KENNESAW STATE
UNIVERSITY

CS 7367
MACHINE VISION

SEMESTER PROJECT REPORT

CHECKMATE VISION

INSTRUCTOR

Dr. Mahmut Karakaya

Imad El Ddine Ghandour
Andrew Trilby-Bassett

1. ABSTRACT

The initial idea and purpose of this project was to exercise Machine Learning techniques in order to train a Neural Network to recognize, in real time, chess pieces on a physical board from a top view. Furthermore, we were planning to import this labeling data and convert it into a 2D array which would be readable by an AI; this would theoretically allow the AI to process the information and decide on the best next move. All in all, this package would allow a user to play chess with an AI on a real board in real time.

Unfortunately, a lot of complications arose. First, we had to construct our own labelled dataset to train out Multi-Object classifier; this dataset proved too strenuous on the computing hardware we had available. The pictures taken for the dataset were too high resolution, even when downsized the limitation of Graphics dedicated memory prevented us from having flexibility with training hyperparameters. For example, we could not increase the mini-batch size above a size of 2.

Going back to the drawing board it was decided that we would go with piece-by-piece individual classification. This meant that prior to classification, the board would have to be detected and segmented into 64 quadrants each to be classified by the classifier. Furthermore, this also meant that the dataset had to be significantly altered to serve the purposes of these change in piece classification modality. In order to make use of the already available dataset, a small XML parser was written to make use of the label data already available for every image in xml file format to go in and crop each image into 64 images each containing a quadrant from the initial image board. Work then went into organizing these images into folders each of which represented the classification of the respective images. A shorthand notation was used for each classification; this was done for simplicity. For example, for Black Knight an abbreviation was designated as BN and for White King WK.

In this form, the entire program was split into two parts. The first part used traditional computer vision techniques to identify the board and segment it out of the input image; techniques explained in section 3.2b of this report. Furthermore, that segmented board was then split up into 64 quadrants also using traditional computer vision techniques; this is also discussed in section 3.2b of the report. The second part involved a trained resnet101 CNN that classified each quadrant and identified what piece that quadrant did or did not have. It should be mentioned that doing it in this manner significantly slowed down the classification process wherein real-time classification is theoretically still possible but with constraints.

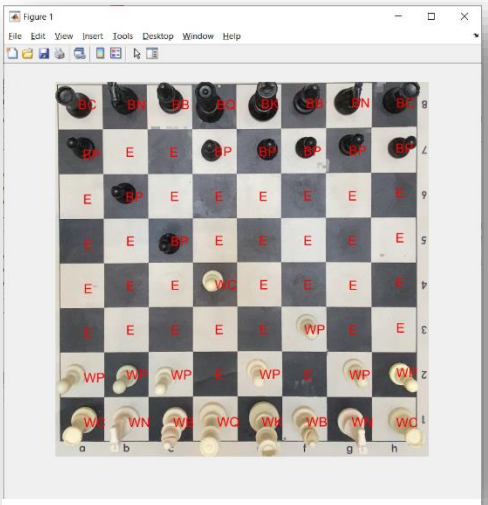


Figure 1: Chess-Board Classified in MATLAB screenshot

Results of the above utilized methods have been positive. We got an average classification accuracy of 98.7682%; an example classification cas be seen in Figure 1. It is

believed that the this loss in accuracy is due to two main factors. The first factor being the lower resolution of the video feed when compared to the training dataset; the pieces segemented from every frame of the video do not have as much detail as the ones provided in the training data-set, therefore, there are less idetifying key features that the pre-trained classifier can use to correctly classify the pieces. The second factor was board segmentation and border piece occlusion. Due to the fact that (mainly King and Queen) pieces are quite tall and get cut off from the top when in their initial positions due to board segementation, it gets quite difficult for the classifier to tell them apart. This can be seen clearly since when the pieces are moved from their initial positions, their classificaiton issue disappears.

2. DATA COLLECTION AND ORGANIZATION

As mentioned in the Abstract section, there were no readily available labeled dataset that matched the purposes of this project, therefore, a dataset had to be constructed and labeled from scratch. An entire day was spend gathering images for this dataset. A mount, purchased from amazon, was used to mount an iPhone 11 Pro Max in such a way that it pointed down at the chess board. Around 350 images were taken. In order to ensure randomization of the data a chess game was played, and a picture was taken for every round. Multiple games were played in different lighting conditions to ensure variation in the dataset.

