# Lab 6
# Pengolahan Citra
# Image Segmentation
# Senin, 23 November 2020

---

## A. Brief Introduction

Image Segmentation adalah membagi suatu citra menjadi wilayah-wilayah yang homogen berdasarkan kriteria keserupaan tertentu antara suatu piksel dengan piksel — piksel tetangganya, kemudian hasil dari proses segmentasi ini akan digunakan untuk proses tingkat tinggi lebih lanjut yang dapat dilakukan terhadap suatu citra, misalnya proses klasifikasi citra dan proses identifikasi objek.

Image Segmentation dapat dilakukan berdasarkan dengan tiga cara, yaitu *edge-based segmentation, distribution-based segmentation, region-based segmentation*.

## B. Edge-Based Segmentation

Edge-based segmentation melakukan segmentasi citra berdasarkan prinsip *discontinuity* atau diskontinuitas, yang artinya segmentasi dilakukan berdasarkan perubahan intensitas yang mendadak pada citra. Terdapat beberapa cara dalam menerapkan edge-based segmentation, yaitu

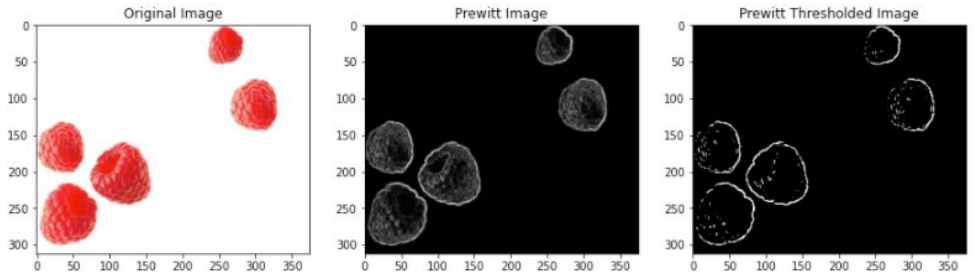
### 1. Basic edge detection

Segmentasi dengan cara ini adalah dengan menggunakan algoritma-algoritma simpel pada *edge-detection* seperti roberts, prewitt, dan sobel. Algoritma-algoritma ini telah kalian pelajari sebelumnya pada lab 1. Silahkan mengacu kepada tutorial 1 untuk penjelasan lebih lengkap bagaimana cara menggunakan.

Setelah melakukan basic edge detection, citra hasil dapat dilakukan juga *thresholding* untuk menampilkan hanya *edge* yang penting (intensitas tinggi). Berikut adalah contoh implementasi *thresholding* dengan menggunakan prewitt filter.

```
import cv2
from matplotlib import pyplot as plt
from skimage import io, color, filters, util

img = io.imread('raspberry.jpg')
gray = color.rgb2gray(img)
prewitt = util.img_as_ubyte(filters.prewitt(gray))
_, prewitt_threshold = cv2.threshold(prewitt, 50, 255,
```

```
cv2.THRESH_BINARY)


plt.figure(figsize=(15,7.5))
plt.subplot(131),plt.imshow(img)
plt.title('Original Image')
plt.subplot(132),plt.imshow(prewitt,cmap = 'gray')
plt.title('Prewitt Image')
plt.subplot(133),plt.imshow(prewitt_threshold,cmap =
'gray')
plt.title('Prewitt Thresholded Image')
plt.show()
```




**2. Canny edge detection**

Canny edge detection juga merupakan salah satu algoritma yang dapat digunakan untuk mendeteksi *edge* pada citra. Algoritma ini adalah algoritma yang terdiri dari beberapa langkah untuk mendeteksi *edge* dan juga menghilangkan noise dengan cara bersamaan. Terdapat 5 langkah yang dilakukan dalam penerapan Canny edge detection yaitu:

a. Noise Reduction

Langkah pertama yang dilakukan saat menjalankan algoritma canny edge detection adalah menghilangkan noise. Hal ini dapat dilakukan dengan menggunakan filter Gaussian yang dapat menghilangkan noise.

b. Gradient Calculation

Langkah kedua yang dilakukan setelah mengurangi noise pada citra adalah dengan menghitung gradien. Untuk menghitung gradien ini kita memerlukan nilai turunan pertama pada sisi horizontal ($G_x$) dan juga pada sisi vertikal ($G_y$). Kedua nilai tersebut bisa didapatkan dengan menerapkan algoritma edge detection simpel seperti Roberts, Prewitt, atau Sobel. Setelah didapatkan kedua nilai tersebut kita bisa menghitung gradien dan juga arah dengan menggunakan persamaan berikut.

