CAVA-TEMA1

Huțan Mihai-Alexandru Gr.343

November 2023

Contents

1 Introduction

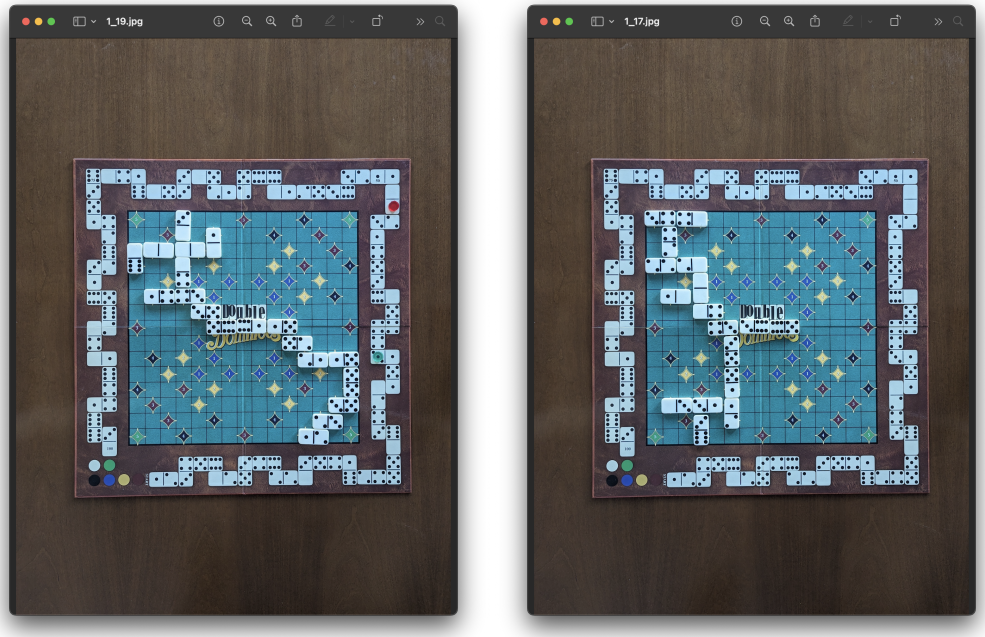
This homework comes down to only three steps, detection, classification and score calculation, all these steps are explained in depth in the following sections.

2 Detection

2.1 Board Extraction

2.1.1 Approach

For detection part we can see two major steps that we need to do before continuing solving the problem. We have a lot of unnecessary data, the table that the board is placed on, and the score board.



(a) Train Image (b) Test Image

Figure 1: Examples

The score board is unnecessary because in the test image the players are not moved anymore, therefore we can not use that part of the board to calculate the score in section 4.

2.1.2 Steps

All parameters values presented in the following steps were found by trial and error, and visualization logic while debugging.

All these steps can be visualized in Figure 2.

- HSV Mask
  - For Board Extraction the main idea is to use an HSV mask to remove parts of the image that are not of interest.
  - The HSV values found are (80, 0, 0) and (102, 255, 255).

```
LOWERB_BOARD = np.array([80, 0, 0])
UPPERB_BOARD = np.array([102, 255,
    255])
```

- Preprocessing
  - In the image of the mask, there is quite a bit of noise. Therefore we reduce it using both median and gaussian blur to get a sharpened version of the original image.

```
image_m_blur = cv.medianBlur(mask, 5)
image_g_blur =
    cv.GaussianBlur(image_m_blur, (3,
    3), 5)
image_sharpened =
    cv.addWeighted(image_m_blur, 1.5,
    image_g_blur, -0.8, 0)
```

  - We threshold the image intro a binary image.

```
_, thresh = cv.threshold(image_sharpened, 151, 255,
```

  - We use morphological operations to erode and then dilate the image.

```
kernel = np.ones((3, 3), np.uint8)
thresh = cv.erode(thresh, kernel,
    iterations=4)
thresh = cv.dilate(thresh, kernel,
    iterations=3)
```

- Edge Detection
  - We use Canny for edge detection on the preprocessed image.

```
edges = cv.Canny(image, 200, 400,
    apertureSize=7)
```

- Contours
  - We find the contours.

```
contours, _ = cv.findContours(edges,
    cv.RETR_EXTERNAL,
    cv.CHAIN_APPROX_SIMPLE)
```

- Find Extreme Points In Contours
  - We use a function to find the topleft, topright, bottomleft, and bottomright points in the image.
- Warp
  - We use warpPerspective from opencv to get our board from the image and resize it to 1500x1500px, each box in the board being 100x100px.

