Metody rozpoznawania obrazów
Zadanie 2 – raport

Mateusz Kocot

31 października 2021

Spis treści

1 Wyszukiwanie krawędzi

1.1 Zamiana RGB w skalę szarości

W celu zastosowania konwolucji wykorzystałem warstwę `Conv2D` z `tensorflow.keras`:

```
rgb_to_gray_conv = tf.keras.layers.Conv2D(1, (1, 1),
    strides=(1, 1),
    padding='valid',
    kernel_initializer=tf.keras.initializers.Constant((0.5, 0.3, 0.2)))
```

Kod ten tworzy obiekt klasy `Conv2D` z następującymi parametrami:

- 1 filtr,
- rozmiar filtra (kernela): `(1, 1)`,
- krok poziomy i pionowy (`strides`): `(1, 1)`
- `padding`: `valid` - czyli brak paddingu; w przypadku konwolucji `1x1` jest on niepotrzebny,
- wagi filtra (`kernel_initializer`): `(0.5, 0.3, 0.2)` - odpowiednio dla koloru czerwonego, zielonego i niebieskiego.

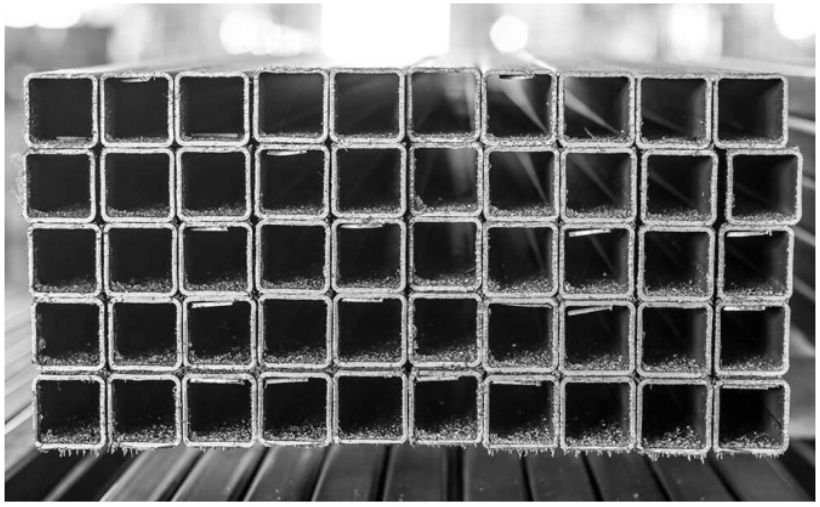
Zastosowanie takiego filtra jest proste, ale trzeba pamiętać o tym, że przetwarzanie wykonywane jest w batchach. W związku z tym, do tensora reprezentującego obraz należy dodać

jeszcze jeden wymiar:

```
gray_image = rgb_to_gray_conv(np.array([image]))[0]
```

Jako że po konwolucji możliwe jest otrzymanie wielu kanałów, na końcu pobieram pierwszy (jedyny - `[0]`).

W tym przypadku, zmiany wag RGB nie miały znaczącego wpływu na obraz w skali szarości. Na użyte wagi zdecydowałem się po kilku eksperymentach. Efekt końcowy widoczny jest na rys. 1.



