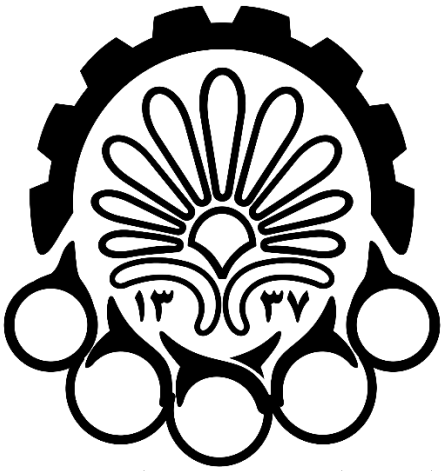


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

تمر ین چهارم درس بینایی ماشین

دکتر صفابخش

غلامرضا دار ۴۰۰۱۳۱۰۱۸

بهار ۱۴۰۱

فهرست مطالب

الف) ۳
ب) ۶
ج) ۱۳
د) ۱۹

الف)

استفاده مستقیم از K-Means بر روی مقادیر پیکسل‌های تصویر، مشکلات زیادی دارد. یکی از مشکلات این کار، در نظر نگرفتن **موقعیت مکانی** پیکسل‌ها و عدم توجه به مقادیر پیکسل‌های همسایه است. نتیجه این امر این است که پیکسل‌هایی با مقدار سطح خاکستری یکسان در یک خوشه قرار می‌گیرند فارغ از اینکه واقعا به یک شئ یا یک بافت متعلق هستند یا خیر.

نتیجه قطعه‌بندی دو تصویر داده شده به کمک این روش را در تصاویر زیر مشاهده می‌کنید.

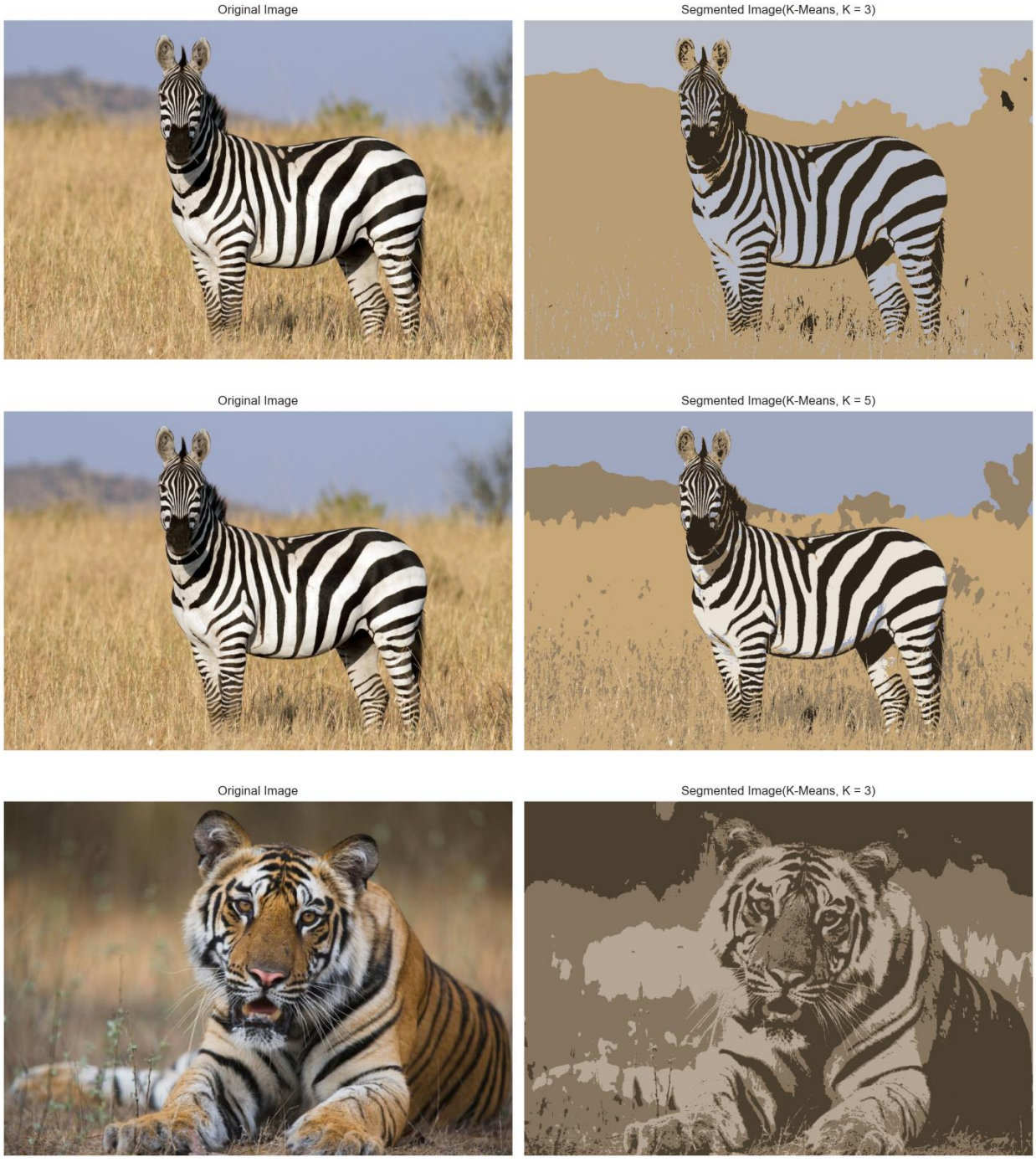
قطعه بندی با استفاده از K=3

قطعه بندی با استفاده از K=5

قطعه بندی با استفاده از K=3

قطعه بندی با استفاده از K=5

نتیجه خوشه‌بندی با تصاویر رنگی



چون در این روش، به طور مستقیم بر روی مقدار عددی پیکسل‌ها خوشه‌بندی را انجام می‌دهیم، بهتر است از هر سه کانال رنگی موجود استفاده کنیم. این کار باعث می‌شود بخش‌هایی مانند آسمان و چمن‌زار که از نظر سطح روشنایی تقریبا شبیه هستند ولی ترکیب مقادیر سه کانال RGB آنها متفاوت است نیز به خوبی تمییز داده شوند.

