Zadanie 01

Krzysztof Dziechciarz

Życie na krawędzi

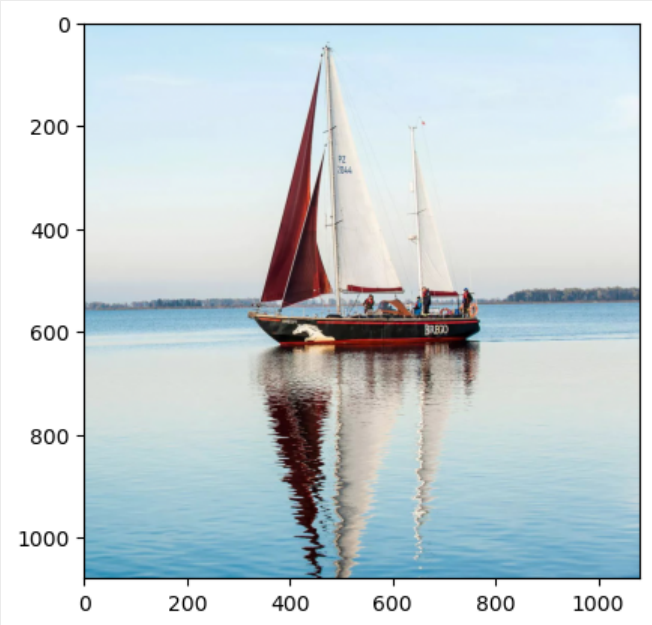
Laboratorium ma na celu zaimplementowanie uproszczonego mechanizmu wykrywania krawędzi metodą Canny'ego.

Technologia

Używać będę frameworku PyTorch i zgodnie z poleceniem będę starał się wykorzystać operację konwolucji tam, gdzie tylko to możliwe.

Wybrany obraz

Wykrywanie krawędzi wykonam na zdjęciu jachtu Brego, po części z sentymentu którym go darzę, a po części dlatego, że obraz zawiera zarówno wyraźne, proste krawędzie pod różnymi kątami, jak i nieco bardziej skomplikowane, rozmyte kształty na wodzie.



1. Brego

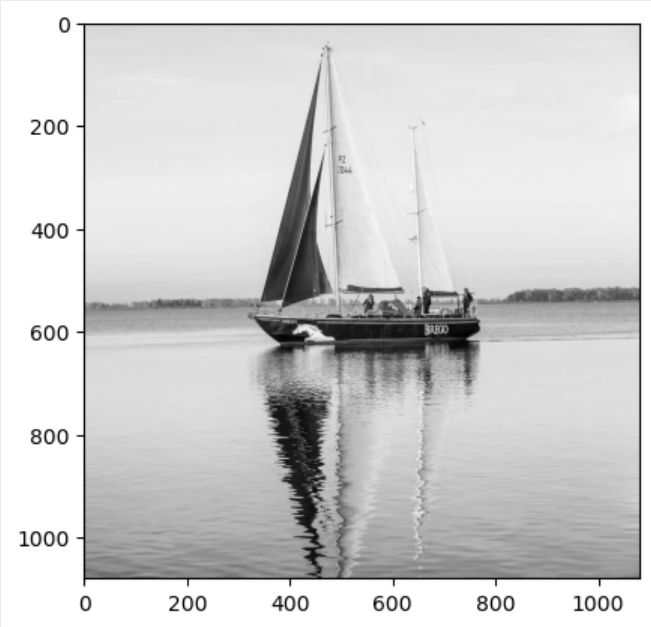
1. Wczytanie i konwersja wartości pikseli

```
img = Image.open('brego-full-crop.jpg')

# konwersja obrazu na tensor i normalizacja wartości do [0,1]
transform = transforms.Compose([
    transforms.ToTensor()
])
image_rgb = transform(img)
```

2. Konwersja na skalę szarości

Aby przekonwertować obraz na skalę szarości wykonałem iloczyn skalarny na każdym pikselu, tak aby powstał obraz czarno-biały. Wagi dla kolorów RGB wybrałem arbitralnie, tak aby obraz "dobrze wyglądał".

```
# wymiary kernela: (out_channels, in_channels, height, width)
greyscale_kernel = torch.tensor([[[[0.4]], [[0.3]], [[0.3]]]])  # Shape: (1, 3, 1, 1)

# robimy konwolucje na obrazie uzywajac powyzszego kernela
image_gs = F.conv2d(image_rgb, greyscale_kernel)

# usuwamy wymiar na batch size i juz niepotrzebny na 3 wartosci kolorow, zostaje 2d z wartoscia kazdego piksela
image_gs = image_gs.squeeze(0).squeeze(0)
```