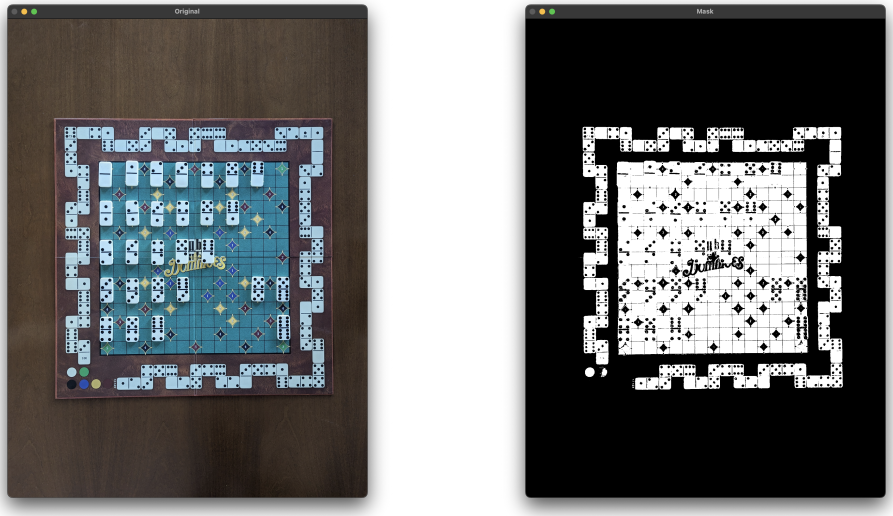
```
width = BOARD_WIDTH * BOX_SIZE  # 15
    * 100
height = BOARD_HEIGHT * BOX_SIZE  #
    15 * 100

puzzle = np.array([top_left,
    top_right, bottom_right,
    bottom_left], dtype="float32")
destination_of_puzzle = np.array([[0,
    0], [width, 0], [width, height],
    [0, height]], dtype="float32")

M =
    cv.getPerspectiveTransform(puzzle,
    destination_of_puzzle)
```

```
result = cv.warpPerspective(image, M,
    (width, height))
```

