Scrabble Board Automatic Detector for Third Party Applications

David Hirschberg
Computer Science Department
University of California, Irvine
hirschbd@uci.edu

Abstract

Abstract – Scrabble is a well-known word game in which two to four players vie for victory by creating words with tiles placed onto the board with rules in similar fashion to those in crosswords. There are several other web apps that exist to help players, but require manual input of letters to create a logical view of the board. This paper presents a Scrabble image processing method to create a logical view of snapshot of a Scrabble game board, to be used by a Scrabble assistant or Scrabble solver for assistance for a player. In test images, this algorithm was able to digitize boards with a 94% accuracy on average, and correctly identify characters with a 67% accuracy on average, this can be greatly improved with a better OCR method; using a third party OCR, 87% accuracy classification of characters is achievable.

1. Introduction

Scrabble is a well-known word game in which two to four players take turns vying for victory by creating words in turn using a set of seven letter tiles and placing them on a grid presented by the board game. Rules for placement of letters follows similarly to crosswords. There are similar games including *Words with Friends*. There are also *Scrabble* solvers, such as http://www.scrabulizer.com/ exist to help players with their play. However, they do require manual input of the current board to enable their utility. Manual input from users is tedious and cumbersome. This paper presents a *Scrabble* image processing algorithm for creating a logical representation of a *Scrabble* board for use in any application, including *Scrabble* solvers.

2. Scrabble Image Processing Algorithm

There are several steps in the *Scrabble* Image Processing Algorithm, henceforth known as the image processing algorithm or algorithm, and elements of the algorithm process when produced can be told from the perspective of us as the algorithm, therefore we. The algorithm's steps include, normalization, segmentations, perspective correction, grid placement, optical character recognition, logical representation of board. Now, let us bring this quixotry to fruition, before we start studying zymurgy!

2.1. Normalization

Load the image, convert to grayscale, and then use an adaptive histogram equalization to account for glares, flash, and other flaws and/or variations in the image.

A requirement to normalize the image such that tiles are 30 pixels wide, the algorithm creates an estimate from empty squares from the image. To locate these empty squares on the *Scrabble* board, maximally stable extremal regions (MSERs) are found. However, we put a restriction on these regions, they cannot be larger than 1 square of a board. There are 15 rows and 15 columns on a *Scrabble* board, giving us 225 squares possible. Since there are 225 squares, we cannot allow any region to be more than $1/225^{th}$ of the area of the total image. Another restriction we add to the regions are they need to be square, therefore we do not allow the height and width of these regions to not differ by more than 0.05%.

Given these regions with their restrictions, take the median width of these regions and resize the image such that tiles are 30 pixels wide.

Figure 1 - Square MSERs

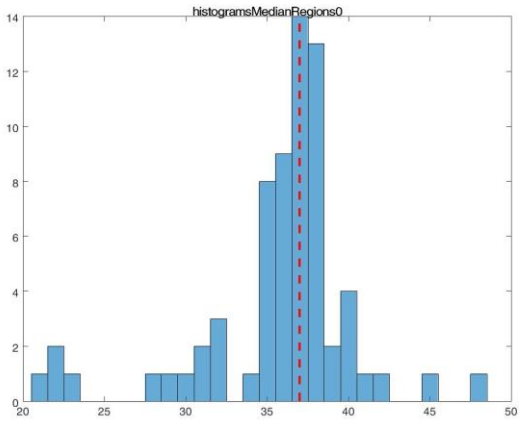


Figure 2 - Median of Square MSERs

2.2. Segmentation

Now that the board is resized, we will segment the board from the background. There are two outcomes to this, in one case we will have just our 15x15 square board with nothing else, or we will have the entire board, including the letter counts and *Scrabble* logo. In the latter case, we will take care of that in the grid fitting section, for now assume we will have the former case only.

Once the image has been resized, we will perform a canny edge detection, giving us binary image of edges.

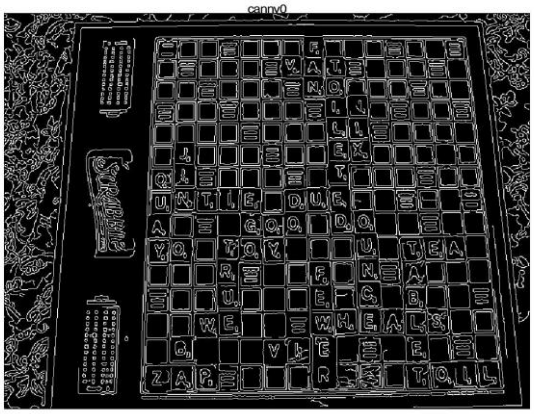


Figure 3 - Canny Edges

We will dilate the image using a vertical and horizontal morphological structuring element (STREL) with a 3 pixel length. After dilation we will fill in holes. This will give us several closed regions in the image.

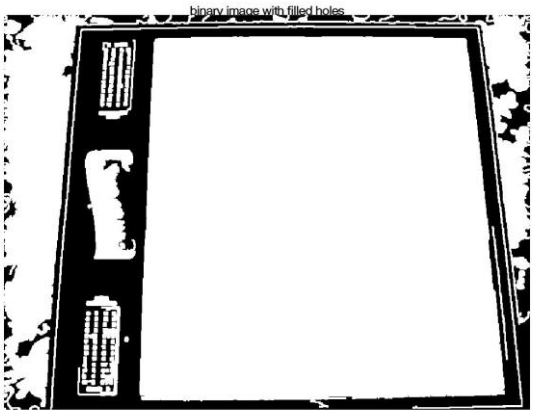


Figure 4 - Dilated Edges

Next perform two erosions with a diamond shaped STREL with a single pixel radius. This will separate regions that are thinly connected, such as the *Scrabble* logo and the actual area of play. Then we extract the single largest blob from this binary image.

Our largest blob is now our game board. Note that the board is a quadrilateral in any perspective.