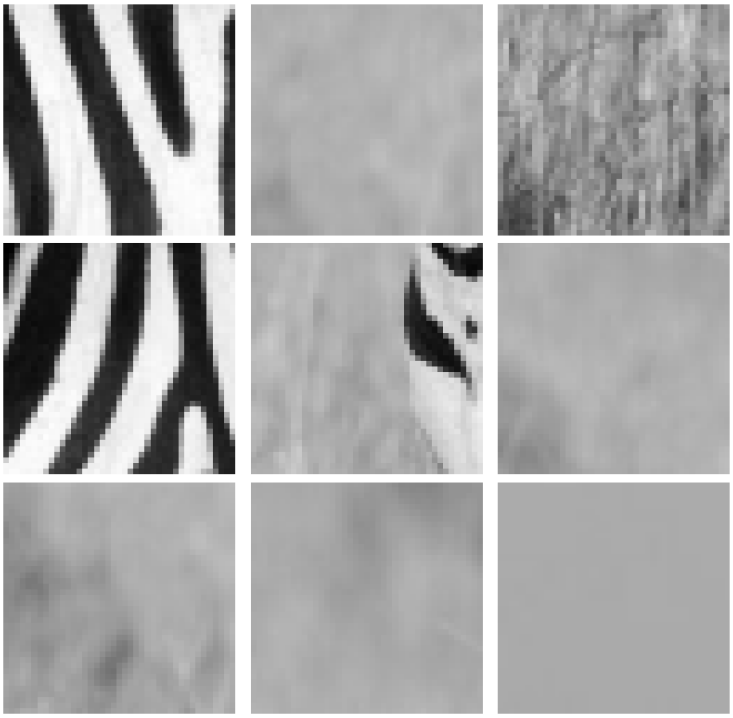
ب)

در این بخش از سوال می‌خواهیم با کمک روش گفته شده در صورت سوال، تصاویر داده شده را قطعه بندی معنایی کنیم.

Image 1 @ 333x500

طبق پیشنهاد سوال، ابتدا تصویر داده شده را به پچ‌هایی با اندازه مشخص تقسیم می‌کنیم. دوست داریم اطلاعات موجود در یک پچ، همگن باشد و به طور مشخص بیان‌گر یک بافت خاص باشد. در تصویر زیر می‌بینید که پچ‌های مربوط به بدن گورخر و گندم‌زار به خوبی از یک‌دیگر قابل تمییز هستند.

Image 1 @ 333x500 Patches | Patch Size: 41x41 | grayscale_levels: 256



در ادامه با استفاده از لینک‌های فراهم شده در صورت سوال، به استخراج ویژگی‌هایی به کمک GLCM برای تصاویر پرداختیم. با مراجعه به وب‌سایت skimage می‌بینیم که ویژگی‌های آماری متعددی را در اختیار داریم.

- 'contrast': $\sum_{i,j=0}^{levels-1} P_{i,j}(i-j)^2$
- 'dissimilarity': $\sum_{i,j=0}^{levels-1} P_{i,j}|i-j|$
- 'homogeneity': $\sum_{i,j=0}^{levels-1} \frac{P_{i,j}}{1+(i-j)^2}$
- 'ASM': $\sum_{i,j=0}^{levels-1} P_{i,j}^2$
- 'energy': $\sqrt{ASM}$
- **'correlation':**

$$\sum_{i,j=0}^{levels-1} P_{i,j} \left[\frac{(i-\mu_i)\,(j-\mu_j)}{\sqrt{(\sigma_i^2)(\sigma_j^2)}} \right]$$

همچنین این ویژگی‌ها به ازای **زوایای** ورودی و **فواصل پیکسلی** مختلف محاسبه می‌شوند و به عنوان ویژگی در اختیار ما قرار داده می‌شوند. در ادامه این ویژگی‌های استخراج شده برای هر پیکسل از تصویر را به عنوان ورودی به الگوریتم K-Means می‌دهیم. پیکسل‌های مختلف بر خلاف بخش الف تمرین، این‌بار نه تنها بر اساس مقدار سطح خاکستری بلکه بر اساس تعداد زیادی ویژگی محلی خوشه بندی می‌شوند. نتیجه این امر این است که پیکسل‌هایی با بافت یکسان، در خوشه‌های یکسان قرار می‌گیرند.

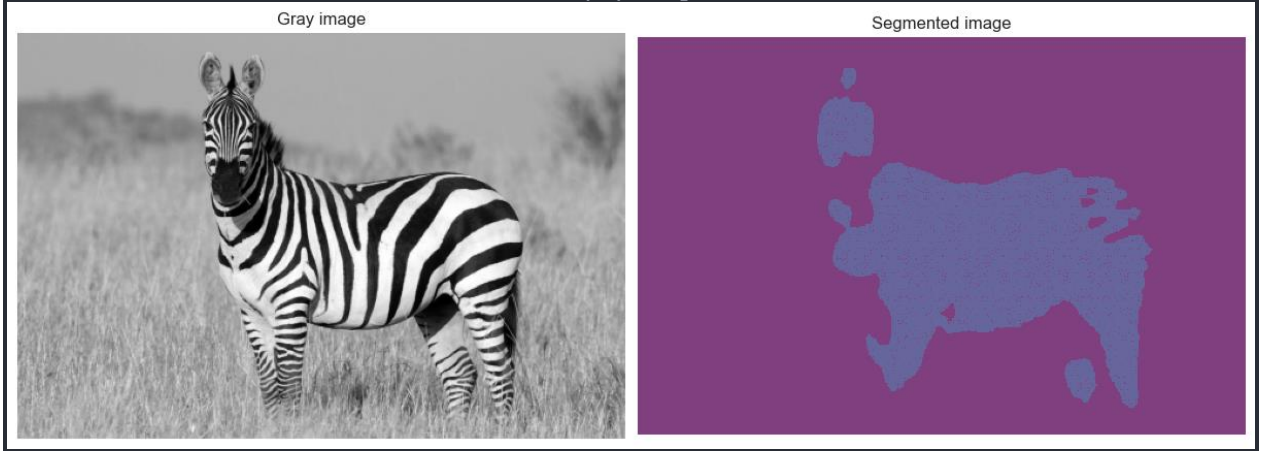
در صفحه بعد، تعدادی از نتایجی که با استفاده از پارامترهای مختلف به‌دست آمده را مشاهده خواهید کرد. لازم به ذکر است که زمان اجرای این الگوریتم بسیار بالا بود و بدون استفاده از پردازش موازی پچ‌ها غیرممکن بود!

