

Imitation of Go Skill Levels and Recognition of Go Playing Styles

112753207 張詠軒

Abstract

This project endeavors to replicate and understand human behavior in Go through comprehensive training and prediction models on a 19x19 Go board. Two levels of playing strength, represented by Dan and Kyu players, are simulated, with predictions evaluated for accuracy against actual moves. Further analysis categorizes playing styles into aggressive, balanced, and territorial in specific game situations.

Convolutional Neural Network (DCNN Model) is chosen for its capability to capture intricate board features, accommodating both local and global considerations, thereby enhancing prediction accuracy in the complex game of Go. Model training specifics are outlined for both Dan and Kyu players and the recognition of playing styles, utilizing various optimization techniques.

In summary, this project contributes to the comprehensive understanding of Go by leveraging advanced computational techniques. By unraveling the intricacies of move prediction and playing style recognition, the project not only delves into the essence of this ancient game but also opens avenues for future applications in artificial intelligence and game theory.

Introduction

Go game have been invented thousands of years ago, and it is originated in China. While the exact date of its invention is uncertain, it is confirmed that Go was widely popular as early as the 10th century BC. Currently, Go is prevalent in many East Asian countries and has gradually spread worldwide. In fact, in the hearts of many, Go is not merely a form of entertainment, due to its inherent qualities, it has long been regarded as an art form.

The allure of Go stems not only from its simple rules and intricate variations, providing ample space for creative imagination, but also from its millennia of study. Numerous tactical concepts and thinking methods have been developed and explored over the years, allowing individuals to swiftly immerse themselves in the world of Go through learning these principles.

Go game integrates various intelligent behaviors and thoughts of humans on the go board. This project aims to mimic and recognize human behavior through Go game records. The goal is to conduct training and predictions on a 19x19 Go board.

It simulates two levels of Go playing strength, namely Dan (higher-level) players with strong skills and Kyu (lower-level) players with weak skills, predictions are made for the next move on the current Go board, with accuracy indicating whether the predicted move matches the actual move on the given board.

After obtaining the moves, further analysis is conducted to identify the playing style in specific game situations, categorized as aggressive (enjoys challenges and fears no battle), balanced (takes a comprehensive view of the overall situation), and territorial (practical and steady).

Related works

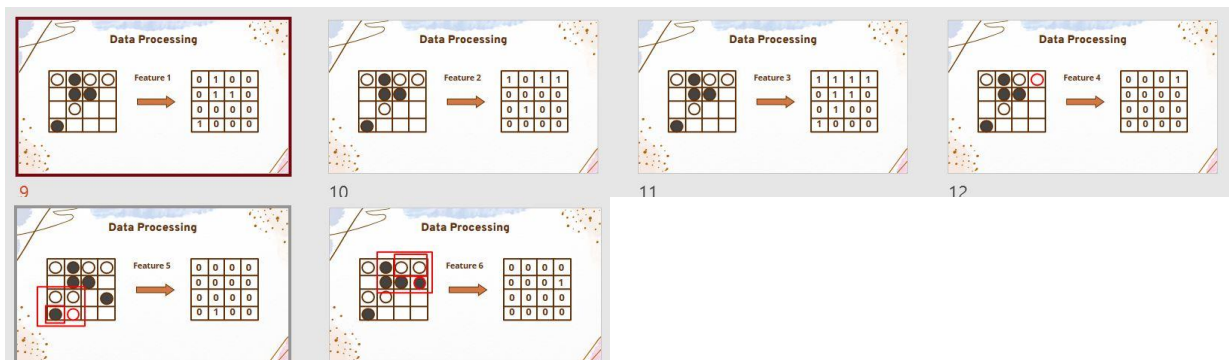
In class we have talked about 'Batch Normalization Layers', it can help Accelerate model convergence and prevent gradient vanishing. In this project, I have added batch normalization after each convolutional layer.

And about 'Data Augmentation' in class, I have added activation function 'ReLU', which function is be like $\max(0, v)$, generating more samples to enhance model generalization.

