Image Processing Python Lab - Morphology

Here is a guide to the different parts of the lab:

❓ - This symbol means that there is a question related to it. Your answer to the specified question should go in the yellow box below it.

ℹ️ - This symbol means that there is some additional or important information relating to the lab or specific exercise.

💬 - This symbol means that the following text is part of the overarching story you follow, giving you more context into what you are doing and how it is applicable in real life situations.

Objectives

- Investigate the use of logical operations in binary images.
- Apply morphological operations to denoise images.
- Use logical operations to match binary images.
- Use the morphological operations to open or close areas in a binary image.

Convert to PDF

First try to convert to PDF via LaTeX:

- First you need to make sure that you have LaTeX installed (this goes for whatever operating system you are using). You can go to https://www.latex-project.org/get/ and get the proper distribution for your operating system.
- After doing that, you can go to where xelatex.exe is found (this should be in AppData\Local\Programs\MiKTeX\miktex\bin\x64\xelatex.exe if you are getting MiKTeXfor Windows, or the equivalent on other any other OS). If you are having a hard time finding the folder, you can use "where xelatex" as a command for Windows, the find command for Mac and Linux or just use the file explorer.

- After making sure you have xelatex setup, you can try the "PDF via LaTeX" in the "File" -> "Download as" menu in jupyter notebook

If the above option does not work you can follow these instructions:

- Go into "File" -> "Print Preview"
- Make sure the generated PDF is readable then you can use Ctrl+P or Command+P (on MacOS) to Print Page, where you select to save to PDF rather than printing (each system has a different interface).
- Before submitting make sure that all answers/code/plots are clearly visible.

If none of the methods work, contact a TA during labs and ask about further steps.

Working with the license plate that Jeeves picked out and cropped, you are aware that it would be most efficient to convert it (and any future license plates) to a binary image. There were a few more cars spotted in the vicinity and you know that throughout your investigation, hundreds of license plates would need to be reviewed. You notice that some of the plates are slightly chipped or dirty, and in order to automate the recognition of plates, you would want to remove noise. With that in mind, Jeeves has picked out a few license plates and converted them to binary images, so that you can perfect your algorithm. After all, if you miss even one license plate, it might be the one of the robbers.

In the folder of the Lab Notebook you are also given several folders with images to accompany the excerises you're about to start. In most of the execises the path to the image is already specified and you only need to add the name of the file you want to work with. Additionally in some code cells you'll be given code that you can use as is. Please do not change the given functions signature (that includes the names, inputs and return values).
You can read the image(s) into the notebook using the command `plt.imread("path/to/file.jpg")`

Some code cells and excercises have some code already provided for you to use. The steps for the exercises you need to do in the code cells are marked with comments. Your code should go under the corresponding comment line describing the specific

step. You can treat those lines as "TODO" items.

In [ ]:
```
##################################################################################################
# Importing various modules into PYTHON. These will be used throughout this Jupyter Notebook
##################################################################################################

# If you're having problems with the imports uncomment the following lines to install the libraries
# !pip install opencv-python
# !pip install numpy
# !pip install pickle-mixin
# !pip install matplotlib
# !pip install imutils

# import matplotlib for data visualisation
import matplotlib.pyplot as plt

# import NumPy for better matrix support
import numpy as np

# import Pickle for data serialisation
import pickle as pkl

# import cv2 and imutils for image processing functionality
import cv2
import imutils
```

Below there are some plotting functions to make your work easier.

In [ ]:
```
# Displays a given gray-scale image using matplotlib.pyplot
def plotImage(img, title=""):
    # Display image
    plt.imshow(img, cmap=plt.cm.gray, vmin=0, vmax=255)
    plt.title(title)
    plt.show()
```

In [ ]:
```
# Loads an image, in gray-scale by default
def loadImage(filepath, filename, grayscale=True):
    return cv2.imread(filepath+filename, cv2.IMREAD_GRAYSCALE if grayscale else cv2.IMREAD_COLOR)
```

```
def makeGrayscale(img):
    return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

In case you want to use custom images, place them in the folder `resources/lab_02/other/` and also read and show the image that you have chosen

Note: Reading and displaying an image using cv2 is also possible. However within the jupyter notebook it causes problems

In [ ]:
```
# Read and show any custom images you are using
```

PYTHON HANDS-ON Assignment II.1: Reflect binary image

In the vicinity of the robbery, a car was hit by a van speeding past. As investigator, you have access to the black box of the car, however the van passed just in the blind spot of the camera, so you can't see the back plate. You start rewatching the accident, and this seems like a useless clue, but just as you're about to give up, you notice that the side mirror caught the face of the van and... Yes, the license plate.
It matches the plate you're looking for, but in order to present it in court it better be easily readable.

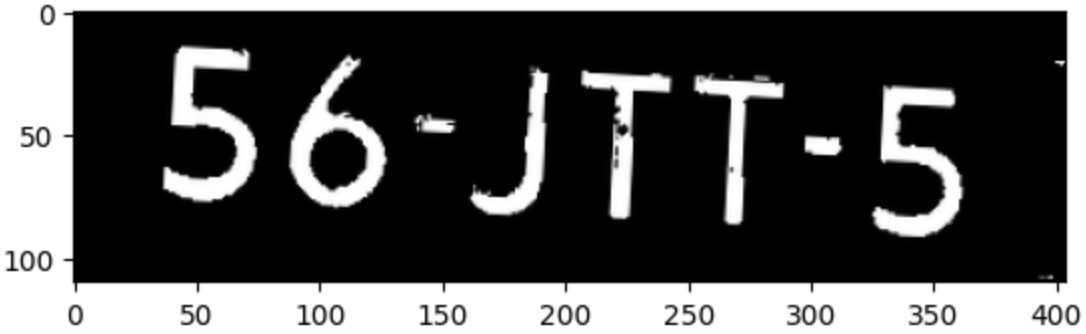
Completion requirements for this assignment:

- ☐ Reflect the binary image of the license plate to make it easily readable

II.1.1 Image reflection

You are given the image in directory `resources/lab_02/exercise_01/`. Read the image and display them to inspect shape. Flip the image along an axis to have readable version of the image.

In [ ]:
```
path = "resources/lab_02/exercise_01/"
plate = loadImage(path, "reflected_bin_plate.png", grayscale=True)

# Flip the image
image = []
for line in plate:
    image.append(line[::-1])

# Show the result
plotImage(image)
```

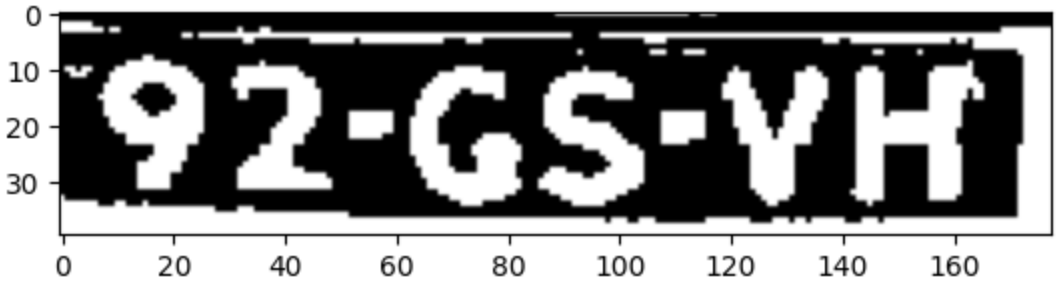


PYTHON HANDS-ON Assignment II.2: Denoising binary images using morphology

Completion requirements for this assignment:

- ☐ Clear the noise using morphological operations
- ☐ Write what structuring element(s) you used to clear each image

In [ ]:
```
# Load image
path = "resources/lab_02/exercise_02/"
filename = "bin_plate1.png"

# Display image
img = loadImage(path, filename) # Loads the image in gray-scale
plotImage(img)
```



II.2.1 Denoise the given images

You are given the images in directory `resources/lab_02/exercise_02/` . Read the images and display them to inspect the noise shape and size. Use morphological operations provided by cv2 to remove as much noise as possible.

i You can try plotting only parts of the image in order to zoom in and look at noise that's just a few pixels wide

i You can make a structuring element like this:

```
element_3x3 = np.array([[0, 1, 0],
                        [1, 1, 1],
                        [0, 1, 0]], np.uint8)
```

And change the size and values to your need

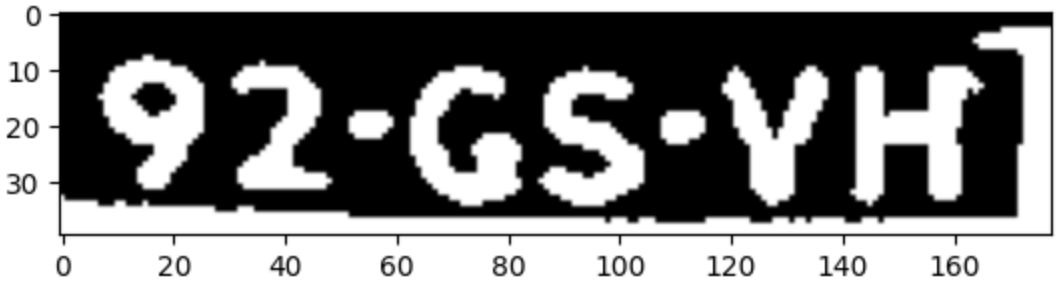
In [ ]:
```python
def denoise(img, structuring_element):
    eroded = cv2.erode(img, structuring_element)
    return cv2.dilate(eroded, structuring_element)

## Plate 1 ##
plate1 = loadImage(path, "bin_plate1.png", grayscale=True)

# Initialize structuring element(s)
cross = np.array([[0,1,0],[1,1,1],[0,1,0]], np.uint8)
cross2 = np.array([[0,1,1,0],[1,1,1,1],[1,1,1,1], [0,1,1,0]], np.uint8)
square2 = np.array([[1,1],[1,1]], np.uint8)
square3 = np.array([[1,1,1],[1,1,1],[1,1,1]], np.uint8)
square4 = np.array([[1,1,1,1],[1,1,1,1],[1,1,1,1], [1,1,1,1]], np.uint8)
vertical =  np.zeros((3, 3), np.uint8)
```

```
vertical[:,1] = 1
diamond = np.array([
    [0, 0, 1, 0, 0],
    [0, 1, 1, 1, 0],
    [1, 1, 1, 1, 1],
    [0, 1, 1, 1, 0],
    [0, 0, 1, 0, 0]
], dtype=np.uint8)

# Apply morphological operations
image_cross = denoise(plate1, cross)
image_cross2 = denoise(plate1, cross2)
image_square2 = denoise(plate1, square2)
image_square3 = denoise(plate1, square3)
image_square4 = denoise(plate1, square4)
image_vertical = denoise(plate1, vertical)
image_diamond = denoise(plate1, diamond)
# Show result
print("Cross")
plotImage(image_cross)
print("Cross 2")
plotImage(image_cross2)
print("Square 2")
plotImage(image_square2)
print("Square 3")
plotImage(image_square3)
print("Square 4")
plotImage(image_square4)
print("Vertical side")
plotImage(image_vertical)
print("Diamond")
plotImage(image_diamond)

br = """
        ##################################################
        #              Next Plate     2                  #
        ##################################################

    """

print(br)
## Plate 2 ##
```

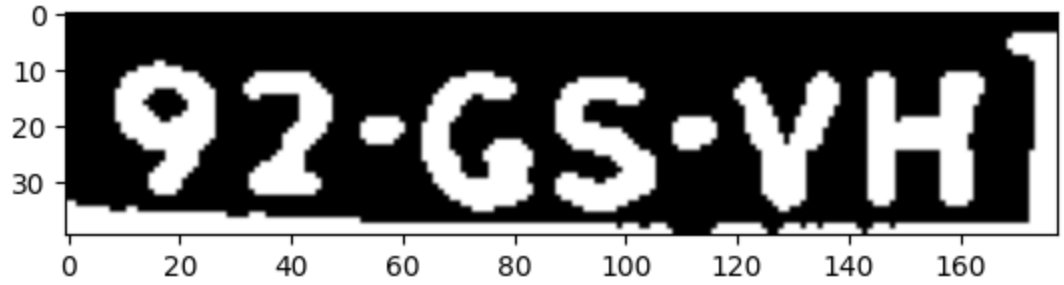
```
plate2 = loadImage(path, "bin_plate2.png", grayscale=True)

# Initialize structuring element(s)

# Apply morphological operations
image_cross = denoise(plate2, cross)
image_cross2 = denoise(plate2, cross2)
image_square2 = denoise(plate2, square2)
image_square3 = denoise(plate2, square3)
image_square4 = denoise(plate2, square4)
image_vertical = denoise(plate2, vertical)
image_diamond = denoise(plate2, diamond)

