}^2} \right) + \sum_{s,r} \beta \delta(\omega_s, \omega_r)$$

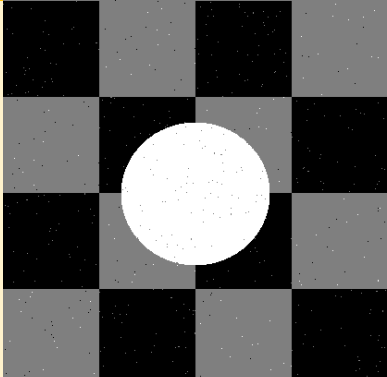
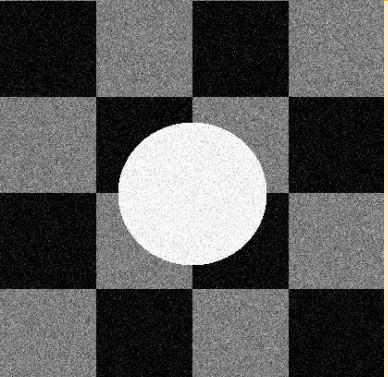
که دلتا در تابع بالا برابر است با:

$$\beta \delta(\omega_i, \omega_j) = \begin{cases} -\beta & \text{if } \omega_i = \omega_j \\ +\beta & \text{if } \omega_i \neq \omega_j \end{cases}$$

تابع اول در این فایل، پتانسیل تصویر را با توجه به شدت روشنایی و برچسب همسایگان به‌دست می‌آورد. ولی تابع دوم برای محاسبه‌ی پتانسیل، پتانسیل را از اول حساب نمی‌کند. در صورتی که پتانسیل یک تصویر برچسب زده شده را داشته باشیم و یکی از برچسب‌های آن را تغییر بدهیم، این تابع، پتانسیل جدید را با تغییر پتانسیل قبلی و با محاسبات بسیار کمتر به دست می‌آورد.

**Simulated\_annealing.py**: در تابع درون این فایل، تصویر نویزی، میانگین و انحراف از معیار سه کلاس‌ها، نوع همسایگی (چهارتایی، هشت‌تایی)، مقدار  $\beta$ ، دمای اولیه، حداکثر تعداد ایپاک‌ها و ضربی که هر ایپاک در دما ضرب می‌کنیم تا آن را کاهش دهیم را به عنوان آرگومان ورودی می‌گیریم. در ابتدا تمام پیکسل‌ها را به صورت تصادفی برچسب می‌زنیم. سپس پتانسیل تصویر را حساب می‌کنیم. به تعداد حداکثر ایپاک‌ها، هر بار به صورت تصادفی یک پیکسل انتخاب می‌شود و برچسبش را به صورت تصادفی به کلاس جدیدی تغییر می‌کند، سپس برای حالت جدید پتانسیل حساب می‌شود و بعد  $\Delta U$  محاسبه می‌شود، در صورتی که  $\Delta U$  صفر و یا منفی باشد تغییرها تأثیر داده می‌شود. اگر  $\Delta U$  بزرگ‌تر از صفر باشد، با احتمال  $\frac{-\Delta U}{Temperature}$  تغییر پذیرفته می‌شود. در پایان هر ایپاک دما به‌روزرسانی می‌شود و کاهش می‌یابد. بدین‌گونه با تغییر کلاس پیکسل‌ها و کمینه کردن  $\Delta U$  قطعه‌بندی مناسب را پیدا می‌کنیم. این تابع با کیفیت بسیار خوبی نوشته شده است به‌طوری که برای ۳ میلیون ایپاک در کمتر از ۳ دقیقه  $\Delta U$  را کمینه می‌کند.

