Computer Vision Laboratories

Project 3

EE/CPE 428 - 03

Computer Vision

Winter 2023

Group 1

Students: Fadi Alzammar, Eitan Klass, Nathan Jaggers

Project Due: 02/09/23

Instructor: Dr. Zhang

Part A – Finding Stones

Assignment

The following image displays Go (Wei Qi), a Chinese board game with white and black stones on the grid. Use template matching to find as many white and black stones in the image as you can. To create a stone template, you can crop one stone from the image.

1. *Apply template matching to find the stones in the image. Display the correlation map and the detected stones in a bounding box overlaid on the original image. Discuss the detection results.*
2. *Discuss the conditions under which template matching works the best.*

Procedure

Use words/sample_code to describe the procedures (How you did it). Explain the steps taken if needed.

To find stone pieces on the Go board, we first need to create templates for the pieces from the original image. To accomplish this, the original image is grayscaled, then isolated white and black pieces are cropped out and saved to serve as templates. For the templates it was important to find pieces that were at the same angle as the other pieces so that MATLAB would be able to correctly identify them.

```
GoBlackPiece = imcrop(GoBoard, [622.5 296.5 50 36]);
GoWhitePiece = imcrop(GoBoard, [691.5 54.5 44 29]);
```

"imcrop()" Function

Apple's snipping tool to capture the templates were not sufficiently accurate and therefore MATLAB's "imcrop()" was used instead.

The built-in Matlab function normxcorr2() was used with the template and the grayscale image as inputs to create the correlation maps for black and white pieces. The correlation maps are used to locate the pieces and determine where to put the bounding boxes. After a lot of testing, a threshold of 0.6 was used to threshold the correlation map.

```
threshold = 0.6;
   resultBlackThresh = ((resultBlack > threshold) * maxResultBlack) .* resultBlack;
   resultWhiteThresh = ((resultWhite > threshold) * maxResultWhite) .* resultWhite;
```

Method of thresholding the correlation map

The method used to threshold was to find all the values of the "normxcorr2()" output that was above the threshold and then multiply it by the max value of the output, which was 1. Then that value was multiplied by every value in the "resultBlack" matrix. MATLAB's "imregionalmax()" was used to get the peak values of each piece on the board. This specific function was used since some peak values that were identified were part of the same piece, thus it was unnecessary to use it. Peak values correlate to the location of the piece and you can determine the offset from the center of the piece needed to circumscribe a box around it. Lastly, you can show the final image and draw the boxes for each piece located.

```
imshow(GoBoard);
   for i = 1:rowBlack
       for j = 1:colBlack
           if regMaxBlack(i, j) ~= 0
               xCoord = j;
               yCoord = i;
               xOffset = xCoord - size(GoBlackPieceG,2);
               yOffset = yCoord - size(GoBlackPieceG,1);
               rectangle('Position',[xOffset+1, yOffset+1, size(GoBlackPieceG,2),
size(GoBlackPieceG,1)],'EdgeColor','r');
           end
       end
   end
```

Method of drawing a box around each piece

Results

Show and evaluate the results. Use words to describe the results and provide explanations if needed.

The original image was grayscaled and from the grayscale image, the templates were generated. They can be seen below. The size of the original image and all derived images is 522 x 800 pixels. The size of the two templates created are both 28 x 39 pixels.