# Show result
print("Cross")
plotImage(image_cross)
print("Cross 2")
plotImage(image_cross2)
print("Square 2")
plotImage(image_square2)
print("Square 3")
plotImage(image_square3)
print("Square 4")
plotImage(image_square4)
print("Vertical")
plotImage(image_vertical)
print("Diamond")
plotImage(image_diamond)

br = """
        ##################################################
        #              Next Plate     3                  #
        ##################################################

    """

print(br)

## Plate 3 ##
plate3 = loadImage(path, "bin_plate3.png", grayscale=True)

# Initialize structuring element(s)
```

```
# Apply morphological operations
image_cross = denoise(plate3, cross)
image_cross2 = denoise(plate3, cross2)
image_square2 = denoise(plate3, square2)
image_square3 = denoise(plate3, square3)
image_square4 = denoise(plate3, square4)
image_vertical = denoise(plate3, vertical)
image_diamond = denoise(plate3, diamond)

# Show result
print("Cross")
plotImage(image_cross)
print("Cross 2")
plotImage(image_cross2)
print("Square 2")
plotImage(image_square2)
print("Square 3")
plotImage(image_square3)
print("Square 4")
plotImage(image_square4)
print("Vertical")
plotImage(image_vertical)
print("Diamond")
plotImage(image_diamond)
```

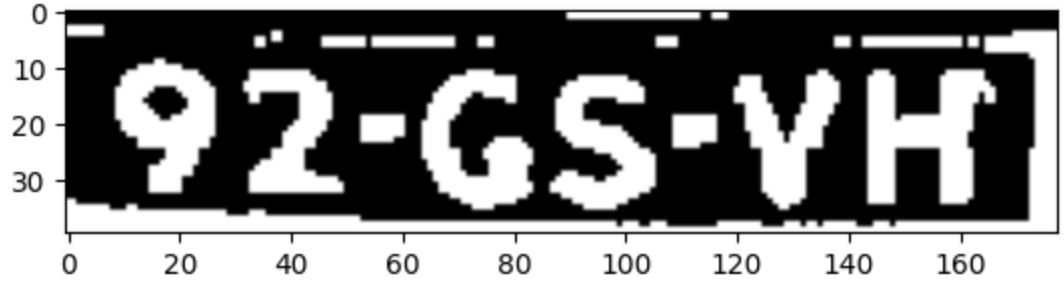
```
Cross
```



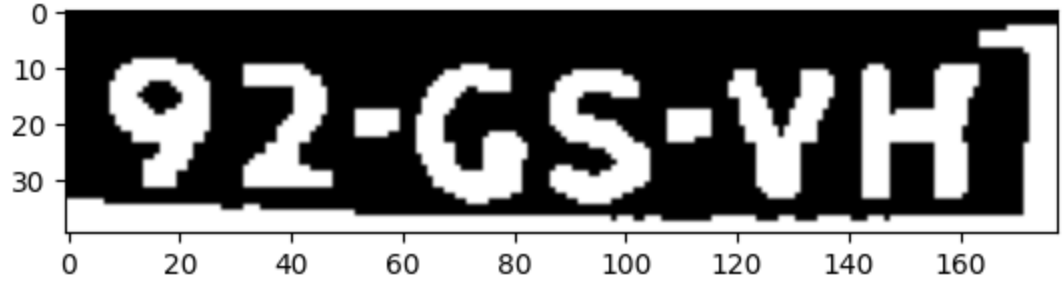

```
Cross 2
```



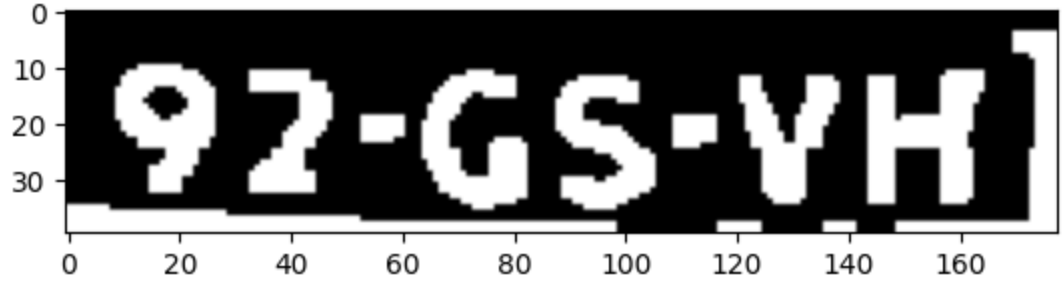
Square 2



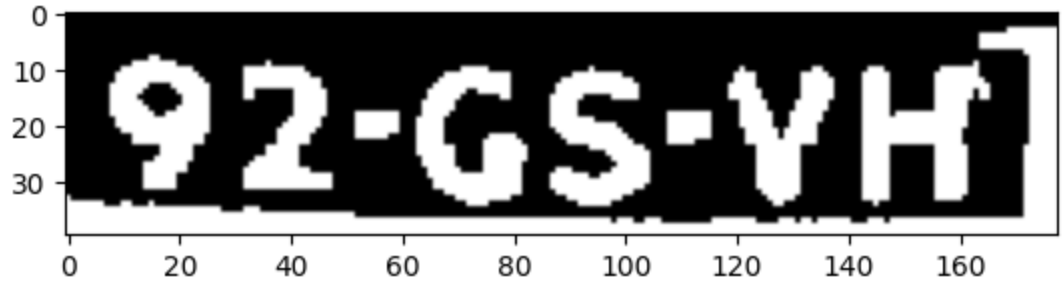
Square 3



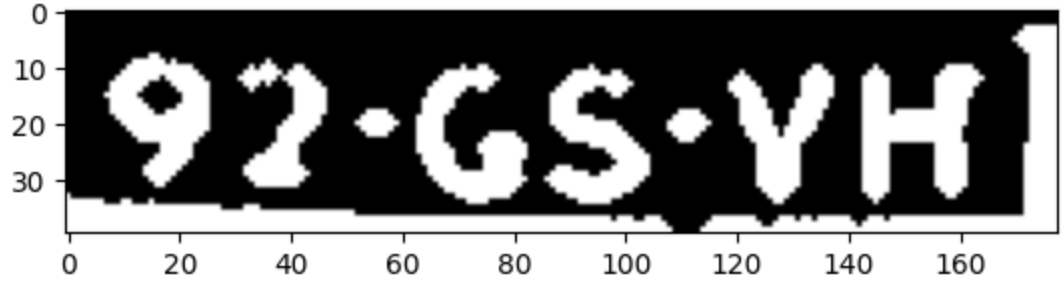
Square 4



Vertical side

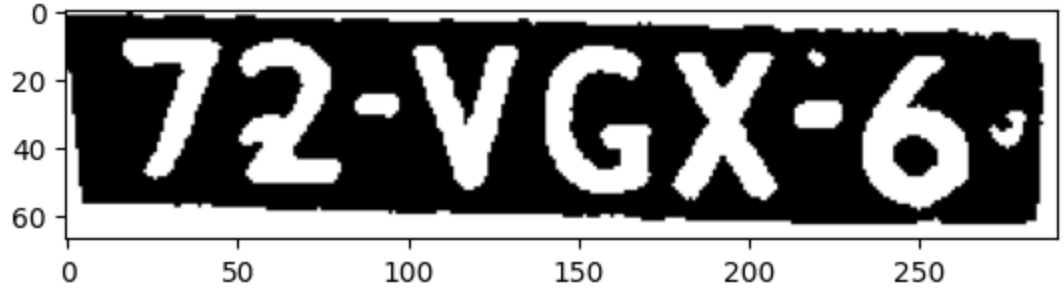


Diamond

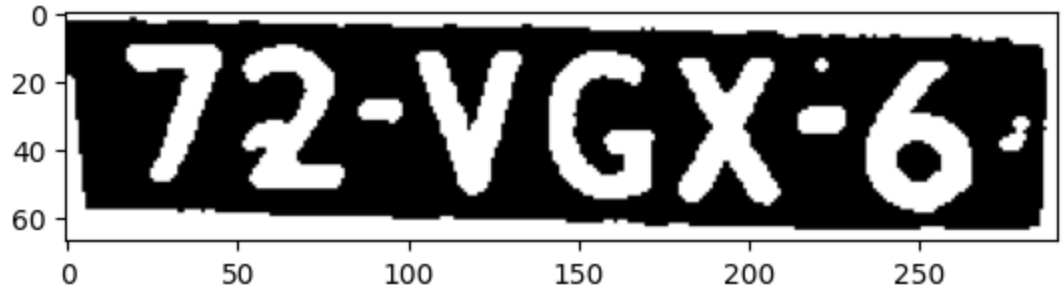



```
##################################################
#              Next Plate     2                  #
##################################################
```

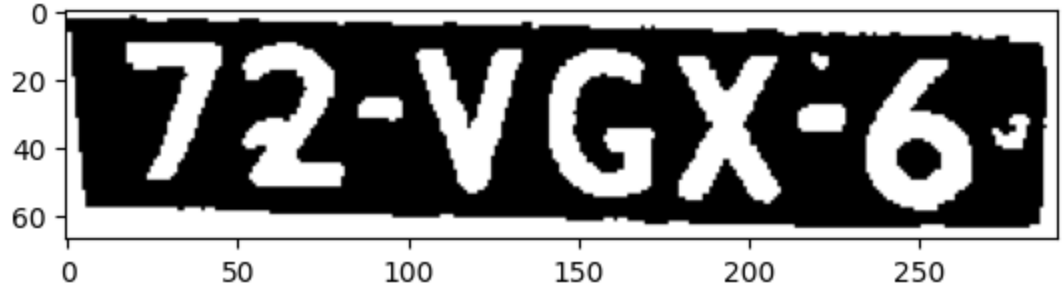
Cross



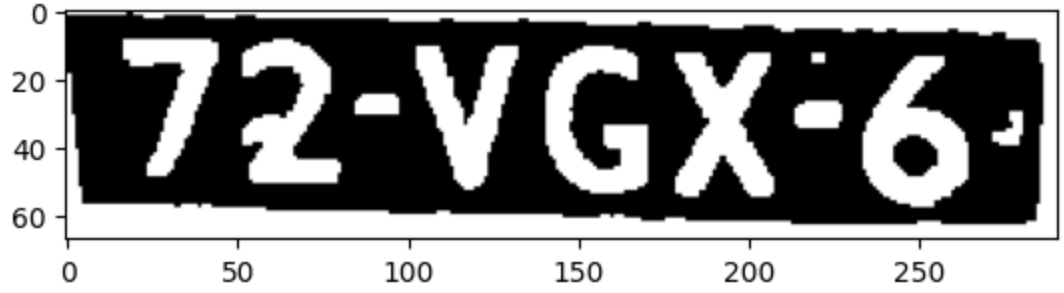
Cross 2



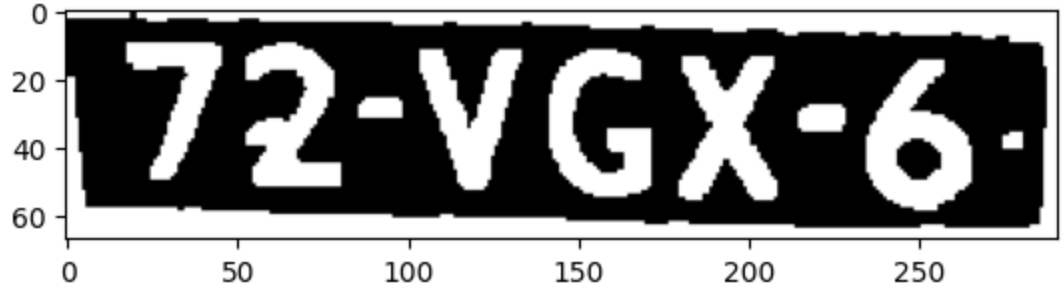
Square 2



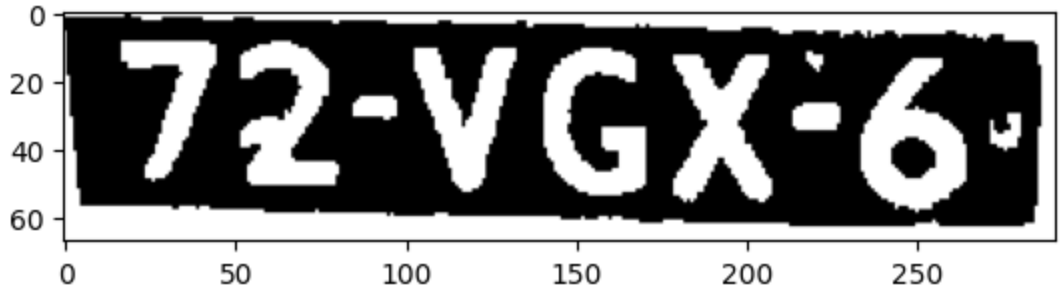
Square 3



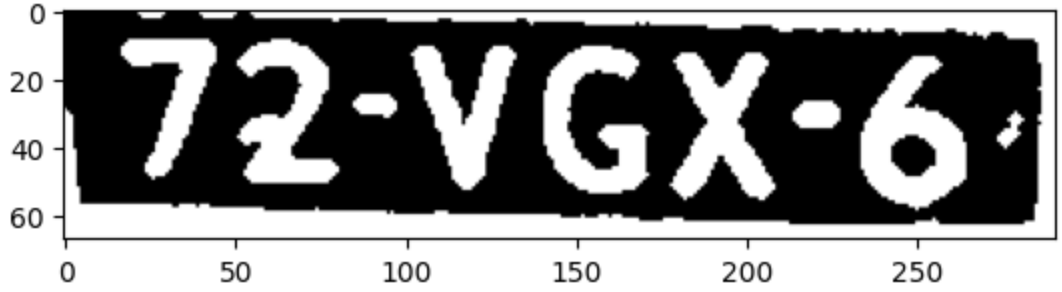
Square 4



Vertical



Diamond




