Zachodniopomorski Uniwersytet Technologiczny w Szczecinie

Wydział Informatyki

BINARYZACJA

Karol Działowski

nr albumu: 39259
przedmiot: Ekstrakcja cech

Szczecin, 2 grudnia 2020

Spis treści

1 Cel laboratorium

Celem laboratorium było przeprowadzenie binaryzacji obiektów badawczych. Należało wyznaczyć maski obiektów (segmentacja) za pomocą dowolnej metody.

2 Proces binaryzacji

W celu binaryzacji wykorzystano bibliotekę *OpenCV* [1]. Wykonano następujące kroki:

1. Dodanie białej ramki o grubości 20 pikseli

2. Progowanie binarne z wykorzystaniem *cv2.ADAPTIVE_THRESH_GAUSSIAN_C* (progowanie adaptacyjne)

3. Tworzenie maski tła wypełniając jasne piksele zaczynając od punktu (0, 0) – *cv2.floodFill*

4. Operacja otwarcia i zamknięcia *cv.morphologyEx*

W pierwszym kroku dodano białą ramkę do obrazów. Pozwala to na wykonanie operacji wypełnienia w przypadku, gdy kształt nachodzi na krawędzie ekranu. Kod tej operacji przedstawiono na listingu (1).

Kod źródłowy 1: Kod dodawania ramki
Źródło: Opracowanie własne

```
im_in = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
im_in = cv2.copyMakeBorder(im_in, 20, 20, 20, 20,
                           cv2.BORDER_CONSTANT, value=[255, 255, 255])
```

Następnie przeprowadzono progowanie binarne. W tym celu wykorzystano progowanie adaptacyjne, które dawało najlepsze rezultaty dla logotypów z światłocieniem. Stworzono maskę obrazu przeprowadzając operację wypełnienia (*flood fill*) zaczynając od lewego górnego rogu i wypełniając wszystkie białe sąsiadujące piksele. Kod progowania i wypełnienia przedstawiono na listingu (2).

Kod źródłowy 2: Kod progowania binarnego
Źródło: Opracowanie własne

```
im_th = cv2.adaptiveThreshold(
    im_in, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 2)
im_floodfill = im_th.copy()
h, w = im_th.shape[:2]
mask = np.zeros((h+2, w+2), np.uint8)
cv2.floodFill(im_floodfill, mask, (0, 0), 255)
im_floodfill_inv = cv2.bitwise_not(im_floodfill)
im_out = im_th | im_floodfill_inv
```

Na końcu wykorzystano operacje morfologiczne w celu zniwelowania zakłóceń oraz w celu połączenia kształtów w logotypach składających się z kilku rozłącznych figur, np. Tesla lub Mitsubishi. Kod operacji morfologicznych przedstawiono na listingu (4).