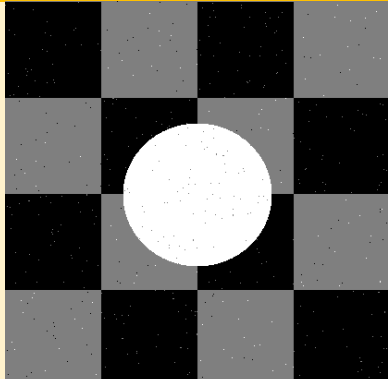
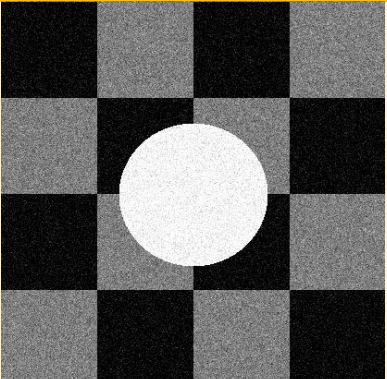
**P1c.py**: در این کد، ابتدا تصویر بدون نویز را می‌خوانیم و نویز گاوسی به آن می‌افزایم، سپس سه ناحیه  $50 \times 50$  از تصویر که هر کدام در یک کلاس متفاوت قرار دارند انتخاب می‌شوند. از این سه ناحیه برای تخمین میانگین و انحراف از معیار  $p(Intensity|Label)$  کلاس‌های مختلف که فرض شده است که توزیع نرمال دارند استفاده می‌کنیم. در ادامه تابع `Simulated Annealing` را با دمای اولیه‌ی ۴۰۰۰، حداکثر ایپاک برابر با ۲۰۰۰۰۰۰، همسایگی ۴‌تایی، ضربی کاهش دما در هر ایپاک برابر با ۰.۹۵ و بتای برابر با ۱، فرامی‌خوانیم و قطعه‌بندی مناسب را به دست می‌آوریم. در انتها دقت دسته‌بندی را محاسبه می‌کنیم و تصویر نویزی و قطعه‌بندی شده را نمایش می‌دهیم. در زیر خروجی کد را مشاهده می‌کنید:

دقت بیژ ساده	دقت قطعه‌بندی MRF	تصویر نویزی بعد از قطعه‌بندی MRF	تصویر نویزی (میانگین ۰، واریانس ۰،۰۱)
98.82%	99.68%		

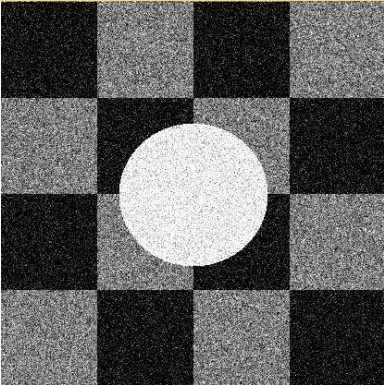
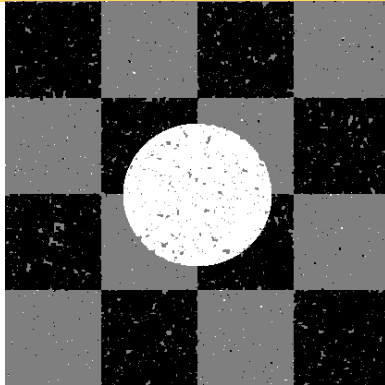
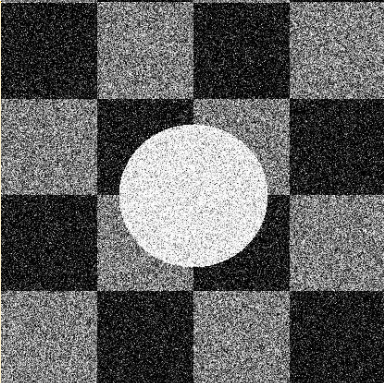
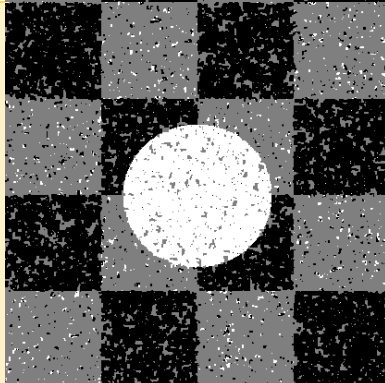
همین طوری که در جدول بالا دیده می‌شود، قطعه‌بندی توسط MRF دقتی برابر با ۹۹،۶۸٪ دارد در حالی که قطعه‌بندی همین تصویر نویزی با بیژ ساده دقتی برابر با ۹۸،۸۲٪ داشت. همین طور که مشاهده می‌شود MRF عملکرد بهتری دارد. البته واریانس ۰،۰۱ نویز کمی ایجاد می‌کند و بهتر بودن روش MRF از بیژ ساده به صورت قابل لمسی قابل مشاهده نیست، در قسمت‌های بعدی تفاوت بین دو روش به خوبی قابل مشاهده خواهد بود.