```
################################################
#              Next Plate     3                #
################################################
```

Cross

Cross 2

Square 2

Square 3

Square 4

Vertical


```
Diamond
```

❓ *Could you use a single structuring element to denoise all the images? What issues could arise when doing that?*

Type your answer here. We could have used one for all, but the results wouldnt be the best possible. Some struct work better with certain cases. For example the Diamond works rlly well on plate 3, but not well ar all on the first plate. How ever we cant make a filter for each, there is no way to know which one will work best, thus we need to pick one that works well overall.

PYTHON HANDS-ON Assignment II.3: Improve the mask using morphology

Completion requirements for this assignment:

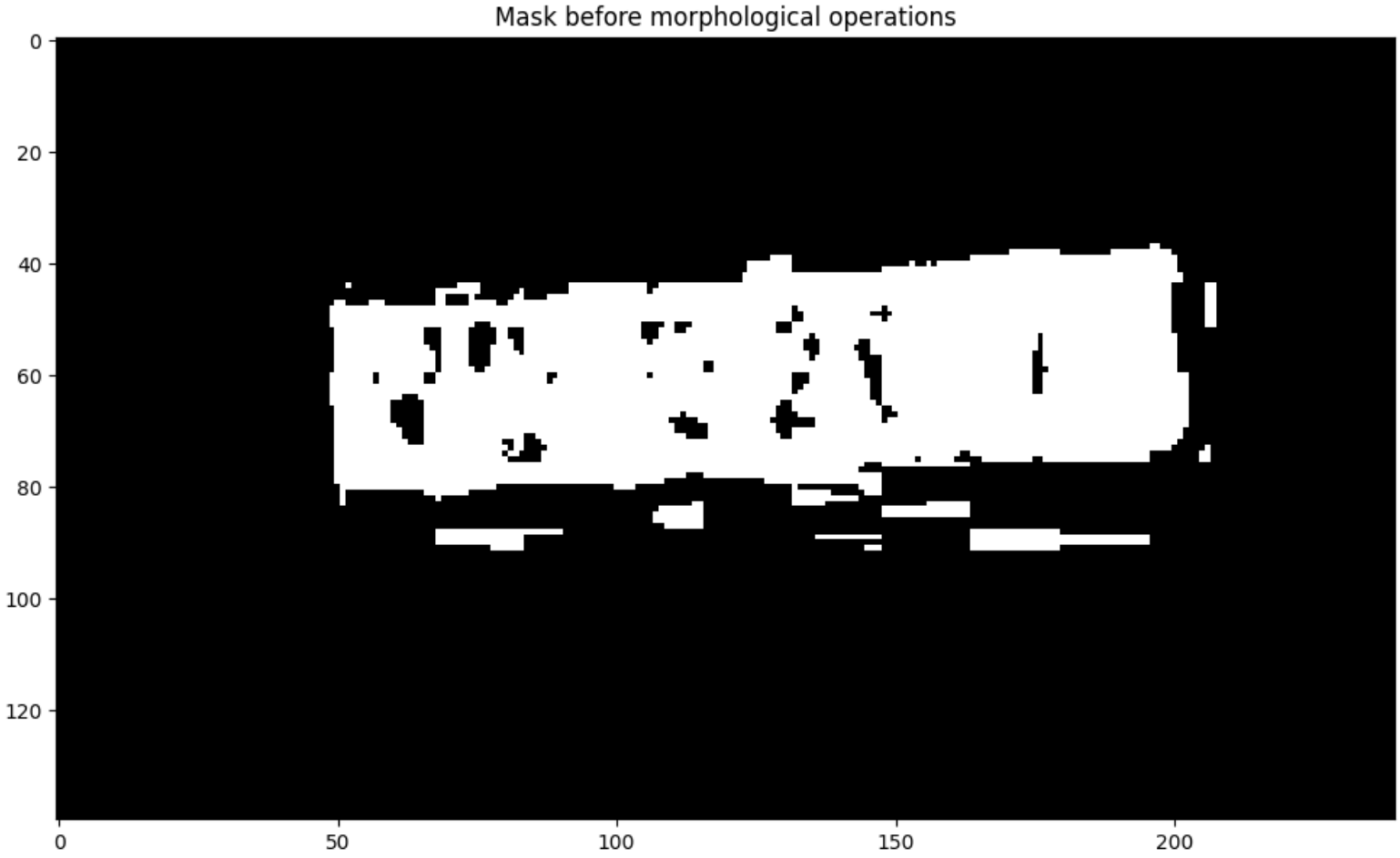
- ☐ Improve the mask such that the final masked image has no holes in the license plate and minimal noise around the license plate

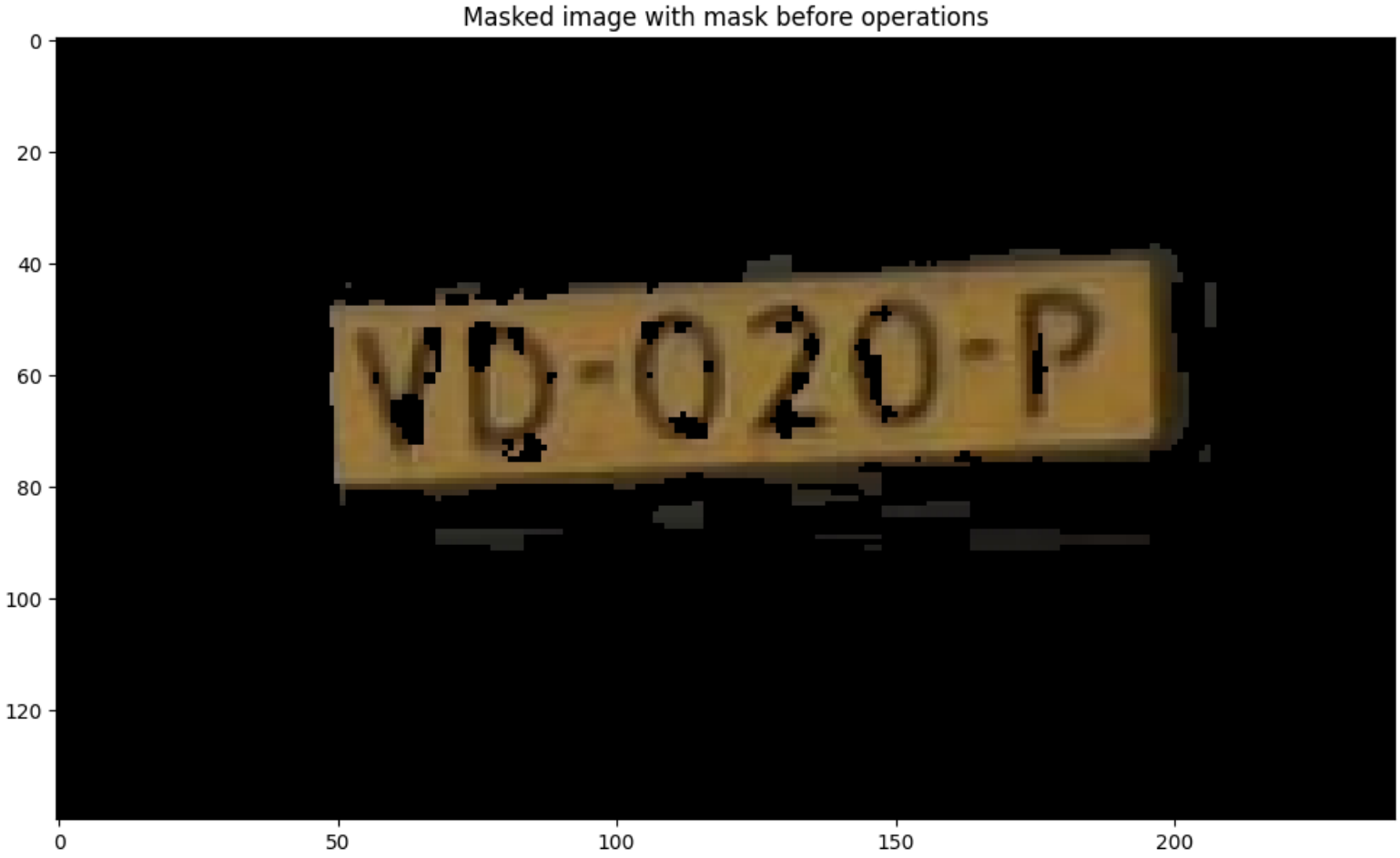
II.3.1 Improve the mask with morphology

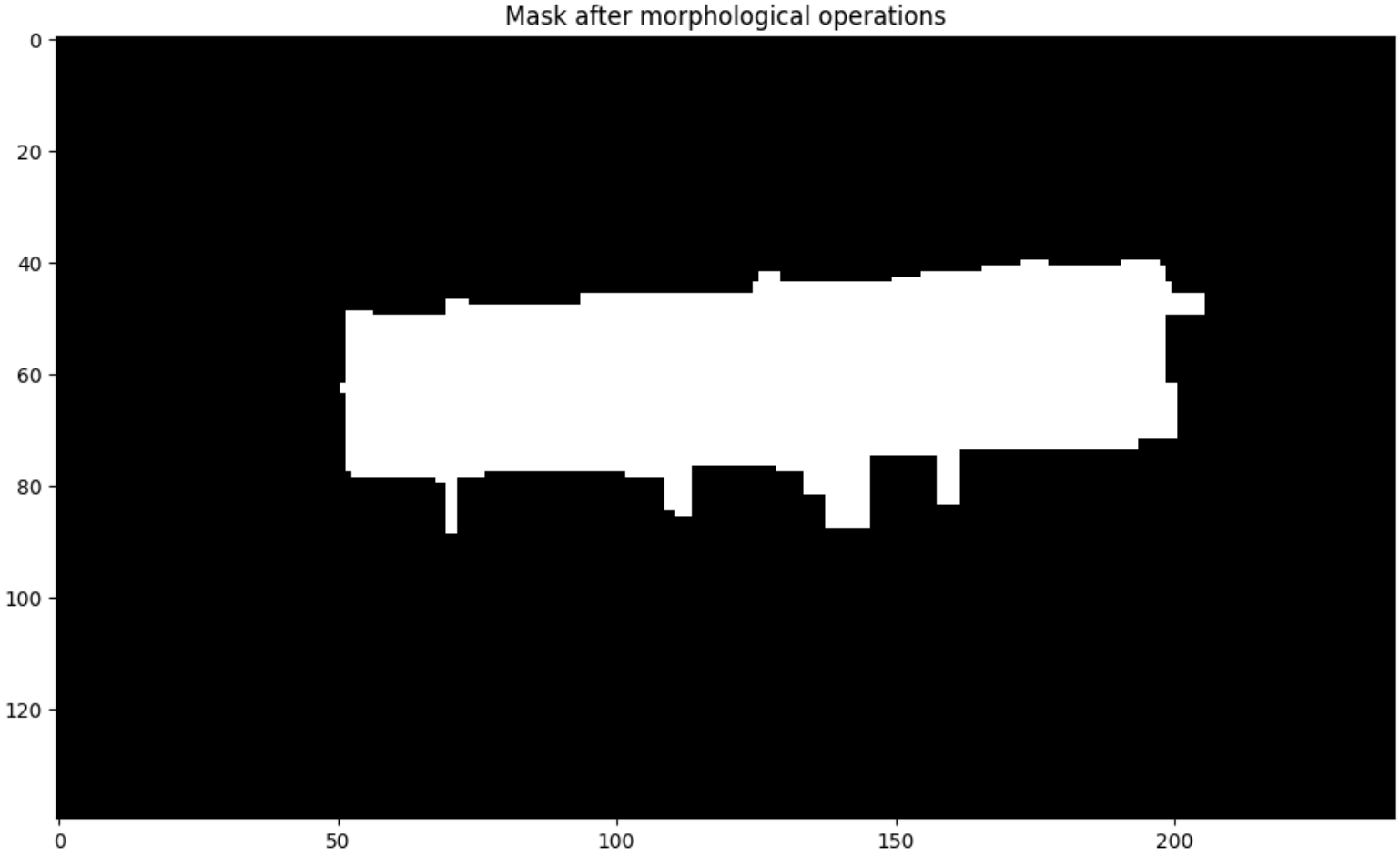
You are given a noisy mask with holes, which is used to mask a license plate in an image, similar to in lab 1. Use morphological dilation and erosion to make sure that the final masked image has no holes in the license plate, and there is minimal noise around the license plate.

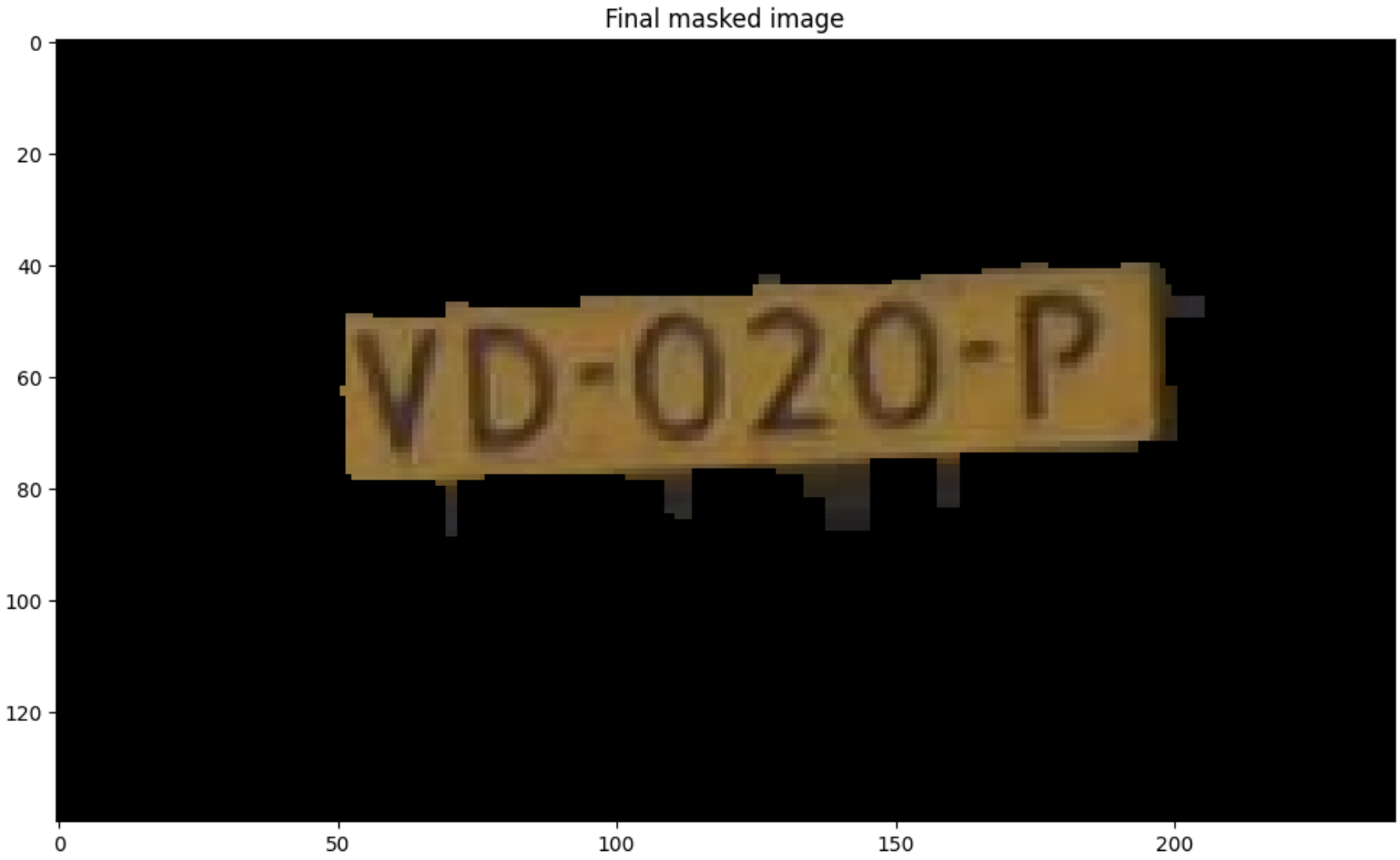
In [ ]:
```python
def improveMask(mask):

