گزارش تمرین نهم بینایی ماشین

نام تهیه‌کننده: ملیکا نوبختیان

شماره دانشجویی: ۹۷۵۲۲۰۹۴

نسخه: ۱

۱- سوال اول

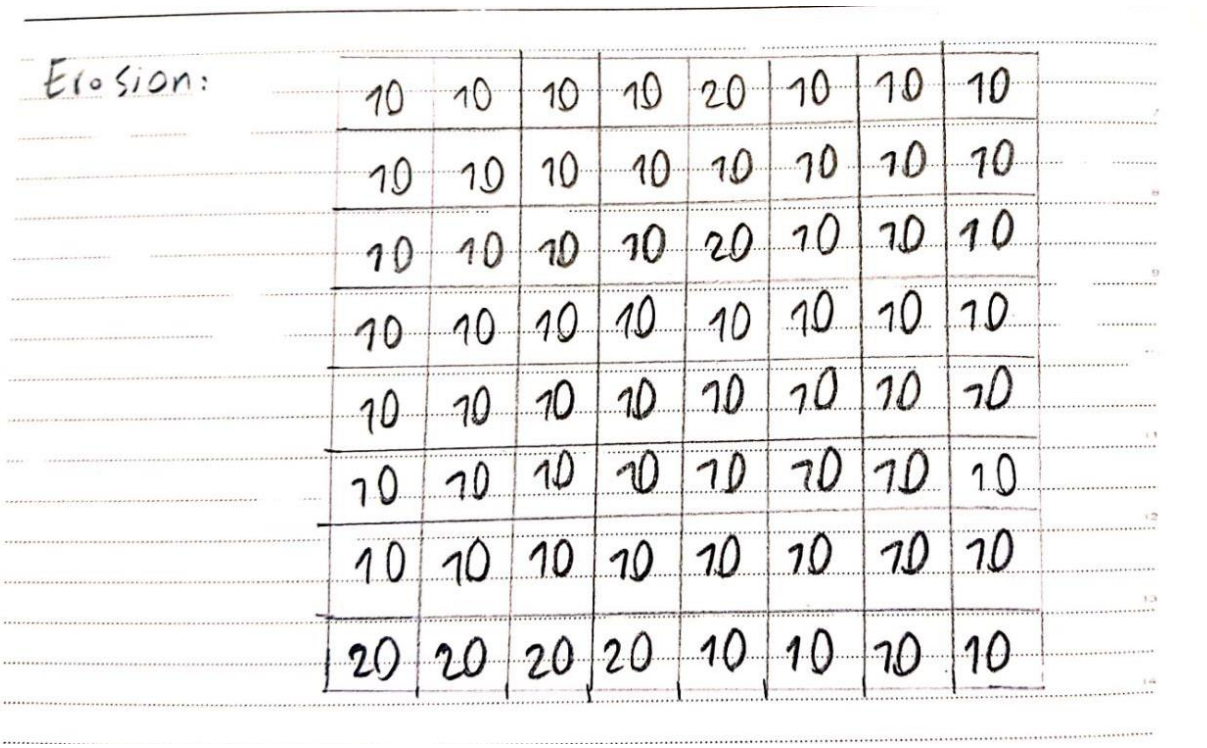
مرداد
شنبه
Sat.
Jul. 2022
۱ محرم ۱۴۴۴
30

در اینجا تصویری برداریم non-binary است. در dilation برای چنین تصاویری زمانی که structuring element بر تصویر تراری گیرد، در سلول های که یک داریم حداکثر max انتخاب می شود و جایگزاری می شود.

در erosion به جای max، min pixel را انتخاب می کنیم.

Dilation:

20	20	20	20	20	20	20	20
20	10	20	20	20	20	20	20
20	20	20	20	20	20	20	20
10	10	20	30	30	20	30	30
10	20	30	30	30	30	30	30
20	30	30	30	30	30	30	20
20	30	30	30	30	30	30	20
20	20	20	30	20	20	30	20



۲- سوال دوم

برای اینکه مرز مجموعه را حساب کنیم از structuring element های مختلف و هم چنین dilate یا erode استفاده می‌کنیم و نتیجه‌های متفاوتی خواهیم داشت.

