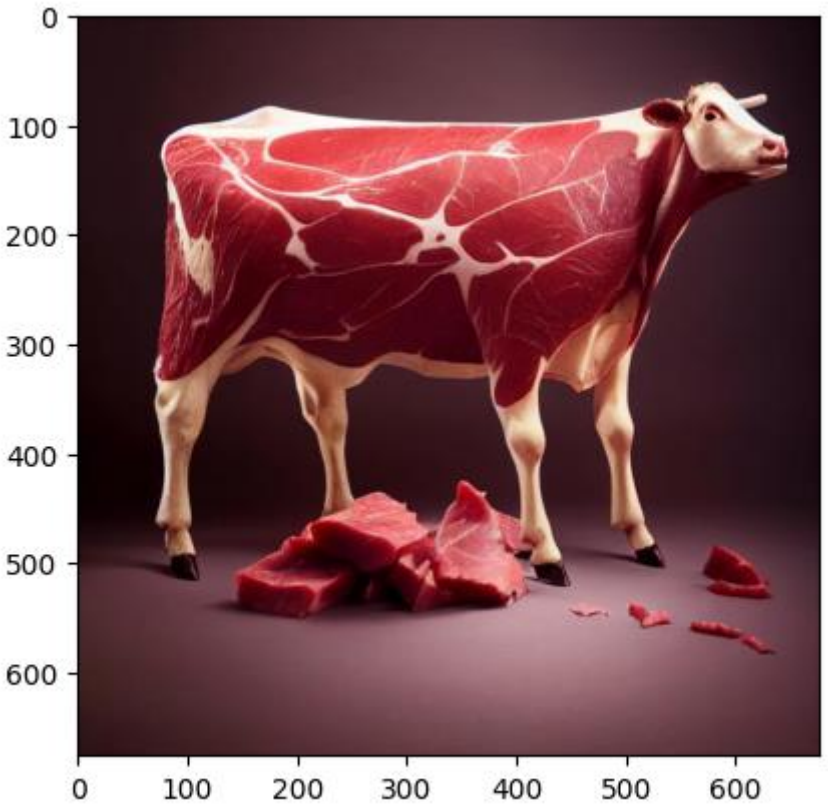
Lab 1 – Canny (WERSJA BEZ ML'a)

Mateusz Mazur

1. Wczytujemy obraz.

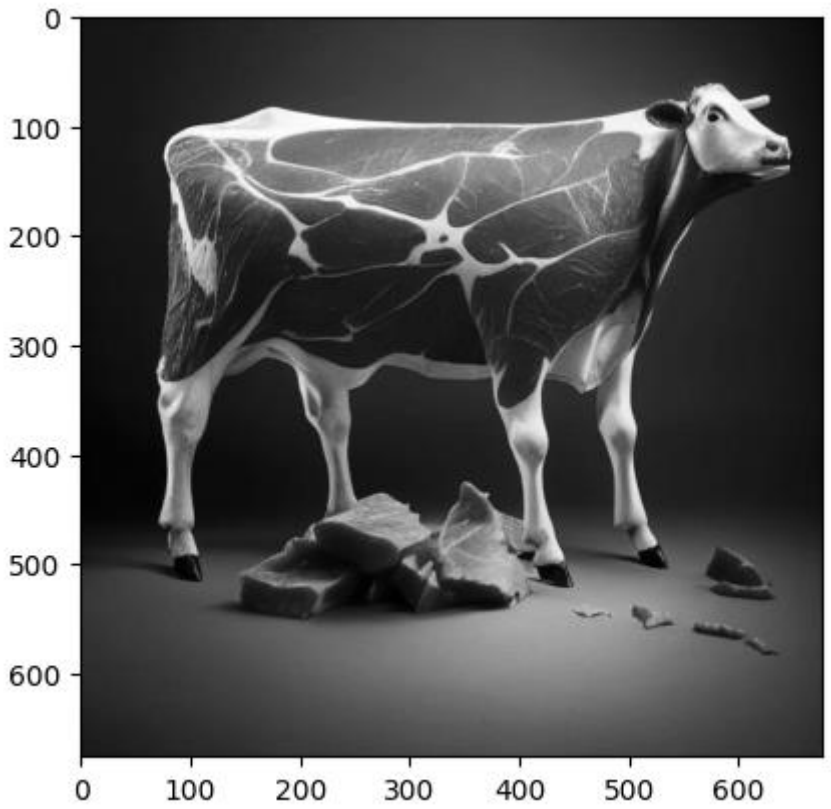
Do wczytania obrazu użyłem **cv2** i metody **imread,** oraz przekonwertowałem go z BGR do RGB.



