Reliable Go Game Images Recognition under Strong Light Attack

Yiyao Zhuo¹, Han Fang¹, Kai Zhu²³, Hongchen Zhan², Jie Yuan¹²
¹ Electrical Engineering Department, Nanjing University, Nanjing, Jiangsu, 210046, China
² Electrical Engineering Department, Jinling College of Nanjing University, Nanjing, Jiangsu, 210089, China
*Corresponding author: Kai Zhu (e-mail: zhukai@jlty.nju.edu.cn).

ABSTRACT Go is a popular global game whose win or loss is only determined by the number of intersection points surrounded by black or white pieces. Among all the counting methods, the traditional manual counting method is time-consuming. Additionally, the current Go game images recognition technology cannot endure light reflection attacks or extreme image capture angles effectively. In this paper, a reliable Go game images recognition method is proposed which not only can resist light reflection attacks but also can endure various image capture angles. To obtain this goal, we propose a detection method based on the optimized CNNs (Convolutional Neural Network) framework. Experiments on recognizing 3220 images show that the average accuracy with our proposed method is over 99.99%, which is 22 times better than the accuracy of the state-of-the-art approach on Go game images recognition. Besides, our study provides potential references for the recognition of interfered small objects in groups that have few features. It provides a reference in similar application scenarios such as the detection of animal crowds, industrial parts, physiological tissues, and micro-particles.

INDEX TERMS Go board detection, Go pieces recognition, CNN, model ensemble

I. INTRODUCTION

Machine-assisted methods have been applied in many competitive games [1-3]. Compared with manual methods, they can record the real-time play and judge the final result with faster speed and higher accuracy. As one kind of competitive sport, the Go game also needs machine assistance. Go game originated in China and soon spread all over the world. It uses a square chessboard and black-and-white circular chess pieces where 19 vertical and horizontal lines divide the chessboard into 361 intersection points. When playing, two contestants put pieces alternately until the end and confirm the final result by counting methods (the contestant who has more territory including pieces and the intersection points surrounded by them is the winner) using manual or machine-assisted methods. However, the manual method is inefficient and the current machine-assisted technology cannot endure some special circumstances. However, the problem of reliably recognizing Go game images is still unsolved [4].

This problem can be split into two stages: The first stage is to detect the chessboard and then recognize chess pieces and their location. When detecting the chessboard, most studies simply combined common transformations. For example, Dela used the corner detection method proposed by Harris and make Hough Transform to enforce linearity constraints and abandon substandard chessboard lines [5]. However, their methods made extra simplifications and assumptions and required too many manual operations. They were sensitive to image capture angles where the shooting angles needed to be fixed directly above the chessboard. When detecting crosslines and intersections in chessboards, related technologies were applied in the field of camera calibration and three-dimensional surface reconstruction [6]. Canny tried to catch the desirable properties by making a set of edge detection criteria. His system cannot work well in conditions with unsatisfactory lighting and shooting angles [7]. By utilizing intensity features, the ChESS (Chessboard Extraction by Subtraction and Summation) detector spent 5.82 ms processing a single image without prior assumptions. A drawback of this approach is that this algorithm can only handle simple situations which had difficulty identifying grid points around obstacles [8]. Afterward, the author continued to present the LAPS (Lattice Points Search) detector which was directly inherited from the ChESS detector acquiring an accuracy rate of 99.5% in grid points detection. To handle the problem of detecting images with partial occlusion and corner-missing, Chen proposed an intersection point detection framework based on the CCDN (Checkerboard Corner Detection Network) [9-10] model. During recognitions on chess pieces, most systems combined computer vision with
robotic development for application in a real usage scenario [11-14]. They can trace each move in a chess game with a real-time speed by finding differences between two neighboring movements [15]. However, most tracing systems must run under the premise that the initial layout of the chess map is known, which added redundant manual interventions. Illeperuma got rid of this premise using basic algorithms such as the color histogram and edge detection and finally got a 95% accuracy [16].

The recent breakthroughs in DNN (Deep Neural Networks) [17] had provided an astonishing advance in the application of image classification algorithms. Many researchers had tried to apply DNN-based methods or compare the performance of their methods with DNN. Xie introduced an oriented chamfer matching method whose performance was comparable with CNNs [18]. Delgado synthesized virtual 3D chess images on Blender and improved the VGG16 convolutional network using Python API [19], which achieved a 97% accuracy for chess pieces classification. In 2020, Quintana implemented a functional framework called LiveChess2FEN which was able to digitize a chess game image in real-time [20]. This system employed CNNs to classify all individual divided squares after locating the board. When classifying, they tested six kinds of deep learning models to select one with a reasonable speed-performance ratio. As a result, LiveChess2FEN reached a speed of 1 fps and an accuracy of 92%.

When it comes to the digitization of the Go game images, relevant researches are sparse and most of them mainly utilize traditional approaches. As early as 1997, Huang adopted the chain coding theory to recognize chess records on paper [21]. In 2016, Liang used geometric transformation, histogram threshold, and image projection to locate Go pieces [22]. In 2020, Gui applied traditional transformations such as binarization, color space transformation, threshold segmentation, high-pass filtering, Huff transform, etc. to detect, locate, and segment chess pieces [23], achieving an accuracy rate of 93.3%. In addition to taking photos, some studies also tried to apply video content for Go game images recognition. For example, Zhang used the planar measurement technology for pieces locating, which got a 10fps speed and a 98% accuracy. However, this system had a limitation that the shooting angle must be fixed directly above the chessboard [24]. Unlike Zhang, a team called Opensoft from South Korea solved this problem. Their system can withstand a low capture angle leading to a 99.99% accuracy. But it acted poorly on chessboards under light reflection attacks [25]. Besides the above scientific researches, commercial Go software with Go game images recognition functions also appeared based on these technologies. Among all software, Go Sweep, Go Camera, Go Eye, and Tencent Go were most commonly downloaded. Instead of recognizing Go games in a real scenario, Go Sweep can only identify Go game images on paper with a limitation that the chessboard should keep in the recognition frame with a frontal angle. The second software, Go Camera, can convert both photos and videos into electronic records with a real-time speed. These records were saved as an SGF format file which can be added with remarks information such as player names and Go ratings and then copied to other platforms. However, the recognition accuracy of Go Camera needed to be improved since sometimes users had to manually adjust the sensitivity of black-and-white color and fine-tune the final position of the chess pieces. The third software is Go Eye that worked on IOS (Internetworking Operating System-Cisco) devices. It can recognize both Go games in a real scenario and computer-synthesized images with an accuracy of 90%. Before identifying pieces, users must manually locate the four corners of the chessboard. The last introduced software is Tencent Go. It employed CNN-based image object detection models with an accuracy of 99%. Its accuracy will slightly lift since more images would be automatically collected for further training of the deep learning model with increasing players applying this function. When recognizing Go games in a real scenario, the chessboard must be kept in the recognition frame with a frontal angle. Finally, it should be noted that this function was remotely operated online through the server, leading to the limitation that an account can only use it 20 times a day to relieve the computational pressure.

In this paper, we propose a Go game images recognition method. Firstly, we collected 3,220 images and segmented them into three different types of data sets. Then we trained three optimized detection networks based on the CNN framework. Next, Go board images were detected by these three networks. At last, a model ensemble was employed to find an optimal result. Experiments on recognizing 3,220 images show that the average accuracy with our proposed method is over 99.99%.

II. MATERIALS AND METHODS

A. IMAGE ACQUISITION AND PRE-PROCESSING

The quality of data sets affected the performance of deep learning heavily. To confirm the quality of our datasets, we collected 3,220 Go game images in chess clubs using mobile phones, tablets PC, and digital cameras whose resolutions ranged from 960×720 to 4,208×2,368 instead of generating synthetic images. Meanwhile, various conditions were considered such as light-reflective areas and locations, capture angles, chessboard types, and background colors, which enriched the diversity of the data sets. Fig.1 shows some examples in the image dataset. Images in Fig.1(a)(b) are captured under light attacks with different positions, sizes, and intensities. There are different levels of brightness in Fig.1(a) resulting in various sensitivity of color recognition between black pieces, white pieces, and background. Fig.1(b) shows special chessboard types and background colors. In Fig.1(c), chessboards tilt in different directions under low capture angles.
As illustrated in Fig. 2, our Go game images recognition system contained four sub-stages using three deep learning networks. In the first sub-stage, we pre-processed all images utilizing graying, cropping, non-linear transformation, and histogram equalization and then split them into train set, validation set, and test set. After successfully locating all Go boards by the trained network GO_CORNER, we employed perspective transformation to straighten all tilted boards captured with various angles and cut out the board area. Subsequently, two networks named GO_PIECEX1 and GO_PIECEX2 recognized all pieces simultaneously and then generated two kinds of predictions. In the third sub-stage, we sorted all two predictions by positional information and mapped each Go game into digital records Go_Board1 and Go_Board2. At last, a model ensemble was applied to get an optimal result.

1) BOARD DETECTION
In the stage of board detection, we chose YOLO (You Only Look Once) framework which is a one-stage target detection algorithm.

As shown in Fig. 3, the detailed Go game images detection processes are described as follows:

---

**FIGURE 1.** The original Go board images.

We pre-processed images utilizing graying, cropping, non-linear transformation, and histogram equalization. The original images were firstly converted into gray-colored since useful features depended on gray value and texture when recognizing Go game images. This pre-processing helped to reduce the calculation and storage space. Subsequently, we performed non-linear transformation and histogram equalization processing to adjust the image brightness in a reasonable range. After that, these images were split as follows: 80% (2,576) of the images were chosen as the train set, 10% (322) the validation set, and the remaining 10% (322) the test set.

As mentioned above, our method needed to detect the chessboard and the Go pieces one after the other. Therefore, we needed to label the chessboard and each intersection point (for placing chess pieces) separately. The ground truth of four vertices in each chessboard was annotated with LabelImg software [27]. When labeling intersection points, we abandoned the method of manual annotation and adopted the perspective algorithm to calculate every location automatically. It was worth noting that we labeled the intersection points in two ways. That is to say, the ground truth encircled two sizes of target detection areas which can learn different reflective features. There were $19 \times 19 = 361$ rectangular boxes in an image and each box included one point. An annotated image had $18 \times 19 = 342$ boxes and each of them contained two neighboring points.

**B. BOARD DETECTION AND PIECES RECOGNITION**

We need to locate the chessboard first before recognizing Go pieces since the chess pieces that are scattered outside the chessboard would interfere with the detection results and increase the detection time. Besides, a single Go piece occupied a little space in the whole image, leading to a low recall rate of deep learning networks [28-30].

---

**FIGURE 2.** An overview of our proposed Go game image recognition.

**FIGURE 3.** The framework of the Go game images detection.
Feature extraction: After being resized to $640 \times 640$, Go map images were successively received by a Focus module and four BottleNeckCSP (BottleNeck Cross Stage Partial) modules, generating four feature layers with different sizes. These layers passed basic operations such as upsampling, convolution, and channel fusion to get three feature maps which constituted a feature pyramid. In a feature pyramid, the largest map was responsible for identifying small target objects and vice versa.

Feature map division: Our model divided the three feature maps into $S \times S$ equal-sized grids ($S$ was chosen as 20, 40, and 80 separately in this study). Each grid had $B$ kinds of prior bounding boxes and each bounding box produced a coordinate vector $(x_i, y_i, w_i, h_i)$, a confidence score and $C$ class probabilities, where $(x_i, y_i)$ was coordinate of an upper-left point and $(w_i, h_i)$ was the width-height of a box. In total, the number of outputs from each feature map was $S \times S \times B \times (5 + C)$.

Bounding box prediction: During the surmising period, predictions with the largest IOU (Intersection Over Union) were considered as a positive example. At the same time, detections would be ignored once the confidence score was under 0.6. Finally, we employed NMS (Non-Maximum value Suppression) to find the best match from overlapping redundant boxes.

To measure the performance of the model being trained, we adopted the following loss function which mainly included box loss, object loss, and classification loss (see Equation 1-4).

\[
\text{Loss} = L_{\text{box}} + L_{\text{obj}} + L_{\text{cls}}
\]

\[
L_{\text{box}} = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \sum_{l=0}^{B} q_{ij}^{ob} (x_i - \tilde{x}_i)^2 + (y_i - \tilde{y}_i)^2 + (\sqrt{w_i} - \sqrt{\tilde{w}_i})^2 + (\sqrt{h_i} - \sqrt{\tilde{h}_i})^2
\]

\[
L_{\text{obj}} = \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} q_{ij}^{noob} p_i(c) \log(p_i(c))
\]

\[
+ \lambda_{\text{obj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} q_{ij}^{ob} p_i(c) \log(\hat{p}_i(c))
\]

\[
L_{\text{cls}} = \lambda_{\text{class}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \sum_{c=0}^{C} q_{ij}^{ob} p_i(c) \log(\hat{p}_i(c))
\]

where $\lambda_{\text{coord}}, \lambda_{\text{noobj}}, \lambda_{\text{class}}$ denote the weight coefficients in box detection, object detection, and classification respectively. $q_{ij}^{ob} = 1$ represents the object that appears in the $j$th the bounding box of cell $i$ and $q_{ij}^{ob} = 0$ represents the non-appearance. The value of $q_{ij}^{noob}$ equals to $1 - q_{ij}^{ob}$. $(x_i, y_i, w_i, h_i)$ is coordinate vector offsets of the bounding box in cell $i$ and $(\tilde{x}_i, \tilde{y}_i, \tilde{w}_i, \tilde{h}_i)$ is related to the positive sample. $p_i(c)$ denotes the probability when the object is the target class, while $\hat{p}_i(c)$ is the true label.

When the network GO_CORNER completed detecting, we performed post-processing on all Go board images including perspective transformation and cropping operation. Firstly, we calculated the center of each bounding box to get the specific coordinates of four chessboard corners. These coordinates were used by perspective transformation to correct the tilted Go boards, which eliminated deformation of Go pieces and crosslines on the chessboard caused by low capture angles. The next step was cropping the images to cut out the chessboard area alone, which was helpful for the locating and classification of Go pieces. Moreover, our system could generate new coordinates of four corners employed to map each Go game into Go_Board1 and Go_Board2 in the next stage.

3) GO PIECES RECOGNITION AND LOCATION

In this stage, we built our pieces recognition networks GO_PIECEX1 and GO_PIECEX2 with the following features: Add CBAM (Convolutional Block Attention Module): We added CBAM to the first and last convolution layers in the backbone of the network (colored green in Fig.4). special modules, channel attention module, and spatial attention module, contained in CBAM helps to extract useful features and ignore irrelevant features. Therefore, the accuracy in target detection can be effectively improved.

Change NMS (Non-Maximum Suppression): In the original NMS algorithm, all bounding boxes below the preset confidence value will be filtered out, remaining the one with the highest probability. Then all predictions were traversed by class and sorted from big to small, which led to a drawback that our system may detect the same intersection point more than once. To solve this problem, all predictions were sorted at once without considering the value of class when surmising. With the improved NMS algorithm, our model can first generate many bounding boxes and then delete extra predictions at one intersection point.

Adjust Evolutionary Hyperparameters: Hyperparameters that match a specific data set are essential since they can help minimize the monitoring indicator’s total loss and improve the training efficiency. To find optimal hyperparameters of our own data set, we used GA (Genetic Algorithm) for Hyperparameter evolution. GA not only can work well in high-dimensional search space but also can avoid excessive calculations compared with traditional methods such as grid searches.

After building our pieces recognition networks GO_PIECEX1 and GO_PIECEX2, we began to train them. Both networks were responsible for identifying and classifying every intersection point on the chessboard. However, the sizes of target detection areas were slightly different: GO_PIECEX1 detected one point in a single bounding box while GO_PIECEX2 detected two neighboring points. By combining outputs from these two networks, the last result would be optimized since various target detection areas could learn more reflective features. We can also weigh two outputs and minimized the defects of each network for pieces recognition. 2,898 GO board images were used as train datasets and resized to $640 \times 640$ pixels. The network was
trained for 1,000 epochs with a batch size of 16. The initial learning rate was set to 0.01.

We ordered pieces recognition results by location information. Coordinates of 19 × 19 = 361 intersection points were calculated by the perspective algorithm. All bounding boxes were split into one-piece-sized parts and then matched for each point (undetected points were set to the value of non-piece). The corresponding values of chosen boxes in point i was recorded as (cls_i, conf_i, conf_j, label_i), which represented class and confidence score of GO_PIECE1, class and confidence score of GO_PIECE2 and true class of point i. At last, for a total of 322 Go board images, we obtained 322 × 361 = 116,242 chess pieces data for the model ensemble in the next step.

4) MODEL ENSEMBLE FOR OPTIMAL RESULT

The network that encircles more Go pieces learns more reflective features but increases the numbers of the target object classes. Thus, a model ensemble is a good way to satisfy the requirement of both reflective features learning and the target object classification. Compared with other model ensemble methods (voting, averaging, blending, etc.), stacking uses a hierarchical model integration framework and hardly needs parameters adjusting or features selection. Therefore, we adopt it in our study. As shown in Fig.4, the first layer with multiple basic learning models receives the original data set; The second layer formed by one learning model is trained with data predicted by the first layer. By generalizing the output results of multiple models, the overall prediction accuracy can be improved.

![FIGURE 4. Architectural diagram of stacking.](image)

In this study, KNN (K-Nearest Neighbor), XGB (eXtreme Gradient Boosting), and RF (Random Forest) constructed the first layer and LR (Logistic Regression) was the second layer (see in Fig.5). Among all chess pieces dataset, we selected 5 × 361 = 1,805 as training data and the rest 317 × 361 = 114,437 as testing data. Unlike deep learning, these weak classifiers in the algorithm can work well with a small amount of low-dimensional data, which increases the utilization of data sets. The specific calculation process is listed as:

Algorithm 1: Model ensemble.

Input: (5,1,805) train data
- // (cls_i, conf_i, cls_j, conf_j, label_i) for data i
- (4,114,437) test data
- // (cls_i, conf_i, cls_j, conf_j) for data i

Output: 114,437 results after model ensemble

function five_fold_stacking

input
- train data, test data, basic model in the first layer
- output
- train data and test data for the second layer

begin
- for item 1 to 5 step 1 do
- learn, predict = train_data_split(train_data, item)
- trained_model = train_model(model, learn)
- prediction[item] = trained_model(predict)
- predict [item] = trained_model(test_data)

end

model_train_pred = vertical_overlay(prediction)
model_test_pred = mean (predict)
return model_train_pred, model_test_pred

// the first layer
KNN_train_pred, KNN_test_pred = five_fold_stacking (train_data, test_data, KNN)
XGB_train_pred, XGB_test_pred = five_fold_stacking (train_data, test_data, XGB)
RF_train_pred, RF_test_pred = five_fold_stacking (train_data, test_data, RF)

// the second layer
train_data2 = concat(KNN_train_pred, XGB_train_pred, RF_train_pred)
train_data2 = concat(KNN_test_pred, XGB_test_pred, RF_test_pred)
trained_model = train_model(LR, train_data2)
results = trained_model(test_data)
return results

III. RESULTS AND DISCUSSION

A. EXPERIMENTAL SETUP

All networks were trained on a computer with an Intel(R) Xeon(R) Gold 6226 2.70GHz CPU, 512 GB of RAM, and a GeForce RTX 3090 GPU card. The algorithm was developed using Python 3.7. To use our system without the limitation of locations and hardware devices, the neural network model is transplanted to the mobile phone with GPU (Graphic Processing Unit) Adreno 200 and deployed with Android studio 2019.

To evaluate the performance of networks, we adopt the following three indicators: precision, recall, and mAP@0.5. The mAP@0.5 (mean Average Precision) is the value of the area under the P-R curve with the value of IOU > 0.5. It is the most commonly used performance evaluation metric for object detection.

\[
\text{precision} = \frac{TP}{TP + FP} \quad (5)
\]

\[
\text{recall} = \frac{TP}{TP + FN} \quad (6)
\]

\[
\text{average error} = \frac{\sum_{i=1}^{N} E_i}{N} \quad (7)
\]
average missing rate \(= \sum_{i=1}^{N} \frac{M_i}{T_i} \) \(\cdot \frac{1}{N}\) \(\ldots \) (8)

where \(TP\) is true positives, \(FP\) is false positives, \(FN\) is false negatives. \(E_i, M_i\) are the numbers of counting errors and missing errors of the \(i\)th image. \(N\) is the number of total images. \(T_i\) is the actual number of objects for detection in each Go board image.

B. Quality Assessment

The performance of three networks are listed in Table 1:

| Model        | precision | recall | mAP@0.5 |
|--------------|-----------|--------|---------|
| GO_CORNER    | 95.72%    | 100%   | 99.48%  |
| GO_PIECEX1   | 93.50%    | 99.98% | 99.50%  |
| GO_PIECEX2   | 84.46%    | 94.51% | 93.40%  |

As we can see, GO_CORNER can correctly locate all target chessboards. Sometimes it will detect other chessboard corners appearing in the same image. The GO_PIECEX1 model only covers a small board area, it may ignore chess pieces at the corners and classify undetected reflective areas as white pieces. The performance of the GO_PIECEX2 model has a similar disadvantage but it shows better performance in reflective areas than GO_PIECEX1 does. Besides, most missed examples of the two networks are in different non-piece areas.

After mapping predictions into electric records, we evaluated the performance of our method in an end-to-end manner. Detailed results are listed as:

| Model        | average error | average missing rate |
|--------------|---------------|-----------------------|
| GO_PIECEX1   | 0.1626%       | 0.1503%               |
| GO_PIECEX2   | 8.8582%       | 1.2417%               |
| After ensemble | 0.0087%       | 0.0000%               |

The average accuracy of our method is up to \(1 - 0.0087\% = 99.9913\%\). Compared with the best result of a single network, the mean accuracy after the model ensemble rises 18.7 (equals to \(0.1626/0.0087\)) times. Therefore, it is necessary to combine these two results.

C. COMPARISON WITH OTHER METHODS

Research teams have studied commercial Go board images recognition systems for years. For example, the team Opensoft from Korea converted video content into a digital Go board. They claimed that their system has an excellent performance except in light reflective conditions. Many mobile application software (such as Tencent Go, Go Eye, Go Camera, Go Sweep) supported the images-to-records function. Among them, Tencent Go has the highest accuracy rate without manual intervention. Thus, we selected Tencent Go as the comparison method using the same test sets.

Table 3: Average Error Rate Under Comparison

| Test sets  | NUMBERS | symbol  | our method | Tencent Go |
|------------|---------|---------|------------|------------|
| All images | 317     | NEIP    | 10         | 219        |
|            |         | AEREIP  | 0.00874%   | 0.19137%   |
|            |         | NEIP    | 8          | 31         |
|            |         | AERI    | 2.52366%   | 9.77918%   |
|            |         | NIEMD   | 1          | 10         |
|            |         | AERIEMD | 0.31546%   | 3.15457%   |
| Images     | 135     | NEIP    | 8          | 160        |
|            |         | AEREIP  | 0.01642%   | 0.32831%   |
|            |         | NEIP    | 6          | 24         |
|            |         | AERI    | 4.44444%   | 17.77778%  |
|            |         | NIEMD   | 0          | 7          |
|            |         | AERIEMD | 0.00000%   | 5.18519%   |
| Images     | 192     | NEIP    | 2          | 50         |
|            |         | light reflection attacks | AEREIP | 0.00304% | 0.9948 | 0.08978% |
|            |         | light reflection attacks | NEIP    | 2          | 0.9948 | 7          |
|            |         | light reflection attacks | AERI    | 1.09890%  | 3.84615% |
|            |         | light reflection attacks | NIEMD   | 1          | 0.54945% | 1.64835% |

\(\text{NEIP} = \text{numbers of erroneous intersection points, AEREIP} = \text{average error rate of erroneous intersection points, NEIP} = \text{numbers of images where errors appear, AERI} = \text{average error rate of images where errors appear, NIEMD} = \text{numbers of images where errors may disrupt the judgment of the winner, AERIEMD} = \text{average error rate of images where errors may disrupt the judgment of the winner.}\)

All 317 test images with 114,437 intersection points are subdivided into 135 images with light reflection attacks and 182 images without light reflection attacks. As shown in Table 2, our method has 10 erroneous detected results in the location and classification of all intersection points within 8 images. Among these 10 errors, 8 erroneous results appear in images with light reflection attacks and 2 erroneous results show in images without light reflection attacks. The third column of Fig.6(c) shows an example that the reflective area is recognized as a white piece whose area is small and the edge seems rounded, which differentiates from conventional reflective situations. Despite these errors, only in one Go game competition did the scores of both players are affected according to the rules. At the same time, the final decision of the winner is correct in all competitions. In contrast, Tencent Go gets 219 errors from 31 Go images where 160 errors appear in images with light reflection attacks and 59 errors in images without light reflection attacks. When counting the game scores, the results of 3.15457% games get wrong, which is far more than the 0.31546% error rate obtained by our method. In general, the results illustrate that the average error of Tencent Go is 21.9 (equals to 0.19137/0.00874) times higher than ours. Moreover, our method can resist light reflection attacks 20 (equals to 0.32831/0.01385) times better. When recognizing images without reflection attacks, Tencent Go software also makes more than 29.5 times as many mistakes which almost gather in images under abnormal capture angles. Here are comparison results between our method and Tencent Go software:
In Fig.5(a), the overall brightness of the image is low, which makes white chess pieces much darker than light reflection areas. If the step of brightness adjustment is ignored, these white pieces will be detected as black. In Fig.5(b)(c), the chessboards are not in the position of orthographic projection, which leads to the deformation of the chess pieces especially in areas with long distances. In these cases, the chess pieces are easily missed or regarded as in other positions.

IV. CONCLUSION

In this paper, we have proposed a Go board images recognition method that outperforms the state-of-the-art techniques with an average accuracy of Go piece recognition over 99.99%. It works well in situations where the game boards are under light reflection attacks and extreme capture angles. Notably, this is the first attempt at combining two kinds of features from target areas with different sizes. Two networks were adopted in the Go pieces recognition process, which ensures the learning of reflective features and the accuracy of classification at the same time. Besides, we make the perspective transformation to eliminate the distortion of the pieces caused by the shooting angles after locating chessboards. The result of the experiment on real Go board images demonstrated that our method has defeated current recognition technology use images demonstrated that our method has defeated current recognition technology use images and accuracy in markets, which attracts several people who hope to put our method into practical applications.

In addition to detecting Go board images, our method can indirectly apply in other fields. For example, self-driving car systems can adopt the way of recognizing chessboard because the feature of road lines is similar to Go board lines. Face recognition which has an outline close to a circle can benefit from the stage of detecting pieces. Besides, the properties of our proposed method suggest the potential performance for recognition of interfered small objects in groups that have few features. Consequently, it matches conditions such as animal crowds, industrial parts, physiological tissues, micro-particles, crops.

APPENDIX

All datasets and project files of this study are available at https://github.com/zhuoyi97/YOLO-GO.git for free access.

ACKNOWLEDGMENT

This work was supported by the Natural Science Funds of Jiangsu Province of China (BK20181256).

REFERENCES

[1] Hsu and Yu-Chia, "Using Machine Learning and Candlestick Patterns to Predict the Outcomes of American Football Games," Applied Sciences, vol. 10, pp. 4484, Jun. 2020, 10.3390/app10134484.
[2] Z. Liu, "Application of Artificial Intelligence Technology in Basketball Games," IOP Conference Series: Materials Science and Engineering, vol. 750, pp. 012093, Mar. 2020, 10.1088/1757-899X/750/1/012093.
[3] H. Zhang, C. Sciutto, M. Agrawala, and K. Fatahalian, "Video2Player: Controllable video sprites that behave and appear like professional tennis players," ACM Transactions on Graphics, vol. 40, no. 24, pp. 1–16, Jul. 2021.
[4] M. Czyzewski, A. Laskowski, and S. Wasik, "Chessboard and Chess Piece Recognition with the Support of Neural Networks," Foundations of Computing and Decision Sciences, vol. 45, pp. 257–260, Dec. 2020, 10.2478/fcds-2020-0014.
[5] C. G. Harris and M. J. Stephens, “A combined corner and edge detector," Alvey vision conference, vol. 15, pp. 10–3244, Jan 1988.
[6] A. De la Escalera and J. M. Armingol, "Automatic Chessboard Detection for Intrinsic and Extrinsic Camera Parameter Calibration," Sensors, vol. 10, no. 3, pp. 2027–2044, March. 2010, 10.3390/s100302027.
[7] J. Canny, "A computational approach to edge detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679–698, 1986, 10.1109/TPAMI.1986.4767851.
[8] S. Bennett and J. Lasenby, “CHESS - Quick and Robust Detection of Chess-board Features," Computer Vision and Image Understanding, vol. 118, pp. 197–210, 2014, 10.1016/j.cviu.2013.10.008.
[9] B. Chen, Y. Liu, and C. Xiong, "Automatic Checkerboard Detection for Robust Camera Calibration." in 2021 IEEE International Conference on Multimedia and Expo (ICME), 2021, pp. 1–6.
[10] B. Chen, C. Xiong, and Q. Zhang, "CCDN: Checkerboard corner detection network for robust camera calibration," in 2018 IEEE ICIRA, May 2018, pp. 324–334.
[11] A. Chen and K. Wang, "Robust computer vision chess analysis and interaction with a humanoid robot," Computers, vol. 8, pp. 14, Feb. 2019, 10.3390/computers8010014.
[12] T. Cour, R. Lauranson, and M. V achette, “Autonomous chess-playing robot,” International Journal of Engineering Research & Technology (IJERT), vol. 4, Mar. 2015, 10.3390/computers8010014.
[13] N. Banerjee, D. Saha, A. Singh, and G. Santhal, “A simple autonomous robotic manipulator for playing chess against any opponent in real time,” in Proceedings of the International Conference on Computational Vision and Robotics, 2011.
[14] C. Matuszek, B. Mayton, R. Aimi, M. P. Deisenroth, L. Bo, R. Chu, and D. Fox, "Gambit: An autonomous chess-playing robotic system," in 2011 IEEE International Conference on Robotics and Automation IEEE, 2011, pp. 4291–4297.
[15] P. Kołosowski, A. Wolniakowski, and K. Miatlik, "Collaborative robot system for playing chess," in 2020 International Conference Mechatronic Systems and Materials (MSM), 2020, pp. 1–6.
[16] G. D. Illeperuma, “Using Image Processing Techniques to Automate Chess Game Recording,” Sri Lankan Journal of Physics, vol. 27, pp. 76-83, Jan. 2011.

