

MRO - zadanie 2: Kwadraty bez trójkątów (za to z wysokim stężeniem konwolucji)

Piotr Zawiślan

24 Październik 2022

1 Wykorzystane technologie.

Kod pisałem w języku Python z wykorzystaniem frameworka Pytorch.

2 Najpierw poszukajmy krawędzi...

W tej sekcji opiszę wyniki pierwszej części zadania (metoda Canny'ego).

2.1 Wczytanie i wyszarzenie obrazu



Rysunek 1: Zdjęcie wejściowe.

W celu zamiany 3 kanałów (*RGB*) w 1 kanał w skali szarości, zastosowałem na źródłowym obrazie o kształcie (3, 541, 555) konwolucję z filtrem o kształcie (1, 3, 1, 1) i wartościami odpowiadającymi wejściowym kanałom: 0.2, 0.6 i 0.6. Poniższy skrawek kodu przedstawia wykorzystane parametry w klasie inicjalizującej konwolucję *torch.nn.Conv2d*.

```
1 reduce_ch_conv = nn.Conv2d(in_channels=3, out_channels=1, kernel_size=1,
2                               padding='same', bias=False, padding_mode='replicate')
3 reduce_ch_conv.weight.data = reduce_ch_filter
4 furniture_grey = reduce_ch_conv(furniture_tensor)
```

Listing 1: Konwolucja wyszarzająca



Rysunek 2: Zdjęcie po zastosowaniu konwolucji opisanej w powyższym fragmencie kodu.

2.2 Pooling szarego obrazu

Aby zmniejszyć nieco rozdzielczość obrazu zastosowałem *max-pooling* z oknem o rozmiarze 4×4 . Wybrałem wersję *max* a nie *average* ze względu na to, że nie było większych różnic pomiędzy wynikami, a wersja *max* wydawała mi się sensowniejsza, w celu podkreślenia kontrastów między jasnymi i ciemnymi miejscami obrazu. Rozdzielczość otrzymanego obrazu wynosiła 135×138 .

```
1 max_pool_4 = nn.MaxPool2d(kernel_size=4, stride=4)
2 furniture_max_pooled = max_pool_4(furniture_grey)
```

Listing 2: Max-pooling 4×4



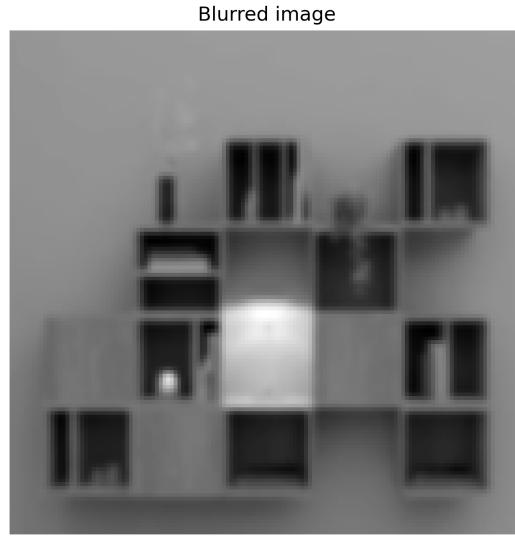
Rysunek 3: Zdjęcie po zastosowaniu poolingu z powyższego fragmentu kodu.

2.3 Rozmycie gaussowskie obrazu

Wybrałem gaussowski filtr o rozmiarze 5×5 .

```
1 kernel = get_gaussian_kernel(k_size=5).unsqueeze(0).unsqueeze(0)
2 blur_conv = nn.Conv2d(in_channels=1, out_channels=1, kernel_size=k_size, stride=1,
   padding='same', bias=False, padding_mode='replicate')
3 blur_conv.weight.data = kernel
4 furniture_blurred = blur_conv(furniture_max_pooled)
```

Listing 3: Rozmycie gaussowskie 5×5



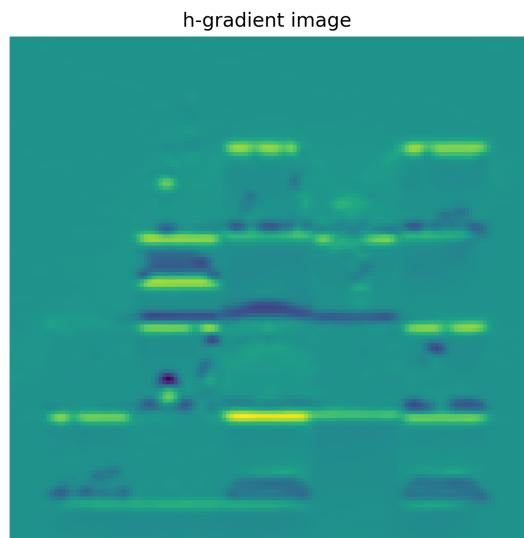
Rysunek 4: Zdjęcie po zastosowaniu rozmycia gaussowskiego z powyższego fragmentu kodu.

