Project 1 - Sudoku Solver
Cojocariu Sebastian - 407 IA

Files description

There is a file containing multiple reusable functions called *utils.py* and three entry points for each task: *task_1.py, task_2.py, task_3.py.* For the task3 there exists a file called *mnist_model.onnx* (a trained model on MNIST dataset to predict the digits) and two cube templates: *own_template.jpg, rotated_cube.jpg.*

utils.py:

- *transform_to_black_and_white*: Converts a RGB photo to Black and White + some repair techniques (morphological)
- *split_lines_from_numbers*: Takes a black and white image, find the largest contour using floodfill (from get_largest_component). Then, it detects the sudoku contour based on the outer_grid determined. Next, we will remove the outer grid and we will map the countour twice: one time for the numbers inside the grid, and one time for the grid lines (while applying a perspective transformation: see transform_perspective function)
- *detect_sudoku_contour*: Detects the largest area based on findContours function. It only choose the largest area that can be approximated by a quadrilateral (by varying the contour error through epsilon of the cv2.approxPolyDP).
- *get_largest_component*: Returns the largest component from image through floodfilling
- *transform_perspective:* Finds a perspective transformation from a specific contour from an image to a rectangle of specified width and height
- *center_digit*: Given an image, it find the largest contour, find the corresponding enclosing rectangle and center the contour. A new image is returned
- *show_images:* Helper that shows multiple images in a grid format (specified by nrows and ncols)
- *remove_image_from_image*: Removes an image from another image.
- *find_no_white_pixels_from_image*: Finds the number of pixels in a given image. The percent is used to evenly shrink the region of interest
- *split_into_bboxes:* Splits an image into bounding boxes. Each bounding box has the structure: (center, width, height)
- *convert_bbox:* Converts a boundingbox specified by (center, width, height) into the corresponding(row_left: row_right), (col_left: col_right) (checking for boundary issues that might occur)
- *extract_numbers_from_image*: Extracts the numbers from an image with numbers (arranged as per sudoku description).
- + *several other functions* that werent used in the end (for a better interpolation technique based on connected components for each cell of the grid)

task_1.py:

- makes use of function from utils.py and has only the function task1.

task_2.py:

- *get_no_white_pixels_from_intersection_line*: Returns the number of white pixels measured on a fixed portion between 2 bounding boxes.
- *get_sudoku_structure*: Returns the sudoku structure based on the thicknes of the lines that compose it.
- *dfs*: Function that compute the sudoku structure based on a Depth-First-Search approach.

task_3.py:

- *apply_model:* Loads a NN model trained on MNIST dataset, split the image into 81 parts, centers the largest component (it assumes this is the digits), run the model and propose a list of possible candidates based on the accuracy).
- *check_sudoku_solver:* Enforce the conditions for sudoku.
- *trim_sudoku_possibilities:* The first trim algorithm before starting the backtracking to reduce the complexity.
- *backtracking_sudoku:* Backtracking based on possible candidates per digit.
- *check_second_condition:* Check second condition (the line condition adjiaceny from the requirement).
- *get_all_three_sudoku:* Function that return the 3 sudokus + their contours on the original image
- *transform_perspective_onto_cube_face*: Maps an image's contour onto a bbox from a template_image.
- *construct_cube_on_own_template*: Maps an image as in the testing onto an own cube template (this will later be mapped on the desired template automatically).
- *template_matching:* Find the transformation from our own template to a given template using a template matching algorithm. After that, it applies the tranformation matrix M onto the own_template_with_sudoku.

Algorithms:

For **Task1** and **Task2** the strategy is almost the same:

- Initial Image.
- Convert it to black and white.
- Find the sudoku contour (it assumes it is the largest component in the image). Adopt a floodfilling approach to find it.
- Split the images into 2 parts: numbers only and grid only.
- Warp this images into fixed 2000x2000 images.
- Extract numbers (and for Task 2 extract grid structure using a DFS approach).
- Final output.

For **Task3** the following strategy is used:

- Initial Image.
- Convert it to black and white.
- Recursively find the largest contour (which must be a sudoku), remove it and repeat the algorithm (3 times to get all the sudokus).
- Split the images into 2 parts: numbers only and grid only (not needed at this task).
- Warp the last image into fix 1000x1000 images.
- For each image containg the numbers, split the image into 81 equal parts, and run the neural model against them to find the possible candidates based on the prediction confidence. Multiple candidates can be proposed AT this stage.
- Trim the candidates (since there will be some cells that contain only one digit as the proposed candidate, we trim based on that to reduce the complexity of backtracking).
- Solve the remaining sudokus via backtracking.
- Match the sudoku on the cube (find their relative posesional).
- Map each sudoku face onto an own cube template.
- Find a transformation from own_cube_template to given_cube_template via invariant template matching.
- Final mapping between own_cube_template_with_sudoku onto given_cube_template.