Computer Vision
Dr. Mohammadi
Fall 2022
Hoorieh Sabzevari - 98412004
HW9

۱.

روش کار برای عملگر گسترش به این صورت است که ابتدا کرنل را ۱۸۰ درجه چرخانده، روی تمامی پیکسل‌ها قرار داده و سپس بین خانه‌هایی از کرنل که مقدار آن یک است روی تصویر اصلی مقدار بیشینه را انتخاب می‌کنیم و در خانه‌ی وسط قرار می‌دهیم. برای عملگر سایش نیز به همین نحو عمل می‌کنیم اما نیازی به چرخاندن کرنل نیست و همچنین مقدار مینیمم را در خانه‌ی مرکزی قرار می‌دهیم. برای padding نیز از reflect-padding استفاده می‌کنیم. برای مثال:

10	10	10
10	20	20
10	10	10

*

1	1	1
1	0	0
1	0	0

در این مثال مقادیری که کرنل آن خانه‌ها برابر یک است عبارتند از ۱۰،۱۰،۱۰،۱۰و۱۰.

که مینیمم این اعداد همان مقدار ۱۰ می‌شود. پس بجای ۲۰ در عملگر سایش عدد ۱۰ را قرار می‌دهیم.

برای گسترش در این مثال ابتدا باید فیلتر با ۱۸۰ درجه بچرخانیم.

بقیه‌ی خانه‌ها نیز تا انتها به همین صورت مقداردهی می‌شوند. مقادیری که کرنل آن خانه‌ها برابر یک است عبارتند از ۱۰،۲۰،۱۰،۱۰،۱۰. ماکسیمم این اعداد ۲۰ می‌شود. پس به جای ۲۰ در مرکز همان ۲۰ را قرار می‌دهیم.

نتیجه‌ی اعمال عملگر سایش:

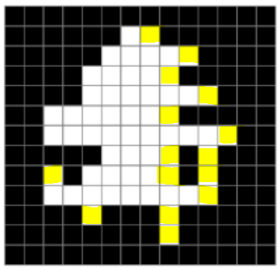
۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰
۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰
۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰
۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰
۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰
۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰
۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰	۱۰
۱۰	۱۰	۱۰	۱۰	۲۰	۲۰	۲۰	۲۰

نتیجه‌ی اعمال عملگر گسترش:

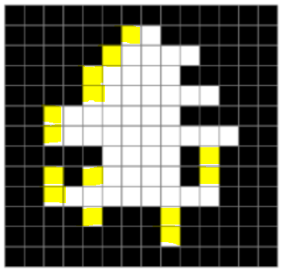
۲۰	۲۰	۲۰	۲۰	۲۰	۲۰	۲۰	۲۰
۲۰	۲۰	۲۰	۲۰	۲۰	۲۰	۲۰	۲۰
۳۰	۳۰	۳۰	۳۰	۳۰	۳۰	۲۰	۲۰
۲۰	۳۰	۳۰	۳۰	۳۰	۳۰	۳۰	۱۰
۲۰	۳۰	۳۰	۳۰	۳۰	۳۰	۳۰	۲۰
۲۰	۲۰	۲۰	۳۰	۳۰	۲۰	۳۰	۲۰
۳۰	۳۰	۳۰	۳۰	۳۰	۳۰	۳۰	۲۰
۳۰	۳۰	۳۰	۳۰	۳۰	۳۰	۳۰	۲۰

۲. در عملگر hit-or-miss ما به دنبال استخراج الگویی در تصویر هستیم.