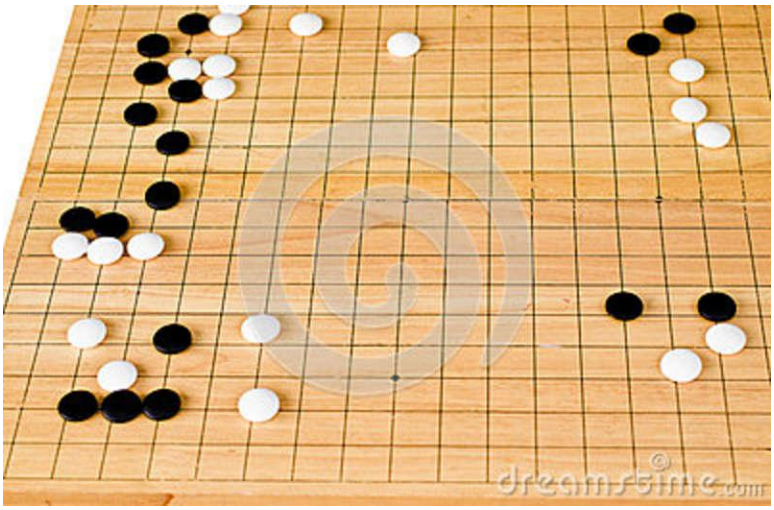


Figure 1: Original Go (Wei Qi) Image

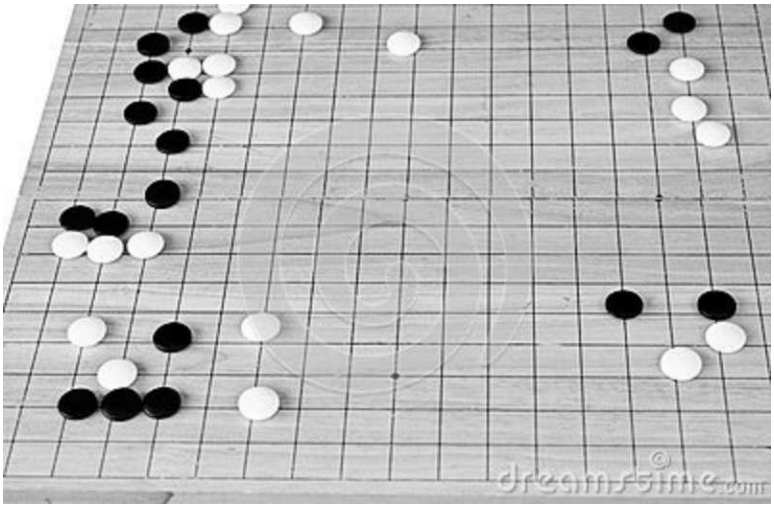
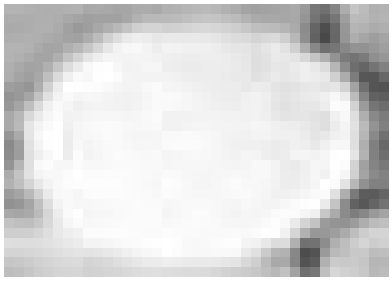


Figure 2: Grayscale Go (Wei Qi) Image



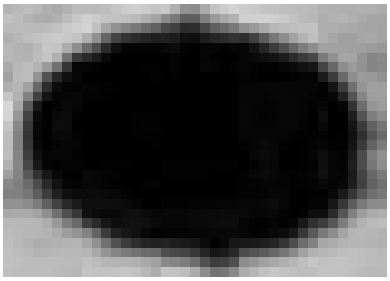


Figure 3 & 4: White Piece Template and Black Piece Template

The templates above were used to create the correlation maps. Then the correlation maps were used to create the final image with the bounding boxes around detected images. Note that in the templates above the pieces are not perfectly isolated and that there are lines from the board in the images as well. These will skew our results and detection slightly. We can see in the correlation maps below that we have many of our high intensities at the center of the pieces being detected but we also have some erroneous high intensities around the board at the line crossings.