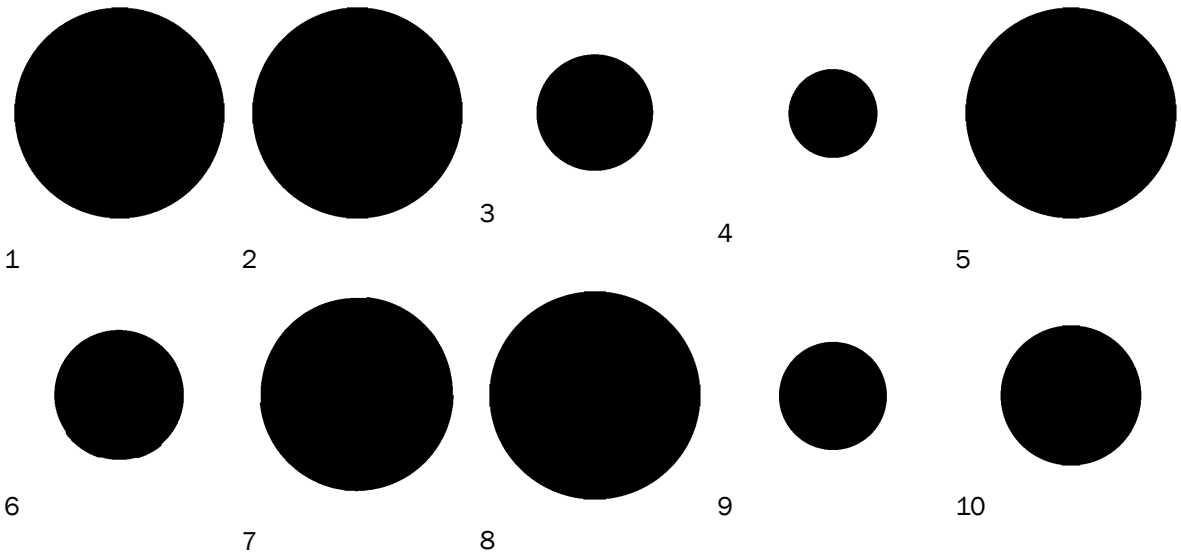
Kod źródłowy 3: Kod operacji morfologicznych
Źródło: Opracowanie własne

```
kernel = np.ones((5, 5), np.uint8)
im_out = cv2.morphologyEx(im_out, cv2.MORPH_OPEN, kernel)
kernel = np.ones((20, 20), np.uint8)
im_out = cv2.morphologyEx(im_out, cv2.MORPH_CLOSE, kernel)
im_out = cv2.bitwise_not(im_out)
```

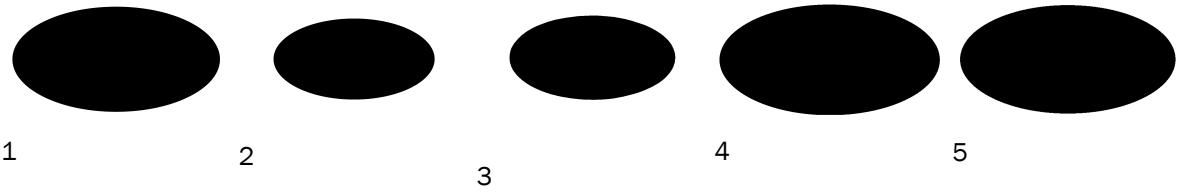
3 Zbiór danych po binaryzacji

Przeprowadzono binaryzacje wszystkich 10 klas obiektów:

1. BMW (rysunek 1)
2. KIA (rysunek 2)
3. Mitsubishi (rysunek 3)
4. Volvo (rysunek 4)
5. Peugeout (rysunek 5)
6. Honda (rysunek 6)
7. Subaru (rysunek 7)
8. Tesla (rysunek 8)
9. Renault (rysunek 9)
10. Toyota (rysunek 10)