    # Improve the mask using morphological dilation and erosion
    structure = np.array([[0,1,0],[1,1,1],[0,1,0]], np.uint8)
    structure2 = np.array([[1,1,1],[1,1,1],[1,1,1]], np.uint8)
    dilated = cv2.dilate(mask, structure2)
    dilated = cv2.dilate(dilated, structure2)
    dilated = cv2.dilate(dilated, structure2)
    eroded = cv2.erode(dilated, structure2)
    eroded = cv2.erode(eroded, structure2)
    eroded = cv2.erode(eroded, structure2)
    eroded = cv2.erode(eroded, structure2)
    eroded = cv2.erode(eroded, structure2)

    #we can add more erosion but i feel like that works well for this example and cant justify the extra er
    return eroded




# ====== The code below should not be changed ======

# Load image
path = "resources/lab_02/exercise_03/"
mask = loadImage(path, "mask.png", grayscale=True)
image = loadImage(path, "0009.jpg", False)
image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

# Plot the original mask
plt.figure(figsize=(14, 7))
plt.imshow(mask, cmap="gray", vmin=0, vmax=255)
plt.title("Mask before morphological operations")
```

```
plt.show()

# Plot the image masked with the original mask
masked = -image[180:320, 300:540] * mask[:, :, np.newaxis]
plt.figure(figsize=(14,7))
plt.imshow(masked, vmin=0, vmax=255)
plt.title("Masked image with mask before operations")
plt.show()

# Improve the mask with the improveMask function
mask = improveMask(mask)

# Plot the improved mask
plt.figure(figsize=(14, 7))
plt.imshow(mask, cmap="gray", vmin=0, vmax=255)
plt.title("Mask after morphological operations")
plt.show()

# Plot the image masked with the improved mask
masked = -image[180:320, 300:540] * mask[:, :, np.newaxis]
plt.figure(figsize=(14, 7))
plt.imshow(masked, vmin=0, vmax=255)
plt.title("Final masked image")
plt.show()
```


Mask before morphological operations
0
20
40
60
80
100
120
0
50
100
150
200


Masked image with mask before operations
0
20
40
60
80
100
120
0
50
100
150
200


Mask after morphological operations
0
20
40
60
80
100
120
0
50
100
150
200



PYTHON HANDS-ON Assignment II.4: Character recognition using logical operations

Completion requirements for this assignment:

- ☐ Implement XOR comparison algorithm
- ☐ Label the given test characters using the XOR comparison algorithm based on the lowest score
- ☐ Label the given test characters using the XOR comparison algorithm based on the two lowest scores
- ☐ Answer all theory questions

II.4.1 Template matching

You have a dataset of labeled characters in the same font as in the license plate from the crime. This dataset can be used to recognise a new unlabeled image of a character. This can be done using the XOR logical operation to implement a template matching approach.

For this you're given 11 reference characters and 14 test characters (and notice that there are some test characters that are not in the reference set).

Apply the XOR comparison algorithm below.

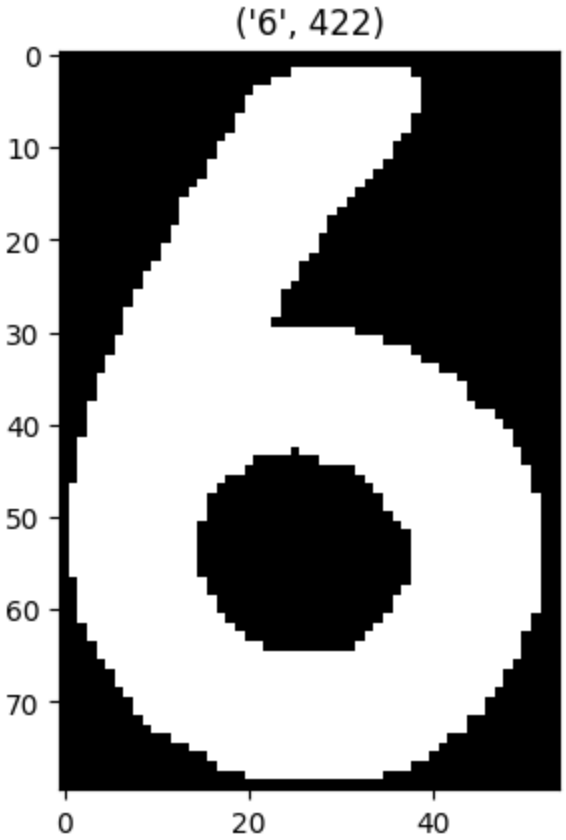
XOR Comparison Algorithm

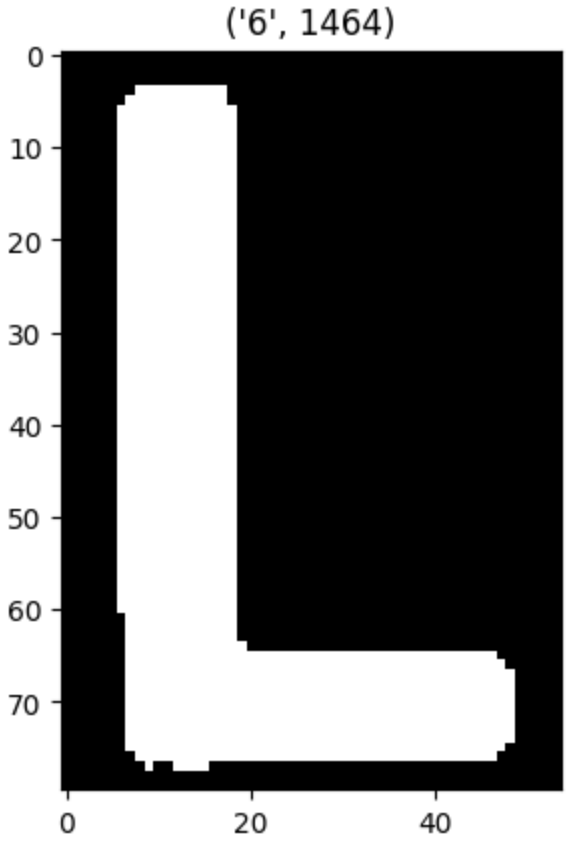
1. XOR the test image with each of the 11 reference characters
2. For each, let the difference score be the number of non-zero pixels after step 1.
3. Label the test image as the character with the lowest score from step 2.

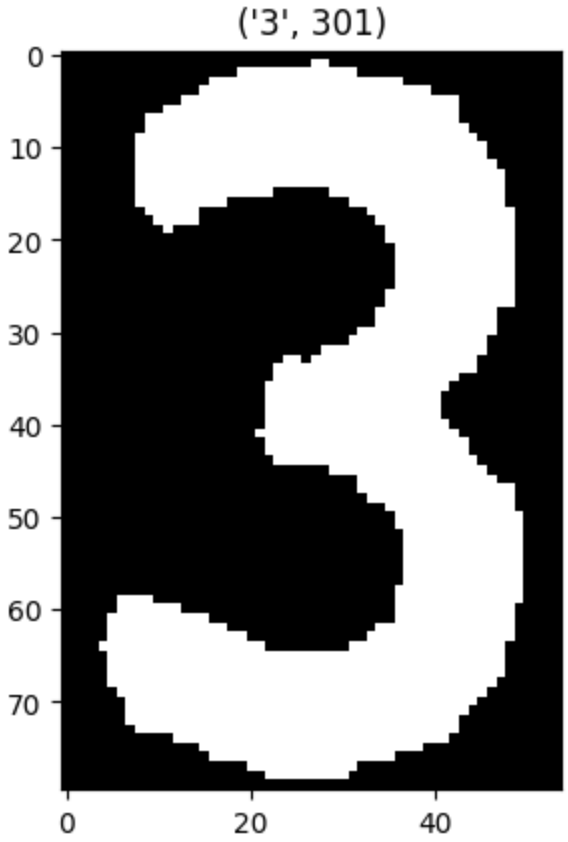
In [ ]:
```
path = "resources/lab_02/exercise_04/"
# Load the reference characters - they are named ref_X.png where X is the character in the image
character_set = set(['B','D','J','N','T','2','3','4','5','6','8'])
reference_characters = {}
for char in character_set:
    reference_characters[char] = loadImage(path, "ref_" + char + ".png")