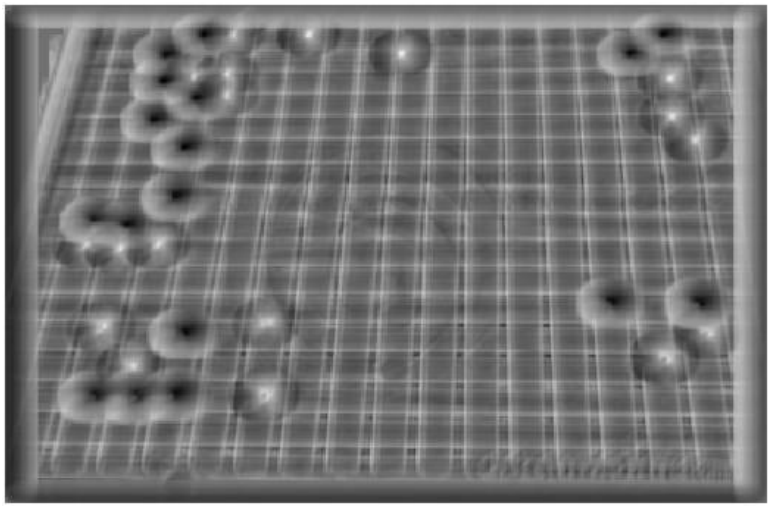


Figure 5: White Piece Correlation Map

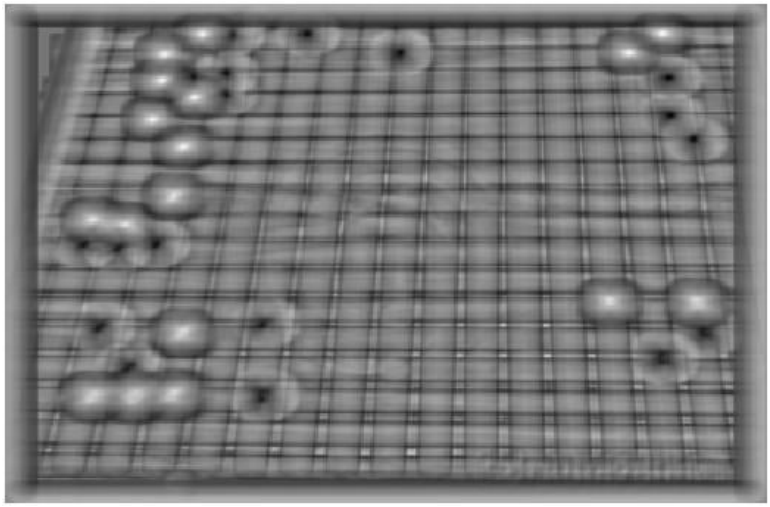


Figure 6: Black Piece Correlation Map

In the correlation maps above, values range from -1 to 1 where places of high correlation correspond to values closer to 1, places of low correlation correspond to values closer to 0 and places of anticorrelation correspond to values closer to -1. When displaying the image, we scale these results to the range 0-255. To determine if a piece is detected, we have a threshold value which is a percentage of the max value found in the correlation maps (max = 1). This threshold value can be adjusted to increase the amount of pieces detected at the risk of detecting the same piece multiple times or detecting something that isn't a piece all together. The results below are from experimentally determining the best threshold value where the most pieces were detected without boxing something that isn't a piece. In this case, the threshold was 0.62.

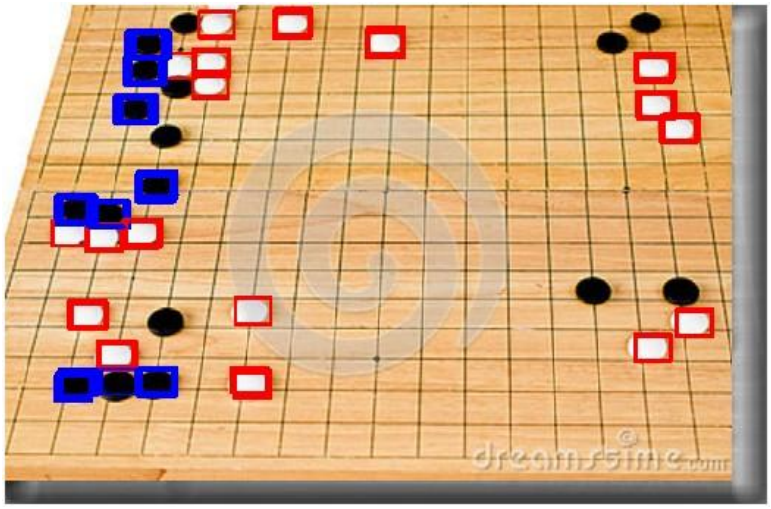


Figure 7: Original Image with Detected pieces and a threshold of 0.62

Above are the results from the first attempt at Part A. Improvements on the method of determining the detection of a piece were made through the addition of using a built-in Matlab regional max function. This allowed for the suppression of repeated detections on the same piece and for more accurate boxing around the center of a piece.