#### قسمت د)

کد این قسمت در p1d.py قرار دارد، کد این قسمت مانند قسمت قبلی است با این تفاوت که با واریانس‌های متفاوت تصویر را نویزی می‌کنیم در زیر خروجی کد را مشاهده می‌کنید:

دقت قطعه‌بندی بیژ ساده	دقت قطعه‌بندی MRF	تصویر نویزی بعد از قطعه‌بندی	تصویر نویزی	واریانس
99.13%	99.70%			0.009



0.04			97.44%	84.51%
0.08			89.50%	70.40%

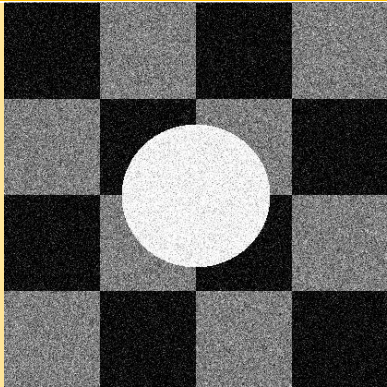
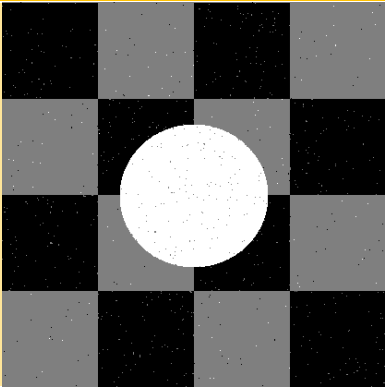
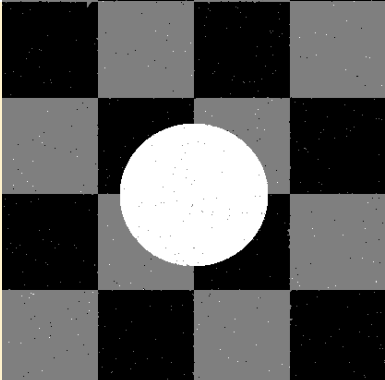
در زیر خلاصه‌ای از دقت دو روش MRF و بیز ساده را برای واریانس‌های مختلف مشاهده می‌کنید:

واریانس	دقت قطعه‌بندی MRF	دقت قطعه‌بندی بیز ساده
0.009	99.70%	99.13%
0.04	97.44%	84.51%
0.08	89.50%	70.40%

با افزایش نویز دقت قطعه‌بند MRF کاهش می‌یابد ولی افت در دقت، نسبت به بیز ساده کمتر است. همین‌طور که مشاهده می‌شود با افزایش میزان نویز عملکرد بیز ساده بسیار افت می‌کند در حالی که با در نظر گرفتن همسایگی‌ها در MRF به دقت بیشتری دست پیدا می‌کنیم و هر چه میزان نویز بیشتر می‌شود تفاوت عملکرد بیز ساده و MRF بیشتر دیده می‌شود به‌طوری‌که در نویز با واریانس 0.08، MRF در حدود بیست درصد نسبت به بیز ساده بهتر عمل کرده است. MRF به دلیل در نظر گرفتن همسایگان مقاومت بیشتر نسبت به نویز دارد.

