

Refort

과제3 : Canny Edge Detection

담당 교수 : 감진규 교수님

수강과목 : 컴퓨터비전개론

학번 : 201524479

이름 : 변찬호

제출일자 : 2021년 04월 07일

1. Noise reduction (20 point) Load the image and switch to the grayscale image. It then blurs the image using the 'gaussconvolved2d(array,sigma)' function to eliminate noise.

Use PIL to show both the original and filtered images.

```
232 print("1. Noise reduction")
233
234 #이미지 불러오기
235 image = Image.open("iguana.bmp")
236 image.show()
237
238 #이미지 Grayscale 로 변환
239 image = image.convert('L')
240
241 #이미지 배열로 가져온다.
242 imageArray = np.asarray(image)
243
244 #gaussconvolve2d를 이용하여 Low filtering image를 구한다.(노이즈 제거)
245 filtered_image_array = gaussconvolve2d(imageArray,1.6)
246
247 #uint8의 타입으로 변환 후 array로 부터 PIL 파일로 바꾼다.
248 filtered_image = Image.fromarray(filtered_image_array.astype(np.uint8))
249 filtered_image.show()
```

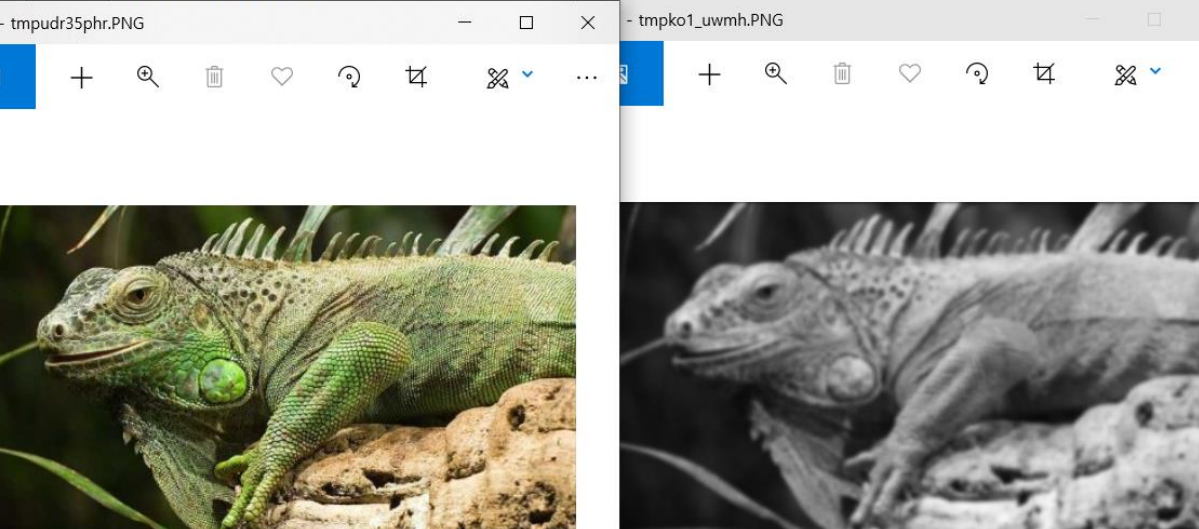


Figure 1-1 gaussconvolved2d를 이용한 결과 사진



Figure 1-2 원본사진

Figure 1-3 필터된 사진

```

"""
def gauss1d(sigma):
    # sigma 값이 반드시 0보다 커야하는 양수여야 한다.
    assert (sigma > 0), 'Sigma Value must be positive value'

    # 필터의 길이를 sigma의 6배를 하고 소수점을 반올림한 수를 넣는다..
    FilterLenth = int(math.ceil(float(sigma) * 6))

    # 필터의 길이가 짝수이면 +1해서 홀수로 만들어준다.
    if (FilterLenth % 2 == 0): FilterLenth = FilterLenth + 1

    # 중앙이 몇 번째 배열인지 알아하기 위해 필터 배열길이를 2로 나눈다.
    Middle = int(FilterLenth / 2)

    # x값이 중앙에서의 거리 차이만큼 값이 들어가도록 배열을 만든다.
    xValue = np.arange(-Middle, Middle + 1)

    # 가우시안 함수를 만들어 주었다.
    Gaussian = lambda x: np.exp(-(x ** 2) / (2 * sigma ** 2))

    # 가우시안의 공식에 x값을 넣어준 후 결과를 받는다.
    GaussianResult = Gaussian(xValue)

    # 배열을 정규화하여 합이 1이 되도록 한다.
    Result = GaussianResult / np.sum(GaussianResult)

    return Result