# !The test_image and reference_character must have the same shape
def difference_score(test_image, reference_character):
    # XOR images
    reference_resized = cv2.resize(reference_character, (test_image.shape[1], test_image.shape[0]))
    xorImg = cv2.bitwise_xor(test_image, reference_character)
    # Return the number of non-zero pixels
    return np.count_nonzero(xorImg)

def give_label_lowest_score(test_image):
    # Get the difference score with each of the reference characters
    # (or only keep track of the lowest score)
```

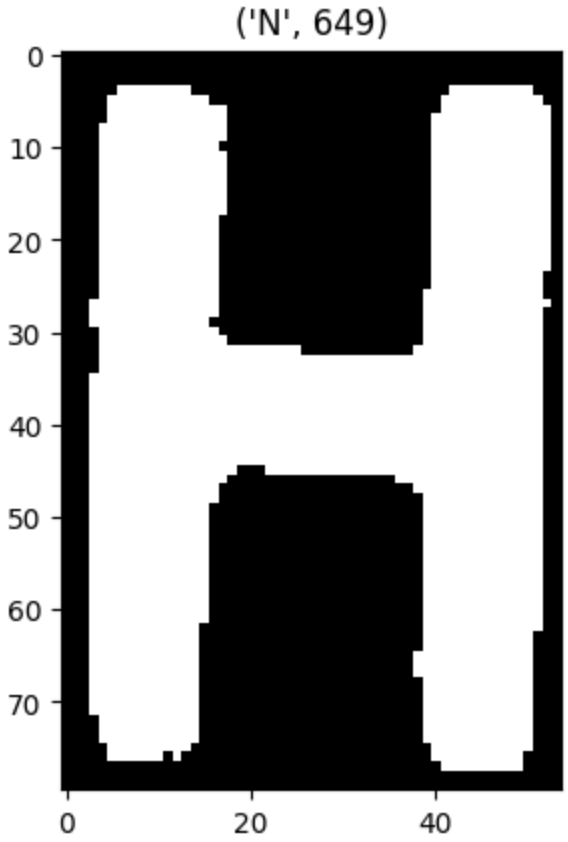
```
    scores = [(char, difference_score(test_image, reference_characters[char])) for char in reference_char
    return min(scores, key=lambda x: x[1])
```

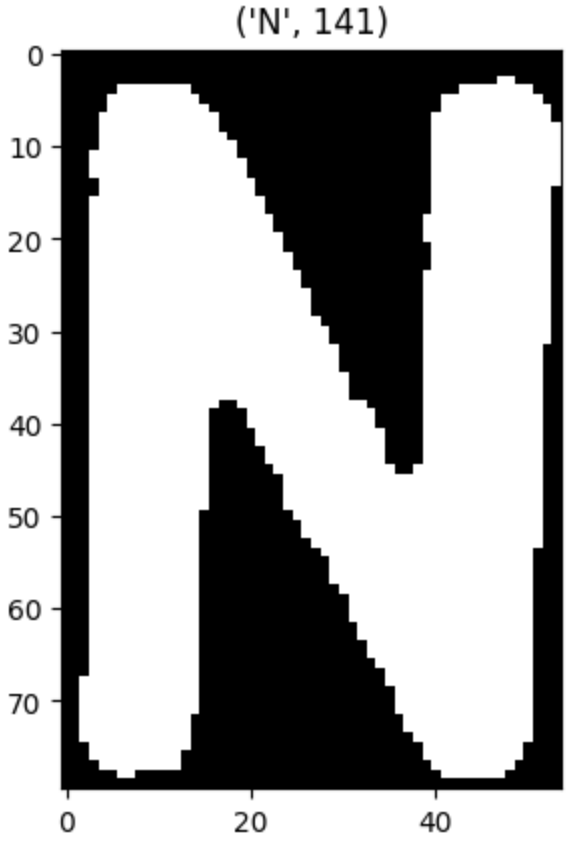
In [ ]:
```
# Load the test set - they are named test_N where N is a number
test_images = []
for i in range(1, 14):
    image_name = "test_" + ("0" if(i < 10) else "") + str(i) + ".png"
    test_images.append(loadImage(path, image_name))