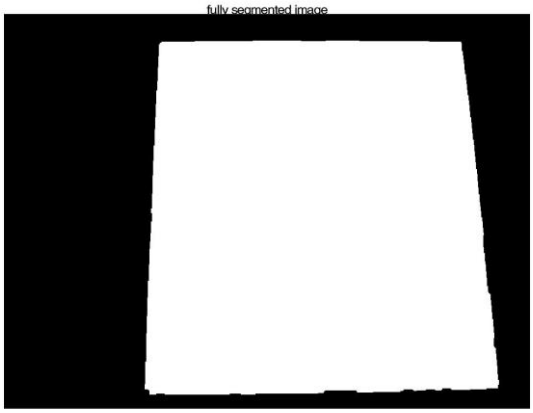


Figure 5 - Segmentation Mask

2.3. Perspective Correction

Photos of a *Scrabble* board are rarely from a directly overhead view; they usually have some other perspective. Given a quadrilateral for our board, we will transform this into a square, where the known points of the game board are the corners of the quadrilateral from the original image. These are then transformed such that the corners of the quadrilateral become the corners of a square. Crop the image around the square, and we are left with an overhead view of the board.

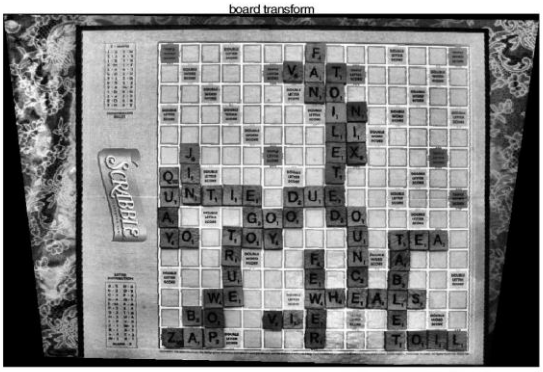


Figure 6 – Perspective Corrected Image

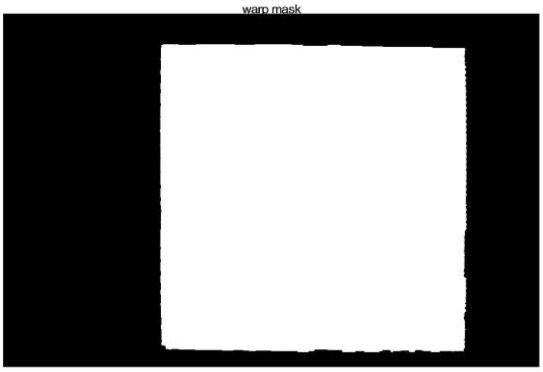


Figure 7 - Perspective Corrected Mask

Figure 8 - Area of Play Image Overhead

2.4. Grid Placement

The algorithm has so far given us a cropped image resized such that tiles are 30 pixels wide and viewed from directly overhead. Now the algorithm needs to separate the board into a 15x15 grid where each cell consists of one square or tile.

We find MSERs just like before, and they will be square as well. The grid is then placed using a k = 15, k clustering of the square regions, giving us 32 lines, 16 in the horizontal direction and 16 in the vertical direction giving us a constrained 15x15 grid. This does account for when the cropped image has the *Scrabble* logo on the side by only placing the grid on the actual area of play.

Thus, the algorithm given us an image divided by the rows and columns of the area of play.

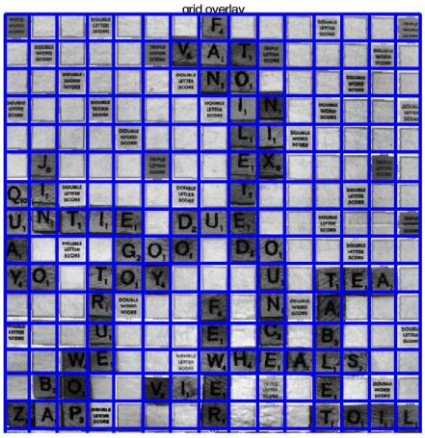


Figure 9 - Grid Placed

2.5. Optical Character Recognition (OCR)

The algorithm has given us a gridded board, and we now have individual access to every tile on the board at will at this point.

Our algorithm's self-made OCR method is mentioned here, look at future work for alternates that do work better.

For every tile on the board, with coordinates (i , j), we will run a k nearest neighbor classification algorithm to classify tiles.

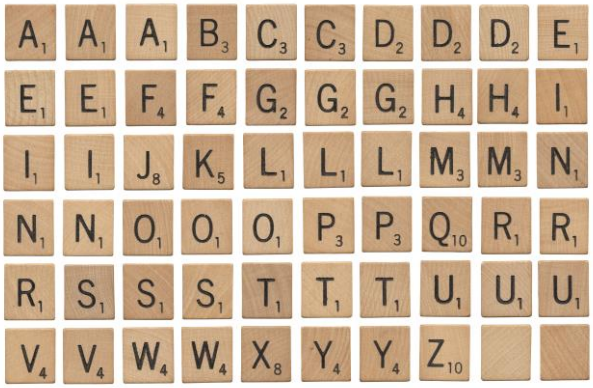


Figure 10 - Supervised Data

Our data is taken from a high quality image of tiles. After extracting the individual tiles from the image, we create our data from extracting the letter form an individual tile. Given an image of a tile, use an adaptive histogram equalization, just like we did at the beginning of the algorithm for normalizing the board. Find the center of

mass of the tile, based upon pixels under a threshold. Extract the pixels under the threshold value for the tile, note that letters are darker than the tile itself. Place the extracted pixels in the center of the image.

Now that we have a single letter placed in the center of the image, we want to create more data that can account for several variations, the tile was placed crooked in rotation, was slightly offset from the center of the square, the camera was not in perfect focus and the image is slightly blurry. We create more data by copying our original reference data of the centered letter and iteratively rotating, blurring, and translating the letter in its 30x30 square. This gave us in our implementation 80,704 letters to work with.

Now that we have our data, for each tile, we push it through the same algorithm for which we created our data, but without the rotation, blurring, or translation. This gives us a meaningful image that we can run our kNN classification algorithm, we ran it with k = 3. This allows us to classify our tiles.