```

Figure 1-4 gauss1d 함수

```

def gauss2d(sigma):
    # 1D array를 함수gauss1d를 통해 받아온다.
    Gauss1D = gauss1d(sigma)

    # 1D array에 외적(outer product)을 하여 2D array로 바꾸었다.
    Result = np.outer(Gauss1D, Gauss1D)

    total = np.sum(Result) # 가우시안 필터의 합을 구하기 위한 변수
    Result /= total # 정상화로 합이 1 이 되게 한다

    return Result

```

Figure 1-5 gauss2d 함수

```

def convolve2d(array,filter):
    # convolution하기 위해서는 일단 filter를 일단 위아래 좌우를 바꿔준다...
    filter = np.flipud(np.fliplr(filter))

    # 커널과 이미지의 shape를 구한다.
    xKernelShape = filter.shape[0] # Filter X
    yKernelShape = filter.shape[1] # Filter Y
    xImgShape = array.shape[0] # Array(Image) X
    yImgShape = array.shape[1] # Array(Image) Y

    # 출력에 대한 Array 선언 및 타입 설정
    Output = np.zeros((xImgShape, yImgShape))
    Output = Output.astype('float32')

    # zero padding된 공간 값을 찾는다 --> m = (f-1) / 2
    Padding = int((xKernelShape - 1) / 2)

    # Padding 을 적용한다.
    if Padding != 0:
        ImagePadded = np.zeros((xImgShape + Padding * 2, yImgShape + Padding * 2))
        ImagePadded[Padding:-Padding, Padding:-Padding] = array
    else:
        ImagePadded = array # padding이 없으면 원래 이미지의 array를 따른다.

    # 합성공을 실행한다. 실질적으로 cross-correlation한다.
    for y in range(yImgShape):
        for x in range(xImgShape):
            Output[x, y] = (filter * ImagePadded[x:x + xKernelShape, y:y + yKernelShape]).sum()

    return Output

```

Figure 1-6 convolve2d 함수

```

def gaussconvolve2d(array,sigma):
    # 필터로 가우시안 2D를 받아온다.
    Gauss2dFilter = gauss2d(sigma)

    # convolve2d에 적용한다.
    Result = convolve2d(array, Gauss2dFilter)

    return Result

```

Figure 2 gaussconvolve2d 함수

2. Finding the intensity gradient of the image (20 point) Write a function 'sobel_filter(img)' that applies Sobel filter in the x,y direction and apply the convolve2d function to obtain the intensity x,y value. And apply the formula to obtain Gradient and Theta values. Note that gradient values should be mapped to values between 0-255. And the function must return both gradient and theta

Show the results of applying sobel_filter Suppression

```
print("2. Finding the intensity gradient of the image")
G, theta = sobel_filters(filtered_image_array)
G_image = Image.fromarray(G.astype(np.uint8))
G_image.show()
```