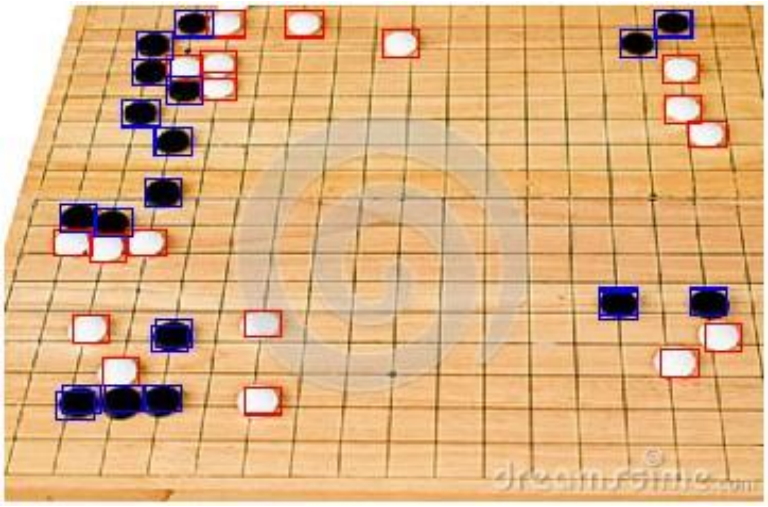


Figure 8: Original Image with Detected pieces and a threshold of 0.63

Part B – Finding Rock Paper Scissors

Assignment

Freeman et al. in paper "Computer Vision for Interactive Computer Graphics" [1] introduced an interface where an open hand turns the television on. Read the paper, and in particular, the sections on "Small object tracking".

Design a proof-of-concept experiment in which template matching is used to identify three hand shapes – rock, paper, or scissors, in a given image. Note, the templates and test images showing the hand shapes should come from different persons.

Procedure

Use words/sample_code to describe the procedures (How you did it). Explain the steps taken if needed.

1. Prepare template images: Take pictures of each of the three hand shapes (rock, paper, and scissors) and save them as templates.
2. Load the test image: Load the test image that contains the hand shape that you want to identify into MATLAB.
3. Pre-processing: Perform pre-processing on the test image, such as converting it to grayscale and filtering out noise, to improve the accuracy of the template matching.
4. Perform template matching: Use the "normxcorr2()" function in MATLAB to perform template matching between the test image and each of the three templates. This function returns a matrix that represents the correlation between the template and the test image. The closer it is 1, the more correlation there is.

5. Find the location of the maximum correlation: Find the location of the maximum correlation in each of the three matrices returned by the normxcorr2 function. The location of the maximum correlation indicates the location of the hand shape in the test image. The built-in function “imregionalmax()” accomplishes this.
6. Identify the hand shape: Based on the location of the maximum correlation, identify the hand shape as rock, paper, or scissors.
7. Validation: Validate the results by manually checking the identified hand shape against the actual hand shape in the test image.
8. Repeat the experiment: Repeat the experiment on a set of test images to see if the template matching approach can identify the hand shapes accurately in different images.

Results

Show and evaluate the results. Use words to describe the results and provide explanations if needed.

Though the procedure here was very similar to that of Part A, the results were very different. Just as before, we used a portion of one image to find similar matches, this time in other images rather than the same one. Due to the variance in our hand sizes, distances and angle from the camera, and skin tones, the detection was significantly less reliable than in the Go example.

Figure 8 shows the image we used to create templates for the various hand positions. These templates were then applied across images where all participants displayed the same hand position. Figure 9, showing rock, was found with a correlation threshold of 0.72. In order for all

three hands to be detected, the threshold had to be lowered to 0.62, producing Figure 10 with a significant number of false positives at the bottom of the image.

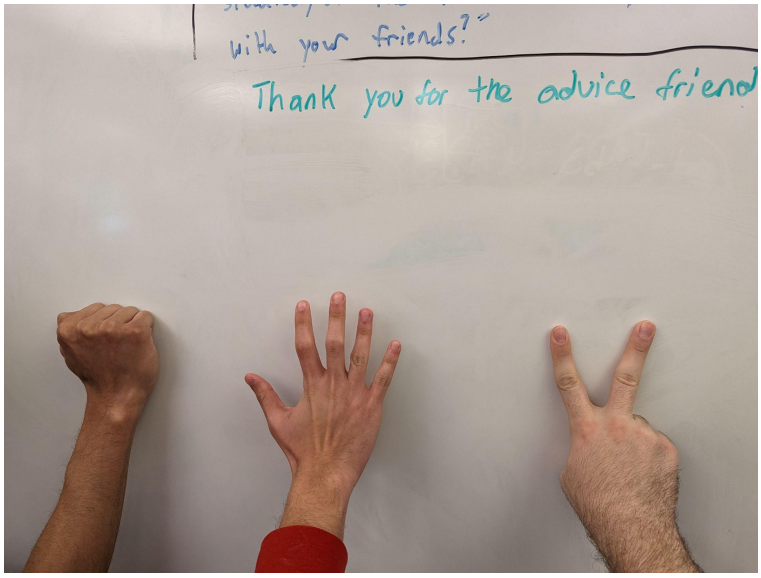


Figure 8: Hand detection templates

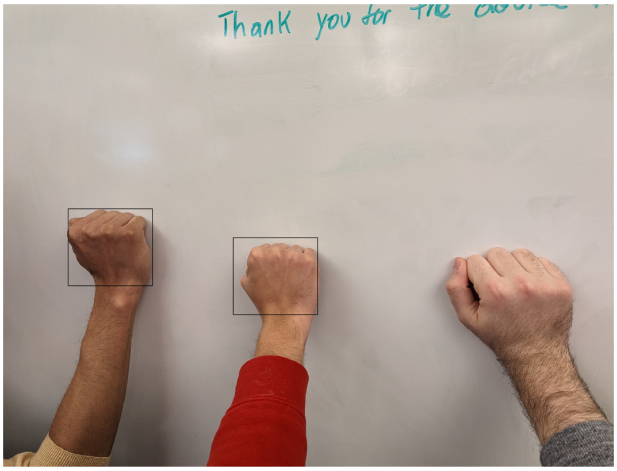


Figure 9: Detecting rock with a correlation threshold of 0.72

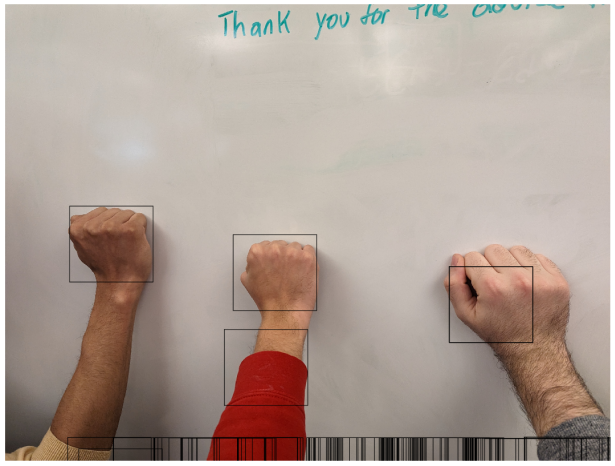


Figure 10: Detecting rock with a correlation threshold of 0.62

The results for paper proved to be very similar, producing significant false positives when a low enough threshold was selected to detect all hands. For paper, the maximum threshold to find two hands, that is, the same hand used for the template and one other group member's was 0.69 as shown in Figure 11. In order to detect all three hands as in Figure 12, a threshold of 0.62 was required.

