CSCI 1430 Final Project Report: GameNight

Team name: Justin Barlas, Dan Wexler
Brown University

Abstract

We seek to recognize the board states of tic-tac-toe and connect-4 games from images and then return the best move for the current player. We use a combination of filtering, edge detection, contours, and other methods to extract the board states, with the exact methods varying for each game.

1. Introduction

The problem we aim to solve is the transformation of an image into a game state that can be analyzed with adversarial search algorithms such as minimax and alpha-beta pruning. The difficulty in this endeavor comes from identifying the key features within each image and translating them into something more useful. The identification of features should be resistant to noise and translation, which can be difficult. For tic-tac-toe, we use the approach of breaking the board into 9 distinct regions to check. We differentiate between a blank square, an X, and an O through calculating the solidity of the contour. For connect 4, we identified the board and its grid through a series of assumptions about the contours of the image.

2. Related Work

Numpy and cv2 are the two python packages we used for this project. Some general ideas for recognizing the board state for both tic-tac-toe and connect-4 were found online. More specifically, for tic-tac-toe we used ideas from this Stack Overflow post. This post discusses using large scale contours to determine the 9 different boxes of the tic-tac-toe board and then using solidity to differentiate between an "X" and an "O". For connect-4, we used ideas from this website. This website discusses using contours and the dimensions of contour polygons to find circles, and then color filtering to differentiate between red and yellow pieces.

3. Method

3.1. Tic-Tac-Toe

The intial image is resized to 600 by 600 pixels and then converted into grayscale. image:

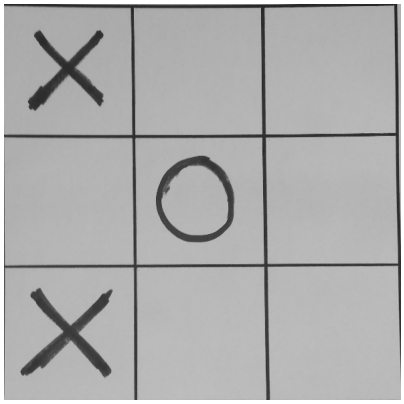

Figure 1. Tic-Tac-Toe Grayscale

This grayscale image is then put through a binary threshold.

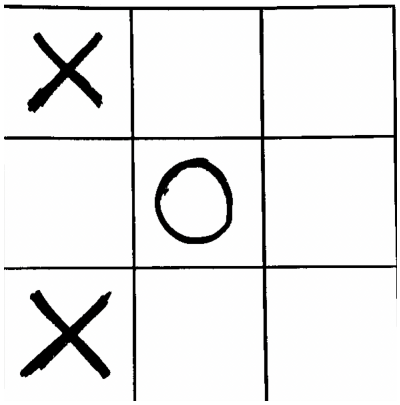

Figure 2. Tic-Tac-Toe Binary

This binary threshold converts all pixels in the grayscale image below a certain value (127) to black

and all pixels below this value to white. Canny edge detection is then run on this binary image.

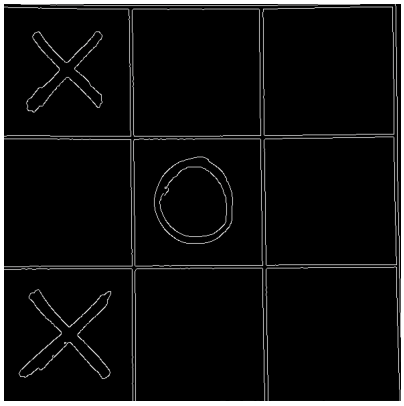

Figure 3. Tic-Tac-Toe Edge Detection

From the edge image, the rightmost, leftmost, lowest and highest pixels that are edges (are white) are found, and the coordinates of these pixels are found. A bounding rectangle is then drawn using these pixels as its corners. This bounding rectangle ideally contains the entire board within. The image is then cropped down to the bounding rectangle, leaving only the board in the image.

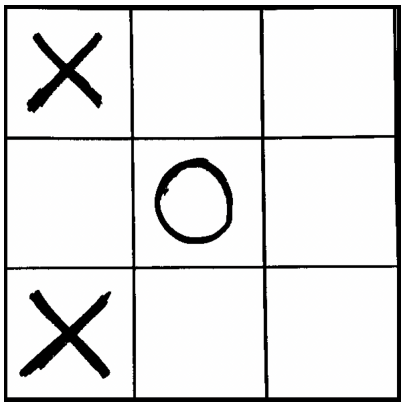

Figure 4. Tic-Tac-Toe

From this cropped image with a bounding rectangle, the large-scale contours are found. The parameter cv2.RETR-EXTERNAL is passed into the contour method, which changes the "hierarchy" of contours returned to just the outermost contours (contours that are not contained within other contours). The parameter cv2.CHAIN-APPROX-SIMPLE is also passed into the contour method as opposed to cv2.CHAIN-APPROX-NONE. cv2.CHAIN-APPROX-SIMPLE saves memory by only returning a few points from the contour that define its boundaries, as opposed to every point that lies along the edge of the contour. Because only the outer contours are found, this call of the contour method should return at least 9 contours (sometimes more) that represent the squares of the tic-tac-toe board.