2. Konwersja do skali szarości.

```
3. I = I_o/255
4. I_gray = np.zeros((I.shape[0], I.shape[1]))
5. for y in range(I.shape[0]):
6.   for x in range(I.shape[1]):
7.     # NTSC fromula
8.     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.

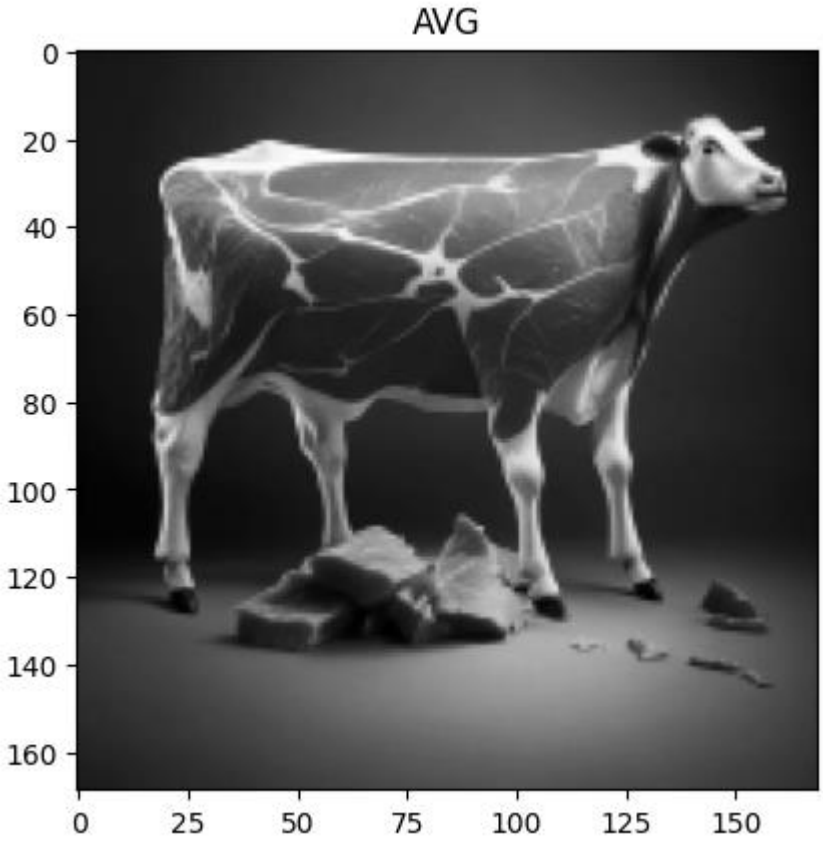


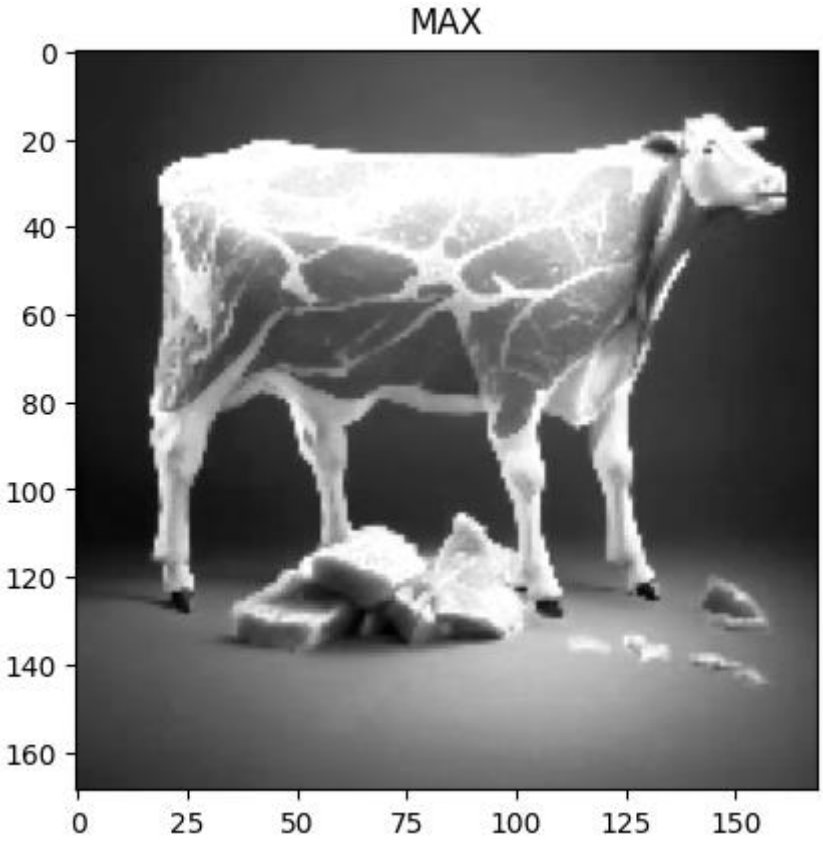
3. Pooling