Method

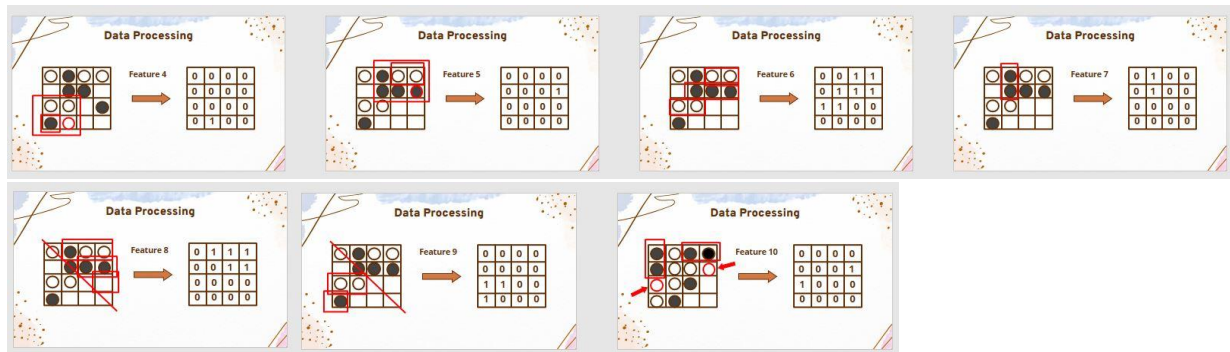
Data Processing:

Firstly, feature extraction is performed for predicting the next move of a player in the game of Go. Six features are utilized and transformed into one-hot encoded. The first feature designates positions with black stones as 1 and others as 0. The second feature designates positions with white stones as 1 and others as 0. The third feature marks positions with stones as 1 and empty positions as 0. The fourth feature indicates the position of the last move as 1 and others as 0. The fifth feature marks positions where a black stone may be surrounded in the next move as 1 and others as 0. The 6th feature marks positions where a white stone may be surrounded in the next move as 1 and others as 0.



For predicting player's playing style, thirteen features are employed and transformed into one-hot encoded. The first three features are similar to the move prediction case. Additionally, the fourth feature marks positions where a black stone may be surrounded in the next move as 1 and others as 0. The fifth feature does the same for white stones. The 6th feature designates positions with a horizontal connection of stones as 1 and others as 0. The 7th feature designates positions with a vertical connection of stones as 1 and others as 0. The 8th feature designates positions with a stone in the upper-right diagonal as 1 and others as 0. The 9th feature designates positions with a stone in the lower-left diagonal as 1

and others as 0. The 10th feature identifies positions where the opponent may place a stone in the next move for an offensive strategy, assuming the player is controlling black stones. The 11th feature serves a similar purpose but assumes the player is controlling white stones. The 12th feature identifies positions where the player may place a stone in the next move for a defensive strategy, assuming the player is controlling black stones. The 13th feature serves a similar purpose but assumes the player is controlling white stones.



We choose a Convolutional Neural Network model (DCNN Model) as it can learn intricate board features, aiding in predicting optimal moves. The DCNN can capture both local and global board features, effectively handling the image data of the board and establishing appropriate balances in the model to enhance prediction and training accuracy for the complex game, Go.

Model Training:

We have both an old and a new version in model training. The old version is used during competitions, while the new version represents the improvements we've made thereafter.

For training the model for Dan (higher-level) players, the model consists of multiple convolutional layers using the Conv2D function with filter sizes of 3x3. Each convolutional layer with BatchNormalization for improved training stability. MaxPooling2D is applied for pooling operations, and the Flatten layer is used to convert the output into a one-dimensional tensor. The model includes fully connected layers with ReLU activation functions and Dropout layers to reduce overfitting. The output layer uses the Softmax activation function to output a 19x19 tensor representing the probabilities of each position for the next move. The model is compiled using the RMSprop optimizer.

For the model trained on Kyu (lower-level) players, the model consists of multiple convolutional layers using the Conv2D function with filter sizes of 3x3. Each convolutional layer with BatchNormalization for improved training stability. MaxPooling2D is applied for pooling operations, and the Flatten layer is used to convert the output into a one-dimensional tensor. The model includes fully connected layers with ReLU activation functions and Dropout layers to reduce overfitting. The output layer uses the Softmax activation function to output a 19x19 tensor representing the probabilities of each position for the next move. The model is compiled using the RMSprop optimizer.

For training the model to recognize playing styles, the model includes

convolutional layers with Dropout regularization, and Denselayers with L2 regularization. The output layer uses Softmax activation for a 3-dimensional vector representing the three playingstyles (aggressive, balanced, territorial). The model is compiled using the RMSprop optimizer.

