Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчет по лабораторной работе № 3
«Обработка бинарных изображений. Морфология»

Курс: «Системы обработки изображений»

Студент: Грязнов Илья Евгеньевич
(фамилия, имя отчество)
Группы 6131-010402D

Самара 2022

Оглавление

Задание

Вариант№ 6

Варианты задания

№ варианта	Вероятность p	Вид структурного элемента
1	0,1	крест 5×5
2	0,15	квадрат 3×3
3	0,2	крест 3×3
4	0,25	квадрат 5×5
5	0,3	квадрат 3×3
6	0,35	крест 5×5
7	0,1	крест 3×3
8	0,2	квадрат 5×5
9	0,3	крест 3×3
10	0,25	крест 5×5

1. Создать средствами OpenCV бинарное изображение с простым рисунком.
   * Рисунок из простых фигур: квадрат, круг и т.д.
2. Зашумить изображение импульсным шумом с вероятностью p (по вариантам).
   * Вход: изображение из пункта 1
   * Вывод: зашумленное изображение
3. Написать функции реализации эрозии и дилатации

   Функции вида fun (Image, struct)

   Где struct - структурный элемент в виде матрицы

4. Выполнить операции эрозии и дилатации для зашумленного изображения со структурным элементом, заданным по вариантам.
   * Вход: изображение из пункта 2
   * Вывод: Исходное (из пункта 1), зашумленное (из пункта 2) и изображения после эрозии и дилатации.

5. Отфильтровать зашумленное изображение при помощи морфологических операций вскрытия и закрытия (структурный элемент задан по вариантам). Подсчитать коэффициент шума для результата фильтрации.

* Вход: изображение из пункта 2
* Вывод: исходное (из пункта 1), зашумленное (из пункта 2) и изображения после вскрытия и закрытия. Коэффициент шума.
* Коэффициент шума считать, как количество не совпавших пикселей между отфильтрованным изображением и исходным (не зашумленным) из пункта 1.

6. Отфильтровать изображение при помощи логического фильтра. Подсчитать коэффициент шума для результата фильтрации.

* Вход: изображение из пункта 2
* Вывод: исходное (из пункта 1), зашумленное (из пункта 2) и отфильтрованное изображение. Коэффициент шума.
* Коэффициент шума считать, как количество не совпавших пикселей между отфильтрованным изображением и исходным (не зашумленным) из пункта 1.
* Таблица логического фильтра в лекции слайд 17.

7. На исходном изображении с помощью морфологических операций выделить контур объекта. Выяснить, когда контур получается внешним, внутренним, четырёхсвязным, восьмисвязным.

* Вход: изображение из пункта 1
* Вывод: исходное изображение, изображение с выделенными контурами.

8. На исходном изображении с помощью морфологических операций выделить горизонтальные и вертикальные контуры объекта.

* Вход: изображение из пункта 1
* Вывод: исходное изображение, изображение с выделенными горизонтальными контурами, изображение с выделенными вертикальными контурами.

Ход выполнения работы:

In [2]:

```
p = 0.1 # Вероятность

# Крест 5x5 соглачно варианту
cross = np.array([
    [0, 0, 1, 0, 0],
    [0, 0, 1, 0, 0],
    [1, 1, 1, 1, 1],
    [0, 0, 1, 0, 0],
    [0, 0, 1, 0, 0],
])
```

In [3]:

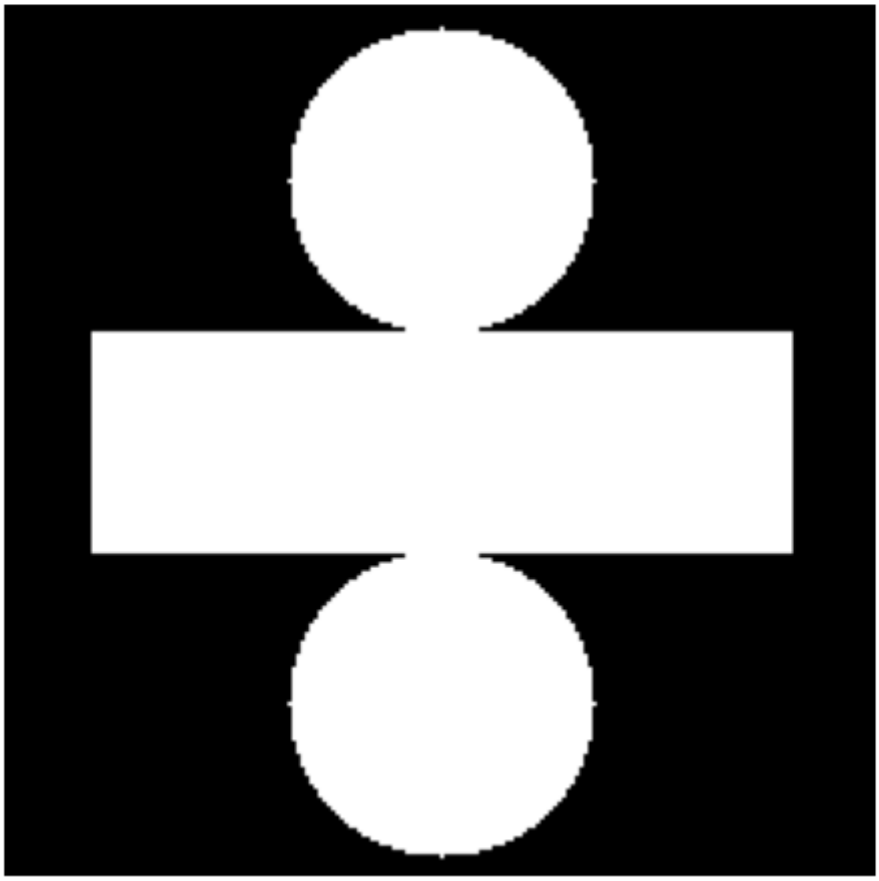
```
def show_me(image): # вывести изображение
    figure(figsize=(8, 8))
    plt.axis('off')
    plt.imshow(image, cmap='gray')
```

1. **Создать средствами OpenCV бинарное изображение с простым рисунком. (Рисунок из простых фигур: квадрат, круг и т.д.)**

Создал средствами OpenCV изображение:

In [4]:
```
img = np.zeros([200, 200], np.uint8)# Задали размерность

cv2.circle(img, (100, 40), 35, (255, 255, 255), -1)  # Круг
cv2.rectangle(img, (180, 75), (20, 125), (255, 255, 255), -1) # Прямоугольник
cv2.circle(img, (100, 160), 35, (255, 255, 255), -1) # Круг

show_me(img)
img.shape
```

Out[4]:
```
(200, 200)
```



Бинарное изображение

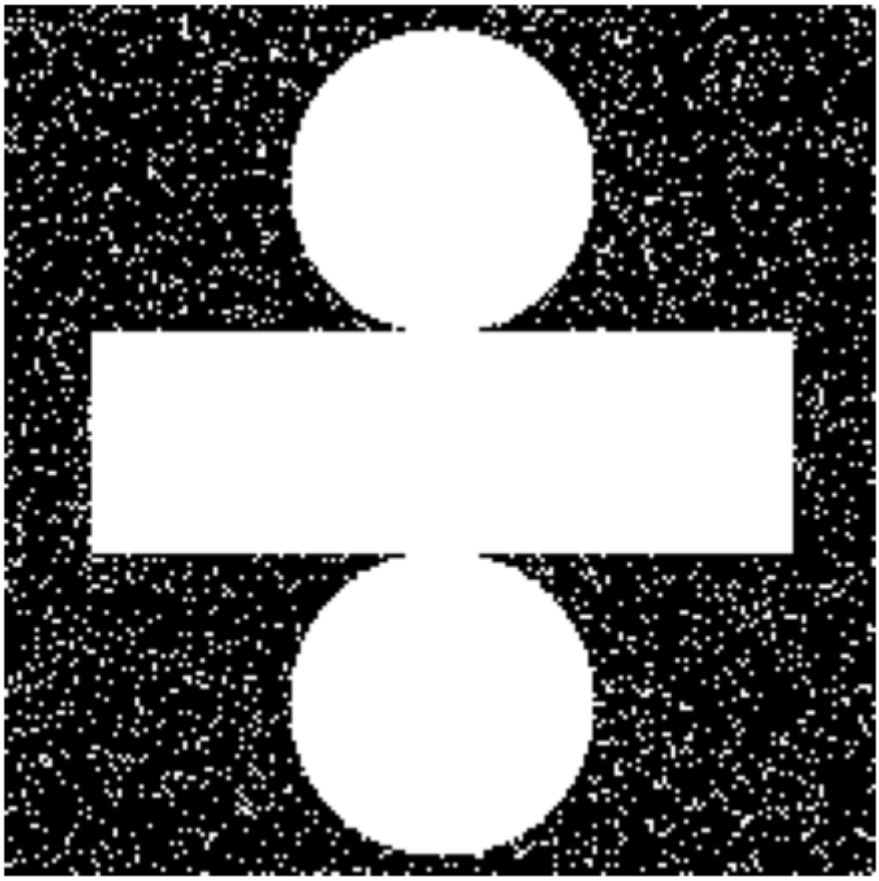
2. **Зашумить изображение импульсным шумом с вероятностью p (по вариантам. Вход: изображение из пункта 1. Вывод: зашумленное изображение).**

Произведено зашумление выше созданного изображения импульсным шумом с вероятностью 10%.

