Raport z zadania 1. - Wykrywanie krawędzi

Michał Stefanik

9 października 2023

1 Co trzeba było zrobić?

Własna implementacja algorytmu wykrywania krawędzi Canny'ego. Do tego celu użyłem pythona z pytorchem.

2 Oryginalny obrazek

Załadowane zdjęcie widać na rysunku 1.

Rysunek 1: Oryginalny obrazek

3 Grayscale

Do konwersji obrazka na skalę szarości użyłem kowolucji. Jako wartość szarości została wybrana średnia z wartości RGB. Domyślne wartości to: padding=0, stride=1. Widoczne na obrazku 2.

```
conv = nn.Conv2d(3, 1, kernel_size=1, bias=False)
conv.weight = nn.Parameter(torch.ones(1, 3, 1, 1) / 3)
grayscale = conv(tensor.unsqueeze(0)).squeeze(0)
```

Rysunek 2: Oryginalny obrazek w skali szarości

4 Zmniejszenie rozmiaru obrazka

Do zmniejszenia obrazka użyłem poolingu maksymalnego. W przypadku krawędzi zależałoby nam, żeby jasny obszar pozostał bardzo jasny, a nie był ściemniany przez okoliczne piksele. Wydaje się, że max pooling jest w tym przypadku lepszy niż average pooling. Widoczne na obrazku 3.

Rysunek 3: Obrazek w skali szarości o dwukrotnie mniejszym rozmiarze

5 Filtr Gaussa

Użyłem filtru Gaussa o rozmiarze 7x7 i odchyleniu standardowym 1. Doszedłem do tych wartości eksperymentalnie. Maskę tworzyłem w następujący sposób:

```
def gauss_kernel(size, sigma=1):
    kernel = torch.zeros(1, 1, size, size)
    half_size = size // 2
    double_sigma_sq = 2 * sigma**2
    for i in range(size):
        for j in range(size):
            kernel[0, 0, i, j] = exp(
                -((i - half_size) ** 2 + \
                (j - half_size) ** 2) / (double_sigma_sq)
            )
    return kernel / kernel.sum()
```

Wynik tej operacji przedstawiłem na obrazku 4.

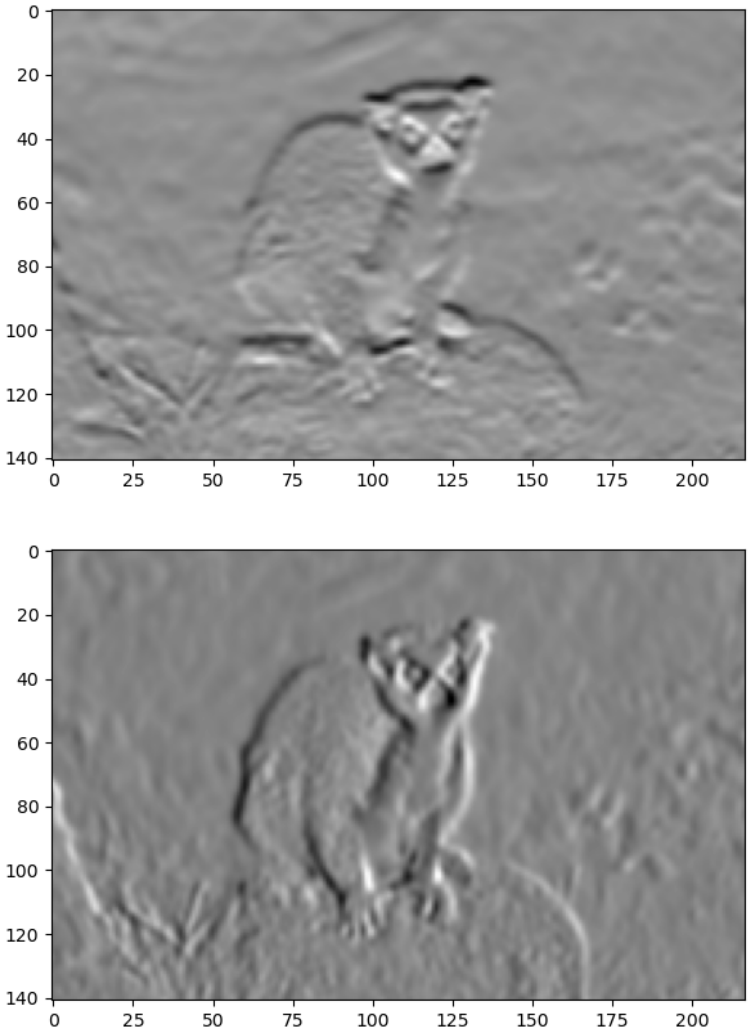
Rysunek 4: Obrazek po zastosowaniu filtru Gaussa

6 Gradienty

W dalszej kolejności zastosowałem filtry Sobela. Implementacja poniżej:

```
sob_vert = torch.FloatTensor([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])
sob_hori = torch.FloatTensor([[1, 0, -1], [2, 0, -2], [1, 0, -1]])

conv = nn.Conv2d(1, 2, kernel_size=3, stride=1, bias=False)
conv.weight = nn.Parameter(torch.stack([sob_vert.unsqueeze(0), sob_hori.unsqueeze(0)]))
sobel = conv(gauss.unsqueeze(0)).squeeze(0)
```

Rysunek 5 przedstawia gradienty w obu kanałach.



Rysunek 5: Gradienty obrazka

Dane z obu kanałów połączyłem w jeden kanał intensywności, używając normy euklidesowej. Można ten etap zobaczyć na obrazku 6.

Rysunek 6: Norma gradientów obrazka

Do liczenia kąta użyłem funkcji torch.atan2.

