Bananagrams Tile Extraction and Letter Recognition for Rapid Word Suggestion

Steven Leung, Steffi Perkins, and Colleen Rhoades
Department of Bioengineering
Stanford University
Stanford, CA

***Abstract*—Bananagrams is a competitive free-form crossword-building game that hinges on a player's ability to recognize words from a pool of available letter tiles. This paper describes an Android application that may assist a player in word formation. The application takes an image of letter tiles as input and outputs a list of possible words. This is performed using a series of image processing steps for tile extraction and letter recognition.**

Keywords—Bananagrams; tile extraction; letter recognition; locally adaptive thresholding; Hough transform; Hu moments; Android

I. Introduction

Bananagrams is a popular crossword-building game where players use lettered tiles to build their own free-form crosswords. Players continue drawing new tiles and incorporating them into their crosswords until all tiles in the communal resource pool are gone. The first player to complete a crossword with all of his or her tiles wins the game; therefore players who can quickly identify possible words to incorporate into their crosswords have the advantage. This paper describes an Android application that utilizes image processing to provide a list of possible words that can be built from the tiles in a player's possession. The application uses the phone camera to take a picture of a player's tiles, extracts the tiles based on region properties, rotates them, and then compares their Hu moments with those of letter templates in a database to find a best match. The string of letters from the extracted tiles is then sent to an online anagram solver and the generated list of possible words is returned to the player on the phone screen. Players have the option of including additional letters in the query string, since new words need to be incorporated into an already existing crossword.

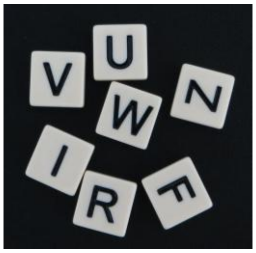


Fig. 1. Example query image used as input to the Android application.

II. Image Processing Algorithm

A. Image Acquisition and Pre-Processing

En face color images were taken with a phone camera and sent to MATLAB. An example image is shown in Fig. 1. The images were converted to grayscale then binarized using a locally adaptive thresholding windowing approach [1]. Each image was divided into approximately 100 x 100 windows and the variance in pixel intensity was calculated for each window. If the variance within a window was lower than a given threshold, all pixels in that window were set to background (0, black). Otherwise, Otsu's method was performed [1]. The resulting images contained many connected regions that formed outlines of letter tiles, as shown in Fig. 2A. Some portions of the tile face were set to background because of low variance in pixel intensities.

B. Tile Extraction

Fig. 2 shows the process of tile extraction. Convex hull areas of connected regions were combined to produce a tile mask (Fig. 2B). The convex hull is the smallest convex polygon that contains every pixel in a region. Tile regions were isolated using selection criteria of area, eccentricity, and Euler number [1]. The bounding boxes of isolated tile regions were used to extract the tiles from the original grayscale images (Fig. 2C). Hough transforms were performed on edge maps to determine tile orientations [2,3]. The tiles were then rotated and cropped so the edges of the image lined up with the edges of the tiles (Fig. 2D).

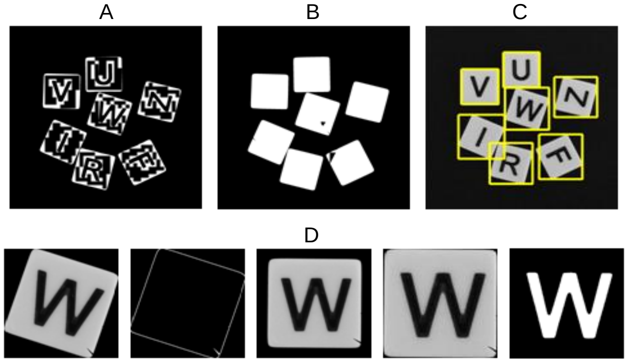


Fig. 2. Intermediate images of the image processing algorithm. A) Result of locally adaptive thresholding. B) Tile mask constructed from region convex hull areas. C) Bounding boxes of extracted tiles. D) Process to rotate and crop tiles extracted from grayscale image.