# Apply your algorithm
for img in test_images:
    plotImage(img, give_label_lowest_score(img))
```

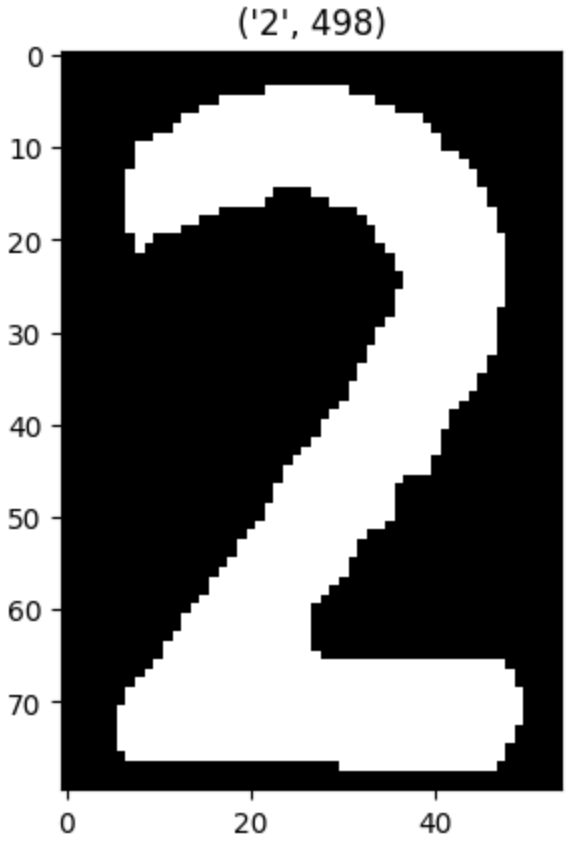


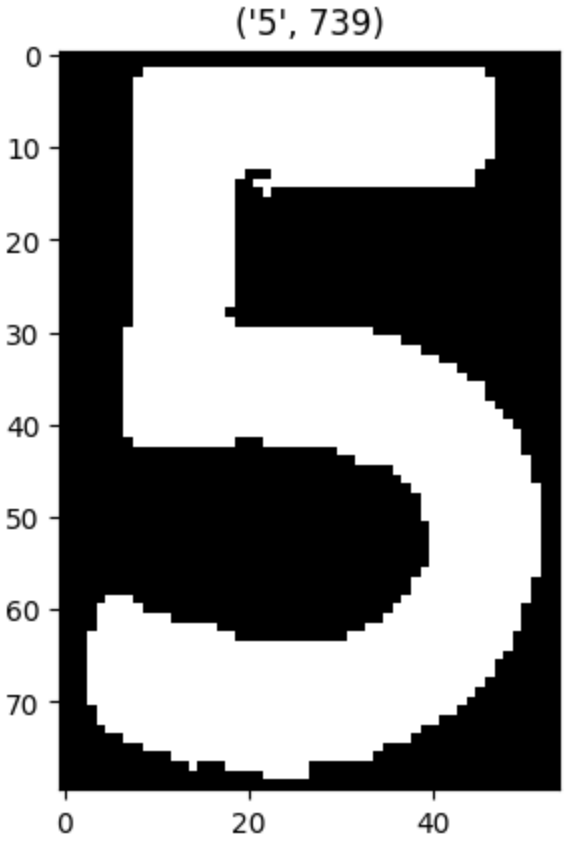

('6', 1464)
0
10
20
30
40
50
60
70
0
20
40

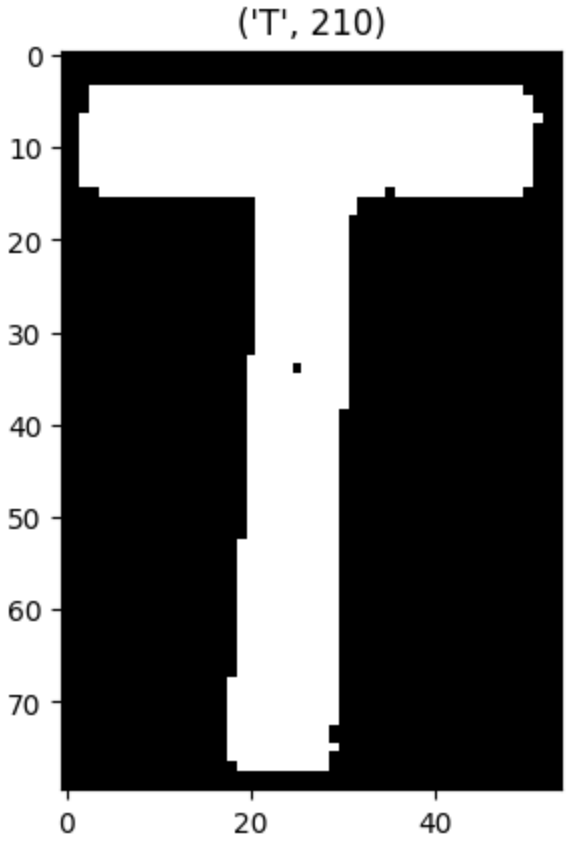

('3', 301)
0
10
20
30
40
50
60
70
0
20
40

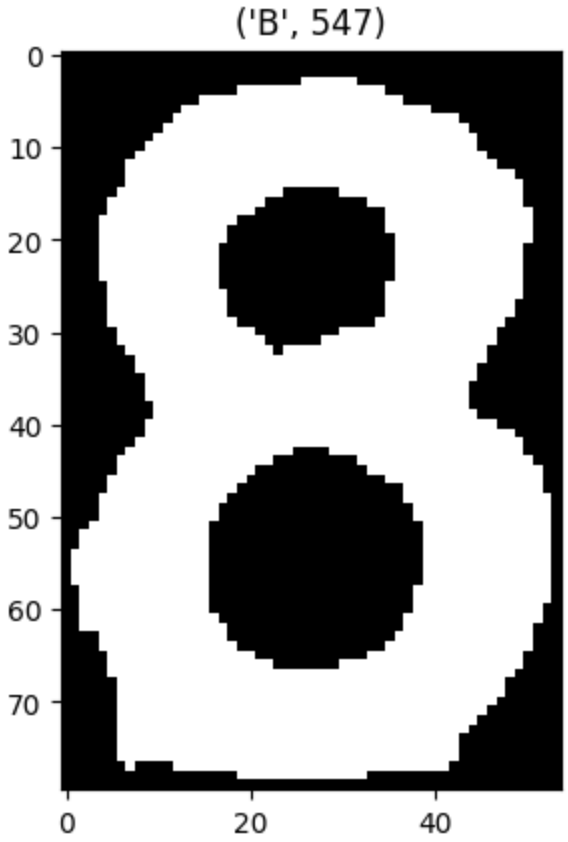

('N', 649)
0
10
20
30
40
50
60
70
0
20
40

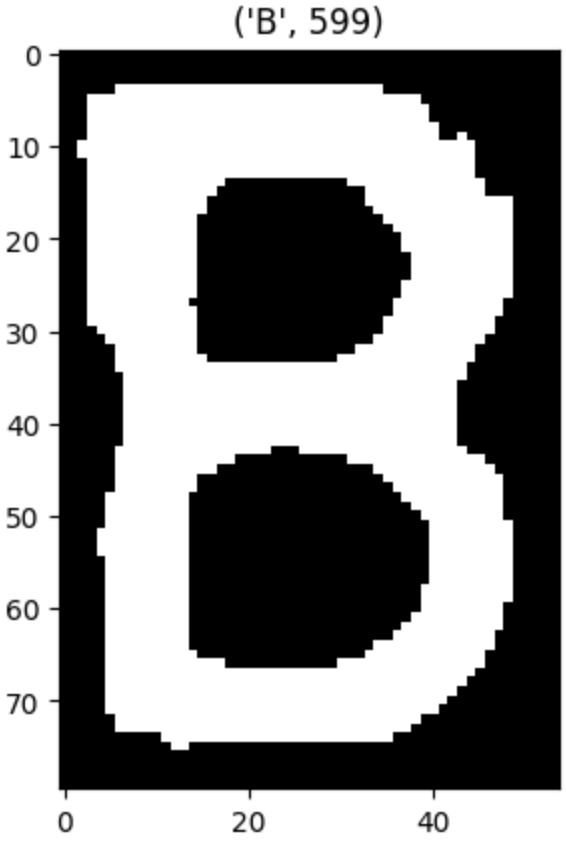

('N', 141)
0
10
20
30
40
50
60
70
0
20
40

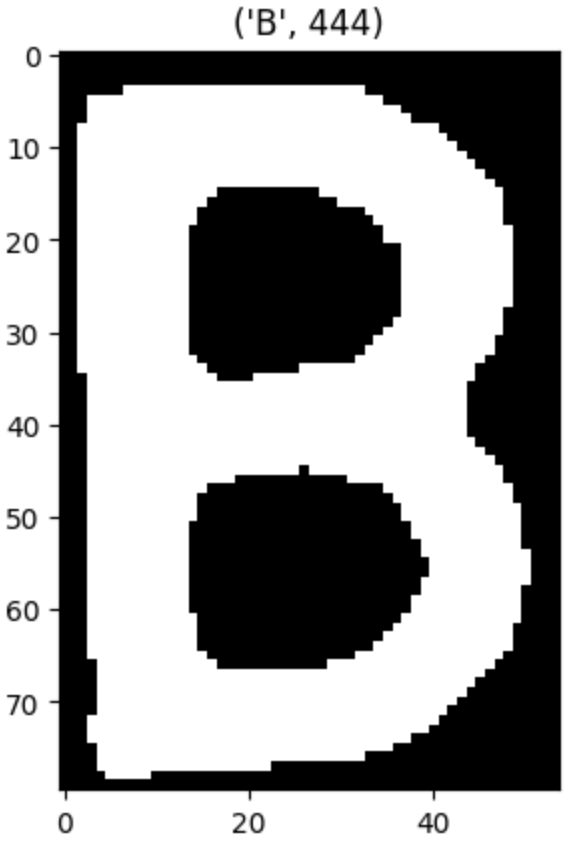

('2', 498)
0
10
20
30
40
50
60
70
0
20
40

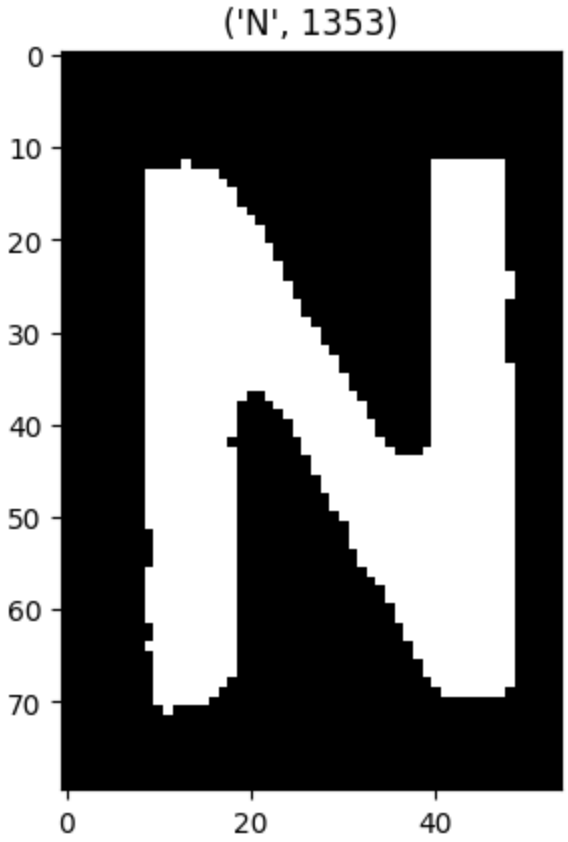

('5', 739)
0
10
20
30
40
50
60
70
0
20
40

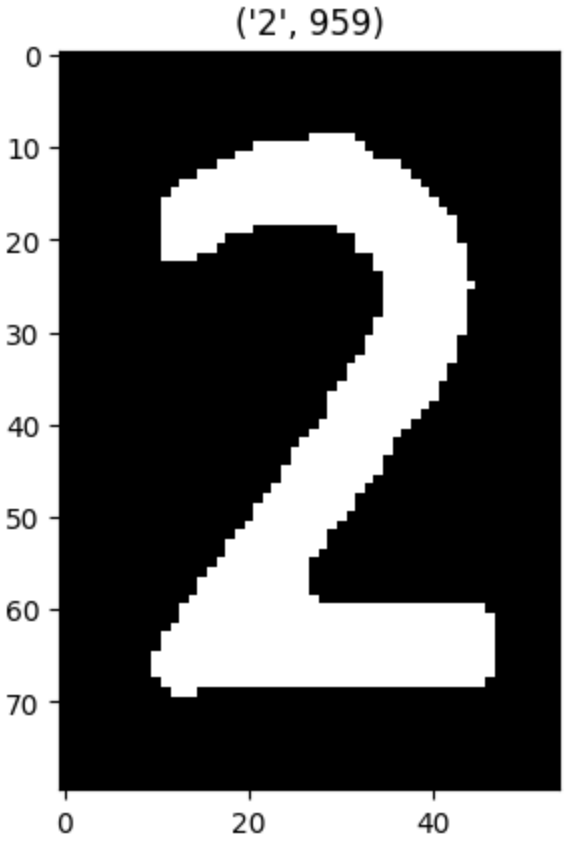

('T', 210)
0
10
20
30
40
50
60
70
0
20
40


('B', 547)
0
10
20
30
40
50
60
70
0
20
40


('B', 599)
0
10
20
30
40
50
60
70
0
20
40


('B', 444)
0
10
20
30
40
50
60
70
0
20
40


('N', 1353)
0
10
20
30
40
50
60
70
0
20
40



II.4.2 Sure or Ambiguous

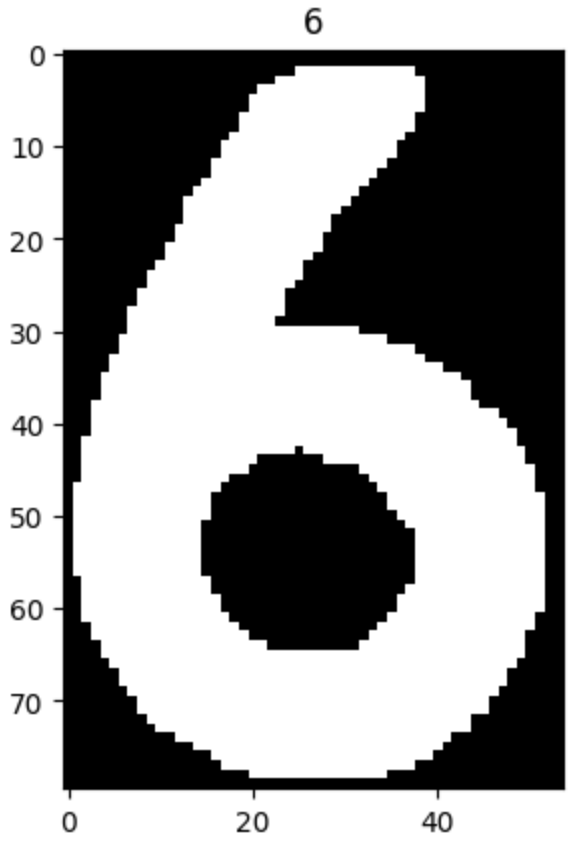
Instead of directly taking the lowest score, we could first compare it with the second lowest. In case they are very close, we cannot be certain we have classified the image correctly.

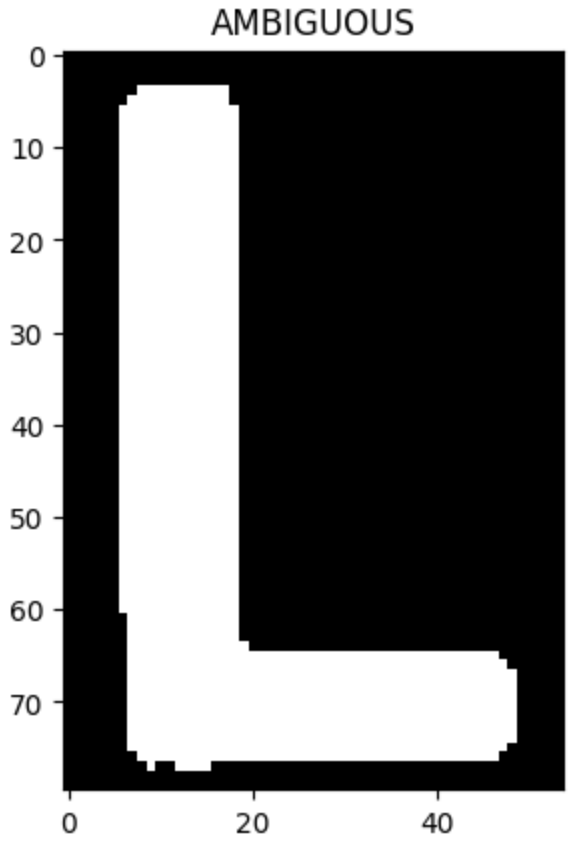
Implement the labeling function again, this time returning AMBIGUOUS_RESULT if the lowest scores are very close as values.

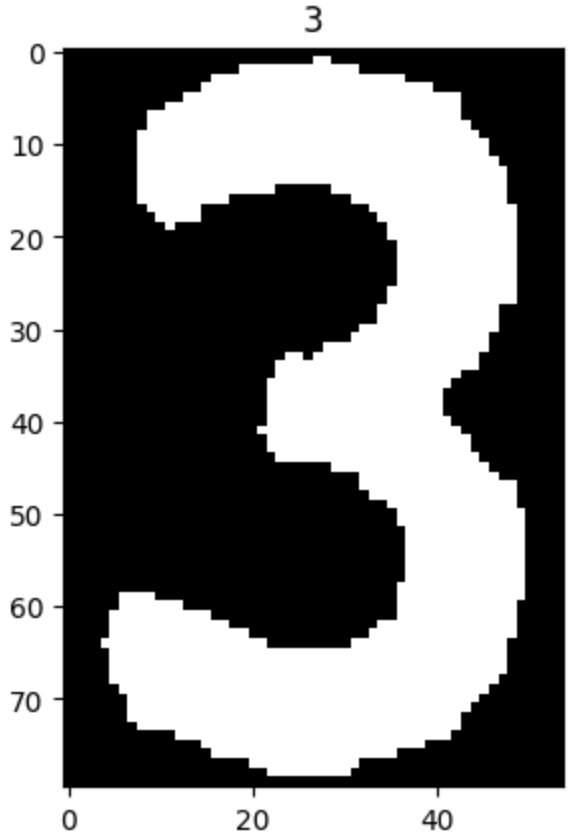
In [ ]:
```
AMBIGUOUS_RESULT = "AMBIGUOUS"
EPSILON = 0.15
def give_label_two_scores(test_image):
    # Get the difference score with each of the reference characters
    scores = [(char, difference_score(test_image, reference_characters[char])) for char in reference_cha
```

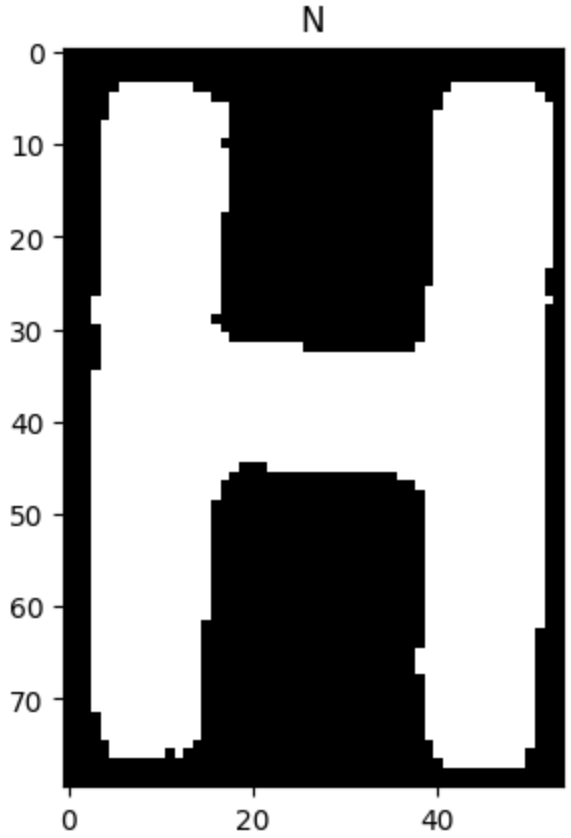
```
    scores.sort(key=lambda x: x[1])
    # Check if the ratio of the two scores is close to 1 (if so return AMBIGUOUS_RESULT)
    min1 = scores[0][1]
    min2 = scores[1][1]

