Computer Vision Project: Mathable Game Score Calculator

Lungu Laura-Vanesa

December 3, 2024

1 Introduction

Mathable is a number-based game where players form mathematical equations by placing pieces on a game board, following specific constraints. The purpose of this project is to automate the detection of game pieces, the number recognition and the calculation of scores based on predefined rules.

2 Solution Overview

The solution consists of one preprocessing step and three primary tasks, as follows:

1. **Image Preprocessing** – This step involves preparing the image for further analysis by extracting the grid structure.
2. **Task 1: Piece Placement Detection** – In this task, the position of the piece placed on the board is detected.
3. **Task 2: Number Recognition on Game Pieces** – This task focuses on recognizing the numbers on the placed pieces.
4. **Task 3: Score Calculation** – In this task, the score for the current round is calculated based on the valid mathematical equations formed by the player.

3 Image Preprocessing Steps

- **Image Resizing**: Shrinks the image by 50% to speed up processing.
- **Image Blurring**:
  - *Median Blur*: Reduces salt-and-pepper noise by replacing each pixel with the median of surrounding pixels.
  - *Gaussian Blur*: Smooths the image for better edge and feature detection.
- **Image Sharpening**: Enhances edges by contrasting the results of median and Gaussian blurs.
- **HSV Color Space Conversion and Masking**: Converts the sharpened image to HSV color space to isolate blue areas representing the game board.
- **Color Masking**: Applies a color mask to extract the game board.
- **Mask Post-Processing**:
  - *Erosion*: Removes noise and small artifacts from the mask.
- **Contour Detection**:
  - Identifies non-zero points in the mask and computes a bounding rectangle around the game board.
  - Crops the blue margins of the board and resizes the grid region to a standard size (14x14 cells, 90x90 pixels each).

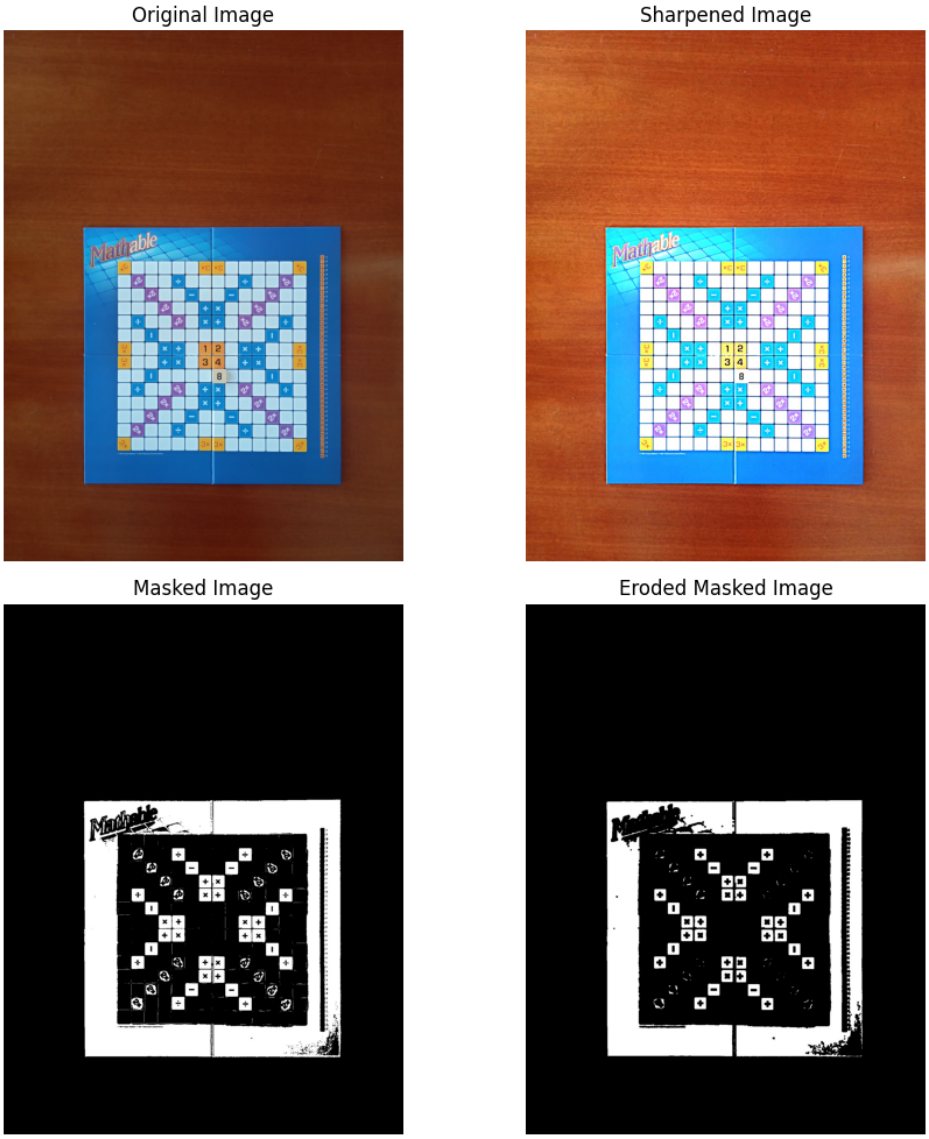

Figure 1: Image Preprocessing Results.