C. *Hu Moment Calculation*

Each grayscale extracted tile was globally binarized using Otsu's method. In addition, holes in letters {A, B, D, O, P, Q, R} were filled with white pixels (justification for this decision is outlined in Appendix B). The first four Hu moments were then calculated for each extracted tile, as shown in equations (5) – (8). The moments are invariant to scale, rotation, and translation [4,5].

Equation (1) was used to calculate raw image moments of the image *I(x,y)*. *i* and *j* designate the moment orders in the x- and y-directions. Equation (2) was used to calculate the centroid $(\bar{x},\bar{y})$. Equations (3) and (4) were used to calculate the normalized central moments η_{ij}. These values were required in order to calculate the Hu moments.

$$M_{ij} = \sum_x \sum_y x^i y^j \, I(x,y) \qquad (1)$$

$$\bar{x} = \frac{M_{10}}{M_{00}} \qquad \bar{y} = \frac{M_{01}}{M_{00}} \qquad (2)$$

$$\mu_{ij} = \sum_x \sum_y (x-\bar{x})^i (y-\bar{y})^j \, I(x,y) \qquad (3)$$

$$\eta_{ij} = \frac{\mu_{ij}}{\mu_{00}^{\left(1+\frac{i+j}{2}\right)}} \qquad i+j \geq 2 \qquad (4)$$

$$H_1 = \eta_{20} + \eta_{02} \qquad (5)$$

$$H_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \qquad (6)$$

$$H_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \qquad (7)$$

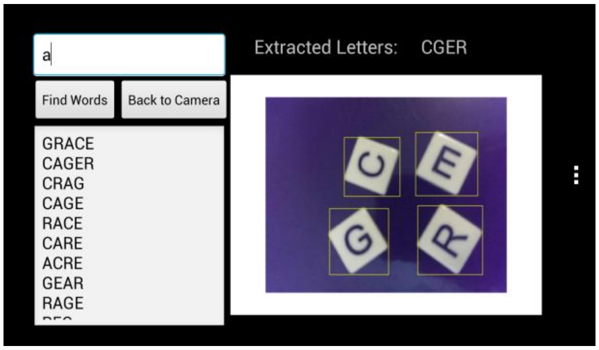
$$H_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \qquad (8)$$

D. *Template Database*

26 different letter tiles were used to generate a database of template letters; the pictures were taken with a digital camera with the same conditions and parameters to ensure consistency across the images. The first four Hu moments were calculated for each template and included in the database.

E. *Letter Recognition*

For any given query image, letter tiles were extracted and Hu moments were calculated for each tile. The L_1 distance was calculated between the Hu moments of each extracted tile and database template pair [6]. The minimum distance signified the best match, and the letter corresponding to the best match was recorded in a string array. The final string included the best letter match for each extracted tile.



a

Extracted Letters: CGER

GRACE
CAGER
CRAG
CAGE
RACE
CARE
ACRE
GEAR
RAGE

Fig. 3. Screenshot of the Android application interface, with an example query image, string of extracted letters, additional inputted letter, and generated list of possible words.

The code for the MATLAB implementation of the image processing algorithm, as well as a video demonstration, are available in the supplementary information.

Although locally adaptive thresholding and Hu moments were implemented in the final algorithm, a variety of alternative methods were explored. These other methods are detailed in Appendix C.

III. Android Application Interface

Implementing the image processing algorithm on Android makes the system practical for real-time use during a game of Bananagrams. The code for the application, as well as a video demonstration, are available in the supplementary information.

A. *Image Acquisition*

The Android application allows the user to take a picture of his or her tiles using the phone camera.