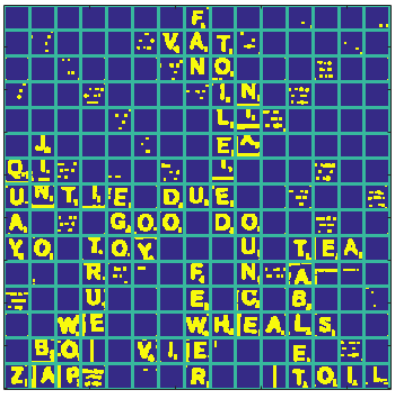


Figure 11 - Tiles to go through kNN

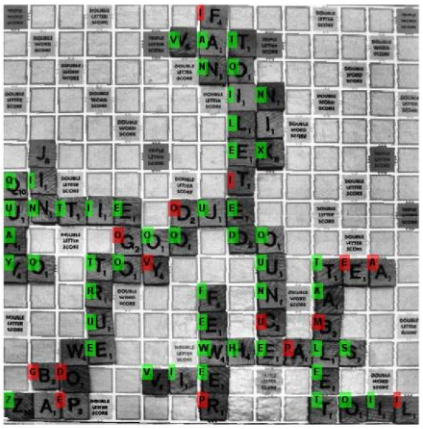


Figure 12 - Classification of Image

2.6. Logical Representation

Once we have gotten all of our squares identified, we have them stored as a 15x15 character matrix in MATLAB. We can store this in any format we wish, and will be fully up to the reader to implement at will for whichever application they wish to use it in.

3. Results

The *Scrabble* image processing algorithm was tested on a seven images all taken from *Scrabble* sweepstakes, which contained images of the full area of play with some background taken at an angle.

The rate at which tiles are correctly classified are shown in figure 13, with an average of 67%. However, game square classification rate is at a 94% classification rate.

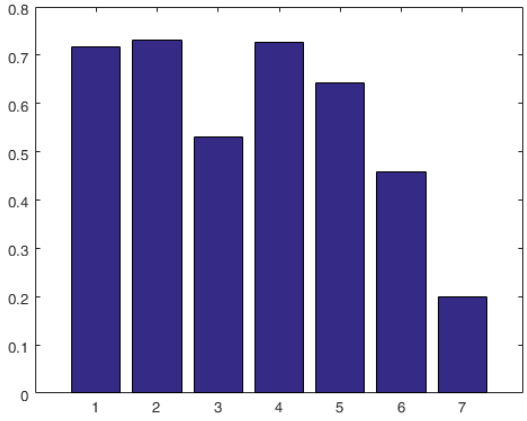


Figure 13 – Tile Classification Rate of Test Images

4. References

[1] Chen, Huizhong, et al. “Robust Text Detection in Natural Images with Edge-Enhanced Maximally Stable Extremal Regions.” Image Processing (ICIP), 2011 18th IEEE International Conference on. IEEE, 2011.

[2] Paul Rivas, “Sunday Scrabble Sweepstakes.” One Sorry Blog. WordPress. 2007. https://onesorryblog.wordpress.com/category/sundayscrabble-sweepstakes/

[3] Warburton, Tim. "OCR." OCR. N.p., n.d. http://www.caam.rice.edu/~timwar/CAAM210/OCR.html

[4] Brummitt, Liam. “Scrabble Word Recognition” Scrabble Referee, https://www.youtube.com/watch?v=c3ywTfeTqOE

[5] MATLAB and Statistics Toolbox Release 2015b, The MathWorks, Inc., Natick, Massachusetts, United States.

5. Appendix

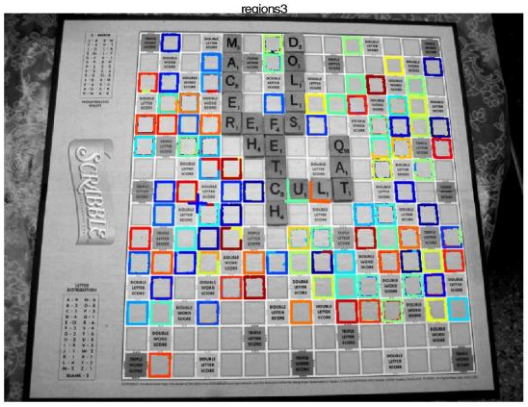
Note, images are ordered column by column, top down from 1 through 7