Training models:

Dan modle: old version vs new version

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 19, 19, 4)]	0
conv2d_7 (Conv2D)	(None, 19, 19, 32)	1184
batch_normalization_7 (Batch Normalization)	(None, 19, 19, 32)	128
conv2d_8 (Conv2D)	(None, 19, 19, 32)	9248
batch_normalization_8 (Batch Normalization)	(None, 19, 19, 32)	128
conv2d_9 (Conv2D)	(None, 19, 19, 32)	9248
batch_normalization_9 (Batch Normalization)	(None, 19, 19, 32)	128
conv2d_10 (Conv2D)	(None, 19, 19, 32)	9248
batch_normalization_10 (Batch Normalization)	(None, 19, 19, 32)	128
...		
Total params: 411214 (1.57 MB)		
Trainable params: 410796 (1.57 MB)		
Non-trainable params: 418 (1.63 KB)		

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 19, 19, 6)]	0
conv2d_7 (Conv2D)	(None, 19, 19, 32)	1760
batch_normalization_7 (Batch Normalization)	(None, 19, 19, 32)	128
conv2d_8 (Conv2D)	(None, 19, 19, 32)	9248
batch_normalization_8 (Batch Normalization)	(None, 19, 19, 32)	128
conv2d_9 (Conv2D)	(None, 19, 19, 32)	9248
batch_normalization_9 (Batch Normalization)	(None, 19, 19, 32)	128
conv2d_10 (Conv2D)	(None, 19, 19, 32)	9248
batch_normalization_10 (Batch Normalization)	(None, 19, 19, 32)	128
...		
Total params: 411790 (1.57 MB)		
Trainable params: 411372 (1.57 MB)		
Non-trainable params: 418 (1.63 KB)		

Kyu modle: old version vs new version

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 19, 19, 4)]	0
conv2d_6 (Conv2D)	(None, 19, 19, 32)	6304
conv2d_7 (Conv2D)	(None, 19, 19, 32)	50208
conv2d_8 (Conv2D)	(None, 19, 19, 32)	25632
conv2d_9 (Conv2D)	(None, 19, 19, 32)	25632
conv2d_10 (Conv2D)	(None, 19, 19, 32)	9248
conv2d_11 (Conv2D)	(None, 19, 19, 1)	289
flatten_1 (Flatten)	(None, 361)	0
softmax_1 (Softmax)	(None, 361)	0
...		
Total params: 117313 (458.25 KB)		
Trainable params: 117313 (458.25 KB)		
Non-trainable params: 0 (0.00 Byte)		

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 19, 19, 6)]	0
conv2d (Conv2D)	(None, 19, 19, 32)	1760
batch_normalization (Batch Normalization)	(None, 19, 19, 32)	128
conv2d_1 (Conv2D)	(None, 19, 19, 32)	9248
batch_normalization_1 (Batch Normalization)	(None, 19, 19, 32)	128
conv2d_2 (Conv2D)	(None, 19, 19, 32)	9248
batch_normalization_2 (Batch Normalization)	(None, 19, 19, 32)	128
conv2d_3 (Conv2D)	(None, 19, 19, 32)	9248
batch_normalization_3 (Batch Normalization)	(None, 19, 19, 32)	128
...		
Total params: 411790 (1.57 MB)		
Trainable params: 411372 (1.57 MB)		
Non-trainable params: 418 (1.63 KB)		

Playing-style modle:

Layer (type)	Output Shape	Param #
conv2d_28 (Conv2D)	(None, 17, 17, 32)	3776
dropout_71 (Dropout)	(None, 17, 17, 32)	0
conv2d_29 (Conv2D)	(None, 15, 15, 64)	18496
dropout_72 (Dropout)	(None, 15, 15, 64)	0
flatten_14 (Flatten)	(None, 14400)	0
dense_57 (Dense)	(None, 16)	230416
dropout_73 (Dropout)	(None, 16)	0
dense_58 (Dense)	(None, 32)	544
dropout_74 (Dropout)	(None, 32)	0
dense_59 (Dense)	(None, 16)	528
dropout_75 (Dropout)	(None, 16)	0
...		
Total params: 253,811		
Trainable params: 253,811		
Non-trainable params: 0		

Experiment

The experiment is divided into two main parts: The Imitation Go Skill Competition and The Recognition of Go Playing Styles. In the Imitation Go Skill Competition, we aim to replicate the playstyles of 1-dan and 10-kyu Go players. For each player level, we provide predictions for the most likely move and predictions for five possible moves, considering a prediction as correct if it includes the actual move. The accuracy is assessed separately for the 1st and 5th predictions, denoted as One_Dan_1, One_Dan_5, Ten_Kyu_1, and Ten_Kyu_5.

