









# Zadanie 01

## Życie na krawędzi


W ramach tego zadania zaimplementujemy (uproszczony) mechanizm wykrywania krawędzi metodą Canny'ego (po więcej informacji na jej temat sięgnij do materiałów dodatkowych). Nie jest to jednak naszym jedynym celem. Ważne jest też to, by przy okazji odświeżyć sobie to, jak (w swoim ulubionym frameworku - np. Keras Core z dowolnym backendem, TensorFlow, PyTorch, JAX, etc.) implementuje się różne rodzaje konwolucji oraz dowiedzieć się jak obserwować efekty ich stosowania. W przeciwnym wypadku użylibyśmy po prostu gotowej implementacji bibliotecznej. ;]

- W przypadku punktów oznaczonych ikoną  poinformuj w jaki sposób je zrealizowano - wspomnij kluczowe klasy/metody/funkcje lub załącz powiązany fragment kodu źródłowego.
- W przypadku punktów oznaczonych ikoną  załącz w raporcie obraz przedstawiający stan kanału będącego wynikiem danej operacji.


## Co więc trzeba zrobić?



1. Zaczynamy od wczytania wybranego przez siebie obrazu. Nie ma potrzeby poddawania go na starcie intensywnej obróbce - upewnij się tylko, że można go użyć jako wejścia do zdefiniowanego dalej potoku przetwarzania. Warto jedynie przeskalować wszystkie wartości w pikselach tak, by mieściły się w zakresie od 0 do 1 (zwykle wystarczy podzielić je przez 255).
2. Obraz jest kolorowy i składa się z trzech kanałów - czerwonego, zielonego i niebieskiego. Dla uproszczenia późniejszych kroków zamieńmy go na wariant korzystający ze skali szarości - czyli składający się tylko z jednego kanału.
  1. W tym celu zastosuj konwolucję  $1 \times 1$  z jednym kanałem wyjściowym. Przypisz odpowiednie wagi do wykorzystywanego filtra tak, by wynik był średnią ważoną wartości RGB dla każdego piksela (proporcje poszczególnych kolorów możesz dobrać "na oko" oceniając po prostu jakość wyniku). Dopilnuj by użyty był właściwy *padding* i *stride* - chcemy, by wynikowy obraz był tego samego rozmiaru co wejściowy. [ 
3. Obraz jest nieco za duży. Chociaż nie jest to praktyczne podejście do tego problemu, postaramy się zmniejszyć jego rozmiar z użyciem tzw. *poolingu* - operacji, która zastępuje każde okno o zadanym kształcie jednym nowym pikselem. W tym przypadku wystarczy nam okno  $4 \times 4$ . Pozostaje jeszcze pytanie o to, jakiego typu *pooling* wybrać - *average* czy *max*? Zdecyduj się na jeden z nich (i uzasadnij swój wybór). [ 
4. Obraz jest już mały i czarno-biały, teraz pora pozbyć się niepotrzebnych szumów i detali. W tym celu zastosujemy rozmycie gaussowskie.
  1. Zaczynaj od przygotowania kodu, który dla zadanego rozmiaru filtra  $n$  przygotowuje tablicę  $n \times n$  zawierającą zgodne z rozkładem normalnym współczynniki - największe na środku, stopniowo malejące im bliżej krawędzi. Suma całej tablicy powinna oczywiście wynosić 1.
  2. Następnie na zwróconym w poprzednim kroku obrazie zastosuj konwolucję  $n \times n$  wykorzystującą przygotowaną chwilę wcześniej macierz wag. Wyjściowy obraz powinien być zgodny z oryginałem co do rozmiaru - lecz estetycznie rozmyty. [ 
  3. Poeksperymentuj z różnymi wartościami  $n$  - dobierz taki rozmiar filtra rozmywającego, który usunie drobne elementy, ale zachowa ogólny sens tego co na obrazie.