با استفاده از پردازش موازی پچ‌ها، اجرای الگوریتم بر روی تصویر ۱۰۰۰ در ۶۶۷ پیکسل با اندازه پچ حدود ۳۰ و تعداد سطوح خاکستری حدود ۳۰، به چیزی بین ۱۰ تا ۲۰ ثانیه زمان نیاز داشت. (16 threads @ 4.7GHz). بدون استفاده از موازی سازی و کاهش تعداد سطوح خاکستری، اجرای الگوریتم برای هر تصویر به چندین دقیقه زمان نیاز داشت.

```
levels=2, patch_size=5, metrics=["dissimilarity"], distances = [1], angles = [0]
```

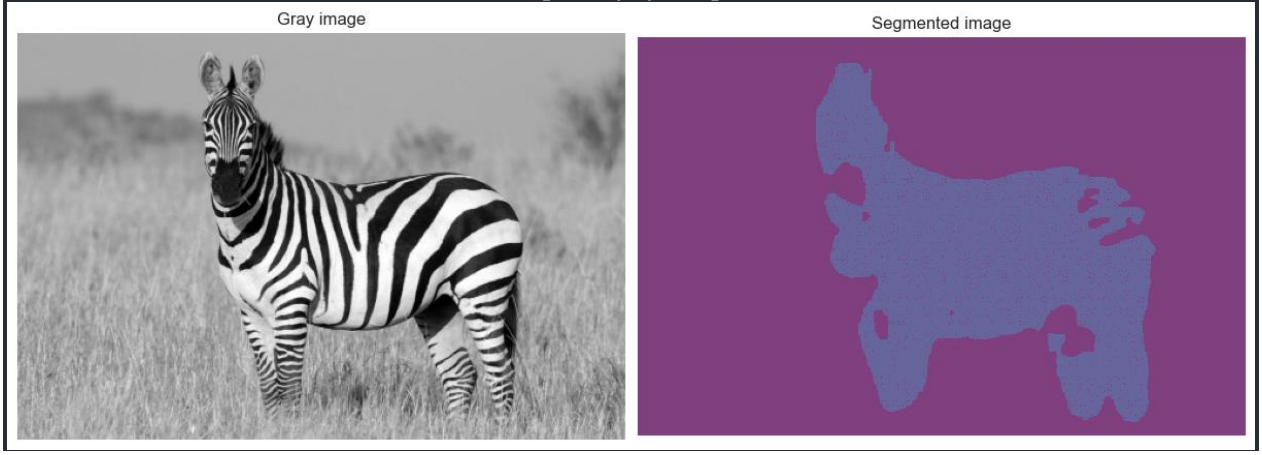

```
levels=2, patch_size=5, metrics=["ASM"], distances = [1], angles = [0]
```


```
levels=20, patch_size=21, metrics=["contrast"], distances = [1,5], angles = [0,
np.pi/2]
```

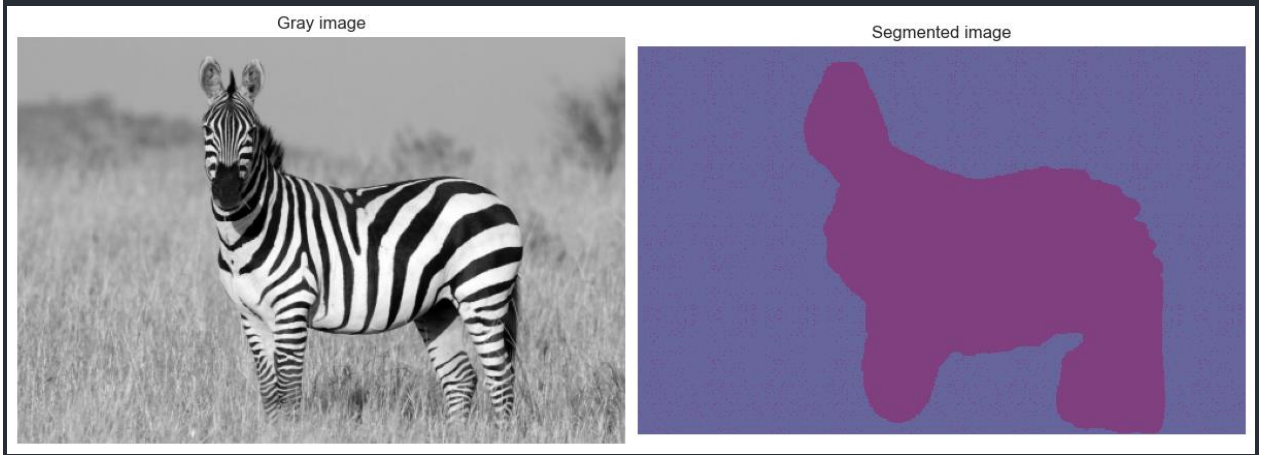


```
levels=20, patch_size=21, metrics=["homogeneity"], distances = [1,5], angles =
                                    [0, np.pi/2]
```

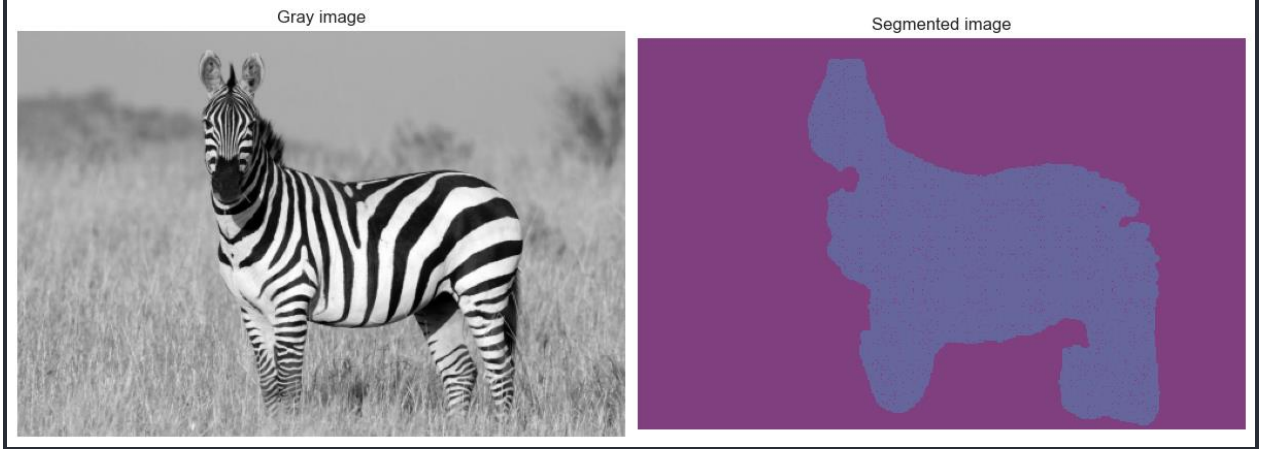

```
levels=20, patch_size=21, metrics=["dissimilarity"], distances = [1,5], angles =
                                    [0, np.pi/2]
```