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

$$\mathbf{\Theta} = \operatorname{atan2}(\mathbf{G}_y, \mathbf{G}_x),$$

c. Non Maximum Suppression

Non Maximum Suppression adalah teknik yang digunakan untuk mempertipis *edge* yang telah ditemukan. Teknik ini dibutuhkan karena pada idealnya adalah *edge* yang didapatkan haruslah setipis mungkin. Secara simpel, algoritma ini melakukan pengecekan untuk setiap piksel pada citra gradient. Untuk setiap pikselnya, bandingkan nilai piksel tersebut dengan piksel pada arah gradien positif dan negatif. Apabila kekuatan edge tersebut paling besar dibandingkan dengan piksel-piksel lain pada arah yang sama. Maka nilai pada piksel tersebut akan dipertahankan, dan apabila tidak maka nilai pada piksel tersebut akan dihilangkan.

d. Double Threshold

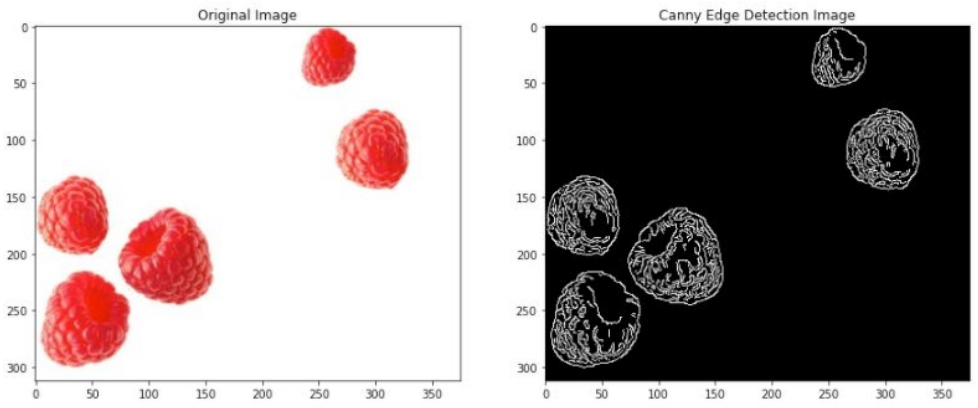
Setelah melakukan non maximum suppression, akan ada *edge* yang merupakan representasi dari citra. Tetapi *edge* tersebut ada yang dibentuk akibat hasil dari *noise* ataupun karena adanya variasi pada warna. Oleh karena itu, untuk menangani masalah tersebut, digunakan 2 buah *thresholding*. Thresholding pertama adalah *high threshold* yang digunakan untuk mengkategorikan apakah *edge* tersebut termasuk *edge* yang kuat atau bukan. Apabila nilai pada piksel melebihi *threshold* maka akan dikategorikan sebagai *edge* kuat (*strong edge*). Thresholding kedua adalah untuk mengkategorikan apakah *edge* yang lemah atau bukan. Apabila nilai pada piksel melebihi *threshold* maka akan dikategorikan sebagai *edge* lemah (*weak edge*). Sisa *edge* yang tidak tergolong pada kedua kategori tersebut akan dibuang.

e. Edge Tracking by hysteresis

Langkah terakhir yang dilakukan adalah *edge tracking*. Hal ini digunakan untuk mengkonversi *edge* lemah tadi menjadi *edge* yang kuat apabila di tetangganya terdapat *edge* kuat. Hal ini berdasar dari sebuah *edge* biasanya saling terhubung satu sama lain. Selain itu *edge* yang merupakan hasil noise juga biasanya merupakan *edge* yang terpisah. Setelah melalui proses ini maka *edge* yang tidak terkonversi menjadi *edge* kuat akan dihilangkan menyisakan hanya *edge* kuat yang tersisa pada citra dan proses canny edge detection berakhir.