Figure 2: Board Extraction Steps

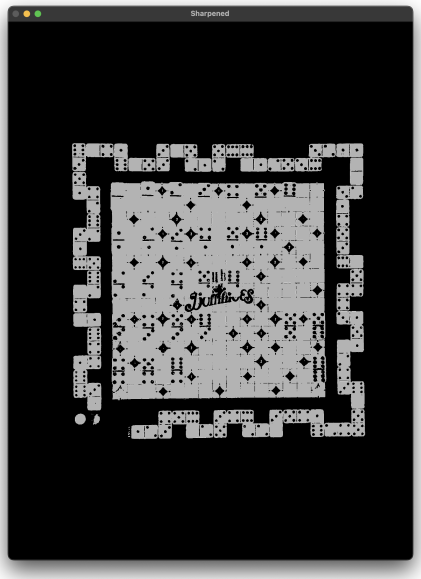


(a) Original Image

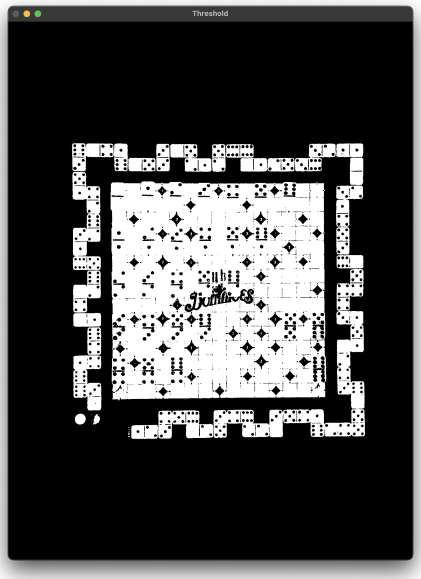
(b) HSV Mask

2.2 Piece Detection

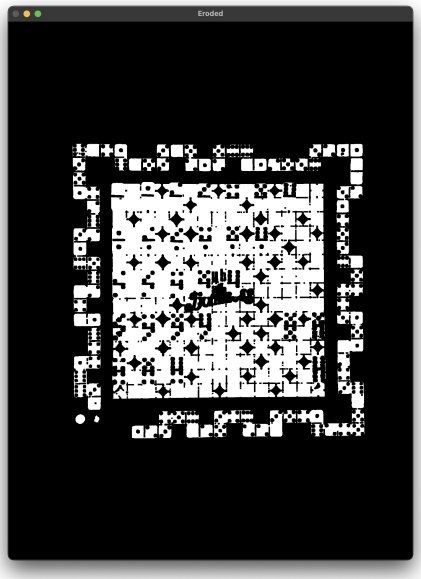
- The core idea is to use the same mask as before and get each patch at a time, calculate the mean value of the pixels and use a threshold to detect if there is a piece of domino or not.
- We've faced two main problems here, the center of the board appears in the mask, (the text of Double Double Dominoes is white therefore appears in the mask) and some pieces are not placed correctly, therefore the black pixels from the board appear in the image influencing the mean.
- The center text problem is solved by using template matching. I've created templates from the auxiliary images with each patch of the center text, and if they match we ignore that when detecting. The created templates can be visualized in Figure 3.