[n [5]:
```
def noise(image, p): # Функция для импульсного зашумления
    result = image.copy()
    r, col = image.shape
    for i in list(product(range(r), range(col))):
        if np.random.random() <= p:
            result[i[0], i[1]] = 255

    return result
```

[n [6]:
```
noise_image = noise(img,  p )
print('Вывод: зашумленное изображение:')
print('_____')
show_me(noise_image)
```

```
Вывод: зашумленное изображение:
_____
```



3. Написать функции реализации эрозии и дилатации (*Функции вида fun (Image, struct) Где struct - структурный элемент в виде матрицы).*

Далее приведены реализованные функции эрозии и дилатации, которые в дальнейшем будем использовать в работе.

[7]:
```
def give_me(window, kernel): # функция подсчета значений из скользящего окна по заданнному кресту
    vals = []
    k_r, k_c = kernel.shape

    for i, j in np.ndindex((k_r, k_c)):
        if kernel[i, j] == 1:
            vals.append(window[i, j])

    return vals
```

При выполнении операции эрозии структурный элемент "проходит" по всем пикселам входного изображения. Если в некоторой позиции каждый единичный пиксел структурного элемента совпадает с единичным пикселом бинарного изображения, то выполняется логическое «И» центрального пиксела структурирующего элемента с соответствующим пикселом выходного изображения. Т.о. текущему элементу изображения присваивается черный цвет только тогда, когда все значимые (черные) значения маски совпадают со значимыми (черными) значениями изображения.

В результате выполнения эрозии исходное изображение сжимается на эту же величину. На изображениях могут исчезнуть объекты, диаметр которых меньше диаметра структурирующего элемента.

[8]:

```python
def eros(img, kernel): # Функция эрозии
    img_r, img_c = img.shape
    kernel_rows, kernel_cols = kernel.shape


    # создаем пустой массив нулей размером больше изображения на + размер выходящего за пределы окна
    temp_img = np.zeros(
        (img_r + 2 * (kernel_rows // 2), img_c + 2 * (kernel_cols // 2)))


    # Копируем в него временно наше изображение
    temp_img[
        (kernel_rows // 2): img_r + 2 * (kernel_rows // 2) - (kernel_rows // 2),
        (kernel_cols // 2): img_c + 2 * (kernel_cols // 2) - (kernel_cols // 2)
    ] = img


    out = img.copy() #Создаем копию нашей картинки для выходного значения

    # Далее бежим по всем пикселям нашего изображения и заменяем на минимальное значение из "округи" согласно нашего ядра
    for i, j in tqdm(np.ndindex((img_r, img_c))):
        window = temp_img[i: i + kernel_rows, j: j + kernel_cols]
        out[i, j] = np.min(give_me(window, kernel)) # тут подставляем мин

    clear_output()
    return out
```

Дилатация — метод обратный эрозии. Структурирующий элемент также применяется ко всем пикселям бинарного изображения. Каждый раз, когда начало координат структурирующего элемента совмещается с единичным бинарным пикселом, ко всему структурирующему элементу применяется перенос и последующее логическое «ИЛИ» с соответствующими пикселями бинарного изображения. Т.о. когда хотя бы одно значимое (черное) значение маски совпадает со значимым (черным) значением изображения, то текущему элементу изображения присваивается черный цвет.

Дилатация расширяет структуру исходного изображения в зависимости от свертки структурирующего элемента с исходным изображением. В результате выполнения дилатации могут окраситься впадины в объектах и пустоты в изображениях, чей диаметр меньше структурного элемента.

[9]:

```python
def dilat(img, kernel): # Функция эрозии

    img_r, img_c = img.shape
    kernel_rows, kernel_cols = kernel.shape

    # создаем пустой массив нулей размером больше изображения на + размер выходящего за пределы окна
    temp_img = np.zeros(
        (img_r + 2 * (kernel_rows // 2), img_c + 2 * (kernel_cols // 2)))

    # Копируем в него временно наше изображение
    temp_img[
        (kernel_rows // 2): img_r + 2 * (kernel_rows // 2) - (kernel_rows // 2),
        (kernel_cols // 2): img_c + 2 * (kernel_cols // 2) - (kernel_cols // 2)
    ] = img


    out = img.copy() #Создаем копию нашей картинки для выходного значения

     # Далее бежим по всем пикселям нашего изображения и заменяем на Максимальное значение из "округи" согласно нашего ядра
    for i, j in tqdm(np.ndindex((img_r, img_c))):
        window = temp_img[i: i + kernel_rows, j: j + kernel_cols]
        out[i, j] = np.max(give_me(window, kernel)) # тут подставляем Макс

    clear_output()
    return out
```

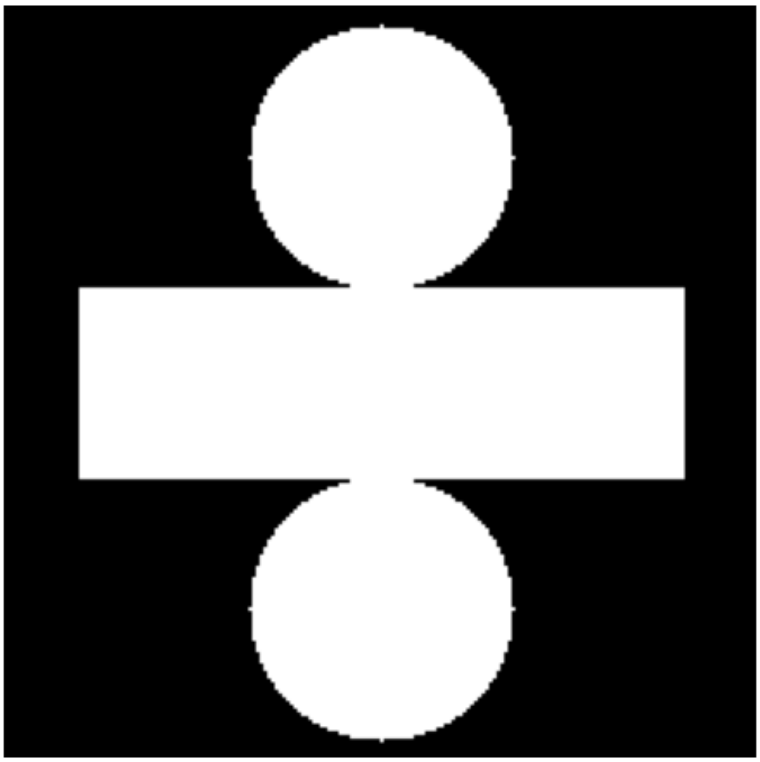
4. **Выполнить операции эрозии и дилатации для зашумленного изображения со структурным элементом, заданным по вариантам. (Вход: изображение из пункта 2. Вывод: Исходное (из пункта 1), зашумленное (из пункта 2) и изображения после эрозии и дилатации.).**

Применив структурный элемент согласно варианту, выполняем функцию эрозии и дилатации:

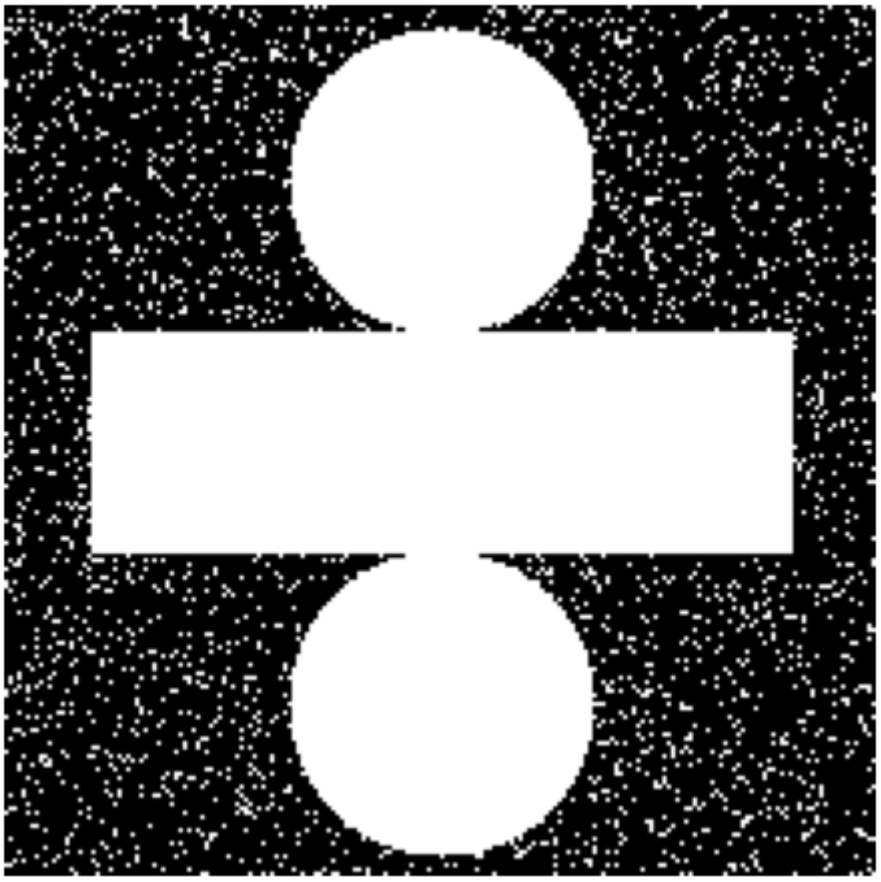
[10]:

```
# Вывод: Исходное (из пункта 1), зашумленное (из пункта 2) и изображения после эрозии и дилатации.
print('Исходное изображение и зашумленное изображение')
show_me(img)
show_me(noise_image)
```

```
Исходное изображение и зашумленное изображение
```



Исходное изображение

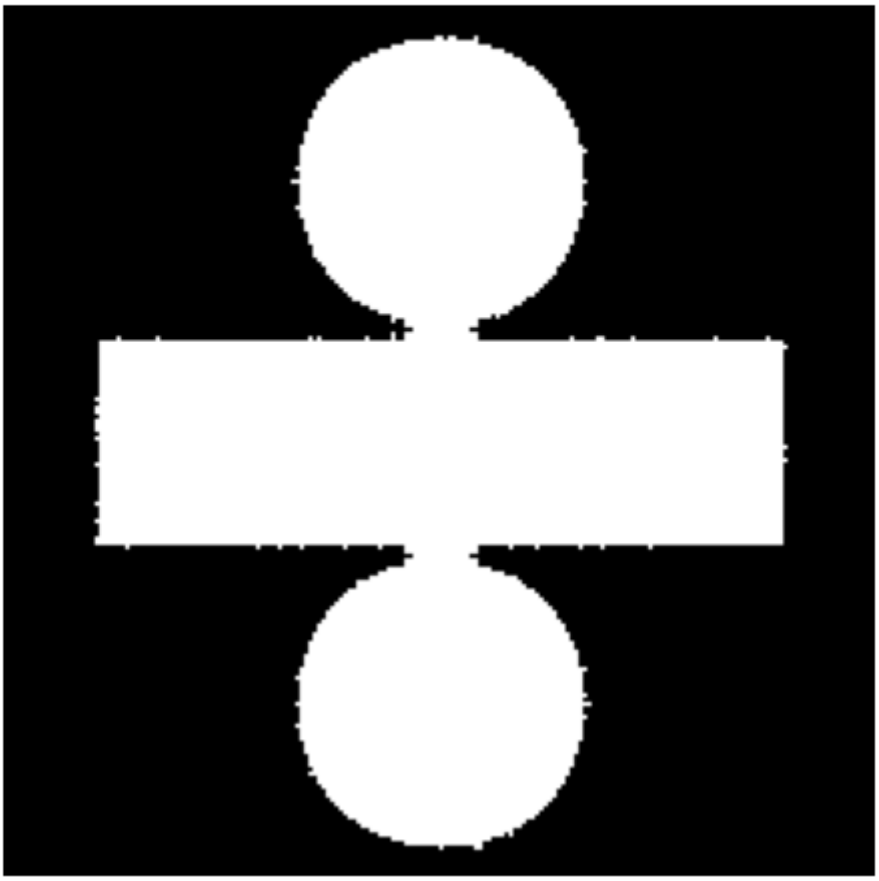


Зашумленное изображение

Применяем эрозию:

[11]:

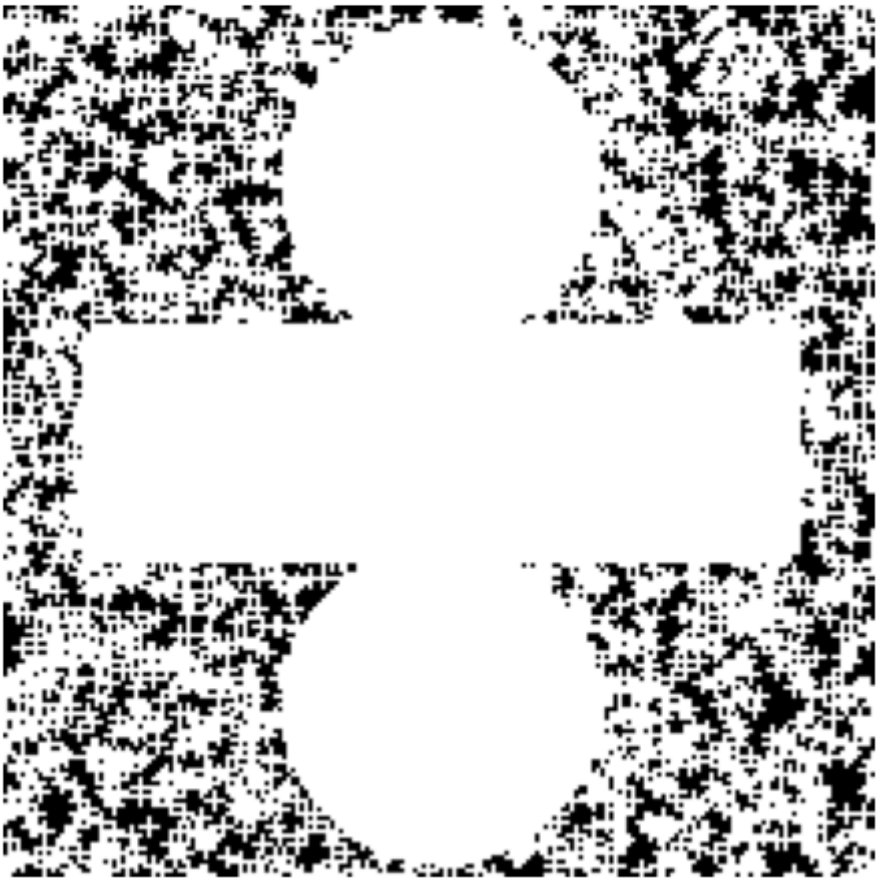
```
# Изображения после эрозии.
erosimg = eros(noise_image, cross)
show_me(erosimg)
```



Изображение после эрозии

Применяем дилатацию:

[12]:
```
# Изображения после дилатации.
dilatimg = dilat(noise_image, cross)
show_me(dilatimg)
```