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)


AVG
0
20
40
60
80
100
120
140
160
0
25
50
75
100
125
150


MAX
0
20
40
60
80
100
120
140
160
0
25
50
75
100
125
150

4. Rozmywanie Gaussem

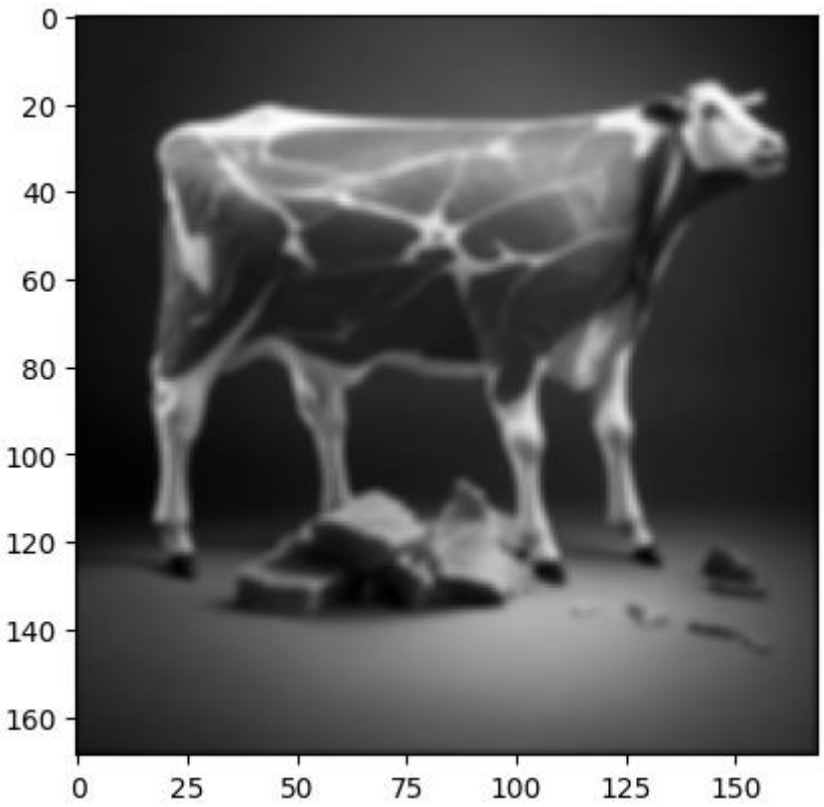
Funkcja do tworzenia kernela:

```
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:

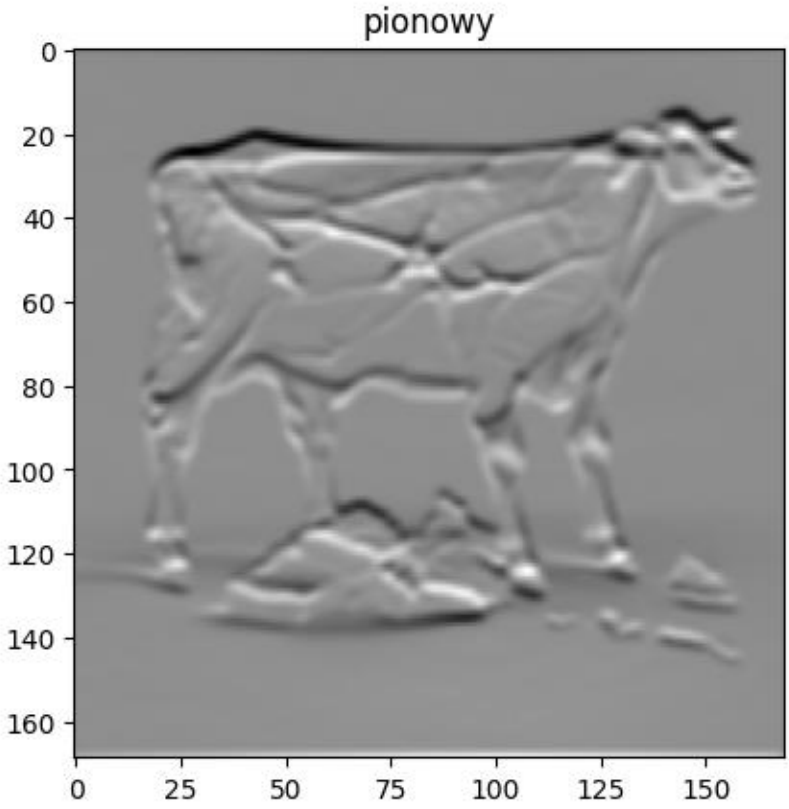
```
def convolution(I, kernel):