#### قسمت ه)

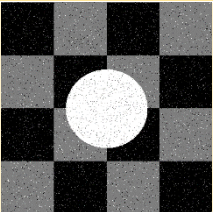
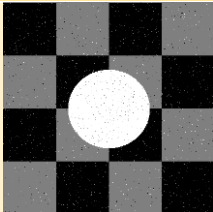
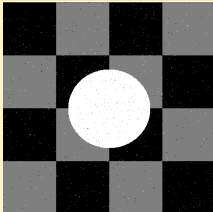
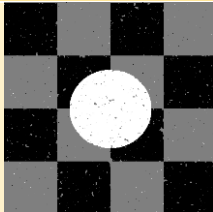
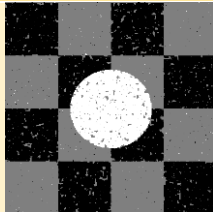
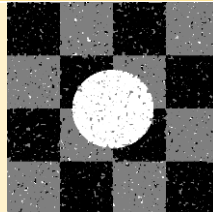
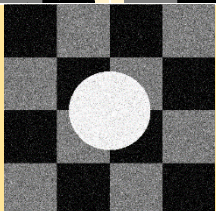
کد این قسمت در p1e.py قرار دارد. این کد مانند کد p1c.py است و از همان پارامترها استفاده شده است با این تفاوت که به تابع Simulated Annealing و Potential که نوشته‌ام به‌جای آرگمان 'four' آرگمان 'eight' را پاس می‌کنم به همین دلیل کد را دوباره توضیح نمی‌دهم. در زیر خروجی کد را مشاهده می‌کنید:

تصویر نویزی (واریانس ۰,۰۲)	تصویر قطعه‌بندی شده توسط MRF	دقت	
		99.42%	همسایگی چهارتایی
		99.62%	همسایگی هشت‌تایی

همین‌طور که در نتیجه‌ی آزمایش دیده می‌شود، MRF با هشت همسایگی دقت بیشتر نسبت به MRF با همسایگی چهار داشته است.

#### قسمت و)

کد این قسمت در `p1f.py` قرار دارد. کد این قسمت مانند قسمت ج است و از همان پارامترها استفاده می‌کند فقط به جای  $\beta$  برابر یک از  $\beta$ های مختلفی استفاده می‌کند. در زیر نتیجه‌ی کد را به ازای  $\beta$ های مختلف مشاهده می‌کنید:

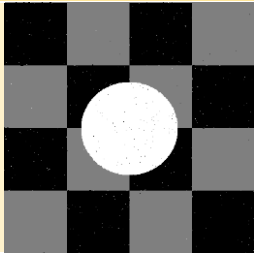
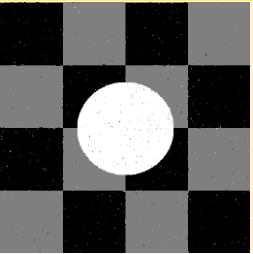
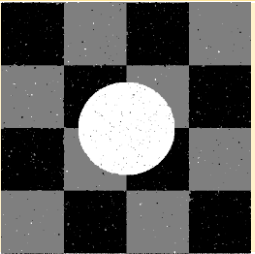
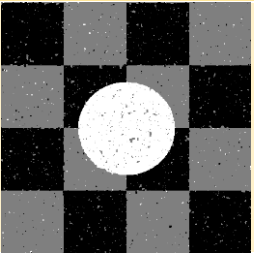
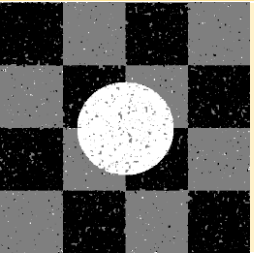
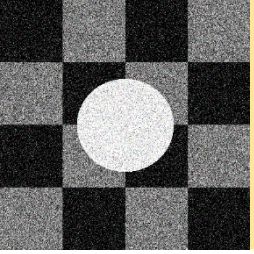
0.1	0.5	1	2	5	10	$\beta$
95.71%	98.59%	99.49%	98.72%	96.40%	94.84%	دقت
						تصویر قطعه‌بندی شده با MRF
						تصویر با نویز گاوسی (۰,۰۱, ۰)

**نکته:** همین‌طور که در ابتدا گزارش اشاره شد، برای دیدن تصویرهای درون گزارش در اندازه‌ی واقعی می‌توانید به پوشه‌ی `output` مراجعه فرمایید.