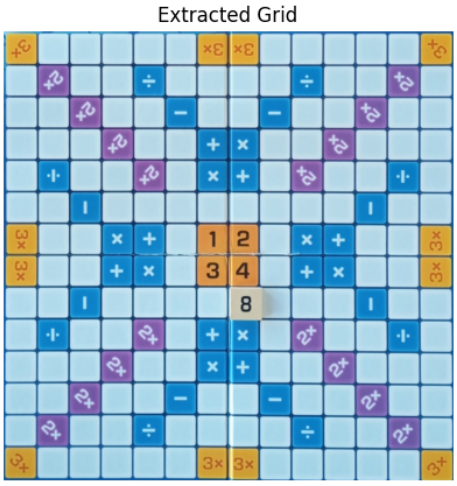

Figure 2: Image Preprocessing Results.

4 Task 1: Piece Placement Detection

To accurately detect the newly placed piece in each round, the `process_game()` function loads two consecutive images at once and calls the `detect_and_extract_piece()` function for further analysis:

- **Grid Extraction**: The board region is isolated from the images using the preprocessing techniques discussed previously.
- **Difference Calculation**: The absolute difference between the two images is computed and converted to grayscale, highlighting the changes introduced by the new piece.
- **Cell Scanning**: The difference image is divided into grid cells, and each cell is analyzed for its mean intensity. The cell with the highest mean intensity indicates the location of the new piece. The piece's location is accurately identified because the grid has a fixed size of 1260 x 1260 pixels (14x14 cells, 90x90 pixels each).
- **Patch Extraction**: The identified cell's region is cropped from the second image for further analysis.

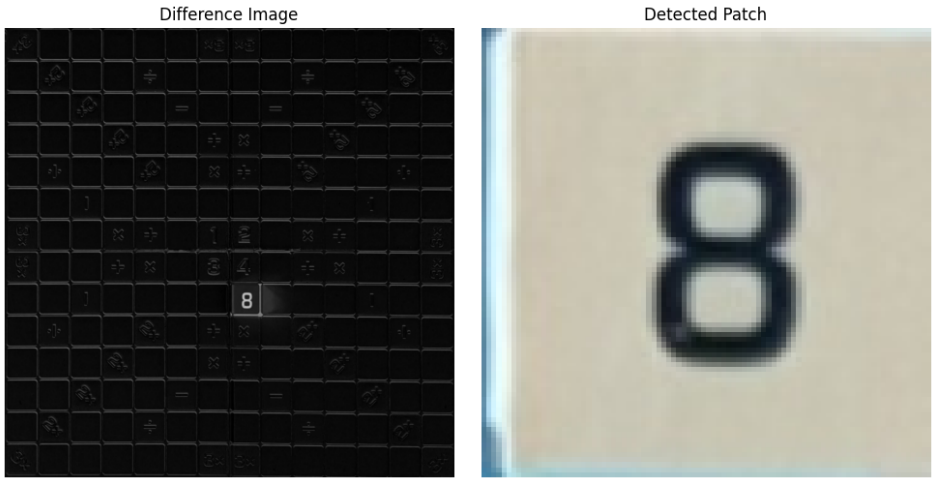

Figure 3: Patch Detection Results.

5 Task 2: Number Recognition on Game Pieces

Once the patch containing the new piece is extracted, the `detect_digits()` function processes it to identify the numerical value of the piece. The steps involved are as follows:

1. **Preprocessing:**
   - The patch is cropped to remove any border noise.
   - It is converted to grayscale (if necessary) and binarized using thresholding to enhance digit visibility.
2. **Contour Detection:**
   - Contours are identified in the binary image to isolate potential digit regions.
   - Small contours are filtered out, and the remaining regions are sorted by their horizontal positions to ensure correct digit order.
3. **Region Extraction:**
   - Each digit region is isolated from the patch image based on the bounding boxes identified in the contour-detection phase.

4. **Classification:**
    - Each digit region is matched against a collection of manually cropped reference images (templates) representing digits.
    - The matching process, handled by the `classify_digit()` function, involves iterating through the templates, resizing the digit patch to align with the template's dimensions, and computing a similarity score.
    - This score is calculated using a normalized cross-correlation method (`cv.TM_CCOEFF_NORMED`) to measure how closely the digit patch resembles each template.
    - The template with the highest similarity score is identified as the corresponding digit, ensuring accurate recognition despite minor variations or distortions.
5. **Multiple-Digit Handling:**
    - If the number on the game piece contains multiple digits, the classification process is performed separately for each digit region.
    - The individual digits are then combined to form the complete number.
6. **Output Generation:**
    - The identified number (token) is written to the requested output file alongside the detected position of the piece on the board.