Figure 2: LabelImg Canvas; Quadrants are surrounded by Bounding-Boxes

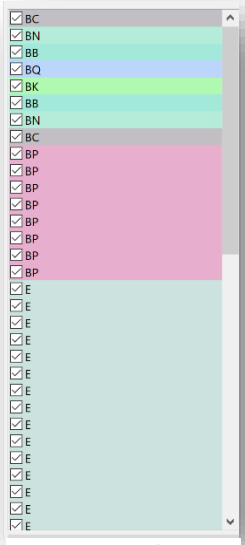


Figure 3: Bounding-Box Labels of the Canvas in Figure 2

Another entire day was taken to Label the images. Each image included 64 labels, one label per quadrant in the board as seen in Figures 2 and 3. The LabelImg program was used to perform this labeling and the output files were in xml format. A shorthand notation was used for this labeling; the notation is as follows:

- BB – Black Bishop
- BC – Black Castle
- BK – Black King
- BN – Black Knight
- BP – Black Pawn
- BQ – Black Queen
- E - Empty
- RQ – Replacement Queen (Upside-down Black or White Castle)
- WC – White Castle
- WB – White Bishop
- WK – White King
- WN – White Knight
- WP – White Pawn
- WQ – White Queen

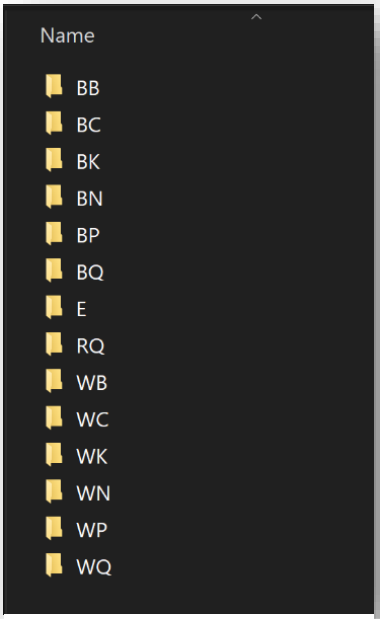


Figure 4: Folders containing the images of each chess piece

As mentioned in the Abstract, the team underwent a shift in methodology when encountering limitations and

Figure 5: The Folder Containing Black Bishop

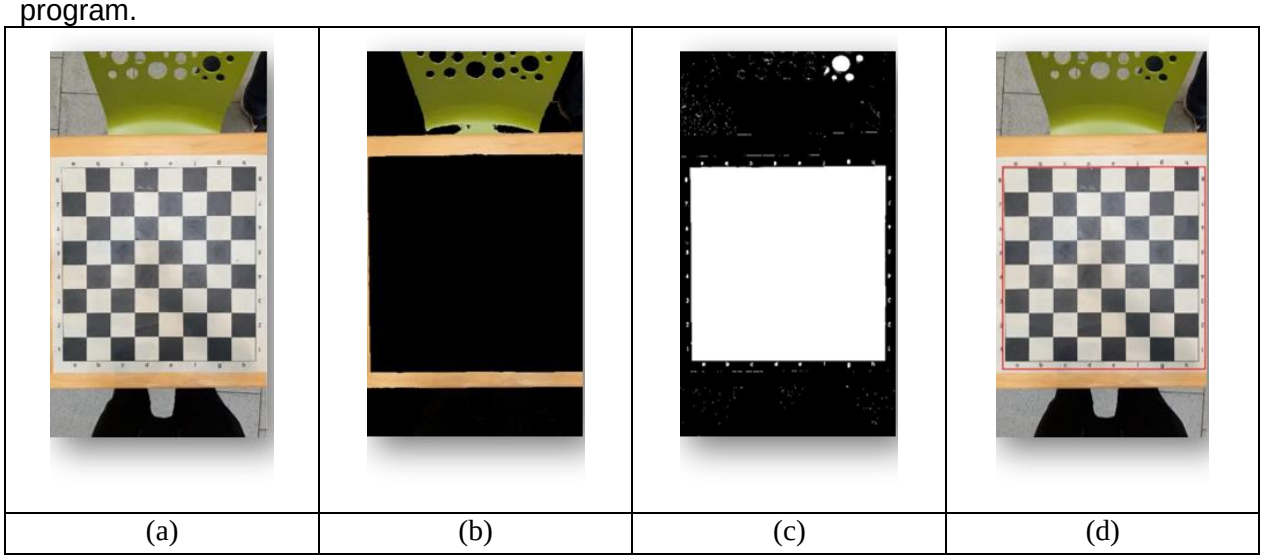
issues with computing resources, to that end the dataset had to be modified in order to serve the purposes of this project. A small python program was written in order to segment the image data into 64 cropped images each containing a chess board quadrant from the original image. These images were then organized into folders each labeled appropriately and each containing its respective piece as seen in Figures 4 and 5. It is important to mention that due to the fact that there were a lot of pieces that did not change any places on the baord from image to image this resulted in many duplciated images in the dataset. With that being said, it was decided that these images served no purpose in improving the training of the neural network and bloated the data set. Therefore, these duplicated images were removed in order to lighten the data set for training. A seperate testing set was constructed from video frames, this was in order to match real life real time scenrio, to test and validate the results of the training.