بهترین عملکرد را  $\beta$  برابر با ۱ دارد که با آن به دقت ۹۹,۴۹٪ در قطعه‌بندی می‌رسیم. همین‌طور که مشاهده می‌شود وقتی  $\beta$  کم است دقت کم است با زیاد کردن  $\beta$  دقت افزایش می‌یابد و بعد از افزایش  $\beta$  از یک حدی به بعد دقت کاهش می‌یابد. وقتی  $\beta$  کم است تاثیر در نظر گرفتن همسایگی کم است و اینگونه به نظر می‌رسد که همسایگی را در نظر نمی‌گیریم، به همین دلیل وقتی  $\beta$  کم است دقت پایین است، با افزایش  $\beta$  تاثیر همسایگی در مدل بیشتر و بیشتر می‌شود و این باعث افزایش دقت می‌شود ولی بعد از افزایش از حدی، دقت افت می‌کند زیرا همین‌طور که در تصویر دیده می‌شود نویزها پخش می‌شوند و بسیار بیشتر از  $P(\text{Intensity}|\text{Label})$  در نظر گرفته می‌شوند. به همین دلیل  $\beta$  نباید مقدار زیاد و یا کمی داشته باشد و یک حالت میانه دارد.

### قسمت ز)

کد این قسمت در `p1g.py` قرار دارد. کد این قسمت مشابه با قسمت ج و از همان پارامترها استفاده شده است با این تفاوت که به جای استفاده از تابع `simulated_annealing` از تابع `simulated_annealing_controlled_initial` استفاده می‌کنیم. این تابع مانند تابع `simulated_annealing` است با این تفاوت که دو آرگمان دیگر هم می‌گیرد، آرگمان اضافی اول، تصویر بدون نویز است که برچسب صحیح هر پیکسل در آن مشخص است، آرگمان جدید دوم، یک احتمال بین ۰ تا ۱ است، اسم این احتمال را  $\Omega$  می‌گذاریم. در تابع جدید مانند تابع قبل عمل می‌کنیم و فقط در قسمت انتخاب تصادفی برچسب‌های اولیه متفاوت عمل می‌کنیم. در تابع جدید ابتدا به صورت تصادفی پیکسل‌ها را برچسب می‌زنیم سپس برای هر پیکسل با احتمال داده شده در آرگمان جدید دوم، تصمیم می‌گیریم که از برچسب تصادفی برای پیکسل استفاده کنیم یا برچسب تصادفی را با برچسب واقعی پیکسل در تصویر بدون نویز جایگزین کنیم. در زیر خروجی کد را مشاهده می‌کنید:

احتمال $\Omega^*$	90%	60%	30%	10%	0% (برچسب‌زنی اولیه کاملاً تصادفی)
دقت	99.71%	99.49%	98.94%	97.86%	96.58%
تصویر قطعه‌بندی شده با MRF					
تصویر با نویز گاوسی (۰, ۰, ۰, ۵)					

$\Omega^*$ : احتمالی که با استفاده از آن تصمیم می‌گیریم که برچسب اولیه هر پیکسل در Simulated Annealing را با مقدار برچسب درست آن پیکسل جایگزین کنیم و یا از همان برچسب تصادفی استفاده کنیم.

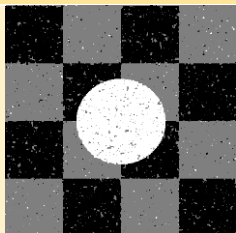
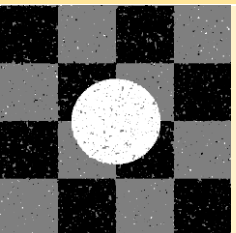
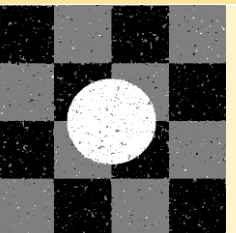
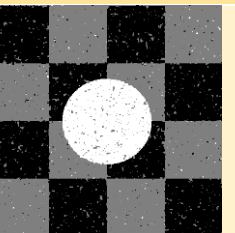
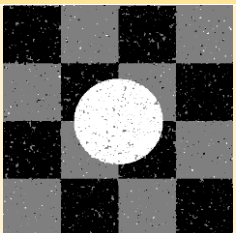