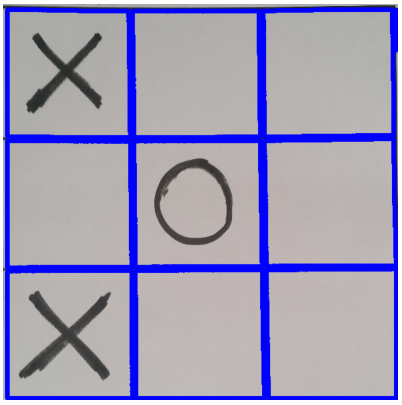

Figure 5. Tic-Tac-Toe Contours

We loop through all of the contours and check if their size is above 10000. If their size is not above 10000, it is probably not a square of the tic-tac-toe board and instead a random extraneous contour. Within each contour above size 10000 (square on the board) we perform a number of operations. First, we determine where on the board this contour corresponds to by dividing the x and y position of the contour by the width or height of the contour, respectively. Next, we work on finding out whether the box contains nothing, an "X", or an "O".

To find what exactly is in each box, we find all of the contours within the box. This time, the parameter cv2.RETR-LIST is passed in, as opposed to cv2.RETR-EXTERNAL which was used earlier. cv2.RETR-LIST returns all of the contours in an image. We can used this instead of cv2.RETR-EXTERNAL because we expect there to only be one or zero things within the image we are checking and therefore do not care whether contours are nested inside one another.

Figure 6. Tic-Tac-Toe Individual Box

We then perform another size check - if the contour size is below 10000 and above 1000, we are confident that it is an "X" or an "O". Now we must differentiate

between X's and O's. To do this, we use solidity, a property of all contours. The solidity of a contour is defined as the ratio of the contours area to the ratio of the contours convex hull. The area of the contour is simple - it is simply the area contained within the contour. The convex hull can be thought of as the minimum perimeter surrounding all of the points within the contour. A good analogy for the convex hull of a figure is that if you were to stretch a rubber band over a series of points and then let go, the area of the convex hull would be the area contained within this rubber band. O's, for example, have a solidity that is near to 1, because their area the the area of the convex hull are essentially the same. X's, on the other hand, have a much lower solidity, because their area is much smaller than the area of the square that their convex hull defines.

So, knowing the solidity and knowing the place on the board that the current square corresponds to, we can place determine the board state. This board state is then passed into a min-max algorithm, and the next best moved is found and placed on the board. This program also checks to make sure the game is over, and works if there are no moves yet made. The program assumes that X always goes first.

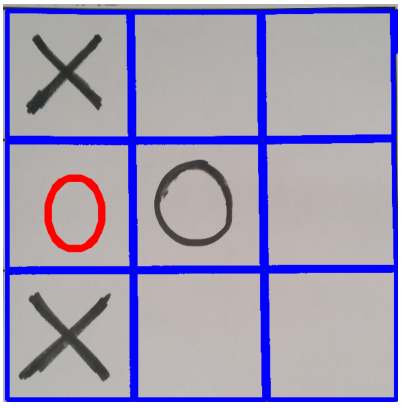

Figure 7. Tic-Tac-Toe Result

Figure 8. Connect 4 Input Image

A bilateral filter is applied to the image to reduce noise but preserve corners. The preservation of edges in this step is important, as edges are crucial to identifying the contours of the board and grid.

Figure 9. Bilateral Filter Applied

3.2. Connect 4

This section will discuss the method of solving Connect 4 with a given input image:

Next, we outline the edges of the filtered image using Canny algorithm.