3. TEST RESULTS

3.1 Board Detection

3.1a – Whole Board Detection Results

For chess board detection, first the input image was converted to grayscale and a gaussian filter applied to blur the image. The blurred image was then thresholded using Otsu's method to separate the pixels in the image into foreground and background, where the foreground pixels are the region of interest (the chess board). The original image is converted to HSV, and K-means clustering is conducted on the saturation channel to segment the image into other regions of interest. After segmentation, the image was further blurred using median filters, then the edges were detected using Sobel filters. Structuring elements of lines at 45 and 90 degrees were made, then used to dilate the binary image to close lines on the board, when was then followed by the MATLAB `imfill` function to close holes. Erosion was then used with a diamond structure to remove unnecessary features. The result of this blurring and morphological functions is then used to remove unnecessary details from the segmented images. Sobel edge detection is applied once again, this time just on the segmented image, where the `regionprops` function is used to get the bounding boxes of the segmented image. From these bounding boxes, the largest box is the chessboard itself, when is then cropped and the cropped image as well as the coordinates of the cropped box in the original image are returned to the main program.



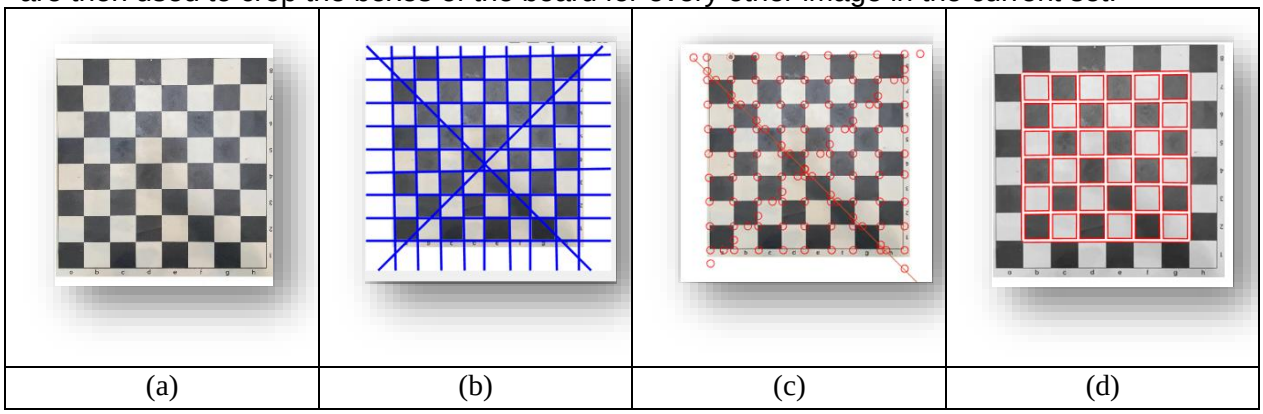
(a)	(b)	(c)	(d)

Figure 6: (a) Input image. (b) Image after applying gaussian blurring, Otsu thresholding, and K-means clustering. (c) Sobel edges on segmented image. (d) Bounding box of chessboard.

3.1b – Board Square Detection Results

After cropping the chessboard, the cropped board is then fed into the cropImages.m function. Here, the cropped board is first blurred using gaussian filters, and edges are once again detected using Sobel edge detection. After dilating the detected lines, Hough transformation is applied, followed by `houghpeaks` and `houghlines` to get the top 20 candidates for lines in the image. After, the InterX MATLAB code is used to find the intersection between the Hough lines. Then, the Omnidirectional Camera Calibration Toolbox is used for the function `chessboardsFromCorners`, which uses the intersection points to detect and get the coordinates of each box on the chessboard. As pieces on the outer boxes of the chessboard tend to hang over the edge, the bounding box is only drawn on the inner 6x6 square on the 8x8 chessboard. The outer edge is then expanded to manually to allow the crops to expand slightly outside the

box. The coordinates of these boxes on the board are then returned to the main program, which are then used to crop the boxes of the board for every other image in the current set.



(a) (b) (c) (d)

Figure 7: (a) Cropped image received from Figure 6. (b) Hough lines detected on image. (c) Intersections of Hough lines. (d) Bounding box of inner 6x6 chess boxes.