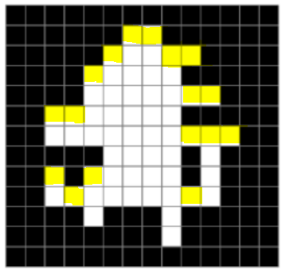
عنصر ساختاری اول: $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix}$



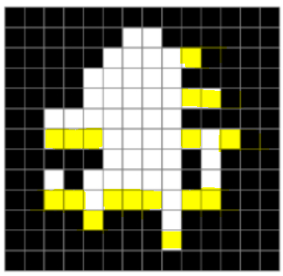
عنصر ساختاری دوم: $\begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$



عنصر ساختاری سوم: $\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

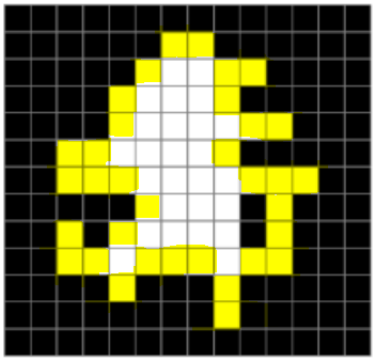


عنصر ساختاری چهارم: $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix}$



خروجی: از اجتماع ۴ تصویر بالا حاصل می‌شود.

البته با این عناصر ساختاری مرز داخلی تصویر با همسایگی ۴تایی بدست آمد. اگر در عناصر ساختاری جای ۱ و ۱- را عوض کنیم مرز خارجی شکل حاصل می‌شود.



۳. با استفاده از فرمول‌های زیر توابع را پیاده‌سازی می‌کنیم.

$$S(A) = \bigcup_{k=0}^{K} S_k(A)$$

$$S_k(A) = (A \ominus kB) - (A \ominus kB) \circ B$$

$$A \ominus kB = \big((A \ominus B) \ominus B\big) \ominus \cdots)$$

$$K = max\{k | (A \ominus kB) \neq \emptyset\}$$

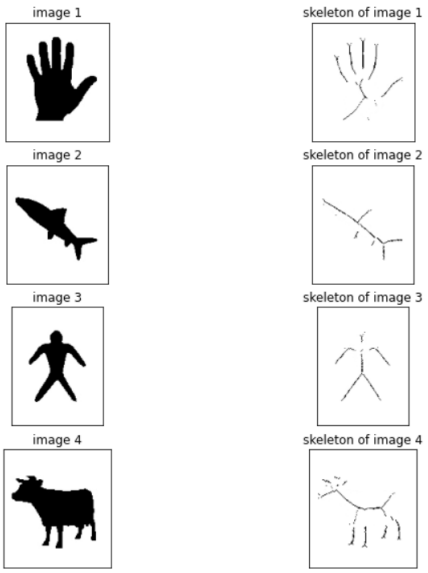
$$A = \bigcup_{k=0}^{K} S_k(A) \oplus kB$$

ابتدا لیستی برای نگهداری اطلاعات برای بازگرداندن تصویر تعریف می‌کنیم. برای اینکه فرمول‌ها درست کار کنند نواحی سفید و سیاه را جابجا می‌کنیم.

سپس عنصر ساختاری cross را تعریف می‌کنیم و فرمول‌ها را پیاده‌سازی می‌کنیم.

در هر مرحله S_k و k را در params ذخیره می‌کنیم. سپس برای هر تصویر، نتیجه‌ی هر مرحله‌ای را که ذخیره کردیم به اندازه‌ی k بار گسترش می‌دهیم تا به تصویر اولیه برسیم.

```
params = []
result = np.zeros(image.shape, np.uint8)
image = cv2.bitwise_not(image)
# image = cv2.GaussianBlur(np.uint8(image),(9,9),0)

se = cv2.getStructuringElement(cv2.MORPH_CROSS,(3,3))
k = 0

while True:
  if cv2.countNonZero(image)==0:
      break
  eroded = cv2.erode(image, se)
  opened = cv2.morphologyEx(eroded, cv2.MORPH_OPEN, se)
  subtracted = cv2.subtract(eroded, opened)
  result = cv2.bitwise_or(result, subtracted)
  params.append((k, subtracted))
  image = eroded
  k += 1

result = cv2.bitwise_not(result)
return result, params
```



```
se = cv2.getStructuringElement(cv2.MORPH_CROSS,(3,3))
result = np.zeros(image.shape, np.uint8)

for i, sk in params:
  img = cv2.dilate(sk, se, iterations = i)
  result = cv2.bitwise_or(result, img)

result = cv2.bitwise_not(result)

return result
```