    if(1-EPSILON <= min1/min2 <= 1+EPSILON):
        return AMBIGUOUS_RESULT

    # Return a single character based on the lowest score
    return scores[0][0]
```

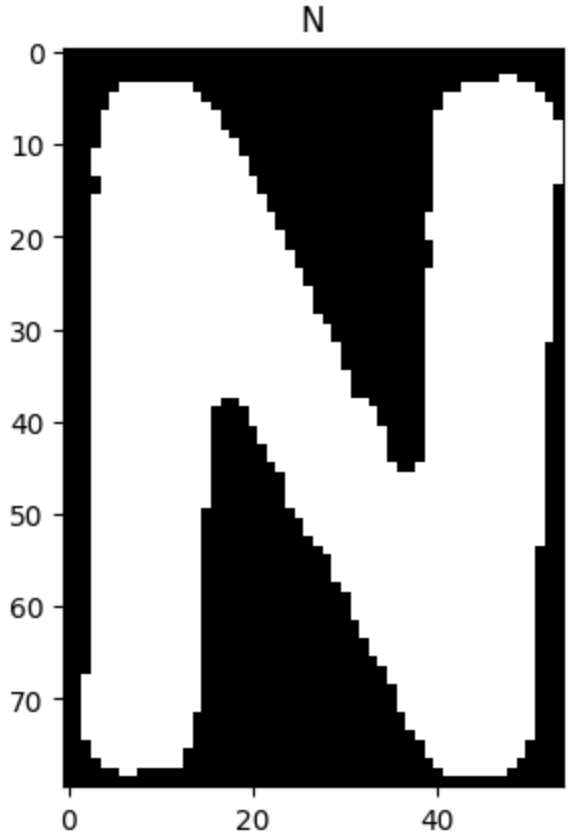
In [ ]:
```
# Apply the new function to the test images loaded earlier
for img in test_images:
    plotImage(img, give_label_two_scores(img))
```



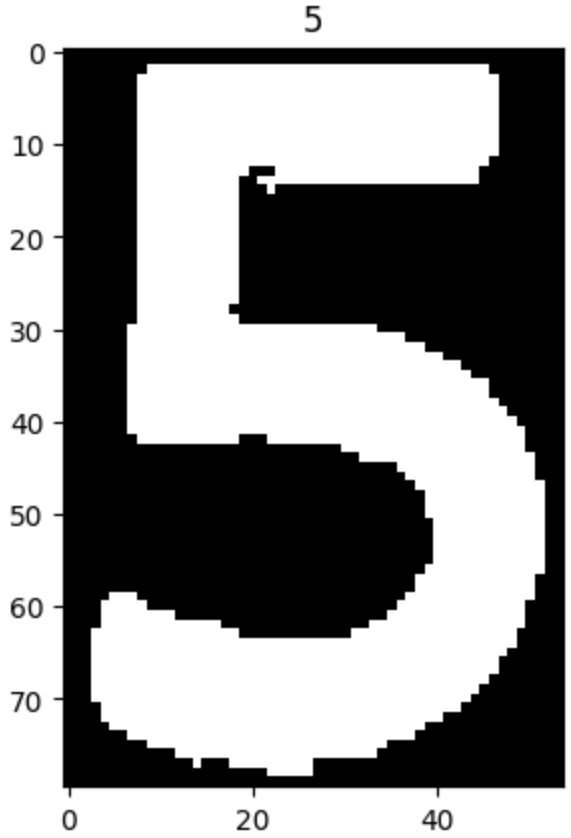

AMBIGUOUS
0
10
20
30
40
50
60
70
0
20
40

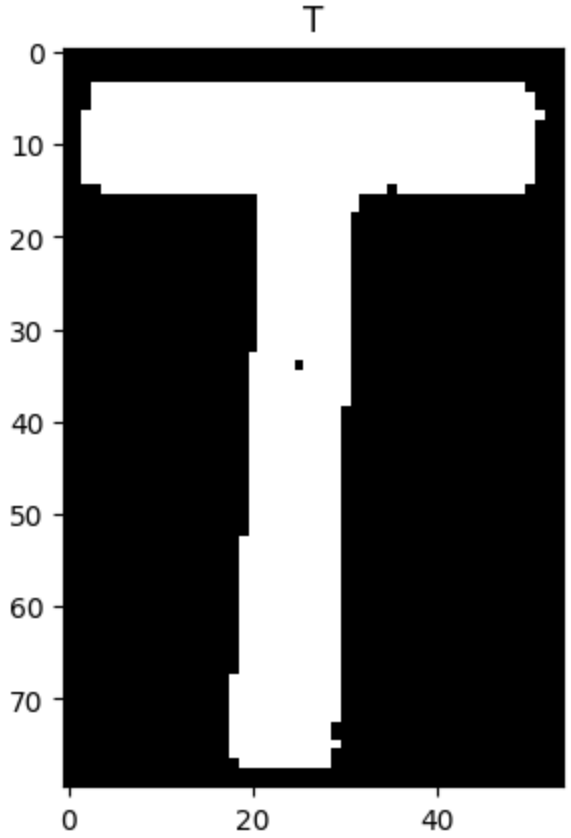

3
0
10
20
30
40
50
60
70
0
20
40

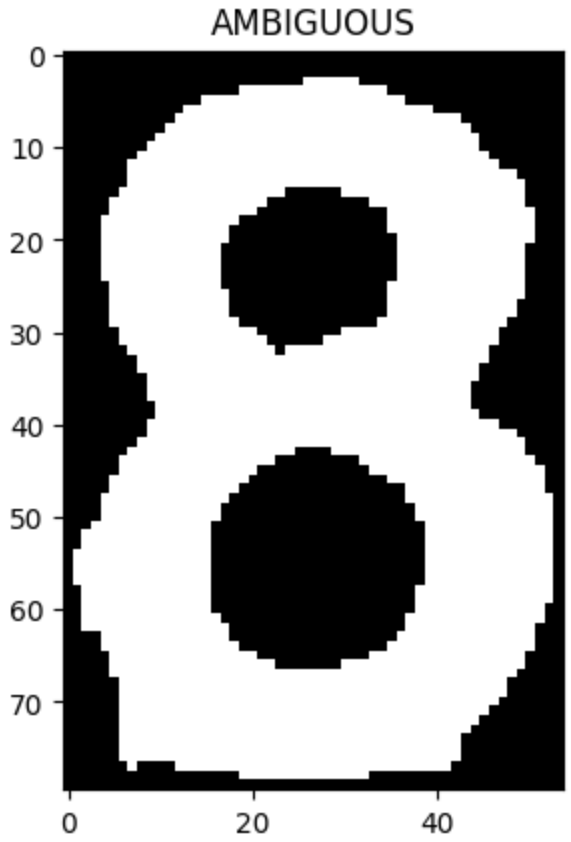

N
0
10
20
30
40
50
60
70
0
20
40

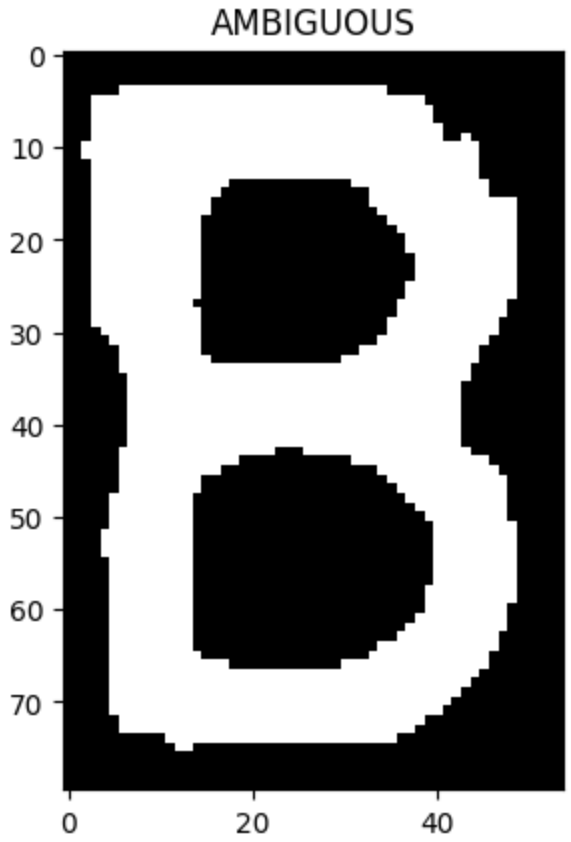

N
0
10
20
30
40
50
60
70
0
20
40

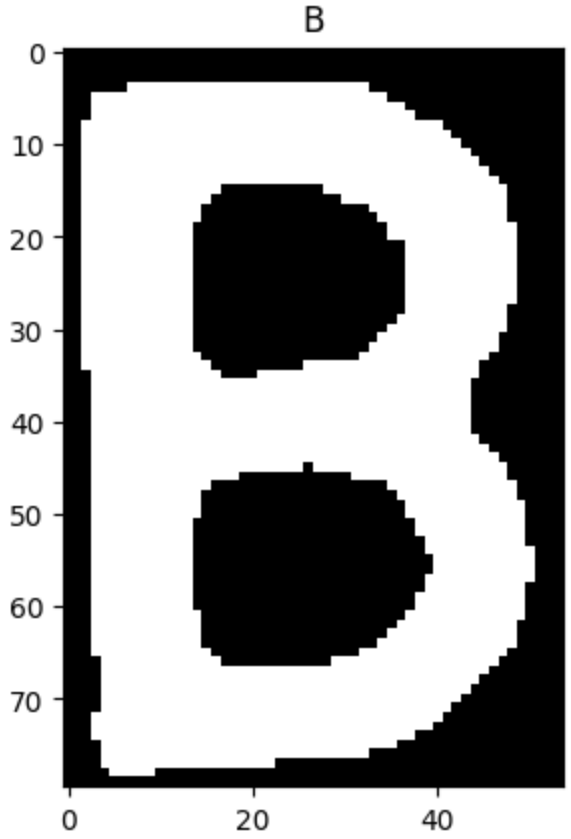

2
0
10
20
30
40
50
60
70
0
20
40

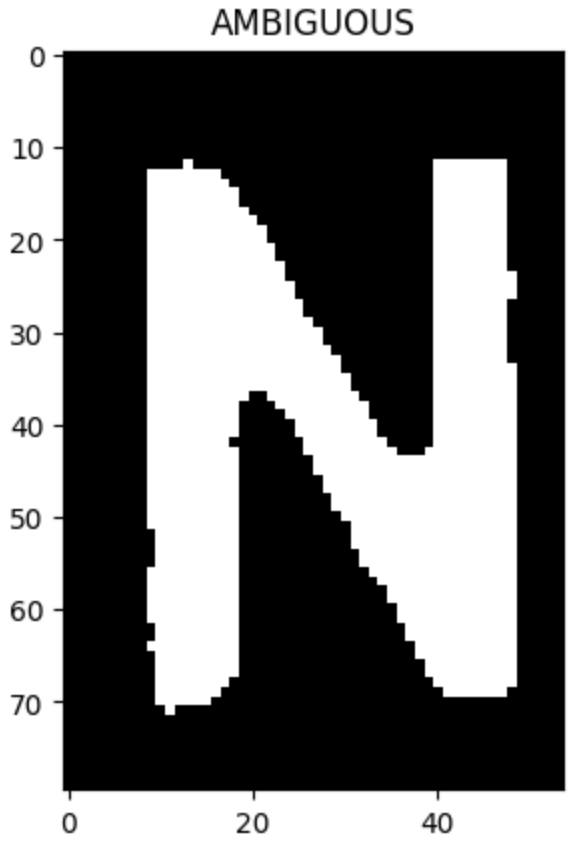

5
0
10
20
30
40
50
60
70
0
20
40

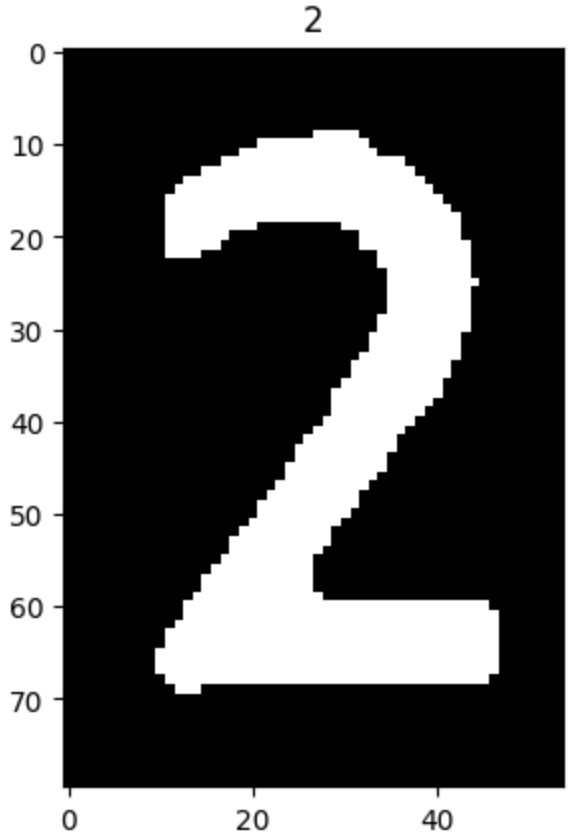

T
0
10
20
30
40
50
60
70
0
20
40


AMBIGUOUS
0
10
20
30
40
50
60
70
0
20
40


AMBIGUOUS
0
10
20
30
40
50
60
70
0
20
40


B
0
10
20
30
40
50
60
70
0
20
40


AMBIGUOUS
0
10
20
30
40
50
60
70
0
20
40



Questions Section

❓ *What did the first algorithm (for one lowest score) return for test characters that weren't in your reference set (unexpected data)?*

Type your answer here. For L i got 6 and for N i got H

❓ *Which test characters did the first algorithm (one lowest) **not** label correctly? (write the file_name, the actual_character and the given_label - for example test_00_D_0 would be an image of D that was labeled as 0)*

Type your answer here in a list.

- test_09.png is an image of an 8 that was labeled as 0
- test_02.png is an image of an L that was labeled as 6

❓ *Were all of these characters labeled AMBIGUOUS by the second algorithm? If not, suggest 1 way this can be improved*

Yes they were, both the 8 and L were both labeled ambigious

❓ *Propose at least 1 idea to enhance the algorithm's performance (to label more characters correctly) if the signature has to remain "binary image in, character label out"*

Type your answer here. A better approach would be to use a differnt classification algorithm. This one relies too much on the reference image being the same as the image characters. Another method is to add more characters in different forms and do a majority vote, aka knn. This solves the problem of only having one references per character.

PYTHON HANDS-ON Assignment II.5: Fix characters using morphology opening & closing

Completion requirements for this assignment:

- ☐ Fix the license plates by either disconnecting characters from the frame or connecting parts of the symbol that should be together

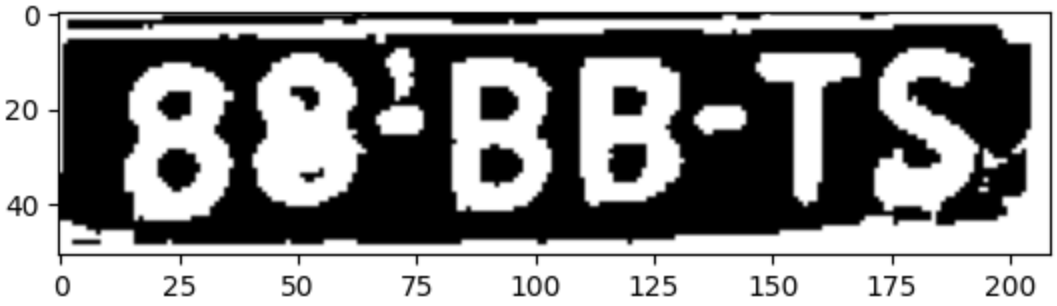
II.5.1 Enhance image quality using opening & closing

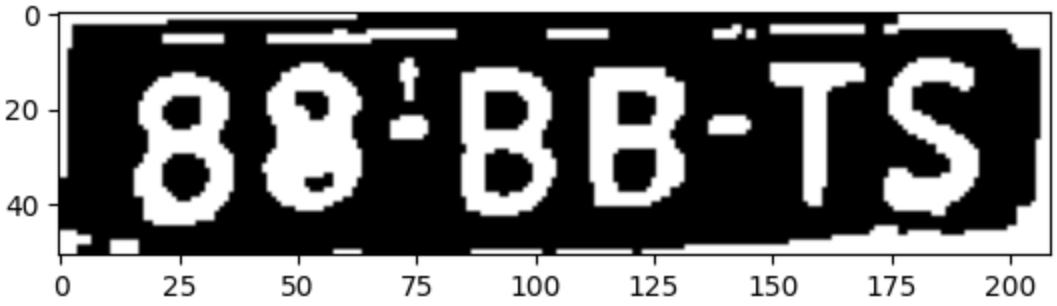
You have a set of binary images of characters that are of rather low quality. Use morphology opening & closing to improve each image.

In [ ]:
```
path = "resources/lab_02/exercise_05/"

