Metody Rozpoznawania Obrazów
Zadanie 01 - Życie na krawędzi

Wojciech Michaluk

12.10.2025

Opis kolejnych etapów zadania

Jako obraz do analizy wybrałem *lena.png*:

Przeskalowałem piksele z zakresu $[0, 255]$ do $[0, 1]$ i przekonwertowałem na odcienie szarości:

```
# przeskalowanie wartości pikseli
lena = lena / 255.0

# konwersja do skali szarości - średnia ważona
# na podstawie: https://brandonrohrer.com/convert_rgb_to_grayscale.html
lena_gray = 0.2126 * lena[:,:,0] + 0.7152 * lena[:,:,1] + 0.0722 * lena[:,:,2]
```

Listing 1: Przeskalowanie wartości pikseli i konwersja do skali szarości

Jako efekt uzyskuję:

Następnie zmniejszam rozmiar obrazu poprzez zastosowanie *average poolingu*. Wybrałem ten rodzaj poolingu, ponieważ jako wejście do algorytmu Canny'ego chcemy mieć "rozmyty" obraz (stąd później filtracja Gaussa). Average pooling pozwala przybliżyć do tego celu lepiej niż max pooling, który może powodować zakłócenia.

```
# pooling 4x4 w celu zmniejszenia rozmiaru obrazu
pooling = torch.nn.AvgPool2d(kernel_size=(4, 4), stride=4)
lena_pooled = pooling(torch.tensor(lena_gray).unsqueeze(0).unsqueeze(0))
    .squeeze().numpy()
```

Listing 2: Average pooling 4x4

Obraz po poolingu:

Następnym etapem jest filtracja Gaussa. Funkcja przygotowująca filtr jest zaczerpnięta z artykułu dołączonego do zadania.

```
def gaussian_kernel(size, sigma):
    size = int(size) // 2
    x, y = np.mgrid[-size:size+1, -size:size+1]
    normal = 1. / (2. * np.pi * sigma**2)
    g = np.exp(-((x**2 + y**2) / (2. * sigma**2))) * normal

    return g
```

Listing 3: Funkcja do generowania filtra Gaussa zadanego rozmiaru

Sprawdzając różne wartości parametrów, uznałem że dla `size=5` oraz `sigma=0.75` wyniki są sensowne.

```
# zastosowanie filtra Gaussa do obrazu
gauss = gaussian_kernel(size=5, sigma=0.75)
lena_gauss = cv2.filter2D(
    lena_pooled, -1, gauss, anchor=(-1, -1), delta=0, borderType=cv2.BORDER_DEFAULT
)
```

Listing 4: Realizacja filtracji Gaussa z zadanymi parametrami

Kolejny krok to wyznaczenie gradientu intensywności. Poniższa funkcja (zmodyfikowana względem artykułu) oblicza składowe poziomą i pionową gradientu oraz efekt scalenia i obraz kierunków gradientu.

```
def sobel_filters(image):
    Kx = np.array([[-1, 0, 1],
                   [-2, 0, 2],
                   [-1, 0, 1]], np.float32)

    Ky = np.array([[1, 2, 1],
                   [0, 0, 0],
                   [-1, -2, -1]], np.float32)

    Ix = cv2.filter2D(
        image, -1, Kx, anchor=(-1, -1), delta=0, borderType=cv2.BORDER_DEFAULT
    )

    Iy = cv2.filter2D(
        image, -1, Ky, anchor=(-1, -1), delta=0, borderType=cv2.BORDER_DEFAULT
    )

    grad = np.hypot(Ix, Iy)
```

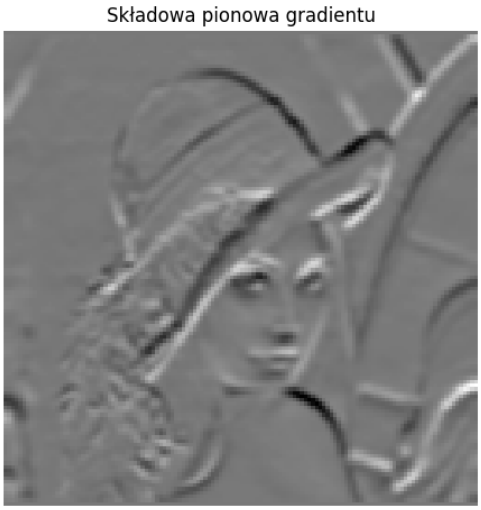
```
    grad = grad / grad.max() * 255
    theta = np.arctan2(Iy, Ix)

    return (Ix, Iy, grad, theta)
```