5. Kolejny etap to ustalenie, w których regionach obrazu gradient intensywności jest największy.


1. By poznać poziomą i pionową składową gradientu przetwórz (korzystając z konwolucji) aktualny stan obrazu przez wspomniane na zajęciach filtry Sobela. Zauważ, że tym razem na wyjściu uzyskujemy dwa kanały! 

```
[1, 2, 1   [1, 0, -1
0, 0, 0   2, 0, -2
-1, -2, -1] 1, 0, -1]
```

2. Scal uzyskane kanały w jeden, który reprezentuje intensywność gradientu (czyli długość odpowiedniego wektora - obliczaną w zgodnie z normą Euklidesa). W tym celu kolejno. 

1. Podnieś wszystkie wartości w obu warstwach do kwadratu.
  2. Zsumuj je - możesz użyć konwolucji 1x1 lub dowolnej innej wygodnej metody.
  3. Wyciągnij pierwiastek z sumy.
  3. Analogicznie - stwórz kanał w którym przechowamy informację o kierunku gradientu - potrzebny będzie arcus tangens z ilorazu składowych. 
6. Teraz zabierzemy się za "odchudzenie" wykrytych krok wcześniej krawędzi. Interesują nas tylko punkty o lokalnie największej intensywności (gdyby traktować intensywność gradientu jak wysokość nad poziomem morza, to w tym kroku szukalibyśmy odpowiednika grani). 

1. W tym celu należy porównać wartość danego piksela z dwoma pikselami sąsiednimi.
  1. Dany piksel zawiera potencjalną krawędź tylko jeżeli intensywność gradientu w tym punkcie jest większa niż dla dwóch pikseli sąsiednich.
  2. O których dwóch sąsiadach mowa? To z kolei zależy od kierunku gradientu (pomoc tu może również wizualizacja z materiałów pomocniczych).
2. Ten krok jest możliwy do zaimplementowania bez "opuszczania" wybranego frameworku - ale odpowiedzialna za niego warstwa i tak nie będzie różniczkowalna (ponieważ zawiera instrukcję typu if-else). W związku z tym, jeżeli będziesz miał trudności z jej implementacją, to po prostu skorzystaj z fragmentu kodu w "czystym" Pythonie znajdującego się w materiałach uzupełniających (sekcja o *non-max suppression*).

7. Krawędzie to miejsca gwałtownych przejść tonalnych - można je rozpoznać po bardzo dużych gradientach. Aby pozostawić na obrazie krawędzie, odfiltruj regiony z niewielkim gradientem. 

1. Można tu użyć np. tzw. progowanego ReLU (większość frameworków oferuje gotową implementację). Ta operacja sprowadza się w tym przypadku do "zachowaj wartość, jeżeli jest większa niż próg - w przeciwnym wypadku zastąp ją zerem".
  2. Poeksperymentuj z różnymi poziomami odcięcia i znajdź taki, dla którego znalezione krawędzie są tolerowalnej jakości.
  3. Na koniec znormalizuj wartości w kanale tak, by były albo 1 (krawędź) albo 0 (brak krawędzi).
  4. W tym kroku odbiegliśmy nieco od "wzorcowej" metody Cannyego. Jakich negatywnych konsekwencji możemy się spodziewać z tego powodu?
8. To koniec części odpowiedzialnej za wykrywanie krawędzi. Jeżeli znasz metodę Canny'ego to na pewno zauważyłeś, że pominięto niektóre z jej istotnych kroków. Jeżeli chcesz, to możesz przetestować drugą połowę zadania z wyższej jakości krawędziami wykrytymi przez gotową implementację tego algorytmu - i sprawdzić na ile poprawią się finalne rezultaty - ale jest to zupełnie opcjonalne!

9. Na koniec scal uzyskane wyniki z pierwotnym obrazem. Zwiększ ich rozdzielczość tak, by znów zgadzała się z oryginalną (korzystając z dowolnego zaimplementowanego już *upscaleingu*), a następnie nałóż na startowy obraz, np. wzmacniając kanał zielony i osłabiając pozostałe - wtedy odnalezione krawędzie będą widoczne na końcowej fotografii. Czy ich pozycje mają sens? [