skeleton of image 1

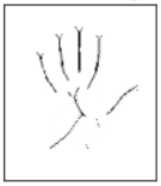


image 1

skeleton of image 2

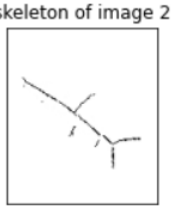


image 2

skeleton of image 3

image 3

skeleton of image 4

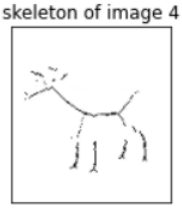
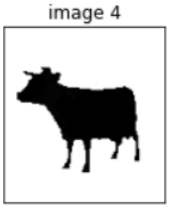


image 4

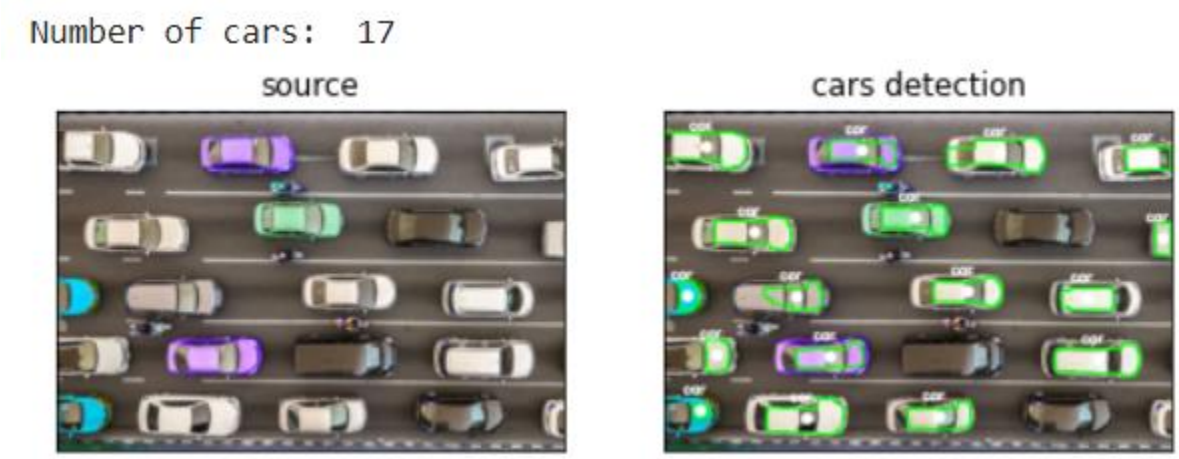


۴.

الف) ابتدا تصویر را grayscale کرده و سپس یک فیلترگاوسی روی آن اعمال می‌کنیم. با تابع threshold() تصویر را باینری کرده و سپس با دو عنصر ساختاری ساخته‌شده ابتدا عملگر open را روی تصویر باینری‌شده و سپس عملگر close را روی نتیجه‌ی آن اعمال می‌کنیم. با انجام این کار نواحی ماشین‌ها جدا شده و به رنگ سفید در بک‌گراند مشکی در می‌آیند که برای تشخیص فرمت مطلوب است. با استفاده از تابع findContours() و کتابخانه‌ی imutils هر ناحیه‌ی سفید را تشخیص داده و به عنوان یک ماشین در نظر می‌گیریم.

```
result = image.copy()
cars_num = None

#Write your code here
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
thresh = cv2.threshold(blurred, 180, 255, cv2.THRESH_BINARY)[1]

se1 = cv2.getStructuringElement(cv2.MORPH_RECT,(11,11))
se2 = cv2.getStructuringElement(cv2.MORPH_RECT,(31,31))
opened = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, se1)
closed = cv2.morphologyEx(opened, cv2.MORPH_CLOSE, se2)
# cv2_imshow(thresh)
# cv2_imshow(opened)
# cv2_imshow(closed)

cnts = cv2.findContours(closed.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
cars_num = len(cnts)
print("Number of cars: ", cars_num)

for c in cnts:
  M = cv2.moments(c)
  cX = int(M["m10"] / M["m00"])
  cY = int(M["m01"] / M["m00"])
  cv2.drawContours(result, [c], -1, (0, 255, 0), 2)
  cv2.circle(result, (cX, cY), 7, (255, 255, 255), -1)
  cv2.putText(result, "car", (cX - 20, cY - 20),
  cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)

return result ,cars_num
```