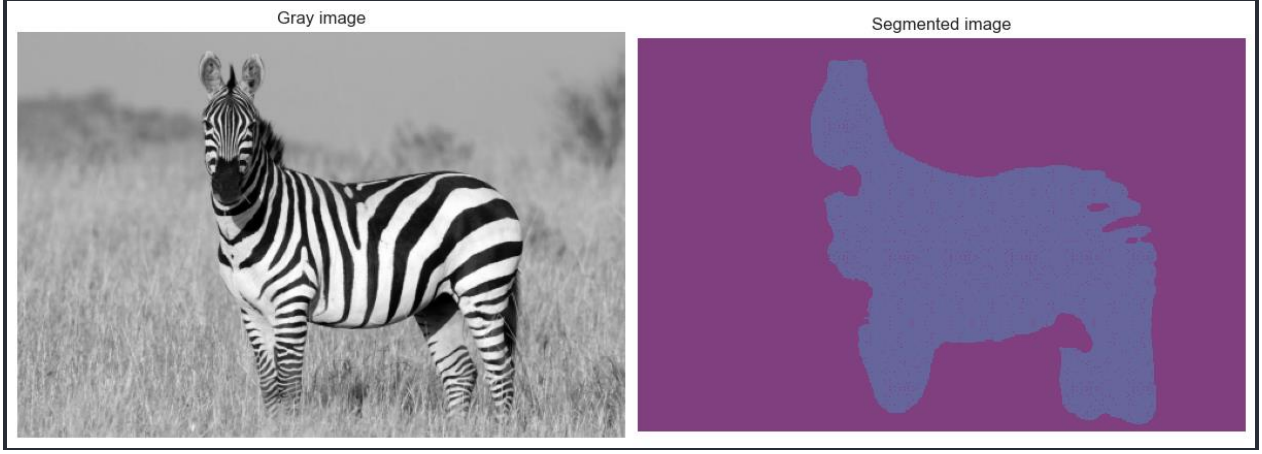

```
levels=20, patch_size=41, metrics=["dissimilarity"], distances = [1,5], angles =
                                    [0, np.pi/2]
```



```
levels=20, patch_size=35, metrics=["dissimilarity"], distances = [1,5], angles =
                                    [0, np.pi/2]
```



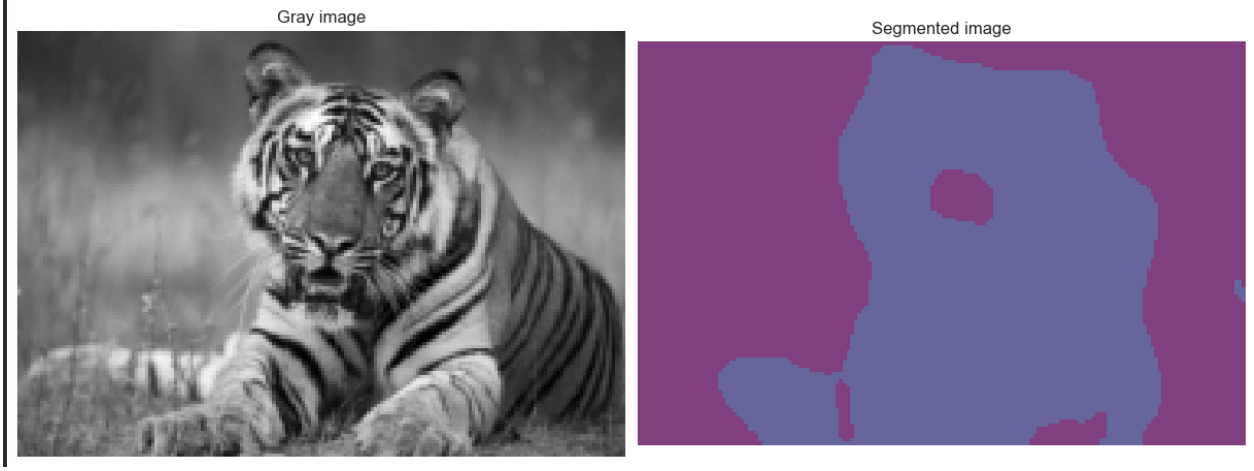

```
levels=20, patch_size=31, metrics=["dissimilarity"], distances = [1,5], angles =
                                    [0, np.pi/2]
```



همان‌طور که مشاهده می‌شود، نتایج ناحیه پس زمینه از گورخر تقریبا جدا شده اند و اختلاف مقدار سطح روشنایی راه‌راه‌های بدن گورخر باعث نشده‌اند در خوشه‌های مختلف قرار بگیرند. همچنین یکی از پر اهمیت ترین، پارامترها در این الگوریتم، اندازه پچ است. اندازه پچ بسیار کوتاه باعث می‌شود بافت‌های بزرگ تشخیص داده نشوند، و اندازه پچ بزرگ باعث می‌شود اطلاعات شکل اجسام از بین برود و نواحی بسیار نرم شوند.

```
levels=50, patch_size=3, metrics=["dissimilarity"], distances = [1,3,5], angles =
                        [0, np.pi/4, np.pi/2, 3*np.pi/4]
```