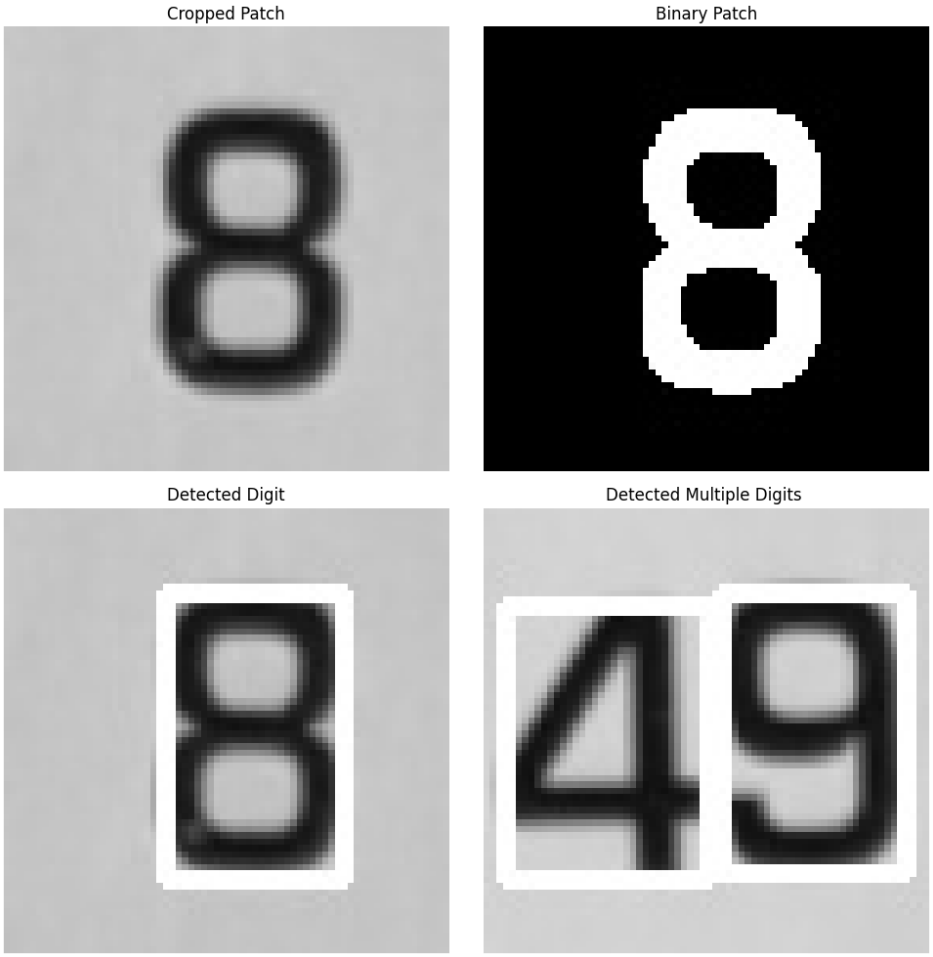

Figure 4: Token Recognition Results.

6 Task 3: Score Calculation

After the token is detected, the score for the move is calculated using a predefined scoring function, `calculate_move_score()`, which takes into account the token's value, its position on the matrix, and the constraints imposed by this position. The detected token is then added to the matrix, updating the game state accordingly. At the end of each round: player scores are stored and a summary of all moves is saved to a text file for record-keeping. Once the game concludes, final scores for all players are compiled and outputted to a summary text file. This ensures a complete record of the game, including move details and the total scores of each player.

7 Overall Code Structure

The program is divided into multiple functions, each responsible for a specific part of the project. This modular design enhances readability, maintainability, and reusability of the code. Below is a brief description of the main functions and their roles:

7.1 Main Functions

- `main()`: Handles input files, manages folder creation, and loads the necessary templates. It organizes the files into dictionaries based on the game number and calls `process_game()` to compute scores for each game.
- `process_game()`: Processes each round by reading the images, detecting the grid, and identifying placed pieces. This function calculates scores for each move, tracks the game's state, monitors player turns, and saves results to output files.
- `extract_grid()`: Performs image preprocessing and grid extraction using computer vision techniques.
- `detect_and_extract_piece()`: Detects changes - a newly placed piece between two consecutive images of the same game.
- `classify_digit()`: Classifies a given image patch (digit) by comparing it with predefined templates.
- `calculate_move_score()`: Calculates the score for a move based on the detected position and value of a piece.

7.2 Additional Utility Functions

- `load_images()`: Loads the images for each round. The current and previous round images are processed together to detect game pieces and changes.
- `load_templates()`: Loads digit templates for matching numbers on game pieces.
- `load_turn_data()`: Loads game turn data to track player actions.
- `initialize_matrix()`: Creates and initializes a 14x14 matrix representing a game board with specific values (board rules) at predefined coordinates.
- `generate_lines()`: Returns lists of coordinates representing the start and end points of horizontal and vertical lines for identifying the boundaries of cells in a grid.
- `multiple_equations()`: Checks if the piece placed at a specific position completes multiple equations, considering the constraints imposed by the cell where the piece is placed.
- `detect_bonus()`: Detects any bonus multiplier at a given position in the matrix.
- `detect_digits()`: Detects and classifies digits in a given patch using predefined templates.