  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



5. Gradient

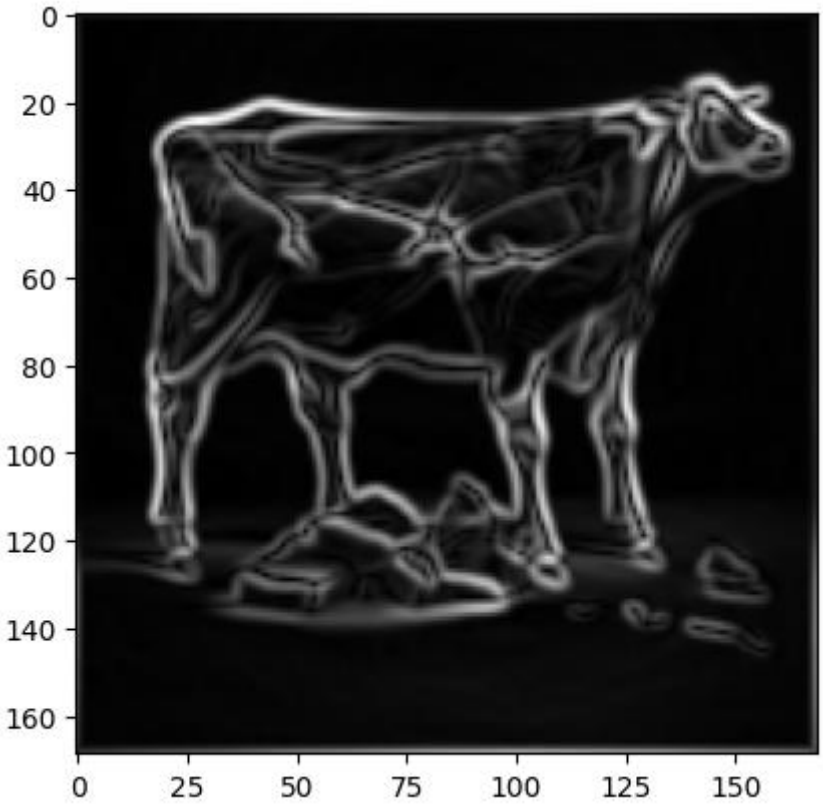
Obliczając gradient za pomocą filtrów Sobela dostajemy dwa kanały, poziomy i pionowy. Jak można zauważyć poniżej, w poziomym wariancie bardziej uwydatnione są krawędzie pionowe, ponieważ gradient działa „z lewej do prawej”. Drugi przypadek jest analogiczny.