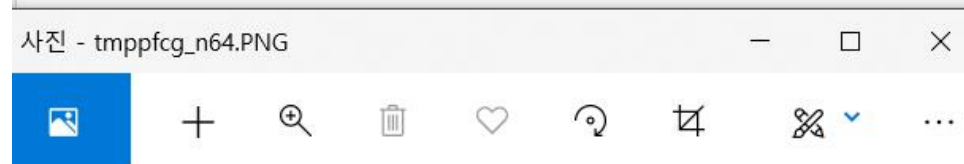


Figure 2-1 sobel filter suppression 적용된 결과


```

def sobel_filters(img):
    """ Returns gradient magnitude and direction of input img.
    Args:
        img: Grayscale image. Numpy array of shape (H, W).
    Returns:
        G: Magnitude of gradient at each pixel in img.
            Numpy array of shape (H, W).
        theta: Direction of gradient at each pixel in img.
            Numpy array of shape (H, W).
    Hints:
        - Use np.hypot and np.arctan2 to calculate square root and arctan
    """

    #Sobel filter 설정
    xSobel_filter = np.array([[ -1,  0,  1], [ -2,  0,  2], [ -1,  0,  1]], np.float32)
    ySobel_filter = np.array([[ 1,  2,  1], [ 0,  0,  0], [ -1, -2, -1]], np.float32)

    #Sobel filter 와 blur(노이즈 제거된) 이미지를 convolve한다.
    xSobel_array = convolve2d(img, xSobel_filter)
    ySobel_array = convolve2d(img, ySobel_filter)

    #gradient value 값을 구한다.
    G = np.hypot(xSobel_array, ySobel_array)

    #0~255로 맵핑
    G = (G / G.max()) * 255

    #theta 값을 구한다.
    theta = np.arctan2(ySobel_array, xSobel_array)

    return (G, theta)

```

Figure 2-2 sobel_filters 함수

3. Non-Maximum Suppression (20 point) Write a function 'non_max_suppression(gradient, theta)' that extract the thin edge by applying Non-max-suppression according to the angle of the edge (0,45,90,135).

Show the results of applying Non-Maximum Suppression.