در ابتدا از این عنصر ساختاری استفاده می‌کنیم:

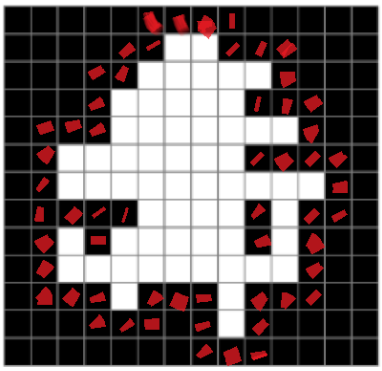
1	1	1
1	1	1
1	1	1

با توجه به اینکه بخواهیم از dilation یا erosion استفاده کنیم از فرمول‌های زیر استفاده خواهیم کرد:

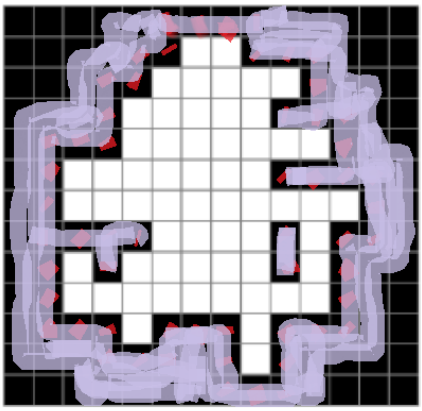
$$\beta(A) = A\ xor\ (A \ominus B)$$

$$\beta(A) = A\ xor\ (A \oplus B)$$

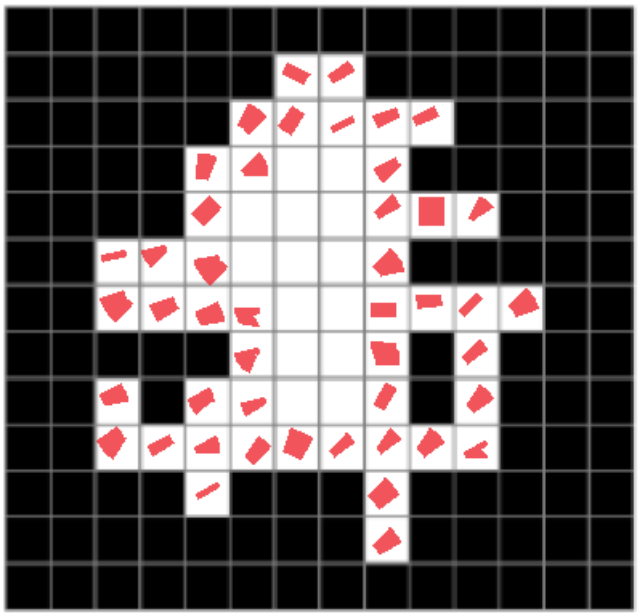
در ابتدا با dilate شروع می‌کنیم. عنصرهایی که غیر از عنصرهای خود شکل dilate خواهند شد به شکل زیر خواهند بود:



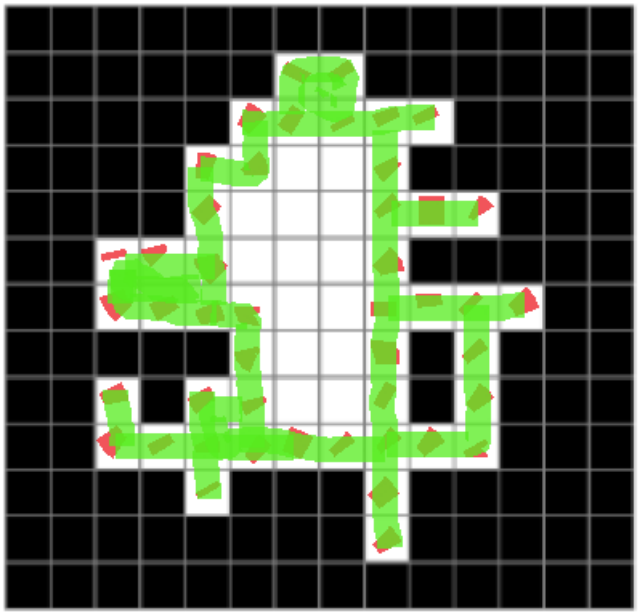
با توجه به نتیجه dilation و فرمول مرز مطابق تصویر زیر خواهد شد:



پیکسل‌های در erosion با این عنصر حذف خواهند شد به شکل زیر خواهند بود:



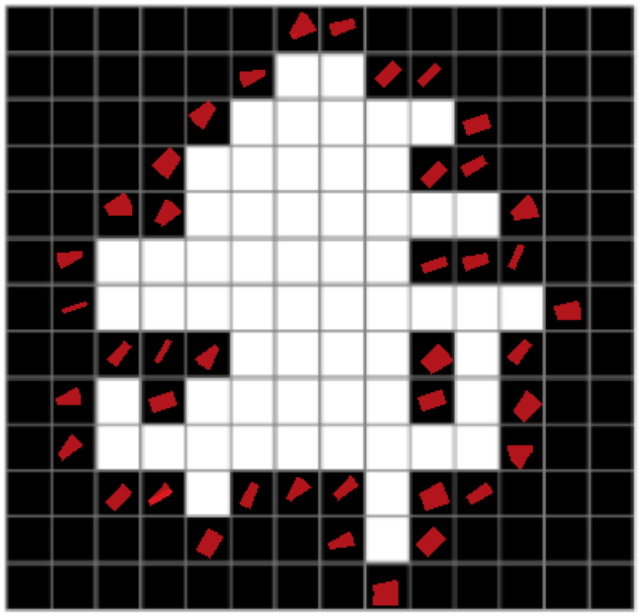
و مرز ما در این روش و با این عنصر ساختاری به شکل زیر خواهد بود:



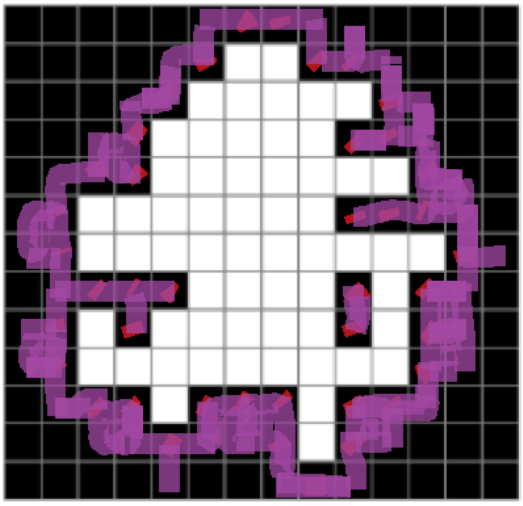
این بار از عنصر ساختاری زیر استفاده می‌کنیم:

0	1	0
1	1	1
0	1	0

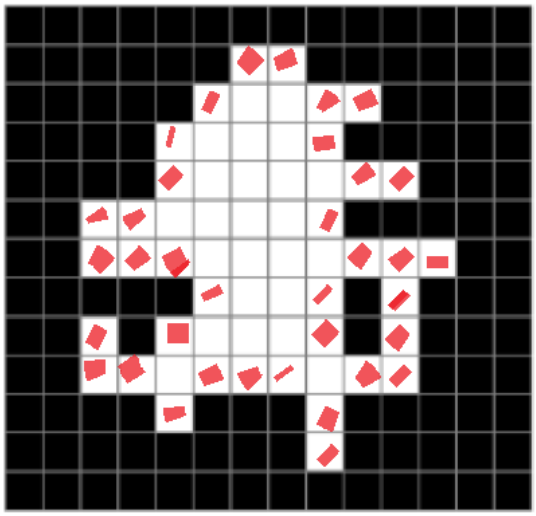
این بار عنصرهایی که علاوه بر پیکسل‌های شکل dilate خواهند شد به شکل زیر خواهند بود:



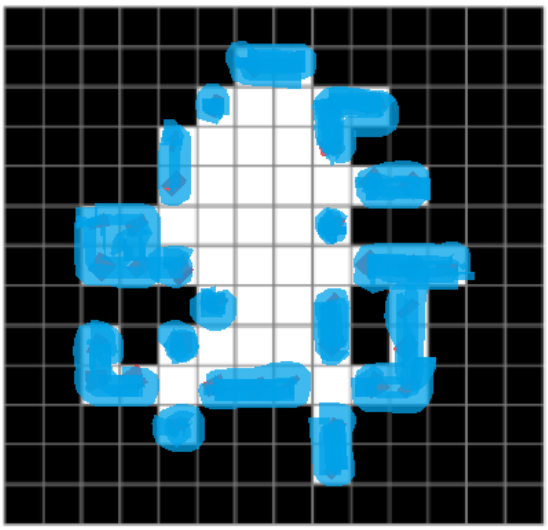
و مرز تصویر از این طریق به شکل زیر است:



پیکسل‌هایی که در عملیات erosion حذف خواهند شد به شکل زیر خواهند بود:



و در نهایت مرز شکل در این حالت نیز به شکل زیر خواهد شد:



۳- سوال سوم

ابتدا برای اینکه بتوانیم اسکلت تصویر را به دست آوریم لازم است که چند عملگر مورفولوژی را تعریف کنیم.

