WORDOKU SOLVER

Surya Chandra
EENG510
December 3, 2014
Outline

• Background
• Goal
• Steps to solve
• Algorithm used for each step
• Testing and Results
• Future work
• Questions
A wordoku puzzle is similar to a sudoku, but uses alphabets instead of numbers.

A **keyword** is given at the bottom of each puzzle, consists of 9 alphabets that will be used to solve the puzzle.
Goal

- For any given wordoku puzzle (with *any font*), solve the puzzle and display the solution image.
Steps Involved

① Separate the keyword and grid
② Extract elements
③ Find Matches
④ Solve as a sudoku
⑤ Paste the solution back
Step 1: Separation

- First, threshold and use Hough transform to find boundary.
Step2: Extraction

1) Keyword :-

- Got rid of any small noises by using `imopen` and `imclose` commands.
- Used the `regionprops` to extract the individual characters.
- Assigned values.

![Characters extracted](image-url)
Step2: Extraction

2) Puzzle Elements :-

- Used the *houghlines* to extract 20 lines.
- The lines were in random order and it was necessary to sort them.
- Some of the puzzles had thick and thin grid lines. So, I created a new set of lines with average of distances.
Step2: Extraction

2) Puzzle Elements :- (cont…)

- Cropped each element between intersections and stored separately.
- Also, region properties of each element was extracted.
Step3: Matching

1) Classification Tree:

- Use various region properties (area, centroid, axis lengths) to create a decision tree.

- Used this tree to predict the classes (1-9).

This classification gave wrong matches in cases where two or more letters had almost similar areas and axis lengths.
Step3: Matching

2) Normalized Cross-Correlation:

- The max scores when each element of the puzzle was cross-correlated with all the 9 key characters was calculated and the max score among these was used to predict the class(1-9) of that element.

-- Eroding before using the cross-correlation gave better results.

-- Limitation: When one letter could completely fit inside another letter, there was a chance for wrong detection.
Step 3: Matching

3) Support Vector Regression:

- I used different region properties of key characters as a training set to train an SVR algorithm.
- The predictions I found were more accurate and reliable.

-- Limitation: This still depends on the properties like Area, Axis lengths of each element and how much they vary between the different characters used.

http://www.mathworks.com/matlabcentral/fileexchange/43429-support-vector-regression
Solution?

Combining algorithm -

- All the max cross-correlated scores > 0.68 are stored as the closest matches.

- And use support vector regression prediction to further narrow down the value.

-- Combining both of the algorithms matched 10 different puzzles with 100% accuracy. However, it significantly increases computational time.
Step4: Solving

- The location matrix reduces the problem to a sudoku.
- Using a MATLAB script I found which recursively solves for all possible values for every blank cell in the matrix.

UNSOLVED

```
 0  7  0  6  0  8  5  0  3
 3  0  0  2  0  0  0  0  8
 0  2  0  4  0  0  7  1  0
 7  0  0  0  6  9  1  0  2
 0  0  0  0  0  8  6  0  0
 0  1  3  0  0  0  0  0  0
 2  0  0  0  4  0  0  0  7
 1  0  0  0  0  0  4  0  0
 4  8  0  7  0  6  2  0  0
```

SOLVED

```
 9  7  4  6  1  8  5  2  3
 3  5  1  2  9  7  6  4  8
 8  2  6  4  5  3  7  1  9
 7  4  8  5  6  9  1  3  2
 5  9  2  3  7  1  8  6  4
 6  1  3  8  2  4  9  7  5
 2  6  9  1  4  5  3  8  7
 1  3  7  9  8  2  4  5  6
 4  8  5  7  3  6  2  9  1
```

Step5: Printing

- I used the sorted hough-lines to find their intersection points using `polyxpoly`, and hence, width and height of each cell separately.

- Paste according to the dimensions of the element.

- Used the amount of black color in each character as reference to print.
Step 5: Testing/results
Step 5 : Testing/results

Real Images had Real problems -

1) Had to use `cpselect` to manually project the onto a square template

2) Problems with thresholding and detecting houghlines

Tweaking the algorithm-

1) Divide the grid into equal parts and extract.

Needs more testing -

Matching was successful for 1 out images out of 2 images.
Conclusions:

- Able to handle a tilt up to 10 degrees without user interaction.
- It could solve the puzzle for any given font.
- It could solve when lines were missing in the puzzle.
- Has 90 percent (10/11) accuracy.

Limitations:

- Keyword characters need to be separated.
- Could not handle real images without user interaction.
- Noise/thresholding issues for real images.
Future Work

- To optimize the algorithm to handle varying size of keyword and puzzle elements.
- To extend this algorithm to solve for real images with thresholding issues.