در خروجی نهایی ۱۷ ماشین تشخیص داده شد و علت خطا این است که ماشین‌های مشکی رنگ هنگام باینری کردن تصویر رنگی، با آسفالت یکی در نظر گرفته می‌شدند که پس از باینری کردن تفاوتی باهم نداشتند و مشکی در نظر گرفته می‌شدند. برای حل این مشکل می‌توان از توابع مشتق-گیر برای پررنگ کردن لبه‌ها استفاده کرد، اما بعد از انجام این کار نیز برخی از جزئیات کم‌اهمیت مانند موتورها و خطوط جدول هم پررنگ می‌شدند که حذف این گونه جزئیات با استفاده از عملگرهای مورفولوژی چالشی دیگر داشت. در انتها پس از تست کردن انواع روش‌های باینری کردن تصویر و تغییر اعداد و پارامترهای عملگرها و عناصر ساختاری بهترین نتیجه‌ی حاصل در تصویر زیر قابل‌مشاهده است.



ب) در این قسمت نیز ابتدا تصویر را grayscale کرده و سپس یک فیلترگاوسی روی آن اعمال می‌کنیم تا نویزهای بی‌اهمیت حذف شوند. سپس عنصر ساختاری مطلوب خود را تعریف کرده و عملیات سایش را روی تصویر اعمال می‌کنیم تا نواحی روشن کوچک‌تر و دایره‌های تصویر قابل-شناسایی شوند. سپس با استفاده از تبدیل هاف دوایر را شناسایی کرده و نمایش می‌دهیم. در این سوال هم در انتها پس از تست کردن انواع روش‌های تشخیص دوایر نامتقارن و تغییر اعداد و پارامترهای عملگرها و عناصر ساختاری بهترین نتیجه‌ی حاصل در تصویر زیر قابل‌مشاهده است.

(لینک کمکی)

```
result = image.copy()
flowers_num = None

#Write your code here
img = cv2.imread('img6.jpg', cv2.IMREAD_COLOR)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray_blurred = cv2.blur(gray, (5, 5))

se = cv2.getStructuringElement(cv2.MORPH_RECT,(9, 9))
# opened = cv2.morphologyEx(gray_blurred, cv2.MORPH_OPEN, se)
# closed = cv2.morphologyEx(gray_blurred, cv2.MORPH_CLOSE, se)
# dilate = cv2.dilate(gray_blurred, se)
erode = cv2.erode(gray_blurred, se)
# cv2_imshow(opened)
# cv2_imshow(closed)
# cv2_imshow(dilate)
# cv2_imshow(erode)


detected_circles = cv2.HoughCircles(erode,
                   cv2.HOUGH_GRADIENT, 1, 20, param1 = 50,
               param2 = 30, minRadius = 10, maxRadius = 51)

if detected_circles is not None:

    detected_circles = np.uint16(np.around(detected_circles))
    flowers_num = len(detected_circles[0])
    for pt in detected_circles[0, :]:
        a, b, r = pt[0], pt[1], pt[2]
        cv2.circle(result, (a, b), r, (0, 255, 0), 2)
        cv2.circle(result, (a, b), 1, (0, 0, 255), 3)
```

19

source

flowers detection