In the Recognition of Go Playing Styles, the task involves identifying the playing style from a given game record and outputting the predicted playing style of the final move. We assume that the predicted playing style is accurate if it matches the actual playing style, denoted as PSA.

The final composite score is calculated by taking the accuracy of each task to four decimal places, multiplying it by the predefined weight, and summing them up.

This study aims to comprehensively assess the model's performance by simulating and recognizing various skill levels and playing styles in the game of Go, providing valuable insights for the advancement of artificial intelligence in the field of Go.

Dan Model accuracy:
Old version:

```
och 1/10
6/836 [=====] - 15s 14ms/step - loss: 4.8747 - accuracy: 0.0469 - val_loss: 4.5146 - val_accuracy: 0.0753
och 2/10
6/836 [=====] - 11s 14ms/step - loss: 4.4061 - accuracy: 0.0832 - val_loss: 4.4040 - val_accuracy: 0.0860
och 3/10
6/836 [=====] - 11s 14ms/step - loss: 4.2904 - accuracy: 0.1002 - val_loss: 4.3429 - val_accuracy: 0.0974
och 4/10
6/836 [=====] - 11s 14ms/step - loss: 4.1692 - accuracy: 0.1231 - val_loss: 4.2078 - val_accuracy: 0.1219
och 5/10
6/836 [=====] - 11s 14ms/step - loss: 4.0474 - accuracy: 0.1439 - val_loss: 4.1036 - val_accuracy: 0.1414
och 6/10
6/836 [=====] - 11s 14ms/step - loss: 3.9493 - accuracy: 0.1646 - val_loss: 4.0474 - val_accuracy: 0.1534
och 7/10
6/836 [=====] - 11s 14ms/step - loss: 3.8689 - accuracy: 0.1814 - val_loss: 4.0065 - val_accuracy: 0.1680
och 8/10
6/836 [=====] - 11s 14ms/step - loss: 3.7963 - accuracy: 0.1965 - val_loss: 3.9360 - val_accuracy: 0.1758
och 9/10
6/836 [=====] - 11s 14ms/step - loss: 3.7316 - accuracy: 0.2087 - val_loss: 3.9254 - val_accuracy: 0.1796
och 10/10
6/836 [=====] - 11s 14ms/step - loss: 3.6743 - accuracy: 0.2208 - val_loss: 3.9139 - val_accuracy: 0.1884
```

New version:

```
h 6/20
'/1607 [=====] - 52s 32ms/step - loss: 4.0472 - accuracy: 0.1829 - val_loss: 3.9261 - val_accuracy: 0.2509
h 7/20
'/1607 [=====] - 52s 32ms/step - loss: 3.9812 - accuracy: 0.2102 - val_loss: 3.8651 - val_accuracy: 0.2635
h 8/20
'/1607 [=====] - 52s 32ms/step - loss: 3.9305 - accuracy: 0.2286 - val_loss: 3.8659 - val_accuracy: 0.2760
h 9/20
'/1607 [=====] - 52s 32ms/step - loss: 3.8936 - accuracy: 0.2436 - val_loss: 3.8179 - val_accuracy: 0.2977
h 10/20
'/1607 [=====] - 52s 32ms/step - loss: 3.8567 - accuracy: 0.2579 - val_loss: 3.7500 - val_accuracy: 0.3115
h 11/20
'/1607 [=====] - 52s 32ms/step - loss: 3.8289 - accuracy: 0.2672 - val_loss: 3.7434 - val_accuracy: 0.3057
h 12/20
'/1607 [=====] - 52s 32ms/step - loss: 3.7980 - accuracy: 0.2763 - val_loss: 3.7971 - val_accuracy: 0.2974
h 13/20

h 19/20
'/1607 [=====] - 52s 32ms/step - loss: 3.6780 - accuracy: 0.3120 - val_loss: 3.6906 - val_accuracy: 0.3283
h 20/20
'/1607 [=====] - 52s 32ms/step - loss: 3.6651 - accuracy: 0.3139 - val_loss: 3.6442 - val_accuracy: 0.3337
it is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings
```