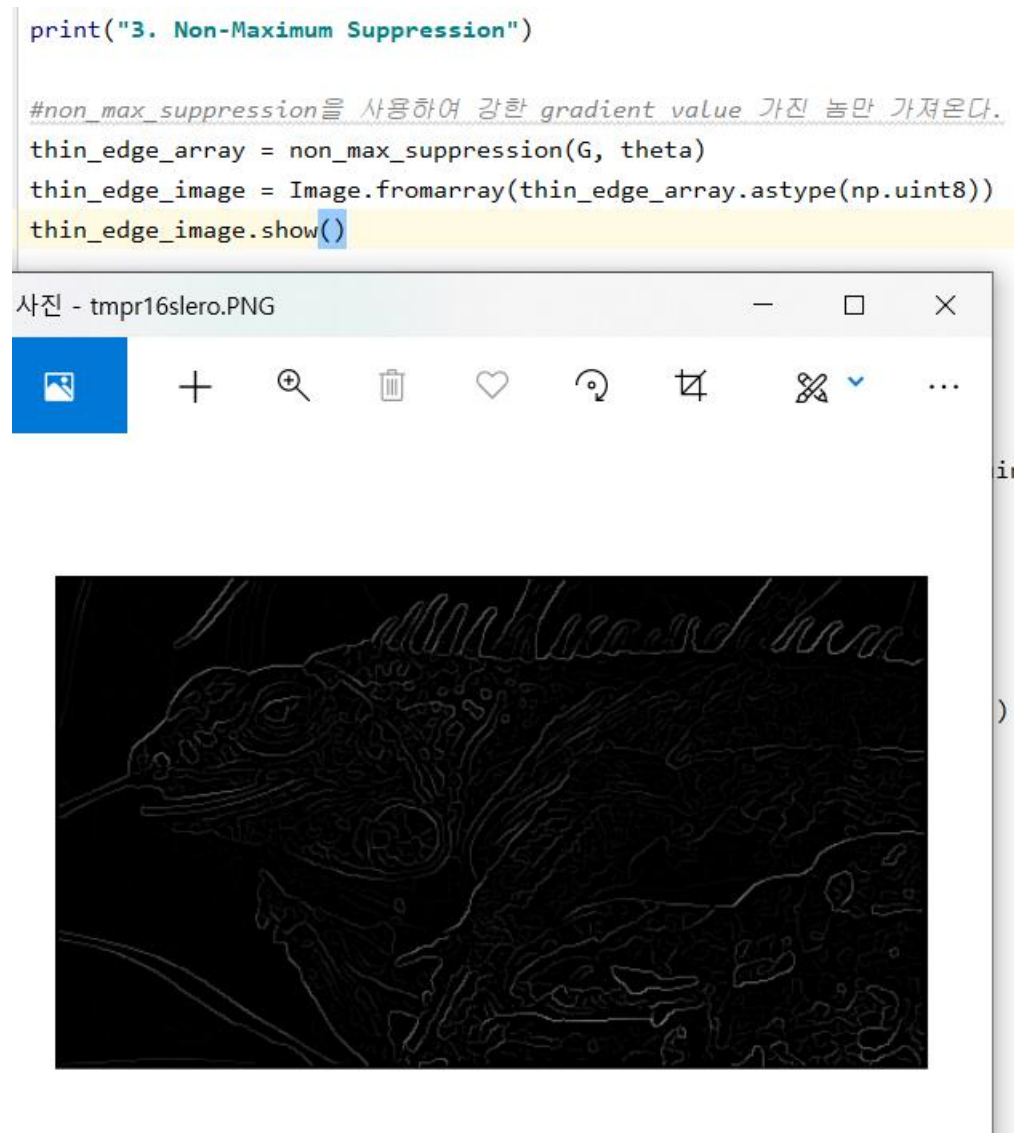


Figure 3-1 Non-Maximum Suppression 적용 결과

```

def non_max_suppression(G, theta):
    """ Performs non-maximum suppression.
    This function performs non-maximum suppression along the direction
    of gradient (theta) on the gradient magnitude image (G).
    Args:
        G: gradient magnitude image with shape of (H, W).
        theta: direction of gradients with shape of (H, W).
    Returns:
        res: non-maxima suppressed image.
    """
    #결과를 받을 변수 0 배열로 초기화.
    res = np.zeros(G.shape)

    #theta를 각도로 바꾼다.
    angle = theta * 180 / np.pi
    angle[angle < 0] += 180

    #Non-Maximum Suppression 적용
    for i in range(1, int(G.shape[0] - 1)) :
        for j in range(1, int(G.shape[1] - 1)) :
            #초기화
            nearPoint1 = 0;
            nearPoint2 = 0;

            #각도가 0일경우
            if(0 <= angle[i, j] < 22.5) or (157.5 <= angle[i, j] <= 180) :
                nearPoint1 = G[i, j + 1]
                nearPoint2 = G[i, j - 1]

            # 각도가 45일경우
            elif (22.5 <= angle[i,j] < 67.5):
                nearPoint1 = G[i + 1, j - 1]
                nearPoint2 = G[i - 1, j + 1]

            # 각도가 90일경우
            elif (67.5 <= angle[i,j] < 112.5):
                nearPoint1 = G[i + 1, j]
                nearPoint2 = G[i - 1, j]

            # 각도가 135일경우
            else :
                # 인접한 것중 gradient value가 제일 크면 그대로 저장하고 아니면 지운다.
                nearPoint1 = G[i - 1, j - 1]
                nearPoint2 = G[i + 1, j + 1]

            #인접한 것중 intensity value가 제일 크면 그대로 저장하고 아니면 지운다.
            if((G[i,j] > nearPoint1) and (G[i, j] > nearPoint2)) :
                res[i, j] = G[i, j]

            else :
                res[i, j] = 0

```

Figure 3-2 non_max_suppression 함수

4. Double threshold (20 point) Write a function 'double_thresholding(img)' that double threshold step aims at identifying 3 kinds of pixels: strong, weak, and non-relevant

1) Strong pixels are pixels that have an intensity so high that we are sure they contribute to the final edge.

2) Weak pixels are pixels that have an intensity value that is not enough to be considered as strong ones, but yet not small enough to be considered as nonrelevant for the edge detection.

3) Other pixels are considered as non-relevant for the edge.