Berikut adalah contoh implementasi dari penggunaan canny edge detection

```
import cv2
from matplotlib import pyplot as plt
from skimage import io

img = io.imread('raspberry.jpg')
edges = cv2.Canny(img,100,200)

plt.figure(figsize=(20,10))
plt.subplot(121),plt.imshow(img)
plt.title('Original Image')
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Canny Edge Detection Image')
plt.show()
```




## C. Region-Based Segmentation

Region-based segmentation melakukan segmentasi citra berdasarkan prinsip *similarity* atau kemiripan, yang artinya melakukan segmentasi berdasarkan wilayah/*region* yang mirip.

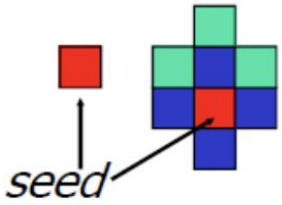
### a. Region Growing

Region growing juga diklasifikasikan sebagai metode segmentasi gambar berbasis piksel karena melibatkan pemilihan *initial seed*. Berikut proses yang dilakukan dalam *region growing*:

1. Menentukan beberapa *initial seeds* m
2. Periksa homogenitas menggunakan 4- atau 8- *neighbors*.
3. Menggunakan *criteria of uniformity*

$$\max_{P \in R} |f(P) - m| < T$$

4. Jika *similar* dengan *seed*, gabungkan/*merge* dengan *region* tersebut. Jika tidak, jangan gabungkan dengan *region*.




**b. Simple Linear Iterative Clustering (SLIC)**

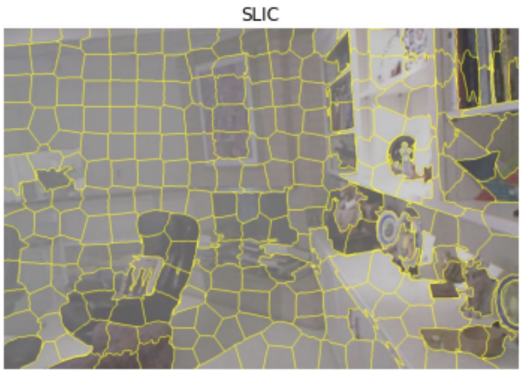
Filter ini membuat superpiksel berdasarkan pengelompokan k-means.

Superpiksel adalah sekelompok kecil piksel yang memiliki properti serupa. Superpiksel menyederhanakan gambar dengan sejumlah besar piksel sehingga lebih mudah ditangani di banyak domain (computer vision, pengenalan pola dan kecerdasan mesin).

k-means clustering adalah salah satu algoritma yang paling banyak digunakan untuk membuat superpiksel. Warna superpixel adalah rata-rata warna piksel di wilayah yang sesuai.

```
from skimage.segmentation import slic
from skimage.color import rgb2gray
from skimage.filters import sobel
from skimage.segmentation import slic
from skimage.segmentation import mark_boundaries
from skimage.util import img_as_float
import numpy as np

img2 = io.imread('slic.jpg')

segments_slic = slic(img2, n_segments=250, compactness=10,
sigma=1, start_label=1)

print(f"SLIC number of segments:
{len(np.unique(segments_slic))}")

fig, ax = plt.subplots(2, 2, figsize=(10, 10),
sharex=True, sharey=True)
```

```
ax[0, 1].imshow(mark_boundaries(img2, segments_slic))
ax[0, 1].set_title('SLIC')

for a in ax.ravel():
   a.set_axis_off()

plt.tight_layout()
plt.show()
```




```
SLIC number of segments: 241
```

Hasil segmentasi dari algoritma SLIC di atas tidak secara langsung berupa citra sedemikian hingga pada contoh di atas untuk tujuan visualisasi, kita perlu menimpa citra dengan segmen hasil SLIC. Kita dapat mengolah lebih jauh lagi hasil *clustering* ini dengan menyeragamkan nilai-nilai piksel dalam tiap superpiksel dengan nilai tertentu, misalnya rata-rata piksel dalam superpiksel, sehingga didapatkan visualisasi berupa citra superpiksel.

```
def superpixel_mean_image(image, slic_labels):
    """
        Code taken from
https://stackoverflow.com/a/57746835
        Turn image into superpixel image, given SLIC
segment labels.
    """

im_rp=image.reshape((image.shape[0]*image.shape[1],image.s
hape[2]))
    sli_1d=np.reshape(slic_labels,-1)
    uni=np.unique(sli_1d)
    new_img=np.zeros(im_rp.shape)
    for i in uni:
        loc=np.where(sli_1d==i)[0]
        #print(loc)
        mm=np.mean(im_rp[loc,:],axis=0)
```

```
        new_img[loc,:]=mm
    return
np.reshape(new_img,[image.shape[0],image.shape[1],image.sh
ape[2]]).astype('uint8')


superpixel_image = superpixel_mean_image(img2,
segments_slic)
plt.imshow(superpixel_image)
```

Citra superpiksel dengan tiap piksel dalam superpiksel dirata-ratakan.