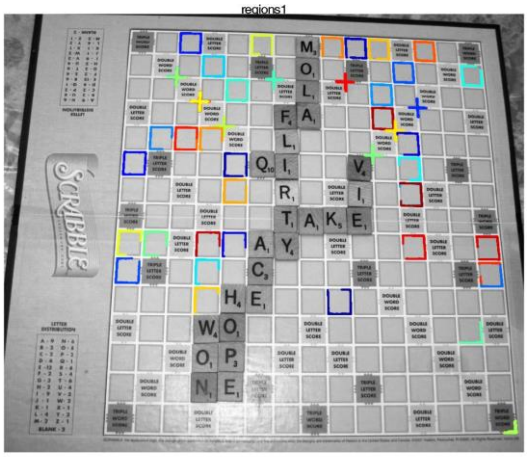
Test Images

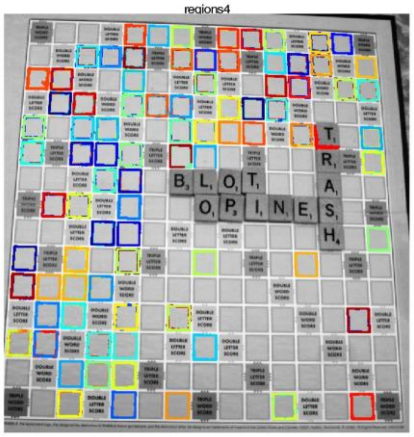

SCRABBLE
BRAZE
JOKER
OR
QUAT
LOUNGE
IN

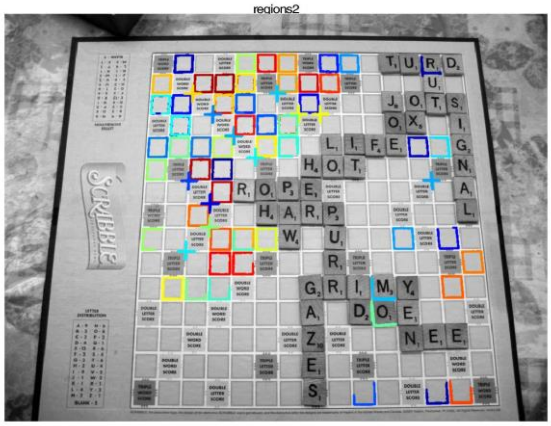

SCRABBLE
MOTHER
TOILET
YARN

Square MSERs from Original Images

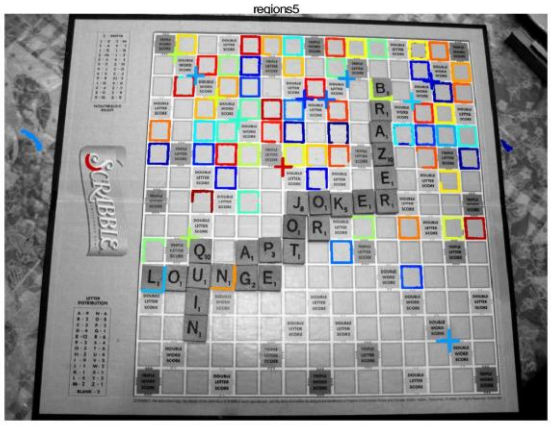




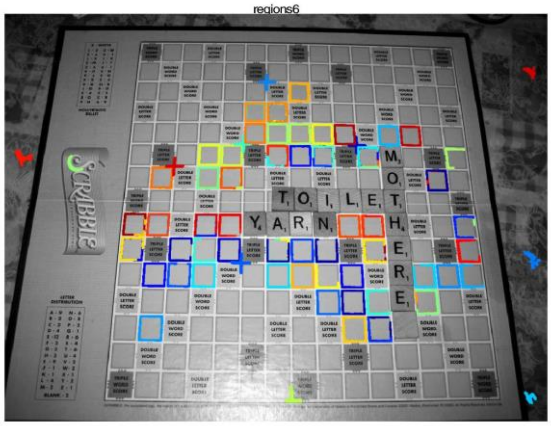




regions5

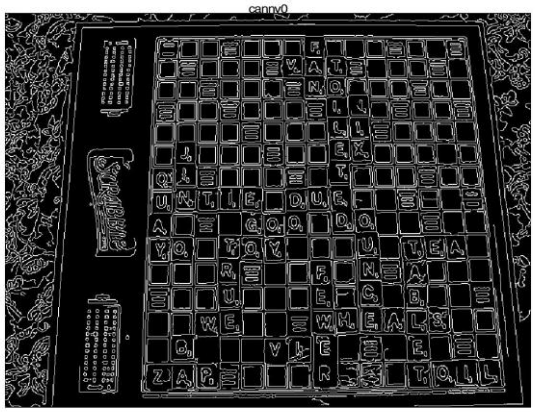


regions6

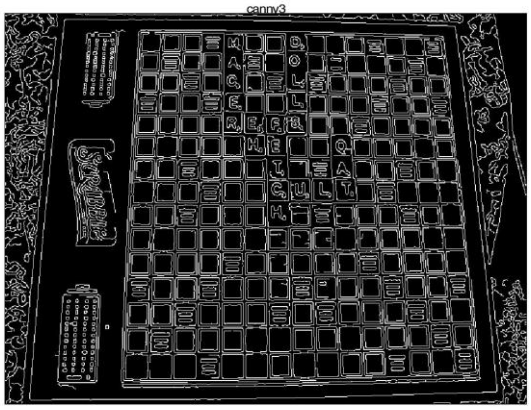


Canny Edges