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]:

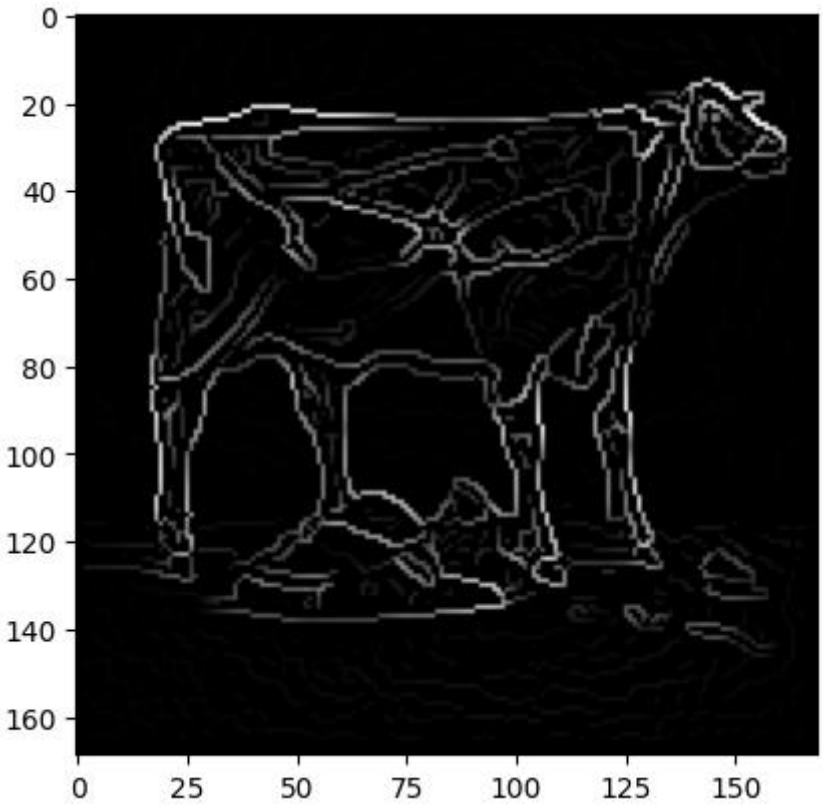


Obliczanie kierunku gradientu:

```
  theta = np.arctan2(I_y, I_x)
```

6. Non Max Suppression

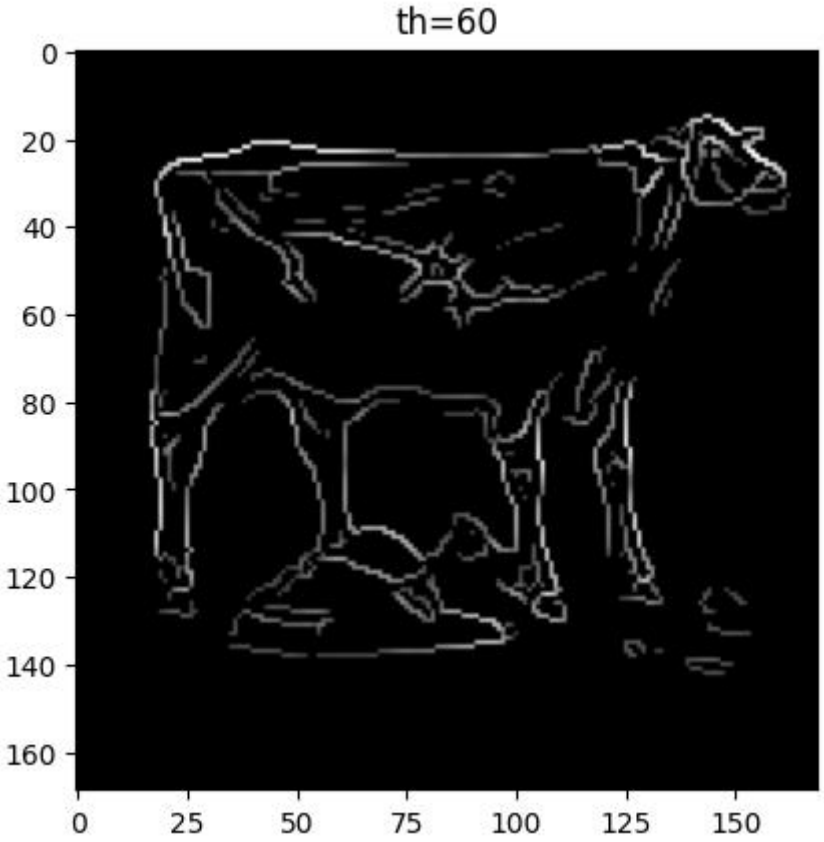