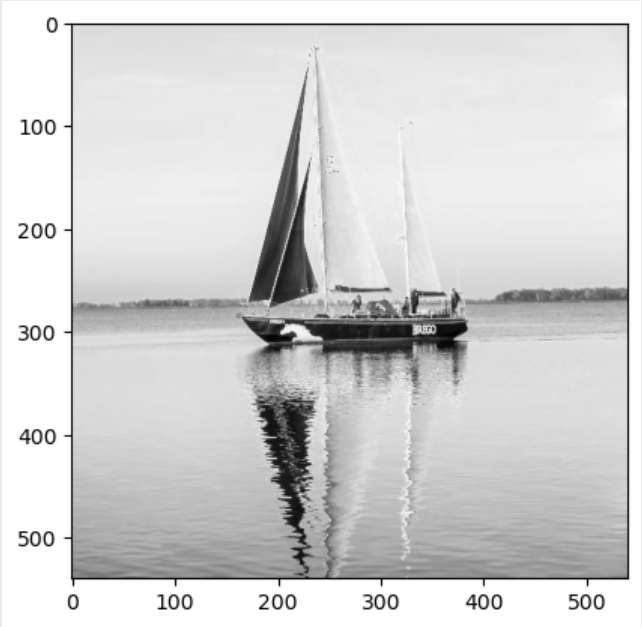


2. Brego - skala szarości

3. Pooling - redukcja rozmiaru

Wykonałem pooling w oknie mniejszym niż w poleceniu - obraz wyjściowy ma 1080 x 1080 pikseli, więc redukcję do 540 uznałem za wystarczającą. Wybrałem max pooling, ponieważ w przeciwieństwie do average poolingu uwydatnia on cechy w obrazie, a nie wygładza go, co bardziej pasuje do naszego zastosowania w wykrywaniu krawędzi.

```
pool = torch.nn.MaxPool2d(2)
image_small_gs = pool(image_gs.unsqueeze(0))
```

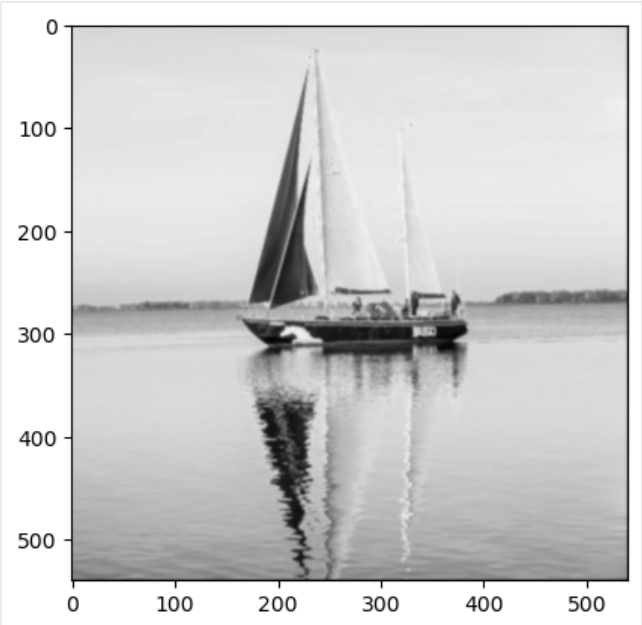


3. Obraz po poolingu

4. Rozmycie Gaussowskie

W celu usuniecia szumów i zbyt małych detali stosuję rozmycie gaussowskie. Użyłem kernela 5x5, stride=1 i padding=2.

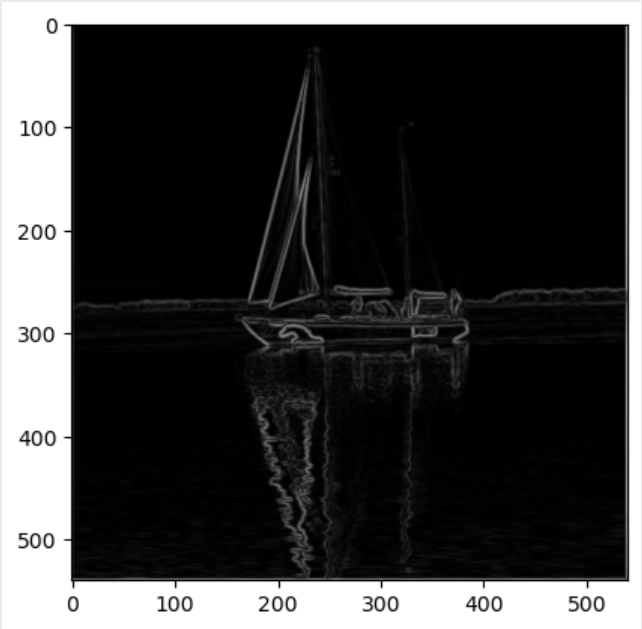
```
def gaussian_kernel(size, sigma=1):
    size = int(size) // 2
    x, y = np.mgrid[-size:size+1, -size:size+1]
    normal = 1 / (2.0 * np.pi * sigma**2)
    g = np.exp(-((x**2 + y**2) / (2.0*sigma**2))) * normal
    return g

gaussian_kernel_tensor = torch.tensor(gaussian_kernel(5), dtype=torch.float32)

image_small_gs = image_small_gs.squeeze(0)

image_blurred = F.conv2d(image_small_gs.unsqueeze(0).unsqueeze(0),
gaussian_kernel_tensor.unsqueeze(0).unsqueeze(0), stride=1, padding=2).squeeze().squeeze()
```