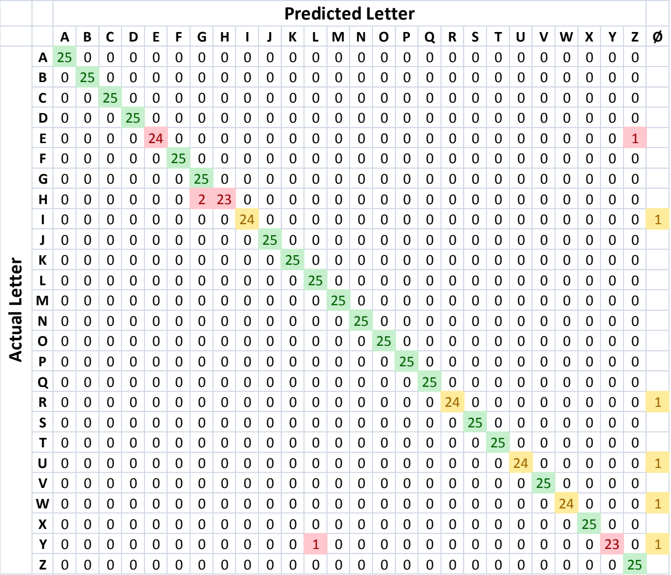
B. *Server Communication*

The picture is sent as a query image to the server, which processes the image using the algorithm described previously. The server returns two pieces of information: the processed image and a text file with the string of extracted letters [7]. A screenshot of the phone interface is shown in Fig. 3. The processed image is displayed on the right side of the application layout. The string is displayed above the image, which allows the user to verify if the letters were correctly recognized.

C. *Word Suggestion*

The user is allowed to input additional letters on the left side of the application. During gameplay, words must be constructed around existing words in the crossword puzzle; this feature ensures that the user can add the letter(s) he or she would like to build off of. Upon pressing the "Find Words" button, the application sends the query string and any additional letters to an anagram-solving website. The website's results are parsed into a scrollable text box on the bottom left of the application view. With these results, the player can quickly decide which word to add to the board with his or her tiles.

There were some challenges in the development of the Android application; the details can be found in Appendix D.



		Predicted Letter																								
		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	Ø
Actual Letter	A	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	B	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	C	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	D	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	E	0	0	0	0	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
	F	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	G	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	H	0	0	0	0	0	0	2	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	I	0	0	0	0	0	0	0	0	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	J	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	K	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	L	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	M	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	
	N	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	
	O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	
	P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	
	Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	0	0	0	0	0	0	0	0	1
	S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	
	T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	
	U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	0	0	0	0	0	1
	V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	
	W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	0	0	0	1
	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	
	Y	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	23	0	1
	Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	

Fig. 4. Confusion matrix showing letter misclassifications with the test images. The column "Ø" corresponds to an error when a tile was not extracted due to uneven illumination. The algorithm was able to correctly recognize each letter most of the time, although small similarities in Hu moments occasionally led to misclassification of E, H, and Y. Red boxes indicate where letters were misclassified, yellow boxes indicate where tiles were not detected, and green boxes indicate letters recognized with 100% accuracy.

IV. Results

25 test images were taken for each letter tile. The image processing algorithm was performed on each image. Letter recognition accuracy (the number of correct matches divided by 25) was determined for each letter. Although most letters were recognized with 100% accuracy, a few were not. These are highlighted in the confusion matrix in Fig. 4.

The algorithm was able to correctly recognize letters most of the time. There were several mismatches with letters E, H, and Y due to unexpected similarities between Hu moments of the extracted tiles and incorrect database letters. It is important to note that I, R, U, W, and Y each had one failed detection due to uneven illumination. In addition, one non-tile region was detected in each of four different images, also due to uneven illumination.

V. Discussion

The first four Hu moments, as calculated with equations (5) – (8), are calculated from normalized central moments. The central moments contain information about the distribution of white pixels along the x- and y-directions of the binarized tile images. These distributions are approximations of 1-D probability distribution functions (pdfs) in each of the coordinate directions. Changes in the pdfs affect the central moment values, and thus the Hu moment values. If the L_1 distance between an extracted tile and an incorrect template tile is smaller than the L_1 distance between the extracted tile and the correct template tile, then a misclassification can occur. This explains some of the results in Fig. 4.