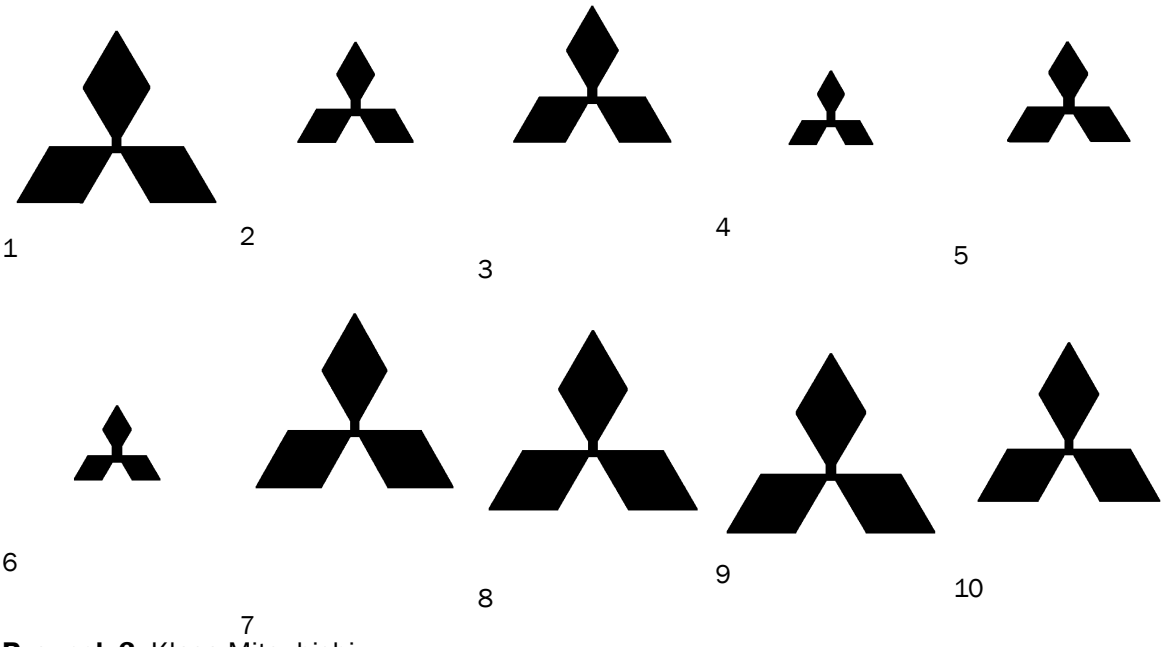
Rysunek 1: Klasa BMW



1 2 3 4 5

6 7 8 9 10

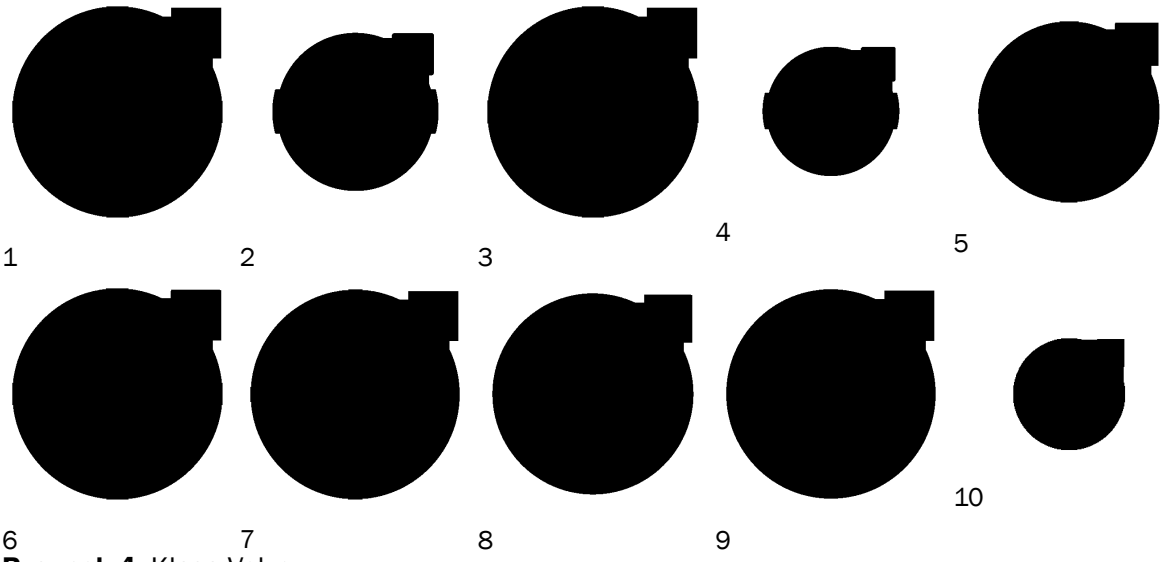
Rysunek 2: Klasa KIA



1 2 3 4 5

6 7 8 9 10

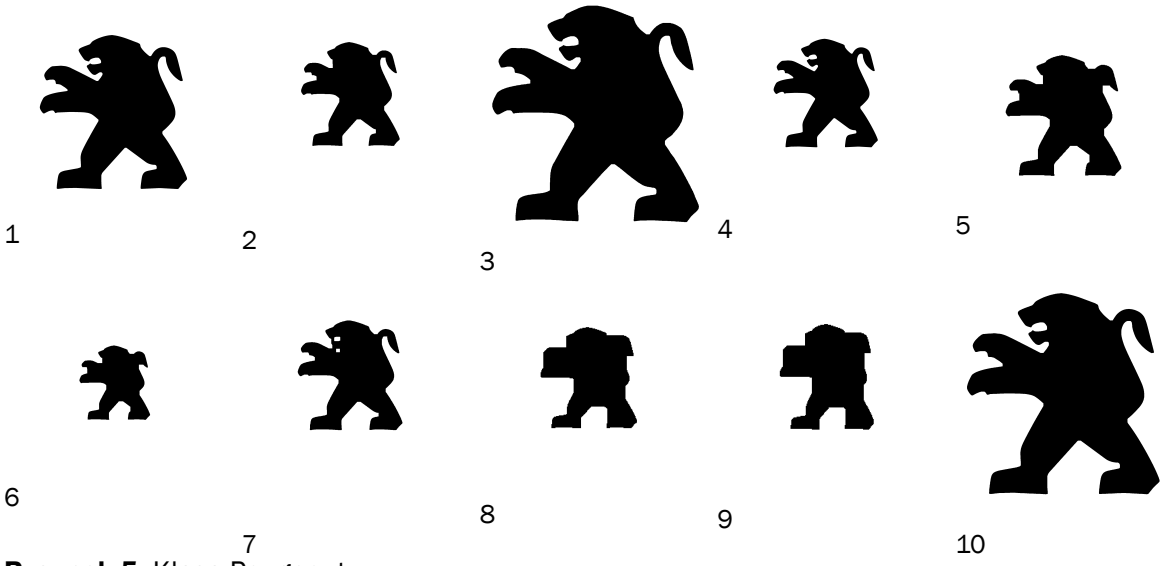
Rysunek 3: Klasa Mitsubishi



1 2 3 4 5

6 7 8 9 10

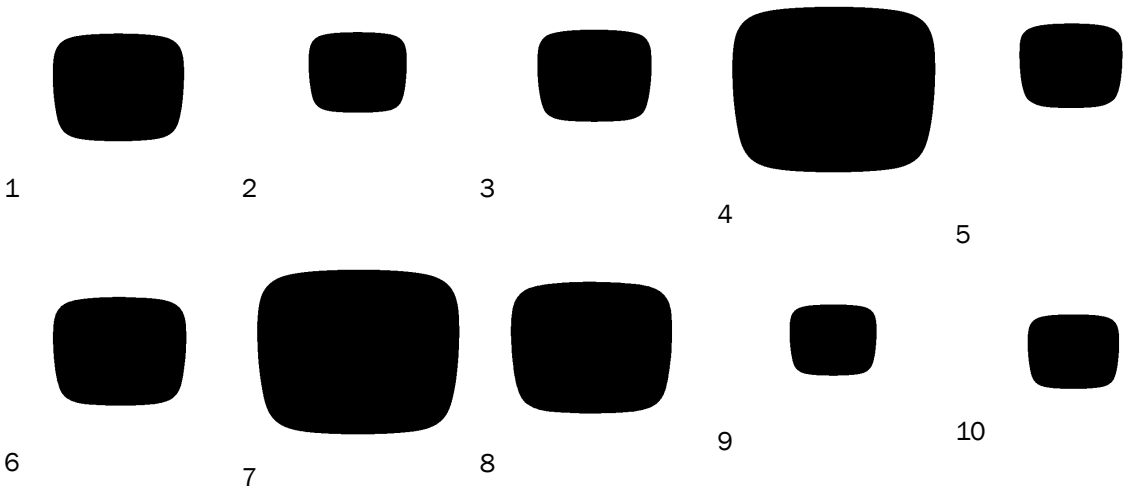
Rysunek 4: Klasa Volvo



1 2 3 4 5

6 7 8 9 10

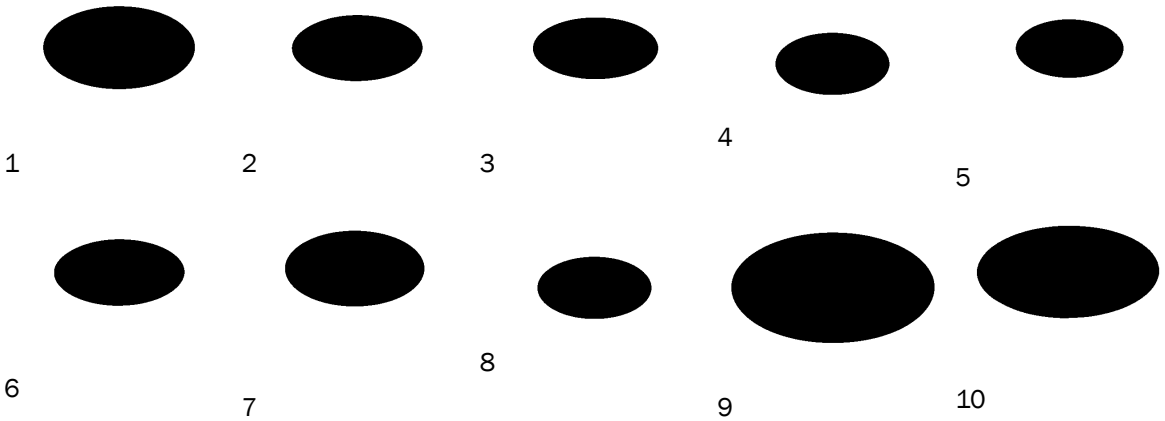
Rysunek 5: Klasa Peugeout



1 2 3 4 5

6 7 8 9 10

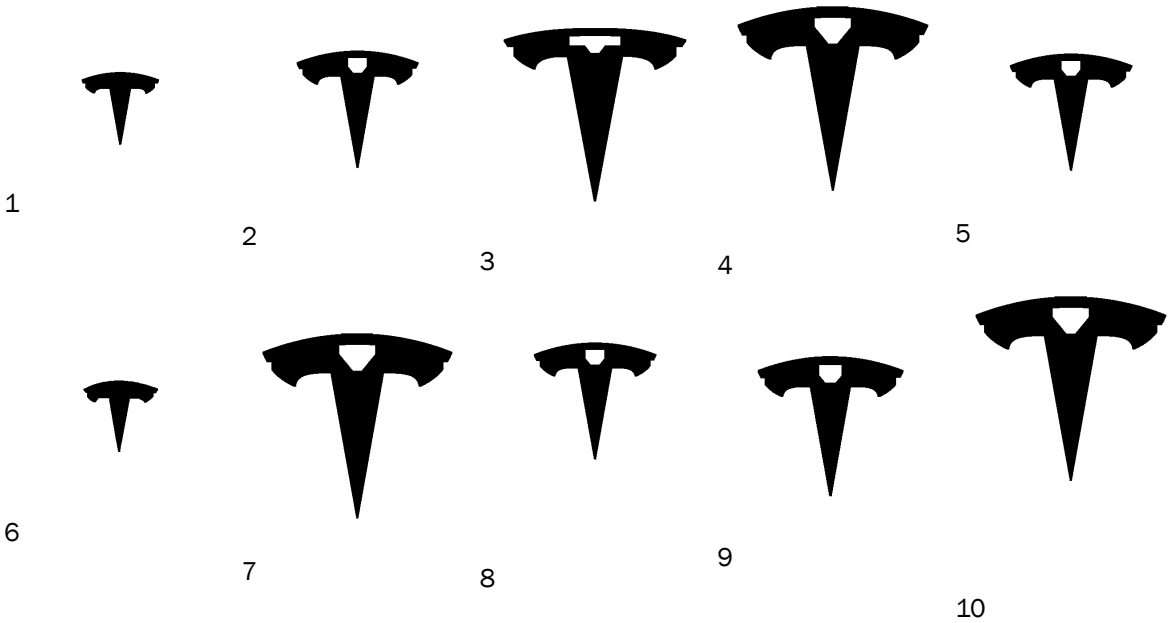
Rysunek 6: Klasa Honda



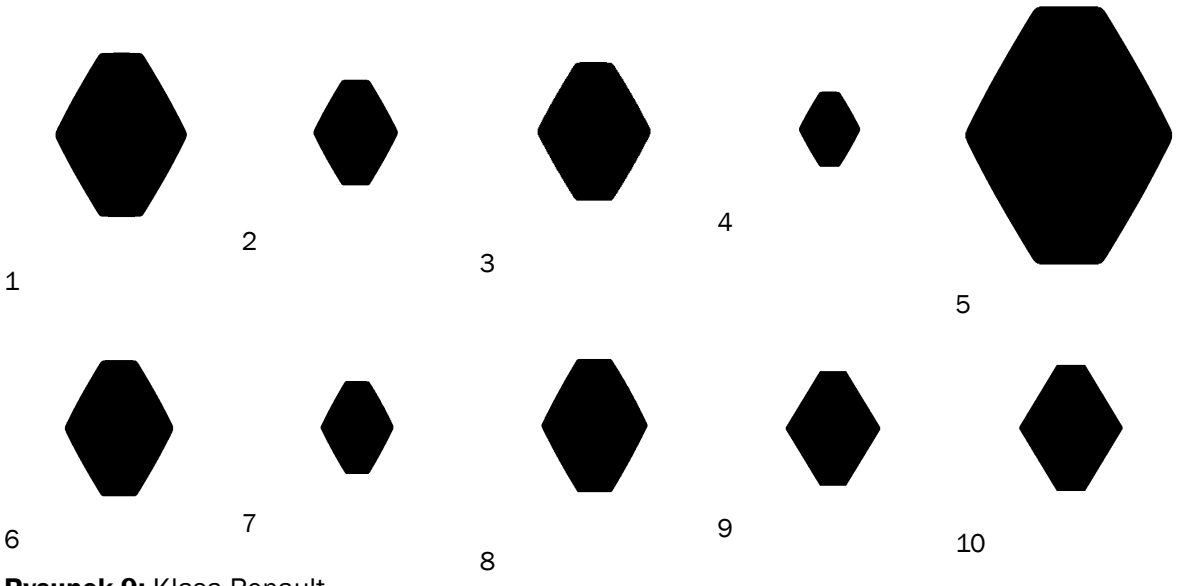
1 2 3 4 5

6 7 8 9 10

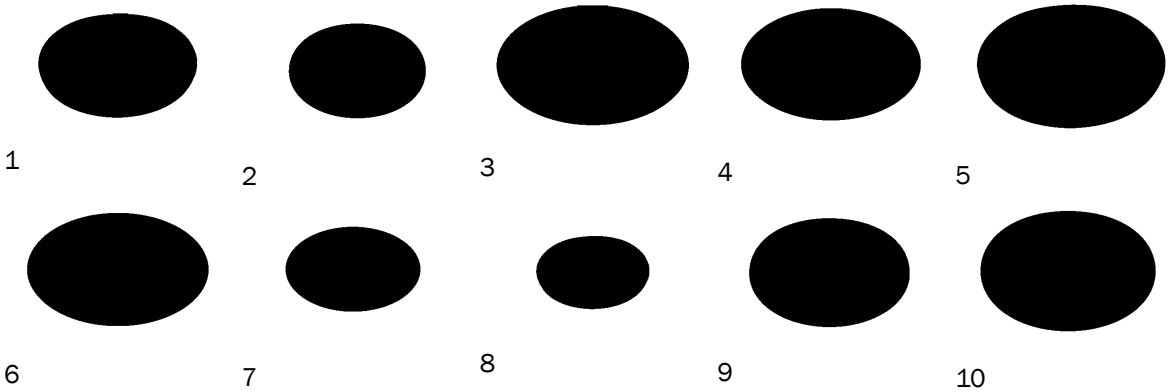
Rysunek 7: Klasa Subaru



Rysunek 8: Klasa Tesla