Rys. 1: Wykorzystywany obraz w skali szarości

1.2 Rozmycie gaussowskie

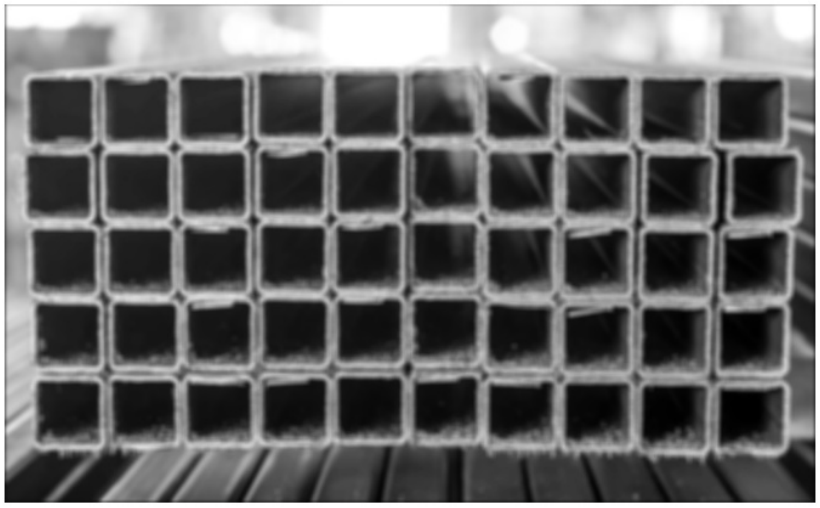
Przygotowałem funkcję `gaussian_kernel` tworzącą tablicę współczynników kernela gaussowskiego zgodnie z zadanymi argumentami `size` oraz `sigma`. Funkcję tę wykorzystałem przy tworzeniu obiektu `Conv2D`:

```
blur_conv = tf.keras.layers.Conv2D(1, (9, 9),
    strides=(1, 1),
    padding='same',
    kernel_initializer=tf.keras.initializers.Constant(
        gaussian_kernel(9, sigma=3.5)))
```

Po przeprowadzeniu kilku eksperymentów przyjąłem rozmiar filtra `9x9` oraz `sigma=3.5`. Wartości te są dość duże, a wyjściowy obraz dość mocno rozmyty, lecz poszukiwane kwadraty są wciąż dobrze widoczne, a wszelkie „niedoskonałości” (zwłaszcza w dolnej części obrazka) widocznie rozmazane (rys. 2).

Poza tym, wykorzystano drugi z dostępnych paddingów: `same`, który pozwala na zachowanie pierwotnego rozmiaru dzięki odpowiedniemu dopełnieniu obrazka zerami. W przypadku rozpatrywanego obrazka wiąże się to niestety z przekłamaniami na brzegach, gdyż jest on tam w

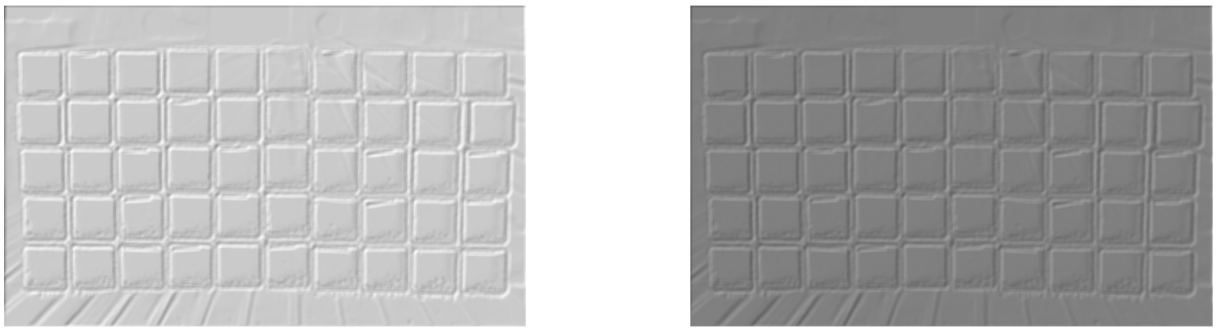
większości jasny. Skutkuje to relatywnie dużymi wartościami gradientu, nawet po rozmyciu. Problemem tym zajmuję się w kolejnej podsekcji.



Rys. 2: Obraz po rozmyciu

1.3 Gradient

Najpierw zastosowałem konwolucję `3x3` z filtrami Sobela. Uzyskałem w ten sposób dwa kanały (rys. 3).