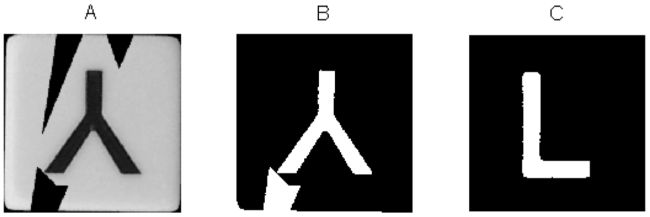


Fig. 5. The extracted Y tile from the test image where Y was misclassified as L. A) The extracted grayscale tile for the letter Y. The original image was binarized using locally adaptive thresholding. Some portions of the tile were assigned to background due of low variance within a window. B) A small portion of the unfilled tile background was treated as part of the letter Y due to 8-pixel neighborhood connectivity. C) Including the extra region alters the Hu moments and throws off letter classification. The extracted Y tile is therefore misclassified as an L.

For example, in one test image the letter E was misclassified as Z. It seems plausible that this could occasionally occur due to morphological similarities between the letters: they both have long horizontal lines at the top and bottom, which could lead to similar central and Hu moments. The first Hu moment is the sum of the two normalized central moments η_{20} and η_{02}, which represent the variance of the letter in the x- and y-directions. This Hu moment is therefore affected by changes in the 1-D letter pdfs. In this example, the first Hu moment is 0.4516 for the extracted tile, 0.4156 for the E template, and 0.4409 for the Z template. The first Hu moment of the Z template is closer in value to that of the extracted tile. This helps explain the misclassification, because the L_1 distance between the extracted tile and the template Z is smaller than that of the extracted tile and the template E.

Another more complicated example is one instance where Y is misclassified as L, which is detailed in Fig. 5. There is no clear shape similarity that leads to similar Hu moments, but further examination of the particular test image is illuminating. The locally adaptive thresholding during tile extraction leads to a few areas of the tile being assigned to background because of low window variance (Fig. 5A). Most of these are subsequently filled because the largest white section of the region is selected as the letter. Unfortunately, in this case, one of the side sections is connected to the letter by an 8-pixel neighborhood. It is therefore included in the extracted tile and is used when calculating the Hu moments (Fig. 5B). This changes the 1-D probability distributions of the letter, and thus its central and Hu moments. For the extracted tile, the first Hu moment is 0.5878. The first Hu moment for the Y template is 0.4430, while the first Hu moment for the L template is 0.5226. The first Hu moment for the L is closer in value to the first Hu moment of the extracted tile, and explains the misclassification in this case (Fig. 5C).

VI. Future Development

The current Android application performs robustly when tile images meet the following criteria:

- Tiles are placed on a dark, non-reflective surface.
- Tile images are taken *en face*, without perspective distortions.

- The camera flash is not used to maintain high image contrast.
- The tiles are evenly illuminated.
- The letters on the tiles are in focus.

However, it is desirable to make the algorithm more robust to perspective distortions, uneven illumination, and low image contrast. Locally adaptive thresholding significantly improves performance under uneven illumination compared to other methods (as described in Appendix C), but could potentially be refined by using smaller windows and/or adaptive variance thresholds. Robustness to perspective distortions can be implemented by estimating homographies between extracted and template tiles. This mapping could be calculated by comparing corresponding FAST keypoints between the extracted tiles and the templates [8,9]. SIFT keypoints would not work well, as we found during development, because the tiles do not have enough texture to generate a large number of keypoints [8,10].

As seen in the supplementary video demonstration of the Android application, the application currently has trouble controlling the camera focus. As a result, the query images are blurry and letters are misclassified. Implementing either an auto-focusing method or a user-dependent focusing method would ensure accurate letter recognition by the image processing algorithm. Additionally, the speed of the application could be improved by performing image processing on the phone instead of on the server. Although the application currently returns results in under four seconds, the processing time is limited by the phone's ability to contact the server. Implementing image processing on the phone would require writing functions in Java/C++ and using the OpenCV function (instead of the MATLAB function) to calculate Hu moments. The speed of the application could also be improved by caching previous results on the phone. The anagram website query step can be skipped if a previous set of letters has been queried before.

ACKNOWLEDGMENT

The authors would like to thank Huizhong Chen for his advice in designing the image processing algorithm, David Chen for his help in designing and implementing the Android application, and Professor Bernd Girod for teaching the EE 368 course in Winter 13-14.