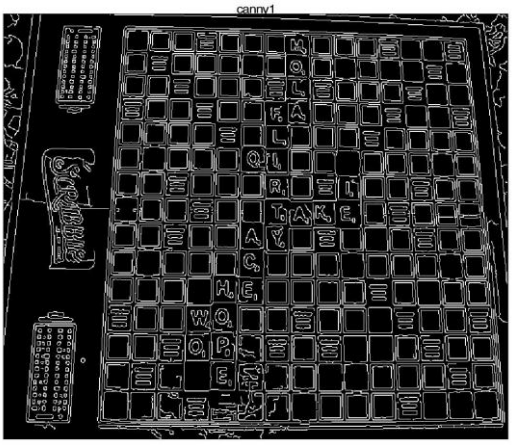
canny0



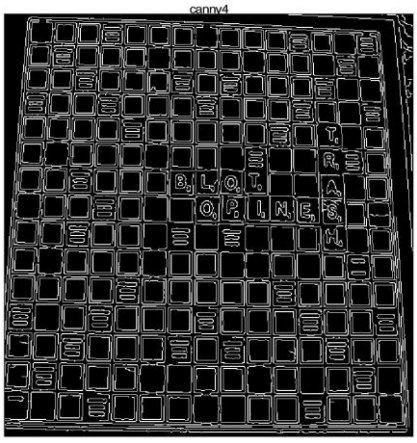
canny3



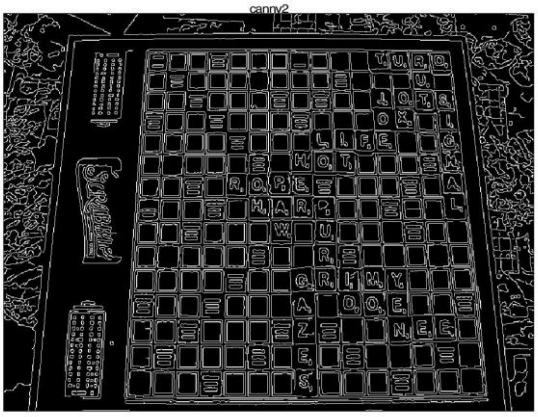
canny1



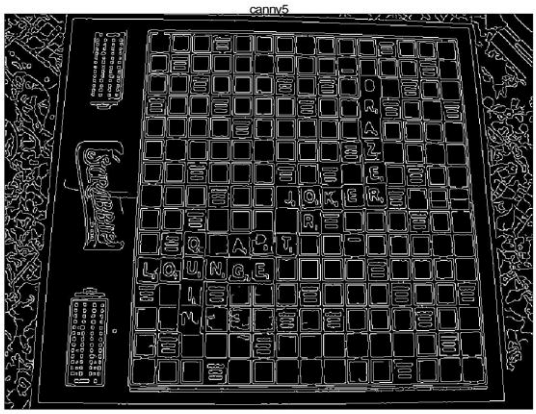
canny4



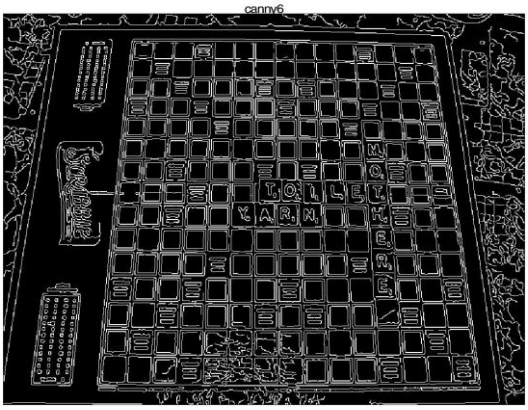
canny2



canny5

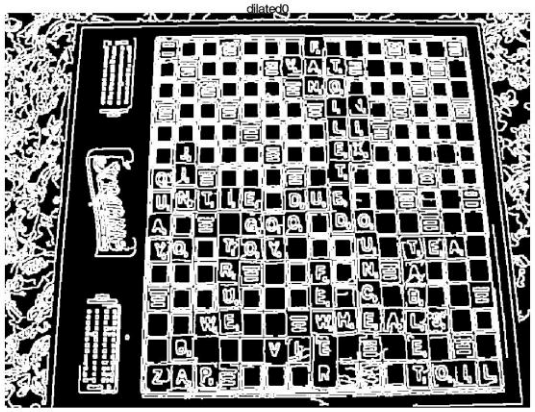


canny6

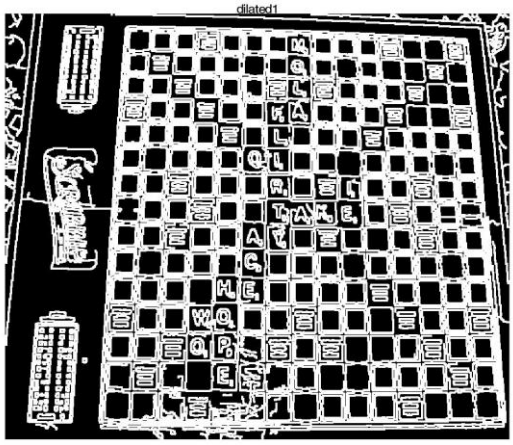


Dilated Canny Edges

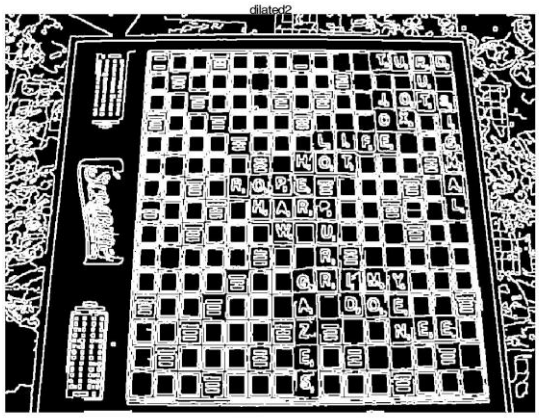
dilated0



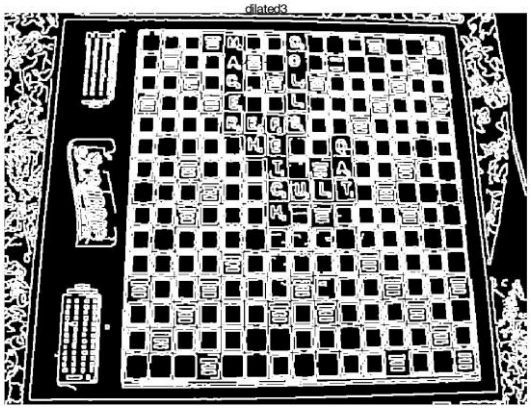