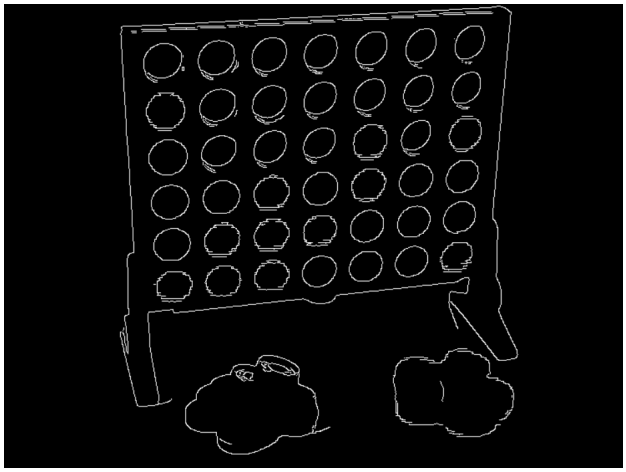

Figure 10. Edges calculated using Canny algorithm

We then use the image of the edges to identify contours using cv2.findContours to determine different parts of the board. After finding all of the contours, we identify the board through returning the largest four-vertexed contour. We then search within the bounding rectangle of the board for contours that fit a number of criteria to find each of the circular grid points on the board. We filter contours on the following:

1. Within the board region
2. Reasonable size relative to area of board
   (a) No larger than the area of the board divided by the product of the number of rows and the number of columns
   (b) No smaller than one fourth of the upper bound
3. Bounding rectangle of the contour is roughly square (calculated based on the ratio of width to height of the bounding rectangle)
4. Reasonable area-of-contour to area-of-bounding rectangle ratio to ensure that the contour takes up roughly as much space as a circle should within the bounding rectangle (the lower bound for this number is low enough to account for contours seen from an angle which would have more of an elliptical shape)

The identified features are outlined in the figure below:

Figure 11. Board and pieces identified

Next, we used the spacing between two adjacent pieces from each bottom row and rightmost row to create two lines. We calculated the intersection of the lines to approximate the center of the bottom right-hand tile. From this point, using the slopes of the lines, we populate a board grid aiming to approximate the centers of each piece on the board.

Figure 12. Board Grid Estimation

Next, we filtered the image within a red and yellow color range and compared our estimated grid to these images to determine piece value (red, yellow, empty).

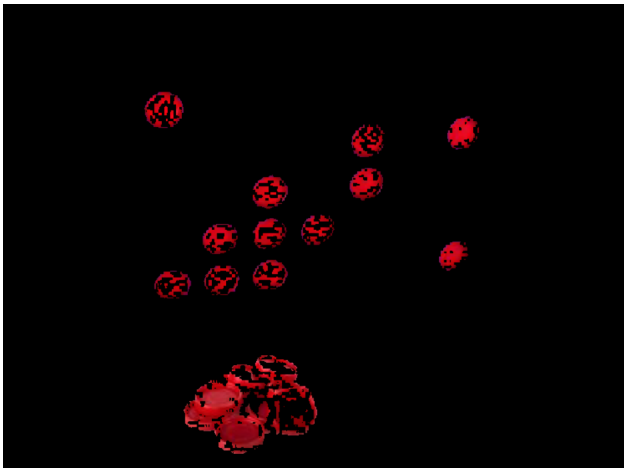

Figure 13. Red Mask Filter

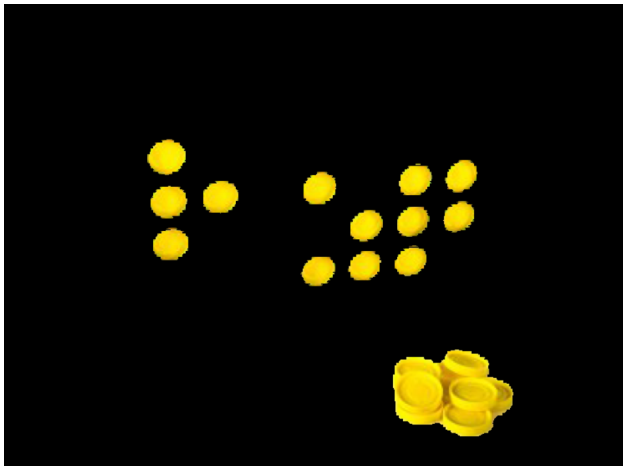

Figure 14. Yellow Mask Filter

Using these estimations, we were able to generate a game state from grid and piece value, which we turned into a numpy array. This array could then be passed into an alpha-beta pruning algorithm to determine the best move for the player. We assume that yellow always goes first in determining whose move it is (i.e. if the number of red and yellow pieces are equal $\implies$ yellow's turn). The move returned is a tuple of the column in which to place a piece and the player index of who is to play the move.

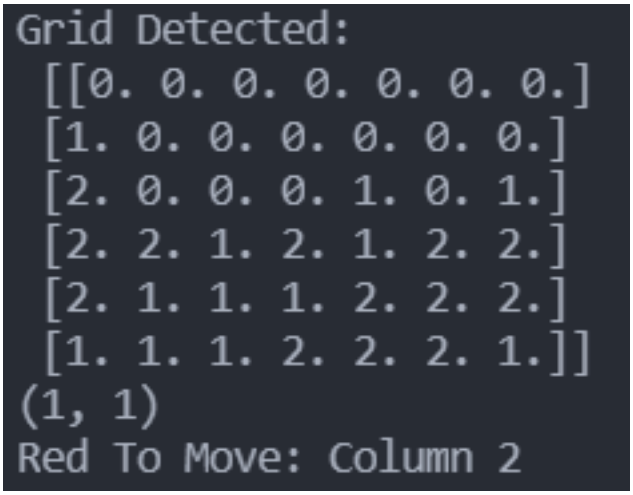

Figure 15. Array of board estimation and move output

We then output best move overlaid on original image using the move output from the alpha-beta pruning algorithm and the board's grid estimation:

Figure 16. Output Image

4. Results

Present the results of the changes. Include code snippets (just interesting things), figures (Figures **??** and **??**), and tables (Table **??**). Assess computational performance, accuracy performance, etc. Further, feel free to show screenshots, images; videos will have to be uploaded separately to Gradescope in a zip. Use whatever you need.

4.1. Tic-Tac-Toe

The tic-tac-toe program is able to correctly identify the state of the game if the input image is head-on and without too much noise (e.g. shadows, reflections,

extraneous lines and markings). The images below are of successful outputs of the program.

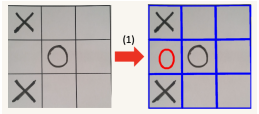

Figure 17. Tic-Tac-Toe Correct Result

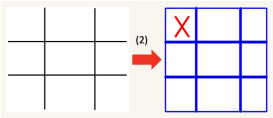

Figure 18. Tic-Tac-Toe Correct Result

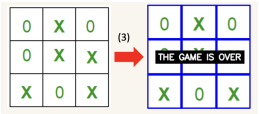

Figure 19. Tic-Tac-Toe Correct Result

If the drawn board is messy, or if there are shadows or reflections in the image, the program fails. This is due to a number of reasons. One of the reasons why it fails on images with noise is that the edge detection is used to get the bounding box of the board, which is then used to find the contours. If there is a lot of noise outside of the board, the Canny edge detection will pick this up, and the bounding box will include this noise. This could be improved upon in the future by only looking at edges that line up with contours with large areas when defining the bounding box. Also, in the figure below, the board state is incorrectly identified because the wiggly line in the top left of the board is seen as an "O" contour in the top left box. Wobbly lines could be accounted for in the future by allowing for more fluid bounding boxes for the individual squares, instead of just rectangles.

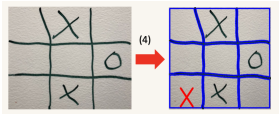

Figure 20. Tic-Tac-Toe Incorrect Result

Also, in the future, it would be nice to make the program work for pictures of tic-tac-toe boards taken at angles. As it stands right now, the program works very well for boards with little noise.

4.2. Connect 4

The Connect 4 program is able to correctly identify the state of the game for some images, but has difficulty with others. It struggles with images with lots of glare since it cannot correctly identify the board and grid from the contours. There also is an issue with the population of the grid from the features since images taken from the right-hand side of the board are also inaccurate despite the features being correctly identified. This is not an issue with the computer vision aspect of the project, as the features are able to be correctly identified, as shown in the following figure:

Figure 21. Correct Feature Identification

The issue arises when estimating the grid:

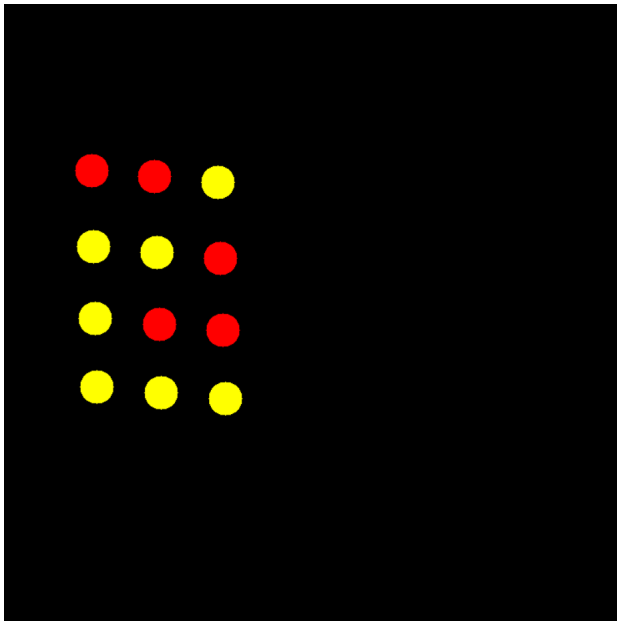

Figure 22. Incorrect Board Render

The resulting move generated from the image is a bit silly:

Figure 23. Correct Evaluation of Incorrect Game State

Overall, the methodology works well and is able to identify game states from some boards at an angle.