Use the following expressions to determine high and low threshold values.

$$diff = \max(image) - \min(image)$$
$$Thigh = \min(image) + diff * 0.15$$
$$Tlow = \min(image) + diff * 0.03$$

After classifying into three types, map non-related items to zero, map weak to 80, and map strong to 255. **Show the results of applying Double threshold**

```
print("4. Double threshold")
```

```
#threshold 실행
```

```
threshold_image_array = double_thresholding(thin_edge_array)
```

```
threshold_image = Image.fromarray(threshold_image_array.astype(np.uint8))
```

```
threshold_image.show()
```

사진 - tmp1p6hx9i7.PNG

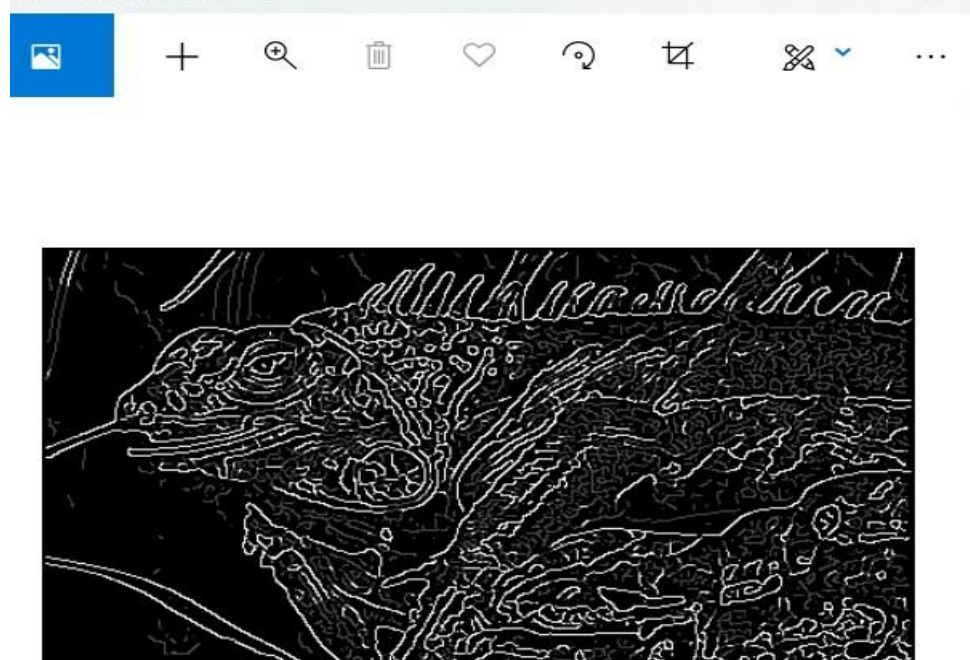


Figure 4-1 Double threshold 결과

```

def double_thresholding(img):
    """
    Args:
        img: numpy array of shape (H, W ) representing NMS edge response.
    Returns:
        res: double_thresholded image.
    """
    # res img크기만큼 0으로 초기화
    res = np.zeros(img.shape)

    #diff 값 과 T_high, T_low 값 구하기
    diff = np.max(img) - np.min(img)
    T_high = np.min(img) + (diff * 0.15)
    T_low = np.min(img) + (diff * 0.03)

    #strong한 부분과, weak 부분, 관계없는 부분을 찾아낸다
    strongPart = np.where(img >= T_high)
    zeroPart = np.where(img < T_low)
    weakPart = np.where((img >= T_low) & (img < T_high))

    #strong 부분은 255으로 weak 부분은 80으로 관계없는 것은 0으로 맵핑한다.
    res[strongPart] = 255;
    res[zeroPart] = 0;
    res[weakPart] = 80;

    return res

```

Figure 4-2 double_thresholding 코드

5. Edge Tracking by hysteresis (20 point) Write a function 'hysteresis(array)' that based on the threshold results, the hysteresis consists of transforming weak pixels into strong ones, if and only if at least one of the pixels around the one being processed is a strong one.