cross = np.array([[0,1,0],[1,1,1],[0,1,0]], np.uint8)
cross2 = np.array([[0,1,1,0],[1,1,1,1],[1,1,1,1], [0,1,1,0]], np.uint8)
square2 = np.array([[1,1],[1,1]], np.uint8)
square3 = np.array([[1,1,1],[1,1,1],[1,1,1]], np.uint8)
square4 = np.array([[1,1,1,1],[1,1,1,1],[1,1,1,1], [1,1,1,1]], np.uint8)
vertical =  np.zeros((3, 3), np.uint8)
vertical[:,1] = 1
diamond = np.array([
    [0, 0, 1, 0, 0],
    [0, 1, 1, 1, 0],
    [1, 1, 1, 1, 1],
    [0, 1, 1, 1, 0],
    [0, 0, 1, 0, 0]
], dtype=np.uint8)

line = np.array([
    [0,0,0],
    [1, 1, 1],
    [0,0,0]
], dtype=np.uint8)


# Load Plate 1
plate = loadImage(path, "bin_plate1.png", grayscale=True)
plotImage(plate)

# Initialize structuring element(s)
eroded = cv2.morphologyEx(plate, cv2.MORPH_ERODE, square2)
eroded = cv2.morphologyEx(eroded, cv2.MORPH_ERODE, square2)
dilated = cv2.morphologyEx(eroded, cv2.MORPH_DILATE, square2)
#opened = cv2.morphologyEx(plate, cv2.MORPH_OPEN, square3)
# Show result
plotImage(dilated)

print("""
        #########
```

```
        ########
        ########
        ########
        ########
        ########""")


# Load Plate 2
plate2 = loadImage(path, "bin_plate2.png", grayscale=True)
plotImage(plate2)

# Apply morphological operations
dilated = cv2.morphologyEx(plate2, cv2.MORPH_DILATE, line)
dilated = cv2.morphologyEx(dilated, cv2.MORPH_DILATE, line)
closed = cv2.morphologyEx(dilated, cv2.MORPH_CLOSE, line)
eroded = cv2.morphologyEx(closed, cv2.MORPH_ERODE, line)
eroded = cv2.morphologyEx(closed, cv2.MORPH_ERODE, line)
plotImage(eroded)

# Show result




print("""
        #########
        ########
        ########
        ########
        ########
        ########""")


# Load Plate 3
plate3 = loadImage(path, "bin_plate3.png", grayscale=True)
plotImage(plate3)
# Initialize structuring element(s)
closed = cv2.morphologyEx(plate3, cv2.MORPH_CLOSE, square3)
dilated = cv2.morphologyEx(closed, cv2.MORPH_DILATE, square3)
# Apply morphological operations
# Show result
```

```
plotImage(dilated)


print("""
        #########
        ########
        ########
        ########
        ########
        ########""")

# Load Plate 4
plate4 = loadImage(path, "bin_plate4.png", grayscale=True)
plotImage(plate4)
# Initialize structuring element(s)
close = cv2.morphologyEx(plate4, cv2.MORPH_CLOSE, line)
# Apply morphological operations
plotImage(close)

# Show result
```



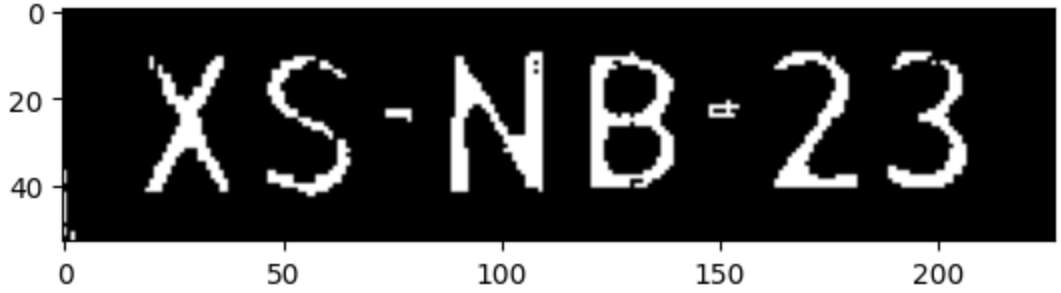


```
########
########
########
########
########
########
```

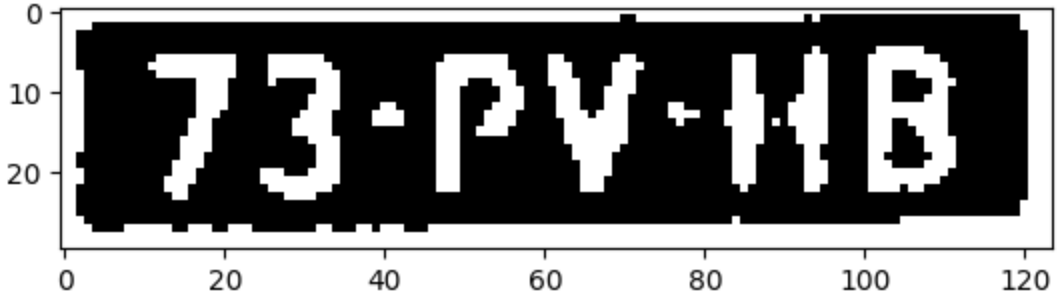

67-FP-SJ

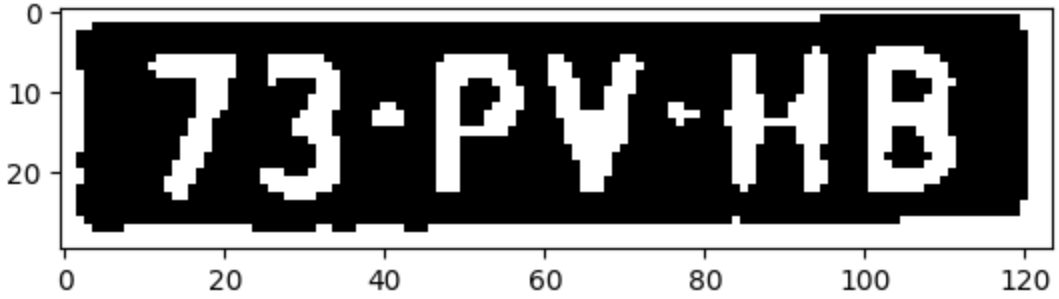

67-FP-SJ

```
########
########
########
########
########
########
```


XS-NB-23


```
#########
########
########
########
########
########
```





Checklist before submitting

(try to fix anything that you can't tick from the checkboxes below)

In [ ]:
```
# !pip install ipywidgets

import ipywidgets as widgets

check1 = widgets.Checkbox(
    value=False,
    description='Filled in all of the yellow fields with answers to the questions marked with ❓'
    indent=False,
    layout={'width':'1000px'}
)
check2 = widgets.Checkbox(
    value=False,
    description='Haven\'t used imports outside of those provided in the original notebook for thi
    indent=False,
    layout={'width':'1000px'}
)
check3 = widgets.Checkbox(
    value=False,
    description='All the code blocks can be run in sequence and execute successfully',
    indent=False,
    layout={'width':'1000px'}
)
check4 = widgets.Checkbox(
    value=False,
    description='Haven\'t changed the layout or formatting in the notebook and have only written
    indent=False,
    layout={'width':'1000px'}
)
display(check1, check2, check3, check4)
```

```
Checkbox(value=False, description='Filled in all of the yellow fields with answers to the question
s marked wit…
Checkbox(value=False, description="Haven't used imports outside of those provided in the original
notebook for…
Checkbox(value=False, description='All the code blocks can be run in sequence and execute successf
ully', inden…
Checkbox(value=False, description="Haven't changed the layout or formatting in the notebook and ha
ve only writ…
```

In [ ]: