

Received September 6, 2021, accepted November 14, 2021, date of publication November 25, 2021, date of current version December 9, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3130594

Reliable Go Game Images Recognition Under Strong Light Attack

YIYAO ZHUO¹, HAN FANG¹, KAI ZHU², HONGCHEN ZHAN², AND JIE YUAN^{1,2} 

¹School of Electronic Science and Engineering, Nanjing University, Nanjing, Jiangsu 210046, China

²School of Information Science and Engineering, Jinling College of Nanjing University, Nanjing, Jiangsu 210089, China

Corresponding author: Kai Zhu (zhukai@jlxj.nju.edu.cn)

This work was supported in part by the Natural Science Funds of Jiangsu Province of China under Grant BK20181256.

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 the potential references for 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, etc.

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. So 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,

The associate editor coordinating the review of this manuscript and approving it for publication was Zhouyang Ren .

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. Fortunately, 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, Hough 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 $4,208 \times 2,368$ 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.



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 corners in each chessboard was annotated with LabelImg software [26]. 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. In the first way, there were $19 \times 19 = 361$ rectangular boxes in an image and each box included one point. In the second way, an annotated image had $18 \times 19 = 342$ boxes and each of them contained two neighboring points in the horizontal direction.

B. BOARD DETECTION AND PIECES RECOGNITION

We need to locate the chessboard first before recognizing Go pieces since the chess pieces that were scattered outside

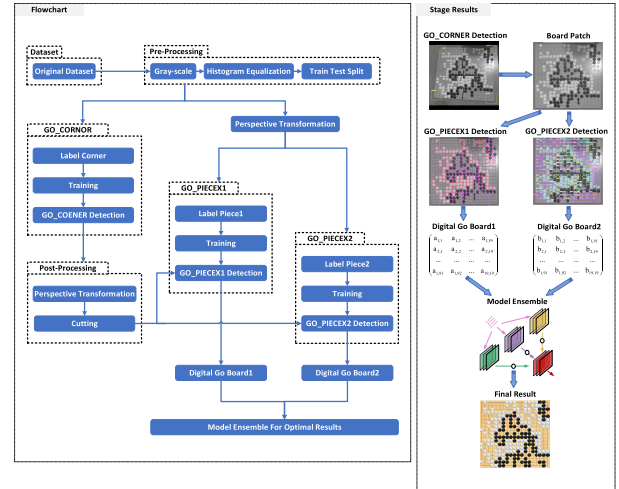


FIGURE 2. An overview of our proposed Go game image recognition.

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 [27]–[29].

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_CORNOR, we employed perspective transformation to straighten all tilted boards captured with various angles and cut out the board area. The purpose of this stage was to create new Go game images with standard shooting angles, which was more conducive to the detection of Go pieces. Subsequently, two networks named GO_PIECEX1 and GO_PIECEX2 recognized all intersection points on the chessboards 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 electric records Go_Board1 and Go_Board2 (where $a_{i,j}$ and $b_{i,j}$ respectively represented the mapped result of the intersection point (i, j) in Go_Board1 and Go_Board2). At last, model ensemble was applied to get an optimal result.

1) BOARD DETECTION

In the stage of board detection, we chose a one-stage target detection algorithm YOLO (You Only Look Once) [30]. YOLO discards the step of regional nomination and predicts the bounding box and class probabilities directly. The framework YOLOv5 is the latest release of YOLO that offers one-third size of YOLOv3 but has quicker surmising speed. YOLOv5s (a small version), YOLOv5m (a medium version), YOLOv5l (a large version) and YOLOv5x (an extra-large version) are four versions in this series that need a trade-off between accuracy and speed when chosen.

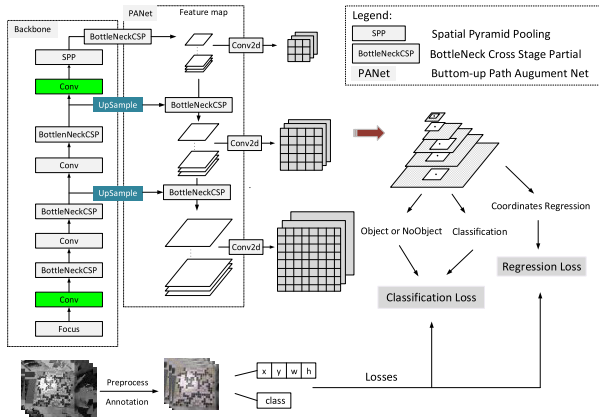


FIGURE 3. The framework of the Go game images detection.