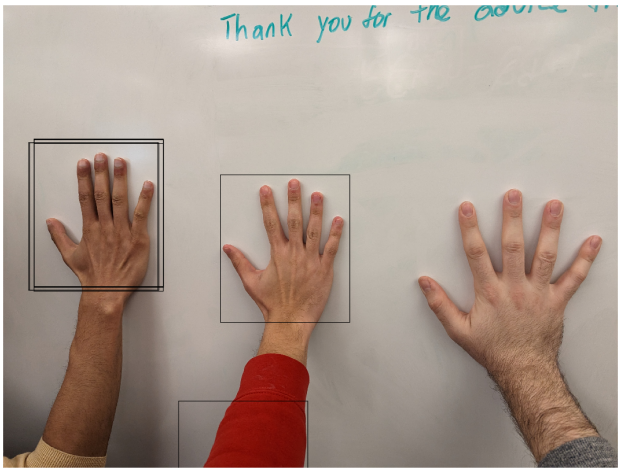


Figure 11: Detecting paper with a correlation threshold of 0.69

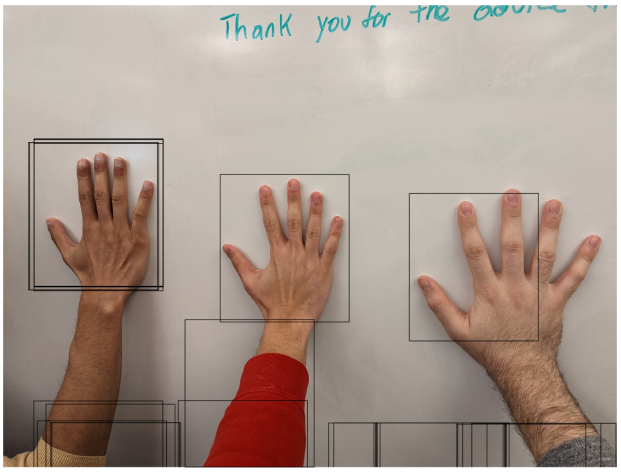


Figure 12: Detecting paper with a correlation threshold of 0.62

Scissors was the most challenging hand position to detect. A threshold of 0.6 was required to get only two hands to be detected in Figure 13, which already produced an unacceptable number of false positives on the whiteboard. With a threshold of 0.48, all three hands were finally detected as displayed in Figure 14, though the results are essentially unusable.

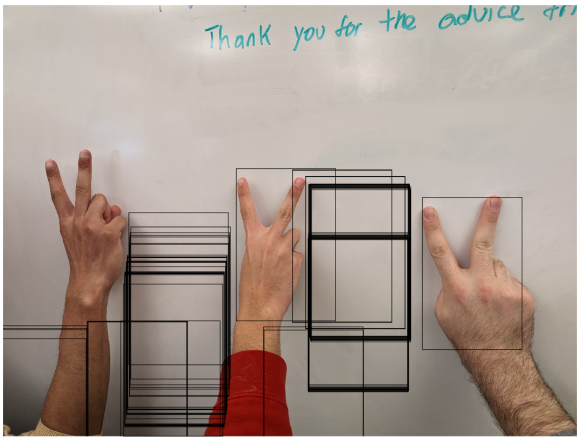


Figure 13: Detecting scissors with a correlation threshold of 0.6

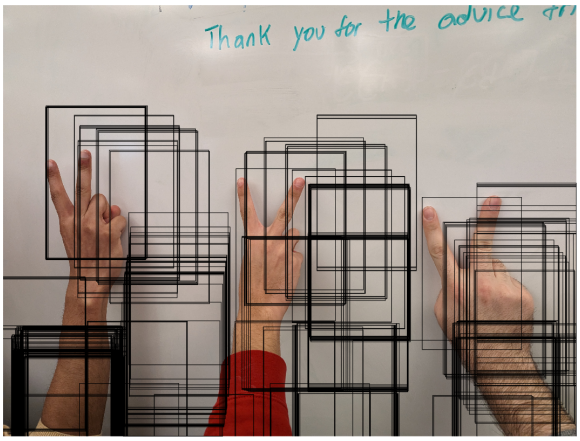


Figure 13: Detecting scissors with a correlation threshold of 0.48

Because of the variance in each hand, it is clear that further processing, such as feature mapping, is necessary to properly detect hand signs. In addition, the problem observed in the Go pieces where bounding boxes were incorrectly sized due to simply following that of the template was emphasized with the differences in hand size.

Reflection

Fadi

This project gives us a taste of the power computer vision systems have. The ability to identify and match known objects has many obvious applications, including in automated manufacturing and self-driving. That said, the algorithms implemented in this lab are far from production-ready for most purposes, as especially indicated in Part B. Basic template matching does not account for rotation, translation, or lighting differences in images, all of which can be expected in most real-world applications. This project provided good insight in the implementation details of template matching, with creating correlation maps and finding local maxima to detect matches, and the next project on feature mapping seems very promising in its ability to address the shortcomings found here.