در جدول بالا  $\Omega$ ، احتمالی است که با آن تصمیم می‌گیریم که برای هر پیکسل از برچسب واقعی و درست پیکسل به عنوان برچسب اولیه استفاده کنیم و یا از مقدار برچسب تصادفی به عنوان برچسب اولیه استفاده کنیم. هر چه میزان  $\Omega$  بیشتر باشد بدین معنی است

که پیکسل‌های بیشتری برچسب اولیه برابر با مقدار برچسب درست در تصویر بدون نویز دارند. برای بررسی تاثیر مقداردهی دستی برخی از پیکسل‌ها، تصویر نویزی شده با نویز گوسی  $(0,0.05)$  را با ۵ مقدار احتمال  $\Omega$  قطعه‌بندی کردیم. همین‌طور که در دقت‌های ارائه شده در جدول بالا دیده می‌شود هر چه میزان پیکسل‌هایی که برچسب اولیه‌ی برابر با مقدار واقعی داشته‌اند بیشتر شده است دقت قطعه‌بند افزایش یافته است. البته این نتیجه قابل پیش‌بینی بود زیرا از Simulated Annealing استفاده می‌کنیم، در این روش به طور تصادفی یک پیکسل را انتخاب می‌کنیم و یک برچسب تصادفی جدید برای آن پیکسل انتخاب می‌کنی. در این روش ممکن است یک پیکسل در کل فرآیند انتخاب نشود. در نتیجه هر چه تعداد پیکسل‌هایی که در ابتدا برچسب اولیه درست دارند بیشتر باشد با احتمال بیشتری به دقت بهتری دست می‌یابیم.

### قسمت ح)

کد این قسمت در `p1h.py` قرار دارد. کد این قسمت مانند کد `p1c.py` است و از همان پارامترها استفاده شده است با این تفاوت که عمل قطعه‌بندی را با دماهای اولیه‌ی و ضریب‌های کاهش دمای (ضریبی که هر ایپاک در دما ضرب می‌شود و آن را کاهش می‌دهد) متفاوت انجام می‌دهیم. در زیر خروجی کد را مشاهده می‌کنید:


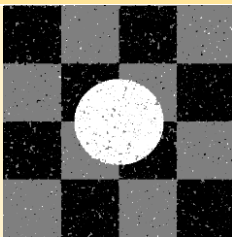
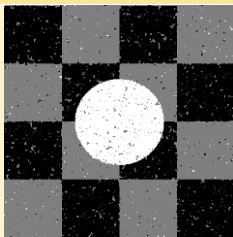
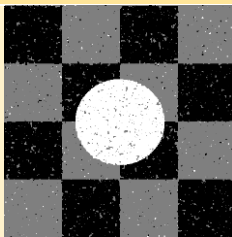
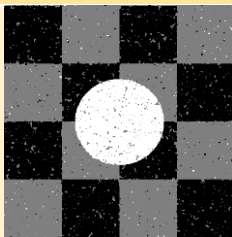

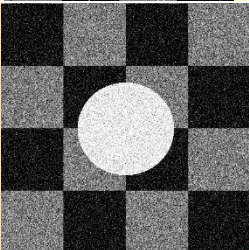
ابتدا برای دماهای اولیه‌ی متفاوت عمل قطعه‌بندی را انجام داده‌ایم که در زیر نتیجه‌ها را مشاهده می‌کنید:

دمای اولیه	10000000	1000000	10000	100	1
دقت	96.43%	96.51%	96.66%	96.65%	96.65%
تصویر قطعه‌بندی شده با MRF					
تصویر با نویز گوسی $(0, 0.05)$					

دقت را در جدول بالا برای دماهای اولیه‌ی مختلف مشاهده می‌کنید. برای دماهای اولیه‌ی مختلف، دقت‌ها بسیار نزدیک هستند. برای مقدارهای زیاد، دقت مقدار ناچیزی کمتر شده است. از آنجایی که دما به صورت نمایی کاهش پیدا می‌کند و به سرعت کم می‌شود مقدارهای اولیه متفاوت تاثیر قابل مشاهده‌ای بر دقت قطعه‌بند نداشته‌اند.