Przyznaję skorzystałem z kodu dostępnego w PDF'ie, a wynik wyszedł taki:

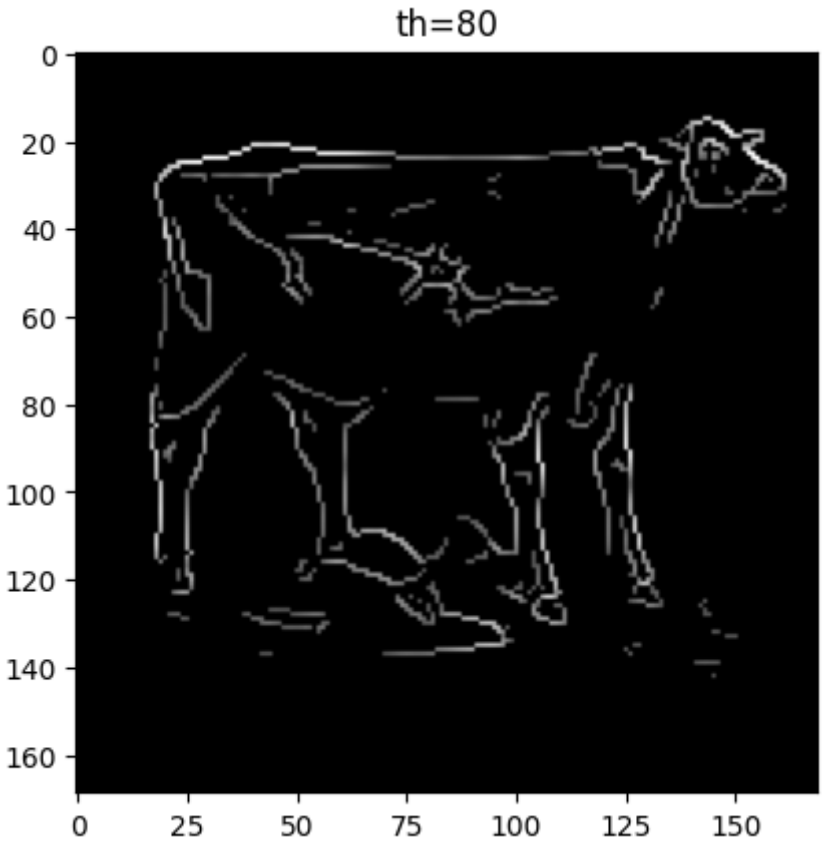


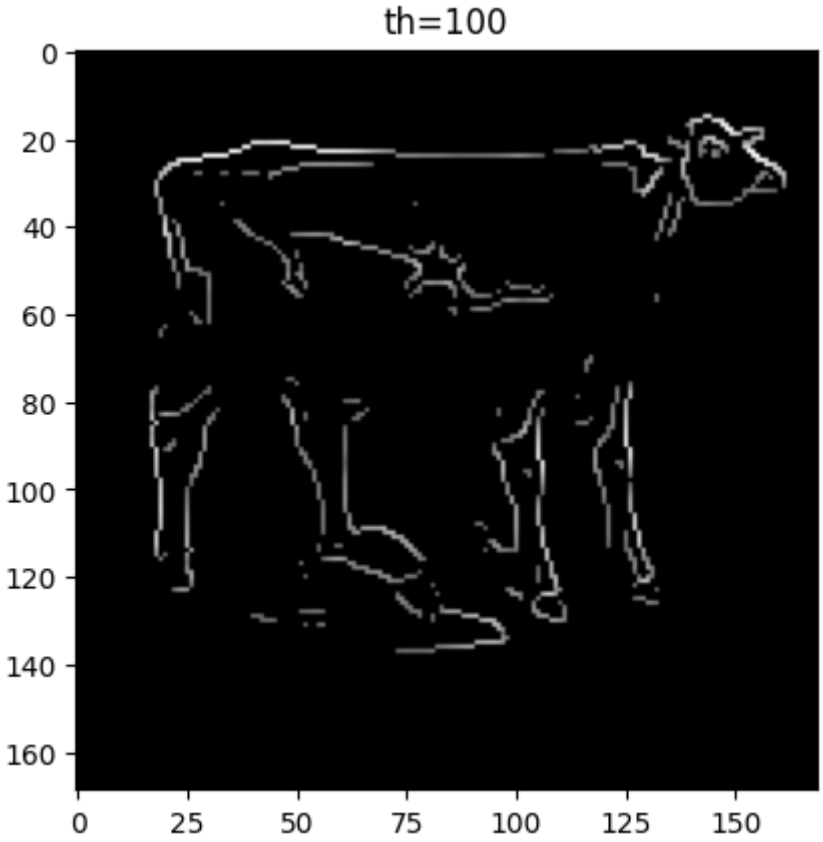
Linie zostały dobrze „odchudzone", a ilość „niepotrzebnych" wartości zminimalizowana.