Eitan

Although template matching is an incredibly powerful technique in Computer Vision, the limitations of it became very apparent with this project. In part B, the output of the code was less accurate with Nathan's hand as when we took the picture, the angle of his "scissor" hand orientation was slightly different. This goes to show that for template matching to be accurate, the optimal conditions need to be satisfied as well as there needs to be minimal differences between the template and an image. Another example of this was in part A when I chose which piece to crop as the template. Since the picture was taken at an angle, it made each piece have a slightly different shape. Therefore, if there was an apparent difference between the template and the current piece, it would miss it.

Nathan

This laboratory has been a great exploration of the concepts of templates and correlation as well as the capabilities of template matching. On one hand, this can be a very simple and effective solution if the environment allows it. We do not need a fancy neural network to be trained and do image processing for us so that we can identify similar objects in an image. We can just use the techniques explored in this lab and put together an effective and efficient solution. The downside to this is that the environment has to be very well-behaved for this to work well. As we see in the paper mentioned in Part B as well as from our own exercise in part B, variances like the angle and size of our hands can make a big difference in how easily our targets are detected. Another important note is that the selection of a good template itself is super important because that can make a huge difference in if a target is detected or not.

Teamwork

Fadi

- Wrote the Results for Part B
- Included Part B code
- Gathered contact info and organized Work Meeting

Eitan

- Found improvements for detection algorithm in Part A
- Included Improved Part A code
- Added details to Part A and B

Nathan

- Set up and reviewed document
- Included Part A code
- Wrote Part A procedure and results

Appendix

Part A - Finding Stones

```
%% Part A
close all;
clear;
clc;

%%
% load image
bg = imread("go1.jpg");
imshow(bg);
% grayscale image
gray = rgb2gray(bg);
imshow(gray);

%%
% crop pieces out of image
white = gray(28:55, 399:437);
black = gray(28:55, 649:687);
% show cropped images
figure('Name','Piece Templates')
subplot(121);
imshow(white);
subplot(122);
imshow(black);
figure;

%%
% Apply normalized correlation (matched filtering)
% and find other pieces on board
w_nc = normxcorr2(white,gray);
b_nc = normxcorr2(black,gray);
% figure, surf(ncw), shading flat;
% figure, surf(ncb), shading flat;
% correlation maps for white and black pieces
figure(2), imshow(w_nc,[]);
figure(3), imshow(b_nc,[]);

%%
% find location of maxes for pieces
% (we expect max to be 1 and min to be -1)
% play with threshold for different results!
thresh = 0.62;
w_thresh = thresh*max(w_nc(:));
b_thresh = thresh*max(b_nc(:));
[w_ypeak, w_xpeak] = find(w_nc>=w_thresh);
[b_ypeak, b_xpeak] = find(b_nc>=b_thresh);
% find offsets for bounding boxes
w_yoffSet = w_ypeak-size(white,1);
```

```
w_xoffSet = w_xpeak-size(white,2);
b_yoffSet = b_ypeak-size(black,1);
b_xoffSet = b_xpeak-size(black,2);
%display image and boxes
figure(4);
hold on;
imshow(bg);
for i = 1:size(w_xoffSet)
rectangle('Position', ...
   [w_xoffSet(i)+1, w_yoffSet(i)+1, size(white,2), size(white,1)], ...
   'EdgeColor','r');
rectangle('Position', ...
   [b_xoffSet(i)+1, b_yoffSet(i)+1, size(black,2), size(black,1)], ...
   'EdgeColor','b');
end
hold off

% Another Version of Part A

function tempMatching(I, GoBlackPiece, GoWhitePiece)

   % Convert board as well as pieces to grayscale

   GoBoard = I;
   GoBoardG = rgb2gray(I);
   GoBlackPieceG = rgb2gray(GoBlackPiece);
   GoWhitePieceG = rgb2gray(GoWhitePiece);

   % Make the correlation map

   resultBlack = normxcorr2(GoBlackPieceG, GoBoardG);
   maxResultBlack = max(resultBlack(:));
   resultWhite = normxcorr2(GoWhitePieceG, GoBoardG);
   maxResultWhite = max(resultWhite(:));

   % Threshold correlation map

   threshold = 0.6;
   resultBlackThresh = ((resultBlack > threshold) * maxResultBlack) .* resultBlack;
   resultWhiteThresh = ((resultWhite > threshold) * maxResultWhite) .* resultWhite;

   % Get the center pixel of each black and white piece

   regMaxBlack = imregionalmax(resultBlackThresh);
   regMaxWhite = imregionalmax(resultWhiteThresh);

   [rowBlack, colBlack] = size(regMaxBlack);
   [rowWhite, colWhite] = size(regMaxWhite);

   % Create a box around the center pixels of each black and white piece

   hold on;
   imshow(GoBoard);
```

```
   for i = 1:rowBlack
       for j = 1:colBlack
           if regMaxBlack(i, j) ~= 0
               xCoord = j;
               yCoord = i;
               xOffset = xCoord - size(GoBlackPieceG,2);
               yOffset = yCoord - size(GoBlackPieceG,1);
               rectangle('Position',[xOffset+1, yOffset+1, size(GoBlackPieceG,2),
size(GoBlackPieceG,1)],'EdgeColor','r');
           end
       end
   end
   for i = 1:rowWhite
       for j = 1:colWhite
           if regMaxWhite(i, j) ~= 0
               xCoord = j;
               yCoord = i;
               xOffset = xCoord - size(GoWhitePieceG,2);
               yOffset = yCoord - size(GoWhitePieceG,1);
               rectangle('Position',[xOffset+1, yOffset+1, size(GoWhitePieceG,2),
size(GoWhitePieceG,1)],'EdgeColor','b');
           end
       end
   end
   hold off

end
```