2.4 Uzyskanie gradientów obrazu

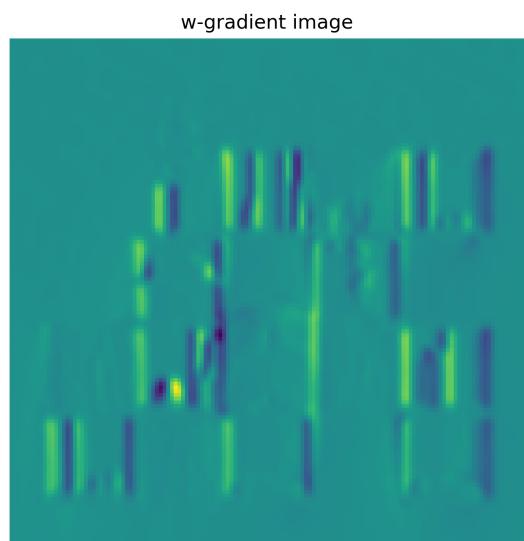
Aby uzyskać gradienty (w dwóch kierunkach obrazu - w i h) zastosowałem konwolucję z dwoma kernelami Sobela połączonym w jeden tensor o kształcie $(2, 1, 3, 3)$. Dwójka odpowiada tutaj dwóm wyjściowym kanałom.

```
1 gradient_conv = nn.Conv2d(in_channels=1, out_channels=2, kernel_size=3, padding='same'
   , bias=False, padding_mode='replicate', groups=1)
2 gradient_conv.weight.data = sobel_kernels_stacked
3 result = gradient_conv(furniture_blurred)
4 gradient_h = min_max_scale(result[0])
5 gradient_w = min_max_scale(result[1])
6 gradient_length = min_max_scale((gradient_h**2 + gradient_w**2).sqrt())
```

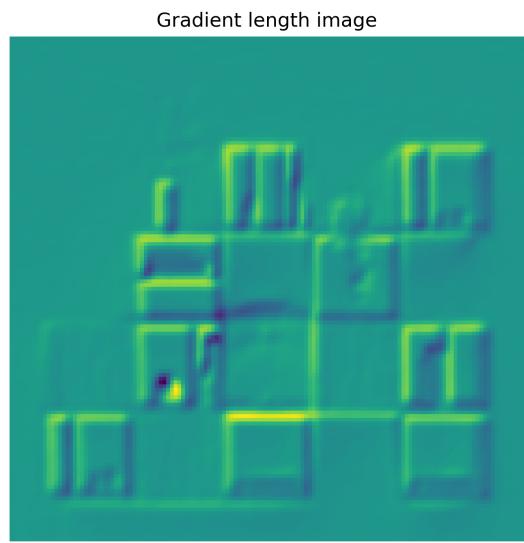
Listing 4: Konwolucja z filtrami Sobela



Rysunek 5: Gradient 'h' po zastosowaniu konwolucji z "pionowym"kernelem Sobela.

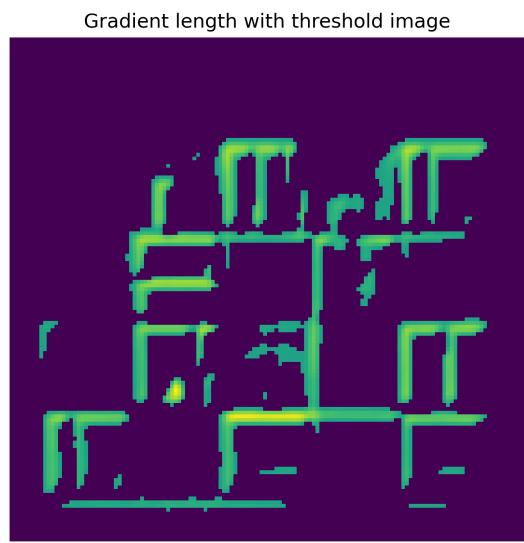


Rysunek 6: Gradient 'w' zastosowaniu konwolucji z "poziomym"kernelem Sobela.