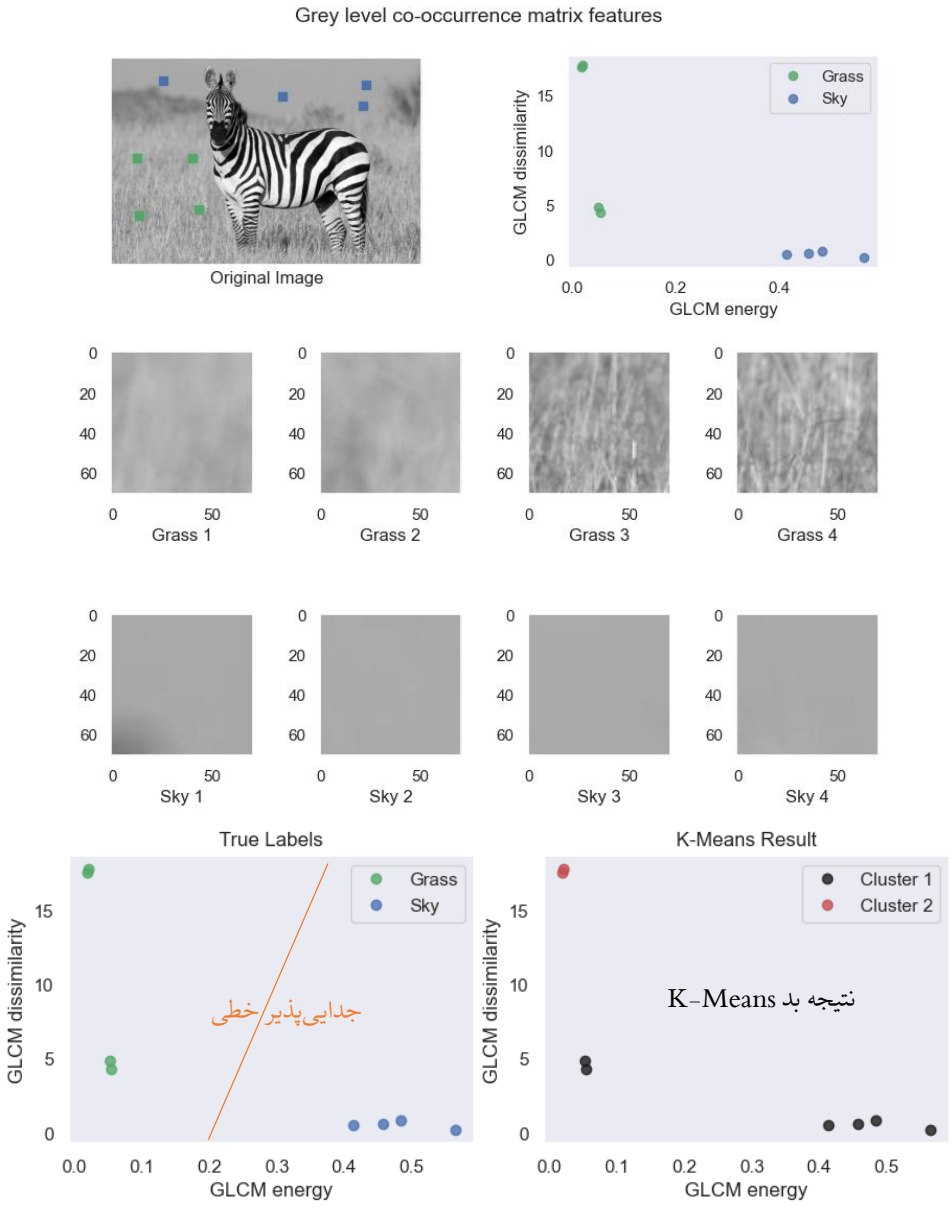

```
levels=50, patch_size=21, metrics=["dissimilarity"], distances = [1], angles =
                        [0, np.pi/4, np.pi/2, 3*np.pi/4]
```



بررسی یکی از دلایل ضعف این روش:

با توجه به نمونه‌کد موجود در وبسایت Skimage، دو ویژگی Energy و Dissimilarity را برای تعدادی پچ از چمن‌زار و آسمان رسم کردیم. همان‌طور که واضح است، همین دو ویژگی باعث شده اند پچ‌های مختلف به صورت خطی جدایی‌پذیر باشند. یافتن این خط جداکننده مستلزم داشتن تعدادی پچ نمونه برچسب‌خورده است، شبیه به یادگیری Supervised.

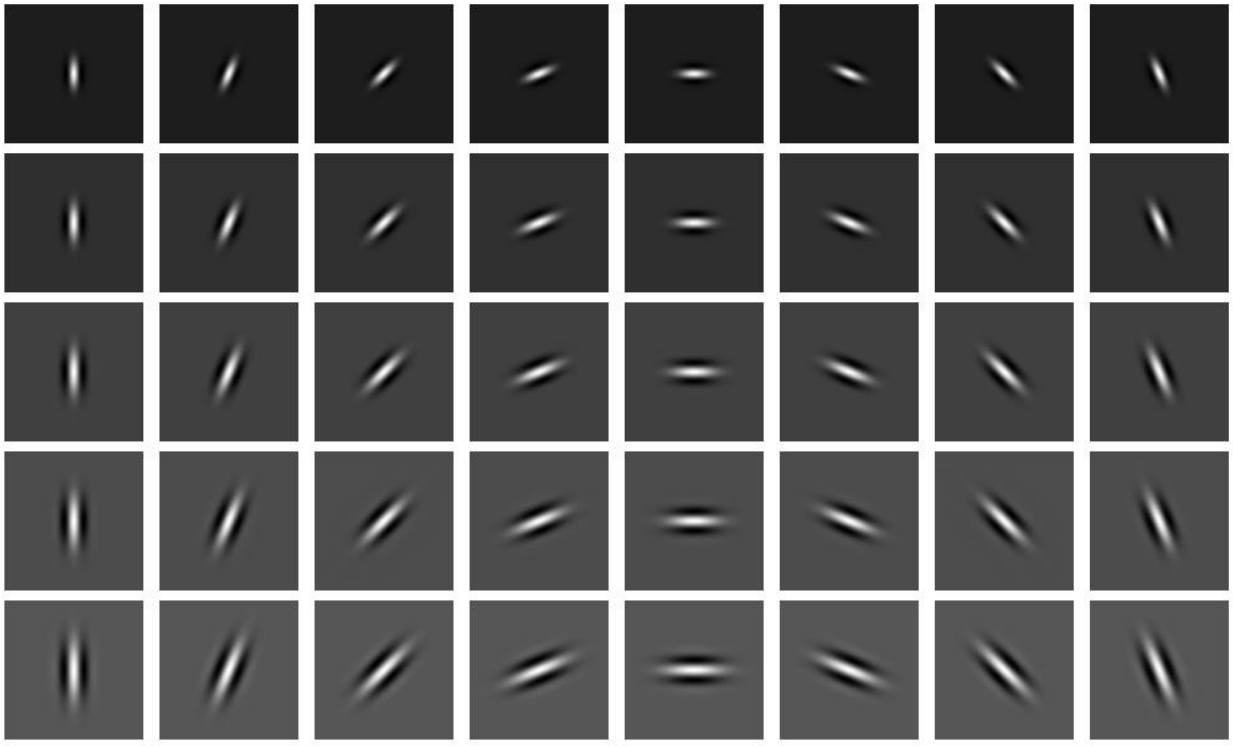
اما در این مسئله هیچ اطلاعات پیشینی در مورد تصاویرو بافت‌ها نداریم و می‌خواهیم این نقاط را به‌کمک الگوریتم K-Means خوشه بندی کنیم. در تصویر پایین نتیجه K-Means را مشاهده می‌کنید که صرفا به فاصله نقاط توجه کرده و عملکرد بسیار نادرستی داشته است.