3.1c – Board Detection Discussion

The biggest challenge with board detection, which led to the two-tier segmentation, was the detection of too many or too few lines on the chessboard when using Hough lines. Without first isolating the chessboard from the background, the detected lines would often be drawn on the background and cause a chessboard detection failure. Limiting the image to just the board itself greatly cut down on this problem. The other major problem that caused issues with Hough line detection occurred when pieces on the board overlapped squares, especially on corners of squares. In such cases, the overlap could cause the Hough line to end early, or even split into two separate Hough lines, which caused the detected chessboard to be unpredictably missing rows or columns. The team attempts to solve this issue first by more morphological functions, then by using the coordinates of the entire board to create an estimated crop as the board should in theory be 8x8 even sized squares. However, the camera lens resulted in a warped image where the size of each box varied slightly and applying warps still did not solve the overlap problem. Ultimately, it was decided that a blank board should always be used for detection first while another solution could be worked on.

3.2 Neural Network

3.2a – Neural Network Results

The results of this resnet101 were very good, more details about why this network was chosen in section 3.2b. The validation accuracy of this network was about 98%, the testing accuracy was in line with the validation results as well. To achieve these results a variable learning rate was used that started out at alpha = 0.0003 and then decreased at a rate of 0.2 every few iterations of training; this was done to ensure that higher end features were learnt in the first few iterations and lower-end features were learnt at later iterations building off the higher end features. Furthermore, to prevent variation in the results, the data was regularized using L2Regularization of 0.005. The training was run with the original image set split up into two groups; 70% of the images were used for training and 30% were used for validation. Another completely different set was used for testing, more on this set testing will be discussed in section 3.2b. The training was done on a single GPU to ensure high training speed and ability to adjust parameters and retrain in a short period of time. Accuracy results of the Demo are in Figure 8, the resnet101 training graph results are in Figure 9, and lastly the training parameters are in Figure 10.

	Contains	Correct	Misplaced	Accuracy
BB	73	73	0	100%
BC	76	76	0	100%
BK	64	64	1	98%
BN	75	75	0	100%
BP	79	79	4	95%
BQ	39	36	0	100%
E	83	83	0	100%
RQ	0	0	0	100%
WB	63	62		100%
WC	75	73	2	97%
WK	63	63	0	100%
WN	74	69	0	100%
WP	80	80	4	95%
WQ	49	49	0	100%
Total	893		98.7682%	

Figure 8: Testing Results

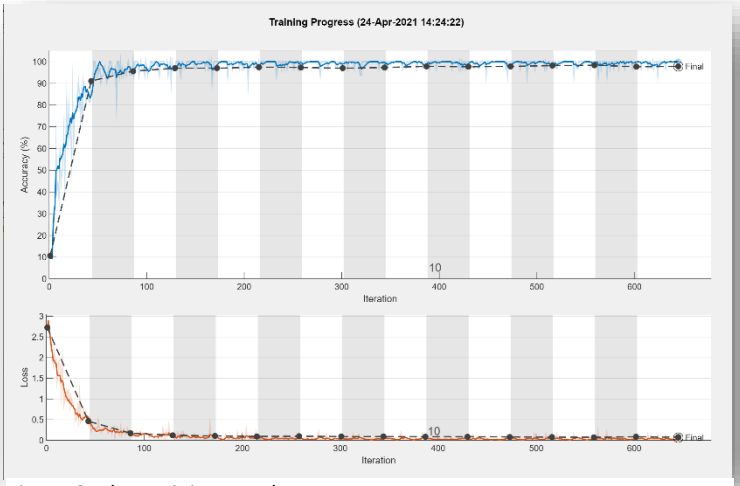


Figure 9: The Training Results

```
trainOpt = trainingOptions('sgdm', ...
    'Momentum', 0.9, ...
    'MaxEpochs', 15, ...
    'InitialLearnRate',3e-4, ...
    'Shuffle','every-epoch', ...
    'ValidationData', augimgsValidation, ...
    'ValidationFrequency', valFrequency, ...
    'LearnRateSchedule','piecewise','LearnRateDropFactor',0.2 ,'LearnRateDropPeriod',5,...
    'L2Regularization',0.0005, ...
    'MiniBatchSize', miniBatchSize, ...
    'Verbose', false, ...
    'Plots', 'training-progress', ...
    'ExecutionEnvironment', 'multi');
```