REFERENCES

[1] B. Girod, "Image segmentation" [Lecture notes]. January, 2014.

[2] B. Girod, "Edge detection" [Lecture notes]. February, 2014.

[3] P. V. C. Hough, "Method and means for recognizing complex patterns," US Patent 3 069 654, Dec 18, 1962.

[4] M. Hu, "Visual pattern recognition by moment invariants," IRE Trans. Inf. Theory, vol. 8, no. 2, pp. 179-187, February 1962.

[5] M. AlHourani. (2011, May 6). *Hu's Seven Moments Invariant (Matlab Code for invmoments.m)* [Blog post]. Available: http://limitless-thoughts.blogspot.com/2011/05/hus-seven-moments-invariant-matlab-code.html

[6] "EE 368 Homework 7: Problem 3," unpublished.

[7] D. Chen, "Mobile image processing" [Lecture notes]. January, 2014.

[8] B. Girod, "Feature-based image matching" [Lecture notes]. February, 2014.

[9] B. Girod, "Keypoint detection" [Lecture notes]. February, 2014.

[10] B. Girod, "Scale-space image processing" [Lecture notes]. February, 2014.

Appendix A: Contributions

A. Steven Leung

Steven implemented the pre-processing and letter recognition stages of the algorithm in MATLAB. He also implemented the application's anagram solving component in Android.

B. Steffi Perkins

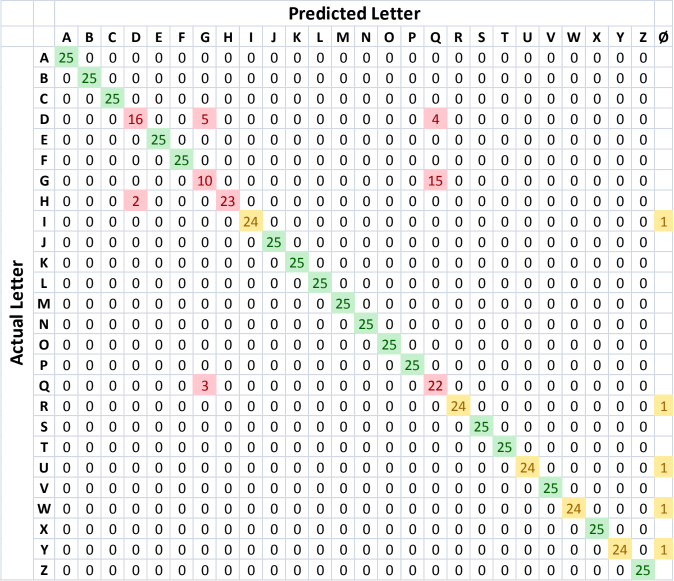
Steffi implemented the tile extraction stage of the algorithm in MATLAB and created a database of template and test images. She also characterized the accuracy of the algorithm.

C. Colleen Rhoades

Colleen developed a prototype of the Android application. She designed an interface that could capture an image with the phone camera, query a local server that she set up, and display the results.

Appendix B: Justification for Filling Holes in Letters

An initial attempt of Hu moment-based letter recognition (with unfilled letter holes) was not very accurate, as shown in the confusion matrix in Fig. B1. The letter G was often misclassified as the letter Q and the letter D was misclassified as G or Q. This occurred when the L_1 distance between the extracted G and the Q template was smaller than the L_1 distance between the two Gs—a result of inconsistent Hu moments due to inter-tile differences in letter shape.



		Predicted Letter																									
		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	Ø
Actual Letter	A	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	B	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	C	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	D	0	0	0	16	0	0	5	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	
	E	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	F	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	G	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0	
	H	0	0	0	2	0	0	0	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	I	0	0	0	0	0	0	0	0	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	J	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	K	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	L	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	M	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	
	N	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	
	O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	
	P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	
	Q	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	0	0	
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	0	0	0	0	0	0	0	0	1
	S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	0	
	T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	0	0	
	U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	0	0	0	0	0	1
	V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	
	W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	0	0	0	1
	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	
	Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	0	1
	Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	

Fig. B1. Confusion matrix showing letter misclassifications with the test images using unfilled letter holes. The column "Ø" corresponds to an error when a tile was not extracted due to uneven illumination. The algorithm had more misclassifications than when filled letters were used in calculating the Hu moments (Fig. 4). Red boxes indicate where letters were misclassified, yellow boxes indicate where tiles were not detected, and green indicates that a letter was recognized with 100% accuracy.