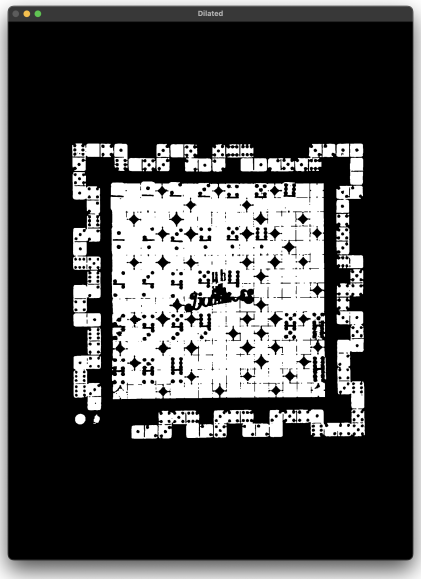
(c) Sharpened



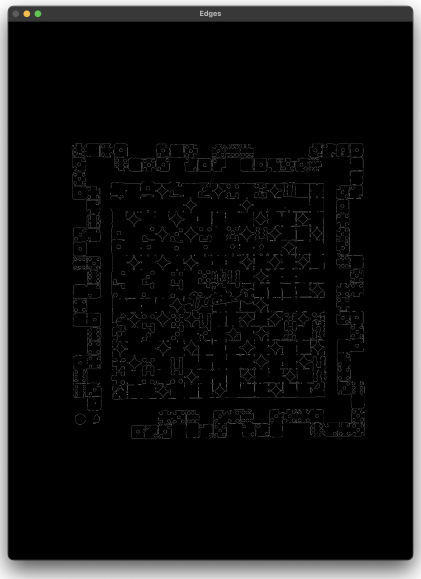
(d) Threshold



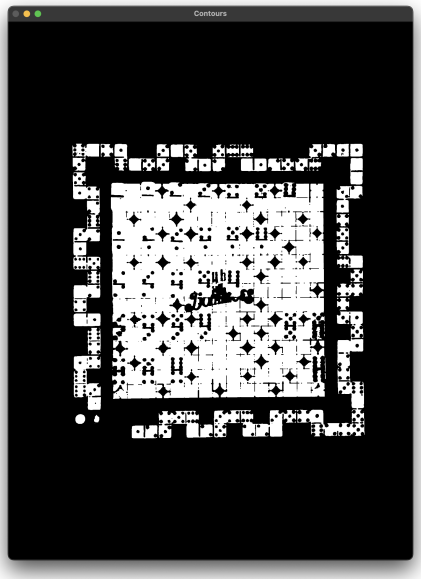
(e) Eroded



(f) Dilated

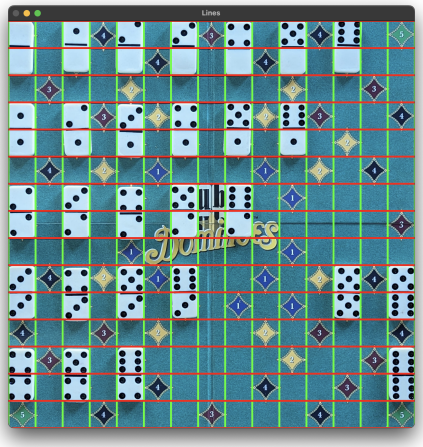


(g) Canny Edges

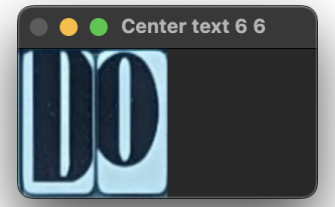


(h) Contours

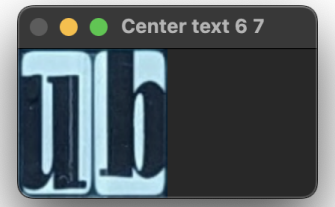
(i) Extreme Points



(j) Lines



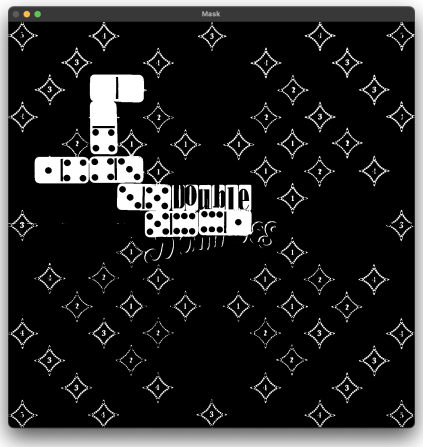
(a) Center Text Template Example 1



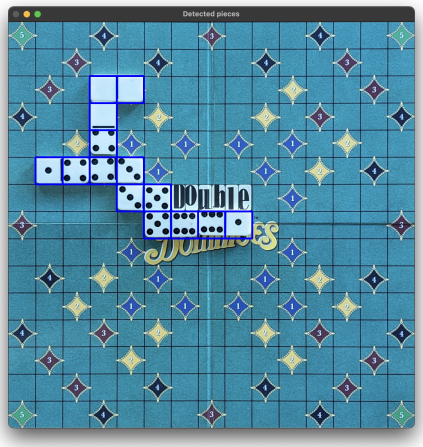
(b) Center Text Template Example 2

Figure 3: Center Text Template Creation

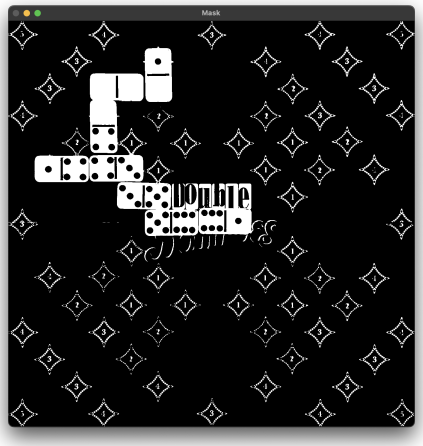
- The bad placed dominoes problem I've solved by not using the entire patch, but removing 15 pixels from each side, remaining with a 70x70px patch.
- The way we create the mask and use it in our algorithm to calculate mean and detect pieces can be visualized in Figure 4



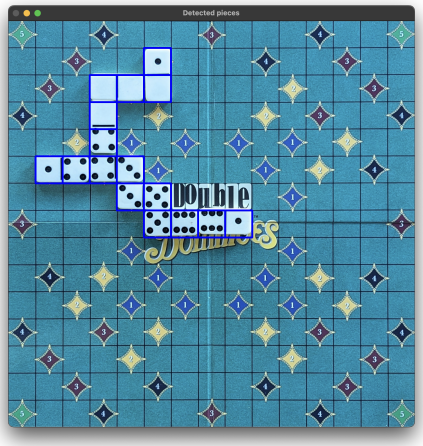
(a) Piece Detection Mask 1



(b) Piece Detection Visualization 1



(c) Piece Detection Mask 2



(d) Piece Detection Visualization 2

Figure 4: Piece Detection Steps

3 Classification

3.1 Templates Creation

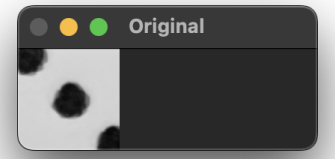
For template creation I used a programatic approach, to remove human error. I hardcoded the places on both the vertical and horizontal auxiliary images.