Изображение после дилатации

5. **Отфильтровать зашумленное изображение при помощи морфологических операций вскрытия и закрытия (структурный элемент задан по вариантам). Подсчитать коэффициент шума для результата фильтрации. (Вход: изображение из пункта 2. Вывод: исходное (из пункта 1), зашумленное (из пункта 2) и изображения после вскрытия и закрытия. Коэффициент шума. Коэффициент шума считать, как количество не совпавших пикселей между отфильтрованным изображением и исходным (не зашумленным) из пункта 1.)**

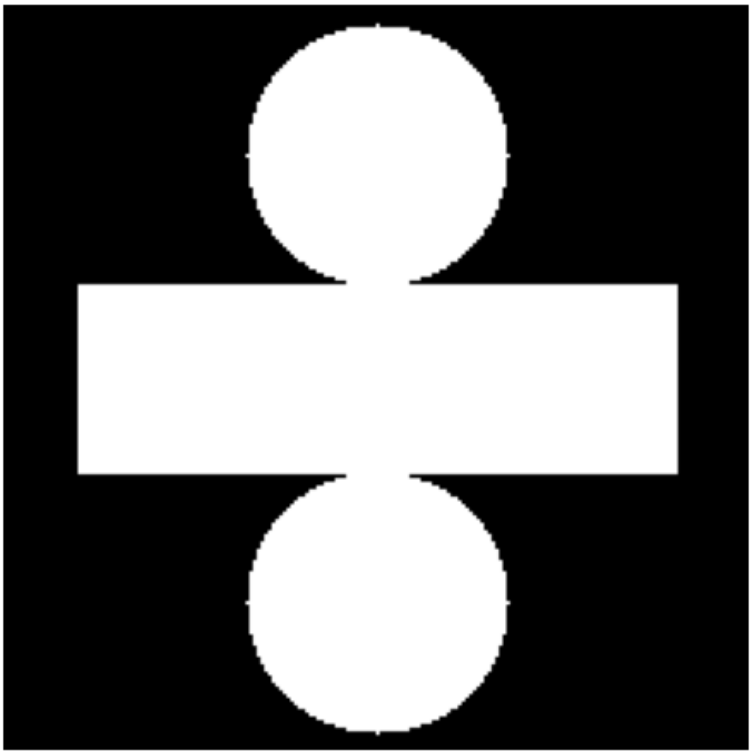
Проведем операции вскрытия и закрытия. Открытие (или размыкание) — это операция, которая включает в себя сначала вызов оператора эрозии, а после вызывается оператор дилатации над полученным изображение после оператора эрозии. Такая комбинация приведёт к тому, что одиночные пиксели и малые их скопления будут удалены, крупные объекты обрезаются по краям. Если внутри больших объектов были маленькие дырки, то и они станут больше. Последующая операция расширения восстановит обрезанные объекты до первоначальных размеров, но мелкого мусора, удаленного сужением уже не будет. Т.о. операция открытия сглаживает контуры объекта, обрывает перешейки и ликвидирует выступы небольшой ширины.
Закрытие (или замыкание) – это операция, которая включает в себя сначала вызов операции дилатации, а после вызывается операция эрозии над полученным изображением после операции дилатации. На практике операция закрытия сглаживает контуры объекта, в общем случае «заливает» узкие разрывы и длинные углубления малой ширины, а также ликвидирует небольшие отверстия и заполняет промежутки контура.

[13]:
```
# количество несовпавших пикселей между отфильтрованным изображением и исходным
```

[14]:
```
def opening(image, kernel): # Данная функция выполняет операцию вскрытия над исходным изображением
    return dilat( eros(image, kernel), kernel)
def closing(image, kernel): # Данная функция выполняет операцию закрытия над исходным изображением
    return eros(dilat(image, kernel), kernel)
def noise_factor(image1, image2): # Функция подсчета коэффициента шума
    return (np.sum(image1 != image2)/(image1.shape[0] * image1.shape[1])) * 100
```

[15]:
```
# Первоначаньное изображение (Тестируем)
show_me(img)
print("Коэффициент шума в процентах:")
print(noise_factor(img, img))
```

```
Коэффициент шума в процентах:
0.0
```

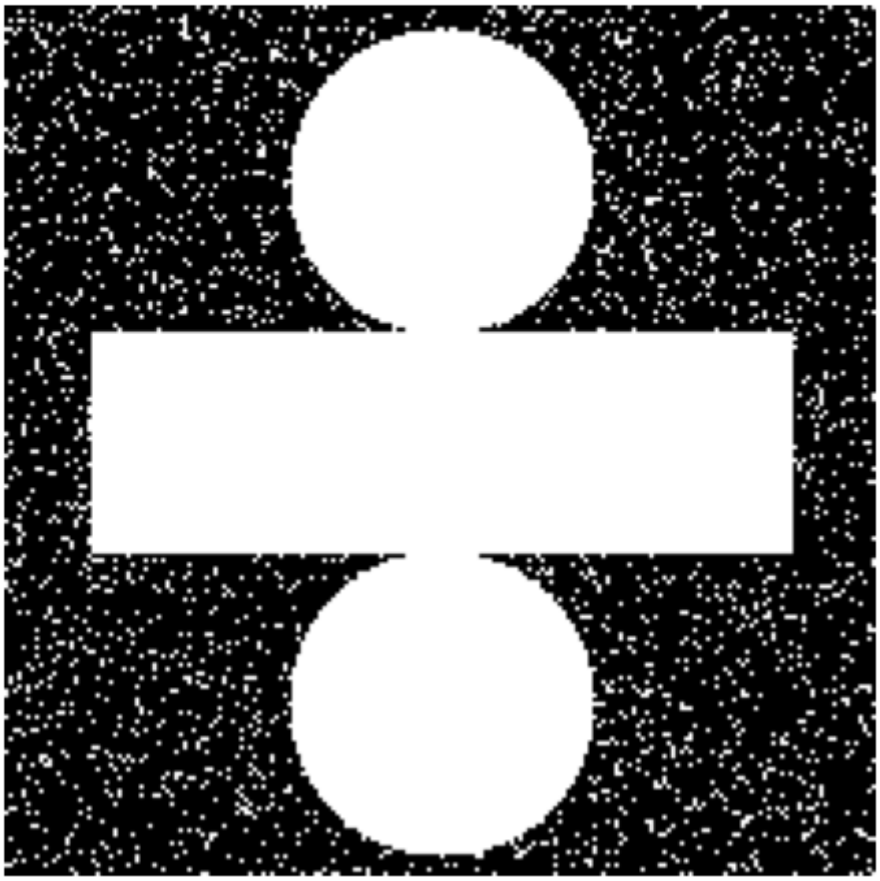


Исходное изображение

[16]:

```
# Изображение с импульсным шумом
show_me(noise_image)
print("Коэффициент шума в процентах:")
print(round(noise_factor(img, noise_image),4))
```

```
Коэффициент шума в процентах:
6.045
```

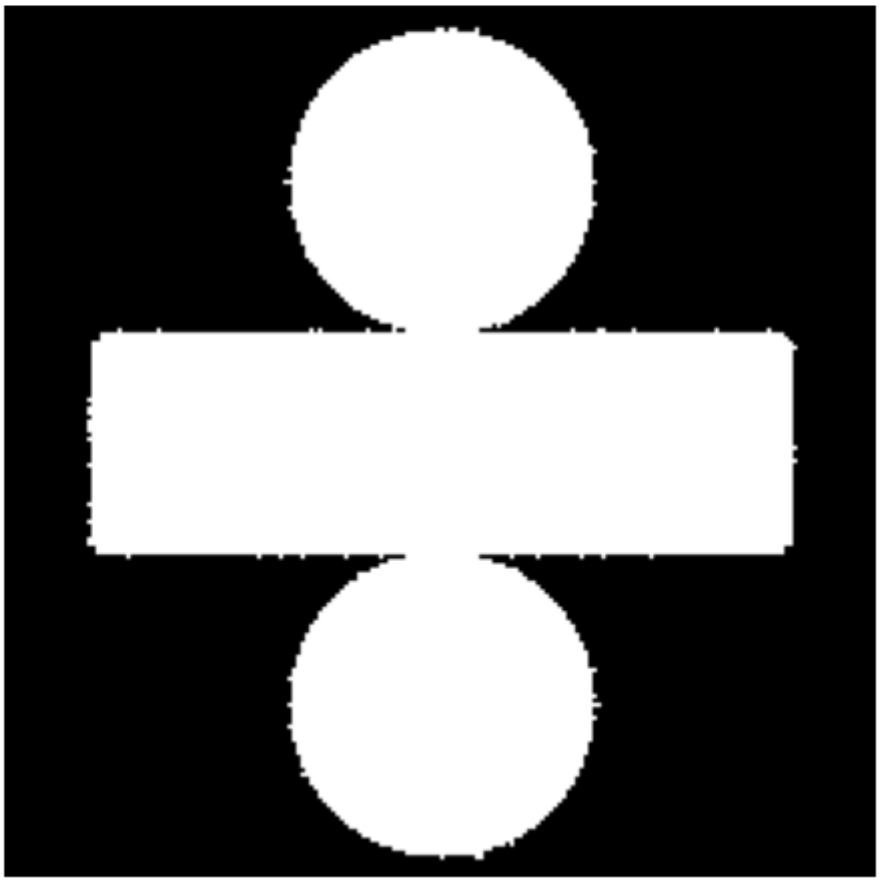


Изображение с импульсным шумом

Производим «вскрытие»:

[17]:
```
# Изоюражение после вскрытия
openimg = opening(noise_image, cross)

show_me(openimg)
print("Коэффициент шума в процентах:")
print(round(noise_factor(img, openimg),4))
```

```
Коэффициент шума в процентах:
0.2025
```

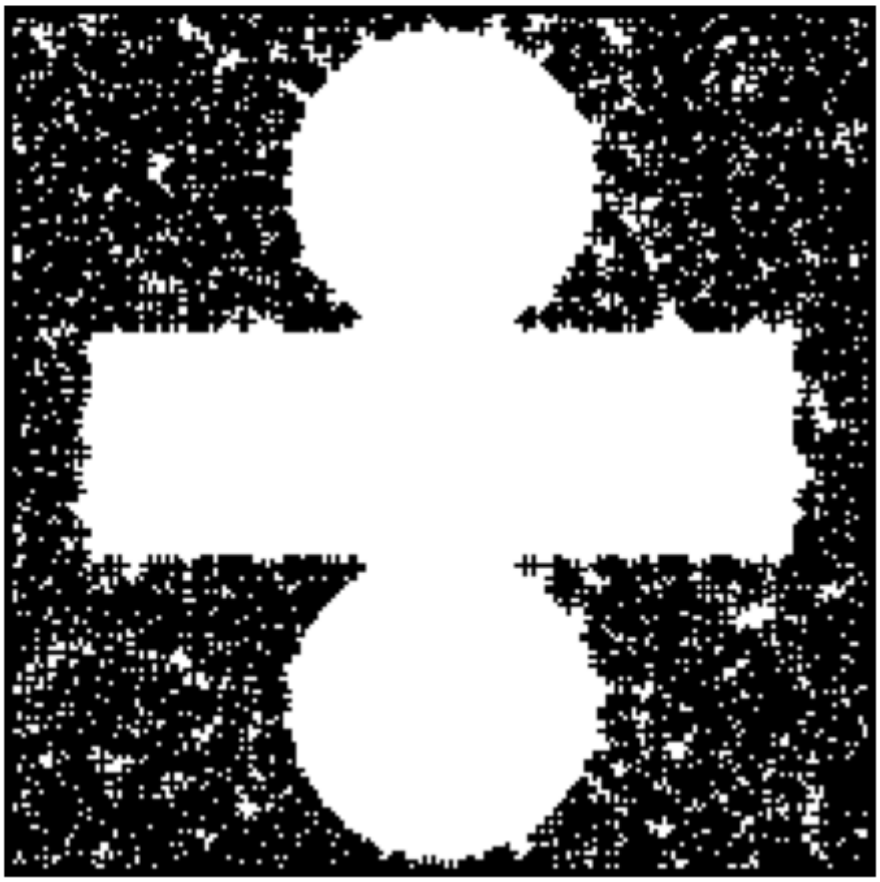


Изображение после «вскрытия»

Производим «закрытие»:

[18]:
```
# Изокражение после закрытия
closimg = closing(noise_image, cross)

show_me(closimg)
print("Коэффициент шума в процентах:")
print(round(noise_factor(img, closimg),4))
```

```
Коэффициент шума в процентах:
9.2
```



Изображение после «закрытия»

6. **Отфильтровать изображение при помощи логического фильтра. Подсчитать коэффициент шума для результата фильтрации. (Вход: изображение из пункта 2. Вывод: исходное (из пункта 1), зашумленное (из пункта 2) и отфильтрованное изображение. Коэффициент шума. Коэффициент шума считать, как количество не совпавших пикселей между отфильтрованным изображением и исходным (не зашумленным) из пункта 1. Таблица логического фильтра в лекции слайд 17).**

Алгоритмы бинарной обработки часто называются логическим фильтром или логической пространственной обработкой.

Изображение считается имеющим 1 (логическое да) и 0 (логическое нет), т.е. имеющим логические переменные и им соответствуют операции пространственной обработки (применение логических функций к изображению).

Операция клеточной логики – это пространственно-инвариантные преобразования:

$$y(m,n) = \Phi\{(x(k,l)), (k-m, n-l) \in D\}, \text{где}$$

$x(k,l)$ – это входное изображение, D – окно (клетка), область действия оператора Ф.

Оператор Ф называют также логической функцией от отсчетов исходного изображения, попавших в клетку (клетка – это окно небольших размеров).

Логический фильтр убирает из изображения несвязные точки поля яркости - выпавшие 0 и 1, т.е. фильтрует несвязные 0 и 1. Двоичное число x_0 x_1 x_2 x_3 x_4 показывает номер строки в таблице. По номеру смотрим значения y и присваиваем выходным значениям.

Альтернативный вариант – таблица заменяется формулой:

$$y = x_0 \wedge (x_0 \vee x_1 \vee x_2 \vee x_3 \vee x_4) \vee \neg x_0 \wedge (x_0 \wedge x_1 \wedge x_2 \wedge x_3 \wedge x_4)$$

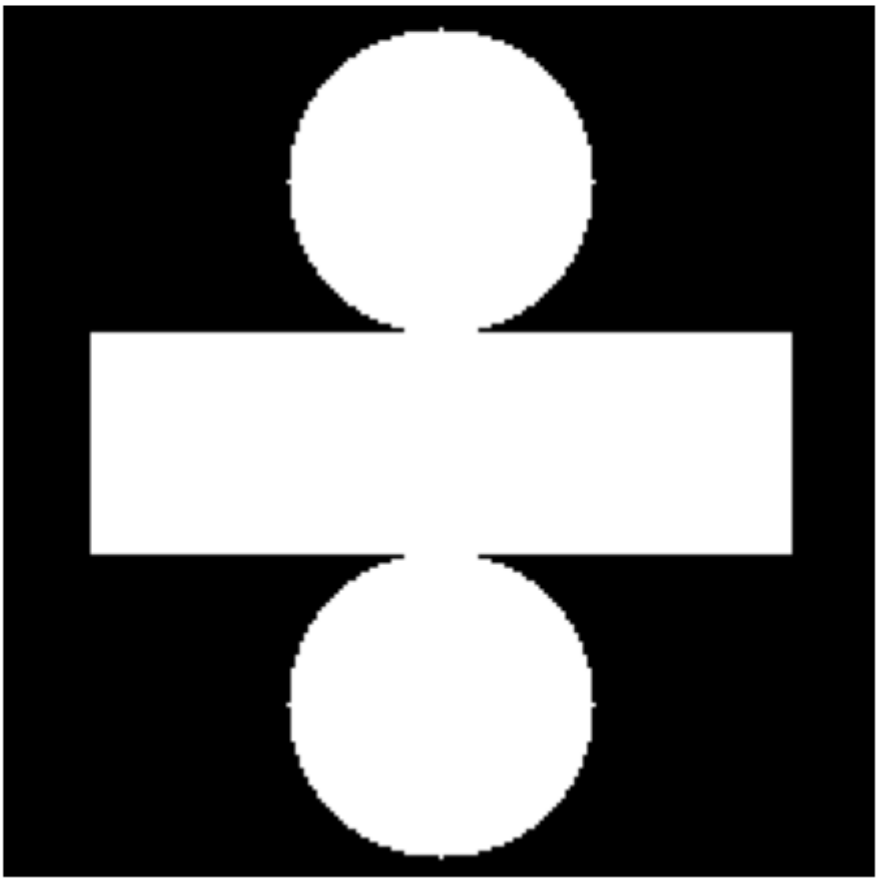
[19]:
```python
def value_filter(image, i, j): # Формирует значения для логического фильтра
    filter_value = image[i, j] and (image[i - 1, j] or image[i, j - 1] or image[i, j + 1] or image[i + 1, j]) \
        or (not image[i, j]) and (image[i - 1, j] and image[i, j - 1] and image[i, j + 1] and image[i + 1, j])
    return int(filter_value)
```