Figure 10: Training Hyper Parameters

3.2b – Neural Network Discussion

The team went through multiple variations of the neural network before settling on resnet101 network. As can be seen in Figure 11, AlexNet, VGG19, and Googlenet were all different contenders as classifier of chess pieces. To minimize bias and ensure consistency and fairness of testing, each of these neural networks were tested with the exact same testing dataset. Furthermore, we also changed training options to see whether that affected the results of these different networks. While the validation results of all these networks were high ranging from 96%-98% (which we believe is due to the relatively large dataset and use of data augmentation), the test accuracy results were not that high for all of them. The team saw a pattern wherein with the increase in layers meant an increase in testing accuracy. While, by convention, an increase in layers does not directly positively correlate with an increase in testing accuracy, it is believed that the depth of the neural network was directly correlated to the amount of features the network was able to pick up from each image it processed and classified. This is especially true and seen when down sampling the test images to various lower resolutions before resizing them in order to speed up classification. What the team observed was a consistent degradation in classification accuracy across the board, which makes sense all things considered. However, the network that saw the least degradation and held up the best across the board was the resnet101 network.

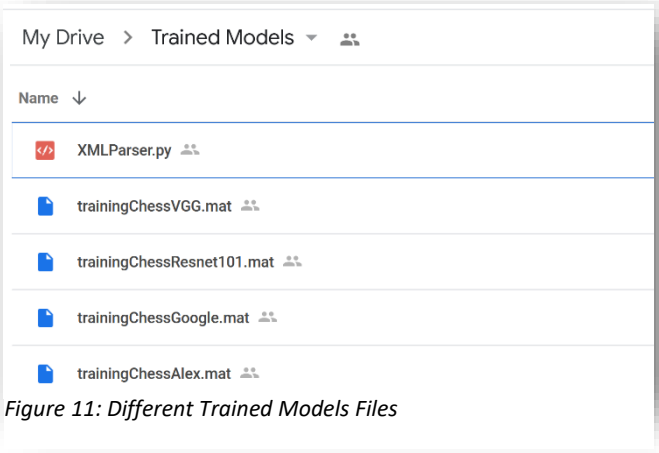


Figure 11: Different Trained Models Files

The purposes of this manner of testing were to ensure that if a lower resolution image of the chess board was fed to the neural network, it will be able to still do its intended job accurately. While our initial purpose for this type of testing was for speed, this did prove useful later when the team was hit with the limitation of video resolution that would be eventually fed into the algorithm for live demo purposes. The classifier was trained with images that had dimensions of 3024x4032 pixels (12MP) which were then subdivided in 64 lower resolution images. However, the video feed that was used for

testing had a 4K video resolution which means that each frame had a resolution of 3840x2160 pixels (8.3MP) at 30 frames per second; this is around two-thirds of the original training resolution of the classifier when subdivided into 64 quadrants. Furthermore, the team had to use a different wider-angle lens to capture the testing footage, this is since the iPhone crops in the field of view when going into video mode. This lens had a smaller *f2.4* aperture (versus *f1.8* of the original lens) which meant that the frames of this video were not just of lower resolution but contained more noise and artifacting due to lower light capturing ability per frame, and distortion due to it being an ultra-wide 120-degree lens. Albite all of this, the resnet101 network was able to deal with this quite well maintaining accurate results. It is important to note that this does not imply that the resnet101 trained network is not impervious; after a certain threshold of resolution loss, the classifier does indeed start falling apart and losing significant accuracy. In fact, we as humans, started having a hard time distinguishing pieces when exposed to this lower resolution as well.

4. CONCLUSION AND FUTHER REMARKS

All in all, we believe that what we ended up with showed promising results. As hinted in the Abstract, we would have preferred to use Multi-Object Detection instead of board segmentation and image classification to achieve chess recognition and labeling in an image. Alas, however, due to time constraints and computing hardware resource limitation we were not able to go with this route. We believe taking the initial route that we planned to go with would have netted a much faster and more accurate network and algorithm. Furthermore, we would have not had to rely on board detection and segmentation; this means that the resulting algorithm would have been not only faster but much more dynamic. As it stands now, the resulting algorithm can only work when calibrated with an empty board. Furthermore, this algorithm requires that the board and camera rest in a static position throughout the duration of classification; any of these two factors can compromise board segmentation and, hence, piece classification.

One of aspects that we wanted to also implement was the AI to play the chess match live with an actual person as an opponent. Theoretically, the data from the classification would have been pulled in, converted into an array representation of the actual chess board and then fed into an AI which would decide its next move. Due to time constraints from switching gears to a different piece classification method, we lost time and were not able to implement this bit of the project. Furthermore, we think that this would have been slightly more difficult to implements in any case with the current classification implementation as it is significantly slower and less dynamic as mentioned above than its equal multi-object detection counterpart.