ج)

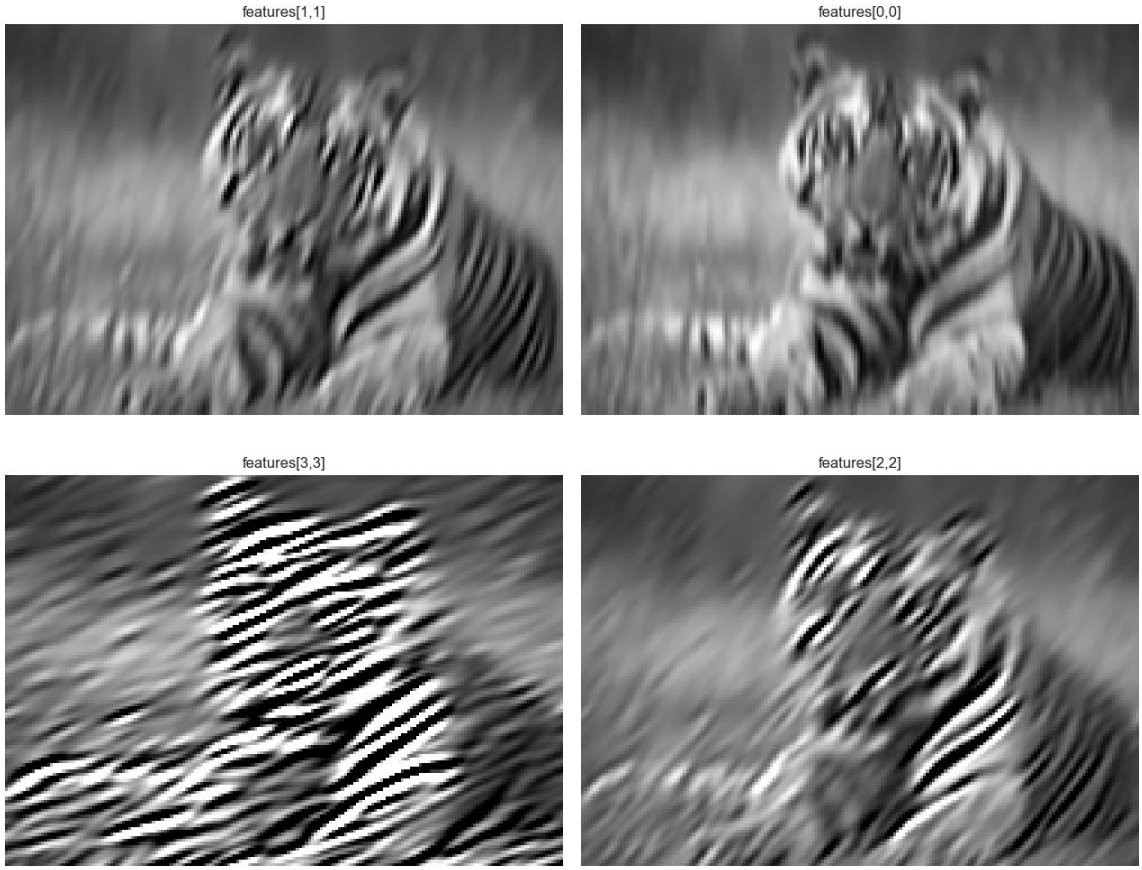
برای روش دوم، نیازی به تقسیم بندی تصاویر به پچ‌های مختلف نیست. در این روش، بر روی هر تصویر تعدادی فیلتر اعمال میکنیم. در نتیجه، برای هر **پیکسل** از تصویر، به تعداد **فیلترها** ویژگی خواهیم داشت. این ویژگی‌ها می‌توانند معانی مختلفی داشته باشند و در مقاصد مختلفی استفاده شوند.

در گام اول، با دادن پارامترهای متعدد به تابع تولید کننده فیلترهای گابور، تعداد ۴۰ فیلتر گابور با ۵ مقیاس مختلف و ۸ زاویه مختلف می‌سازیم. این فیلترها به تصاویر اعمال می‌شوند و هر کدام ویژگی‌های خاصی را از تصویر استخراج می‌کنند.



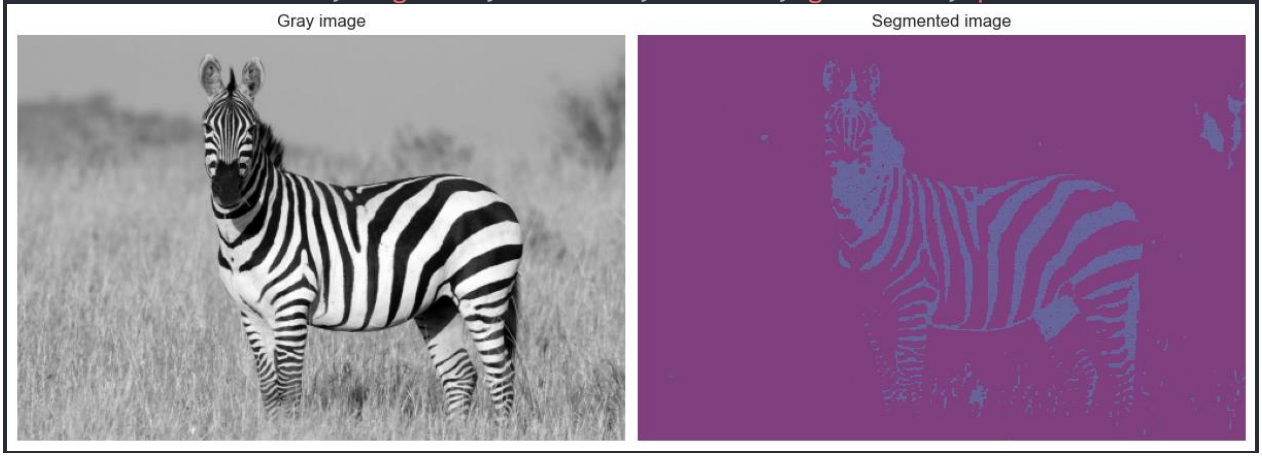
بانک فیلترهای گابور

نتیجه اعمال این فیلترها به تصویر را در این صفحه مشاهده می‌کنید.