***c. Region Splitting & Merging***

Idenya adalah memecah citra menjadi suatu satu set wilayah yang terpisah yang dapat terhubung satu sama lain. Contoh analoginya adalah seperti *jigsaw puzzle*.

Proses-proses pada *region splitting*:

Top Down Splitting:

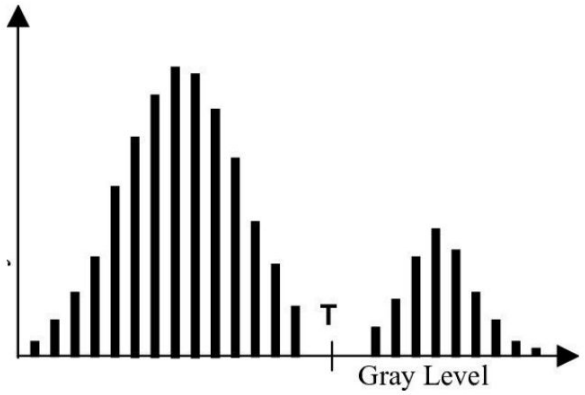
1. Menggunakan *quad tree.*
2. Menggunakan *criteria of uniformity* (misalnya varian).
3. Jika varian terlalu tinggi, pisah dari *region*. Jika tidak, tidak perlu dipisah.

Bottom Up Merging:

1. Kalau sudah tidak ada yang dipisah, merge region-region yang similar
2. Stop ketika sudah tidak ada lagi yang bisa di-merge.

## D. Distribution-Based Segmentation

### a. Histogram Thresholding (Otsu)

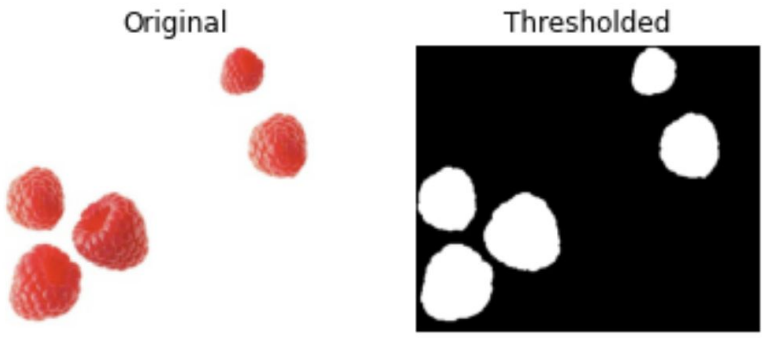



Kita dapat memisahkan citra kedalam area ‘terang’ dan 'gelap' dengan menggunakan Thresholding (T).

$$g(x,y) = \begin{cases} 1, & \text{if } f(x,y) > T \\ 0, & \text{if } f(x,y) \leq T \end{cases}$$

Metode thresholding Otsu adalah masalah binerisasi. Ide dari histogram thresholding Otsu adalah menemukan nilai threshold yang ideal untuk meminimalkan varian dalam kelas dan memaksimalkan varian antar kelas.

```
from skimage import io, color, filters, util
import matplotlib.pyplot as plt

G = util.img_as_ubyte(color.rgb2gray(img1))
T = filters.threshold_otsu(G)
S = util.img_as_float(G > T)
plt.subplot(1,2,1); plt.imshow(img1)
plt.title('Original'); plt.axis("off")
plt.subplot(1,2,2); plt.imshow(1-S, cmap='gray')
plt.title("Thresholded"); plt.axis("off")
plt.show()
```



OpenCV memiliki beberapa pilihan method yang dapat digunakan. Selain OpenCV, dapat juga menggunakan library lain seperti skimage. Selamat mengeksplor :)

credit gambar: https://dlpng.com/png/362960,