این کار را با پیاده‌سازی dilation شروع می‌کنیم:

```
def dilation(image: np.array, kernel: np.array) -> np.array:
    output = np.zeros_like(image)
    image_padded = np.zeros(
        (image.shape[0] + kernel.shape[0] - 1, image.shape[1] + kernel.sha
pe[1] - 1)
    )
    # Copy image to padded image
    image_padded[kernel.shape[0] - 2 : -1 :, kernel.shape[1] - 2 : -
1 :] = image
    # Iterate over image & apply kernel
    for x in range(image.shape[1]):
        for y in range(image.shape[0]):
            summation = (
                kernel * image_padded[y : y + kernel.shape[0], x : x + ker
nel.shape[1]]
            ).sum()
            output[y, x] = 255 if summation > 0 else 0
    return output
```

ورودی تابع dilation تصویر grayscale و عنصر ساختاری ما خواهند بود. چون این عملگرها مانند convolve کردن عمل می‌کنند برای اینکه ابعاد تصویر را نگه داریم لازم است در ابتدا padding انجام دهیم. پس از این کار روی هر پیکسل عملیات dilate را انجام می‌دهیم. به این صورت عمل می‌کنیم که مانند convolveکردن کرنل را در پنجره موردنظر در تصویر ضرب می‌کنیم و جمع عناصر آن را به دست می‌آوریم. اگر این جمع بیشتر از صفر باشد پیکسل موردنظر در تصویر ۲۵۵ خواهد شد و در غیر این صورت ۰ خواهدشد.

پیاده‌سازی erosion به شکل زیر است:

```
def erosion(image: np.array, kernel: np.array) -> np.array:
    output = np.zeros_like(image)
    image_padded = np.zeros(
        (image.shape[0] + kernel.shape[0] - 1, image.shape[1] + kernel.sha
pe[1] - 1)
    )
    # Copy image to padded image
    image_padded[kernel.shape[0] - 2 : -1 :, kernel.shape[1] - 2 : -
1 :] = image
    # Iterate over image & apply kernel
    for x in range(image.shape[1]):
        for y in range(image.shape[0]):
            summation = (
                kernel * image_padded[y : y + kernel.shape[0], x : x + ker
nel.shape[1]]
            ).sum()
```

```
            output[y, x] = 255 if summation == kernel.sum() * 255 else 0
    return output
```

در erosion تمام مراحل مانند dilation است اما تنها زمانی یک پیکسل ۲۵۵ می‌شود که مقدار جمعی که به دست می‌آوریم برابر جمع همه پیکسل‌های کرنل در ۲۵۵ باشد.

عملگر opening نیز به شکل زیر تعریف خواهد شد:

```
def opening(image: np.array, kernel: np.array) -> np.array:

  img_copy = image.copy()
  img_erode = erosion(img_copy, kernel)
  img_open = dilation(img_erode, kernel)

  return img_open
```

برای به دست آوردن اسکلت و هم چنین بازگشتن از اسکلت به تصویر از فرمول‌های زیر استفاده خواهیم کرد:

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

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

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

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

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

در قدم اول لازم است تصویر را باینری کنیم. سپس با k = 0 کار را شروع می‌کنیم:

```
ret, res = cv2.threshold(image,127,255,cv2.THRESH_BINARY_INV)
    # Get skeleton
    s_k=np.zeros(shape=image.shape)
    k=0
```

در مرحله اول که k = 0 است اسکلت با k = 0 برابر با تفاضل تصویر اصلی با opening آن با structuring element خواهد بود. در هر مرحله اسکلت را در یک لیست ذخیره خواهیم کرد تا در ادامه بتوانیم آن را در به دست آوردن تصویر اصلی استفاده کنیم:

```
skeleton_ak = res - openning(res,structure_element)
    sk_list.append(skeleton_ak)
    s_k=s_k + skeleton_ak
```

اینکه k تا چه اندازه بزرگ شود بسته به نتیجه هر بار erode کردن تصویر با عنصر ساختاری بستگی خواهد داشت. اگر پس از k بار erode کردن هنوز پیکسلی با مقدار ۲۵۵ در تصویر باشد باز هم به مقدار k اضافه خواهد شد.