dilated1



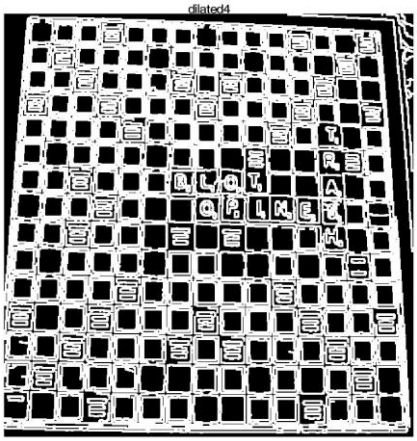
dilated2



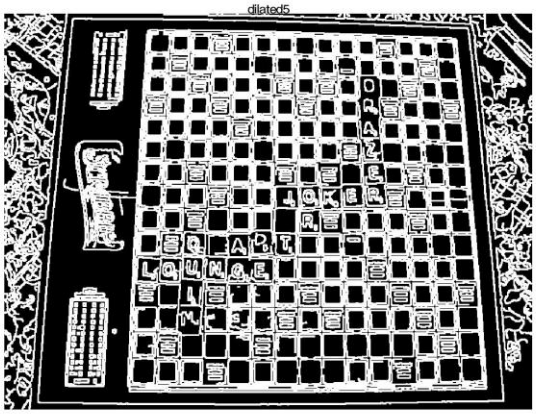
dilated3



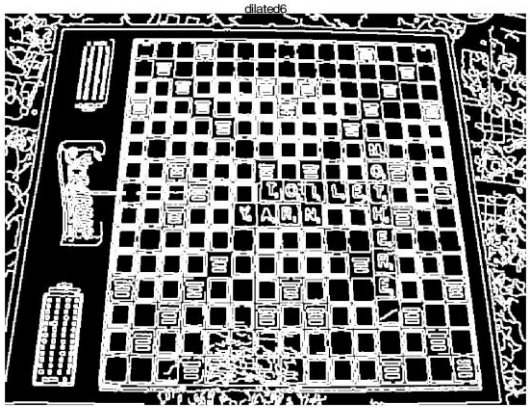
dilated4



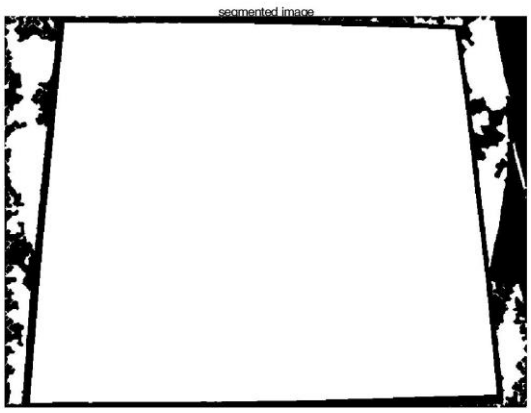
dilated5

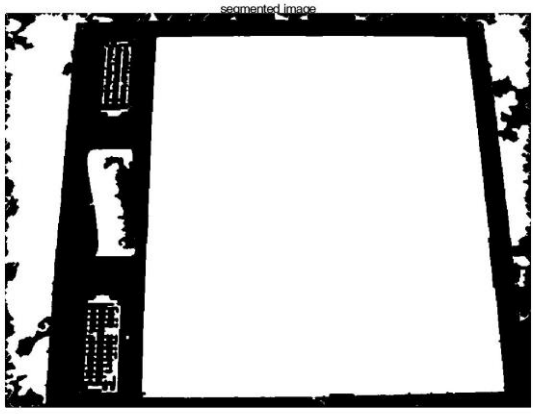


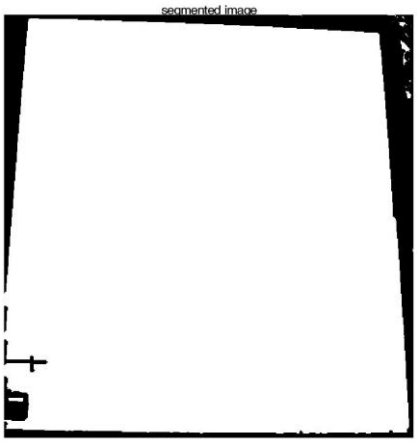
dilated6

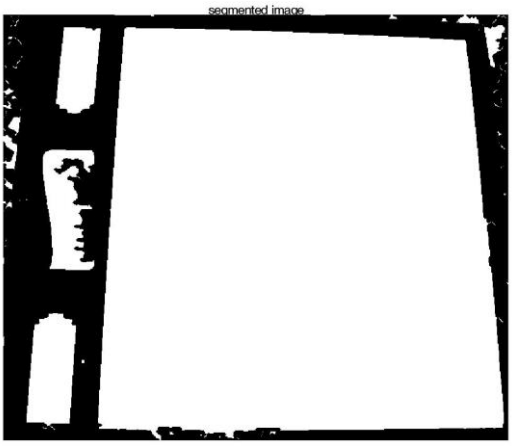


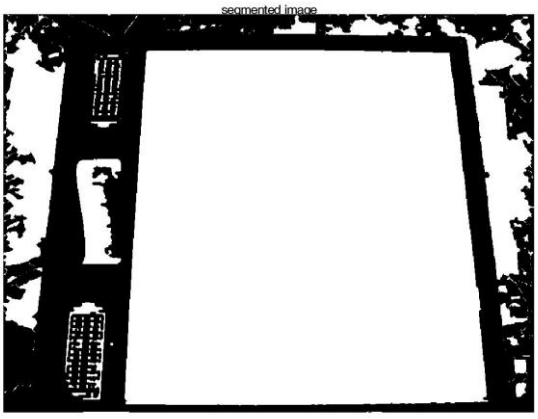
Filled Regions Without Largest Blob Extraction