As outlined in the discussion, this misclassification can be explained in terms of normalized central moments, which represent the variance of the different letters in the x- and y-directions. When the distribution of white pixels for the letters G and Q are projected onto the x-axis, the circular shapes of both letters generate similar probability mass functions, as is shown in Fig. B2. The curved tail of the Q and the horizontal line in the middle of the G have similar contributions when

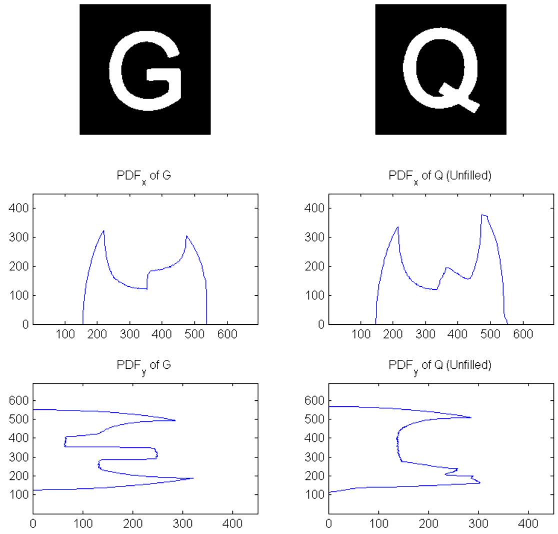


Fig. B2. Comparison of white pixel distributions along the x- and y-axes for the G and the unfilled Q. The distributions look similar, especially along the x-axis, which may lead to misclassification between the two letters.

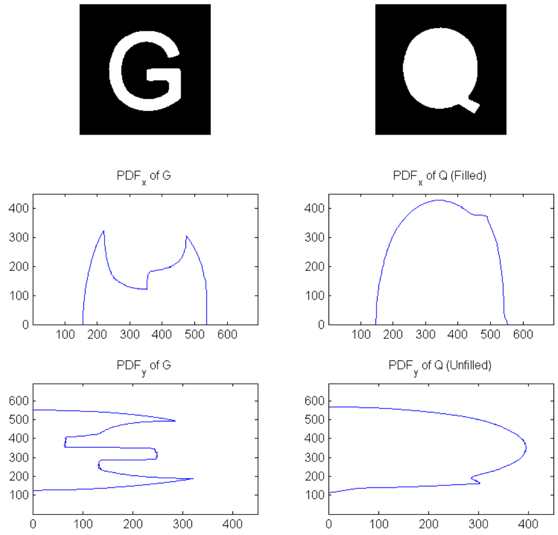


Fig. B3. Comparison of the white pixel distributions along the x- and y-axes for the G and the filled Q. The distributions look significantly different, and therefore the letters are more easily distinguished by the algorithm.

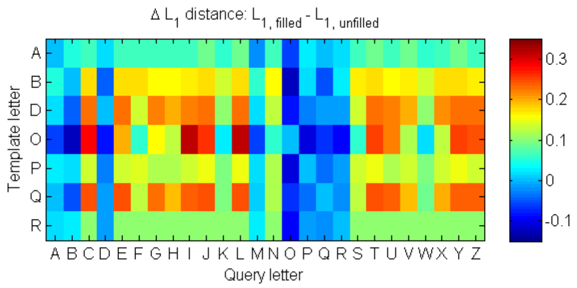


Fig. B4. Changes in L_1 distance resulting from filling letter holes. Filling the holes increased the L_1 distance between the letters with holes {A, B, D, O, P, Q, R} and other letters of the alphabet, which improved the accuracy of the algorithm. Although the L_1 distances decreased between letters within the set, this did not affect the accuracy of the algorithm in distinguishing between those letters.

projected onto the x-axis. Thus, both letters would have similar normalized central moments in the x-direction. Likewise, the normalized central moments in the y-direction would also be similar. Small structural differences in the extracted G (compared to the template G) could make its central moments more closely resemble the central moments of the Q template. This would influence the L_1 distance and may therefore also influence the best letter match.