[20]:
```python
def logica(image): # Логический фильтр

    image_rows, image_cols = image.shape # Разсмерность
    output_image = np.copy(image) # Копируем

    # Временная картика с доп окнами (рамками)
    temp_image = np.zeros((image_rows + 2, image_cols + 2))
    temp_image[1: -1, 1: -1] = image
    temp_image = temp_image.astype(bool)

    for i, j in tqdm(np.ndindex((image_rows, image_cols))):
        output_image[i, j] = value_filter(temp_image, i, j)

    clear_output()
    return output_image
```

Производим фильтрацию:

[22]:
```
# Первоначаньное изображение (Тестируем)
show_me(img)
print("Коэффициент шума в процентах:")
print(noise_factor(img, img))
```

```
Коэффициент шума в процентах:
0.0
```

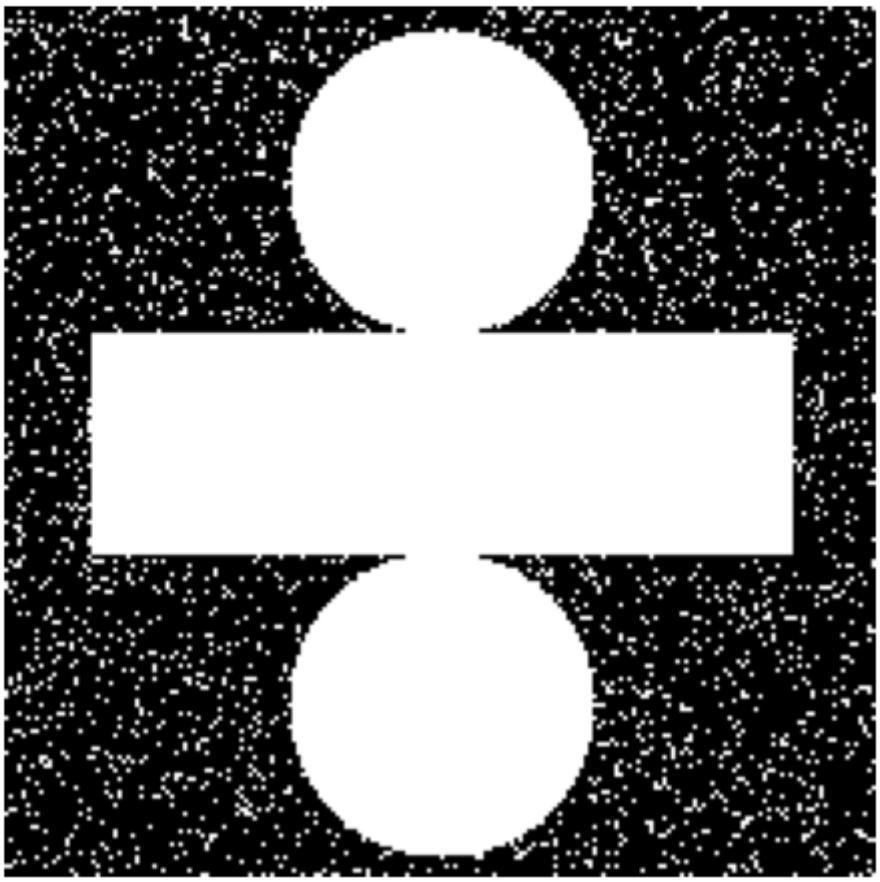


Исходное изображение

[23]:

```
# Изображение с импульсным шумом
show_me(noise_image)
print("Коэффициент шума в процентах:")
print(round(noise_factor(img, noise_image),4))
```

```
Коэффициент шума в процентах:
6.045
```

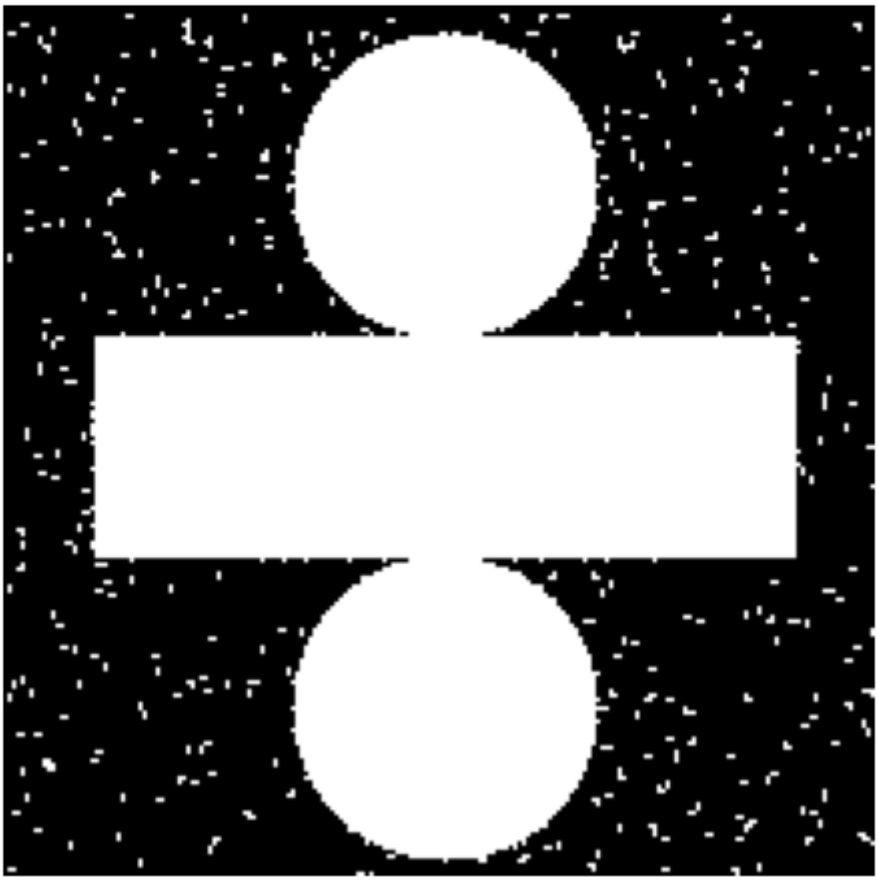


Изображение с импульсным шумом

[24]:

```
# Изображение после обработки логическим фильтром
logicimg = logica(noise_image)
show_me(logicimg)
print("Коэффициент шума в процентах:")
print(round(noise_factor(img, logicimg),4))
```

```
Коэффициент шума в процентах:
42.7775
```



Изображение после обработки логическим фильтром

7. **На исходном изображении с помощью морфологических операций выделить контур объекта. Выяснить, когда контур получается внешним, внутренним, четырёхсвязным, восьмисвязным. (Вход: изображение из пункта 1. Вывод: исходное изображение, изображение с выделенными контурами.**