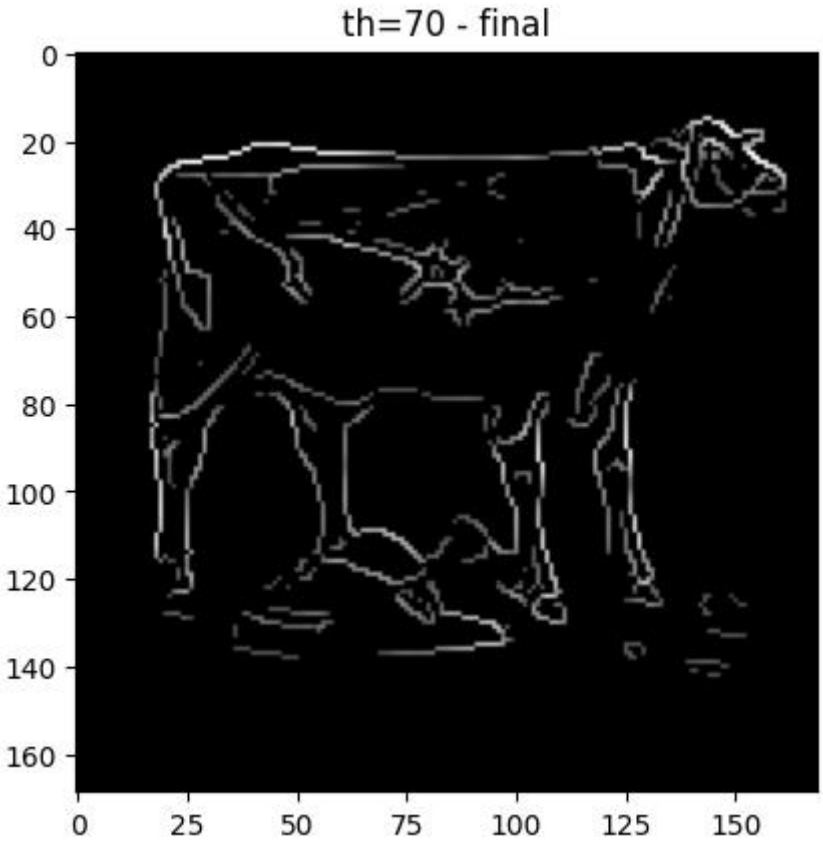
7. ReLU

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).









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.

9. 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.