We found that filling in letter holes avoided problems with central moment similarities, which can be seen in Fig. B3. In fact, filling in holes for the letter set {A, B, D, O, P, Q, R} increased L_1 distances between non-set letters, as shown in Fig. B4. However, L_1 distances decreased between letters within the set. Nevertheless, using this new method for Hu moment calculation, the algorithm did not return any misclassifications between letters in the set.

Appendix C: Investigation of Other Methods

We explored a variety of different methods for image processing before ultimately pursuing locally adaptive thresholding and Hu moments. These alternatives, along with their respective shortcomings, are detailed below.

D. *Pre-Processing*

1) Global Binarization

Global binarization with Otsu's method worked well only when the image background was much darker relative to the letter tiles. In addition, slight changes in scene illumination yielded a significant number of artifacts that hindered tile extraction.

2) Laplacian of Gaussian Edge Detector

Under restrictive conditions (dark background, *en face* image acquisition, and even illumination), we were able to get closed contours of tile edges. However, the method was unstable with uneven illumination.

3) Canny Edge Detector

Also worked under restrictive conditions but was unstable with uneven illumination. It was also difficult to select strong and weak edge threshold values that were robust across different imaging conditions.

4) Morphological Edge Detector

Tile edges were detected by subtracting the original grayscale image from its dilated image. However, the method failed when tiles were close to one another—image dilation caused the boundaries of two tiles to fuse. Problems also arose when images had uneven illumination.

5) Rank Filter Background Subtraction

The 10th ranked value was selected from 10 x 10 windows in order to "capture" uneven illumination on the image. However, the algorithm had a very long runtime and was ineffective in removing uneven illumination when the result was subtracted from the original image.

E. *Tile Extraction*

1) MSERs

Maximally Stable Extremal Regions worked well for a small subset of images given a set of input parameters. However, each set of input parameters was not robust across different imaging conditions. The method also failed for images with uneven illumination.

F. *Letter Recognition*

1) Hit-Miss Filter

We first attempted to perform letter detection using hit-miss filter recognition by erosion. However, variations in image size and tile alignment were difficult to account for. The method was also not invariant to scale and rotation, and was therefore quickly deemed impractical.

2) SIFT Descriptors

We next attempted to match SIFT feature descriptors between tiles and template images. However, very few SIFT keypoints were detected due to the lack of texture to the letter tiles; thus we were unable to perform RANSAC. Additionally, problems arose when images had uneven illumination.

Appendix D: Challenges with Android Programming

There were a number of technical obstacles surrounding implementation on the Android phone.

A. *Asynchronous Server Tasks*

Because server tasks were asynchronous, there were several issues regarding communication timing. The image taken by the camera was first processed on a local server using MATLAB. MATLAB saves the processed image as well as a text file containing the string of recognized letters. A second call to the server loads the saved image and string files into the phone display. In order for the algorithm to function in the correct order, the timing of these server calls had to be coordinated to address timing issues.

B. *Wi-Fi Connection*

The Android phone requires access to the anagram-solving website through a Wi-Fi or mobile data connection. However, the application did not work reliably on all wireless networks. Specifically, it works on the Stanford Residences network, but not on the Stanford network.

Supplementary Information

A. Leung_Perkins_Rhoades.wmv

This video demonstration shows how the image processing algorithm works on a sample test image in MATLAB. It also shows how the final Android application works. Letters were misclassified on the phone due to problems with the camera autofocus, which is addressed in the future development section of the paper.

B. LeungPerkinsRhoades_AndroidCode.zip

The code for the Android application is included in this archive.

C. LeungPerkinsRhoades_MatlabCode.zip

The code for the image processing algorithm in MATLAB is included in this archive.