Computer Vision

Assignment 11

Alireza Moradi

December 1, 2020

1 Hit-or-Miss

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1: Hit-or-Miss

2

2.1 Compactness

$$compactness = \frac{4\pi \times Area}{Perimeter^2}$$

- $Area = a^2$
- $Perimeter = 4a$

$$compactness = \frac{4\pi \times a^2}{16a^2} = \frac{\pi}{4}$$

2.2 Eye Aspect Ratio

I will be computing a metric called the eye aspect ratio (EAR), introduced by Soukupová and Čech in their 2016 paper, Real-Time Eye Blink Detection Using Facial Landmarks.
Basically what it does is as follows:

for every video frame, the eye landmarks are detected. The eye aspect ratio (EAR) between height and width of the eye is computed.

$$EAR = \frac{\| p_2 - p_6 \| + \| p_3 - p_5 \|}{2 \| p_1 - p_4 \|} \tag{1}$$

where $p_1, ..., p_6$ are the 2D landmark locations, depicted in Figure 1.

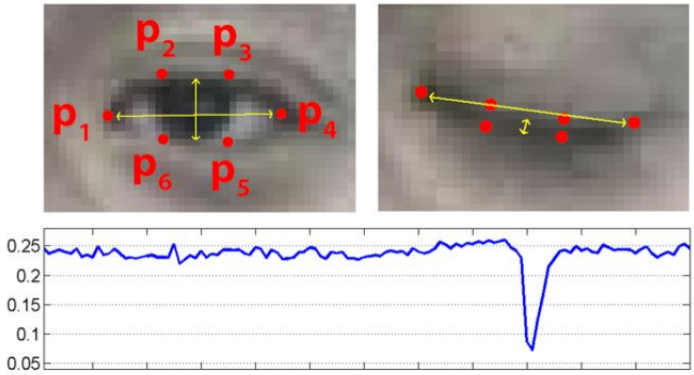


Figure 1: Open and closed eyes with landmarks pi automatically detected. The eye aspect ratio EAR in Eq. (1) plotted for several frames of a video sequence. A single blink is present.

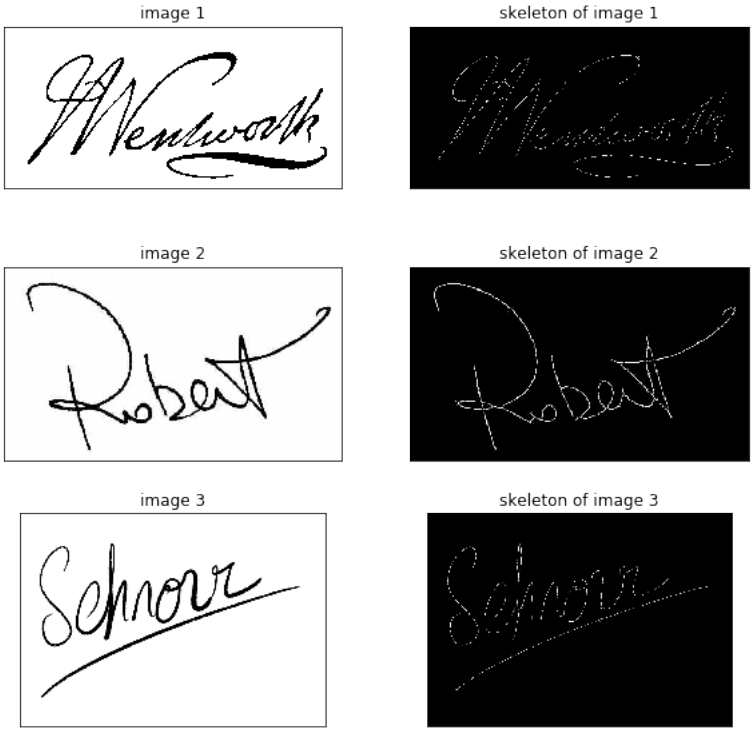
The EAR is mostly constant when an eye is open and is getting close to zero while closing an eye. It is partially person and head pose insensitive. Aspect ratio of the open eye has a small variance among indi- viduals and it is fully invariant to a uniform scaling of the image and in-plane rotation of the face. Since eye blinking is performed by both eyes synchronously, the EAR of both eyes is averaged.