پس از آن نتیجه opening تصویر erode شده با عنصر ساختاری را به دست خواهیم آورد و درنهایت نتیجه erosion را منهای نتیجه opening خواهیم کرد تا اسکلت مرحله بعدی به دست آید و برای به دست آوردن اسکلت نهایی اسکلت آن مرحله را به اسکلت‌های مراحل قبل اضافه می‌کنیم:

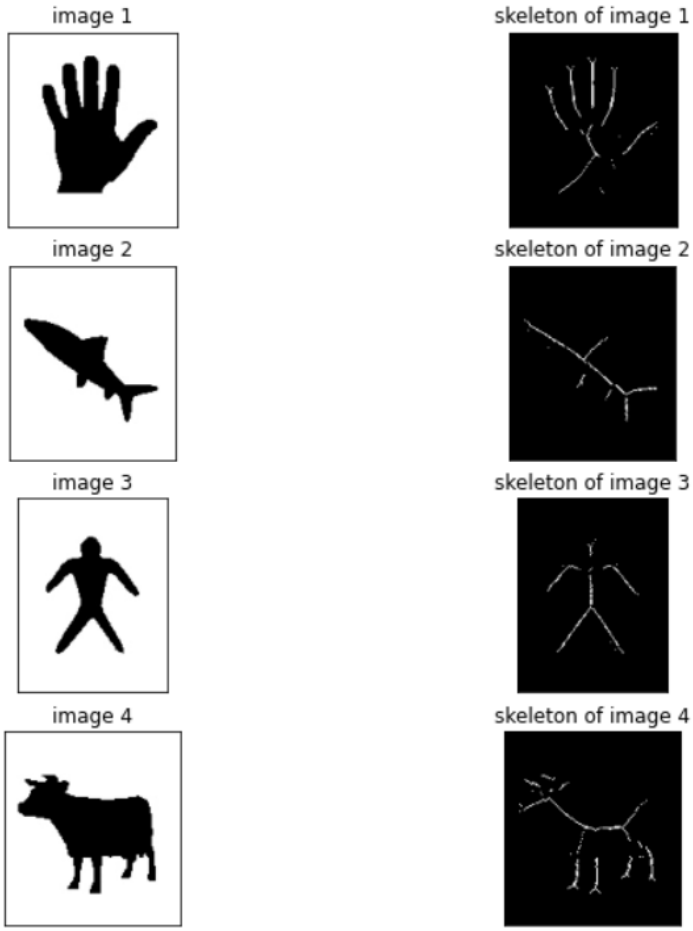
```
while 255 in erosion(res, structure_element):
      # Erode image
      res = erosion(res,structure_element)
      k += 1

      # Open image
      res_open = openning(res,structure_element)

      # Erode image - Open image
      skeleton_ak = res-res_open
      sk_list.append(skeleton_ak)

      # Union
      s_k=s_k + skeleton_ak
```

نتیجه اعمال این تابع روی تصاویر به شکل زیر خواهد بود:

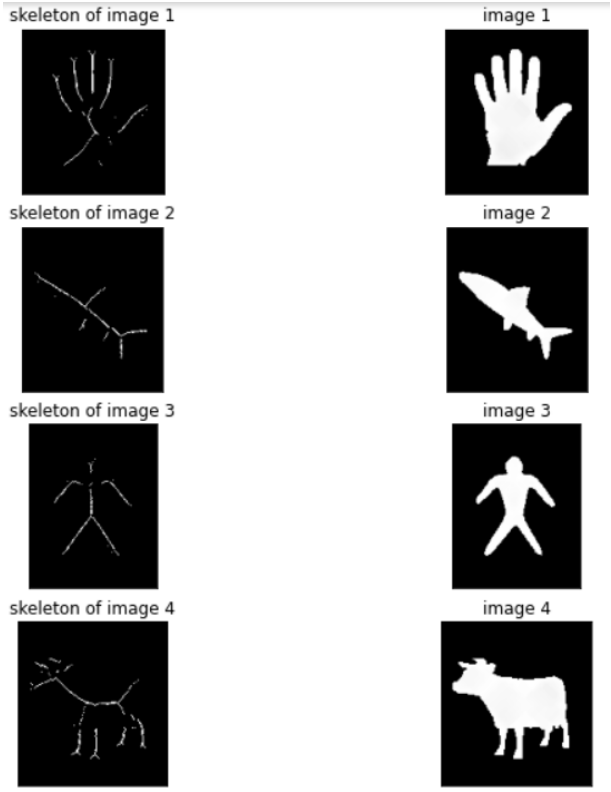


برای برگشتن از اسکلت به تصویر اصلی لازم است اسکلت‌هایی که در هر مرحله به دست آوردیم را k بار با عنصر ساختاری dilate کنیم و با هم دیگر جمع کنیم تا در نهایت به تصویر اصلی برسیم:

```
A = sk_list[0]
    for i in range(1, k):
        dilate_sk = sk_list[i]
        for j in range(i):
            # k dilation of skeleton k with structuring element
            dilate_sk = dilation(dilate_sk, structure_element)
        A += dilate_sk

    res = A
```

نتیجه اعمال این تابع به اسکلت‌ها به شکل زیر خواهد بود:



۴- سوال چهارم

در گام اول برای تشخیص ماشین‌ها تصویر را به gray تبدیل می‌کنیم و برای تشخیص بهتر آن را blur می‌کنیم. سپس adaptive threshold را روی آن اعمال می‌کنیم تا به تصویری با دو مقدار ۰ و ۲۵۵ تبدیل شود.

```
# Converting to gray, Blurring and adaptive threshold
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    thresh = cv2.adaptiveThreshold(blurred, 255, cv2.ADAPTIVE_THRESH_MEAN_
C, cv2.THRESH_BINARY_INV, 21, 18)
```

سپس در ابتدا برای حذف نویزها عملگر opening را به تصویر اعمال می‌کنیم:

```
# opening to remove noises
    kernelSize = (5, 5)
    kernel = cv2.getStructuringElement(cv2.MORPH_CROSS, kernelSize)
    opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)
```

در مرحله بعد عملگر closing را به تصویر اعمال کردم تا اگر حفره‌ای در تصویر موجود است از بین برود:

```
# closing - remove black holes
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, kernelSize)
    closing = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)
```

پس از آن عملگر dilation را به تصویر اعمال کردم تا خطوط عمودی و افقی از تصویر که مرتبط با ماشین هستند تقویت شوند:

```
# dilation - to amplify h and v borders
    kernel = np.ones((5, 5), np.uint8)
    kernel[:,0] = kernel[:,-1] = 0
    dilated = cv2.dilate(closing.copy(), kernel, iterations=1)
```

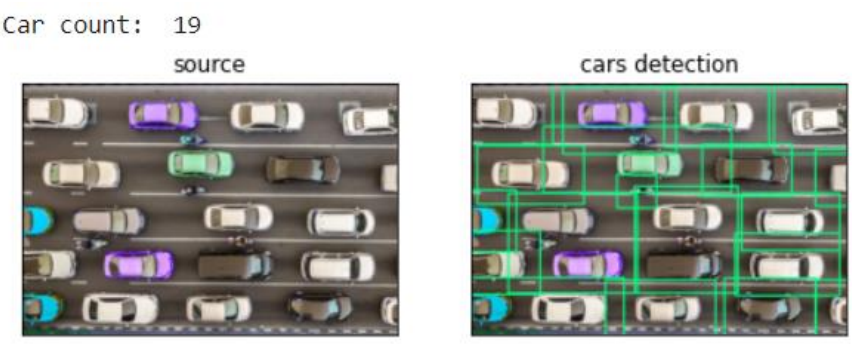
سپس به سراغ پیدا کردن countor های موجود در تصویر می‌رویم. برای هر countor مستطیلی که آن را احاطه می‌کند پیدا می‌کنیم. پارامترهای مستطیل را به دست می‌آوریم. از نظر ما مستطیلی مربوط به یک ماشین خواهد بود که عرض آن بزرگ‌تر از ۳۰ و ارتفاع آن بزرگ تر از ۱۵ باشد. در این صورت یک ماشین را نشان خواهد داد:

```
min_width_rec = 30
    min_height_rec = 15

    for c in cnts:
      rectangle = cv2.boundingRect(c)
      x = rectangle[0]
      y = rectangle[1]
      w = rectangle[2]
      h = rectangle[3]

      if w >= min_width_rec and h >= min_height_rec:
        cv2.rectangle(result, (x-15,y-
15), (x+w+15,y+h+15), (0,255,127), 2)
        cars_num += 1
```

نتیجه کار به صورت زیر خواهد بود:



منابع:

در سوال ۳ و ۴ از خانم نگار زین العابدین کمک گرفتم.