5. SOURCE CODE

5.1.a Whole Board Detection

```
function croppedBoard = cropChessBoard(image)
% Read in image
I = image;

% Convert to grayscale
Ig = rgb2gray(I);

% Get a gaussian kernel for blurring
K = fspecial('gaussian',[10 10],3);

% Blur the image
Igf = imfilter(Ig, K);
Igf = imfilter(Igf, K);
Igf = imfilter(Igf, K);

[counts,x] = imhist(Igf,10);
T = otsuthresh(counts);
BW = imbinarize(Igf,T);

HSV = rgb2hsv(I);
h = HSV(:,:,1);
s = HSV(:,:,2);
v = HSV(:,:,3);

cform = makecform('srgb2lab');
lab_he = applycform(I,cform);

l = lab_he(:,:,1);
a = lab_he(:,:,2);
b = lab_he(:,:,3);

ab = double(s);
nrows = size(ab,1);
ncols = size(ab,2);
ab = reshape(ab,nrows*ncols,1);

nColors = 2;
% repeat the clustering 3 times to avoid local minima
[cluster_idx, cluster_center] =
kmeans(ab,nColors,'distance','sqeuclidean', ...
                                        'Replicates',10);

pixel_labels = reshape(cluster_idx,nrows,ncols);
segmented_images = cell(1,2);
rgb_label = repmat(pixel_labels,[1 1 3]);

for k = 1:nColors
```

```
    color = I;
    color(rgb_label ~= k) = 0;
    segmented_images{k} = color;
end

% Blur the image
Igf = medfilt2(Igf);
% figure, imshow(I), title('original image');

BWs = Igf;
[~, threshold] = edge(Igf, 'sobel');
fudgeFactor = .6;
BWs = edge(Igf,'sobel', threshold * fudgeFactor);

se90 = strel('line', 5, 45);
se0 = strel('line', 5, 90);

BWsdil = imdilate(BWs, [se90 se0]);
BWdfill = imfill(BWsdil, 'holes');
BWnobord = imclearborder(BWdfill, 4);

seD = strel('diamond',1);
BWfinal = imerode(BWnobord,seD);
BWfinal = imerode(BWfinal,seD);
BWfinal = BWfinal*255;
if(cluster_center(2,1)<0.1)
    sg_img = segmented_images{2};
else
    sg_img = segmented_images{1};
end
for i=1:size(Igf,1)
    for j=1:size(Igf,2)
        if BWfinal(i,j)==0
            sg_img(i,j,1)=0;
            sg_img(i,j,2)=0;
            sg_img(i,j,3)=0;
        end
    end
end

sg_img_g = rgb2gray(sg_img);
BW =
imbinarize(sg_img_g,'adaptive','ForegroundPolarity','dark','Sensitivity
',0.4);

se = strel('disk',4);
afterOpening = imopen(BW,se);
afterOpening = imerode(afterOpening,se);

sg_img_g = rgb2gray(sg_img);
```

```
BW =
imbinarize(sg_img_g,'adaptive','ForegroundPolarity','dark','Sensitivity
',0.4);

se = strel('disk',7);
se1 = strel('line',6,60);
afterOpening = imopen(BW,se);
afterOpening = imdilate(afterOpening,se);

BW = edge(Ig,'sobel', 0.4);

stats = regionprops('table',afterOpening,'BoundingBox');
bb = stats.BoundingBox;

largestBoxWidth = 0;
largestBoxHeight = 0;
largestBox=[0 0 0 0];
for i=1:size(bb,1)
    if(bb(i,3)>largestBoxWidth && bb(i,4) > largestBoxHeight)
        largestBoxWidth=bb(i,3);
        largestBoxHeight=bb(i,4);
        largestBox=bb(i,:);
    end
end

largestBox(1) = largestBox(1)-(largestBox(1)*0.01);
largestBox(2) = largestBox(2)+(largestBox(2)*0.01);
largestBox(3) = largestBox(3)*1.03;
largestBox(4) = largestBox(4)*1.03;
%rectangle('Position', largestBox,'EdgeColor','r', 'LineWidth', 1);
%title('Detected Bounding Box');
%croppedBoard = imcrop(I,largestBox);
croppedBoard = largestBox;
```