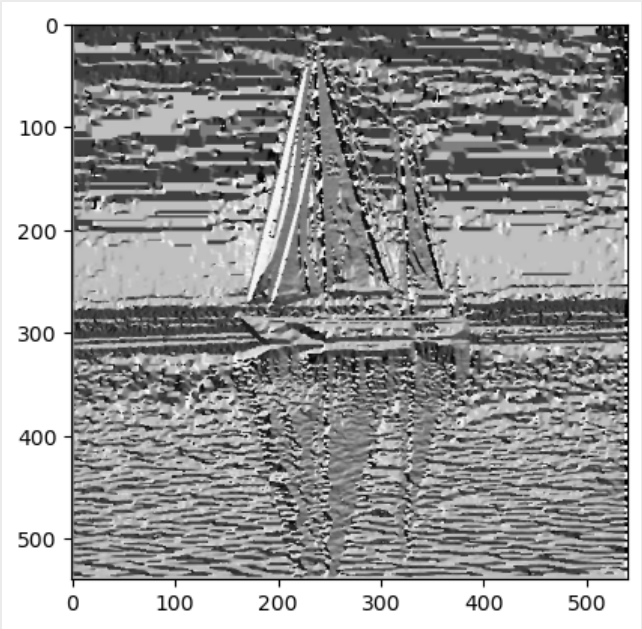
4. Obraz po rozmyciu

5. Obliczenie gradientów i intensywności zmiany pikseli

Korzystając z filtrów Sobela obliczyłem wartości gradientu w obu osiach, a następnie policzyłem intensywność zmiany pikseli obliczając pierwiastek sumy kwadratów wartości gradientów dla każdej osi dla każdego piksela. Policzyłem również kierunek gradientu dla każdego piksela.



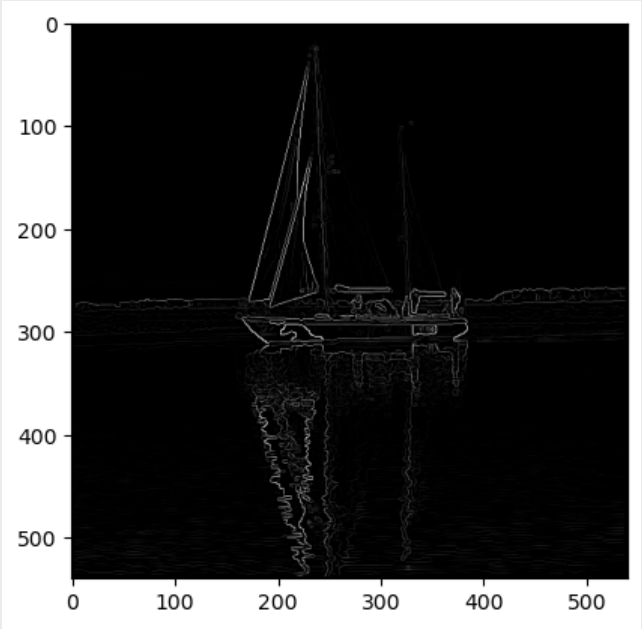
5. Intensywność zmiany pikseli



6. Kierunek gradientu

6. "Odchudzenie" krawędzi

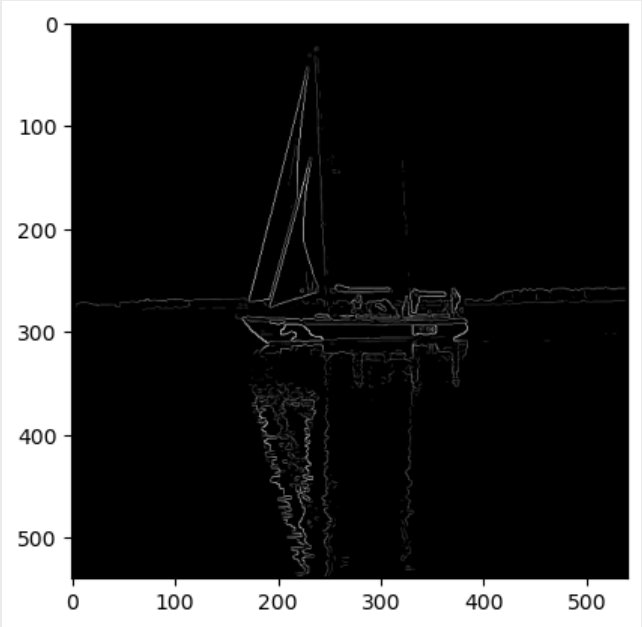
Korzystając z algorytmu non-max-suppresion z materiałów pomocniczych, odchudziłem krawędzie w obrazie.