Rysunek 7: Długość gradientów na zdjęciu.

Następnie odfiltrowałem regiony z gradientem poniżej 0.57.



Rysunek 8: Wykryte krawędzie na zdjęciu.

3 ...a potem sprawdźmy, czy układają się w kwadratowe kształty.

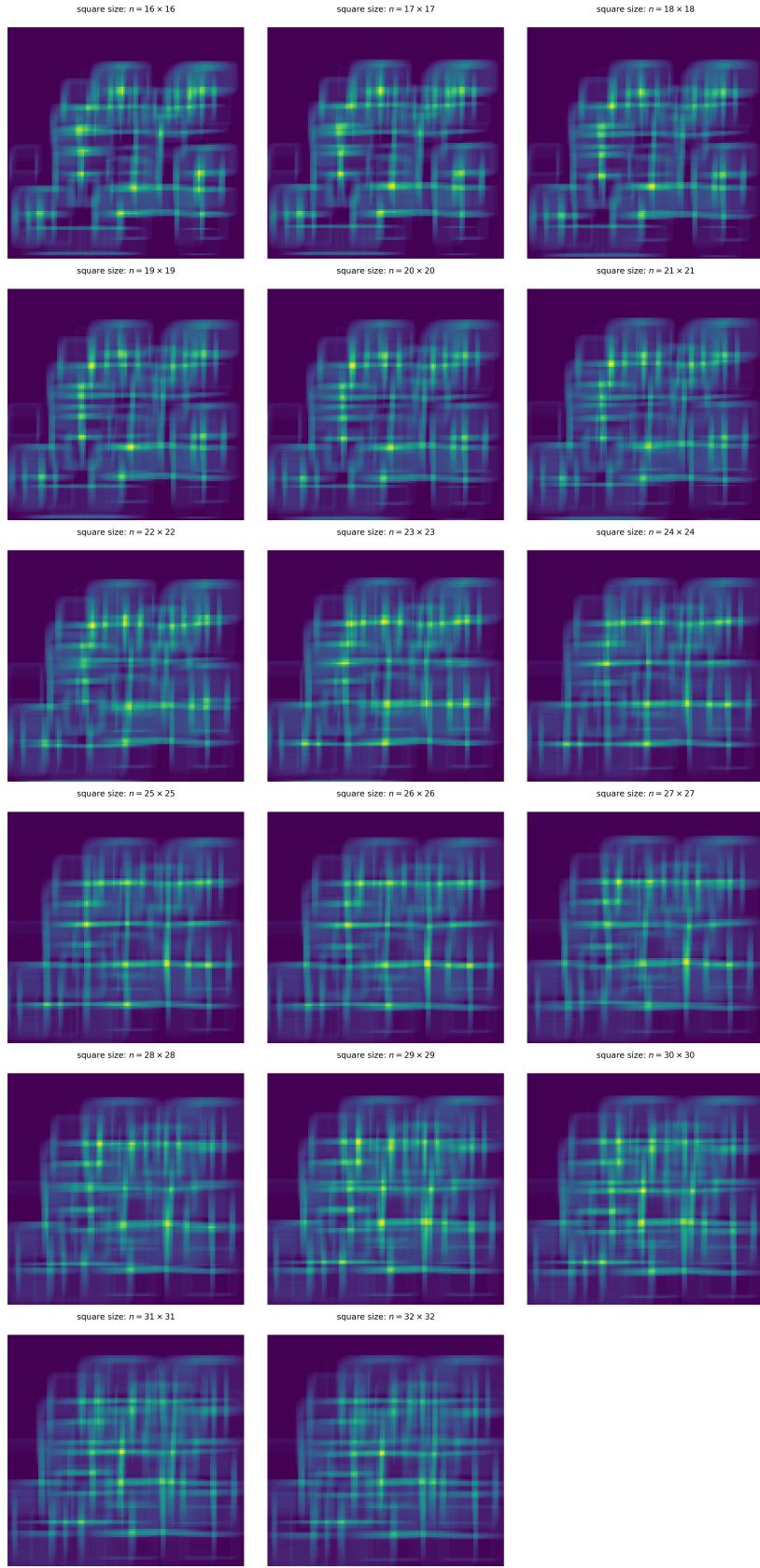
W tej sekcji opiszę wyniki drugiej części zadania (transformata Hougha).

