Lab 1 – Canny (WERSJA BEZ ML’a)

Mateusz Mazur

# Wczytujemy obraz.

Do wczytania obrazu użyłem **cv2** i metody **imread,**  oraz przekonwertowałem go z BGR do RGB.

Obraz zawierający bydło, ssak, krowa, żywy inwentarz

Opis wygenerowany automatycznie

# Konwersja do skali szarości.

1. I = I\_o/255
2. I\_gray = np.zeros((I.shape[0], I.shape[1]))
3. for y in range(I.shape[0]):
4. for x in range(I.shape[1]):
5. # NTSC fromula
6. I\_gray[y][x] = 0.299 \* I[y][x][0] + 0.587 \* I[y][x][1] + 0.114\* I[y][x][2]

Do konwersji przyjąłem miarę z NTSC (0.299 – czerwony, 0.587 – zielony, 0.114 – niebieski), które mają dać najbardziej zbliżone do rzeczywistości odcienie szarości.

Obraz zawierający bydło, krowa, ssak, żywy inwentarz

Opis wygenerowany automatycznie

# 3. Pooling

Stworzyłem własna funkcję do poolingu i sprawdzam obydwa sposoby.

def pooling(I, size, \_type):

  if I.shape[0] % size != 0 or I.shape[1] % size != 0:

    print("Kernel size error!")

    return

  I\_pooled = np.zeros((int(I.shape[0]/size), int(I.shape[1]/size)))

  for y in range(I\_pooled.shape[0]):

    for x in range(I\_pooled.shape[1]):

      window = I[y\*4:y\*4+4,x\*4:x\*4+4]

      if \_type == "max":

        I\_pooled[y][x] = np.max(window)

      elif \_type == "avg":

        I\_pooled[y][x] = np.average(window)

  return I\_pooled

I\_avg = pooling(I,4,"avg")

I\_max = pooling(I,4,"max")

plt.imshow(I\_avg,"gray")

plt.show()

plt.imshow(I\_max,"gray")

plt.show()

Dla sposobu z maksem, zdjęcie wyszło zbyt prześwietlone (faworyzuje jasne piksele, czyli o większej wartości i np. zanikają krawędzie „wewnątrz” krowy), dlatego wybrałem sposób average. (wyniki poniżej)

Obraz zawierający bydło, żywy inwentarz, krowa, ssak

Opis wygenerowany automatycznieObraz zawierający bydło, ssak, żywy inwentarz, tekst

Opis wygenerowany automatycznie

# 4. Rozmywanie Gaussem

Funkcja do tworzenia kernela:

def fgaussian(size, sigma):

    h = size//2

    x, y = np.mgrid[-h:h+1, -h:h+1]

    g = np.exp(-(x\*\*2 + y\*\*2)/(2\*sigma\*\*2))

    return g / g.sum()

Funkcja do przeprowadzania konwolucji:

def convolution(I, kernel):

  I\_conv = np.zeros(I.shape)

  h = kernel.shape[0]//2

  I\_pad = np.pad(I, h)

  for y in range(h, I.shape[0]+h):

    for x in range(h, I.shape[1]+h):

       window = I\_pad[y-h:y+h+1, x-h:x+h+1]

       I\_conv[y-h][x-h] = np.sum(window\*kernel)

  return I\_conv

Przy rozmiarze kernela 3x3 i sigmie-1 dostaje moim zdaniem dobrze wyważone rozmycie

Obraz zawierający bydło, krowa, żywy inwentarz, zrzut ekranu

Opis wygenerowany automatycznie

# 5. Gradient

Obliczając gradient za pomocą filtrów Sobela dostajemy dwa kanały, poziomy i pionowy. Jak można zauważyć poniżej, w poziomym wariancie bardziej uwydatnione są krawędzie pionowe, ponieważ gradient działa „z lewej do prawej”. Drugi przypadek jest analogiczny.

Obraz zawierający zrzut ekranu

Opis wygenerowany automatycznie

Obraz zawierający zrzut ekranu

Opis wygenerowany automatycznie

Obliczanie intensywności:

G = np.sqrt(np.square(I\_y)+np.square(I\_x))

  G = G/ G.max() \* 255

Wynik na obrazie (po przeskalowaniu do zakresu [0,255]:  
Obraz zawierający szkic, sztuka

Opis wygenerowany automatycznie

Obliczanie kierunku gradientu:

  theta = np.arctan2(I\_y, I\_x)

# 6. Non Max Suppression

Przyznaję skorzystałem z kodu dostępnego w PDF’ie, a wynik wyszedł taki: Obraz zawierający szkic, tekst, rysowanie, sztuka

Opis wygenerowany automatycznie

Linie zostały dobrze „odchudzone”, a ilość „niepotrzebnych” wartości zminimalizowana.

# 7. ReLU

Jest to drugi raz, kiedy użyłem Kerasa i dostępnej tam warstwy ReLU. Sprawdziłem 3 progi, 60, 80 i 100, a później widząc wyniki wybrałem próg 70 jako najbardziej mi odpowiadający zachowany główny obwód krowy przy jednoczesnym usunięciu części niepotrzebnych linii).

**Obraz zawierający tekst, szkic, sztuka

Opis wygenerowany automatycznieObraz zawierający tekst, szkic, zrzut ekranu, rysowanie

Opis wygenerowany automatycznie**

**Obraz zawierający tekst, szkic, zrzut ekranu, sztuka

Opis wygenerowany automatycznieObraz zawierający tekst, szkic, rysowanie

Opis wygenerowany automatycznie**

**Użycie pojedynczego progu przynosi zazwyczaj gorsze efekty od podwójnego, którego używamy w oryginalnym Cannym. Podwójny treshold pozwoliłby zapewne na lepsze wypełnienie krawędzi (dzięki zastosowaniu pikseli „pewnych” i „możliwych”) oraz pozwoliłbym dokładniej usunąć te fragmenty obrazu, które nie należą do krawędzi, pomimo spełnienia założenia o wystarczającej intensywności.**

# Wyniki

Po binaryzacji powyższego obrazu, rozszerzeniu do wejściowych rozmiarów i nałożeniu na obraz wejściowy na zielonym kanale dostajemy ostateczny wynik, który szczerze, nie powala, ale źle też nie jest.

Obraz zawierający bydło, żywy inwentarz, tekst, krowa

Opis wygenerowany automatycznie