Rysunek 9: Klasa Renault



Rysunek 10: Klasa Toyota

4 Podsumowanie

Przeprowadzono binaryzację zbioru logotypów liczącego 100 obrazów. Wykorzystano progowanie adaptacyjne, wypełnianie oraz operacje morfologiczne. Wszystkie wybrane logotypy różnią się od siebie kształtem. Najbardziej zbliżone do siebie zbiory to logo Subaru (7), logo Toyoty (10) oraz logo KIA (2), a także Volvo (4) i BMW (1).

Bibliografia

[1] Itseez: *Open source computer vision library*, 2015.

A Całość implementacji binaryzacji

Kod źródłowy 4: Kod przeprowadzania binaryzacji
Źródło: Opracowanie własne

```
import cv2
import numpy as np
import os
from pathlib import Path

folders = ["01_bmw", "02_kia", "03_mitsubishi", "04_volvo",
           "05_peugeout", "06_honda", "07_subaru", "08_tesla",
           "09_renault", "10_toyota"]

for name in folders:
    files = os.listdir('./dataset/'+name)
    for f in files:
        filename = "./dataset/" + name + "/" + f
        im_in = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)

        im_in = cv2.copyMakeBorder(im_in, 20, 20, 20, 20,
                                   cv2.BORDER_CONSTANT, value=[255, 255, 255])

        # Threshold.
        # Set values equal to or above 220 to 0.
        # Set values below 220 to 255.
        # th, im_th = cv2.threshold(im_in, 220, 255, cv2.THRESH_BINARY_INV)
        im_th = cv2.adaptiveThreshold(