اکنون مقدارهای متفاوت ضریبی که در دما در هر مرحله ضرب می‌شود را مورد بررسی قرار می‌دهیم (دماهای اولیه یکسان و برابر ۴۰۰۰ است):



ضریب دما	1	0.95	0.6	0.1	0.01
دقت	33.57%	96.63%	96.655%	96.651%	96.653%
تصویر قطعه‌بندی شده با MRF					
تصویر با نویز گاوسی (0.5, 0.0, 0.5)					




همین‌طور که مشاهده می‌شود برای ضریب‌های کوچک‌تر از یک دقت‌ها بسیار نزدیک به هم هستند و تقریباً یکسان هستند زیرا دما به صورت‌نمایی با ضریب کمتر از ۱ کاهش پیدا می‌کند و برای ضریب‌های مختلف با چند گام زودتر یا دیرتر از یک دمای اولیه یکسان از حد خاصی کمتر می‌شوند به همین دلیل ضریب زیاد تاثیرگذار نیست ولی برای ضریب برابر یک، دمای اولیه کاهش پیدا نمی‌کند به همین دلیل دقت بسیار کم است.

## بخش دوم

به دلیل بزرگ بودن تصویر برای سریع‌تر شدن محاسبات، تصویر ورودی را به اندازه‌ی یک سوم سایز اصلی، **resize** کرده‌ام.

## قسمت الف)

کدهای این قسمت در **p2a-1.py** و **p2a-1.py** قرار دارد. در این قسمت ابتدا تصویر را خوانده‌ام یک بار تصویر را به **Gray Scale** و بار دیگر به **Hue** تبدیل کرده‌ام. همین‌طور طول و عرض تصویر را به یک سوم کاهش داده‌ام. سپس مانند سوال قبل قطعه‌بندی را با مدل مارکو انجام داده‌ام. پارامتر  $\beta$  برابر با ۰.۱ است، دمای اولیه برابر ۴۰۰۰ است، دما در هر ایپاک ضرب در ۰.۹۵ می‌شود، دو میلیون ایپاک انجام شود، همسایگی از نوع ۴ تایی است. در زیر نتیجه‌ی کد را مشاهده می‌کنید:

		Gray Level Intensity
		HUE
		تصویر اصلی

**نکته:** همین طور که در ابتدا گزارش اشاره شد، برای دیدن تصویرهای درون گزارش در اندازه‌ی واقعی می‌توانید به پوشه‌ی output مراجعه فرمایید.

همین طور که مشاهده می‌شود قطعه‌بندی با استفاده از Hue بهتر از Gray Level Intensity است.