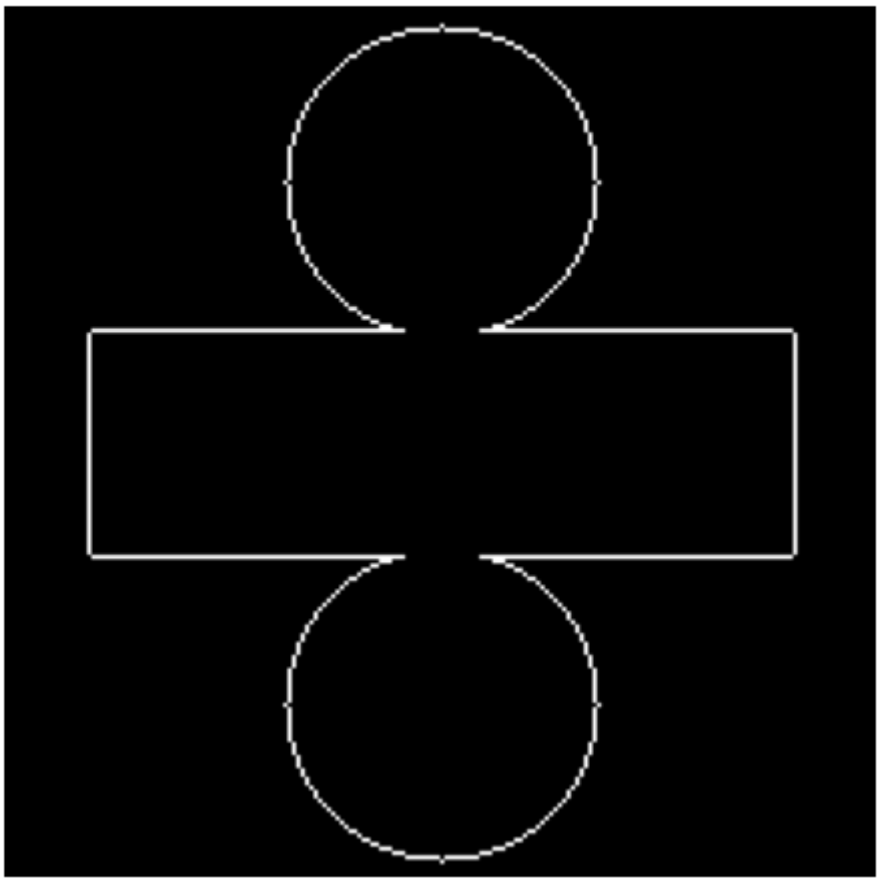
Для выделения внешнего контура берём разность изображения, полученного дилатацией, и исходного изображения.

Для внутреннего – разность исходного изображения и изображения, полученного после применения эрозии.

[26]:
```
outer = np.array([
    [0, 1, 0],
    [1, 1, 1],
    [0, 1, 0]
])
inner = np.array([
    [0, 1, 0],
    [1, 1, 1],
    [0, 1, 0]
])
four_connected = np.array([
    [1, 1, 1],
    [1, 1, 1],
    [1, 1, 1],
])
eight_connected = np.array([
    [0, 1, 0],
    [1, 1, 1],
    [0, 1, 0]
])
```

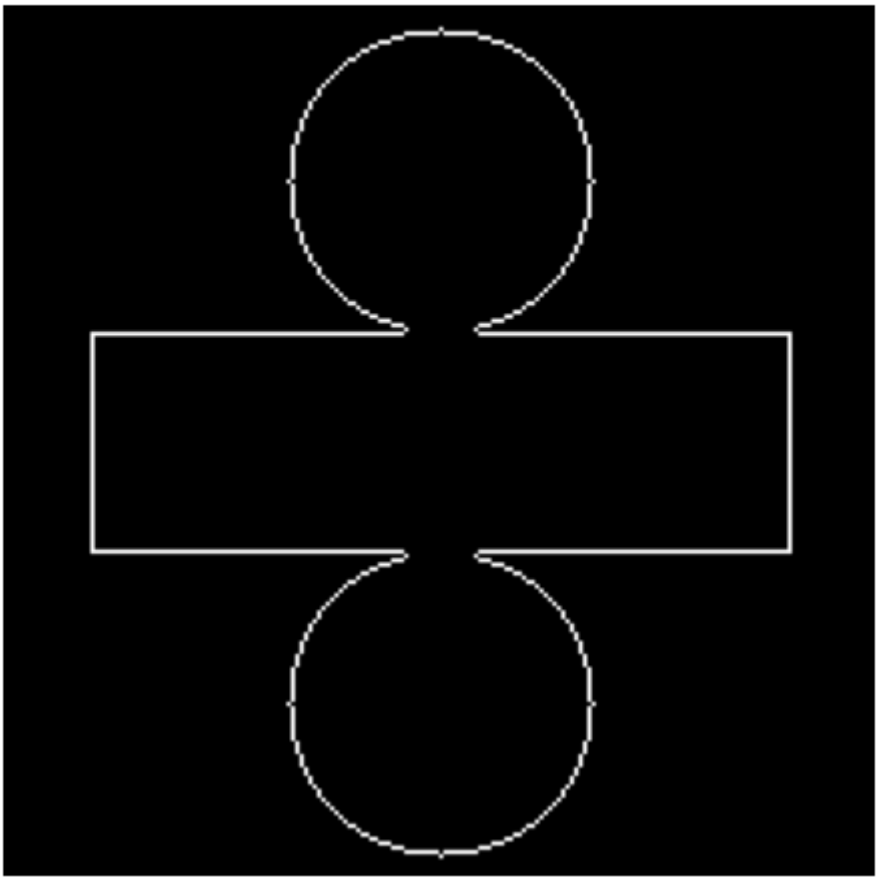
[28]:

```
# Внешний контур
show_me((img ^ dilat(img, outer )))
```



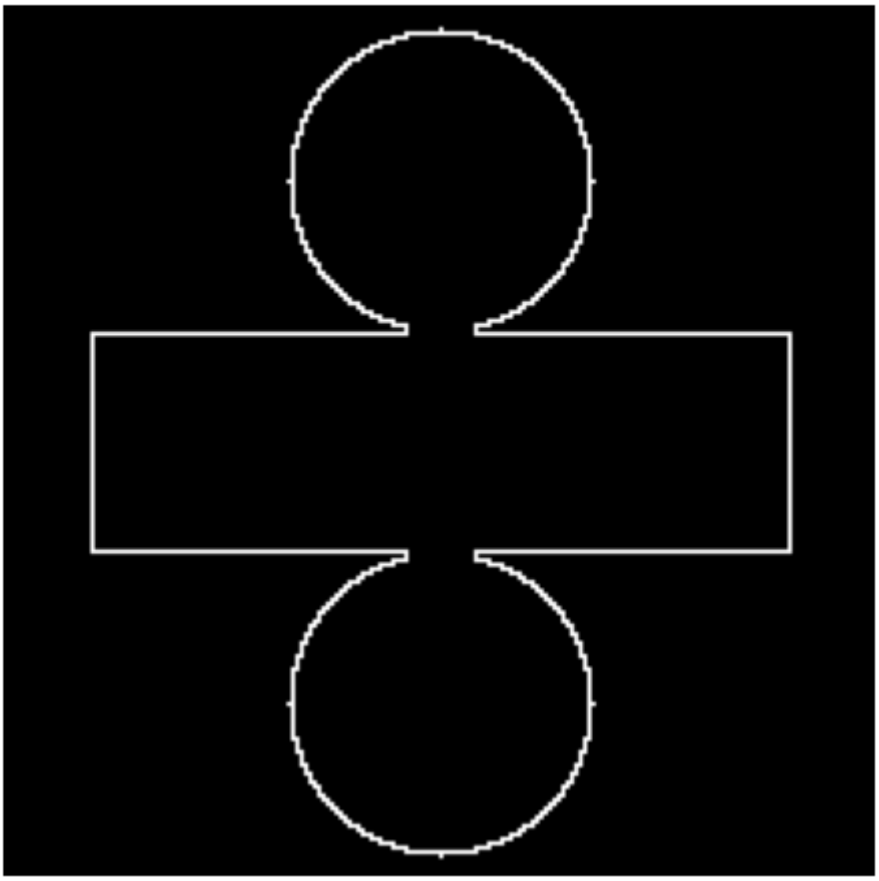
[29]:

```
# Внутренний контур
show_me((img ^ eros(img, inner)))
```