Rys. 3: Kanały otrzymane po konwolucji z filtrami Sobela

Następnie policzyłem długości (w normie Euklidesa) wektorów otrzymanych przez połączenie otrzymanych przed chwilą dwóch kanałów:

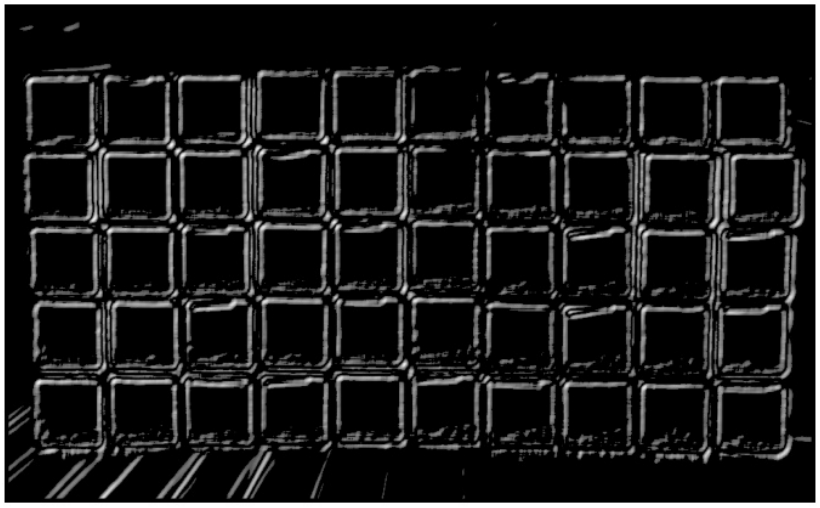
```
grad_length = np.sqrt(np.square(x_grad) + np.square(y_grad))
```

Ostatecznie odfiltrowałem regiony z niewielkim gradientem. Przed wykonaniem tego kroku przygotowałem aktualne dane do dalszej „obróbki".

- Wypełniłem zerami paski o szerokości 5 pikseli na każdej z krawędzi aktualnego obrazka. Dzięki temu problem długich gradientów na krawędziach został rozwiązany.

- Znormalizowałem dane do przedziału $[0, 1]$ poprzez podzielenie wszystkich wartości przez wartość aktualnie największą.

Po tych dwóch procedurach wykorzystałem funkcję ReLU z poziomem odcięcia 0.3. Wartość tą uznałem za dobry kompromis między jakością krawędzi a obecnością szumu. Efekt końcowy widoczny jest na rys. 4.



Rys. 4: Efekt końcowy niepełnego algorytmu Canny'ego

2 Wyszukiwanie kwadratów

2.1 Max pooling

Zmniejszyłem rozdzielczość obrazu wykorzystując max pooling o rozmiarze `4x4`. Rozmiar `2x2` powodował, że następne etapy trwały już stosunkowo długo.

Wykorzystałem klasę `MaxPooling2D`:

```
max_pool = tf.keras.layers.MaxPooling2D(
    pool_size=(4, 4), padding='same')
downscaled_edges = max_pool(tf.reshape(edges, [1, *edges.shape, 1]))[0]
```

Użycie `padding='same'` pozwoliło zachować rozdzielczość. Do macierzy `edges` reprezentującej aktualny stan obrazu musiałem dodać jeden wymiar „z przodu" (batch) oraz „z tyłu" (kanał).

2.2 Głosowanie

Wybrałem zakres rozmiarów szukanych kwadratów od `10x10` do `40x40`. Listę odpowiednich filtrów uzyskałem wywołując następującą funkcję z odpowiednimi wartościami parametru `size`:

```
def square_kernel(size, normalize=False):
    kernel = np.ones(shape=(size, size))
    if normalize:
        kernel /= size
    kernel[1:-1, 1:-1] = np.zeros(shape=(size-2, size-2))
    return kernel
```