در ادامه این ویژگی‌ها را به صورت یک لیست یک بعدی در می‌آوریم و برای هر پیکسل به الگوریتم K-Means می‌دهیم. این الگوریتم بر اساس تعداد زیادی ویژگی از هر پیکسل، پیکسل‌ها را خوشه بندی می‌کند. نتایج این خوشه‌بندی را در صفحه‌های بعد خواهید دید. انتظار می‌رود این روش، به دلیل داشتن اطلاعات کمی در مورد همسابه‌های هر پیکسل، از روش بخش **(الف)** نتیجه بهتری بدهد اما در مقابل روش بخش **(ب)** که اطلاعات بین پیکسلی زیادی داشت شکست بخورد.

scales=5, angles=8, ksize=35, lambd=5, gamma=0.5, psi=0



scales=5, angles=8, ksize=51, lambd=5, gamma=0.5, psi=0

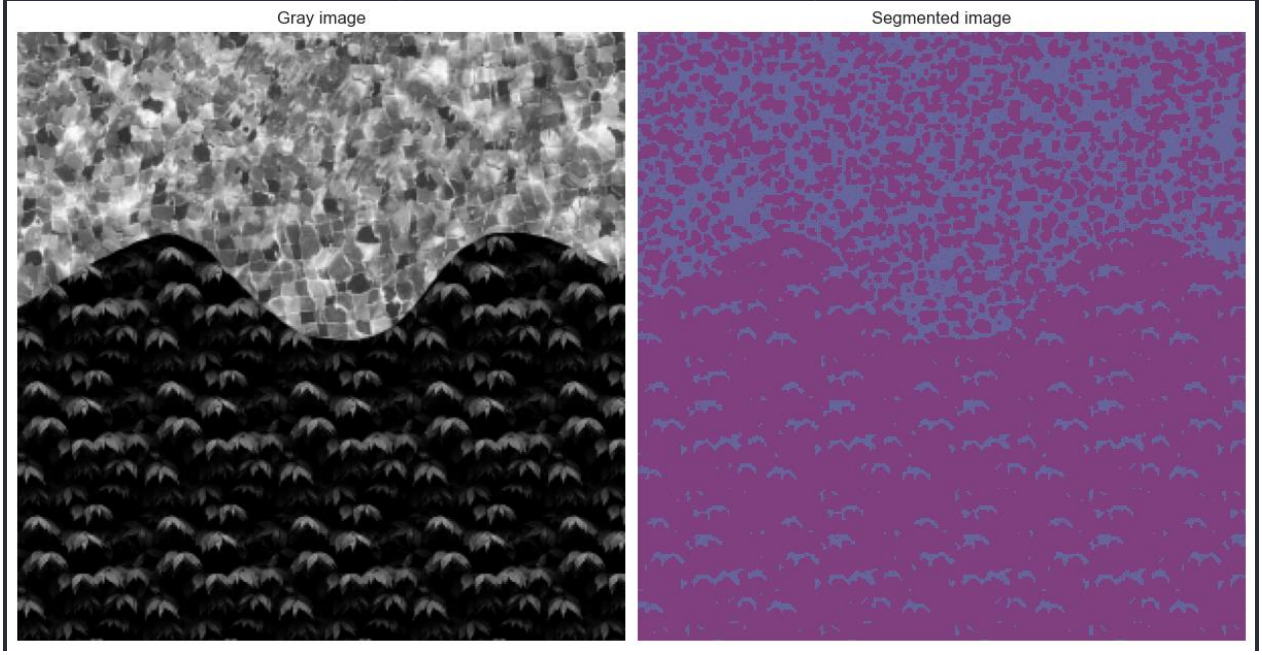
scales=5, angles=8, ksize=71, lambd=31, gamma=0.5, psi=0

```
scales=5, angles=8, ksize=35, lambd=5, gamma=0.5, psi=0
```


```
scales=5, angles=8, ksize=5, lambd=1, gamma=0.5, psi=0
```

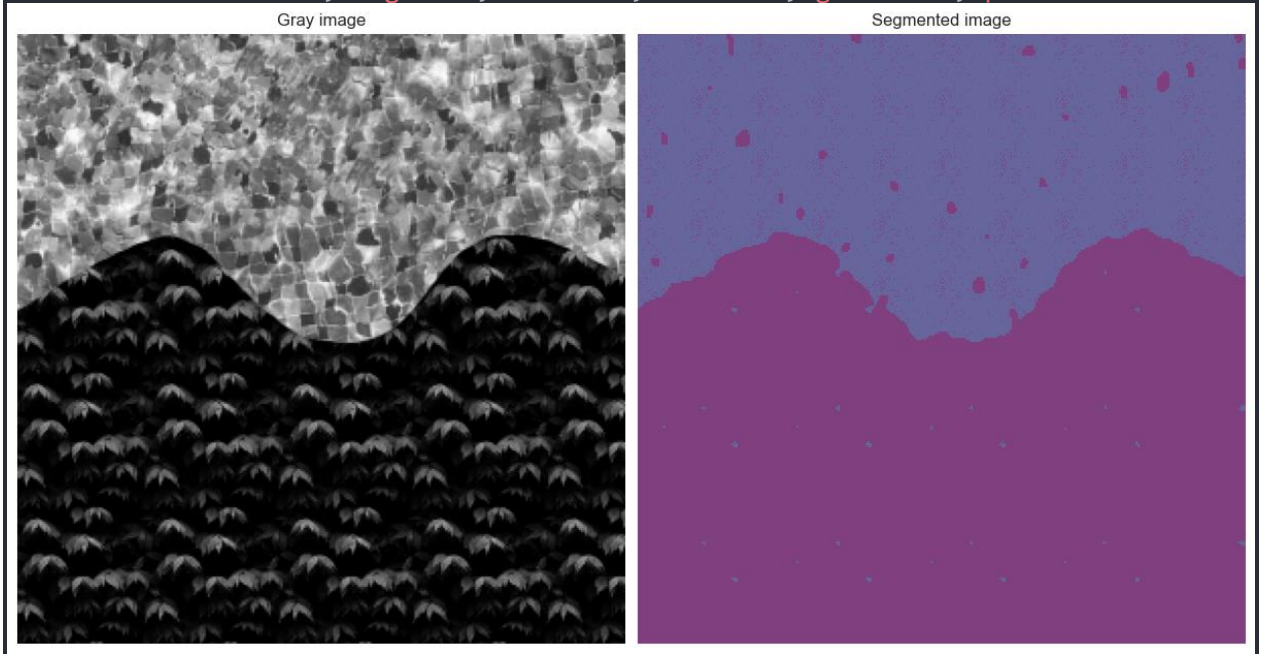
همچنین برای اطمینان از کارکرد الگوریتم، آن را بر روی تصویر ساده زیر اجرا کردیم.

```
scales=5, angles=8, ksize=71, lambd=5, gamma=0.5, psi=0
```