### قسمت ب)

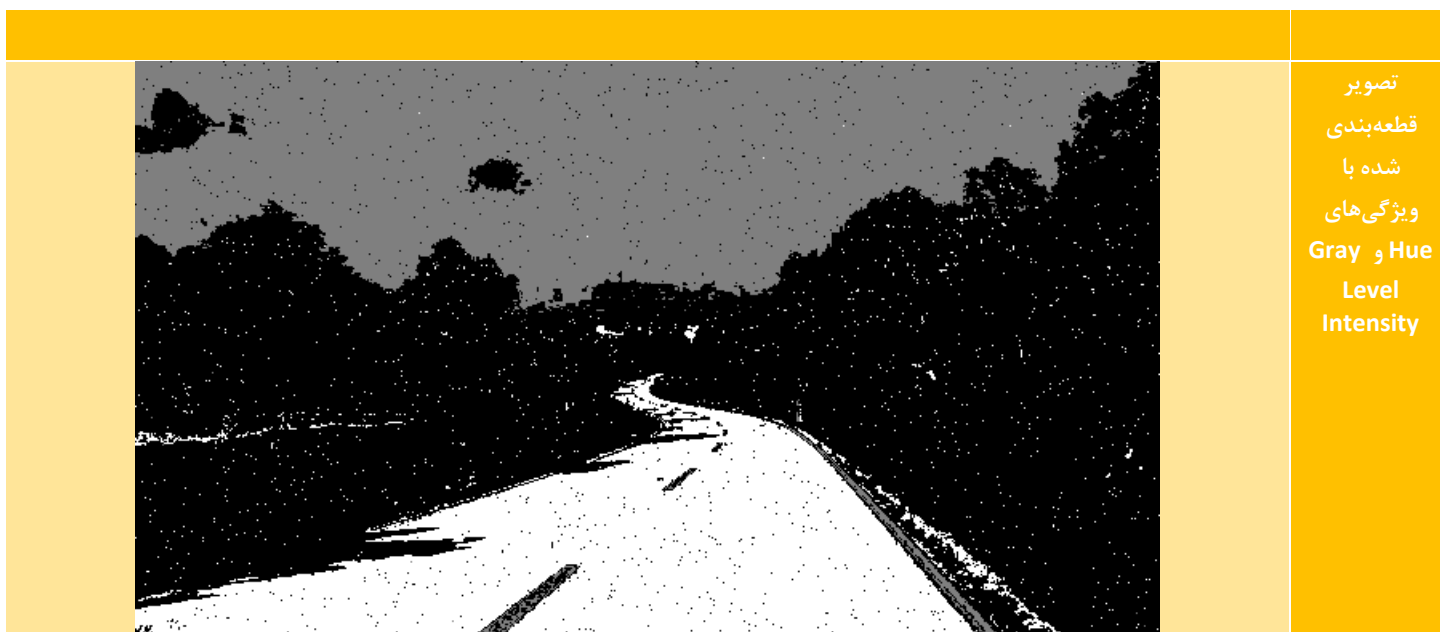
کد این قسمت در p2b.py قرار دارد، برای انجام این بخش تابع‌های قبلی را تغییر دادیم و تابع‌های زیر را نوشتیم (پارامتر  $\beta$  برابر با ۰,۱ است، دمای اولیه برابر ۴۰۰۰ است، دما در هر ایپاک ضرب در ۰,۹۵ می‌شود، دو میلیون ایپاک انجام می‌شود و همسایگی از نوع ۴تایی است):

**calculate\_potential\_multi\_features**: این تابع مانند تابع calculate\_potential در تمرین‌های قبل است و برای محاسبه‌ی پتانسیل تصویر برچسب زده شده با بیش از یک ویژگی است، با این تفاوت که این تابع بر خلاف تابع قبلی از لگاریتم توزیع گوسی چند متغیره استفاده می‌کند.

**calculate\_potential\_by\_modification\_multi\_features**: این تابع برای محاسبه‌ی پتانسیل یک تصویر با بیش از یک ویژگی است و مانند تابع calculate\_potential\_by\_modification است که در قسمت‌های قبل از آن استفاده کردیم با این تفاوت که از لگاریتم توزیع گوسی چند متغیره در آن استفاده شده است.

**Simulated\_annealing\_multi\_feature**: این تابع مانند تابع simulated\_annealing است با این تفاوت که تابع‌های جدید محاسبه‌ی پتانسیل را فرا می‌خواند.

از ویژگی Hue و Gray Scale Intensity برای قطعه‌بندی تصویر استفاده کرده‌ایم که در زیر خروجی کد را مشاهده می‌کنید:





**نکته:** همین طور که در ابتدا گزارش اشاره شد، برای دیدن تصویرهای درون گزارش در اندازه‌ی واقعی می‌توانید به پوشه‌ی output مراجعه فرمایید.

همین‌طور که در تصویر حاصل از قطعه‌بندی با بیش از یک ویژگی دیده می‌شود، نتیجه‌ی قطعه‌بندی بسیار بهتر از تصویر حاصل از یک ویژگی است.