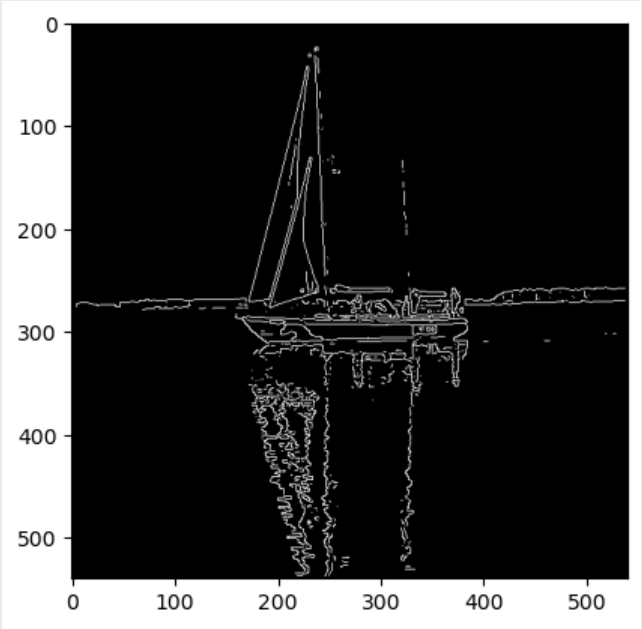
7. Obraz po odchudzeniu krawędzi

7. Odfiltrowanie słabych krawędzi i normalizacja pozostałych.

Nieco eksperymentując odfiltrowałem krawędzie o intensywności zmiany pikseli mniejszej niż 25, a następnie znormalizowałem obraz aby wartości pikseli należały do zbioru `{0, 1}` w zależności od tego czy na danym pikselu znajduje się krawędź czy nie.



8. Odfiltrowanie słabych krawędzi

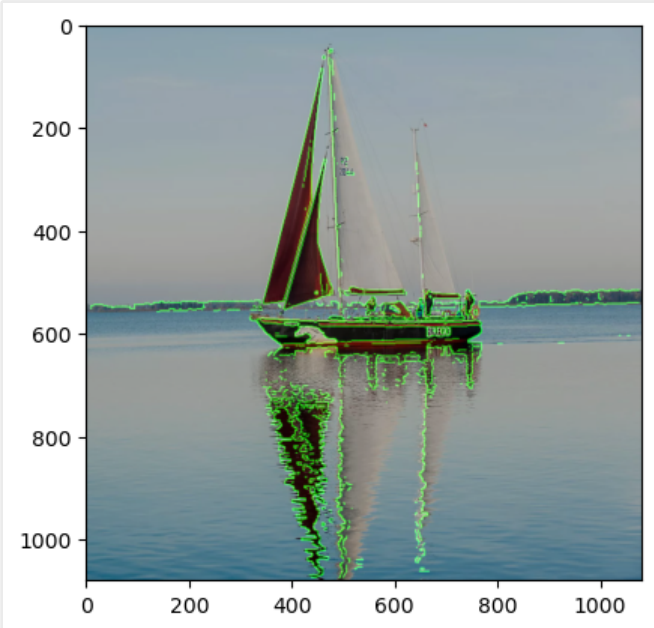


9. Normalizacja

W tym kroku nie użyliśmy podwójnego tresholdu i funkcji histerezy, które normalnie występują w metodzie Cannyego. Pomogłyby zidentyfikować krawędzie, które my uznaliśmy za słabe i nieważne, a które w rzeczywistości są krawędziami istotnymi. Sprawdzilibyśmy które z nich są połączone z krawędziami od początku uznanymi za istotne i również sklasyfikowalibyśmy jako takie.

9. Upscaling obrazu wykrytych krawędzi i połączenie z obrazem oryginalnym

Skorzystałem z funkcji interpolate dostarczanej przez torch.nn.functional aby wykonać upscaling. Następnie połączyłem oba obrazy wszystkie kanały a następnie wzmacniając zielony.



10. Oryginalny obraz z nałożonymi wykrytymi krawędziami