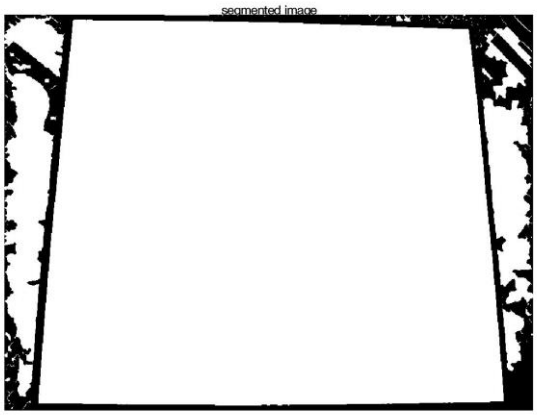


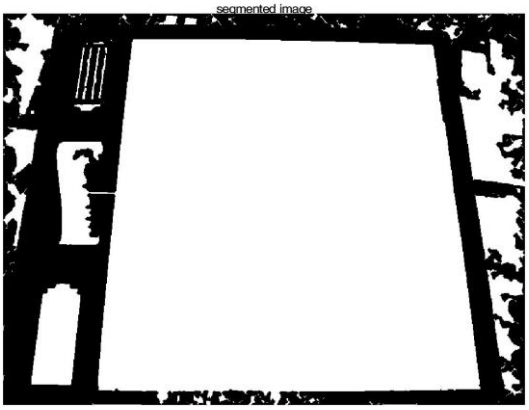




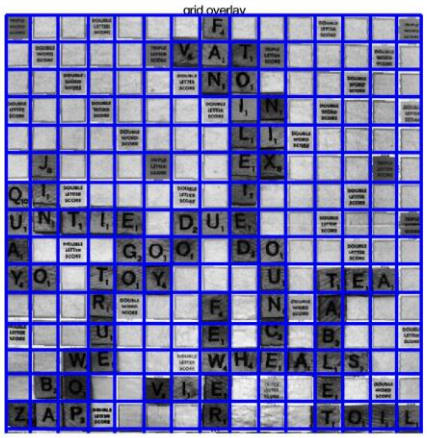


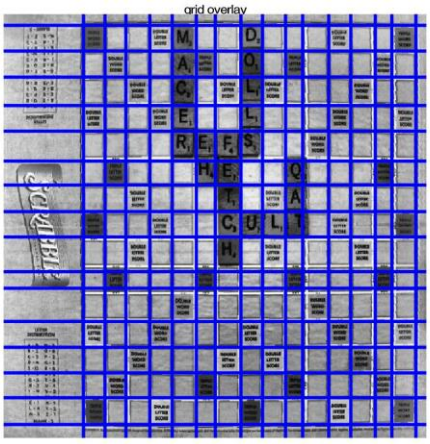


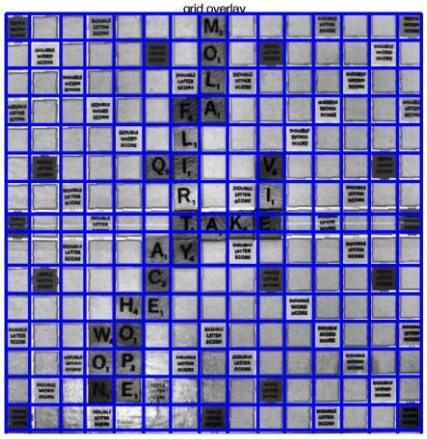

segmented image

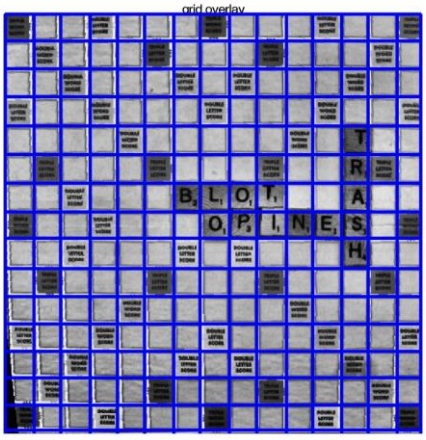

segmented image

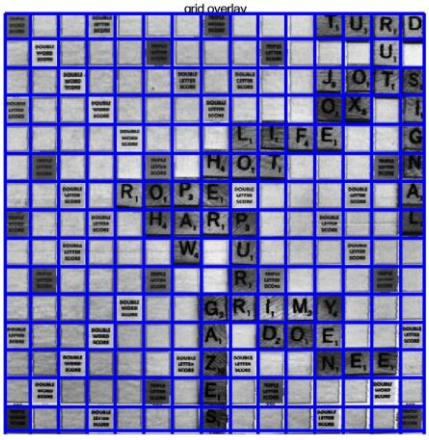
Grid Placement



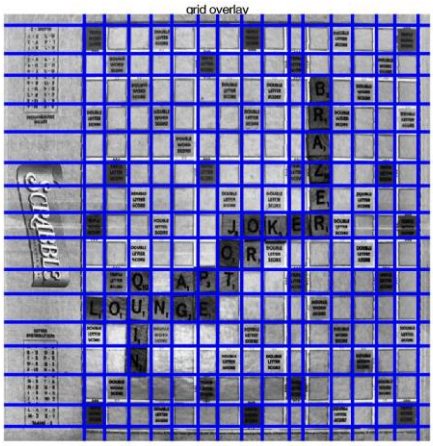




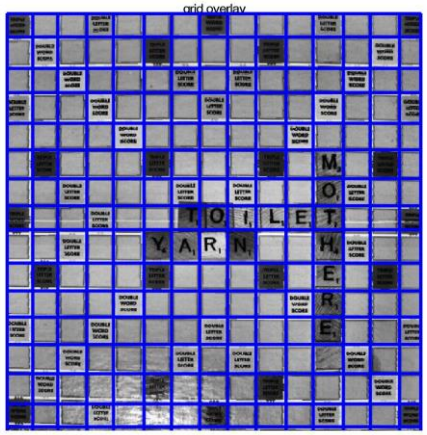




grid overlay

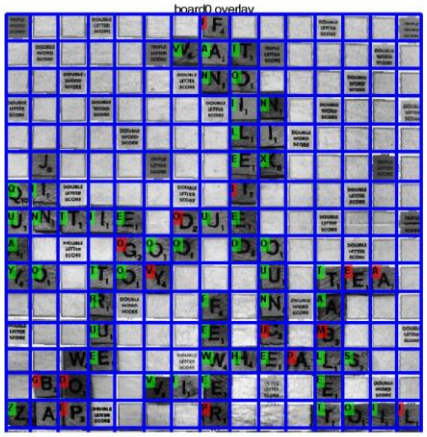


grid overlay

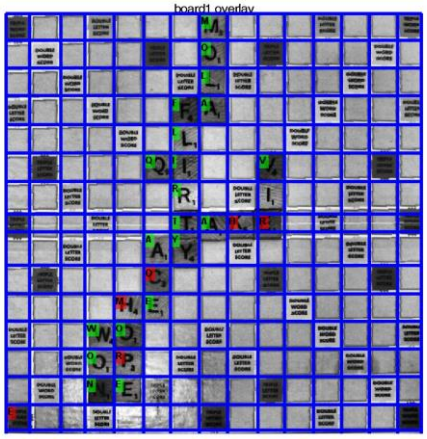


Test Images Classified with Grid