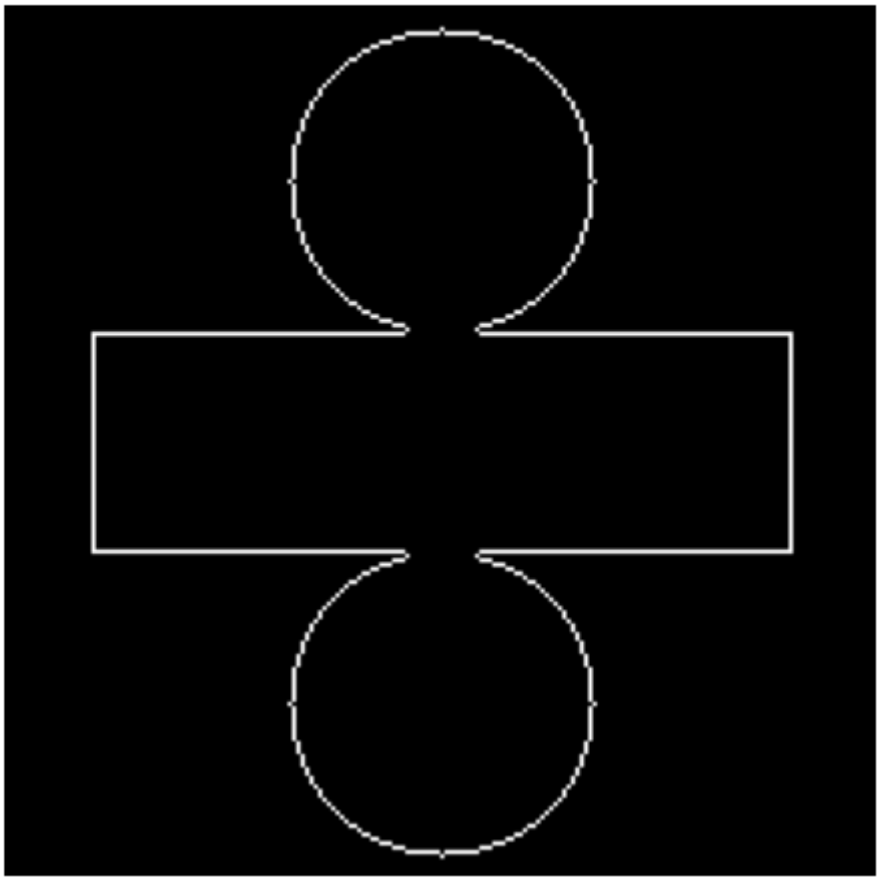
[30]:

```
# Четырехсвязный внутренний контур
show_me((img ^ eros(img, four_connected)))
```



[31]:

```
# Восьмисвязный внутренний контур
show_me((img ^ eros(img, eight_connected )))
```

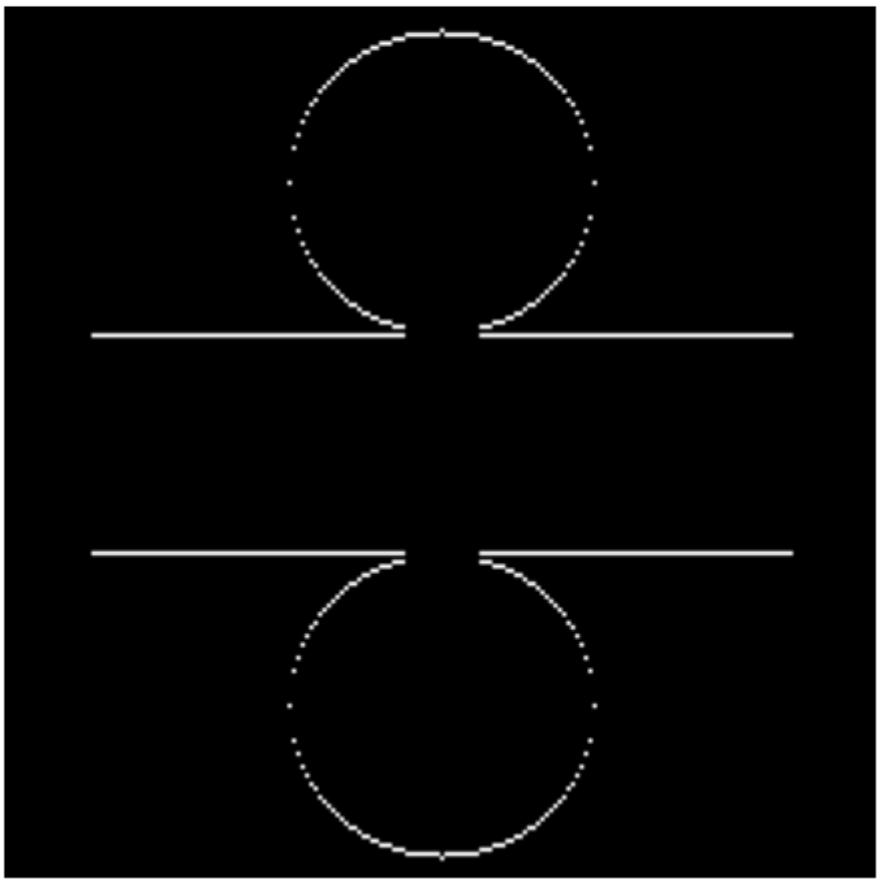


8. **На исходном изображении с помощью морфологических операций выделить горизонтальные и вертикальные контуры объекта. (Вход: изображение из пункта 1. Вывод: исходное изображение, изображение с выделенными горизонтальными контурами, изображение с выделенными вертикальными контурами).**

Для выделения внешнего контура берём разность изображения, полученного дилатацией, и исходного изображения.

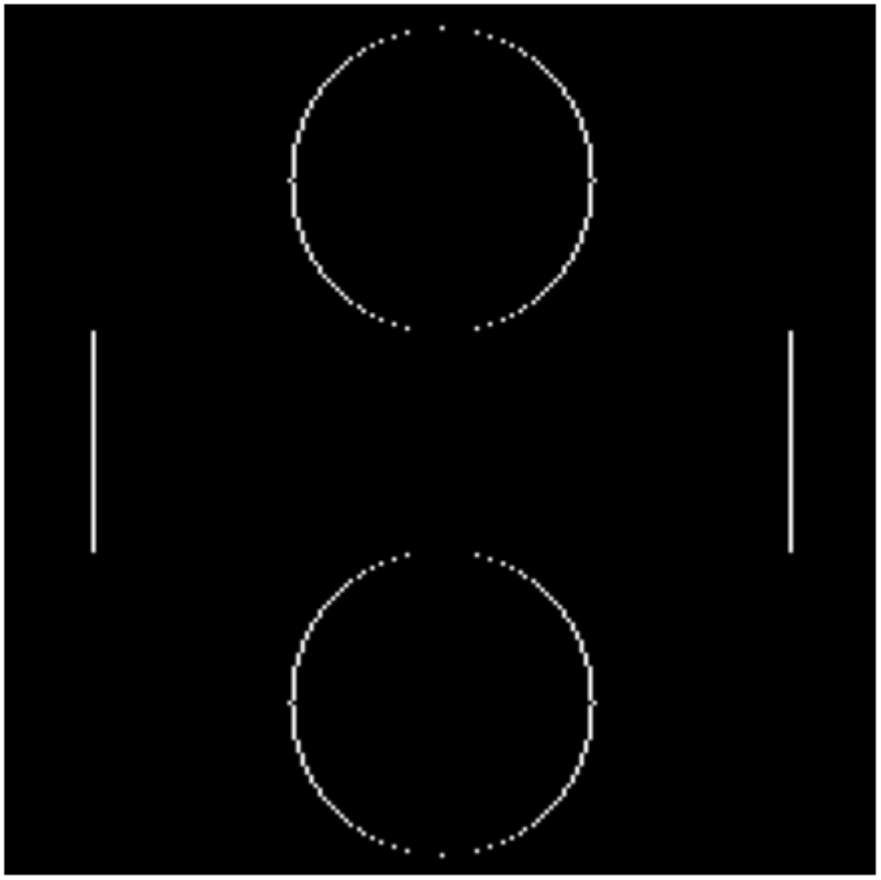
Для внутреннего – разность исходного изображения и изображения, полученного после применения эрозии.

[34]:
```
# С выделенными горизонтальными контурами
contours = np.array([ [1], [1], [1] ])
show_me((img ^ eros(img, contours)))
```



[35]:

```
# С выделенными вертикальными контурами.
contours = np.array([ [1, 1, 1] ])
show_me((img ^ eros(img, contours)))
```



Заключение

В ходе лабораторной работы было изучено взаимодействие с бинарными изображениями в языке Python. Была совершена фильтрация зашумленных изображений с помощью операций открытия, закрытия и логической операции.

Были сделаны выводы и характере контуров в зависимости от операции и формы элемента. Было произведено сравнение качества подавления шумов различных операций, лучше всего справляется вскрытия. Было произведено выделение горизонтальных и вертикальных контуров.