[17] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[18] Y. Xie, G. Tang, and W. Hoff, “Chess Piece Recognition Using Oriented Chamfer Matching with a Comparison to CNN,” in IEEE Winter Conference on Applications of Computer Vision, 2018, pp. 2001-2009.

[19] A. Neto and R. M. Campello, “Chess Position Identification using Pieces Classification Based on Synthetic Images Generation and Deep Neural Network Fine-Tuning,” 2019 21st Symposium on Virtual and Augmented Reality (SVR), 2019, pp. 152-160.

[20] Quintana, David Mallasén, Alberto Antonio del Barrio García, and Manuel Prieto Matías. “LiveChess2FEN: a Framework for Classifying Chess Pieces based on CNNs,” 2012. arXiv:2012.06858. [Online]. Available: https://arxiv.org/abs/2012.06858

[21] S. Huang, “Research on chess record recognition algorithm based on chain coding,” M.S. thesis, East China Normal University, 2007 (in Chinese)

[22] C. Liang, Y. Ding, J. Yuan, and G. Zhu, “Research on Image Recognition Algorithm of Go Game,” Modern Computer (Professional Edition), vol. 24, pp. 39-43, 2016. (in Chinese)

[23] Y. Gui, Y. Wu, Y. Wang and C. Yao, “Visual Image Processing of Humanoid Go Game Robot Based on OPENCV,” in 2020 Chinese Control and Decision Conference (CCDC), 2020, pp. 3713-3716

[24] Z. Jian, W. Zheng, and P. Hu, “Automatic Go Recognition System Based on Image Processing,” in 2015 10th National Sports Science Conference Abstracts Collection (1), 2015, pp.823-824. (in Chinese)

[25] K. Kwang, Unmanned system for video recognition and chess score conversion (in Korea), Mar 23, 2018. Online at: http://www.cyberoro.com/news/news_view.oro?num=524126

[26] C. Chen, M. Liu, O. Tuzel, and J. Xiao, “R-CNN for Small Object Detection,” in Springer, Cham, 2017, pp. 214-230

[27] J. Redmon, S. Divvala, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779-788.

[28] S. M. Silva, C. R. Jung, “Real-time license plate detection and recognition using deep convolutional neural networks,” in Journal of Visual Communication and Image Representation, vol. 71, pp. 1047-1063, Aug. 2020, 10.1016/j.jvcir.2020.102773.

[29] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," International journal of computer vision, vol. 128, no. 2, pp. 261-318, Sept. 2018, 10.1007/s11263-019-01247-4.

[30] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” IEEE Transactions on Pattern Analysis & Machine Intelligence, vol. 39, no. 6, pp. 91-99, Jun. 2017, 10.1109/TPAMI.2016.2577031.