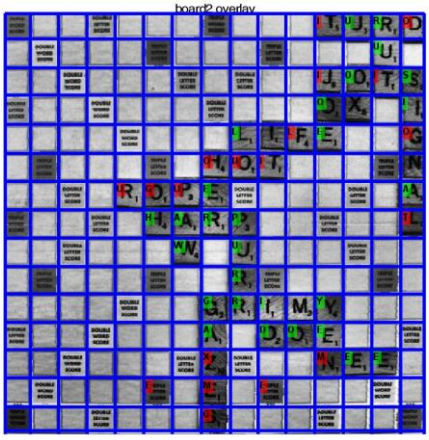
board0.overlay



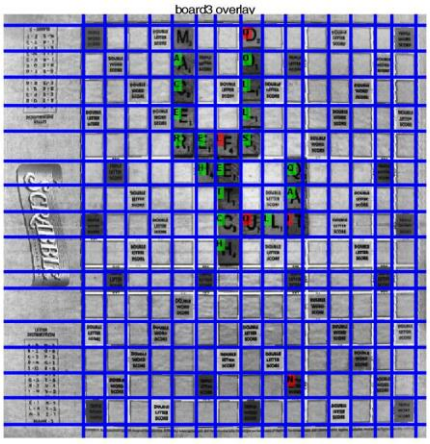
board1.overlay



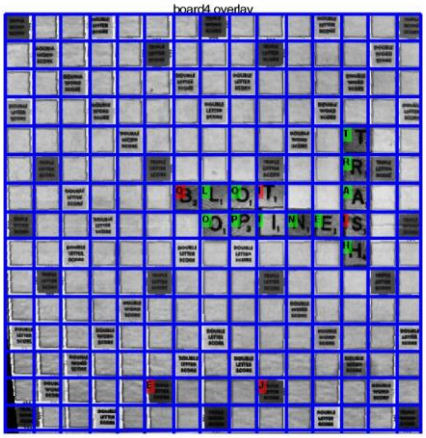
board2.overlay



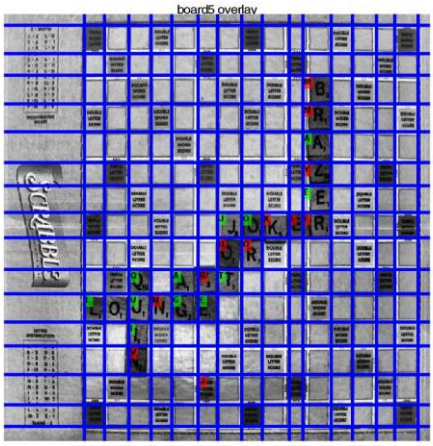
board3.overlay



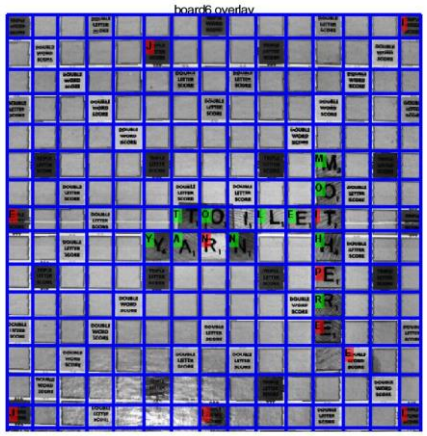
board4.overlay



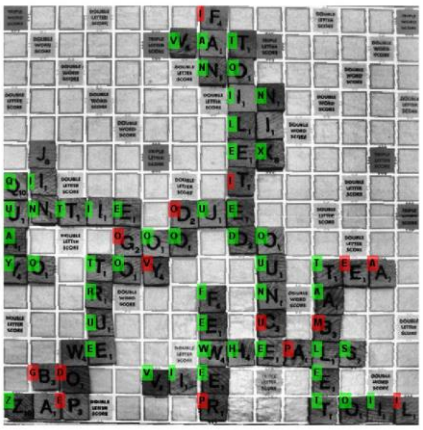
board5_overlay

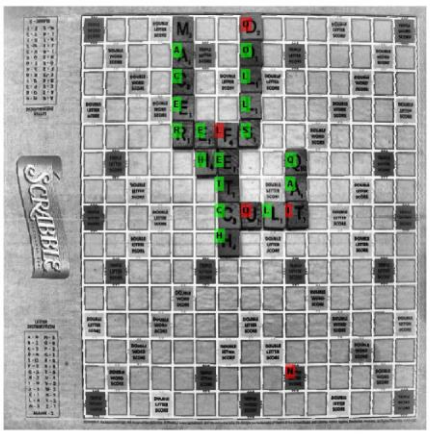


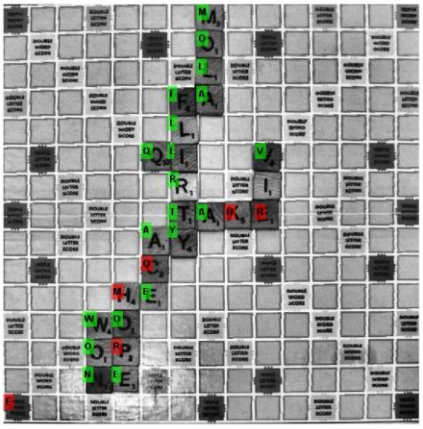
board6_overlay

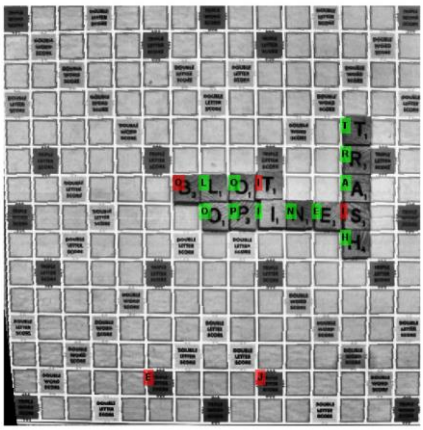


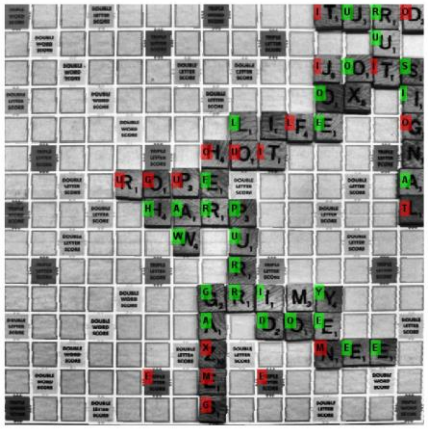
Test Images Classified

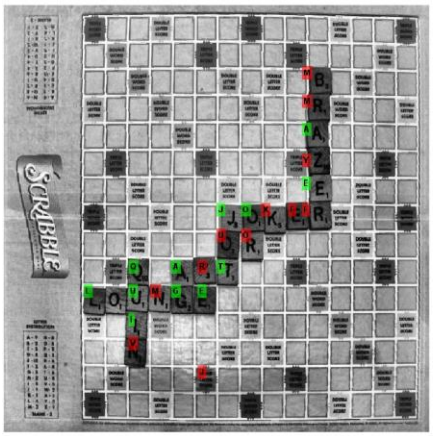











SCRABBLE