1. Introduction

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.

After obtaining the moves, further analysis is conducted to identify the playing-style in specific game situations, categorized as aggressive, balanced, and territorial.

2. How to solve?

Data Processing:

Firstly, feature extraction is performed for predicting the next move of a player in the game of Go. Four 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.

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:

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.

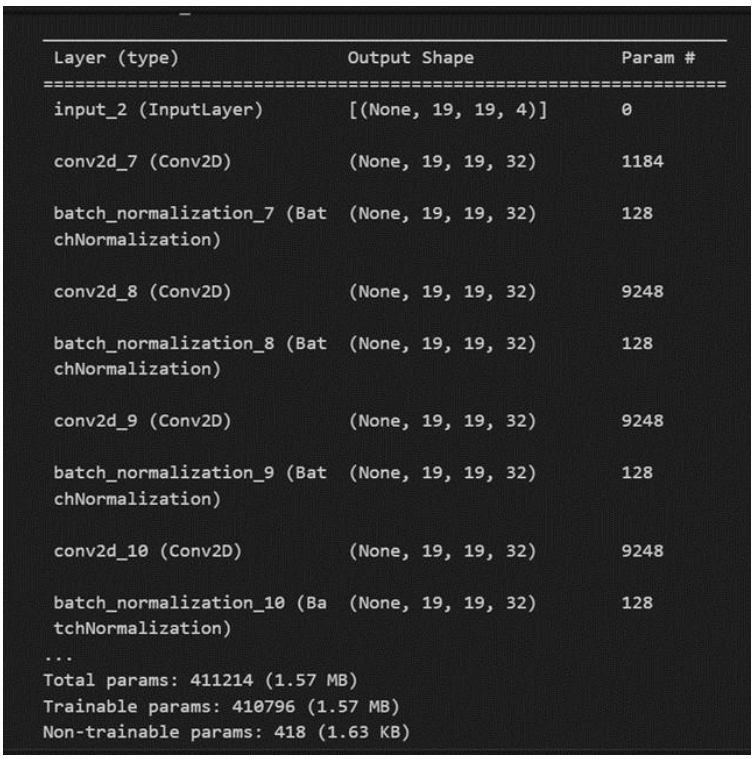
For the model trained on Kyu (lower-level) players, multiple convolutional layers were established using the Conv2D function,

with kernel sizes of 7x7, 5x5, and 3x3, each having 32 filters. The activation function for all convolutional layers is ReLU. A Flatten layer is applied to convert the output of the convolutional layers into a one-dimensional tensor. The output layer utilizes the Softmax activation function to obtain probabilities for the potential next move at each position. The model is compiled using the Adam optimizer to minimize the model's loss.

For training the model to recognize playing styles, the model includes convolutional layers with Dropout regularization, and Dense layers with L2 regularization. The output layer uses Softmax activation for a 3-dimensional vector representing the three playing styles (aggressive, balanced, territorial). The model is compiled using the RMSprop optimizer.

Training models:

Dan model:



Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 19, 19, 4)]	0
conv2d_7 (Conv2D)	(None, 19, 19, 32)	1184
batch_normalization_7 (BatchNormalization)	(None, 19, 19, 32)	128
conv2d_8 (Conv2D)	(None, 19, 19, 32)	9248
batch_normalization_8 (BatchNormalization)	(None, 19, 19, 32)	128
conv2d_9 (Conv2D)	(None, 19, 19, 32)	9248
batch_normalization_9 (BatchNormalization)	(None, 19, 19, 32)	128
conv2d_10 (Conv2D)	(None, 19, 19, 32)	9248
batch_normalization_10 (BatchNormalization)	(None, 19, 19, 32)	128

...

Total params: 411214 (1.57 MB)
Trainable params: 410796 (1.57 MB)
Non-trainable params: 418 (1.63 KB)

Kyu model:

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)

Playing-style model:

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

3.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.