5.1.b Board Square Detection

```
function croppedBoxes = cropImages(image)
addpath('matching');
I = image;

%I = imresize(I, 0.5);
Ig = rgb2gray(I);

% Get a gaussian kernel for blurring
K = fspecial('gaussian');

% Blur the image
Igf = imfilter(Ig, K);
Igf = imfilter(Igf, K);
Igf = imfilter(Igf, K);

% Detect edges
E = edge(Ig, 'sobel');
```

```
se = strel('line',6,90);
se1 = strel('line',6,0);
afterOpening = imdilate(E,se);
afterOpening = imdilate(afterOpening,se1);

% Perform Hough Line transform
[H, T, R] = hough(afterOpening);

% Get top N line candidates from hough accumulator
N = 20;
P = houghpeaks(H, N);

% Get hough line parameters
lines = houghlines(Igf, T, R, P);

coeff =[];

pointInt = [];
% uses InterX:
% https://www.mathworks.com/matlabcentral/fileexchange/22441-curve-
intersections,
% to find the intersection between the lines
for k = 1:length(lines)
     xy = [lines(k).point1(1) lines(k).point2(1);lines(k).point1(2)
lines(k).point2(2)];
    for l = 1:length(lines)
        if(k~=l)
            xy1 = [lines(l).point1(1)
lines(l).point2(1);lines(l).point1(2) lines(l).point2(2)];
            inter = InterX(xy,xy1);
            x = inter(1,:);
            y = inter(2,:);
            if (isempty(x) && isempty(y))
            else
                pointInt = [pointInt; [x y]];
            end

        end
    end
end

pointInt = unique(pointInt, 'rows');
k = boundary(pointInt(:,1),pointInt(:,2),1);

corners = findCorners(Ig,0.05,2);
chessboards = chessboardsFromCorners(corners);

board = chessboards{1,1};
k=1;
try
    bboxfinal = [];
    for i=1:size(board,1)-1
```

```
        for j=1:size(board,2)-1
            points = [];
            a = board(i,j);
            b = board(i,j+1);
            c = board(i+1,j);
            d = board(i+1,j+1);

            points = [points; corners.p(a,:)];
            points = [points; corners.p(b,:)];
            points = [points; corners.p(c,:)];
            points = [points; corners.p(d,:)];

            bb = boundingBox(points);
            xmin = bb(1,1);
            ymin = bb(1,3);
            width = bb(1,2)-bb(1,1);
            height = bb(1,4)-bb(1,3);
            bboxfinal = [bboxfinal;[xmin, ymin, width, height]];
            k=k+1;
        end
    end

    %extend bounding boxes
    centerPoint = board(4,4);
    centerPoint = corners.p(centerPoint,:);
    cornerPoints = [1 1; 1 7; 7 1; 7 7];
    for j=1:4
        firstEdge = board(cornerPoints(j,1),cornerPoints(j,2));
        firstEdge = corners.p(firstEdge,:);
        orientationCase = 1;
        if(firstEdge(1) < centerPoint(1) && firstEdge(2) <
centerPoint(2))
            orientationCase = 3;
            bb2 = [firstEdge(1,1)-(width*1.1),firstEdge(1,2)-
(height*1.1),width*1.1,height*1.1];
            bboxfinal = [bboxfinal;bb2];
            k=k+1;
        elseif(firstEdge(1) > centerPoint(1) && firstEdge(2) <
centerPoint(2))
            orientationCase = 1;
            bb2 = [firstEdge(1,1),firstEdge(1,2)-
(height*1.1),width*1.1,height*1.1];
            bboxfinal = [bboxfinal;bb2];
            k=k+1;
        elseif(firstEdge(1) < centerPoint(1) && firstEdge(2) >
centerPoint(2))
            orientationCase = 4;
            bb2 = [firstEdge(1,1)-
width,firstEdge(1,2),width*1.1,height*1.1];
            bboxfinal = [bboxfinal;bb2];
            k=k+1;
```

```
        elseif(firstEdge(1) > centerPoint(1) && firstEdge(2) >
centerPoint(2))
            orientationCase = 2;
            bb2 = [firstEdge(1,1),firstEdge(1,2),width*1.1,height*1.1];
            bboxfinal = [bboxfinal;bb2];
            k=k+1;
        end

        switch orientationCase
            case 1
                for i=1:6
                    edgePoint = board(i,1);
                    edgePoint = corners.p(edgePoint,:);
                    bb2 = [edgePoint(1,1)-width,edgePoint(1,2)-
(height*1.1),width*1.1,height*1.1];
                    bboxfinal = [bboxfinal;bb2];
                    k=k+1;
                end
            case 2
                for i=7:-1:2
                    edgePoint = board(1,i);
                    edgePoint = corners.p(edgePoint,:);
                    bb2 = [edgePoint(1,1),edgePoint(1,2)-
(height*1.1),width*1.1,height*1.1];
                    bboxfinal = [bboxfinal;bb2];
                    k=k+1;
                end

            case 3
                for i=1:6
                    edgePoint = board(7,i);
                    edgePoint = corners.p(edgePoint,:);
                    bb2 = [edgePoint(1,1)-
width,edgePoint(1,2),width*1.1,height*1.1];
                    bboxfinal = [bboxfinal;bb2];
                    k=k+1;
                end

            case 4
                for i=7:-1:2
                    edgePoint = board(i,7);
                    edgePoint = corners.p(edgePoint,:);
                    bb2 =
[edgePoint(1,1),edgePoint(1,2),width*1.1,height*1.1];
                    bboxfinal = [bboxfinal;bb2];
                    k=k+1;
                end
        end
    end
    croppedBoxes = bboxfinal;
catch
    display("Could not identify full board,Please Realign");
```

```
    croppedBoxes = [];
end
```