            im_in, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 2)

        # im_th = cv2.Canny(im_in, 100, 200)

        # Copy the thresholded image.
        im_floodfill = im_th.copy()

        # Mask used to flood filling.
```

```
        # Notice the size needs to be 2 pixels than the image.
        h, w = im_th.shape[:2]
        mask = np.zeros((h+2, w+2), np.uint8)

        # Floodfill from point (0, 0)
        cv2.floodFill(im_floodfill, mask, (0, 0), 255)

        # Invert floodfilled image
        im_floodfill_inv = cv2.bitwise_not(im_floodfill)

        # Combine the two images to get the foreground.
        im_out = im_th | im_floodfill_inv
        kernel = np.ones((5, 5), np.uint8)
        im_out = cv2.morphologyEx(im_out, cv2.MORPH_OPEN, kernel)
        kernel = np.ones((20, 20), np.uint8)
        im_out = cv2.morphologyEx(im_out, cv2.MORPH_CLOSE, kernel)

        # invert image
        im_out = cv2.bitwise_not(im_out)

        # Display images.
        # save binarized
        if not os.path.exists("./dataset/binarized/" + name):
            os.makedirs("./dataset/binarized/" + name)
        path_img_target = os.path.join(
            "./dataset/binarized/" + name + "/" + f)
        path_img_target = str(Path(path_img_target).with_suffix('.png'))
        print(path_img_target)
        cv2.imwrite(path_img_target, im_out)
```