Show the results of applying hysteresis.

```
print("5. Edge Tracking by hysteresis")
#hysteresis 실행
hysteresis_array = hysteresis(threshold_image_array)
hysteresis_image = Image.fromarray(hysteresis_array.astype(np.uint8))

hysteresis_image.show()
```

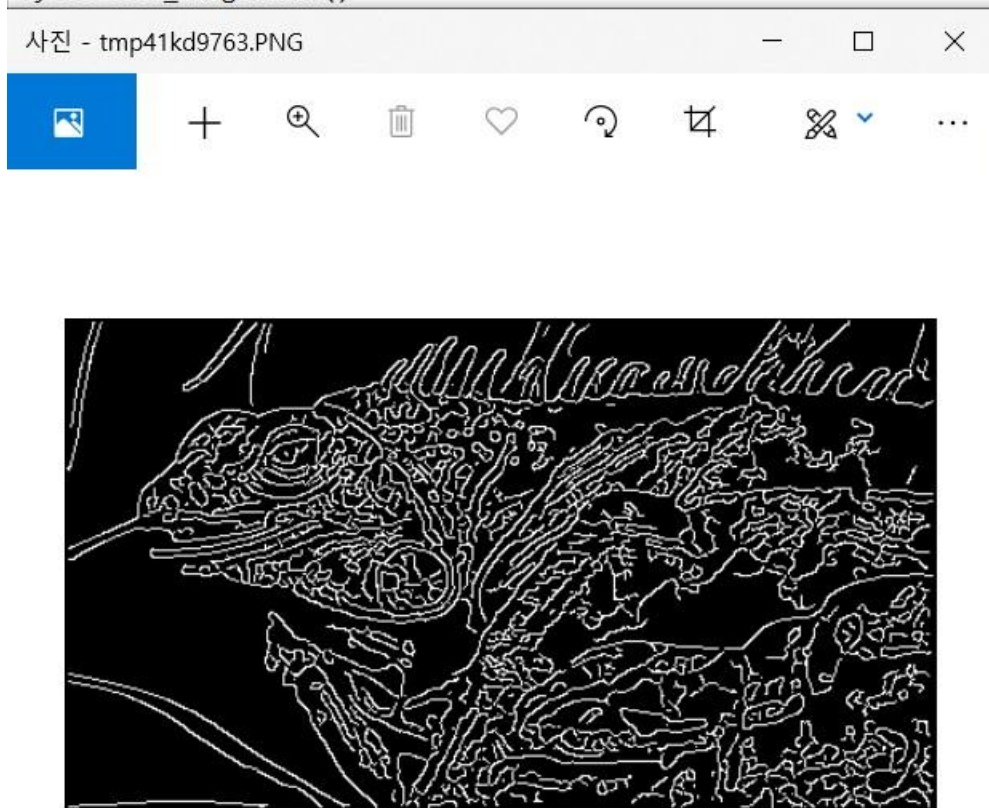


Figure 5-1 hysteresis 적용 결과

```

def hysteresis(img):
    """ Find weak edges connected to strong edges and link them.
    Iterate over each pixel in strong_edges and perform depth first
    search across the connected pixels in weak_edges to link them.
    Here we consider a pixel (a, b) is connected to a pixel (c, d)
    if (a, b) is one of the eight neighboring pixels of (c, d).
    Args:
        img: numpy array of shape (H, W) representing NMS edge response.
    Returns:
        res: hysteresised image.
    """
    zero = 0
    weak = 80
    strong = 255

    #DFS
    while(1):
        numStrongBefore = np.sum(img == strong)
        print(numStrongBefore)
        for i in range(1, img.shape[0] - 1):
            for j in range(1, img.shape[1] - 1):
                if(img[i,j] == weak):
                    if((img[i, j + 1] == strong) or (img[i, j - 1] == strong) or
                       (img[i + 1, j] == strong) or (img[i - 1, j] == strong) or
                       (img[i + 1, j - 1] == strong) or (img[i - 1, j - 1] == strong) or
                       (img[i + 1, j + 1] == strong) or (img[i - 1, j + 1] == strong)):
                        img[i,j] = strong
                    numStrongAfter = np.sum(img == strong)

                    if(numStrongBefore == numStrongAfter):
                        #DFS 끝난 후 weak 제거
                        img = np.where(img != weak, img, zero)
                        break

    res = img

    return res

```

Figure 5-2 hysteresis 코드