5.2 Neural Network

Please note that there are supporting files that help the main driver strip and redefine the pretrained neural network's classification layer and input layers. These source code files are included in the source code zip accompanying this report. Only the main driver code is shown below:

```
close all;
clear;
clc;

%Import the Datasets
originalTrainings = imageDatastore('Chess\Train\', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
originalTest = imageDatastore('ChessDUP\Train\', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');

% To resize the Input
inputSize = [227 227];
originalTrainings.ReadFcn = @(loc)imresize(imread(loc),inputSize);
originalTest.ReadFcn = @(loc)imresize(imread(loc),inputSize);

%Split Dataset into Training and Validation Sets
[imgsTrain,imgsValidation] =
splitEachLabel(originalTrainings,0.7,'randomized');

%Improting Pre-Trained Neural Network
net = resnet101;

%Stripping and Redefined Neural Network's Classification and Input
Layers
%%%%%%%%%%%for DAG NN%%%%%%%%%%%%%%%%%%%%
if isa(net,'SeriesNetwork')
  lgraph = layerGraph(net.Layers);
else
  lgraph = layerGraph(net);
end

[learnableLayer,classLayer] = findLayersToReplace(lgraph);
[learnableLayer,classLayer]

numClasses = numel(categories(imgsTrain.Labels));

if isa(learnableLayer,'nnet.cnn.layer.FullyConnectedLayer')
    newLearnableLayer = fullyConnectedLayer(numClasses, ...
        'Name','new_fc', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);
```

```
elseif isa(learnableLayer,'nnet.cnn.layer.Convolution2DLayer')
    newLearnableLayer = convolution2dLayer(1,numClasses, ...
        'Name','new_conv', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);
end

lgraph = replaceLayer(lgraph,learnableLayer.Name,newLearnableLayer);

newClassLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,classLayer.Name,newClassLayer);

layer = imageInputLayer([227 227 3],'Name','input');

lgraph = replaceLayer(lgraph,'data', layer); %'input_1' for resnet50;
'data' for googlenet and resnet101

%Freeze Layers
layers = lgraph.Layers;
connections = lgraph.Connections;

layers(1:10) = freezeWeights(layers(1:10));
lgraph = createLgraphUsingConnections(layers,connections);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Augmentation of Validation and Training
pixelRange = [-30 30];
imagesize = [227 227 3];

imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
augimgsTrain = augmentedImageDatastore(imagesize,imgsTrain, ...
    'DataAugmentation',imageAugmenter);

augimgsValidation = augmentedImageDatastore(imagesize,imgsValidation);

%Training, Validation and Results
miniBatchSize = 32;
valFrequency = floor(numel(augimgsTrain.Files)/miniBatchSize);
trainOpt = trainingOptions('sgdm', ...
    'Momentum', 0.9, ...
    'MaxEpochs', 15, ...
    'InitialLearnRate',3e-4, ...
    'Shuffle','every-epoch', ...
    'ValidationData', augimgsValidation, ...
    'ValidationFrequency', valFrequency, ...
    'LearnRateSchedule','piecewise','LearnRateDropFactor',0.2
,'LearnRateDropPeriod',5,...
```

```
    'L2Regularization',0.0005, ...
    'MiniBatchSize', miniBatchSize, ...
    'Verbose', false, ...
    'Plots', 'training-progress', ...
    'ExecutionEnvironment', 'multi');

trainingChess = trainNetwork(augimgsTrain, lgraph, trainOpt);
%To Save Chess Training State for Use Later in Classificaiton
save trainingChessResnet101
%To Test Trained Netowrk with Testing Images
[YPred, scores] = classify(trainingChess,originalTest);
YTest = originalTest.Labels;
%Outputs Testing Accuracy
accuracy = mean(YPred == YTest)
```