3 Implementations

3.1 Skeletonization

First applied `inverted binary thresholding` on the input image, then I smoothed the image and finally used Morphological Skeletonization's equation as below to find the skeleton of each image:

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




```python
def get_skeleton(image):
    """
    Finds the skeleton of the input image.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        numpy.ndarray: The skeleton image.
    """
    image = np.float32(image)
    ret,image = cv2.threshold(image,200,255,cv2.THRESH_BINARY_INV)
```

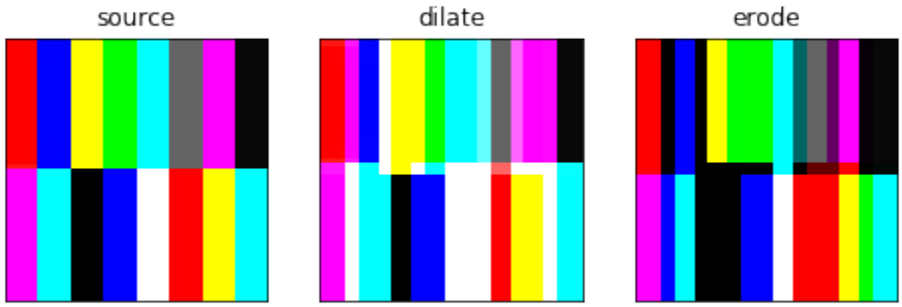
```
    res = image.copy()
    res = np.uint8(cv2.GaussianBlur(res,(3,3),1))
    element = cv2.getStructuringElement(cv2.MORPH_CROSS,(3,3))
    S = np.zeros_like(res,dtype=np.uint8)
    ab = np.uint8(res.copy())

    while True:
        ab = cv2.erode(ab,element)
        if len(np.where(ab > 0)[0])==0:
            break
        Sk = cv2.subtract(ab,
                          cv2.dilate(cv2.erode(ab,element),element))
        S = cv2.bitwise_or(S,Sk)

    return S
```

3.2 RGB Erode & Dilate

First I padded the image, then seperated the RGB channels and in each window in the image, I took the maximum value -to `dilate`- of each color channel (minimum value for `erosion`). Finally I merged all the RGB channels back together.




```
def RGB_dilate(image, structuring_element):
    '''
    Applies dilation in RGB space.

    Parameters:
        image (numpy.ndarray): The input image.
        structuring_element (numpy.ndarray): The structuring element must be square.

    Returns:
        dilated_image (numpy.ndarray): The dilated result image.
    '''

    img = image.copy()
    dilated_image = image.copy()
    size = structuring_element.shape[0]
    r = size//2

    image = cv2.copyMakeBorder(
    image,
    top=r,
    bottom=r,
    left=r,
    right=r,
    borderType=cv2.BORDER_REPLICATE)

    R = image[:,:,0]
    G = image[:,:,1]
    B = image[:,:,2]

    Ro = R.copy()
    Go = G.copy()
    Bo = B.copy()
```

```

    for i in range(Ro.shape[0]):
        for j in range(Ro.shape[1]):
            try:
                Ro[i,j] = np.max(R[i-r : i+r+1 , j-r : j+r+1])
                Go[i,j] = np.max(G[i-r : i+r+1 , j-r : j+r+1])
                Bo[i,j] = np.max(B[i-r : i+r+1 , j-r : j+r+1])
            except:
                continue

    dilated_image = cv2.merge([Ro,Go,Bo])
    return dilated_image[r:-r,r:-r,:]
```

```
def RGB_erode(image, structuring_element):
    '''
    Applies erosion in RGB space.

    Parameters:
        image (numpy.ndarray): The input image.
        structuring_element (numpy.ndarray): The structuring element must be square.

    Returns:
        eroded_image (numpy.ndarray): The eroded result image.
    '''

    img = image.copy()
    eroded_image = image.copy()
    size = structuring_element.shape[0]
    r = size//2

    image = cv2.copyMakeBorder(
    image,
    top=r,
    bottom=r,
    left=r,
    right=r,
    borderType=cv2.BORDER_REPLICATE)

    R = image[:,:,0]
    G = image[:,:,1]
    B = image[:,:,2]

    Ro = R.copy()
    Go = G.copy()
    Bo = B.copy()

    for i in range(Ro.shape[0]):
```

```
        for j in range(Ro.shape[1]):
            try:
                Ro[i,j] = np.min(R[i-r : i+r+1 , j-r : j+r+1])
                Go[i,j] = np.min(G[i-r : i+r+1 , j-r : j+r+1])
                Bo[i,j] = np.min(B[i-r : i+r+1 , j-r : j+r+1])
            except:
                continue

    eroded_image = cv2.merge([Ro,Go,Bo])
    return eroded_image[r:-r,r:-r,:]
```