Part B - Finding Rock Paper Scissors

```
%% Project 3b
close all; clc; clear;

% Import & convert to B/W
group_rock_color = imread("hands_rock.jpg");
group_paper_color = imread("hands_paper.jpg");
group_scissors_color = imread("hands_scissors.jpg");

group_rock = rgb2gray(group_rock_color);
group_paper = rgb2gray(group_paper_color);
group_scissors = rgb2gray(group_scissors_color);

temp_rock = rgb2gray(imread("temp_rock.jpg"));
temp_paper = rgb2gray(imread("temp_paper.jpg"));
temp_scissors = rgb2gray(imread("temp_scissors.jpg"));

% Rock
figure()
imshow(group_rock_color)

[xcorners_r, ycorners_r] = find_template(group_rock, temp_rock, 0.6);
```

```
for i = 1:length(xcorners_r)
   rectangle('Position',[xcorners_r(i),ycorners_r(i), ...
       size(temp_rock,2),size(temp_rock,1)]);
end

% Paper
figure()
imshow(group_paper_color)

[xcorners_p, ycorners_p] = find_template(group_paper, temp_paper, 0.69);

for i = 1:length(xcorners_p)
   rectangle('Position',[xcorners_p(i),ycorners_p(i), ...
       size(temp_paper,2),size(temp_paper,1)]);
end

% Scissors
figure()
imshow(group_scissors_color)

[xcorners_s, ycorners_s] = find_template(group_scissors, temp_scissors, 0.6);

for i = 1:length(xcorners_s)
   rectangle('Position',[xcorners_s(i),ycorners_s(i), ...
       size(temp_scissors,2),size(temp_scissors,1)]);
end


%% Return arrays of x and y locations of upper-left corners of rectangles to be
drawn for each found item
function [xcorners, ycorners] = find_template(im, temp, threshold)
   % Create correlation map
   corr = normxcorr2(temp, im);

   % Threshold
   corr_tmask = corr > threshold;
   corr_thresh = corr.*corr_tmask;
   %imshow(corr_thresh)

   % Suppress non-maxima
   corr_maxima = imregionalmax(corr_thresh);
   %imshow(corr_maxima)

   [ypeaks, xpeaks] = find(corr_maxima==1);
   ycorners = ypeaks-size(temp,1);
   xcorners = xpeaks-size(temp,2);
end
```