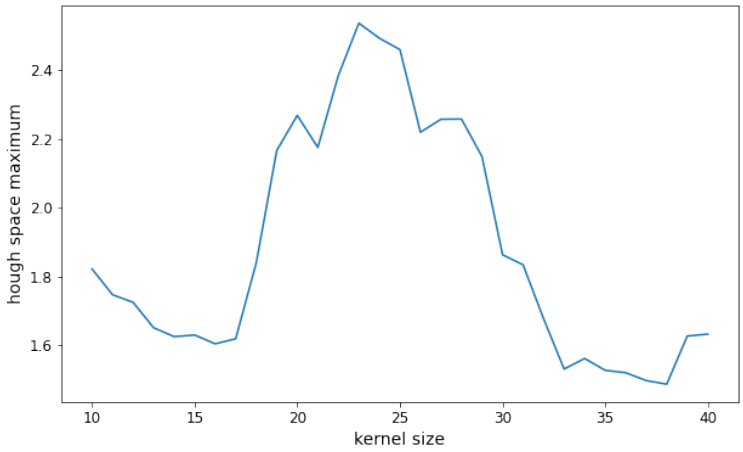
Warty wspomnienia jest parameter `normalize`. W trakcie eksperymentowania okryłem prostą zależność - im większy rozmiar filtra, tym większe maksimum kanału wynikowego dla tego rozmiaru. Żeby w skuteczny sposób wykrywać kwadraty spośród kanałów odpowiadających wszystkim użytym rozmiarom filtra, potrzebna była jakaś normalizacja wartości. Po kilku eksperymentach doszedłem do wniosku, że wystarczy wszystkie wartości kanałów podzielić przez odpowiadający im rozmiar, a to jest równoważne wykonaniu dzielenia już na etapie tworzenia kernela. Takie jest właśnie zastosowanie parametru `normalize`. Po ustawieniu go na `True`, wartości filtra dzielone są przez `size`.

Mając listę filtrów, przestrzeń wynikową Hougha dla danego elementu `kernel` uzyskałem następująco:

```
transpose_conv = tf.keras.layers.Conv2DTranspose(1, kernel.shape,
    strides=(1, 1),
    padding='same',
    kernel_initializer=tf.keras.initializers.Constant(kernel))
current_hough_space = transpose_conv(np.array([downscaled_edges]))[0]
```

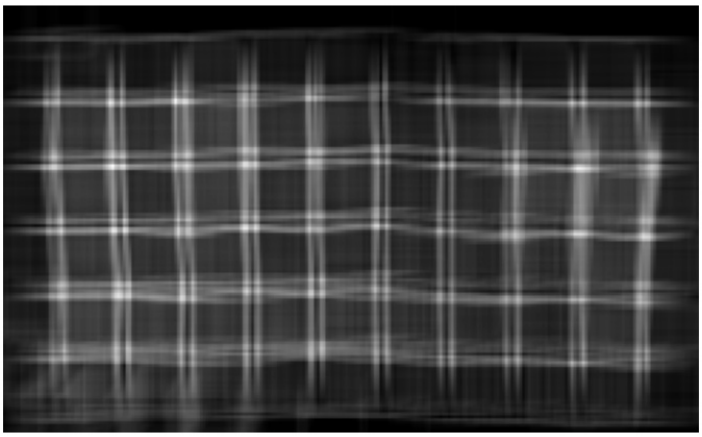
Użycie `padding=same` zapobiegło głosowaniu na piksele poza obrazem.

Po wypełnieniu listy przestrzeni Hougha sprawdziłem, jaki rozmiar z rozpatrywanych jest najlepiej rokujący. W tym celu stworzyłem wykres pokazujący wartość maksymalną przestrzeni dla danego rozmiaru (rys. 5).



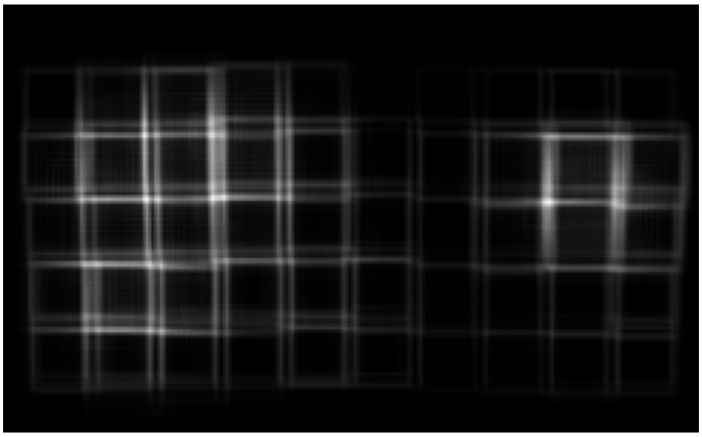
Rys. 5: Wykres maksimum przestrzeni wynikowej Hougha od rozmiaru zastosowanego filtra