3.1 Otrzymanie przestrzeni głosowania

Aby to zrobić wykorzystałem transponowaną konwolucję z użyciem filtra składającego się z siedemnastu kerneli o rozmiarach 32. Kernele te składają się z zer oraz jedynek ułożonych w kwadraty o rozmiarach od 16 do 32. Padding wybrałem taki, aby 'uciąć' niepotrzebne krańce przestrzeni głosowania: $\lfloor \frac{(Kernel.size-1)}{2} \rfloor$. Wynikiem takiej operacji jest siedemnaście kanałów - każdy opisujący przestrzeń głosowania dla kwadratów o długości od 16 do 32 pikseli.

```
1 def get_voting_spaces(image: torch.Tensor, a_min=16, a_max=32):
2     n_kernels = a_max - a_min + 1
3     kernels = get_hough_square_kernels(a_min, a_max).unsqueeze(0).double()
4     padding_size = (a_max - 1) // 2
5     hough_conv = nn.ConvTranspose2d(in_channels=1,
6                                     out_channels=n_kernels,
7                                     kernel_size=a_max,
8                                     stride=1,
9                                     padding=(padding_size, padding_size),
10                                    bias=False,
11                                    padding_mode='zeros')
12     hough_conv.weight.data = kernels
13     return hough_conv(image)
```

Listing 5: Funkcja zwracająca przestrzeń głosowania



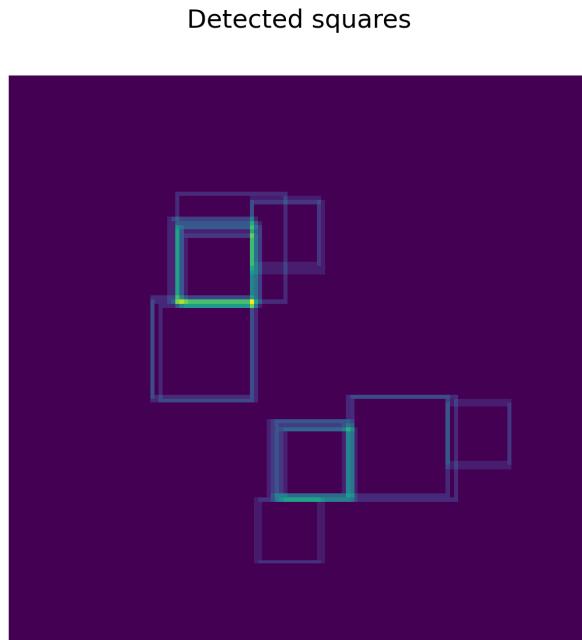
Rysunek 9: Otrzymane przestrzenie głosowania.

3.2 Wykryte kwadraty

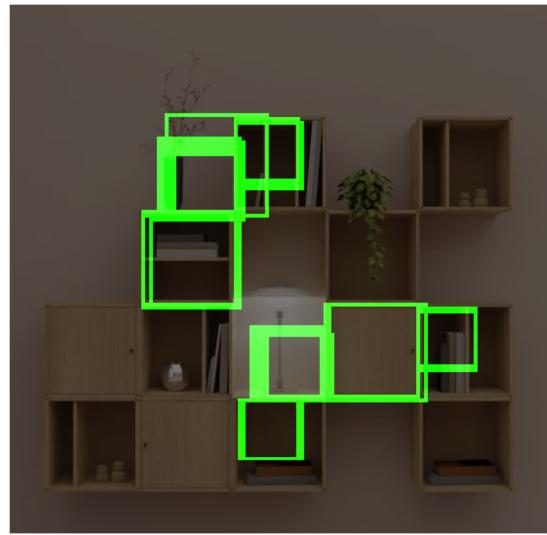
Następnie otrzymałem maksima poprzez zasosowanie progu (dopasowanego do rozmiaru kwadratu). Tę przestrzeń z maksimami (używając konwolucji transponowanej z tymi samymi filtrami co poprzednio) przekształciłem w obraz z wykrytymi kwadratami (odpowiednio przeskalowany do wejściowego obrazu).

```
1 def voting_spaces_to_square(v_spaces: torch.Tensor, a_min=16, a_max=32):
2     n_kernels = a_max - a_min + 1
3     kernels = get_hough_square_kernels(a_min, a_max).unsqueeze(1).double()
4     padding_size = (a_max - 1) // 2
5     hough_conv = nn.ConvTranspose2d(in_channels=n_kernels,
6                                     out_channels=1,
7                                     kernel_size=a_max,
8                                     stride=1,
9                                     padding=(padding_size, padding_size),
10                                    bias=False,
11                                    padding_mode='zeros')
12     hough_conv.weight.data = kernels
13     return hough_conv(v_spaces)
```

Listing 6: Funkcja przekształcająca przestrzeń z maksimami w wykryte kwadraty.

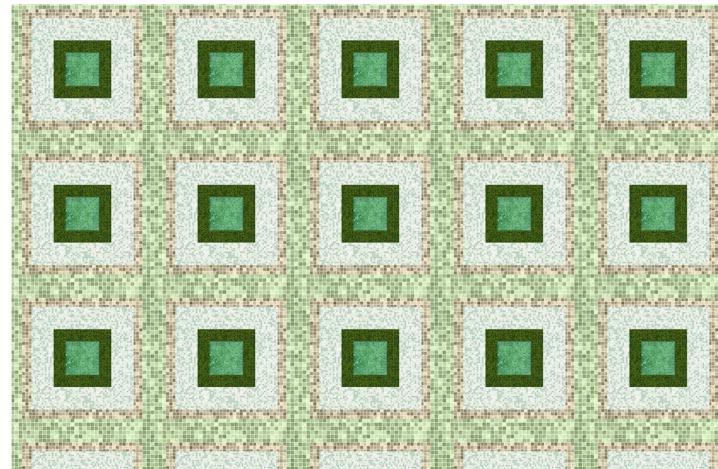


Rysunek 10: Otrzymane kwadraty.

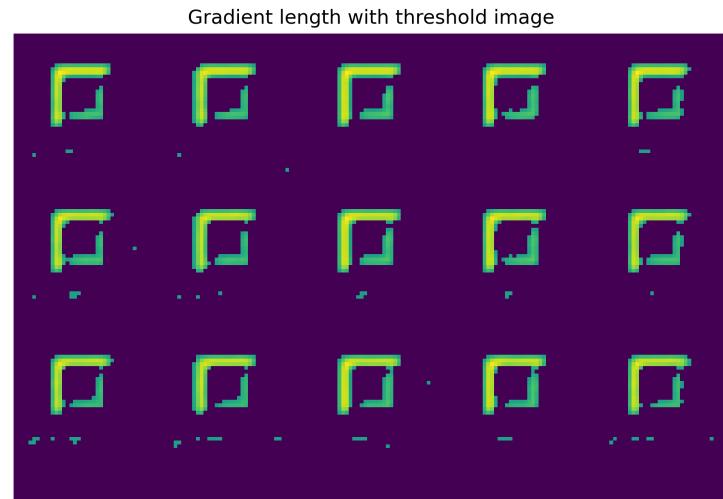


Rysunek 11: Wykryte kwadraty nałożone na obraz wejściowy.

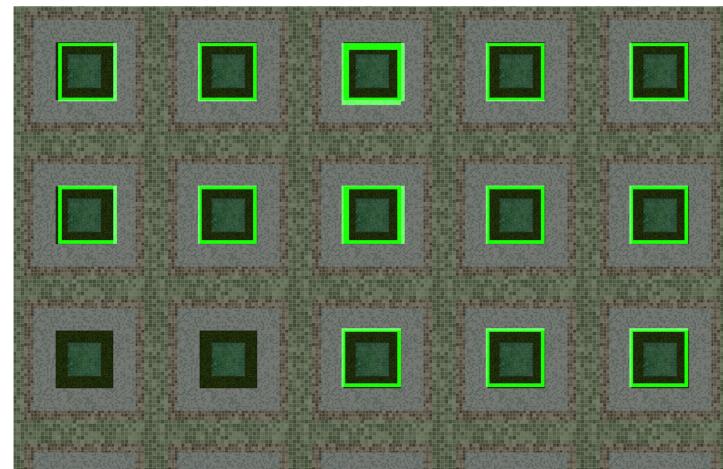
Położenie kwadratów nawet ma sens, ale nie jest to niesamowity wynik :). Poniżej załączam wyniki dla nieco bardziej "banalnego" obrazu.



Rysunek 12: Obraz wejściowy - mozaika z kwadratami (żeby było trochę łatwiej...).



Rysunek 13: Wykryte krawędzie (po zastosowaniu częściowej metody Canny'ego).



Rysunek 14: Wykryte kwadraty nałożone na mozaikę (po zastosowaniu transformaty Hougha).

Tutaj już jest lepiej :). Dwa kwadraty prawdopodobnie też zostały wykryte, gdybym trochę obiżył próg filtrowania maksimów.