4.3. Technical Discussion

Neither tic-tac-toe nor Connect 4 are solved with the aid of a neural net. This design decision was two-fold. Firstly, we wanted to see if we could design a program to recognize game states solely using domain knowledge and feature detection techniques. Given known information about either game, we were able to search the images for specific features using simple methods such as filtering, edge detection and finding contours. A second reason for the decision is that it is extremely difficult to find training data. Not only would we require many images of differing boards, we would also have to know the game state of each board, which would be very time consuming to do by hand. Since the accuracy of a neural net is majorly influenced by the training data, it would not serve us well to have a poor training set. The trade off of not using a neural net is that it may be less accurate across many types of images and also requires some assumptions to hold up. Due to the reliance on detecting contours, as seen in the results, these programs fail to determine game states of images that are too noisy.

Our method could actually be useful for providing neural nets with information. For example, if you wanted to train a network and required an image of a connect four board and its game state data, you could pass the image through our program first, then provide the network with both the original image and the game state determined by our program. This ability could aid in the future training of networks involved in similar tasks.

4.4. Societal Discussion

1. The context of our project is that games are meant to played for fun, and that our tool can be used for players of different skill levels to either analyze their game or learn the best strategies for a game. For example, a skilled player might use this tool to analyze their game and see if a move they made was the best move. A less skilled player, however, might use this tool to learn the best strategies for playing a game. This tool, however, could also be used for cheating. People cheating at games for pride or to win money is nothing new, especially in the context of online video games. It is often very easy to cheat in online games, and this tool could be used to cheat in online tic-tac-toe or online connect-4. This article discusses cheating in video games in more detail.

2. The major stakeholders in this project are people who enjoy playing games for fun or in a competitive setting. Our relationship to these stakeholders is that we too enjoy playing games, both for fun and competitively. Less skilled players may both benefit and be harmed by this project. The will benefit by being able to use this tool to learn the game, but they made be harmed in that this tool could be

used against them in the form of cheating. Veteran players can use this tool to better their game, but they might also use it to cheat.

3. Identifying the board states of games from images is not a new field. Existing work on this topic has resulted in effective programs that are able to identify the board states of a number of games. The societal impact of this work is similar to that of our own work, in that it is small but can help new and veteran players alike learn how to improve at a certain game. The existence of work in this field means that we should not talk about our work as if it is completely new and/or groundbreaking.

4. How could an individual or particular community's civil rights or civil liberties (such as privacy) be affected by your project?

   This project does not have large effects on the civil rights or liberties on a particular community. The largest implication that a project such as this may have on society at large is if it is used for cheating.

5. We did not use "data" in the traditional sense of a large number of images of a certain thing. The only data we collected were online images of tic-tac-toe boards and connect-4 boards, and the collection of these images does not have any obvious bias against any specific group. All of the images used are readily available on the internet.

5. Conclusion

We wrote a program that can identify the board state of connect-4 and tic-tac-toe games from images. These programs work better on some images than others. For example, if the images are taken from angles or if there is a lot of noise in the image, the programs are more likely to fail. There are a number of ways we could improve upon the algorithms we wrote to better account for different images. The algorithms we wrote do work relatively well on images without lots of noise. The impact of this project is small, but it can hopefully be used by new or veteran players of these games to help learn the best strategies or analyze their game.

References

1. https://stackoverflow.com/questions/53684677/identifying-state-of-tic-tac-toe-board-from-image

2. "Contour Properties." OpenCV, https://docs.opencv.org/3.4/d1/d32/tutorial_py_contour_properties.html.

3. https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contour_features/py_contour_features.html

4. https://mattjennings.co.uk/portfolio/Connect-Four%20Computer%20Vision%20A.I/

Appendix

Team contributions

Dan Wexler I wrote the tic-tac-toe side of the project.

Justin Barlas I wrote the connect 4 side of the project and provided the algorithms used to solve tic-tac-toe and connect 4. The algorithms were previously created as an assignment for the course Artificial Intelligence, but were slightly adapted to fit our needs in this project.