Maksimum lokalne widoczne jest dla `size=23`, co oznacza, że w przestrzeni powstałej przy użyciu filtra o takim rozmiarze znajduje się punkt, który spośród wszystkich przestrzeni ma największe prawdopodobieństwo bycia środkiem prawdziwego kwadratu. Obserwacja ta jest jest zgodna ze zmniejszonym obrazem – taki w przybliżeniu jest rozmiar kwadratów „bazowych", tj. tych niezawierających innych kwadratów w środku. Przestrzeń Hougha dla filtra o rozmiarze 23 przedstawiłem na rys. 6. Na obrazku tym dobrze widać punkty, które rzeczywiście są w przybliżeniu środkami kwadratów z obrazka.



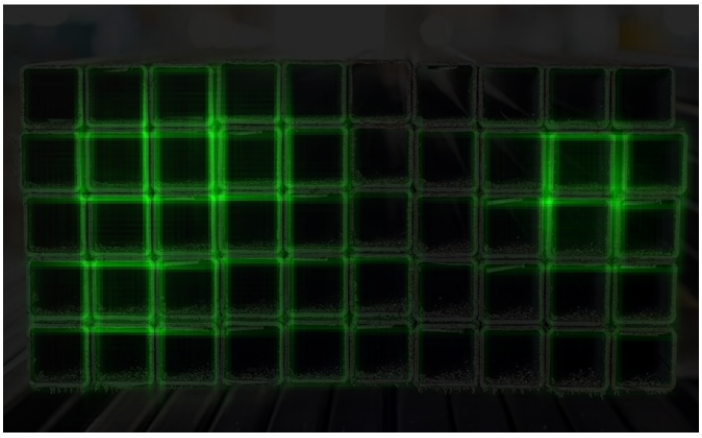
Rys. 6: Przestrzeń wynikowa Hougha dla filtru o rozmiarze 23

Mimo poprzedniej obserwacji, do dalszej części „dopuściłem" także pozostałe przestrzenie. Filtrowanie wykonałem w odniesieniu do wartości maksymalnej wszystkich przestrzeni – `total_max`. Mimo zalecenia z instrukcji próg ustawiłem aż na `0.55 * total_max`. Odsiał on jednak tylko kwadraty nie mające sensu, a po ponownym zastosowaniu konwolucji transponowanej w celu transformacji środków kwadratów we właściwe kwadraty, zaznaczone zostały (nie zawsze tylko raz) niemal wszystkie kwadraty z obrazka (rys. 7).



Rys. 7: Znalezione przez transformatę Hougha kwadraty

Na koniec nałożyłem wyniki na zielony kanał pierwotnego obrazu, wcześniej zmniejszając wartości w tym oraz pozostałych kanałach (rys. 8).



Rys. 8: Znalezione przez transformatę Hougha kwadraty nałożone (na zielono) na pierwotny obraz

Ze względu na lepiej wykryte krawędzie, część kwadratów została znaleziona „lepiej" (intensywniej) niż pozostałe. Prawdopodobnie, gdybym wykonał również dalsze, pominięte kroki metody Canny'ego, wyniki byłyby jeszcze bardziej satysfakcjonujące.

Podobną procedurę przeprowadziłem dla obrazka przedstawiającego szachownicę. Wynik widoczny jest na rys. 9.

Rys. 9: Wynik procedury przeprowadzonej na nowym obrazku

Ponownie zostały wykryte niemal wszystkie kwadraty. W oczy rzucają się czarne pola na rogach – przez długi gradient były one najbardziej popularne. Ogólnie wyniki są bardziej czyste. Wynika to oczywiście z prostszego obrazka, który w przeciwieństwie do poprzedniego nie zawiera praktycznie żadnego szumu (poza pionkami) zaburzającego wykrywanie krawędzi.