Using the board extraction function explained in section 2.1. I parse the board and whenever I find a place of interest, I know the patch true value, and add it in a list corresponding to its value. (I create templates for both vertical and horizontal placed pieces since domino piece number 3 for example is different depending on the direction it is placed).

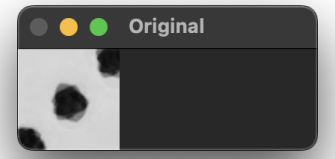
After saving multiple templates for each domino piece both vertically, and horizontally, I generate the mean image for each direction and value, and save them in a folder for future use.

Some examples of the generated templates can be visualized in Figure 5.

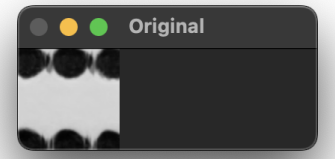
Figure 5: Templates Creation



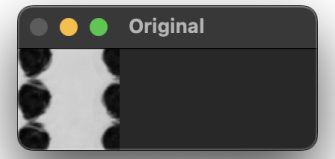
(a) Horizontal 3



(b) Vertical 3



(c) Horizontal 6



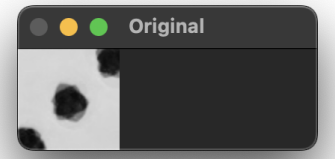
(d) Vertical 6

3.2 Template Matching

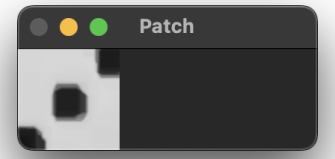
For template matching we use the matchTemplate function from opencv with TM_COEFF_NORMED.

The main trick for good results is to preprocess the patches. We remove noise, try to remove the lines and board from bad placed pieces, also preprocess the templates, and then call the template matching function.

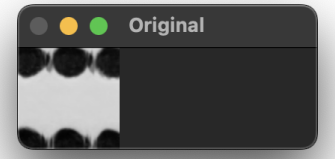
```
kernel = np.ones((3, 3), np.uint8)
image = cv.dilate(image, kernel, iterations=4)
image = cv.erode(image, kernel, iterations=5)
```



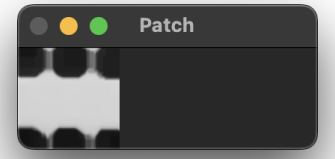
(a) Original Template 1



(b) Preprocessed Template 1

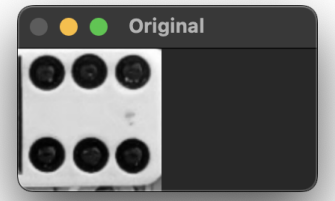


(c) Original Template 2

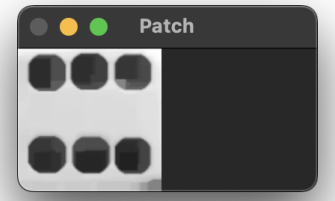


(d) Preprocessed Template 2

Figure 6: Preprocessing Templates Visualization

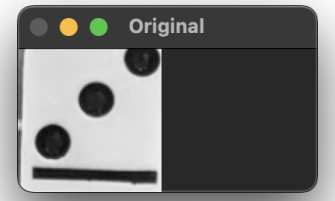


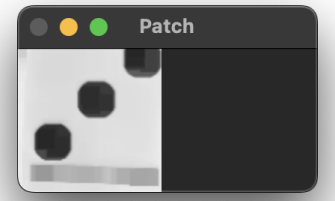
(a) Original Patch 1



(b) Preprocessed Patch 1

The preprocess result on extracted patches from the table can be visualized in Figure 7, and the templates preprocessing result is also represented in Figure 6





(c) Original Patch 2 (d) Preprocessed Patch 2

Figure 7: Preprocessing Patches Visualization

4 Score Calculation

For score calculation, once the detection and classification work well, there is only a simple dynamic programming approach.

Since the score is related to the current position of each player on the score board (aka. their total score) we must save that data constantly at each step. On each move we calculate the current score of the player moving the piece by checking the hardcoded bonus values from the tables, by checking if it is a double piece (3,3 / 6,6 etc.) and by checking for both players if the value placed is the same as their current position on the score board.