7 Wykrywanie krawędzi

Do sprawdzania czy dany piksel jest krawędzią użyłem poniższej implementacji:

```
def non_max_supression(intensity, angle):
    assert intensity.shape == angle.shape
    assert intensity.dim() == 2

    max_intensity = intensity.max()
    height, width = intensity.shape

    # create output tensor
    output = torch.zeros(height, width)

    for i in range(height):
        for j in range(width):

            try:
                a = max_intensity
                b = max_intensity
                # 0 degrees
                current_angle = angle[i, j]
                if current_angle < 22.5 or current_angle > 157.5:
                    a = intensity[i, j + 1]
                    b = intensity[i, j - 1]
                # 45 degrees
                elif 22.5 < current_angle < 67.5:
                    a = intensity[i + 1, j + 1]
                    b = intensity[i - 1, j - 1]
```

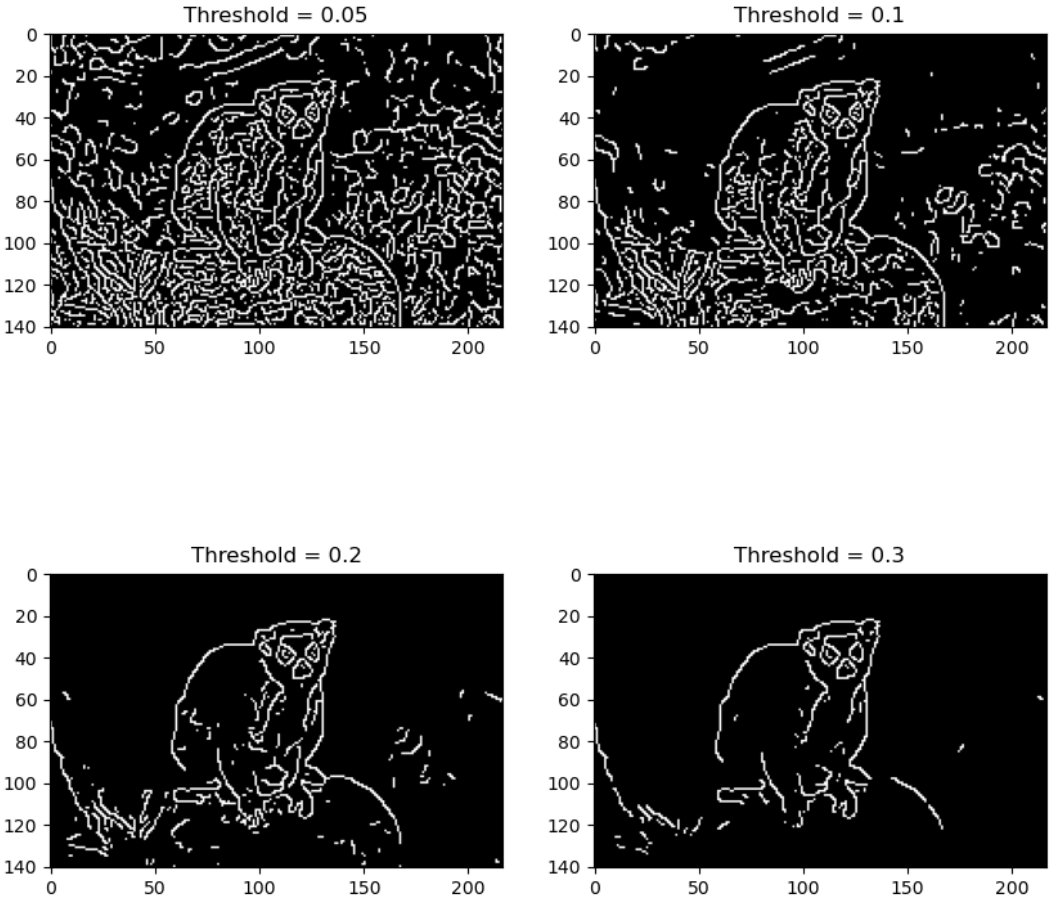
```
                # 90 degrees
                elif 67.5 < current_angle < 112.5:
                    a = intensity[i + 1, j]
                    b = intensity[i - 1, j]
                # 135 degrees
                elif 112.5 < current_angle < 157.5:
                    a = intensity[i - 1, j + 1]
                    b = intensity[i + 1, j - 1]

                if intensity[i, j] >= a and intensity[i, j] >= b:
                    output[i, j] = intensity[i, j]

            except IndexError:
                pass
    return output / output.max()
```

Wynik tej operacji przedstawiłem na obrazku 7.

Rysunek 7: Wykryte krawędzie

8 Progowanie

W tym kroku wartości mniejsze niż dany próg zostały ustawione na 0, a większe zostały ustawione na 1. Testowane dla różnych wartości progu. Widoczne na obrazku 8. W porównaniu do oryginalnej metody Canny'ego, tutaj stosujemy jedynie jeden próg, a nie dwa. Spowoduje to, że nasze krawędzie będą urywane w miejscach, gdzie jest nadal krawędź, ale nie tak wyraźna.



Rysunek 8: Wykryte krawędzie

9 Wynik

Na koniec oryginalny obraz i krawędzie nałożyłem na siebie dodając je. Znalezione krawędzie są widoczne na obrazku 9. Część możemy uznać na nieważne, ale większość ćhcianych"krawędzi została wykryta. Większość znalezionych krawędzi ma sens.

Rysunek 9: Wynik

Dla porównania tutaj wynik z biblioteki OpenCV z progami 100 i 200:

Rysunek 10: Wynik z OpenCV