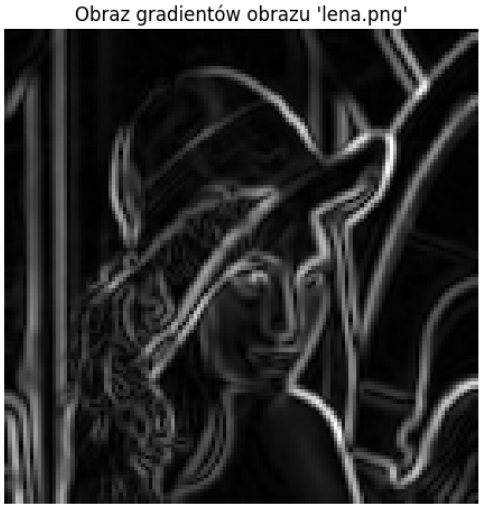
Listing 5: Funkcja do wyznaczania gradientu i kierunku krawędzi

Stosuję tę funkcję na otrzymanym wcześniej obrazie i przedstawiam wyniki:

```
# wyznaczenie gradientów i kierunków krawędzi
grad_x, grad_y, lena_grad, theta = sobel_filters(lena_gauss)
```

Listing 6: Aplikacja powyższej funkcji





Przychodzi pora na *non-maximum suppression*. Funkcję realizującą ten etap przygotowałem na podstawie załączonych materiałów.

```
def non_max_suppression(image, theta):
    H, W = image.shape
    I = np.zeros((H, W), dtype=np.int32)
    angles = theta * 180. / np.pi
    angles[angles < 0] += 180

    for i in range(1, H-1):
        for j in range(1, W-1):
            if 0 <= angles[i, j] < 22.5 or 157.5 <= angles[i, j] <= 180:
                q = image[i, j+1]
                r = image[i, j-1]
            elif 22.5 <= angles[i,j] < 67.5:
                q = image[i+1, j-1]
                r = image[i-1, j+1]
            elif 67.5 <= angles[i,j] < 112.5:
                q = image[i+1, j]
                r = image[i-1, j]
            elif 112.5 <= angles[i,j] < 157.5:
                q = image[i-1, j-1]
                r = image[i+1, j+1]

            if image[i, j] >= q and image[i, j] >= r:
                I[i, j] = image[i, j]

    return I
```

Listing 7: Funkcja realizująca non-maximum suppression

Przekazując do tej funkcji uzyskane w poprzednim etapie obraz gradientów (`lena_grad`) oraz kierunków gradientu (`theta`) otrzymuję taki obraz:

W następnej części nieco odchodzę od "klasycznej" metody Canny'ego. Wykonuję progowanie, ale pojedyncze, badając różne wartości progu i wybierając najlepszą.
W algorytmie Canny'ego w tym miejscu wykonywalibyśmy podwójne progowanie, wyznaczając krawędzie "mocne" oraz "słabe", co finalnie zapewne dałoby dokładniejszy efekt. Wykonana tutaj procedura jest łatwiejsza i szybsza, ale mniej dokładna.

```
def thresholding(image, thresh):
    H, W = image.shape
    I = np.zeros((H, W), dtype=np.int32)
    edge_i, edge_j = np.where(image >= thresh)
    I[edge_i, edge_j] = 1

    return I
```

Listing 8: Funkcja realizująca progowanie

Spośród tutaj przedstawionych najlepszy wydaje się wynik dla progu 25, zatem wezmę go do finalnego etapu. Najpierw jednak porównajmy ten obraz z rezultatem bibliotecznej funkcji:

```
# wybieram najlepszy próg
lena_th = thresholding(lena_nms, thresh=25)

# użycie funkcji bibliotecznej
lena_gauss_u8 = (255 * lena_gauss).astype(np.uint8)
lena_lib = cv2.Canny(lena_gauss_u8, 25, 60, None, 3, 1)
```

Listing 9: Przygotowanie obrazów do porównania

Parametry funkcji dostosowałem tak, żeby były podobne do mojej realizacji (progi 25 oraz 60, rozmiar filtra Sobela wynoszący 3). Widać, że funkcja biblioteczna pozwoliła wykryć krawędzie dokładniej, jest ich więcej, ale własna implementacja znacznie nie odstaje.

Na koniec scalę wyniki (obraz krawędzi) z oryginalnym obrazem, żeby zobaczyć, czy uzyskany rezultat jest sensowny.

```
# przeskalowanie obrazu krawędzi do oryginalnego rozmiaru
lena_res = cv2.resize(
    lena_th, (lena.shape[1], lena.shape[0]), interpolation=cv2.INTER_NEAREST
)

lena_res_3d = np.zeros_like(lena)
lena_res_3d[:,:,0] = lena_res_3d[:,:,1] = lena_res_3d[:,:,2] = lena_res

# nałożenie krawędzi na oryginalny obraz
comb = 0.5 * lena + 1. * lena_res_3d
```

Listing 10: Scalenie wyników z pierwotnym obrazem

Wynik wydaje się w porządku. Większość krawędzi dobrze odwzorowuje kontury obiektów i zgadza się z wizualnym odbiorem oka ludzkiego.