Kyu Model accuracy:
Old version:

```
och 1/10
0/790 [=====] - 34s 42ms/step - loss: 4.6663 - accuracy: 0.0609 - val_loss: 4.2637 - val_accuracy: 0.0892
och 2/10
0/790 [=====] - 33s 42ms/step - loss: 4.1240 - accuracy: 0.1122 - val_loss: 4.0151 - val_accuracy: 0.1306
och 3/10
0/790 [=====] - 33s 42ms/step - loss: 3.9126 - accuracy: 0.1509 - val_loss: 3.8479 - val_accuracy: 0.1610
och 4/10
0/790 [=====] - 33s 42ms/step - loss: 3.7531 - accuracy: 0.1831 - val_loss: 3.8048 - val_accuracy: 0.1731
och 5/10
0/790 [=====] - 33s 42ms/step - loss: 3.6319 - accuracy: 0.2109 - val_loss: 3.6397 - val_accuracy: 0.2104
och 6/10
0/790 [=====] - 33s 42ms/step - loss: 3.5368 - accuracy: 0.2347 - val_loss: 3.6487 - val_accuracy: 0.2079
och 7/10
0/790 [=====] - 33s 42ms/step - loss: 3.4631 - accuracy: 0.2520 - val_loss: 3.5319 - val_accuracy: 0.2450
och 8/10
0/790 [=====] - 33s 42ms/step - loss: 3.3922 - accuracy: 0.2675 - val_loss: 3.5188 - val_accuracy: 0.2427
och 9/10
0/790 [=====] - 33s 42ms/step - loss: 3.3419 - accuracy: 0.2778 - val_loss: 3.4554 - val_accuracy: 0.2560
och 10/10
0/790 [=====] - 33s 42ms/step - loss: 3.2913 - accuracy: 0.2881 - val_loss: 3.4575 - val_accuracy: 0.2613
```

New version:

```
./1541 [=====] - 50s 32ms/step - loss: 3.6868 - accuracy: 0.2834 - val_loss: 3.5352 - val_accuracy: 0.3429
h 8/20
./1541 [=====] - 50s 32ms/step - loss: 3.6277 - accuracy: 0.3039 - val_loss: 3.4806 - val_accuracy: 0.3619
h 9/20
./1541 [=====] - 50s 32ms/step - loss: 3.5814 - accuracy: 0.3227 - val_loss: 3.4919 - val_accuracy: 0.3563
h 10/20
./1541 [=====] - 50s 32ms/step - loss: 3.5375 - accuracy: 0.3344 - val_loss: 3.4612 - val_accuracy: 0.3684
h 11/20
./1541 [=====] - 50s 32ms/step - loss: 3.5047 - accuracy: 0.3448 - val_loss: 3.4203 - val_accuracy: 0.3813
h 12/20
./1541 [=====] - 50s 32ms/step - loss: 3.4792 - accuracy: 0.3534 - val_loss: 3.3959 - val_accuracy: 0.3912
h 13/20

h 19/20
./1541 [=====] - 50s 32ms/step - loss: 3.3501 - accuracy: 0.3854 - val_loss: 3.3242 - val_accuracy: 0.3968
h 20/20
./1541 [=====] - 50s 32ms/step - loss: 3.3347 - accuracy: 0.3894 - val_loss: 3.3125 - val_accuracy: 0.4130
```

Playing-style Model accuracy:

```
375/375 [=====] - 2s 5ms/step - loss: 1.0404 - accuracy: 0.5010 - val_loss: 1.0456 -  
val_accuracy: 0.5222  
Epoch 7/35  
375/375 [=====] - 2s 5ms/step - loss: 1.0150 - accuracy: 0.5403 - val_loss: 0.9468 -  
val_accuracy: 0.6112  
Epoch 8/35  
375/375 [=====] - 2s 5ms/step - loss: 0.9349 - accuracy: 0.6146 - val_loss: 0.9186 -  
val_accuracy: 0.6150  
Epoch 9/35  
375/375 [=====] - 2s 5ms/step - loss: 0.8938 - accuracy: 0.6337 - val_loss: 0.8812 -  
val_accuracy: 0.6281  
Epoch 10/35  
375/375 [=====] - 2s 5ms/step - loss: 0.8649 - accuracy: 0.6442 - val_loss: 0.8919 -  
val_accuracy: 0.6210  
Epoch 11/35  
375/375 [=====] - 2s 5ms/step - loss: 0.8485 - accuracy: 0.6576 - val_loss: 0.8636 -  
val_accuracy: 0.6450  
Epoch 12/35  
375/375 [=====] - 2s 5ms/step - loss: 0.8321 - accuracy: 0.6623 - val_loss: 0.8727 -  
val_accuracy: 0.6360  
Epoch 13/35  
...  
Epoch 34/35  
375/375 [=====] - 2s 5ms/step - loss: 0.5922 - accuracy: 0.8271 - val_loss: 0.9486 -  
val_accuracy: 0.6559  
Epoch 35/35  
375/375 [=====] - 2s 5ms/step - loss: 0.5880 - accuracy: 0.8324 - val_loss: 0.9568 -  
val_accuracy: 0.6600
```

Conclusion

The game of Go itself is incredibly complex, making this project quite challenging for us. The comprehensive assessment across various skill levels and styles, the emulation based on analyzing others' playing styles, predicting the next move's position, and identifying different playing styles have deepened our understanding and application of deep learning.

Although the model trained on the training set exhibits high accuracy, there's substantial room for improvement when tested on formal datasets. Aspects such as more precise feature extraction, multidimensional additions, experimenting with different activation functions, increased regularization or alterations in convolutional layers, alongside variations in training volume and adjustments in learning, aim to prevent overfitting.

Leaderboard:


Public_test & Leaderboard [46/508]

TBrain AI 實戰吧					
Home Competitions Discussion Datasets Success					
Overview Leaderboard Download Dataset Submission History					
1:29:36 PM					
43	TEAM_4073	4	20	0.412673	11/27/2023 7:37:48 PM
44	TEAM_4731	4	20	0.402548	11/24/2023 5:45:42 PM
45	TEAM_4752	1	3	0.401959	11/20/2023 10:32:35 PM
46	TEAM_4864	1	20	0.381946	11/25/2023 3:54:59 PM

public_submission_template_1.csv	test	2023-11-25 03:54:59	0.381946	0.0	Scoring success.
上傳成員 張 詠軒					
Advanced_Public_Ten_Kyu_1	Advanced_Public_Ten_Kyu_5	Advanced_Public_One_Dan_1	Advanced_Public_One_Dan_5	Advanced_Public_PSA	
0.182187	0.528924	0.211636	0.578182	0.57593	

Private_test & Leaderboard [48/508]

47	TEAM_4242	1	20	0.334581	11/28/2023 6:53:56 PM
48	TEAM_4864	1	20	0.318905	11/28/2023 9:07:35 PM
49	TEAM_4564	1	5	0.307644	11/28/2023 12:58:17 AM
50	TEAM_4838	3	4	0.307357	11/28/2023 7:26:31 PM
51	TEAM_4414	1	18	0.303846	11/26/2023 10:36:42 PM
52	TEAM_4517	3	3	0.299448	11/28/2023 11:55:50 PM



508
參賽隊伍

public_private_submission_template.csv	test	2023-11-28 09:07:35	0.380682	0.318905	Scoring success.
上傳成員 張 詠軒					
Advanced_Public_Ten_Kyu_1	Advanced_Public_Ten_Kyu_5	Advanced_Public_One_Dan_1	Advanced_Public_One_Dan_5	Advanced_Public_PSA	
0.182187	0.528924	0.218818	0.592909	0.560824	

Reference

[1] Yuandong Tian and Yan Zhu, "Better Computer Go Player with Neural Network and Long-term Prediction", ICLR, 2016.

[2] Chang-Shing Lee, Mei-Hui Wang, Shi-Jim Yen, Ting-Han Wei, I-Chen Wu, Ping-Chiang Chou, Chun-Hsun Chou, Ming-Wan Wang, and Tai-Hsiung Yang, "Human vs. Computer Go: Review and Prospect", IEEE Computational Intelligence Magazine (IEEE CIM), Vol. 11, No. 3, pp. :67- 72, August 2016.

[3] David Silver, Aja Huang, Chris J. Maddison, "Mastering the game of Go with deep neural networks and tree search" Nature, vol. 529, no. 7587, pp. 484-489, 2016.

[4] David Silver, Thomas Hubert, Julian Schrittwieser, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm", Science, vol. 362, no. 6419, pp. 1140-1144, 2018.

[5] Christopher Clark, Amos Storkey, "Training deep convolutional neural networks to play Go", arXiv preprint arXiv:1412.3409, 2014.

[6] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", Advances in Neural Information Processing Systems (NIPS), 2012.