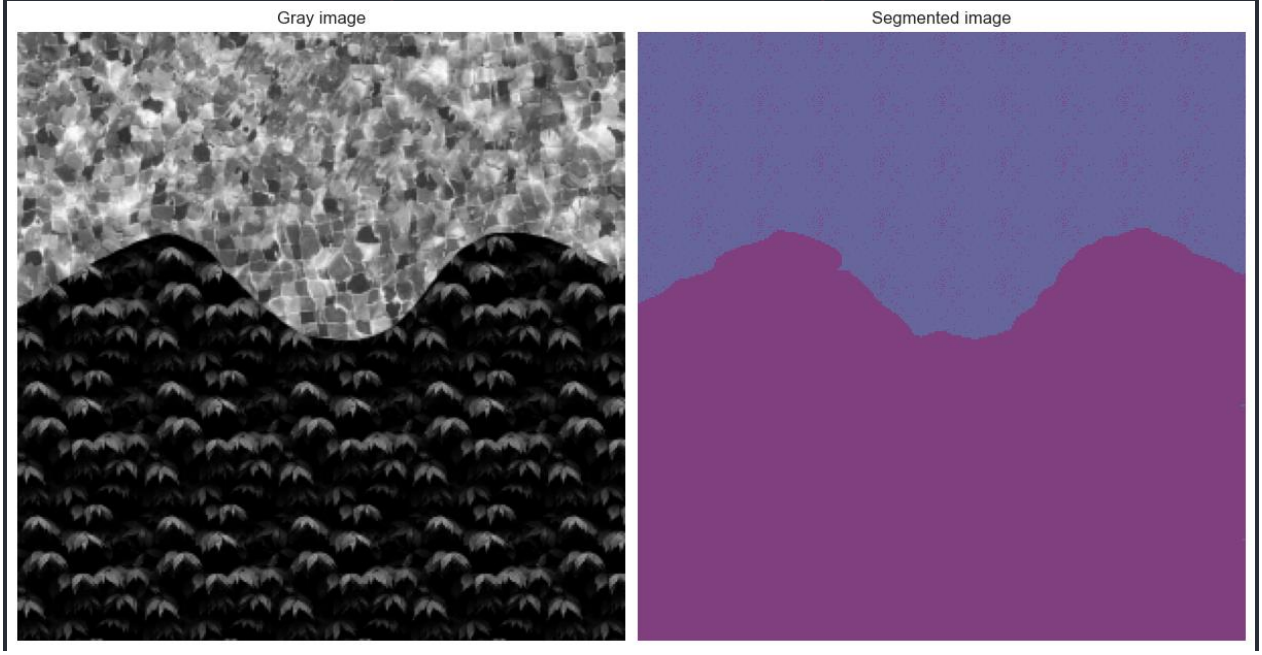
کاهش تعداد فیلترها به یک عدد:

```
scales=1, angles=1, ksize=41, lambd=30, gamma=0.5, psi=0
```



اهمیت وجود فیلترهای متعدد(مقایسه کنید با تصویر قبل):

```
scales=5, angles=8, ksize=41, lambd=30, gamma=0.5, psi=0
```



د)

با توجه به تصاویر خروجی که در دو بخش دیدیم می‌توان این نتیجه را گرفت که روش GLCM با توجه به استفاده از پچ‌هایی از تصاویر، اطلاعات محلی را به خوبی بیشتری حفظ می‌کند. و روش Gabor Filter به دلیل اعمال تعدادی فیلتر به کل تصویر، بسیار سریع‌تر است. تفاوت سرعت اجرای این دو الگوریتم در شرایط ذکر شده در این پیاده سازی حدودا ۲۰ برابر بود. روش Gabor Filters به تغییرات مقدار سطح خاکستری بسیار حساس بود و به عنوان مثال در تصویر گورخر حتی با اندازه کرنل بالا، خطوط گورخر را در خوشه‌های مختلف قرار می‌داد.

- لینک‌های و منابع مفید:

https://github.com/alfianhid/Feature-Extraction-Gray-Level-Co-occurrence-Matrix-GLCM-with-Python/blob/master/Feature_Extraction_Gray_Level_Co_occurrence_Matrix_(GLCM)_with_Python.ipynb

https://scikit-image.org/docs/stable/api/skimage.feature.html#skimage.feature.graycomatrix

https://www.robots.ox.ac.uk/~vgg/research/texclass/without.html

https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_glcm.html#sphx-glr-auto-examples-features-detection-plot-glcm-py

- پارامترهای تابع `cv2.getGaborKernel` :

sigma σ – Standard Deviation – Property of our bell curve. Smaller values emphasize values closer to the center

theta θ – Direction – Identifies direction of our sine wave

lambda λ – Wavelength – Distance between peaks in our sine wave

gamma γ – Ellipticity – Determines how elliptic our 2D bell curve is

psi φ – Offset – Defines the phase offset of our sine wave

پایان