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

Feature Extraction: After being resized to 640×640 , 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 up-sampling, 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 $S \times \sim S$ 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 Equation1-4).

$$Loss = L_{box} + L_{obj} + L_{cls} \quad (1)$$

$$L_{box} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B q_{i,j}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \quad (2)$$

$$L_{obj} = \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B q_{i,j}^{noobj} p_i(c) \log(\hat{p}_i(c)) + \lambda_{obj} \sum_{i=0}^{S^2} \sum_{j=0}^B q_{i,j}^{obj} p_i(c) \log(\hat{p}_i(c)) \quad (3)$$

$$L_{cls} = \lambda_{class} \sum_{i=0}^{S^2} \sum_{j=0}^B q_{i,j}^{obj} \sum_{c \in classes} p_i(c) \log(\hat{p}_i(c)) \quad (4)$$

where λ_{coord} , λ_{noobj} , λ_{class} denote the weight coefficients in box detection, object detection, and classification respectively. $q_{i,j}^{obj} = 1$ represents the object that appears in the j^{th} the bounding box of cell i and $q_{i,j}^{noobj} = 0$ represents the non-appearance. The value of $q_{i,j}^{noobj}$ equals to $1 - q_{i,j}^{obj}$. (x_i, y_i, w_i, h_i) is coordinate vector offsets of the bounding box in cell i and $(\hat{x}_i, \hat{y}_i, \hat{w}_i, \hat{h}_i)$ 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.

2) BOARD IMAGE POST-PROCESSING

When the network GO_CORNOR completed detecting, we performed post-processing on 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. All Go game images were split into 2,576 train datasets, 322 validation datasets and 322 test datasets and resized to 640×640 pixels before being received by models. The network was trained for 1,000 epochs with a batch size of 16. The initial learning rate was set to 0.01. From a functional point of view, Both GO_PIECEX1 and GO_PIECEX2 were responsible for identifying and classifying every intersection point on the chessboard. However, the sizes of target detection areas when they worked were slightly different: GO_PIECEX1 detected one point in a single bounding box while GO_PIECEX2 detected two neighboring points in the horizontal direction. By combining outputs from these two networks, the last result would be optimized since two various target detection areas could learn more features, especially the features of areas under strong light attacks. Moreover, we can weigh two outputs and minimize the defects of each network for pieces recognition as much as possible. See the results and discussion section of the paper for specific analysis.

When networks finished detecting all 322 test datasets, we ordered bounding box predictions and mapped them into two electric records by location information. As shown in Algorithm 1, the precise positions of $19 \times 19 = 361$ intersection points were calculated by the perspective algorithm with new coordinates of four corners outputted by GO_CORNOR. Each point was matched with a bounding box (the prediction from GO_PIECEX2 needed to be split into two one-piece-sized boxes). The corresponding values of the chosen box in point i was recorded as basic data $(cls_1^i, conf_1^i, cls_2^i, conf_2^i, label^i)$, which represented class and confidence score of GO_PIECEX1, class and confidence score of GO_PIECEX2 and true class of point i . We set the class and confidence score to $(-1, 0)$ when the point was undetected. At last, we obtained a total of $322 \times 361 = 116, 242$ intersection points data which would be applied for the model ensemble in the next sub-stage.

4) MODEL ENSEMBLE FOR OPTIMAL RESULT

The network GO_PIECEX2 that encircled two points in a single bounding box learned more reflective features but increased the numbers of the target object classes while the GO_PIECEX1 was the opposite. In other words, a single Go

pieces recognition network cannot meet the requirement of reflective feature learning and high target object detection accuracy at the same time. To solve this problem, we used model ensemble to optimize the output of the two networks. Among all model ensemble methods like voting, averaging, blending, stacking, etc., we adopted stacking since it uses a hierarchical model integration framework and hardly needs parameters adjusting or features selection. Fig.4 shows the diagram of stacking. Multiple basic learning models formed the first layer which received the original data set; The second layer consisting of one learning model was trained with data predicted by the first layer. By generalizing the output results of these multiple models, stacking improved the overall prediction accuracy.

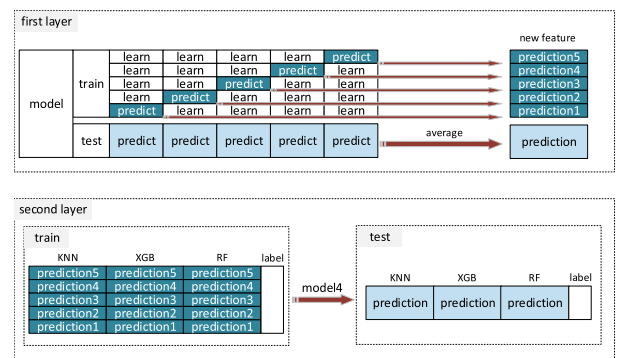


FIGURE 4. Architectural diagram of stacking.

As shown in Fig.4, the first layer was constructed by KNN (K-Nearest Neighbor), XGB (eXtreme Gradient Boosting) and RF (Random Forest) in this study; The second layer was LR (Logistic Regression). Unlike deep learning, these weak classifiers in the algorithm can work well with a small amount of low-dimensional data. Therefore, we only need to select $5 \times 361 = 1, 805$ as training data and the remaining $317 \times 361 = 114, 437$ as testing data among all intersection points data, which increased the utilization of data sets. The specific calculation process is as follows:

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 has been deployed with Android studio2019 and transplanted to the mobile phone with GPU (Graphic Processing Unit) Adreno 200.

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

Algorithm 1 Model Ensemble

```

Input: (5,1,805) train data
          //(cls1i, conf1i, cls2i, conf2i, labeli) for data i
          (4,114,437) test data
          //(cls1i, conf1i, cls2i, conf2i) 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
  end
  // 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)
  test_data2 ← concat(KNN_test_pred, XGB_test_pred,
  RF_test_pred)
  trained_model ← train_model(LR, train_data2)
  results ← trained_model(test_data2)
return results

```

object detection.

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

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

$$\text{average error} = \frac{\sum_{i=1}^N \frac{E_i}{T_i}}{N} \quad (7)$$

$$\text{average missing rate} = \frac{\sum_{i=1}^N \frac{M_i}{T_i}}{N} \quad (8)$$

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 which equals 361.

B. TEST RESULTS

The performance assessments of the three networks are as follow:

As we can see, GO_CORNER can correctly locate all target chessboards. Sometimes it will repeatedly detect the

TABLE 1. Index scores of three networks.

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%

corners of other chessboards close to the target one in the same image. The network GO_PIECEX1 may ignore chess pieces at the corners and classify undetected reflective areas as white pieces. As mentioned before, it covers a small target detection area and learns a few reflective features. The performance of the network GO_PIECEX2 has a similar disadvantage but it shows better performance in reflective areas than GO_PIECEX1 does. We also found that most of the missed examples of GO_PIECEX1 and GO_PIECEX2 are in different non-piece areas, which means that combining their two output results can greatly reduce the missing rate.

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

TABLE 2. Index scores of our method.

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 employ the method of model ensemble that combines results from two pieces recognition networks.

C. COMPARISON WITH OTHER METHODS

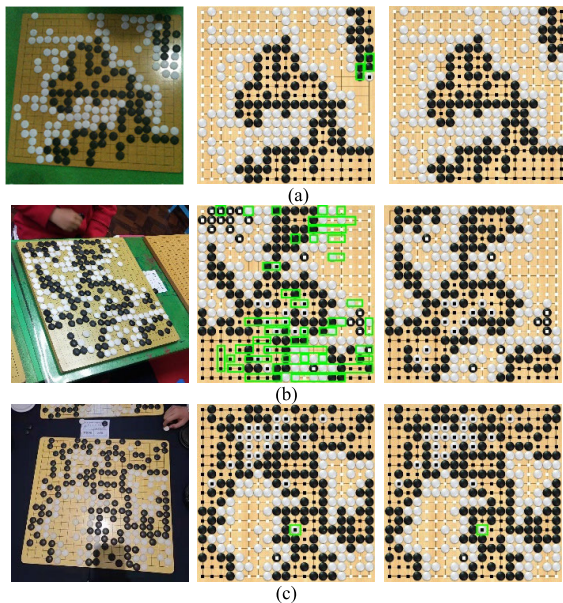
Some research teams have studied commercial Go game images recognition systems for years. For example, the team Opensoft from Korea has implemented the function of converting video content into a digital Go board. They claimed that their system has an excellent performance except on images under strong light attack. Many mobile application software (such as Tencent Go, Go Eye, Go Camera, Go Sweep) also supported the images-to-records function. To show the performance of our method more intuitively, we selected Tencent Go as the comparison system using the same test sets because it has the highest accuracy rate among all software.

All 317 test images having 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 on 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. As shown in the third column of Fig.5(c), the reflective area is wrongly

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%
		NIEP	8	31
		AERI	2.52366%	9.77918%
		NIEMD	1	10
Images with light reflection attacks	135	AERIEMD	0.31546%	3.15457%
		NEIP	8	160
		AEREIP	0.01642%	0.32831%
		NIEP	6	24
		AERI	4.44444%	17.77778%
Images without light reflection attacks	182	NIEMD	0	7
		AERIEMD	0.00000%	5.18519%
		NEIP	2	59
		AEREIP	0.00304%	0.08978%
		NIEP	2	7
		AERI	1.09890%	3.84615%
		NIEMD	1	3
		AERIEMD	0.54945%	1.64835%

^a NEIP = numbers of erroneous intersection points, AEREIP = average error rate of erroneous intersection points, NIEP = numbers of images where errors appear, AERI = average error rate of images where errors appear, NIEMD = numbers of images where errors may disrupt the judgment of the winner, AERIEMD = average error rate of images where errors may disrupt the judgment of the winner.

**FIGURE 5.** Comparison results between our method and Tencent Go software. The first columns mean original images. The second columns are outputs from Tencent Go. The third columns are mapped images from our method for comparison. All erroneous results are labeled in green boxes.

recognized as a white piece. Its edge seems rounded which differentiates from conventional reflective situations. Despite these errors, the players' scores are affected in only one Go game competition and the final judgments of the winner are completely correct in all test images. In contrast, Tencent Go gets 219 errors on 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, 3.15457% of results get wrong, which is far more than the value of 0.31546% from 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 where the erroneous predictions mainly gather on images under strong light attacks or abnormal shooting angles. Moreover, our method works 20 (equals to 0.32831/0.01642) times better on images with reflection attacks and 29.5 (equals to 0.08978/0.00304) times better on images without reflection attacks. Here are comparison results between our method and Tencent Go software:

Images in Fig.5(a) have a low brightness, which makes white chess pieces much darker than light reflection areas. These pieces will be easily detected as black if we skip the step of brightness adjustment. In Fig.5(b)(c), the chessboards are under extreme image capture angles where Go pieces and crosslines deforms on the board. In this case, the pieces are easily missed or located in other positions.

IV. CONCLUSION

In this paper, we have proposed a Go game 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 used in markets, which attracts several people who hope to put our method into practical applications.

In addition to detecting Go game 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.

All datasets and project files of this study are available at <https://github.com/zhuoyiyao97/YOLO-GO.git> for free access.

REFERENCES

- [1] Y.-C. Hsu, "Using machine learning and candlestick patterns to predict the outcomes of American football games," *Appl. Sci.*, vol. 10, no. 13, p. 4484, Jun. 2020, doi: [10.3390/app10134484](https://doi.org/10.3390/app10134484).
- [2] Z. Liu, "Application of artificial intelligence technology in basketball games," *IOP Conf. Mater. Sci. Eng.*, vol. 750, no. 1, Feb. 2020, Art. no. 012093, doi: [10.1088/1757-899X/750/1/012093](https://doi.org/10.1088/1757-899X/750/1/012093).

- [3] H. Zhang, C. Sciutto, M. Agrawala, and K. Fatahalian, "Vid2Player: Controllable video sprites that behave and appear like professional tennis players," *ACM Trans. Graph.*, vol. 40, no. 3, pp. 1–16, Jul. 2021.
- [4] M. A. Czynewski, A. Laskowski, and S. Wasik, "Chessboard and chess piece recognition with the support of neural networks," *Found. Comput. Decis. Sci.*, vol. 45, no. 4, pp. 257–280, Dec. 2020, doi: [10.2478/fcds-2020-0014](https://doi.org/10.2478/fcds-2020-0014).
- [5] C. G. Harris and M. Stephens, "A combined corner and edge detector," in *Proc. Alvey Vis. Conf.*, vol. 15, no. 50, 1988, pp. 10–5244.
- [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, Mar. 2010, doi: [10.3390/s100302027](https://doi.org/10.3390/s100302027).
- [7] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986, doi: [10.1109/TPAMI.1986.4767851](https://doi.org/10.1109/TPAMI.1986.4767851).
- [8] S. Bennett and J. Lasenby, "ChESS—Quick and robust detection of chessboard features," *Comput. Vis. Image Understand.*, vol. 118, pp. 197–210, Jan. 2014, doi: [10.1016/j.cviu.2013.10.008](https://doi.org/10.1016/j.cviu.2013.10.008).
- [9] B. Chen, Y. Liu, and C. Xiong, "Automatic checkerboard detection for robust camera calibration," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, Jul. 2021, pp. 1–6.
- [10] A. Chen and K. Wang, "Robust computer vision chess analysis and interaction with a humanoid robot," *Computers*, vol. 8, no. 1, p. 14, Feb. 2019, doi: [10.3390/computers8010014](https://doi.org/10.3390/computers8010014).
- [11] A. Chen and K. Wang, "Robust computer vision chess analysis and interaction with a humanoid robot," *Computers*, vol. 8, no. 1, pp. 11–14, Feb. 2019, doi: [10.3390/computers8010014](https://doi.org/10.3390/computers8010014).
- [12] N. Banerjee, D. Saha, A. Singh, and G. Sanyal, "A simple autonomous robotic manipulator for playing chess against any opponent in real time," in *Proc. Int. Conf. Comput. Vis. Robot.*, 2011, pp. 1–7.
- [13] C. Matuszek, B. Mayton, R. Aimi, M. P. Deisenroth, L. Bo, R. Chu, M. Kung, L. LeGrand, J. R. Smith, and D. Fox, "Gambit: An autonomous chess-playing robotic system," in *Proc. IEEE Int. Conf. Robot. Automat.*, May 2011, pp. 4291–4297.
- [14] P. Kolosowski, A. Wolniakowski, and K. Miatliuk, "Collaborative robot system for playing chess," in *Proc. Int. Conf. Mech. Syst. Mater. (MSM)*, Jul. 2020, pp. 1–6.
- [15] G. D. Illeperuma, "Using image processing techniques to automate chess game recording," *Sri Lankan J. Phys.*, vol. 27, pp. 76–83, Jan. 2011.
- [16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
- [17] Y. Xie, G. Tang, and W. Hoff, "Chess piece recognition using oriented chamfer matching with a comparison to CNN," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2018, pp. 2001–2009.
- [18] A. de Sa Delgado Neto and R. M. Campello, "Chess position identification using pieces classification based on synthetic images generation and deep neural network fine-tuning," in *Proc. 21st Symp. Virtual Augmented Reality (SVR)*, Oct. 2019, pp. 152–160.
- [19] D. M. Quintana, A. A. del Barrio García, and M. P. Matías, "LiveChess2FEN: A framework for classifying chess pieces based on CNNs," 2020, *arXiv:2012.06858*.
- [20] S. Huang, "Research on chess record recognition algorithm based on chain coding," (in Chinese), M.S. thesis, East China Normal Univ., Shanghai, China, 2007, doi: [10.7666/d.y1073392](https://doi.org/10.7666/d.y1073392).
- [21] C. Liang, Y. Ding, J. Yuan, and G. Zhu, "Research on image recognition algorithm of go game," (in Chinese), *Modern Comput. Prof. Ed.*, vol. 24, pp. 39–43, Aug. 2016, doi: [10.3969/j.issn.1007-1423.2016.24.010](https://doi.org/10.3969/j.issn.1007-1423.2016.24.010).
- [22] Y. Gui, Y. Wu, Y. Wang, and C. Yao, "Visual image processing of humanoid go game robot based on OPENCV," in *Proc. Chin. Control Decis. Conf. (CCDC)*, Aug. 2020, pp. 3713–3716.
- [23] Z. Jian, W. Zheng, and P. Hu, "Automatic go recognition system based on image processing," (in Chinese), in *Proc. 10th Nat. Sports Sci. Conf. Abstr. Collection*, 2015, pp. 823–824.
- [24] K. Kwang. (Mar. 23, 2018). *Unmanned System for Video Recognition and Chess Score Conversion (in Korea)*. [Online]. Available: http://www.cyberoro.com/news/news_view.oro?num=524126
- [25] C. Chen, M. Liu, O. Tuzel, and J. Xiao, "R-CNN for small object detection," in *Computer Vision*. Cham, Switzerland: Springer, 2017, pp. 214–230.
- [26] S. M. Silva and C. R. Jung, "Real-time license plate detection and recognition using deep convolutional neural networks," *J. Vis. Commun. Image Represent.*, vol. 71, pp. 1047–3203, Aug. 2020, doi: [10.1016/j.jvcir.2020.102773](https://doi.org/10.1016/j.jvcir.2020.102773).
- [27] L. Liu, O. Wang, X. Wang, P. Fieguth, J. Chen, and X. Liu, "Deep learning for generic object detection: A survey," *Int. J. Comput. Vis.*, vol. 128, pp. 261–318, Sep. 2018, doi: [10.1007/s11263-019-01247-4](https://doi.org/10.1007/s11263-019-01247-4).
- [28] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 91–99, Jun. 2017, doi: [10.1109/TPAMI.2016.2577031](https://doi.org/10.1109/TPAMI.2016.2577031).
- [29] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.



YIYAO ZHUO was born in Xuzhou, Jiangsu, China, in 1997. She received the B.S. degree from the Jinling College, Nanjing University, Nanjing, China, in 2020, where she is currently pursuing the degree. Her research interests include deep learning and medical image processing.



HAN FANG was born in Taizhou, Jiangsu, China, in 1997. She received the B.S. degree from the School of Electronic Science and Engineering, Nanjing University, in 2019. She is currently pursuing the degree with the Medical Image Processing Laboratory. Her research interests include deep learning and medical image processing.



KAI ZHU was born in Taixing, Jiangsu, China, in 1982. He received the B.S. and Ph.D. degrees in engineering from the Nanjing University of Technology, Nanjing, China, in 2004 and 2011, respectively. He was a Visiting Scholar at The University of North Carolina at Chapel Hill, in 2017. Since 2017, he has been a Researcher and a Teaching Staff at the School of Information Science and Engineering, Jinling College, Nanjing University. His research interests include solar cells and intelligent manufactures.



HONGCHEN ZHAN was born in Nantong, Jiangsu, China, in 1985. She received the B.S. and M.S. degrees in systems analysis and integration from the Nanjing University of Information Science and Technology, Nanjing, China, in 2007 and 2010, respectively. She was a Visiting Scholar at Nanjing University, in 2019. She is currently a Research and Teaching Staff with the School of Information Science and Engineering, Jinling College, Nanjing University. She is the author of more than 15 articles. Her research interests include electronic circuits and embedded development.



JIE YUAN was born in Wuxi, Jiangsu, China, in 1975. He received the B.S., M.S., and Ph.D. degrees from Nanjing University, Nanjing, China, in 1997, 2000, and 2003, respectively. He was a Visiting Scholar at the University of Michigan, Ann Arbor, MI, USA, in 2012, 2013, and 2020. Since 2003, he has been a Researcher and a Teaching Staff at the School of Electronics and Engineering, Nanjing University. He is the author of more than 150 articles. He holds more than 90 national and international patents. His research interests include medical imaging, image reconstruction, image recovery, and real-time video signal processing.

• • •